iPDS™ PERSONAL DEVELOPMENT SYSTEM USER'S GUIDE

Copyright © 1982, 1983, Intel Corporation

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Order Number 162606-003

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A Computing Device pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

AEDIT
BITBUS
BXP
COMMputer
CREDIT
i
iATC
I^2ICE
ICE
iCS
iDBP
iDIS
iLBX
im
iMMX
Insite
Intel
IntelBOS
Intelevision
intelligent Identifier
intelligent Programming
Intellec
Intellink
iOSP
iPDS
iRMX
iSBC
iSBX
iSDM
iSXM
Library Manager
MCS
Megachassis
MICROMAINFRAME
MULTIBUS
MULTICHANNEL
MULTIMODULE
Plug-A-Bubble
PROMPT
Promware
Ripplemode
RMX/80
RUPI
System 2000
UPI

Copyright 1983, Intel Corporation

REV.	REVISION HISTORY	DATE
-001	Original Issue	2/82
-002	Change Notice #1. Change pages, clarify COPY command description. Expand Multimodule application information, and character set data, and correct front matter. Index references and typographical errors.	10/82
-003	Incorporate Change Notice #1, reformat, show current installation of I/O connector, revise to be system specific and reprint.	2/83

PREFACE

This manual describes the Intel Personal Development System (iPDS™) and provides the information needed to install, maintain, and operate the system and the Intel System Implementation Supervisor for the system (ISIS-PDS).

There is a great deal of information in this manual. The order in which to read the material depends on the goals of the reader. For example, Chapters 1 and 2 contain only background and overview information. The user who is interested in using the system as quickly as possible can skip this material.

Figure 1 shows two possible paths through the manual: one for the user who is anxious to get started with the system as quickly as possible and the other for the user who wishes to spend more time gaining background information prior to actually starting with the system. Typically, a person who has had previous experience with an interactive computer system will take the quick path while the inexperienced user will choose the more leisurely path.

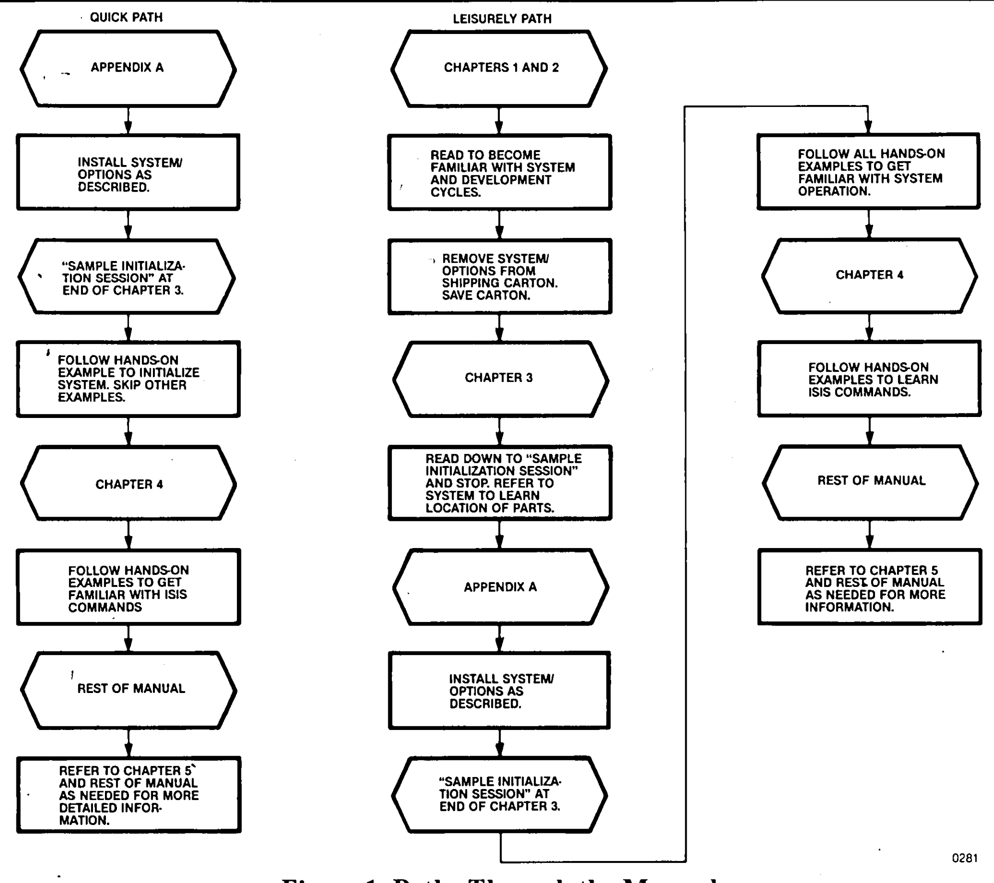


Figure 1 Paths Through the Manual

The key to using the information in this manual is being able to access the needed facts quickly. Features provided to aid in this process are: an Index for the entire manual at the end of the manual, a Table of Contents for the entire manual at the beginning of the manual, a detailed Table of Contents for each chapter at the beginning of the chapter, and tabbed chapters to quickly find the desired section. In addition, an overview is given below of the information in each chapter of the manual.

- Chapter 1 contains overview information. It describes the features of the basic system and describes the available options for the system. It also contains information on the organization and use of this manual and related publications.

- Chapter 2 contains background information. It outlines the process of developing a microcomputer-based product and illustrates how a development system aids in this process.

- Chapter 3 describes the basic operator controls for the system and introduces the operating system software in a hands-on demonstration.

- Chapter 4 describes the operating system commands from a functional point of view, illustrating through hands-on examples how and when to use particular commands.

- Chapter 5 describes the operating system commands in alphabetical order and in a reference format for the experienced user.

- Chapter 6 describes the CREDIT text editing features unique to the iPDS system . The editing macro CMACRO is explained. Command descriptions and examples are found in the *ISIS CREDIT CRT-Based Text Editor User's Guide,* manual order number 9800902.

- Chapter 7 describes the use of the DEBUG command. The material is organized in a reference format with a brief introduction and hands-on debugging session.

- Chapter 8 provides detailed technical information on the system. This chapter is aimed at the systems programmer who will add customized I/O drivers to the system. The information is organized in a reference format.

- Chapter 9 describes the use of the optional processor with the system. Hands-on examples are included.

- Chapter 10 describes software for the PROM programmer option unique to the iPDS system

- Appendix A describes step-by-step procedures for installing the system and its options.

- Appendix B contains all the error messages for the operating system in a standard format and describes error conditions. It also includes a detailed description of the Confidence Test for the iPDS system.

- Appendix C contains reference tables for different number bases, for ASCII code, for the control characters, and for the graphics characters.

- Appendix D contains a comparison of the ISIS-PDS operating system and the ISIS-II operating system.

- Appendix E contains helpful hints for operating at maximum efficiency depending on the configuration (the features and options available) of the system being used.

- A glossary of technical terms and abbreviations is included for reference after the last appendix.

Additional information on this manual and related publications is in Chapter 1.

SERVICE INFORMATION

The best service for your Intel product is provided by an Intel Customer Engineer. These professionals provide prompt, efficient, on-site installlation, preventive maintenance, and corrective maintenance services required to keep your equipment in the best possible operating condition.

The Intel Customer Engineer provides the service needed through a prepaid service contract or on an hourly charge basis. For further information, contact your local Intel sales office.

When the Intel Customer Engineer is not available, contact the Product Service Center.

United States customers can obtain service and repair assistance from Intel Corporation by contacting the Intel Product Service Center in their local area. Customers outside the United States should contact their sales source (Intel Sales Office or Authorized Distributer) for service information and repair assistance.

Before calling the Product Service Center, Have the following information available:

1. The date you received the product.

2. The complete part number of the product (including dash number). On boards, this number is usually silk-screened onto the board. On other MCSD products, it is usually stamped on a label.

3. The serial number of the product. On boards, this number is usually stamped on the board. On other MCSD products, the serial number is usually stamped on a label mounted on the outside of the chassis.

4. The shipping and billing address.

5. If the Intel Product warranty has expired, a purchase order number is needed for billing purposes.

6. Be sure to advise the Center personnel of any extended warranty agreements that apply.

Use the following numbers for contacting the Intel Product Service Center:

Western Region call: (602) 869-4951
Midwest Region call: (602) 869-4392
Eastern Region call: (602) 869-4045
International call: (602) 869-4391

Always contact the Product Service Center before returning a product to Intel for repair. You are given a repair authorization number, shipping instructions, and other important information which helps Intel provide you with fast, efficient service. If you are returning a product because of damage sustained during shipment, or if the product is out of warranty, a purchase order is required before Intel can initiate the repair.

If available, use the original factory packaging material, when preparing a product for shipment to the Intel Product Service Center. If the original packaging material is not available, wrap the product in a cushioning material such as Air Cap SD-240, manufactured by the Sealed Air Corporation, Hawthorne, N.J. Securely enclose it in a heavy-duty corrugated shipping carton, mark it "FRAGILE" to ensure careful handling, and ship it to the address specified by the Intel Product Service Center.

WARNINGS AND CAUTIONS

This section lists all the warnings and cautions in the order they appear in the manual. Refer to the indicated page number for the context of the warning or the caution.

PAGE

WARNING

Never remove the top cover. There is a risk of electric shock or fire from high voltage. Repairs should be performed by qualified service personnel only 1-4/A-7

CAUTION

Refer to Appendix A of this manual for installation instructions before attempting to operate the system 3-1

CAUTION

Ensuring trouble-free storage of data on the flexible disk requires proper care. Specific precautions follow:

- Return the disk to its envelope when not in use
- Do not touch or clean the recording surface
- Do not smoke around the disk
- Do not bend the disk or use paper clips or other mechanical devices on it
- Use a felt tip pen on the user label, not a pencil or ball point pen

The following actions can also damage or modify the data stored on the flexible disk:

- Turning the system on or off with a disk inserted in the drive
- Opening the disk drive door while the drive select light is on
- Pressing the RESET switch while the drive select light is on ... 3-14/3-15

CAUTION

In addition, the use of the pause option (P) for the COPY, DIR, and DELETE commands is prohibited when these commands are run from a SUBMIT file. No error or warning message is issued; however, use of the option can destroy files on one or more of the disks. 5-47

PAGE

CAUTION

Do not operate the system in the carrying position. The system must be opened and in the operating position to dissipate the heat properly. To allow proper cooling of the unit, a minimum of six inches (15.24 cm) of clearance is recommended on all sides and the air vents must be clear of any obstructions, ... A-3

WARNING

Changing the power cord involves hazardous voltage and current levels. To avoid the risk of electric shock and fire, the power cord should be changed only by qualified technical personnel. A-4

WARNING

Installation of some of the options involves working with hazardous voltage and current levels. To avoid the risk of electric shock and fire, options should be installed only by qualified technical personnel. A-7

CAUTION

The plug-in module slot breaches the electrical shielding of the iPDS system. There is a chance of electro-static discharge (ESD) passing, via the plug-in module, to the internal circuitry of the iPDS system and causing system RESET's, disk file damage, or component damage. Ensure that the iPDS system is turned OFF before inserting or removing any plug-in module. A-27

CONTENTS

CHAPTER 1 INTRODUCTION

PAGE

Purpose 1-1
Typical Uses 1-1
Major Characteristics 1-1
System Components 1-1
  Hardware 1-1
    Basic Unit 1-2
    Add-On Mass Storage 1-4
    Dual Processors 1-5
    Plug-In Modules 1-6
    Multimodules 1-6
  Software 1-7
    Operating System 1-8
    Assemblers 1-8
    High Level Languages 1-8
    Utilities 1-8
    Other Software 1-8
  Overview of System Publications 1-8
    Hardware Installation and Checkout 1-9
    System Operations 1-9
    Text Editing 1-10
    Software Debugging 1-10
    Systems Programming 1-10
    Dual Processing 1-10
    PROM Programming 1-10
    Microprocessor Emulation 1-10
    Multimodule Expansion 1-11
    Applications Programming 1-11
    Notational Conventions 1-11
    Other Conventions 1-11
    CAUTION, WARNING, and NOTE
      Symbols 1-11
    Commonly Used Terms 1-12

CHAPTER 2 DEVELOPMENT SYSTEMS

The Development Task 2-1
  Software Development 2-1
  Hardware Development 2-2
  Integration 2-2
  Production Testing 2-3
  Field Service 2-3
The Development Tools 2-4
  Software Development Tools 2-4
  Emulators 2-4
  Summary 2-5
Overview of the Development Cycle 2-6

CHAPTER 3 BASIC SYSTEM OPERATION

PAGE

Hardware Operation 3-1
  Rear Control Panel 3-2
  Removable I/O Panel 3-3
  Storage Area 3-3
  Powering the System On and Off 3-4
  Keyboard 3-5
  Display Screen 3-8
  Disk Drives 3-10
    Care and Use of Flexible Disk 3-11
  Bubble Memory 3-14
  Other Components 3-14
Software Operation 3-15
  Initialization 3-15
    Error Conditions 3-17
  User Configurations 3-17
  Commands 3-17
  Command Lines 3-19
    Command Line Defaults 3-20
  Entering Command Lines 3-20
    Entering Command Lines from the
      Keyboard 3-21
      Editing Command Lines 3-21
      Pausing the Display 3-22
    Entering Command Lines from a File 3-22
    Other Ways to Enter Command Lines 3-23
  Sample Initialization Session 3-23
    Initializing the System from Disk 3-24
    Duplicating the System Disk on Single Drive
      Systems 3-25
    Duplicating the System Disk on Multiple Drive
      Systems 3-27
    Entering Command Lines 3-28
    Using Control Characters 3-34
    Editing Command Lines 3-36
    Initializing the System from Bubble
      Memory 3-39
    Running the Confidence Test 3-42

CHAPTER 4 COMMAND APPLICATIONS

Functional Summary of Commands 4-1
  System Management Commands 4-1
    Sample System Management Commands 4-2
  Device Management Commands 4-7
    Formatting a Non-System Disk 4-8
    Changing the System Input and Output
      Devices 4-9
    Using the Serial Port 4-12
  File Management Commands 4-14
    Displaying a List of Files 4-14

CONTENTS (continued)

	PAGE
Assigning and Removing File Attributes	4-18
Copying Files	4-21
Changing Filenames	4-22
Appending Files	4-22
Displaying a Text File on the CRT	4-23
Using Wildcard Characters	4-34
File Operations With a Single Drive System	4-37
Text Editing Commands	4-40
Editing Text Files	4-40
Creating a Source Program	4-49
Program Development Commands	4-52
Creating an Object File	4-53
Debugging a Program	4-54
Program Execution Commands	4-59
Using the JOB Command	4-60
Automatic Job Execution	4-61
Configuring a User System Automatically	4-65
Using the SUBMIT Command	4-66
Running the SUBMIT File	4-70

CHAPTER 5
COMMAND DICTIONARY

Notational Conventions	5-1
Special Command Format Terms	5-2
Device Names	5-2
Physical Devices	5-2
System-Defined Devices	5-3
User-Defined Devices	5-3
Logical Devices	5-4
Filenames	5-5
Wildcard Filenames	5-6
Pathnames	5-6
Source	5-7
Destination	5-7
N and A	5-7
Jobfile	5-8
Command Description Format	5-8
Functional Summary of Commands	5-9
ASSIGN	5-10
ATTACH	5-13
ATTRIB	5-14
COPY (Transfers files)	5-16
COPY (Appends files)	5-19
DELETE	5-21
DETACH	5-23
DIR	5-24
ENDJOB	5-26
HELP	5-27
IDISK	5-28
JOB	5-30
RENAME	5-32
SERIAL	5-33
SUBMIT	5-36
?	5-42
@	5-43
/	5-44
#	5-45
.	5-46
FUNC <n>	5-47
ESC	5-48

CHAPTER 6
CREDIT TEXT EDITOR

Introduction	6-1
Getting Started With the CREDIT Text Editor	6-1
Screen Mode Features	6-1
The CREDIT Display	6-1
The Keyboard	6-2
The Cursor	6-4
Command Mode Features	6-4
The CREDIT Display	6-4
The Keyboard	6-5
Disk File Use	6-5
Temporary Files	6-6
Backup Files	6-7
Files Used By CREDIT Commands	6-7
Limits on Disk File Use	6-7
Performance and File Size	6-8
CMACRO.MAC	6-8
The CMACRO File	6-8
Cursor Movement Macros	6-8
Text Control Macros	6-9
Block Transfer Macros	6-9
File Formatting Macros	6-10
Data File Macros	6-11

CHAPTER 7
DEBUG COMMAND

Software Debugging and the Development Task	7-1
DEBUG Features	7-1
DEBUG Command	7-2
Command Format	7-2
Comments	7-2
Examples	7-3
Overview of the Debugging Commands	7-4
I/O Interface	7-4
Software Development	7-4

CONTENTS (continued)

	PAGE
Entering Debugging Commands	7-5
Command Format for Debugging Commands	7-5
Entry Errors	7-5
Invalid Characters	7-5
Address Value Errors	7-6
Parameter Errors	7-6
Categories of Debugging Commands	7-7
Program Execution Commands	7-7
I/O Configuration Commands	7-7
I/O Control Commands	7-9
Memory Control Commands	7-9
Register Commands	7-9
Utility Commands	7-10
Sample Debugging Session	7-10
Debugging Commands in Alphabetical Order	7-17
A Assign Command	7-17
C Disassemble Command	7-19
D Display Memory Command	7-20
E Exit Command	7-21
F Fill Memory Command	7-22
FUNCT-R Manual Interrupt Command	7-22
G Execute Command	7-23
H Hexadecimal Add/Subtract Command	7-25
I Input Command	7-26
M Move Memory Command	7-26
N Single Step Command	7-28
O Output Command	7-28
Q Query Command	7-29
S Substitute Memory Command	7-31
T Disassemble Command	7-32
X Display/Modify Registers Command	7-33

CHAPTER 8
SYSTEM PROGRAMMER'S REFERENCE

	PAGE
Operating System Considerations	8-1
Needed Functions	8-2
Features of the ISIS-PDS Operating System	8-2
System Calls	8-3
Overview of System Calls	8-3
Functional Categories of System Calls	8-4
High Level System Calls	8-4
File I/O Operations	8-4
Disk Directory Maintenance	8-4
Console Device Assignment	8-4
Error Message Output	8-4
Program Loading and Execution	8-5
I/O Driver Extensions	8-5
Primitive System Calls	8-5
Peripheral I/O Routines	8-5
System Status Routines	8-5
I/O Driver Extensions	8-5
Differences Between High Level and Primitive System Calls	8-5
System Call Format and Use	8-5
PL/M Calls	8-6
Assembly Language Calls	8-6
Assembly Language Calls to High Level System Routines	8-7
Assembly Language Calls to Primitive System Routines	8-8
Error Handling	8-8
System Calls in Alphabetical Order	8-9
Notational Conventions	8-9
General Format Terms	8-9
Description Formats	8-12
ATTACH	8-14
ATTRIB	8-16
CI	8-18
CLOSE	8-19
CO	8-21
CONSOL	8-22
CSTS	8-24
DELETE	8-25
DETACH	8-26
ERROR	8-28
EXIT	8-30
IOCHK	8-32
IODEF	8-34
IOSET	8-36
LO	8-37
LOAD	8-38
MEMCK	8-40
OPEN	8-41
PO	8-44
READ	8-45
RENAME	8-48
RESCAN	8-50
RI	8-52
SEEK	8-53
SPATH	8-57
WHOCON	8-60
WRITE	8-62
Example Programs Using System Calls	8-64
System Architecture	8-70
Memory Organization and Allocation	8-70
Interrupt Vectors	8-71
ISIS Resident Area 1	8-71
Buffer Area and ISIS Resident Area 2	8-71
User Programs and ISIS Non-resident Area	8-72
Examples of Calculating the User Program Base Address	8-73

CONTENTS (continued)

	PAGE
I/O Address Space	8-73
CRT and Keyboard I/O	8-75
Cursor Addressing and Graphics Mode	8-75
Serial I/O	8-76
Printer I/O	8-77
Multimodule I/O	8-78
Peripheral Device I/O Operations	8-80
File I/O	8-80
Dynamic File Control	8-82
Line Edited Files	8-82
Terminating Characters	8-83
Editing Characters	8-83
Reading from the Line Editing Buffer	8-83
Reading a Command Line	8-84
Disk File Types	8-85
Notation Used to Describe Records	8-85
MCS-80/85 Absolute Object File Format	8-86
Disk Structure	8-88
General Disk File Structure	8-88
Blocks	8-89
Interleaving Factors	8-91
System Disk Files	8-92
ISIS.PDS	8-92
ISIS.CLI	8-92
ISIS.TO	8-93
ISIS.LAB	8-93
ISIS.DIR	8-93
ISIS.FRE	8-95
Disk File Structure Summary	8-96

CHAPTER 9
DUAL PROCESSING

Introduction	9-1
Operating a Dual Processing System	9-1
Sharing the Keyboard	9-2
Sharing the CRT Display	9-2
Sharing Disk Drives	9-4
Sharing Multimodules	9-5
Sharing Files	9-5
Temporary Files	9-6
Data Files	9-6
Initializing the System	9-7
Sample Dual Processing	9-8
Programming on a Dual Processing System	9-12
Shared Resources	9-13
Semaphores	9-13
Shared Multimodules	9-13
Shared Files	9-14

CHAPTER 10
PROM PROGRAMMING

	PAGE
Firmware Development	10-1
EPROM Erasure	10-2
Overview of PROM Programming on the System	10-3
Personality Modules	10-3
Plug-in Module Adapter Board	10-3
iPPS Software	10-3
PROM Programming Subsystem	10-4
iPPS Software	10-5
iPPS Initialization	10-5
Command Line Invocation	10-5
Invocation Via a SUBMIT File	10-6
iPPS General Operation	10-6
Major Functions	10-6
iPPS Storage Devices	10-7
PROM Device	10-7
Buffer Device	10-7
File Device	10-8
Command Entry	10-9
Command Entry Editing	10-10
Form of iPPS Commands	10-10

APPENDIX A
INSTALLATION INSTRUCTIONS

Installation Considerations	A-1
Initial Installation Procedures	A-1
Changing the Fuse	A-5
Installing Options	A-6
Removing the I/O Panel	A-7
Connecting a Serial Device	A-10
Configuring the CTS and RTS Lines	A-11
Configuring the RXC and TXC Lines	A-11
Configuring the DTR Line	A-13
Configuring the RXD and TXD Lines	A-13
Connecting a Serial Device	A-13
Serial Interface Specifications	A-14
Optional Processor	A-14
Multimodule Adapter	A-17
Multimodule	A-19
Plug-in Module Adapter	A-23
Plug-in Module	A-24
Connecting a Line Printer	A-24
Line Printer Interface Specifications	A-24
Functional Description	A-26
System Chassis	A-27
Base Processor Board	A-27
Keyboard	A-27
Integral CRT	A-27
Integral Disk Drive	A-27
Power Supply	A-28

CONTENTS (continued)

PAGE

User Controls A-28
Optional Processor Board A-28
Optional Multimodule Adapter Board A-28
Optional Plug-in Module Adapter Board A-28
Specifications A-28

APPENDIX B ERROR INDICATIONS

Command Entry Error Messages B-1
ISIS-PDS Exception and Error Handling B-2
Non-Fatal Errors B-2
Fatal Errors B-3
Console Interface Errors B-4
Error Messages in Numeric Order B-4
Resident ISIS Routines B-4
Console Interface Routines B-11
Diagnostic Errors B-13
LED Indicators B-13
Diagnostic Error Messages B-14
Confidence Test B-15
PCONF Command B-17
INIT CONPDS Command B-17
Confidence Test Commands B-17
CLEAR Command B-18
DESCRIBE Command B-18
ERROR Command B-19
EXIT Command B-19
IGNORE Command B-19
LIST Command B-20
RECOGNIZE Command B-20
SUMMARY Command B-20
TEST Command B-21
Test 0 - CPU Test B-23
Test 1 - CRT Interface Test B-23
Test 2 - Programmable Timer Test B-23
Test 3 - Line Printer Interface Test B-23
Test 4 - Serial Interface Test B-23
Test 5 - Disk Semaphore Test B-24
Test 6 - Disk Drive Recalibrate and Ready Test B-24
Test 7 - Disk Drive Seek and Read Test B-24
Test 8 - Serial Loopback Test B-25
Test 9 - Disk Format Test B-25

PAGE

Test A - Formatted Disk Data Read Test B-25
Test B - Disk Random Seek/Write/Read Test B-26
Test C - Keyboard Echo Test B-26
Test D - Bubble Memory Seek and Read Test B-26
Test E - Bubble Memory Random Seek/Write/Read Test B-26
Test F - PROM Programmer Plug-in Module Test B-27
Test 10 - 32K RAM Relocating Random Data Test B-27
Confidence Test Error Messages B-27

APPENDIX C REFERENCE TABLES

Hexadecimal To Decimal Conversion C-1
Base Conversions C-2
Powers of Two and Sixteen C-4
ASCII Code List C-5
ASCII Code Definition C-7
ASCII Code in Binary C-7
Control Codes C-8
Function Codes C-8
Graphics Codes and Escape Sequences C-9

APPENDIX D ISIS-PDS AND ISIS-II

ISIS-PDS and ISIS-II Features D-1

APPENDIX E TIPS FOR OPERATING EFFICIENTLY

Single Drive System E-1
Bubble Memory System E-4
Dual Processor System E-5

GLOSSARY

INDEX

TABLES

TABLE	TITLE	PAGE
3-1	Keyboard Characters and Functions	3-6
7-1	Possible Values for <logical device>	7-17
7-2	Possible Values for <physical device>	7-18
7-3	Logical Devices	7-29
7-4	Possible Values for Physical Device	7-30
7-5	Character Symbols for Register Modification	7-34
8-1	Field Values and Physical Device Assignment	8-32
8-2	Mask Values	8-32
8-3	Interrupt Line Pin Numbers	8-80
8-4	'System File locations	8-92
8-5	Values of System file Bit Maps	8-95
A-1	Serial Interface Specifications	A-14
A-2	Printer Interface Specifications	A-26
A-3	Electrical Specifications for Printer Interface	A-26
A-4	Intel Personal Development System Specifications	A-29
A-5	External Disk Drive Power Supply	A-29
A-6	External Disk Drive Physical Characteristics	A-29
A-7	Power Supply	A-29
A-8	Option Electrical Requirements	A-30
B-1	8272 Status Registers	B-6
B-2	7220 Status Registers	B-9

FIGURES

FIGURE	TITLE	PAGE
1-1	Typical Products Created With a Development System	1-2
1-2	Basic System	1-3
1-3	System in Carrying Position	1-4
1-4	System With Options	1-5
1-5	Plug-In Modules	1-6
1-6	Overview of Operating System Software	1-7
2-1	Typical Product Development Cycle	2-1
2-2	Software Development Cycle	2-2
2-3	Hardware Development Cycle	2-2
2-4	Production Testing	2-3
2-5	Field Service	2-3
3-1	Basic System	3-1
3-2	Rear Panel Controls	3-2
3-3	Removable I/O Panel	3-3
3-4	Accessing the Storage Area	3-4
3-5	The Keyboard	3-5
3-6	Closing the Keyboard for Carrying	3-9
3-7	Display Screen	3-10
3-8	Disk Drive	3-11
3-9	Flexible Disk	3-11
3-10	Disk Insertion	3-13
3-11	Door Release on Disk Drives	3-13
3-12	Plug-in Modules	3-14
3-13	Flowchart of Initialization Program	3-18
5-1	Format of Command Descriptions	5-8
6-1	The CREDIT Display	6-2
6-2	The Keyboard	6-3
6-3	Disk File Use	6-6
8-1	Internal and External Environment	8-1
8-2	Needed Capabilities	8-2
8-3	Format of System Call Descriptions	8-13
8-4	Memory Map	8-70
8-5	Interrupt Vectors	8-71
8-6	Record Format Conventions	8-85
8-7	Module Header Record	8-87
8-8	Content Record	8-87
8-9	Module End Record	8-87
8-10	Disk File Components	8-88
8-11	Pointer Block	8-89
8-12	Data Block	8-89
8-13	Relation of Data and Pointer Blocks	8-90
8-14	Pointer and Data Blocks in a File	8-91
8-15	Sector Interleaving	8-91
8-16	Directory Entry	8-93
8-17	Disk File Structure Summary	8-97
9-1	Split Screen Display	9-3
9-2	Logical and Physical Screens	9-4
10-1	Firmware Development Cycle	10-2
10-2	PROM Programming Subsystem	10-4
A-1	Lowering the Keyboard to Operating Position	A-2
A-2	Door Release on Disk Drive	A-3
A-3	Power Cable	A-3
A-4	Line Voltage Switch	A-4
A-5	Power Switch	A-5

FIGURES (continued)

FIGURE	TITLE	PAGE
A-6	Changing the Fuse	A-6
A-7	Removing the I/O Panel	A-8
A-8	Using the Connector Locks	A-9
A-9	Replacing the I/O Panel	A-9
A-10	Schematics for the Serial I/O Interface	A-10
A-11	Removable Jumper Location and Configuration	A-12
A-12	Removing and Replacing the Plug-in Type Jumpers	A-12
A-13	Mounting Locations for Optional Processor	A-15
A-14	Mounting Technique for Optional Processor	A-15
A-15	Aligning the Optional Processor Board	A-16
A-16	Optional Processor Connection to Multimodule Adapter	A-16
A-17	Mounting Locations for Multimodule Adapter	A-17
A-18	Mounting Technique for the Multimodule Adapter Board	A-19
A-19	Aligning the Multimodule Adapter Board	A-18
A-20	Mounting Locations for Double Wide Multimodule Boards	A-20
A-21	Aligning Double Wide Multimodule Boards	A-20
A-22	Mounting Technique for Multimodule Boards	A-21
A-23	Mounting a Single Wide Multimodule Board	A-21
A-24	Removing Rear Panel Cutouts	A-22
A-25	Connecting Cable to the Rear Panel Cutouts	A-22
A-26	Installing the Adapter Board Assembly	A-23
A-27	Cable Connection for Adapter Assembly	A-24
A-28	Installing Plug-in Module	A-25
B-1	Diagnostic LED Indicators	B-14

CHAPTER 1
INTRODUCTION

Purpose

The iPDS system supports the design and development of products that incorporate Intel microprocessors or microcontrollers.

The system and its options aid in both hardware and software development for products based on many different families of chips, such as the following:

- MCS-51 microcontroller family
- MCS-85 general purpose microprocessor family
- iAPX-88 general purpose microprocessor family

Typical Uses

By incorporating microprocessors from these families, products can range in complexity from a simple process controller to an advanced microcomputer system. Software can range from a single-purpose control program to a complex software system. Figure 1-1 illustrates a few application projects which are produced with the aid of a development system.

The iPDS system is useful at all stages of product design from the initial idea to customer support after the product is in the field. See Chapter 2 for further information on role of the development system in the product design cycle.

Major Characteristics

The development system supports integrated hardware and software development by assembling or compiling source programs for execution and by emulating the target microprocessor, the processor used in the product. Emulation is discussed in Chapter 2.

The system optionally includes a PROM Programmer for programming EPROMS as well as E^2PROMs to store software in the target processor's memory.

Another feature of the development system is its portability; the basic system weighs only 29 pounds and has a handle for carrying.

System Components

The system consists of both hardware and software components to aid in the development effort.

Hardware

The following sections describe the hardware components of the basic system and options.

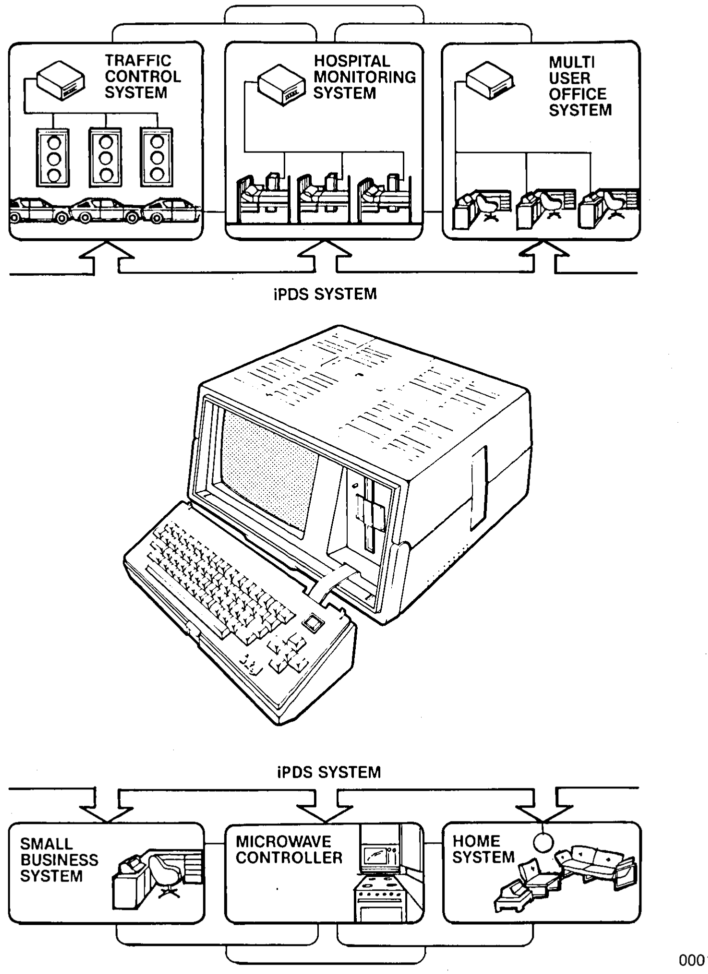


Figure 1-1 Typical Products Created With a Development System

Basic Unit

A single main enclosure and a detachable keyboard enclosure house the system hardware. In operating position (see figure 1-2), the system is 8″H x 16″W x 20″L.

The keyboard detaches from the main enclosure and is connected to the development system with a flat ribbon cable. It consists of standard typewriter keys with cursor control keys, a function key, and a system reset key. Chapter 3 contains a complete description of the keyboard.

The main enclosure without the keyboard is 8"H x 16"W x 18"L and contains the following parts:

- Base processor board
- 9-inch, Cathode Ray Tube (CRT) display unit, 80 characters by 24 lines
- 640K-byte formatted, 5-1/4 inch, flexible disk drive for mass storage
- Switching type power supply

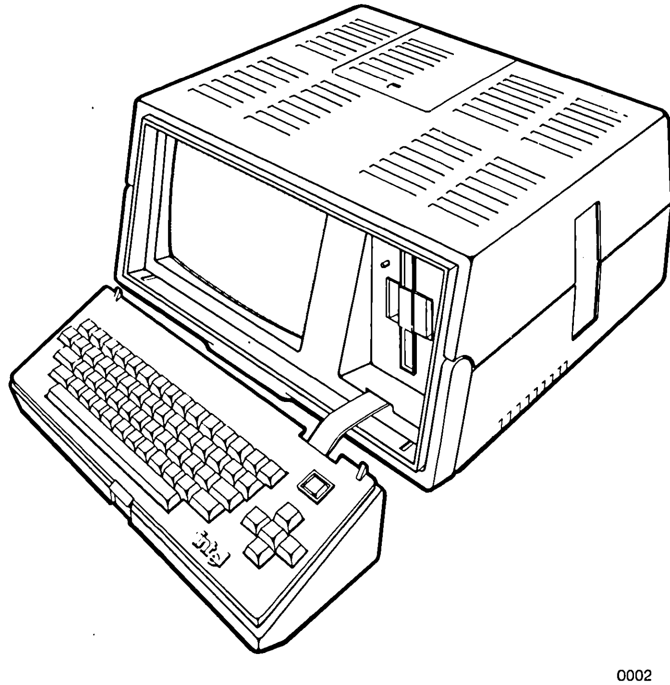


Figure 1-2 Basic System

The power plug, power on/off switch, 115/230 voltage selector switch, and CRT contrast control knob are all on the rear panel as are the connectors for serial I/O, printer I/O, and additional disk drives. See Chapter 3 for the operation of the switches and controls and Appendix A for installation instructions and connector specifications.

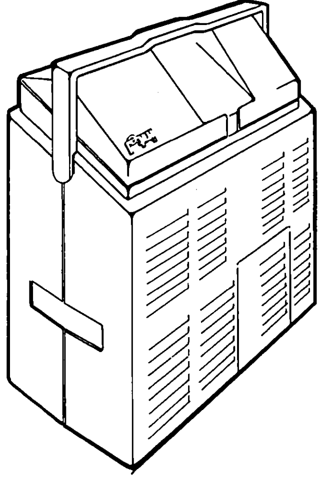
WARNING

Never remove the top cover. There is a risk of electric shock or fire from high voltages. Repairs should be performed by qualified service personnel only.

The base processor board contains the following parts:

- 8085A-2 Central Processor Unit (CPU) operating at 5MHz
- 64K bytes of Random Access Memory (RAM)
- 2K bytes of Read Only Memory (ROM) containing the initialization program and diagnostics
- CRT and keyboard controller
- Flexible disk controller with port for three additional drives
- Emulator and PROM programmer port
- Serial input/output (I/O) port
- Line printer I/O port

The keyboard attaches to the front of the main enclosure for carrying the system; it covers the CRT and flexible disk drive. The handle is attached to the front of the main enclosure and folds out of the way when the system is in use. Figure 1-3 shows the iPDS system in the carrying position.



0003

Figure 1-3 System in Carrying Position

A storage area for cables and plug-in modules is on the top rear of the main enclosure. A slot on the right side of the basic unit is provided for inserting the optional plug-in modules during use. A bail (metal bar) is on the bottom of the unit to position the unit at an angle.

The basic system, without options, provides the foundation for microprocessor product development. Additional hardware and software packages are available as options to build on this foundation. The main enclosure houses the optional system boards and accepts the optional plug-in modules for PROM programming or target microprocessor emulation.

Figure 1-4 shows the iPDS system with an optional plug-in module (a PROM programmer) and three additional flexible disk drives.

Add-On Mass Storage

For many applications, additional mass storage is a desirable feature. One choice is to add mass storage through external disk drives. One to three external drives can be added to the iPDS system. Each additional drive has a 640K-byte capacity (formatted) for a maximum disk storage of 2.56M bytes. The first external drive attaches to the rear of the main enclosure with a round cable. The other two external drives are connected to the rear of the previous external drive. Each additional drive has its own power supply and is mounted in its own housing external to the main system. See figure 1-4.

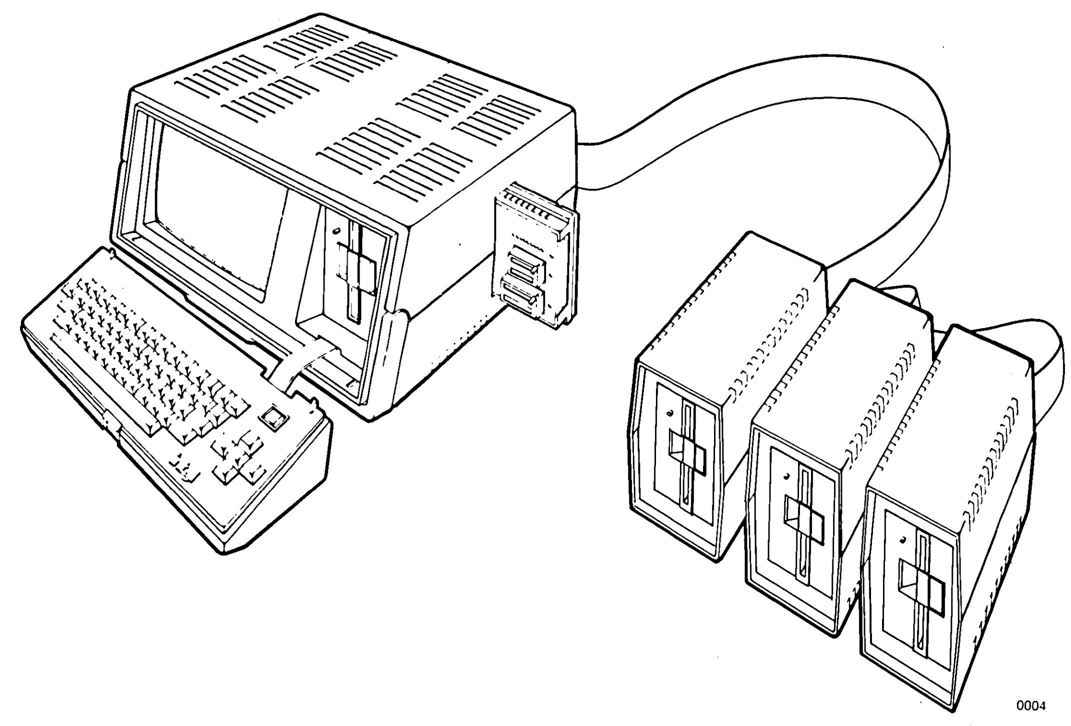


Figure 1-4 System with Options

Another choice for adding mass storage is through iSBX 251 Bubble Memory Multimodules. A maximum of two bubble multimodules, with 128K-bytes each, can be added to a system which already contains the multimodule adapter option. The bubble memory is treated by the system as an additional disk drive with the same file structure and directory structure as a diskette. Additional bubble memory is recommended for systems requiring portability, since the bubble memory boards are completely housed in the main enclosure.

Dual Processors

A second 8085A-2 processor board added to the system increases processing throughput by allowing one program to run on one processor while another program is running on the other processor. For example, while one processor is compiling or assembling a program, the other processor can be used to edit a text file.

This option consists of a single board that is installed through the removable rear panel of the main enclosure. The board contains an 8085A-2 microprocessor with 64K bytes of RAM and is functionally the same as the base processor. However, it does not include the integrated serial and parallel I/O ports and cannot be used with the PROM Programmer or Emulator plug-in modules. The two processors share the flexible disk drives, CRT, and multimodules. The keyboard is used by one processor at a time.

Software to control the optional board is included in the operating system. Optional processor features that extend the capabilities of the development system are covered in Chapter 9 of this manual.

Plug-In Modules

Plug-in modules slide into a slot on the side of the main enclosure adding features to the system. These options include both hardware and software. The following plug-in modules are available (see figure 1-5):

- Emulators that provide debug capabilities for different families of microprocessors and microcontrollers

- PROM Programmer Personality Modules that accept different families of PROMs for programming

For example, the EMV-51 emulator aids in debugging applications based on the MCS-51 family of microcontrollers. The emulator hardware plugs into the side of the iPDS system while the software runs on the base processor. Emulator debugging features are discussed in Chapter 2.

The PROM programmer personality modules plug into the slot on the side of the system and allow programming and verification of Intel EPROMs and E^2PROMs. The PROM programmer software runs as a utility program under the operating system.

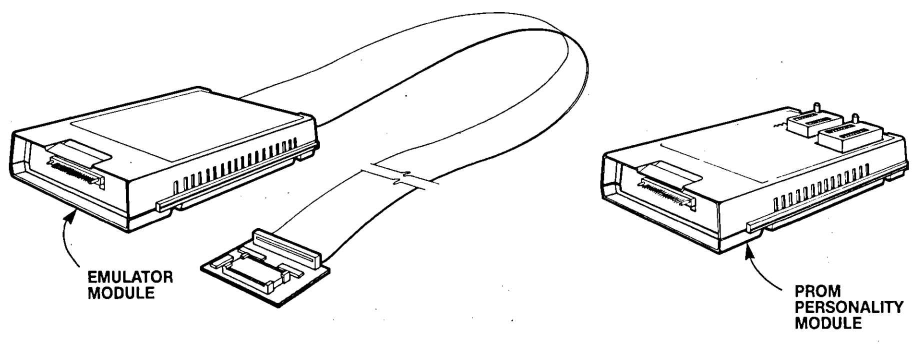


Figure 1-5 Plug-In Modules

Multimodules

The development system can be expanded through the multimodule adapter option which allows a maximum of four multimodule boards to be added. Multimodule boards are small, special function boards which use the iSBX bus to interface to the CPU.

The multimodule adapter board is installed through the rear panel of the system.

The iSBX multimodule boards available for the iPDS system are:

- iSBX 251 Bubble Memory Multimodule Board

- iSBX 350 Parallel Port Multimodule Board

- iSBX 351 Serial Port Multimodule Board
- iSBX 488 IEEE-488 Interface Multimodule Board

The iSBX 251 multimodule board is discussed in this chapter in the section entitled "Add-On Mass Storage". The iSBX 350 and iSBX 351 provide parallel and serial I/O in addition to the parallel printer port and the serial I/O port already on the base processor board. The iSBX 488 provides additional system expansion through the IEEE-488 General Purpose Interface Bus (GPIB).

Software routines for these multimodules must be provided by the user. Many of these routines are available from the INSITE Software Library. See Chapter 8 of this manual for technical information to aid in writing custom I/O drivers.

Software

The following software is supplied on the system disk:

- ISIS-PDS operating system
- Operating system commands
- CREDIT text editor
- DEBUG command
- Customer confidence test
- ASM-80 Macro Assembler
- MCS-80/85 software development utilities

Assemblers, high level languages, and utilities for different target microprocessors are optionally available to aid the software development effort. See figure 1-6.

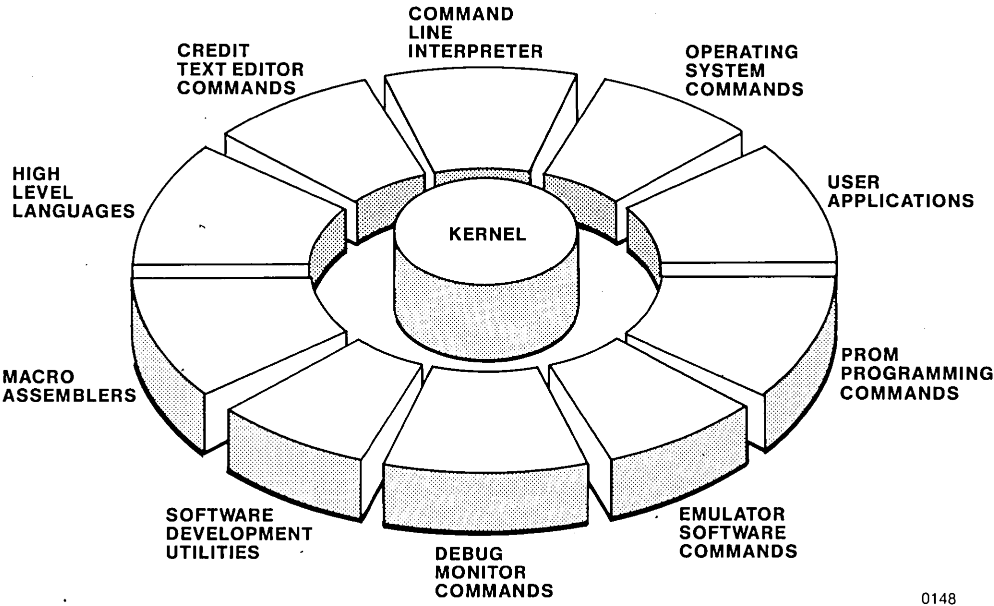


Figure 1-6 Overview of Operating System Software

Operating System

The ISIS-PDS operating system provides an easy-to-use set of commands (including a HELP command) to control system operations. It also includes a set of routines that the systems programmer can incorporate into applications software. The commands and routines provide powerful features to control disk files, to handle I/O from different peripheral devices, and to control the execution of programs in a standard way.

Assemblers

Macro assemblers produce relocatable object code for different families of microprocessors and microcontrollers, such as the MCS-85 and the MCS-51. The code is compatible with the code produced by high level language compilers for the same chip. Therefore, modules written in assembly language can be combined with modules written in a high level language using the link and locate utilities.

High Level Languages

High level languages help to reduce system design and maintenance costs by allowing the programmer to design software at a more abstract level than with an assembler. PL/M, a block structured language, is available for several families of chips. Other languages available include FORTRAN and BASIC.

Utilities

Utility programs are available to edit text, to link and locate program modules, to convert file formats, to debug MCS-80/85 programs, to program PROMs, and to control emulation vehicles. All of these utilities aid in producing reliable, efficient software.

Other Software

Since the ISIS-PDS operating system is functionally compatible with the ISIS-II operating system, most ISIS-II software runs on the development system without modification.

Overview of System Publications

A library of technical manuals support development work using the system and its options. The basic manuals are shipped with the system, and additional manuals are provided with the optional hardware or software packages to which they apply.

The Literature Kit shipped with the basic system contains a customer letter, a software registration card, other informational literature, and the following technical manuals:

- *iPDS™ User's Guide,* order no. 162606
- *iPDS™ Pocket Reference,* order no. 162607
- *MCS™-8085 Utilities User's Guide for 8080/8085-Based Development Systems,* order no. 121617

- *ISIS-II 8080/8085 Macro Assembler Operator's Manual,* order no. 9800292
- *8080/8085 Assembly Language Programming Manual,* order no. 9800301
- *8080/8085 Assembly Language Reference Card,* order no. 9800438
- *ISIS CREDIT™ CRT-Based Text Editor User's Guide,* order no. 9800902
- *iPDS™ Field Service Manual,* order no. 143861.

A three-ring binder and tabs to mark the beginning of each section of the user's guide are provided.

Copies of the technical manuals are shipped with the products that they support. Additional copies of any of these manuals may be ordered through the Literature Department, Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051. The address is also on the back of the title page of this manual.

Use the order numbers shown above when placing an order. Use the order number 164181 to order an extra three-ring binder.

The Literature Department also distributes other Intel literature, such as application notes, magazine article reprints, and brochures on new products.

NOTE

Because of the compatibility between the ISIS-II and ISIS-PDS operating systems, some of the manuals provided for this development system refer to the ISIS-II operating system and were written for the ISIS-II version of the software.

In these cases, the operation of the product is identical under either system. The ISIS-II version of the manual is shipped with the ISIS-PDS version of the product.

Hardware Installation and Checkout

The first material to consult upon receiving a new system or a new option is Appendix A of this manual. Follow the installation instructions before attempting to use the system or an option. Appendix A also contains specifications for I/O connectors.

Hardware troubleshooting is covered in detail in the *iPDS™ Field Service Manual,* order no. 143861, which contains schematic drawings, troubleshooting procedures, and the theory of operation for the system. This guide is included in the Literature Kit shipped with the system.

System Operations

Once the system is installed and ready to run, consult Chapters 3, 4, and 5 of this manual. Chapter 3 explains how to initialize the system, how to use the disk drives, how to give simple commands to the operating system, and how to adjust operator controls on the system. The tutorial demonstration included in Chapter 3 illustrates system initialization, backing up the system disk, and running the confidence test. Running the confidence tests before proceeding to Chapter 4 is recommended.

Chapter 4 provides more detailed information on the operating system commands and gives examples of how and when they are used. The several tutorial demonstrations included in Chapter 4 illustrate how to use these commands.

Chapter 5 is organized as a reference guide for the operating system commands and contains complete information for each command.

After becoming familiar with the material in Chapters 3, 4, and 5 of this manual, consult the *iPDS™ Pocket Reference* for summaries of the commands.

Appendix E contains tips for efficient operation with specific system configurations.

Text Editing

An important feature of the system is the CREDIT text editor. The editor is used extensively in software development to enter the source code for the software modules. To use the CREDIT editor, refer to Chapter 6 of this manual and the *ISIS CREDIT™ CRT-Based Text Editor User's Guide* provided in the literature kit.

Software Debugging

The DEBUG command provides MCS-80/85 software debugging facilities. It is described in Chapter 7 of this manual.

Systems Programming

The operating system provides a set of system calls and standard I/O routines that can be incorporated into a user-written program. A technical description of these features can be found in Chapter 8.

Dual Processing

Information on the optional processor is in Chapter 9 of this manual. Installation instructions for the board are in Appendix A.

PROM Programming

EPROM and E^{2}PROM devices are programmed using the Intel PROM Programming Software (iPPS) that runs under the operating system and controls the optional PROM Personality Modules. An EMV/PROM adapter board is also required to program PROMs on the iPDS development system. The iPPS commands are covered in the *iUP-200/201 Universal Programmer User's Guide,* Order No. 162613. Each Personality Module is shipped with a reference manual containing specifications for the particular module. Installation instructions for the EMV/PROM adapter board and the PROM personality module are in Appendix A.

Microprocessor Emulation

Each emulator is shipped with an operating instructions manual and a pocket reference describing how to use the emulator.

Multimodule Expansion

A hardware reference manual is shipped with each iSBX multimodule board to provide detailed specifications for the board. User-written I/O routines for multimodules can be added to the operating system as described in Chapter 8. General installation instructions for double and single wide multimodule boards are given in Appendix A.

Applications Programming

Each language provided for developing applications software is described in a programmer's reference manual and an operating instruction manual. These two manuals are sometimes combined into a single user's guide, and a pocket reference is also provided.

Software development utilities are covered in the *MCS-80/85 Utilities User's Guide for 8080/8055-Based Development Systems.*

Technical reference information on target microprocessors and microcontrollers can be found in the family user's manual for the chip.

Notational Conventions

Throughout this manual, procedures and operations that can be carried out on the system are described. Because of the general nature of many of these procedures and operations, it is not possible or desirable to list all of the correct ways of carrying out a given task. Instead, general classes of procedures are described using special symbols or notational conventions. Notational conventions are described in Chapter 5.

Other Conventions

In addition to notational conventions, several standard formats have been adopted in describing commands, system calls, and error messages. These formats are described prior to their use in Chapters 5, 7, and Appendix B.

CAUTION, WARNING, and NOTE Symbols

A section of text introduced by the symbol

gives instructions necessary to avoid possible damage to equipment or loss of stored information.

A section of text introduced by the symbol

gives instructions necessary for safety reasons.

A section of text introduced by the symbol

NOTE

gives emphasis to comments with special significance for the user.

Commonly Used Terms

Many special terms and abbreviations are encountered when using computer products. In most cases, specialized vocabulary is needed to clearly explain the technical concepts involved. However, to the new computer user, the unfamiliar terminology can be a source of confusion and frustration.

To help the new computer user, a glossary is included before the Index to define terms and abbreviations. Anyone unfamiliar with computer terminology should skim over the glossary before starting on the rest of the manual, and then refer to it later whenever it is needed.

The Development Task

A typical product development cycle is illustrated in figure 2-1 involving software development, hardware development, integration, and testing.

A development system should aid in all phases of product development. The iPDS development system can be used both in the software and hardware development, as well as in production testing after the development and in customer support after the product is in the field.

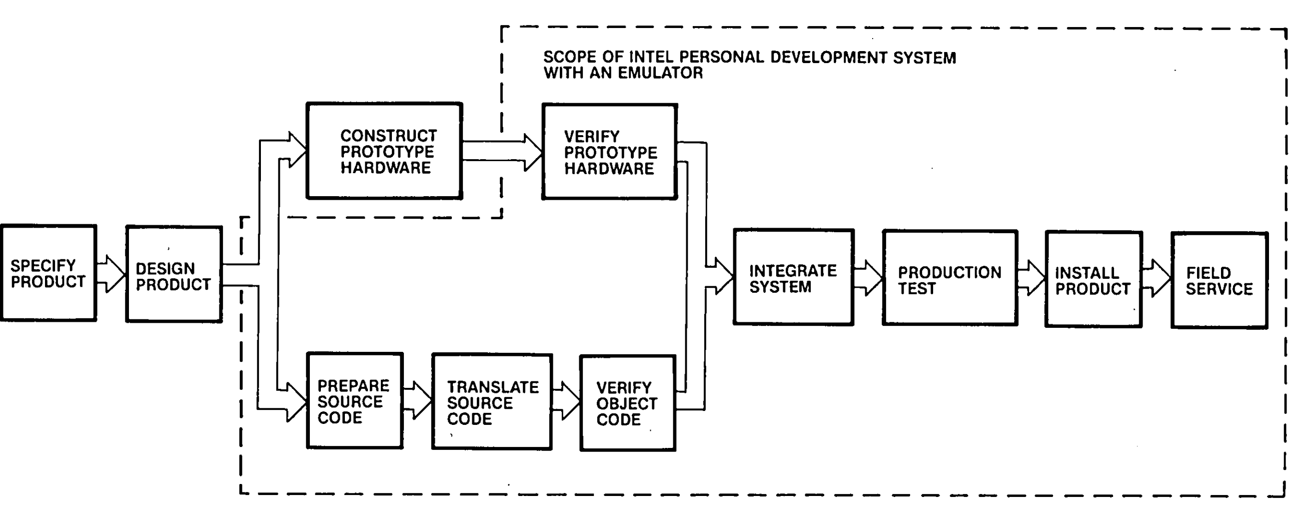


Figure 2-1 Typical Product Development Cycle

Software Development

Software development involves programming the target processor to correctly perform the required task using instructions eventually stored in the product's own memory. This effort requires the following software:

- Text editor program for creating the source code for the software
- Assembler or compiler with related support programs to produce a machine readable form of the software
- Various utility programs that manipulate and test the software as it is developed

With support programs that run under the Intel System Implementation Supervisor (ISIS-PDS), the software development phase of a project can be completed. Software testing and debugging can be carried out with operating system programs and with the aid of emulators available as system options.

The software development task is illustrated in figure 2-2.

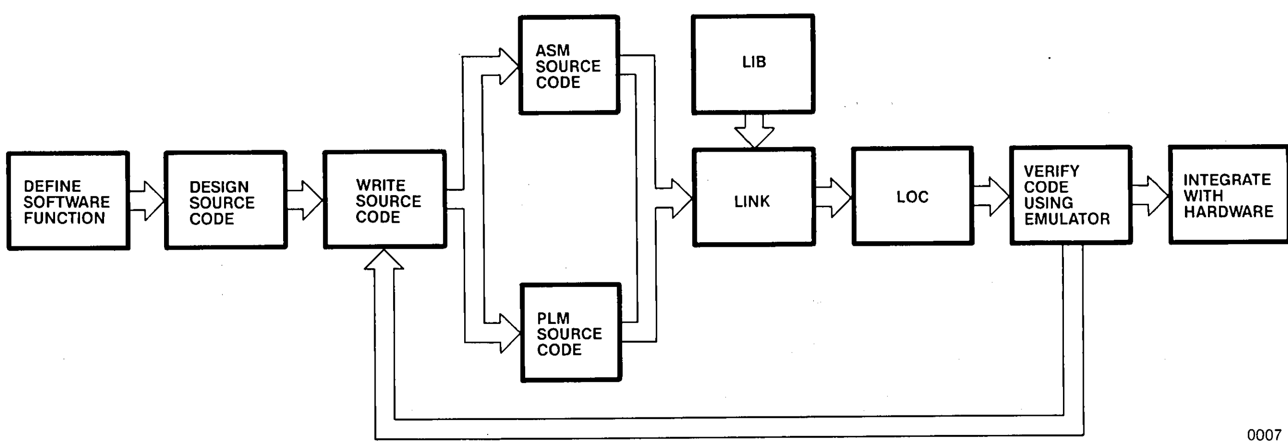


Figure 2-2 Software Development Cycle

Hardware Development

Hardware development involves designing the circuits that make up a product (the microprocessor, memory, and input/output circuits) and designing the relationship of these circuits to one another. As the hardware prototype is assembled, it can be tested and debugged using the appropriate emulator.

If the product includes EPROMs, E^2PROMs, or a part containing EPROM or E^2PROM, a PROM Programmer is required to store the control program in the product's own memory. See figure 2-3.

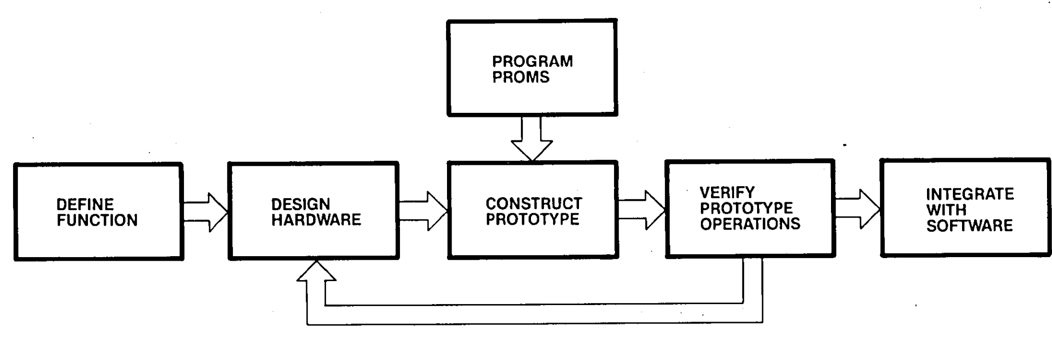


Figure 2-3 Hardware Development Cycle

Integration

The integration of the hardware and the software involves further testing and debugging, requiring the use of the emulator as well as the operating system support software until the project is completed.

Production Testing

During manufacture of the new product, test programs running on the emulator can be used to test samples and to maintain quality control during production. Figure 2-4 illustrates production testing.

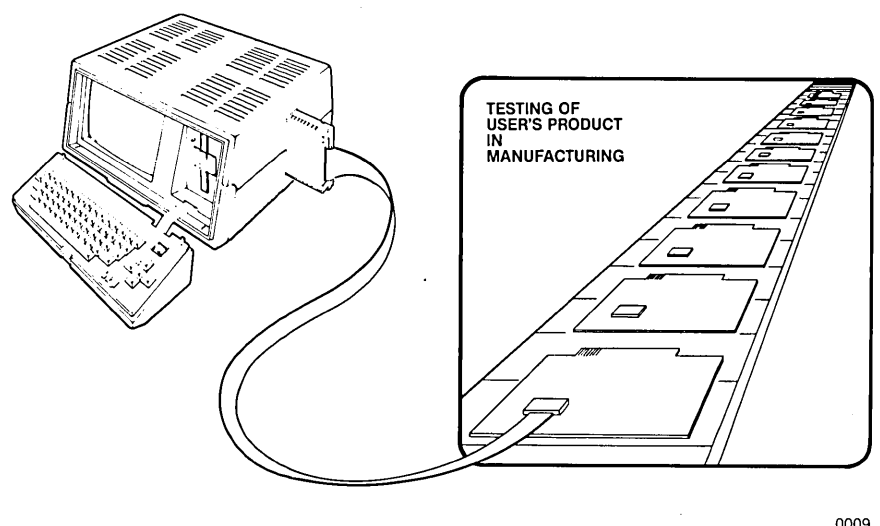


Figure 2-4 Production Testing

Field Service

Because of its portability, the iPDS development system can be used as a diagnostic tool in field service. Additionally, it can be used on-site for product installations and customer training. Figure 2-5 illustrates the field service use of the system.

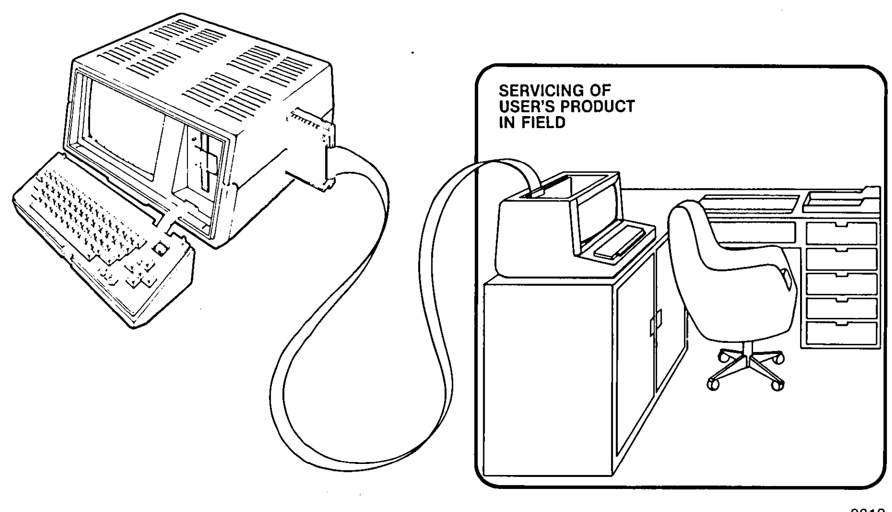


Figure 2-5 Field Service

The Development Tools

The tools required to develop a microprocessor/microcontroller based product differ from the tools used to develop electronic products not incorporating micros.

For a product without micros, the oscilloscope, the logic analyzer, and meters serve as useful development tools. They provide the logic designer with signals generated at different points in the circuit, allowing testing and debugging of the hardware. Software development is not required for this type of product.

For micro-based products, however, the traditional electronics development tools are not sufficient for two reasons. First, they do not support the software development required in a micro-based product. Second, many of the circuits and signals previously available to the designer are integrated onto a single silicon chip and are not accessible through the connector pins on the chip's package.

A development system, such as the iPDS system, provides software development tools and emulators for many families of target micros to satisfy the requirements of a micro-based development task. An optional PROM programmer for EPROMs or E^2PROMs is also available.

Software Development Tools

Writing the source code and debugging the resulting object code are the two most time consuming parts of the software development cycle. Therefore, software development tools should concentrate on supporting these two parts of the cycle.

A development system text editor should ensure ease in entering source code and should also provide high level commands such as block COPYs, block MOVEs, READs and WRITEs to files, and FIND/SUBSTITUTE text to correct errors. Additionally, the editor should allow the automatic execution of sequences of commands, so the user's time is not wasted performing repetitive editing tasks.

The ISIS-PDS CREDIT text editor is designed for software development applications. Since it is a CRT-based text editor, it provides constant visual reference to the text being edited, making text entry and text corrections easy. It provides the advanced commands needed, including macros and compatibility with the ISIS SUBMIT command, to automate repetitive editing tasks.

Language translators should reduce the time spent debugging the software by supporting modular program development and by producing debug data such as cross reference lists and symbol tables. With the language translators available for the iPDS system, program modules can be developed independently and can then be linked and located to form a single software system. Both assemblers and high level language translators produce debug data to reduce the time spent troubleshooting the software.

Emulators (discussed in the next section) also aid in the debugging and verification phase of the software development cycle.

Emulators

To debug a product efficiently, the user must be able to exercise the product (for example, run the software) under controlled conditions and monitor the results. By repeatedly exercising the product and comparing the expected results with the actual results, the user can identify and solve the problems (bugs) in the product.

An emulator has the features to provide a controlled environment for exercising the product and, then, to monitor the results. It can duplicate the behavior of a target microprocessor/microcontroller and, at the same time, can provide information to the user to aid in debugging the hardware and software being developed.

For example, emulators have a breakpoint feature that allows the user to specify a portion of the program to be run real time and then stop. Once stopped at the breakpoint, the emulator acts as a window to the internal registers and logic signals that are inaccessible from the connector pins. In this way, the internal state of the micro can be examined and altered. Data about the internal state of the chip can also be collected and saved in a buffer called the trace buffer.

Additionally, the emulator accepts debug data, such as symbol tables, produced by the language translators. The programmer can reference locations in the program with the symbolic debug information, such as module names and variable names, rather than by using absolute memory addresses.

Another advantage of using an emulator is that functional hardware is not required to begin software debugging. The emulator duplicates the behavior of the target micro and provides some resources, such as memory, that can be used until the hardware prototype is more complete.

The software that controls the emulator consists of a set of commands that the user can enter to directly control an interactive debugging session. Also, sequences of emulator commands can be executed automatically providing the basis of manufacturing and field test routines.

Summary

In summary, a development system should offer the following tools to support the development task:

- Text editing facility
- Language support for software development
- Target microprocessor/microcontroller emulation
- PROM programming capability

In addition, a development system should provide features common to all computer systems:

- File handling utilities
- System configuration utilities
- Job control utilities
- Resource control utilities
- Support for common peripherals

All the tools offered on the development system should be compatible with one another forming an integrated environment for development and testing of products. The ISIS-PDS operating system ensures compatible tools provided for micro-based development projects.

Overview of the Development Cycle

In this section, a summary is given of the sequence of events to follow in developing a micro-based product using the iPDS development system and an emulator.

- Complete the specification for the prototype hardware design, software control logic, and integrated system performance. The CREDIT text editor can be used in preparing this document.

- Organize both the hardware and software design into logical blocks that are understandable, have well-defined inputs and outputs, and are easy to test. Methodical design techniques, such as top down structured design, reduce the time required later for prototyping, programming, testing, and modification. The CREDIT text editor helps in preparing reports on the development progress.

- Program the software modules in PL/M or assembly language, naming and storing program modules as files under the operating system. Compile or assemble the modules using the options necessary to produce debug data. Link and locate the combinations ready for testing to create object code (machine language) version containing the debug data. Manually, verify each module as it is completed before running it on the machine. The CREDIT text editor is used to enter the source code. A number of language translators and development utilities are available to produce object code. See the section on "High Level Languages" in Chapter 1 for further references on the languages available.

- As the software modules are ready for testing, load them into the emulator and execute them. The emulator can be used before any prototype hardware is available. The emulator provides single step execution, breakpoints, software trace capabilities, and processor register examination for testing and debugging the software. It also provides RAM in which the module can run, allowing patches to be made quickly and easily. In later stages, PROM or ROM can be substituted for the RAM in the prototype hardware.

- As software modules pass the initial stages of check-out, they can be loaded in the emulator's memory for real-time testing.

- The emulator is plugged into the microprocessor/microcontroller socket of the user's prototype system. Hardware prototyping can begin with the micro socket alone. As each part of the hardware becomes available, it can be added to the prototype. In this way, modules can be tested as they become available. The emulator's ability to execute in single step mode, to examine or modify the memory and processor registers, to trace the program flow, and to break in real time mode, provides the user with substantial power to debug the hardware system.

- Debugging and testing can proceed through each hardware and software module, using emulator commands to control execution, to check that each module gets data or control information from the correct source, and to place correct data in the proper locations for subsequent modules to use.

- Eventually, all hardware and software is tested together. The emulator is connected to the prototype through the prototype microprocessor/microcontroller socket, so all operations of the system can be tested.

- After the prototype has been completely debugged, the emulator can be used to verify samples during production testing. The test procedures developed for the final prototype testing can serve as the basis for production test and diagnostic routines.

- The test and diagnostic routines used during manufacture can serve as the basis for a set of field diagnostics used in service and repair of equipment in the field.

In summary, the product should be designed methodically, and the development system should be used at every step to increase efficiency during the project.

CHAPTER 3
BASIC SYSTEM OPERATION

Hardware Operation

The hardware subsystem consists of a CPU, memory, mass storage, and peripherals. Figure 3-1 shows the outward appearance of the basic system. The operating procedures for the hardware components are described in the following section. Procedures are given for operating the rear panel controls, the keyboard, and the disk drives.

Refer to Appendix A of this manual for installation instructions before attempting to operate the system.

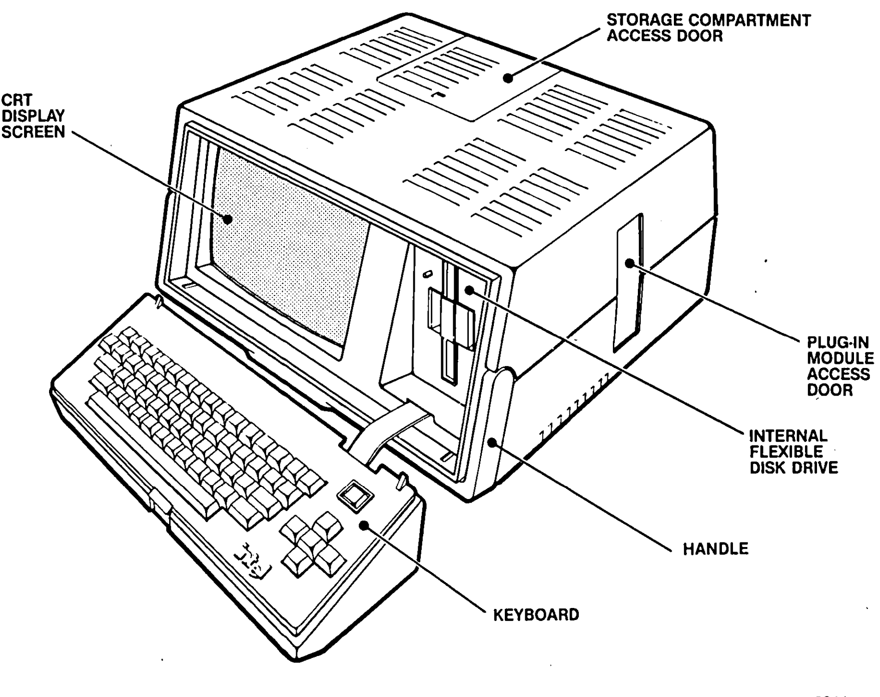


Figure 3-1 Basic System

Rear Control Panel

Figure 3-2 illustrates the rear panel and the location of basic operator controls.

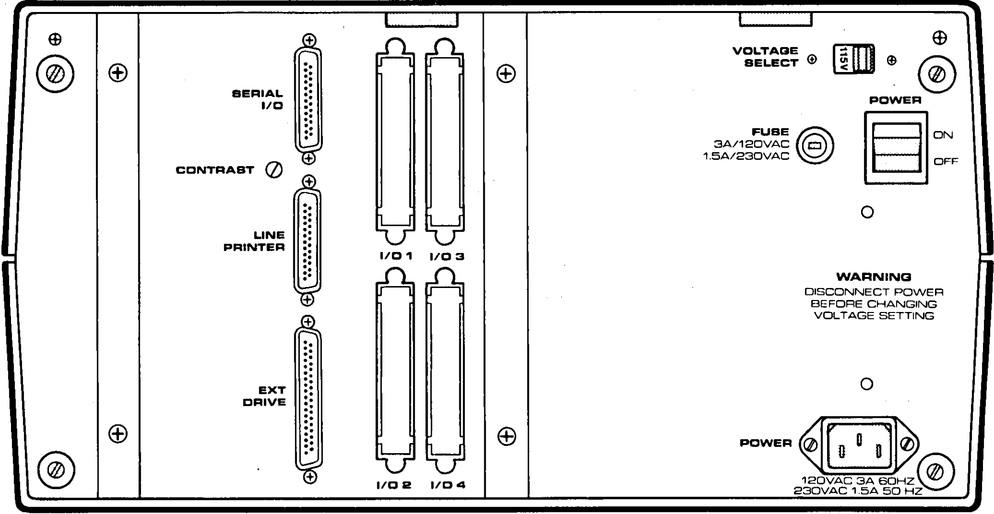


Figure 3-2 Rear Panel Controls

POWER	The switch labeled POWER (upper right corner of rear panel) turns on the power to the basic system and the integral disk drive and resets the system.
POWER PLUG	The power cord is plugged into the socket labeled POWER located in the lower right corner of the rear panel.
VOLTAGE SELECT	The voltage select switch can be set to 115 or 230 volts to accommodate the available line voltage.
CONTRAST	The contrast adjustment for the CRT display screen is on the rear panel next to the serial I/O port.
SERIAL I/O	The serial I/O connector accepts a plug for an RS-232 compatible device which is controlled by the base processor and is not accessible by the optional processor. Technical information on the connector can be found in Appendix A.
LINE PRINTER	The line printer connector accepts a plug for a Centronics*-compatible printer which is controlled by the base processor and is not accessible by the optional processor. Refer to Appendix A for information on the connector.

*Centronics is a trademark of Centronics, Inc.

EXT DRIVE	The external drive connector allows up to three additional flexible disk drives to be daisy-chained to the system with flat ribbon cable. Refer to Appendix A for instructions on connecting the external drives.
I/O 1 through I/O 4	Four multimodule I/O connectors are provided to allow connection of peripherals to the optional multimodule boards. See Appendix A for installation instructions.

Removable I/O Panel

The compartment behind the removable panel slides out from the rear of the system to allow installation of optional boards. The panel contains four knockouts for peripheral connection to the multimodule boards. See figure 3-2 for the knockouts and figure 3-3 for the connectors. The tabs on the knockout must be cut prior to tapping them out. Appendix A contains instructions for removing the knockouts and installing connectors.

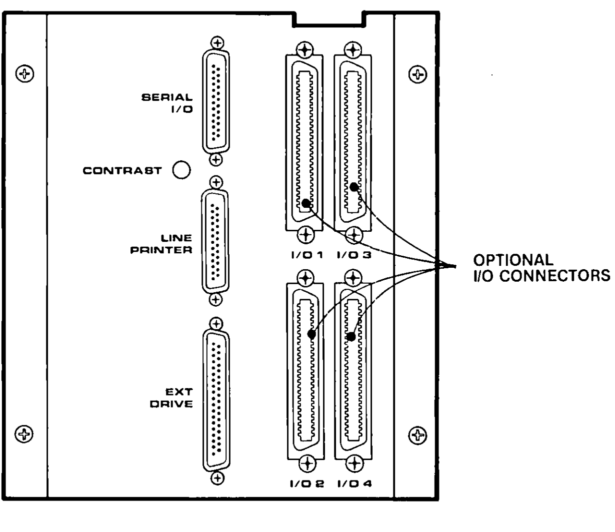


Figure 3-3 Removable I/O Panel

Storage Area

A storage area is accessed from the top cover to allow the storage of two plug-in modules with cables. See figure 3-4. Diskettes can also be stored safely in this area.

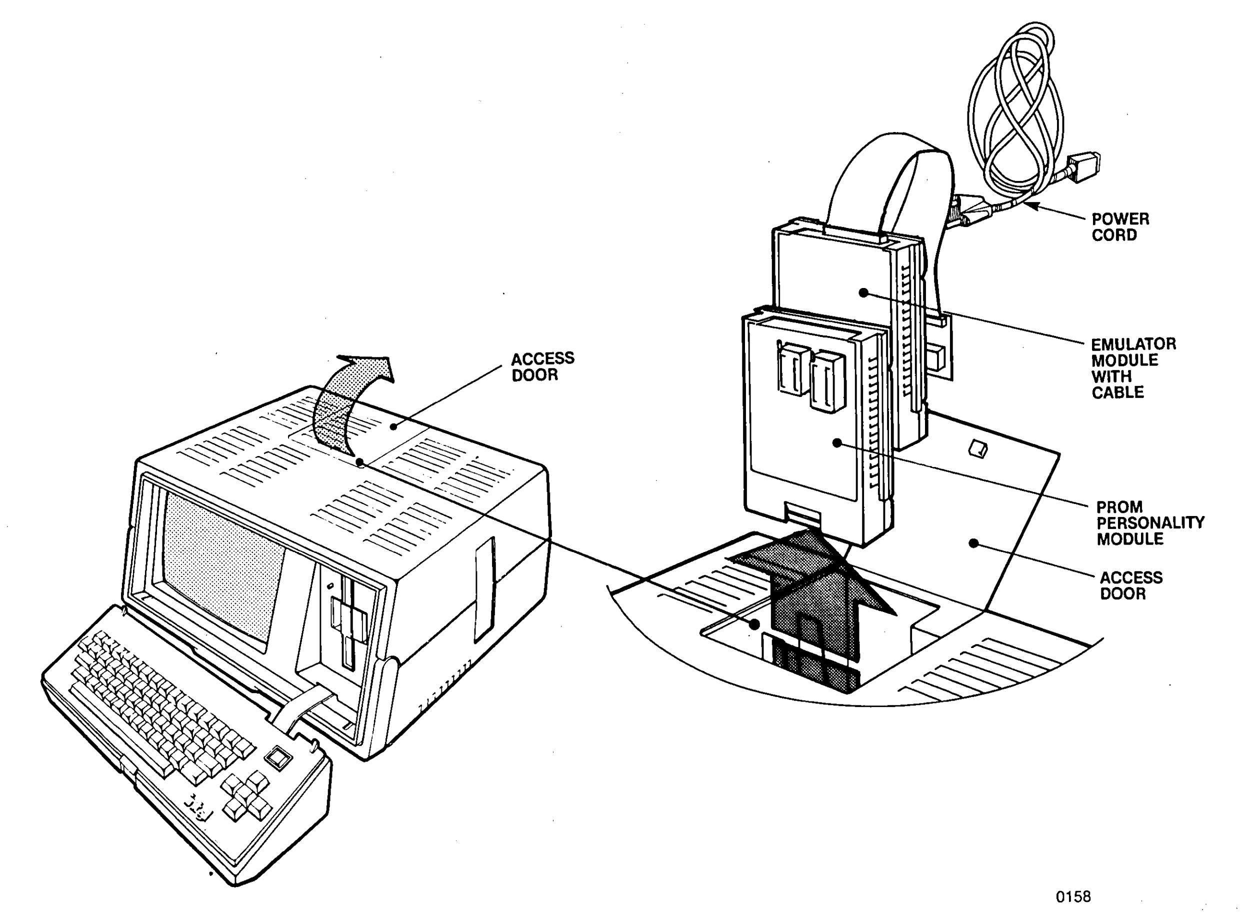


Figure 3-4 Accessing the Storage Area

Powering the System On and Off

NOTE

Before powering the system on, see Appendix A for detailed installation instructions including instructions for setting the line voltage to 115 or 230 Vac and instructions for changing the power connector to the type of connector required locally.

To turn the system on:

1. Ensure that the power cord is disconnected. See figure 3-2 for the location of the plug.
2. Ensure that the Voltage Selector switch is set to the available line voltage. See figure 3-2 for the location of the switch. See Appendix A for instructions on setting the line voltage switch.
3. Connect the power cord to the system and to an external power source. See figure 3-2 for the location of the iPDS power plug.
4. Set the POWER switch to ON. See figure 3-2.

To turn the system off:

1. Remove all flexible disks as described in this chapter in the section entitled "Disk Drives".

2. Hit the RESET key to ensure the disk head is back as far as possible.

3. Turn off the power to any external drives.

4. Turn off the power to the main system.

5. Insert the diskette card to transport unit.

Keyboard

Figure 3-5 shows the keyboard, the basic user interface to the system. Two plastic guides on the back of the keyboard housing attach the keyboard housing to the main enclosure. A flat cable connects the keyboard to the main processor board through a slot on the front of the main enclosure below the flexible disk drive. This cable plugs into a connector on the back of the keyboard housing as shown in figure 3-5.

Through the keyboard, commands and data are entered to the operating system. A temporary holding area in memory, called a line editing buffer, stores the characters typed at the keyboard until the RETURN key is pressed or 122 characters are entered.

Before pressing RETURN, commands and data can be edited or even canceled from the buffer. After pressing RETURN, commands and data can be re-edited. Command line editing and re-editing are described in detail at the end of this chapter.

The keyboard includes an Auto Repeat feature. Any key that is held down will be automatically repeated as if it were repeatedly pressed and released. This feature is useful in editing text files.

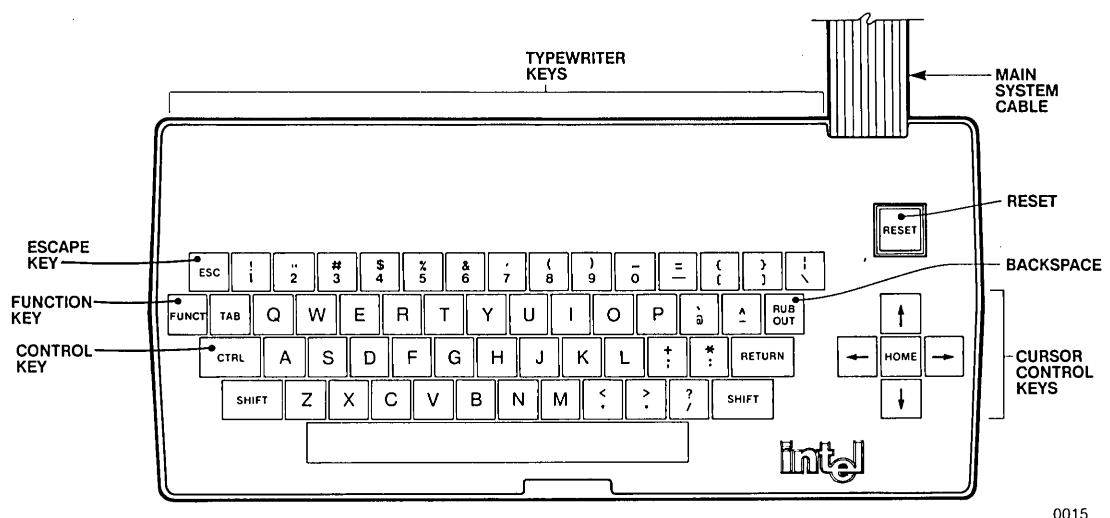


Figure 3-5 The Keyboard

In addition to the standard typewriter keys, the keyboard has several special purpose keys. For example, the CTRL key works with other keys to form control characters. Control characters perform control functions, such as line editing. Control characters are listed in Table 3-1.

The FUNCT key also works with other keys to form function characters. Function characters also perform control functions. For example, if the system contains dual processors, the keyboard is assigned to either the base processor or the optional processor through function characters as described in Chapter 9. The function characters are also listed in Table 3-1.

The demonstration at the end of this chapter illustrates the use of these keys. Some of the special keys are also discussed in connection with the operation of the software.

Keyboard characters and the functions they perform are summarized in table 3-1.

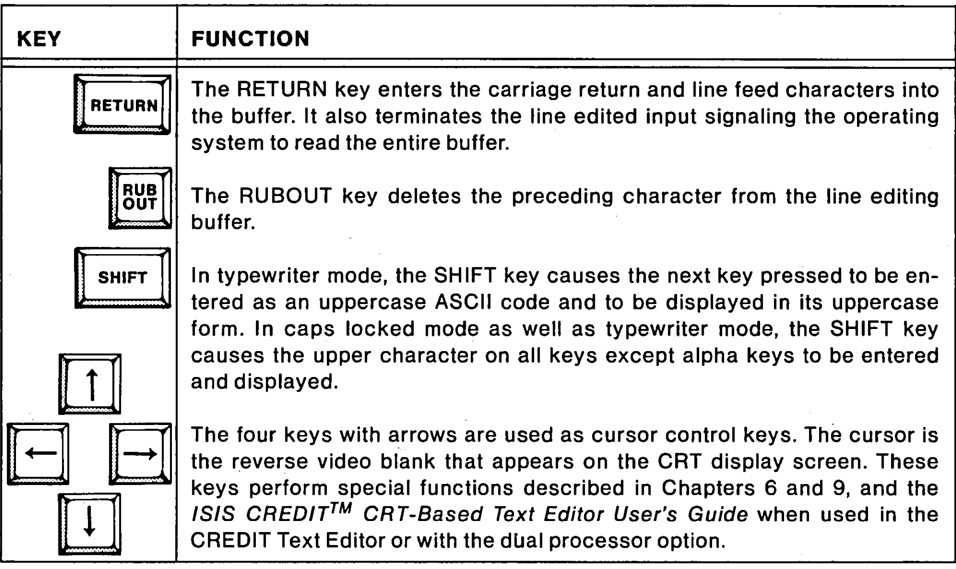
Table 3-1 Keyboard Characters and Functions

KEY	FUNCTION
CTRL	The CTRL (Control) key is used with other keys to perform control functions. A key whose function is changed by the CTRL key is called a control character. To enter a control character, hold down the CTRL key while typing the character. This action is similar to using the SHIFT key on a typewriter. A control character generates a single control code in the line editing buffer even though two keys are pressed. Some examples of control characters are CTRL-R and CTRL-S. Valid control characters are defined below. Control characters are also used as commands within the CREDIT Text Editor and are described in Chapter 6 and the *ISIS CREDIT™ CRT-Based Text Editor User's Guide.*
CTRL A	CTRL-A inserts a character into a command line during command line editing mode. See the ESC key.
CTRL B	CTRL-B is used for two purposes. During command line editing mode, it moves the cursor to the beginning of the command line being edited. It also acts as an alternate ESC key when not in editing mode. It can be used in the display of graphics symbols within programs that use the ESC key for other purposes. See Chapter 8 for information on the use of graphics symbols. Appendix C contains a chart of the graphics symbols available.
CTRL D	CTRL-D deletes the preceding character during command line editing mode. See the ESC key.
CTRL E	CTRL-E is used in processing SUBMIT files. It is described in Chapters 4 and 5.
CTRL L	CTRL-L moves the cursor to the end of the command line being edited during command line editing mode. See the ESC key.
CTRL P	CTRL-P causes a character that normally would be interpreted as a line editing character and perform an editing function (Control Characters, RUBOUT, etc.) to be entered literally into the line editing buffer without performing any function.
CTRL Q	CTRL-Q resumes the display after a CTRL-S.
CTRL R	CTRL-R displays the current contents of the line editing buffer.

Table 3-1 Keyboard Characters and Functions (continued)

KEY	FUNCTION
CTRL S	CTRL-S suspends the display on the CRT screen. This function is useful when output from a program is scrolling off the screen too quickly.
CTRL X	CTRL-X performs two functions. It deletes the entire contents of the line editing buffer without terminating the buffer. A number sign (#) is displayed followed by a carriage return and line feed. CTRL-X also terminates the command line editing mode. See the ESC key.
CTRL Z	CTRL-Z deletes the entire contents of the line editing buffer and also terminates the buffer. It displays a carriage return and linefeed and the operating system prompt appears as the first characters of the next line.
ESC	The ESC (escape) key is used to enter command line editing mode to correct or change command lines. After editing mode has been entered, several control characters can be used to modify the command line. It is also used in the display of graphics symbols. See Chapter 8 for information on graphics symbols.
FUNCT	The FUNCT key is used with other keys to perform predefined functions. The functions are either user defined or are predefined by Intel supplied software. To enter a function character, hold down the FUNCT key while another key is pressed and then release both keys. Valid function characters are defined below and in Chapters 4, 5, and 9.
FUNCT HOME	FUNCT-HOME controls the assignment of the keyboard in dual processor systems. See Chapter 9.
FUNCT R	FUNCT-R is used to reload the ISIS-PDS operating system if interrupts are enabled. See Chapter 9 for information on using this function in dual processor systems. The RESET key generates a hardware reset for the system.
FUNCT S	FUNCT-S switches between two speeds for the CRT display. The slower rate is about ten times slower than the faster rate. See CTRL-S for another character to control the CRT display.
FUNCT T	FUNCT-T alternately switches the keyboard between typewriter mode and non-typewriter (caps locked) mode. In typewriter mode, non-shifted keys result in lower case characters while shifted keys result in upper case characters. In caps locked mode, all non-shifted alpha keys result in upper case characters. Shifted keys result in the upper character for all other keys.
FUNCT 0 thru FUNCT 9	FUNCT-0 through FUNCT-9 are user defined function keys. See Chapters 4 and 5 for details on their use.
FUNCT ↑	FUNCT ↑ is used in dual processor systems to control the display screen. See Chapter 9 for more details.
FUNCT ↓	FUNCT ↓ is used in dual processor systems to control the display screen. See Chapter 9 for more details.
HOME	The HOME key is used in text editing and with the dual processor option. See Chapters 6 and 9.
RESET	The RESET switch generates a hardware reset for the entire system. The top of the RESET switch is flush with the keyboard enclosure so it cannot easily be pressed. This feature helps prevent accidental resets. When the system is reset, any work in progress is terminated.

Table 3-1 Keyboard Characters and Functions (continued)



KEY	FUNCTION
RETURN	The RETURN key enters the carriage return and line feed characters into the buffer. It also terminates the line edited input signaling the operating system to read the entire buffer.
RUB OUT	The RUBOUT key deletes the preceding character from the line editing buffer.
SHIFT	In typewriter mode, the SHIFT key causes the next key pressed to be entered as an uppercase ASCII code and to be displayed in its uppercase form. In caps locked mode as well as typewriter mode, the SHIFT key causes the upper character on all keys except alpha keys to be entered and displayed.
↑ ← → ↓	The four keys with arrows are used as cursor control keys. The cursor is the reverse video blank that appears on the CRT display screen. These keys perform special functions described in Chapters 6 and 9, and the *ISIS CREDIT™ CRT-Based Text Editor User's Guide* when used in the CREDIT Text Editor or with the dual processor option.

Figure 3-6 illustrates how the keyboard is opened and closed. To open the system into operating position:

1. Lower the bail.
2. Set the system on the table horizontally.
3. Lower the handle until it is flush with the system's housing.
4. Press the keyboard latch and pull the keyboard down.

To close the system for carrying:

1. Insert the plastic guides on the back of the keyboard into the tabs on the system's housing.
2. Raise the keyboard until it locks into place covering the CRT and disk drive.
3. Lift the handle away from the housing.
4. Raise the unit vertically on the table.
5. Push the bail against the bottom of the cabinet.

Display Screen

Figure 3-7 shows the CRT display screen, the basic output device for the system. Characters typed at the keyboard are displayed on the screen. Characters without corresponding display symbols, e.g., control characters, are shown as a tilde (~). Error messages and prompts for additional information are displayed by programs on the screen. A cursor (the reverse video block) indicates where the next character will be displayed. Graphics symbols are described in Chapter 8.

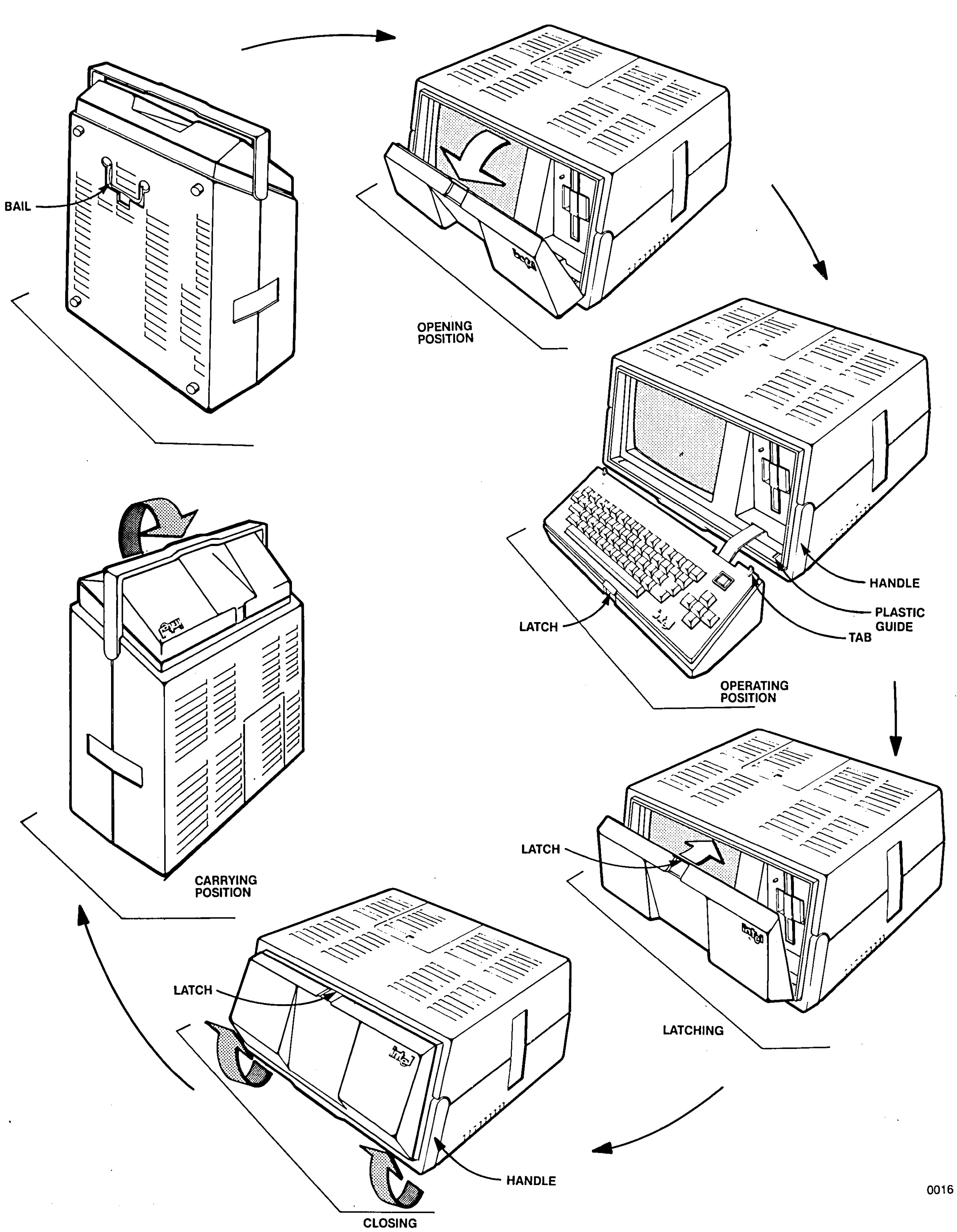


Figure 3-6 Opening and Closing the Keyboard

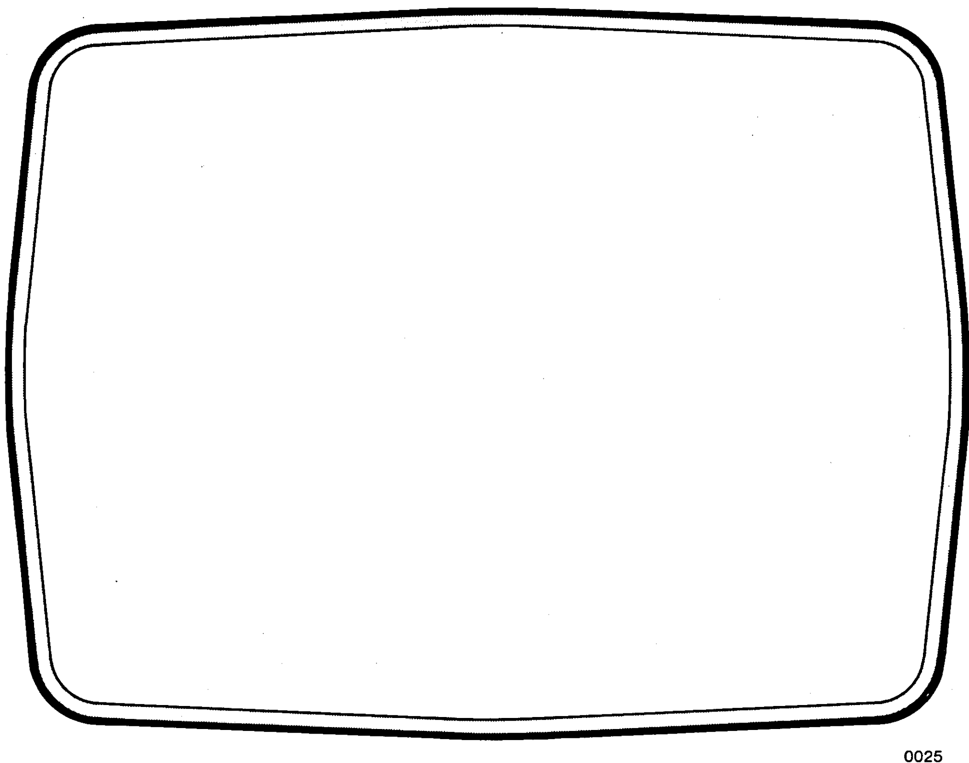


Figure 3-7 Display Screen

The 9-inch CRT has a display area that is 80 characters wide by 24 lines long. With dual processors, this area can be divided between the processors by using the FUNCT keys as described in Chapter 9.

Information on the display screen scrolls up from the bottom of the screen. Scrolling means that as new lines appear at the bottom of the screen, existing lines roll up one at a time.

If the screen is scrolling too fast, the user can slow down the display by a factor of ten by entering FUNCT-S. Alternate FUNCT-S characters restore the normal display speed. The display can be stopped by typing the CTRL-S character. CTRL-S stops the display as well as the program which was running. Any key pressed after CTRL-S is ignored except CTRL-Q which restarts the display.

Disk Drives

Flexible disk drives are shown in figure 3-8. On the front of each drive is a door, a door release mechanism, and a drive indicator which is lit during disk I/O operations. The door release mechanism is shown in figure 3-11.

A maximum of three external drives can be daisy-chained to the development system through the disk drive connector on the rear panel. The first drive is attached to the connector on the rear panel. See figure 3-2. The second drive is attached to the connector on the rear of the first external drive. The third drive is attached to the connector on the rear of the second external drive. See figure 3-8.

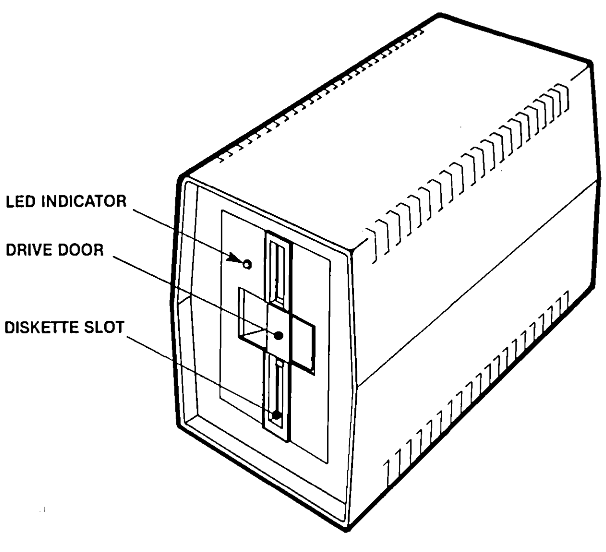


Figure 3-8 Disk Drive

A power on/off switch is also located on the rear panel of external drives.

Care and Use of Flexible Disks

A 5 1/4" diameter flexible disk (96 Tracks per inch) is used with the system. It is a double density, double sided disk (16 sectors per side). Figure 3-9 shows a flexible disk. The write enable notch determines whether or not data can be written to the disk. If the notch is covered with the write protect tab, nothing can be written to the disk. Write protect tabs are opaque, self-adhesive tabs supplied with the disk.

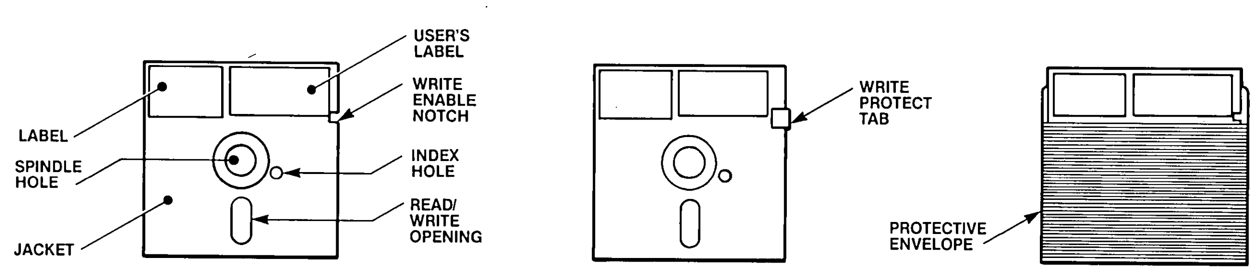


Figure 3-9 Flexible Disk

The spindle hole is used to align the disk inside the drive. The index hole is used by the drive to locate the first sector of the disk. The read/write opening is the point where the disk drive read/write head contacts the surface of the disk. The jacket protects the surface of the disk. A user label can be marked to indicate the contents of the disk and can be attached next to the disk label. Use a felt tip pen to mark the user label.

The following precautions are recommended and should be followed to protect diskettes. In addition to following these precautions, files containing valuable data should be backed up at regular intervals. Backing up a file means making a duplicate copy of the file on a different diskette. The file will then be stored on two different diskettes. If something happens to one of the diskettes, the file will still be available on the other diskette. Back-up procedures are given in the examples at the end of this chapter.

CAUTION

Ensuring trouble-free storage of data on the flexible disk requires proper care. Specific precautions follow:

- Return the disk to its envelope when not in use
- Do not touch or clean the recording surface
- Do not smoke around the disk
- Do not bend the disk or use paper clips or other mechanical devices on it
- Use a felt tip pen on the user label, not a pencil or ball point pen

The following actions can also damage or modify the data stored on the flexible disk:

- Turning on or turning off the power to the system or the power to an external drive with a disk inserted in the drive
- Opening the disk drive door while the drive select light is on
- Pressing the RESET switch while the drive select light is on

Before inserting a flexible disk, be sure that power to the system and to any external disk drives is turned on, that the LED indicator is not on, and that the drive motor is off. Insert the disk with the write enable notch as shown in figure 3-10.

The drives have a lever type door release. Close the door by pressing the latch to the left and towards the drive as shown in figure 3-11.

To remove a disk, follow these steps:

1. Ensure that the drive indicator light and the drive motor are off. If the light remains on for more than 30 seconds and a read/write operation is not in progress (no head movement is detected), disengage the drive by pressing FUNCT-R to reload the ISIS-PDS operating system.

2. To remove the disk from the drive, flip the lever out and to the right. This action releases the door and the disk can be removed. See figure 3-11.

3. Remove the disk and place it in its protective cover.

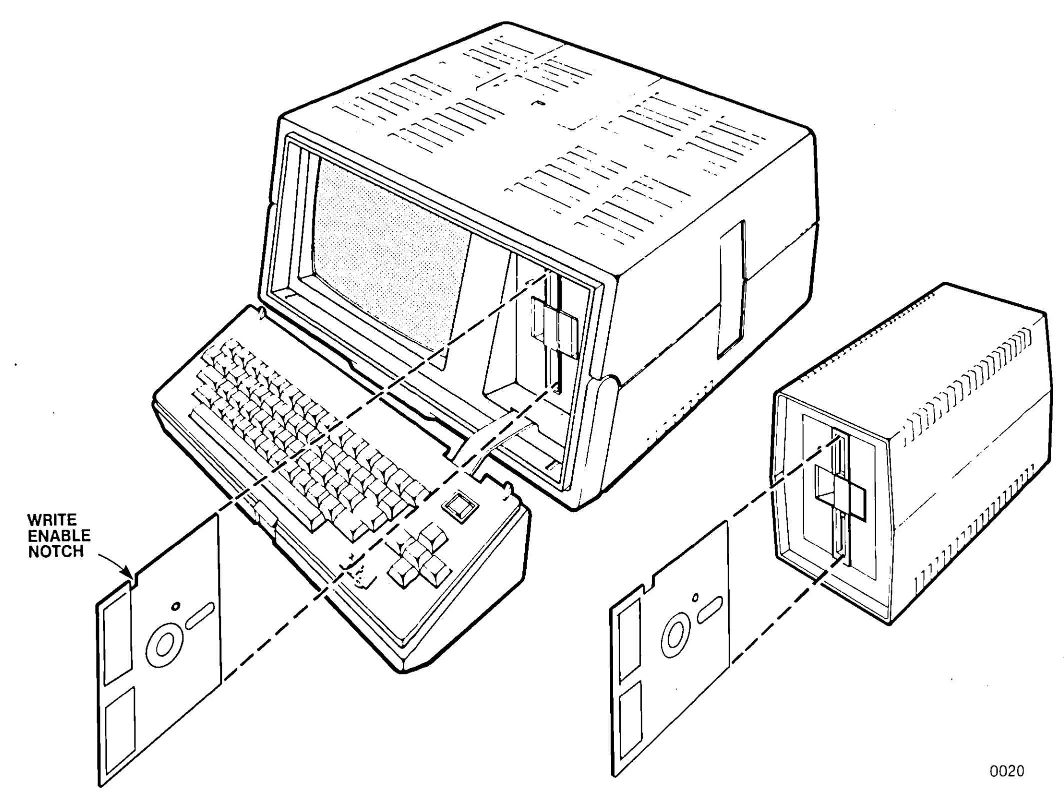


Figure 3-10 Disk Insertion

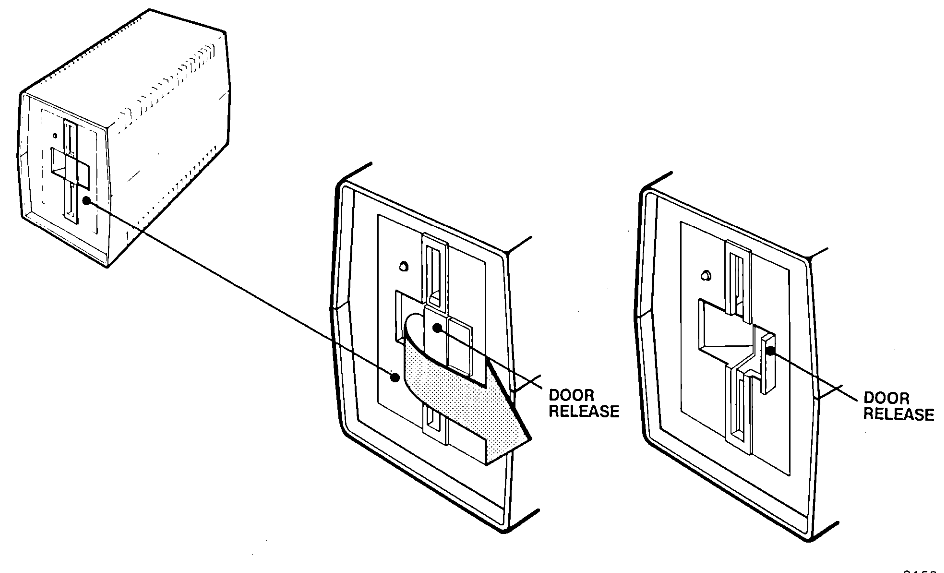


Figure 3-11 Door Release on Disk Drives

Bubble Memory

Each bubble memory multimodule board provides 128K bytes of additional mass storage. The development system supports up to two bubble memory boards. The operating system treats bubble memory the same as a disk. Bubble memory multimodule boards are treated as drive 4 and drive 5 by the operating system. Installation instructions for the bubble memory multimodule boards are in Appendix A of this manual. After installation, the bubble memory must be initialized with the IDISK command as if it were a blank disk as described at the end of this chapter. Technical details on the use of these boards are in Chapter 8.

Other Components

The plug-in modules, either emulators or the PROM programmer personality modules, are shown in figure 3-12. They are inserted into the slot on the side of the system and connect to the Plug-in Module Adapter Board. Operation of the PROM modules specific to the iPDS system is covered in Chapter 10 of this manual. Command descriptions and example of PROM programming are found in the *iUP-200/201 Universal Programmer User's Guide,* order number 162613. Emulator plug-in modules are covered in separate manuals.

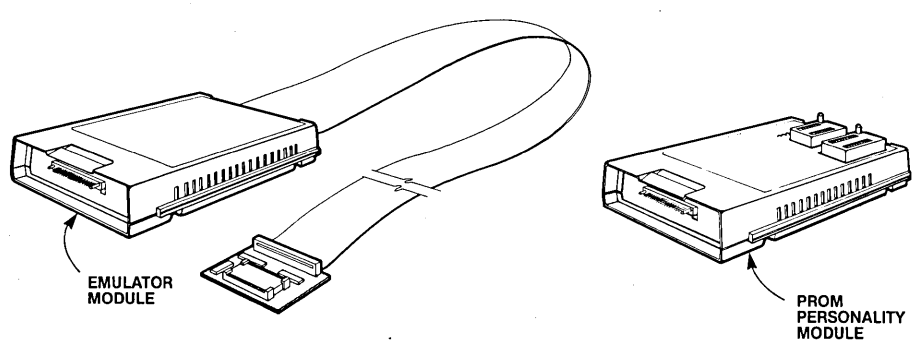


Figure 3-12 Plug-In Modules

Multimodule boards are small single purpose boards that enhance the capabilities of the system. See Chapter 1 for a list of multimodule boards available with the system. Installation instructions for multimodule boards are in Appendix A. Information on software for these boards is in Chapter 8.

A printer with a Centronics*-compatible interface can be connected to the line printer port on the rear panel of the system. See figure 3-2 for location of the connector. Appendix A details the technical specifications for attaching a line printer.

A serial device with an RS-232 interface can be connected to the serial port on the rear panel of the system. See figure 3-2 for the location of the connector. Appendix A details the technical specifications for attaching a serial device.

Dual processing is covered in Chapter 9. Installation of the additional processor board is covered in Appendix A.

*Centronics is a trademark of Centronics, Inc.

Software Operation

The software subsystem is made up of the ISIS-PDS operating system, utility programs, language translators, and user-written, application programs. The system overview in Chapter 1 describes the available software and the corresponding manuals. Chapters 4-10 describe the operating system and many of the commands that are provided with it.

Operating the software involves, first, loading and running the operating system and, then, loading and running programs under the control of the operating system.

The sections "Initialization" and "Configuration" in this chapter describe how to initially load and run the operating system.

Once ISIS is initially loaded, software operation consists of entering command lines that cause programs to be loaded and run. Command lines are accepted by a part of the operating system called the Command Line Interpreter (CLI). The CLI loads and runs a program as specified in the command line. Later sections in this chapter describe how to enter command lines to run programs.

Initialization

The operating system is contained in several files on the system disk or the system bubble multimodule and must be loaded into the development system's memory in a process called initialization or bootstrapping.

The system disk is the disk supplied by Intel containing the operating system files needed to initialize the system. See the IDISK command in Chapter 5 for a more specific description of the files required on a system disk. The disk supplied with the system should be duplicated on another disk or on a bubble memory multimodule. The new disk or bubble also becomes a system disk or system bubble. At least one extra copy of the system disk should be maintained, so that the system can still be initialized even if one of the system disks or bubbles is destroyed. An example of the procedure for duplicating a system disk from the disk supplied by Intel is given at the end of this chapter.

Initialization occurs when the system is powered on or when the RESET key on the keyboard is pressed. Either action causes a program contained in the 2K bytes of PROM to be executed. The PROM program performs a diagnostic test and loads a bootstrap from the disk. The bootstrap program disables the 2K bytes of PROM, enables 32K bytes of RAM, and loads the ISIS-PDS operating system.

To initialize the system from a system disk in the internal disk drive (drive 0), power on the system, insert the system disk, and then press the RESET key. These steps are described in detail in the following. First, the steps are given for a system with no bubble memory installed.

1. Power on the development system. (The power switch is located on the upper right side of the rear panel.) As soon as the system is powered on, the diagnostic/loader program begins executing. This program attempts to initialize the system from drive 0 (the internal disk drive). However, it is not recommended that the system be powered on with a disk in the drive.

2. Power on any peripheral devices, such as printers or external disk drives. Since the system disk is not yet inserted in the drive, the following message is displayed:

   NO BOOT DEVICE

3. Place a system disk in drive 0. Insert the disk as shown in figure 3-10. Drive 0 is initially used as the system drive.

4. Press the RESET key causing the diagnostic/loader program to search for a disk again, this time initializing the system from drive 0.

5. The message

   ISIS-PDS, Vn.m

   is displayed on the screen where n.m is replaced by the actual version number for the system.

6. The operating system displays the prompt characters

   A0>

   on the screen indicating that commands can be entered.

7. If there is a file named ABOOT.CSD on the system drive, the base processor automatically executes operating system commands from that file. This file is used to initially configure a system and is described in more detail in Chapter 4.

To initialize from a system containing bubble memory (drive 4):

1. Make sure that no disk is in drive 0 (the internal disk drive).

2. Make sure that a bubble multimodule that contains the operating system is installed as drive 4. See Appendix A for installation instructions. See the IDISK command in Chapters 4 and 5 for instructions on initializing the bubble memory multimodule as a system disk.

3. Power on the development system. The program will first attempt to initialize the system from drive 0.

   However, since there is no disk in drive 0, the attempt will fail. Since a bubble memory multimodule is installed, the following message is displayed:

   BOOT FROM BUBBLE? (Y or N)

   This message is displayed when the bubble is present and no disk is in drive 0, or when the disk drive door is open.

4. Type Y in response to the message to complete the initialization from the bubble multimodule in drive 4. (Insert a system disk in drive 0 and type N to boot from disk when bubble memory is installed. Steps 5-7 from the previous procedure will then occur.)

5. The message:

   ISIS-PDS, Vn.m

   is displayed on the screen where n.m is replaced by the actual version number for the system.

6. The operating system displays the prompt characters

   A4>

   on the screen indicating that commands can be entered.

7. If there is a file named ABOOT.CSD on the system drive, the base processor automatically executes operating system commands from that file. See Chapter 4 for information on creating ABOOT.CSD.

8. Power on any peripheral devices, such as printers or external disk drives.

Initialization procedures for dual processor systems are covered in detail in Chapter 9. The initialization program flowchart is shown in figure 3-13.

As soon as the system is initialized for the first time, the system disk supplied by Intel should be duplicated. At least two copies of the system disk should be maintained in case one diskette is destroyed. The example at the end of this chapter gives the procedure for duplicating the system disk.

Error Conditions

During the diagnostic phase of the initialization program, errors are indicated either by four diagnostic LED indicators or by a message on the display screen. The LED indicators are on the iPDS processor board and can be checked by holding open the plug-in module door on the side of the system and looking through the opening.

During the initialization phase when the operating system is loaded, errors are indicated by a message on the display screen.

Error conditions and messages are described in detail in Appendix B with instructions on interpreting the LED indicators.

User Configurations

For some user applications, it is necessary to further initialize the system. For example, the 8251 USART serial I/O device may need to be initialized automatically whenever the system is initialized.

ISIS-PDS allows the user to automatically run a program or a series of programs as soon as the system is initialized with no operator interaction. Configuration is accomplished by creating a special JOB file called ABOOT.CSD as described in Chapter 4 and Chapter 5 in the sections on the JOB command. This file contains the commands necessary to configure the user's environment when the system is initialized. The file must end with the ENDJOB command.

Commands

A command causes a program to be loaded and run under the control of the operating system. To be more specific, most commands correspond to an object program stored as a file on a disk or bubble memory device. To issue a command to the operating system, enter the correct command line. The file containing the program is then loaded into memory and run by the operating system. When the program has finished running, it returns control to the operating system so that another command can be issued. Some commands are embedded in the resident portion of ISIS-PDS and are always present in memory. These commands are also run from command lines but no disk file need be loaded.

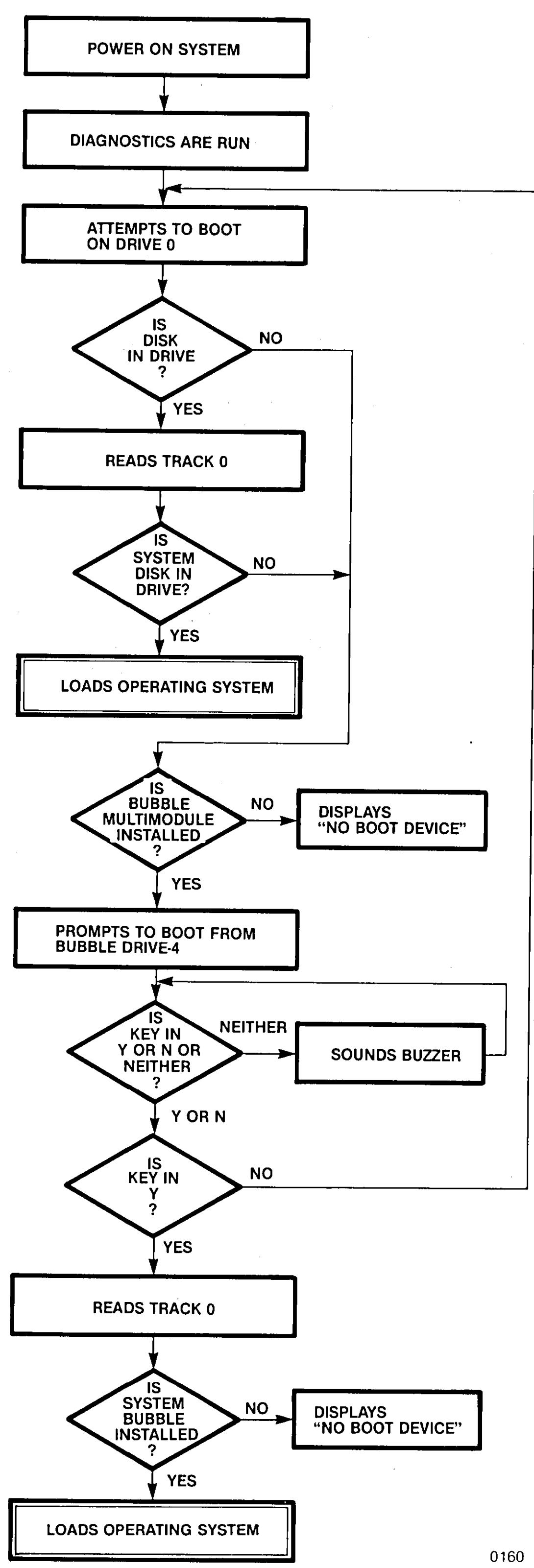


Figure 3-13 Flowchart of Initialization Program

User written programs can also be run as commands. See Chapter 8 for further information.

There are two different types of commands provided with the operating system: interactive and non-interactive commands.

An interactive command performs many different functions through a set of subcommands. Subcommands are entered from the keyboard after the initial command is issued and the corresponding program is loaded. These subcommands are processed by the program loaded, not by the ISIS-PDS command line interpreter.

Some interactive commands provided with ISIS-PDS are:

- CREDIT, which provides screen oriented text editing for source programs and other documents
- DEBUG, which provides a minimum set of debugging commands
- LIB, which allows the user to manage a library of MCS-80/85 program modules

Some of these commands are described in Chapters 4-10; others are in separate manuals. See Chapter 1 for further references.

Non-interactive commands perform a single function through an ISIS-PDS command line. Some non-interactive commands are:

- COPY, which duplicates a file from one device to another
- IDISK, which initializes a disk or a bubble memory multimodule
- DIR, which displays the files currently stored on the specified device
- HELP, which displays information about the system and other ISIS-PDS commands

Most of these commands are described in Chapters 4 and 5; others are in separate manuals. See Chapter 1 for further references.

Command Lines

To issue a command, the user must enter the correct command line. A command line consists of two parts: a command name which corresponds to the filename containing the program and the command parameters that are needed by individual commands. The entire command line is terminated by the RETURN key. The general format for a command line is illustrated below using the notational conventions described in detail in Chapter 5. These conventions are used throughout the manual to describe the format of commands.

```
{ <command name> <parameters> }  RETURN
{          ;<comment>         }
```

The angle brackets (< >) enclose general terms that must be replaced by a specific member of the class specified. For example, <command name> is replaced by a specific command name like COPY, DELETE, or RENAME. The braces ({ }) enclose a vertical list of items and imply a choice of one and only one of the items listed.

The command name is the same as the name of the file containing the program to be run. The complete format of the command name is:

:<logical device name>:<filename>.<extension>

In most cases, the complete format need not be specified to run the program. Instead, enter only the filename allowing the system to default the device name and the extension as described below. The logical device name is a two-character identifier enclosed by colons (:). The filename is one- to six-characters, and the extension is one- to three- characters preceded by a period (.).

Parameters are entered as a sequence of characters on the command line; however, the number of parameters and the form in which they are entered vary from command to command. In general, parameters specify the data used by the command.

Some commands have no parameters. For other commands, a parameter could be an input file identifier or an output device identifier. For example, an input file identifier is a parameter for the COPY command.

Sometimes a command can perform one or more operations, and the parameter identifies which operation to perform. For example, the IDISK command can initialize a system disk or a non-system disk depending on the parameters entered on the IDISK command line.

In addition to entering a command name with its associated parameters, a comment line may be entered. Comment lines are used primarily in SUBMIT files. See the SUBMIT command in Chapters 4 and 5 for details. Comments are preceded by a semicolon (;) and must be followed by the RETURN key.

Command Line Defaults

The default system drive (known by ISIS-PDS as :F0:) is initially assigned to drive 0 or drive 4 depending on whether the system was initialized from the internal disk drive or bubble memory. The system drive can be changed by the ASSIGN command . The current system drive is displayed as the second character in the ISIS-PDS prompt (see the following section describing the ISIS-PDS prompt).

By storing command files on the default device, the device name need not be entered as part of the command name. Command files usually do not have an extension, meaning the filename alone is usually enough to specify the command.

Default values for parameters vary from command to command and are discussed in sections describing a particular command.

Entering Command Lines

Command lines can be entered either through the keyboard or from a file. They are echoed on the display screen as they are entered. Non-displayable characters are echoed as a tilde (e) unless they are preceded by a CTRL-P which enters them literally into the buffer.

Entering Command Lines from the Keyboard

Whenever the ISIS-PDS prompt characters are displayed, a command line can be entered from the keyboard. For example, the DIR command can be run by typing the following on the keyboard:

DIR

The prompt characters are of the form:

Pd>

where P indicates the current processor and can be either A for the base processor or B for the optional processor. The d is the number of the physical drive currently assigned to :F0:, the system default disk device. It can be physical drive 0, 1, 2, 3, 4, or 5 depending on the last ASSIGN command.

The command line can not be greater than 122 characters. The display screen echoes these characters as they are entered. After 77 characters have filled one display line, the cursor automatically wraps around to the line below and the remaining 45 characters can be typed.

Editing Command Lines. The command line is actually stored in a buffer, a holding area in memory, called the line editing buffer as it is typed on the keyboard. The entire command line is presented to the operating system only after the RETURN is typed. The RETURN terminates the line edited input.

A command line can be corrected in two ways: by entering control characters that are only recognized prior to terminating the line edited input with the RETURN key or by entering the editing mode and then using control characters that are recognized during edit mode.

The following characters can only be used to edit a command line prior to terminating the line edited input with the RETURN key. Most of these characters are control characters, characters typed while the CTRL key is held down.

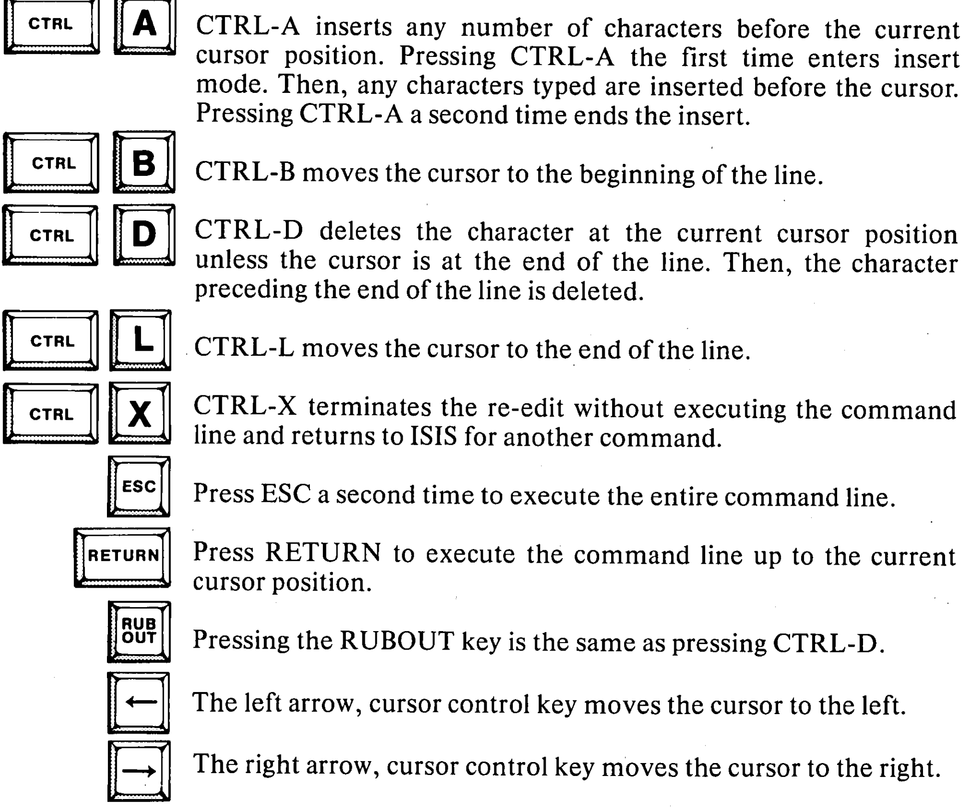
CTRL-R echoes a carriage return/linefeed on the display line followed by the current contents of the line editing buffer. CTRL-R does not cancel or execute the contents of the buffer, nor does it enter the command line editing mode described below. It is useful if a teletype terminal is connected to the system for command line input.

CTRL-X erases the entire buffer, but it must be typed before the RETURN is entered. CTRL-X is echoed on the display line as a "#" followed by a carriage return/linefeed, so the cursor is positioned as the first character of the next line. CTRL-X does not terminate the line editing buffer, so no command line is presented to the operating system. The operating system prompt does not appear in the next line.

CTRL-Z deletes the entire line editing buffer and terminates the buffer, so the operating system prompt appears at the beginning of the next line.

RUBOUT erases the most recently entered key.

The following characters can be used to enter command line editing mode to correct a command line either before or after the RETURN key has been pressed. The ESC key can be pressed to enter editing mode and correct the most recent command line. The most recent command line is displayed as it is stored in the line editing buffer, and the prompt character > changes to + to indicate that editing mode is in effect. The following keys can be used to modify and re-execute the command line.



CTRL A — CTRL-A inserts any number of characters before the current cursor position. Pressing CTRL-A the first time enters insert mode. Then, any characters typed are inserted before the cursor. Pressing CTRL-A a second time ends the insert.

CTRL B — CTRL-B moves the cursor to the beginning of the line.

CTRL D — CTRL-D deletes the character at the current cursor position unless the cursor is at the end of the line. Then, the character preceding the end of the line is deleted.

CTRL L — CTRL-L moves the cursor to the end of the line.

CTRL X — CTRL-X terminates the re-edit without executing the command line and returns to ISIS for another command.

ESC — Press ESC a second time to execute the entire command line.

RETURN — Press RETURN to execute the command line up to the current cursor position.

RUB OUT — Pressing the RUBOUT key is the same as pressing CTRL-D.

← — The left arrow, cursor control key moves the cursor to the left.

→ — The right arrow, cursor control key moves the cursor to the right.

Only command lines of six or more characters (including spaces) are saved for re-editing.

Pausing the Display. Two control characters, CTRL-S and CTRL-Q, allow the operator to control the scrolling of the display screen. CTRL-S stops the scrolling of the output on the display screen and also stops the program generating the display. The display remains stopped until a CTRL-Q is entered from the keyboard. Any characters typed between the CTRL-S and the CTRL-Q are ignored.

FUNCT-S switches the display speed between slow and fast scrolling.

Entering Command Lines from a File

Another way to enter command lines is from a command file. ISIS-PDS allows two types of command files: SUBMIT files and JOB files. The SUBMIT file is a text file that can be created with a text editor, such as the CREDIT text editor described in Chapter 6. It contains command lines that appear in the file just as they would appear if they were typed on the keyboard.

To execute the commands in the SUBMIT file, the SUBMIT program is run as an operating system command. It submits command lines from the file to the operating system as if they had been typed on the keyboard.

The SUBMIT command is described in detail in Chapters 4 and 5.

The JOB file is created with the JOB command and can be executed in several ways as described in Chapters 4 and 5. The JOB command allows the user to type a sequence of command lines at the keyboard which can then be executed in the sequence in which they were typed. The JOB file must end with an ENDJOB command.

Other Ways to Enter Command Lines

There are other ways to enter command lines which are variations and combinations of entering from the keyboard and of entering from a file.

Most of these additional methods involve an operating system command and are discussed in Chapter 4 and Chapter 5.

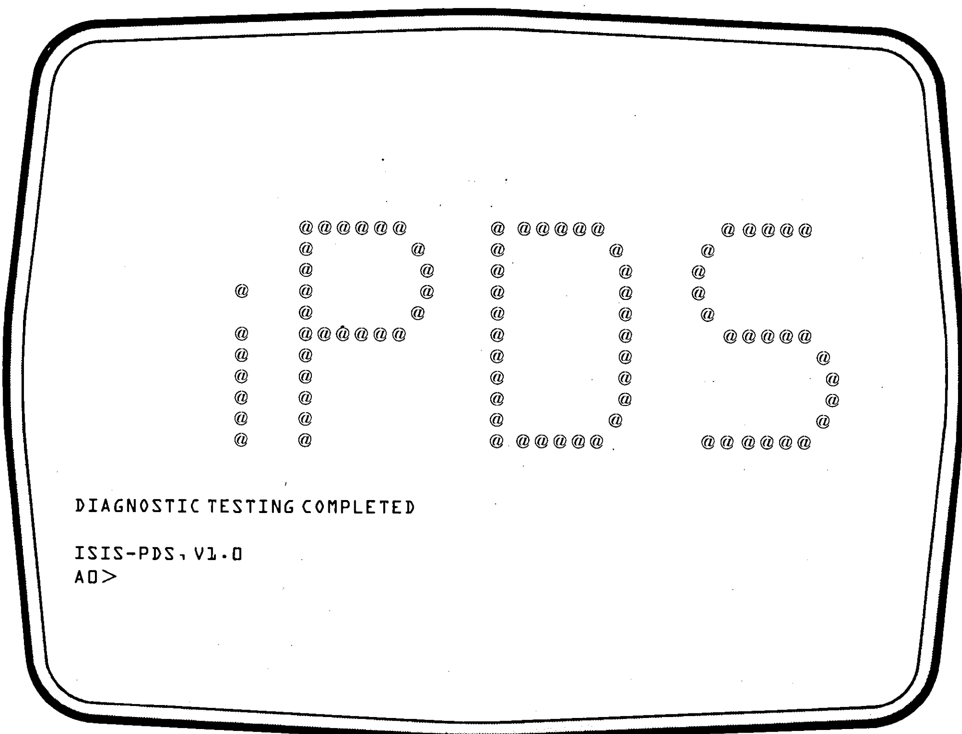
Sample Initialization Session

The rest of this chapter contains a series of examples using screen displays, comments, and key-in sequences. These examples illustrate the concepts introduced in this chapter.

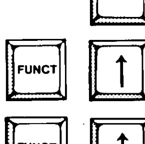
The examples are complete, so that the key-in sequences can be entered to produce the results shown in the screen displays. However, in some cases, operator action besides keying in a command line is required. Thus, the comments should be read prior to entering the key-in sequence. Also, many of the examples depend on the output from the previous examples. In most cases, the examples should be followed in order to guarantee the same results. Finally, in some of the examples, the screen display shown may not be exactly the same as the display generated by the user. In these cases, the exact display depends on the version of the operating system and the order in which the examples were run.

Initializing the System from Disk

This example shows how to initialize a system from the disk in drive 0.



Key-in Sequence

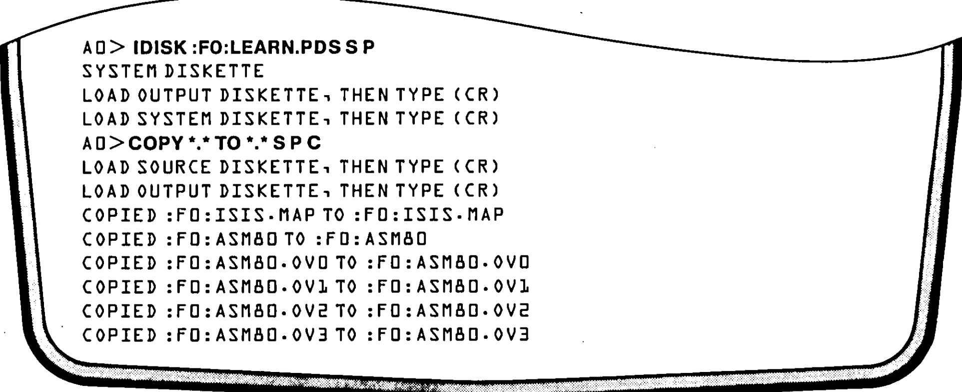


Comments

This example assumes that the bubble memory is not installed. If the bubble memory multimodule is installed, see the bubble memory example at the end of this chapter. Power on the system with no disk in the drive. When the red LED indicator on the drive goes off, insert the system diskette and press the reset key. Immediately after pressing the reset key, the character 'A' appears on the top line of the CRT screen. After a few seconds, the initialization is complete and the screen appears as shown. When the dual processor is installed, the top 2 lines are shown as reverse video with the letter 'B' on the top line. Press the up arrow key two times while holding down the FUNCT key to get rid of the display from the dual processor.

Duplicating the System Disk on Single Drive Systems

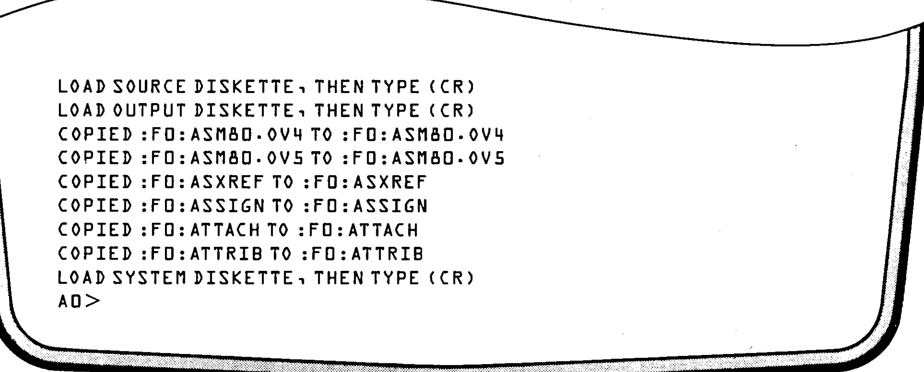
In this series of examples, a back-up copy of the system disk is made.



Key-in Sequence	Comments
IDISK :F0:LEARN.PDS S P	Make a duplicate copy of system diskette. First, initialize a new diskette with the IDISK command. This example shows how to run the IDISK command on a system with a single disk drive. If a mistake is made in typing any of these commands, use the RUBOUT key to backspace and type the correct characters. When more than one drive is available, go to the section entitled 'Duplicating the System Disk on Multiple Drive Systems.'
	Remove the system diskette. Insert a new diskette without a write protect tab and press the RETURN key.
	Remove the newly created diskette and insert the system diskette. Press the RETURN key. The operating system prompt is then displayed.
COPY *.* TO *.* S P C	The second step in duplicating the system diskette is copying the files from the master to the newly created diskette with the COPY command.
	The source diskette in this case is the system diskette. Press the RETURN key to begin reading files to be copied. Files are read until the temporary storage area in memory is full.
	Press the RETURN key after removing the source diskette and inserting the newly created diskette. Files read from the source diskette and stored in memory are written to the output diskette.

(continued)

Key-in Sequence	Comments
	A message is displayed for each file copied. The exact sequence of messages depends on the software package selected by the user and the files on the system disk. When all the files are copied, re-insert the system diskette. Press the RETURN key and more files are read from the system diskette.



Key-in Sequence	Comments
	Remove the system diskette and insert the diskette being created. Press the RETURN key to continue copying.
	The sequence of switching the source diskette and the output diskette is repeated several more times to complete the copying of the system files. These steps are not shown in detail.

Duplicating the System Disk on Multiple Drive Systems

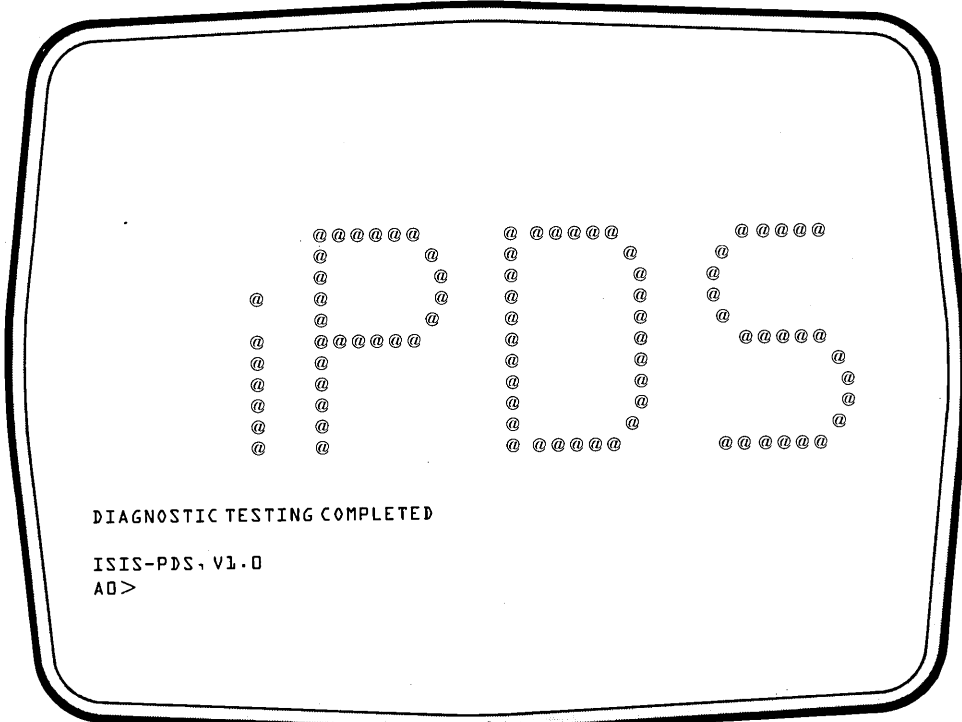
In this example, a back-up of the system disk is made on a multiple drive system.

```
A0> IDISK :F1:LEARN.PDS S
SYSTEM DISKETTE
A0>COPY *.* TO :F1:*.*
COPIED :F0:ATTACH TO :F1:ATTACH
COPIED :F0:ATTRIB TO :F1:ATTRIB
COPIED :F0:COPY TO :F1:COPY
COPIED :F0:CREDIT TO :F1:CREDIT
COPIED :F0:CREDIT.MAC TO :F1:CREDIT.MAC
COPIED :F0:DEBUG TO :F1:DEBUG
COPIED :F0:DELETE TO :F1:DELETE
COPIED :F0:DETACH TO :F1:DETACH
COPIED :F0:DIR TO :F1:DIR
COPIED :F0:HELP TO :F1:HELP
COPIED :F0:HEXOBJ TO :F1:HEXOBJ
COPIED :F0:IDISK TO :F1:IDISK
COPIED :F0:IXREF TO :F1:IXREF
COPIED :F0:LIB TO :F1:LIB
COPIED :F0:LINK TO :F1:LINK
COPIED :F0:LINK.OVL TO :F1:LINK.OVL
COPIED :F0:LOCATE TO :F1:LOCATE
COPIED :F0:OBJHEX TO :F1:OBJHEX
COPIED :F0:PDS.HLP TO :F1:PDS.HLP
COPIED :F0:RENAME TO :F1:RENAME
COPIED :F0:SERIAL TO :F1:SERIAL
A0>
```

Key-in Sequence	Comments
IDISK :F0:LEARN.PDS S RETURN	Place a new diskette without a write protect tab in drive 1 and enter the command as shown. When the initialization is complete, the operating system prompt is displayed.
COPY *.* TO :F1:*.* RETURN	After the IDISK command is complete, the files are copied to the new disk to complete the duplication. A message is displayed for each file copied, and when all the files are copied, the operating system prompt is returned. The message for the first files coped will scroll off the top of the screen.

Entering Command Lines

The next series of examples illustrate how to enter a command line.



Key-in Sequence	Comments
	Remove the system disk and insert the newly created diskette in drive 0. Press the RESET key to re-initialize the system. When the dual processor is installed, press the uparrow key two times while holding down the FUNCT key to get rid of the reverse video display at the top of the screen.

```
A0> DIR
DIRECTORY OF :F0:LEARN.PDS
NAME     .EXT  BLKS  LENGTH   ATTR  NAME     .EXT  BLKS  LENGTH  ATTR
ISIS     .MAP     4     512  S      ASM80            60   14594  S
ASXREF           20    4294  S      ASSIGN           16    3073  S
ATTACH            4     522  S      ATTRIB           24    4999  S
COPY             36    8366  S      CREDIT           80   19740  S
CREDIT   .MAC     4       7  S      DEBUG            12    2502  S
DELETE           20    4699  S      DETACH            4     434  S
DIR              28    6625  S      HELP             16    3771  S
HEXOBJ           20    4344  S      IDISK            32    7035  S
IXREF            44   10216  S      LIB              44   10227  S
LINK             56   13074  S      LINK     .OVL    20    4578  S
LOCATE           60   15021  S      OBJHEX           16    3347  S
RENAME           12    2557  S      SERIAL           16    3148  S
SUBMIT           20    4692  S      SYSPDS   .PDS    16    3101  S
                                684
1456 FREE / 2544 TOTAL BLOCKS
A0>HELP COPY
```

Key-in Sequence	Comments
DIR	The DIR command displays a list of the files on the disk. In this case, the files on the disk in drive 0 are displayed. The display will be similar to display shown above. The exact display depends on the files on the system disk. Notice that IDISK, COPY, and DIR, are also files on the disk. The file named HELP in the second column contains the HELP command which displays information about other operating system commands.
HELP COPY	Typing HELP followed by the RETURN key displays general help about the operating system. Typing HELP followed by a command name displays information about that command. In this example, HELP is displayed for the COPY command.

```
COPY      Transferring files
     COPY <srce> TO <dest> [{S | N}] [{B | U}] [J] [K] [L] [C] [P] [Q]
          <srce>     Pathname of input file, the file being copied.
          <dest>     Pathname of output file.
          S          Copy only system files (with S attribute).
          N          Copy only non-system files (without S or F
                     attribute).
          B          No prompt if destination exists. Delete existing
                     file; copy source to newly created destination.
          U          Same as B except existing file is not deleted
                     first.
          J          Copy only files with User Defined attribute J.
          K          Copy only files with User Defined attribute K.
          L          Copy only files with User Defined attribute L.
          C          Copy the source file's attributes.
          P          Single drive COPY.
          Q          Prompt before processing.

COPY      Appending files
     COPY <srce 1>,<srce 2>[,. . .,<srce n>] TO <dest> [B | U] [C] [P]
          <srce 1>
          thru
          <scre n>  Specifies the input files.
          <dest>    Specifies the output file.

A0>
```

Comments

Information about the COPY command is displayed as a result of entering the previous HELP command. The square brackets ([ ]) indicate options. The simplest way to transfer files without any options is: COPY <srce> TO <dest>.

```
A0> COPY DIR TO CAT
COPIED :F0:DIR TO :F0:CAT
A0>
```

Key-in Sequence

COPY DIR TO CAT

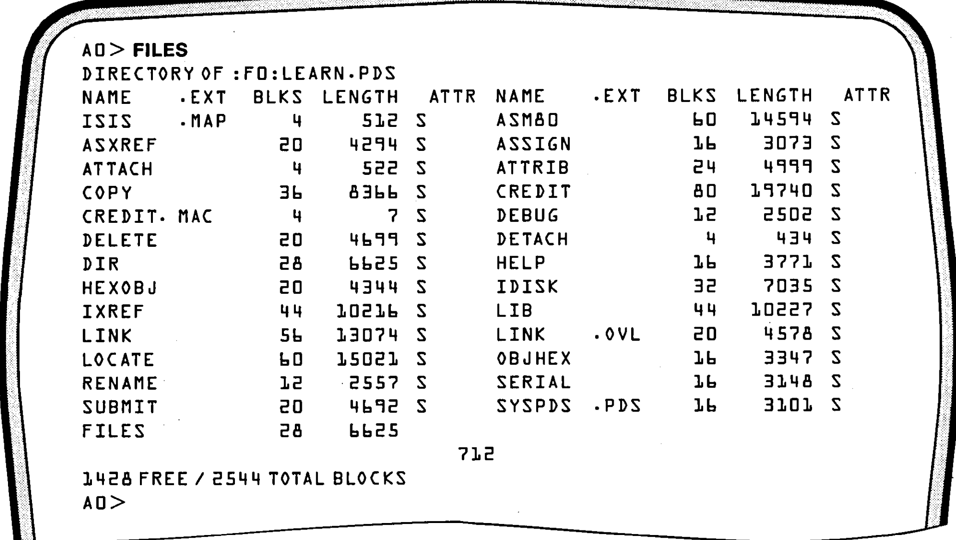
Comments

This command makes a duplicate of the file, DIR, under the name CAT for catalog. Thus, the file, CAT, also contains the directory command. The operating system prompt is always returned after the command is finished.

```
A0> CAT
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT  BLKS  LENGTH  ATTR  NAME    .EXT  BLKS  LENGTH  ATTR
ISIS    .MAP     4     512  S     ASM80           60   14594  S
ASXREF          20    4294  S     ASSIGN          16    3073  S
ATTACH           4     522  S     ATTRIB          24    4999  S
COPY            36    8366  S     CREDIT          80   19740  S
CREDIT  .MAC     4       7  S     DEBUG           12    2502  S
DELETE          20    4699  S     DETACH           4     434  S
DIR             28    6625  S     HELP            16    3771  S
HEXOBJ          20    4344  S     IDISK           32    7035  S
IXREF           44   10216  S     LIB             44   10227  S
LINK            56   13074  S     LINK    .OVL    20    4578  S
LOCATE          60   15021  S     OBJHEX          16    3347  S
RENAME          12    2557  S     SERIAL          16    3148  S
SUBMIT          20    4692  S     SYSPDS  .PDS    16    3101  S
CAT             28    6625
                                712
1428 FREE / 2544 TOTAL BLOCKS
A0>
```

Key-in Sequence	Comments
CAT	This example illustrates that the file, CAT, contains the directory command. Typing CAT causes the file CAT to be loaded and executed as a command. The same function is performed as in the previous DIR command example. The file CAT appears in the directory here and not in the previous directory. This file was added to the directory when it was created by the COPY command.

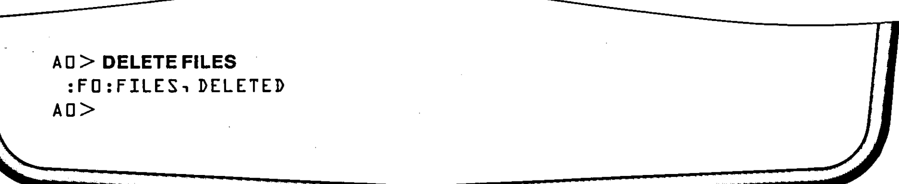
Key-in Sequence	Comments
RENAME CAT TO FILES	The RENAME command only changes the name of the file specified. It does not make a copy of that file. In this case, the file CAT is renamed to FILES. Thus, the file, FILES, is now a copy of the directory command. Many of the operating system command files can be renamed at user convenience.



Key-in Sequence | **Comments**

FILES

Typing FILES is now equivalent to entering the DIR command. Notice that the file CAT no longer appears in the resulting list of files. The file, FILES, now appears in its place.



Key-in Sequence | **Comments**

DELETE FILES

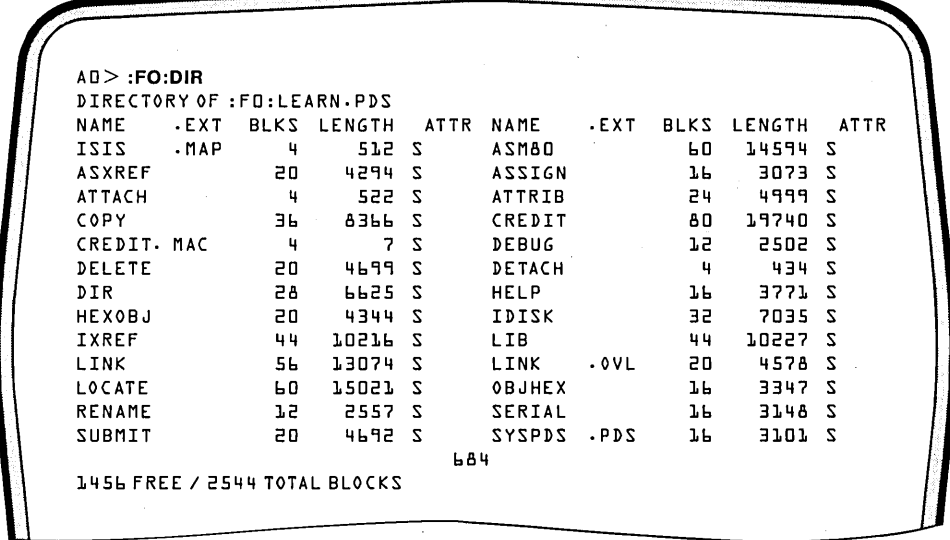
The DELETE command removes the specified file from the disk and from the directory. Now, the file, FILES, no longer appears in the directory listing.

```
A0> DIR
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT  BLKS  LENGTH  ATTR  NAME    .EXT  BLKS  LENGTH  ATTR
ISIS    .MAP     4     512  S     ASM80           60   14594  S
ASXREF          20    4294  S     ASSIGN          16    3073  S
ATTACH           4     522  S     ATTRIB          24    4999  S
COPY            36    8366  S     CREDIT          80   19740  S
CREDIT. MAC      4       7  S     DEBUG           12    2502  S
DELETE          20    4699  S     DETACH           4     434  S
DIR             28    6625  S     HELP            16    3771  S
HEXOBJ          20    4344  S     IDISK           32    7035  S
IXREF           44   10216  S     LIB             44   10227  S
LINK            56   13074  S     LINK    .OVL    20    4578  S
LOCATE          60   15021  S     OBJHEX          16    3347  S
RENAME          12    2557  S     SERIAL          16    3148  S
SUBMIT          20    4692  S     SYSPDS  .PDS    16    3101  S
                                684
1456 FREE / 2544 TOTAL BLOCKS
```

Key-in Sequence / **Comments**

DIR RETURN

Enter the DIR command to verify that the file, FILES, no longer appears in the directory.



```
A0> :F0:DIR
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT  BLKS  LENGTH  ATTR  NAME    .EXT  BLKS  LENGTH  ATTR
ISIS    .MAP     4     512  S     ASM80           60   14594  S
ASXREF          20    4294  S     ASSIGN          16    3073  S
ATTACH           4     522  S     ATTRIB          24    4999  S
COPY            36    8366  S     CREDIT          80   19740  S
CREDIT. MAC      4       7  S     DEBUG           12    2502  S
DELETE          20    4699  S     DETACH           4     434  S
DIR             28    6625  S     HELP            16    3771  S
HEXOBJ          20    4344  S     IDISK           32    7035  S
IXREF           44   10216  S     LIB             44   10227  S
LINK            56   13074  S     LINK    .OVL    20    4578  S
LOCATE          60   15021  S     OBJHEX          16    3347  S
RENAME          12    2557  S     SERIAL          16    3148  S
SUBMIT          20    4692  S     SYSPDS  .PDS    16    3101  S
                                684
1456 FREE / 2544 TOTAL BLOCKS
```

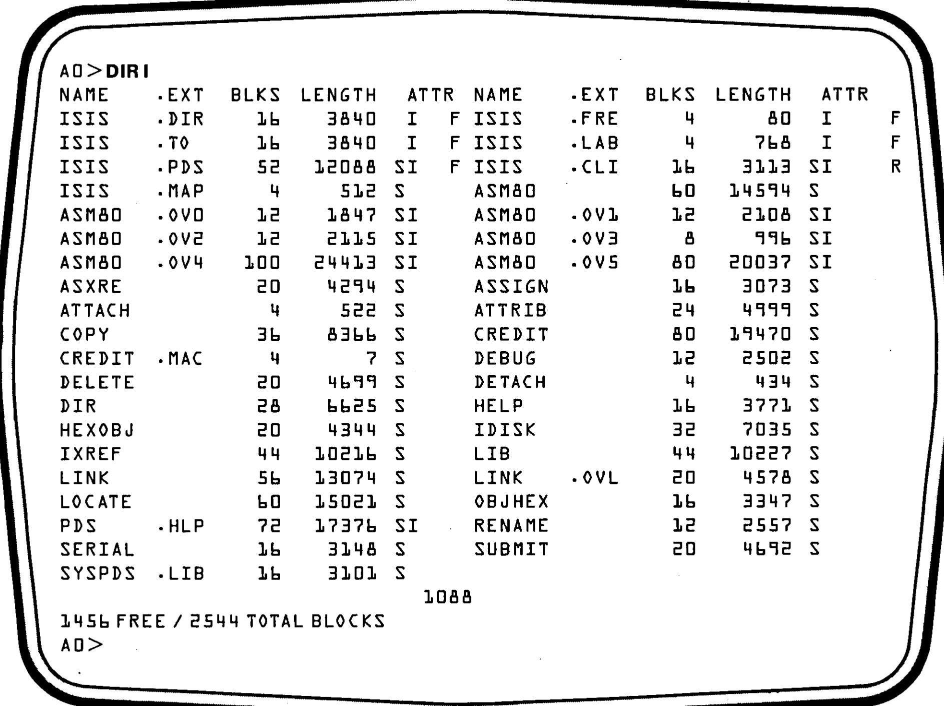
Key-in Sequence / **Comments**

:F0:DIR

The commands entered under the ISIS operating system are actually programs. Many of these command programs are stored in files on the disk and are loaded into memory and executed when needed. Thus, the pathname of the file is entered first on the command line. In this example, the drive number portion of the pathname is entered resulting in the command being executed. If the drive number is left off, logical drive 0 is assumed.

Using Control Characters

The next series of examples illustrate how to use control characters.

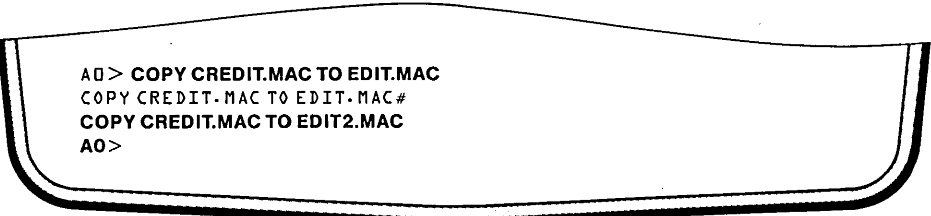


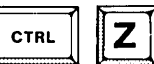
Key-in Sequence | **Comments**

CTRL S

The DIR command only displays a list of the visible files on the disk. A file may be assigned an attribute of I for invisible. It then does not show up on a normal directory listing. To include invisible files in the listing, use the DIR command with the I option. This example also illustrates the use of CTRL-S to stop the output on the CRT display. Depending on when the CTRL-S is typed, the display is similar to the screen shown.

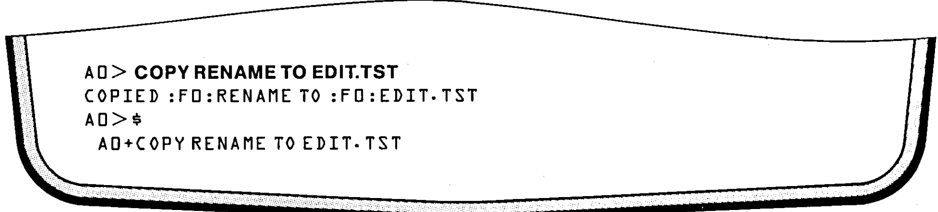
Use CTRL-Q to restart the display. The rest of the directory is then displayed.

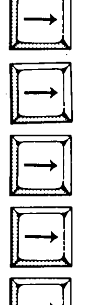


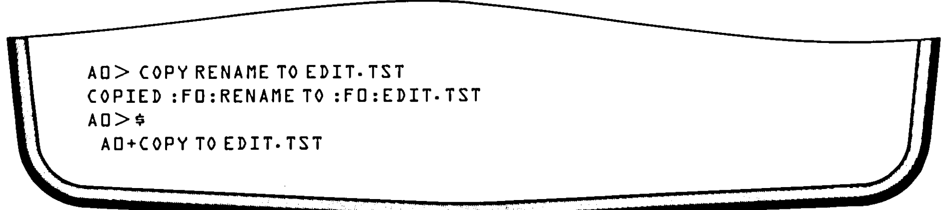
Key-in Sequence	Comments
COPY CREDIT.MAC TO EDIT.MAC	The control character CTRL-R typed before the RETURN key is pressed causes the command line currently being typed to be re-displayed. The CTRL-R function is useful in applications where a TTY (teletype) terminal is connected to the system and used to input commands. TTY terminals print the display on paper. They do not have a rubout function. However, the RUBOUT (or backspace) key still corrects typing errors as the command line is being entered. The line printed on the TTY terminal shows both the error and the correction. After a few corrections on the same command line, the line may not be readable. CTRL-R can then be used to re-display the command line after the corrections so that it is readable.
	Type CTRL-X to delete the command buffer but not close the buffer. The same command or a different command can then be entered.
COPY CREDIT.MAC TO EDIT2.MAC 	Type the COPY command into the buffer emptied by the previous CTRL-X. Then, type CTRL-Z. This deletes the buffer and closes it; returning to the operating system prompt.

Editing Command Lines

The next series of examples illustrate how to edit a command line.

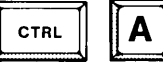


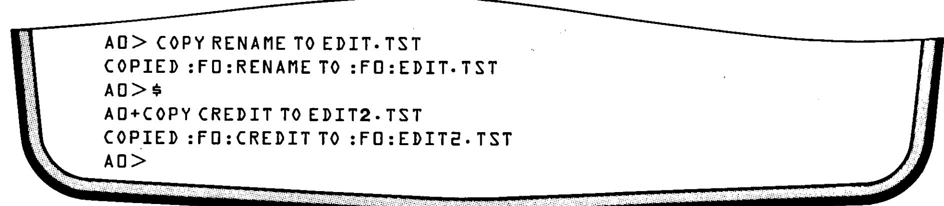
Key-in Sequence	Comments
COPY RENAME TO EDIT.TST	This example is used to illustrate the command line editing features of the operating system. This command copies the file RENAME to the file EDIT.TST.
	After the command is executed,(or while it is still being entered before the RETURN key is pressed), the ESC key can be pressed to enter a mode where the command line can be edited. The command line previously entered (or the one currently being entered) is re-displayed with the cursor at the end of the line. Several characters can then be used to edit the line.
	Type the control character CTRL-B to move the cursor to the beginning of the line.
	Press the right arrow key five times to move the cursor the R of RENAME.



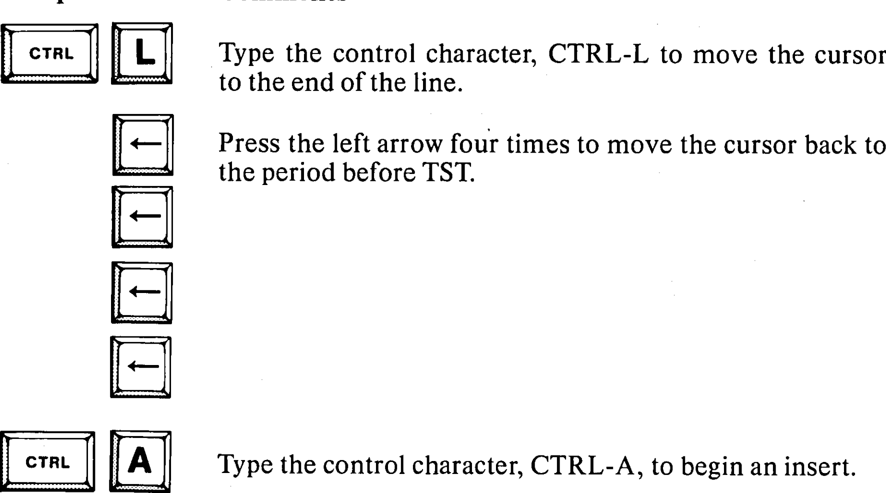
Key-in Sequence	Comments
CTRL D	Type the control character CTRL-D to delete the character R.
CTRL D	Type CTRL-D again to delete the E of RENAME.
	Type CTRL-D four more times to delete the entire word RENAME.
CTRL D	
CTRL D	

```
A0> COPY RENAME TO EDIT.TST
COPIED :F0:RENAME TO :F0:EDIT.TST
A0>$
A0+COPY CREDIT TO EDIT.TST
```

Key-in Sequence	Comments
	Type the control character CTRL-A to begin an insert. The screen opens up to allow any number of characters to be entered.
CREDIT	Type the word CREDIT to replace RENAME.
	Type CTRL-A again to end the insert. The screen closes back up again.

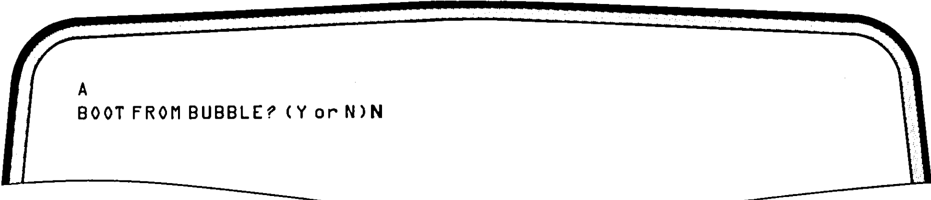


Key-in Sequence	Comments
CTRL L	Type the control character, CTRL-L to move the cursor to the end of the line.
← ← ← ←	Press the left arrow four times to move the cursor back to the period before TST.
CTRL A	Type the control character, CTRL-A, to begin an insert.
2	Type the 2 key to insert the digit 2 in the filename.
CTRL A	Type CTRL-A to end the insert.
CTRL L	Type CTRL-L to move the cursor to the end of the line.
RETURN	Press the RETURN key to execute the new edited command. In addition to the features illustrated in this example, characters may be replaced by moving the cursor to the desired character and typing over it with a single replacement character. When the cursor is at the end of the line, the line is extended by typing characters.

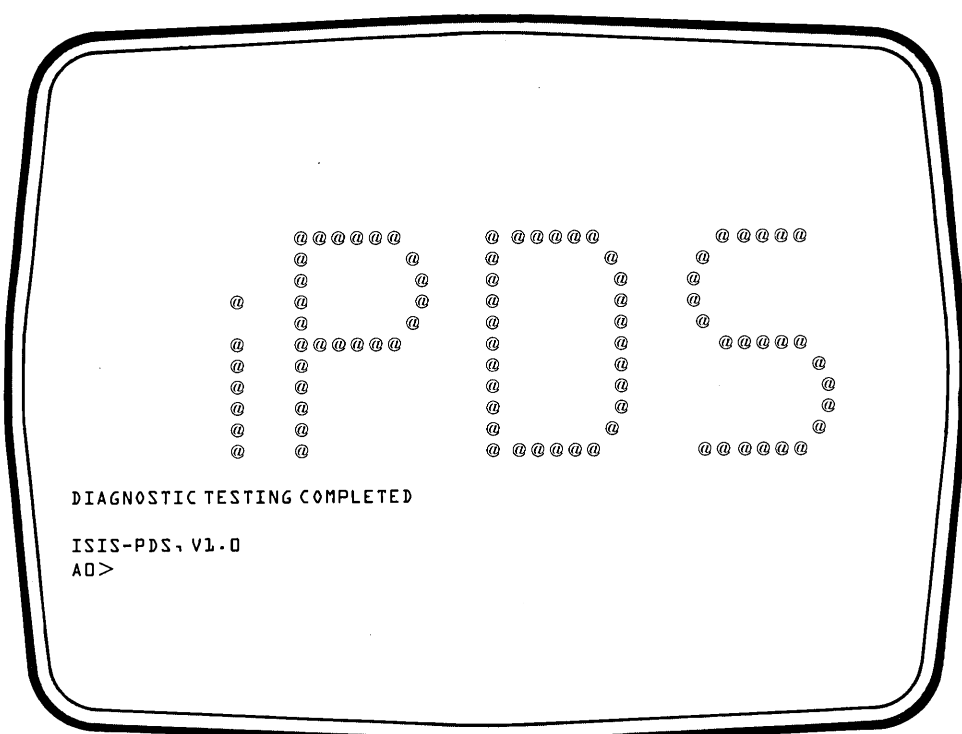


Initializing the System from Bubble Memory

The next series of examples illustrates how to initialize the system from bubble memory.



Key-in Sequence	Comments
N	With the bubble memory multimodule installed in J1/J2 power on the system. When the red LED indicator on the internal disk drive goes off, insert the system disk. Then, type the N key to initialize the system.



Comments

The system is initialized from the disk in drive 0. The bubble memory is like a blank diskette and must be formatted before it can be used.

```
A0>IDISK :F4:BUBBLE.SYS S
SYSTEM DISKETTE
A0>COPY *.* TO :F4:*.*
```

Key-in Sequence / **Comments**

IDISK :F4:BUBBLE.SYS S

Enter the command as shown above to format the the bubble memory.

When the initialization is complete, the operating system prompt is displayed.

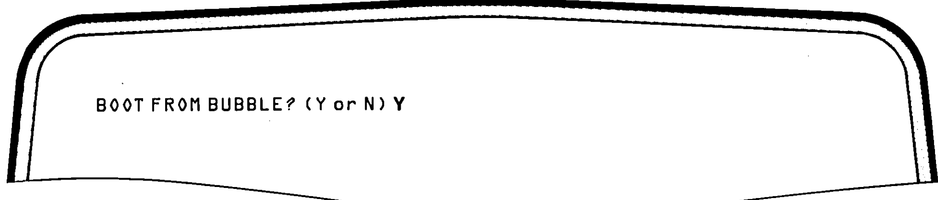
COPY *.* TO :F4:*.*

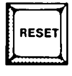
After the IDISK command is done, the files must be copied to the new disk to complete the duplication. Key-in the command as shown.

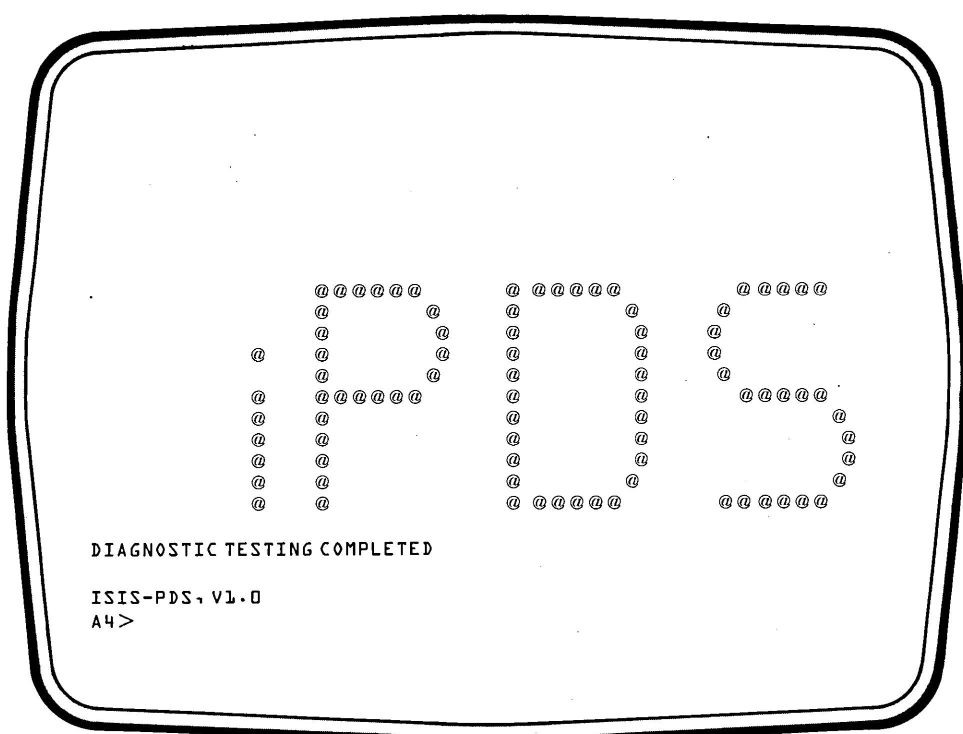
```
COPIED :F0:ATTACH TO :F4:ATTACH
COPIED :F0:ATTRIB TO :F4:ATTRIB
COPIED :F0:COPY TO :F4:COPY
COPIED :F0:CREDIT TO :F4:CREDIT
COPIED :F0:CREDIT.MAC TO :F4:CREDIT.MAC
COPIED :F0:DEBUG TO :F4:DEBUG
COPIED :F0:DELETE TO :F4:DELETE
COPIED :F0:DETACH TO :F4:DETACH
COPIED :F0:DIR TO :F4:DIR
COPIED :F0:HELP TO :F4:HELP
COPIED :F0:HEXOBJ TO :F4:HEXOBJ
COPIED :F0:IDISK TO :F4:IDISK
COPIED :F0:IXREF TO :F4:IXREF
COPIED :F0:LIB TO :F4:LIB
COPIED :F0:LINK TO :F4:LINK
COPIED :F0:LINK.OVL TO :F4:LINK.OVL
COPIED :F0:LOCATE TO :F4:LOCATE
COPIED :F0:OBJHEX TO :F4:OBJHEX
COPIED :F0:PDS.HLP TO :F4:PDS.HLP
COPIED :F0:RENAME TO :F4:RENAME
COPIED :F0:SERIAL TO :F4:SERIAL
COPIED :F0:SUBMIT TO :F4:SUBMIT
COPIED :F0:SYSPDS.LIB TO :F4:SYSPDS.LIB
A0>
```

Comments

A message is displayed for each file copied, and, when all the files are copied, the operating system prompt is returned. Now, the bubble memory can be used interchangeably with a system disk. The files appearing in copied message depend on the software package being used.



Key-in Sequence	Comments
 Y	Remove the diskette from drive 0. Press the RESET key. The message shown above is displayed. Press the Y key to initialize from bubble memory.

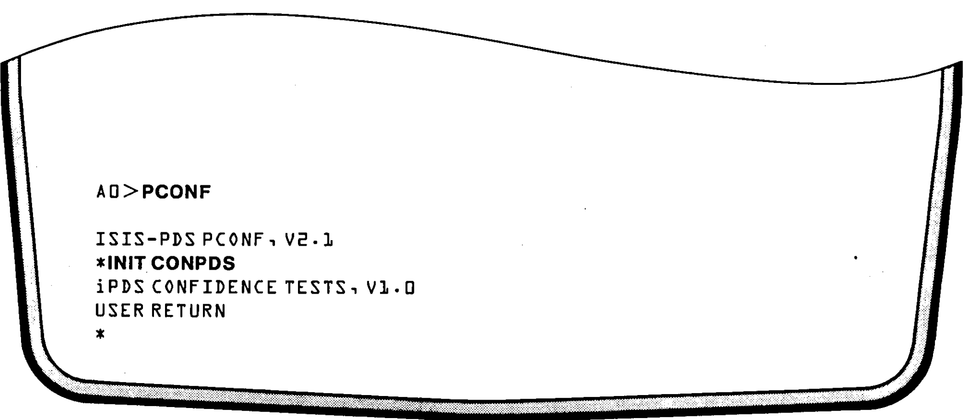


Comments

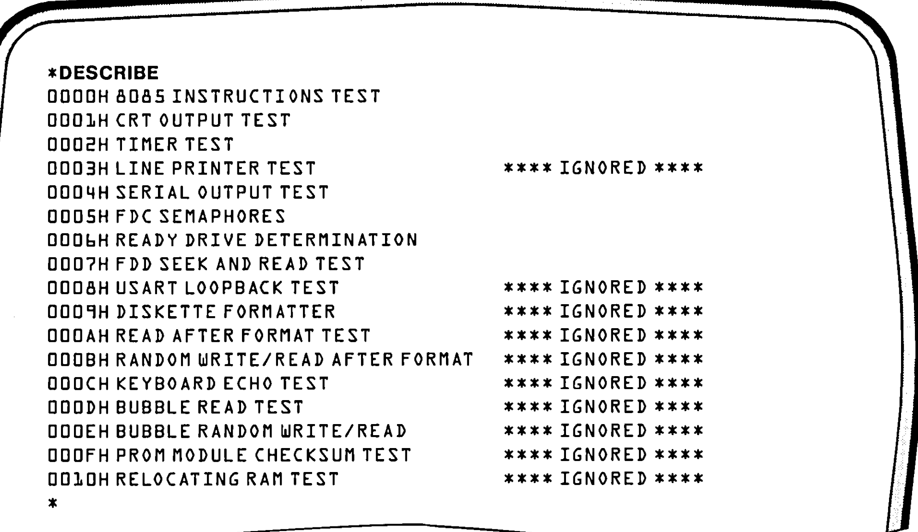
The system will initialize from the bubble memory device. See Appendix D for suggestions on the best use of the bubble memory.

Running the Confidence Test

It is recommended that the confidence test be run before proceeding further to ensure that the system is functioning properly. The confidence test is described in detail in Appendix B. Before running these tests, use the IDISK command as described previously to generate one ISIS-PDS disk for each drive in the system. Place an initialized disk in each drive in the system. The disk test is a read only test, so the disk will not be harmed.



Key-in Sequence	Comments
PCONF	Enter the PCONF command under ISIS-PDS to load the test programs.
INIT CONPDS	The INIT CONPDS commands are now ready for execution. The USER RETURN line displayed on the screen means that the confidence test has been initialized.



Key-in Sequence	Comments
DESCRIBE	The DESCRIBE command displays a listing of the tests available and the tests currently ignored. This screen shows the tests initially recognized when CONPDS is first run. Only these default tests are run in this example. The other tests require iPDS options and/or user interaction. See Appendix B for a complete description of all the tests available. To run a test, it must be recognized. Two CONPDS commands (IGNORE and RECOGNIZE) are used to set up the tests to be run based on the equipment and options in the system.

```
*TEST
TEST 0003H                              **** IGNORED ****
TEST 0008H                              **** IGNORED ****
TEST 0009H                              **** IGNORED ****
TEST 000AH                              **** IGNORED ****
TEST 000BH                              **** IGNORED ****
TEST 000CH                              **** IGNORED ****
TEST 000DH                              **** IGNORED ****
TEST 000EH                              **** IGNORED ****
TEST 000FH                              **** IGNORED ****
TEST 0010H                              **** IGNORED ****
0000H 8085 INSTRUCTIONS TEST            ''PASSED''
0001H CRT OUTPUT TEST
```

Key-in Sequence	Comments
TEST	The TEST command runs the tests recognized. Tests 03H and 08H through 10H are ignored. Tests 00H through 02H and 04H through 07H are run. None of these tests require user interaction. However, one of the tests is a read only disk test, so there is disk activity during the testing. In the initial state, pass/fail messages are displayed for each test run.

```
0002H TIMER TEST                        ''PASSED''
0004H SERIAL OUTPUT TEST                ''PASSED''
0005H FDC SEMAPHORES                    ''PASSED''
0006H READY DRIVE DETERMINATION         ''PASSED''
DRV-0 READY, DRV-1 N RDY, DRV-2 N RDY, DRV-3 N RDY
                                        ''PASSED''
0007H FDD SEEK AND READ TEST            ''PASSED''
*EXIT
A0>
```

Key-in Sequence	Comments
	This screen shows the rest of the display after the CRT characters test. In this example, only one disk drive was ready for the disk drive tests. See Appendix B for a detailed description of the confidence test and other commands that can be run.
EXIT	When the tests are complete, the EXIT command is entered to return to the ISIS-PDS operating system.

CHAPTER 4 COMMAND APPLICATIONS

Functional Summary of Commands

The commands recognized by the ISIS-PDS operating system can be divided into six functional groups:

- System management group
- Device management group
- File management group
- Text editing group
- Program development group
- Program execution group

These groups are described in more detail in the following sections. The commands for each group are listed with that group. A reference is included for further information on each command. Chapters that are mentioned refer to further information in this manual; titles refer to other manuals. Examples are also given to show how to use these commands.

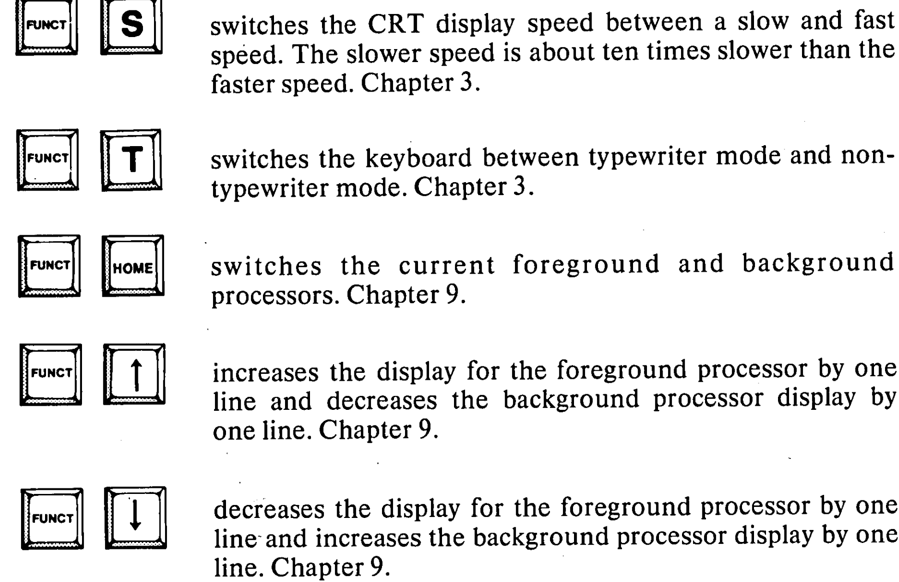
The examples in this chapter assume the disk that was created in the demonstration in Chapter 3 is available. Many of the examples in this chapter and in the rest of the manual assume the files created in previous examples.

Additionally, in some of these examples, the screen shown in the manual may not exactly match the screen resulting from actually running the examples. However, the differences are insignificant. For example, the version numbers actually appearing on the screen when a command is run may differ from that shown in the manual if a new version of the command is used.

System Management Commands

The system management commands display status and help information for the system. Some of these commands also control the processors in a dual processor system. The following commands are in this group and are described in the chapters indicated. Only the HELP command has a corresponding command file on the disk. A sample dual processing session is given in Chapter 9.

HELP	displays help information for operating system commands. Chapter 5.
[?]	displays the version number of the current Command Line Interpreter (CLI). Chapter 5.
[FUNCT] [R]	reloads the ISIS-PDS operating system. Chapter 9.



switches the CRT display speed between a slow and fast speed. The slower speed is about ten times slower than the faster speed. Chapter 3.

switches the keyboard between typewriter mode and non-typewriter mode. Chapter 3.

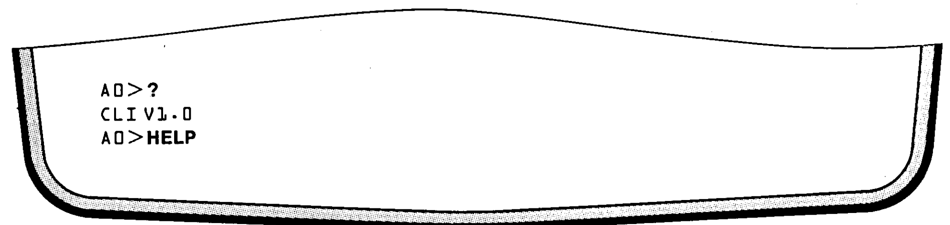
switches the current foreground and background processors. Chapter 9.

increases the display for the foreground processor by one line and decreases the background processor display by one line. Chapter 9.

decreases the display for the foreground processor by one line and increases the background processor display by one line. Chapter 9.

Sample System Management Commands

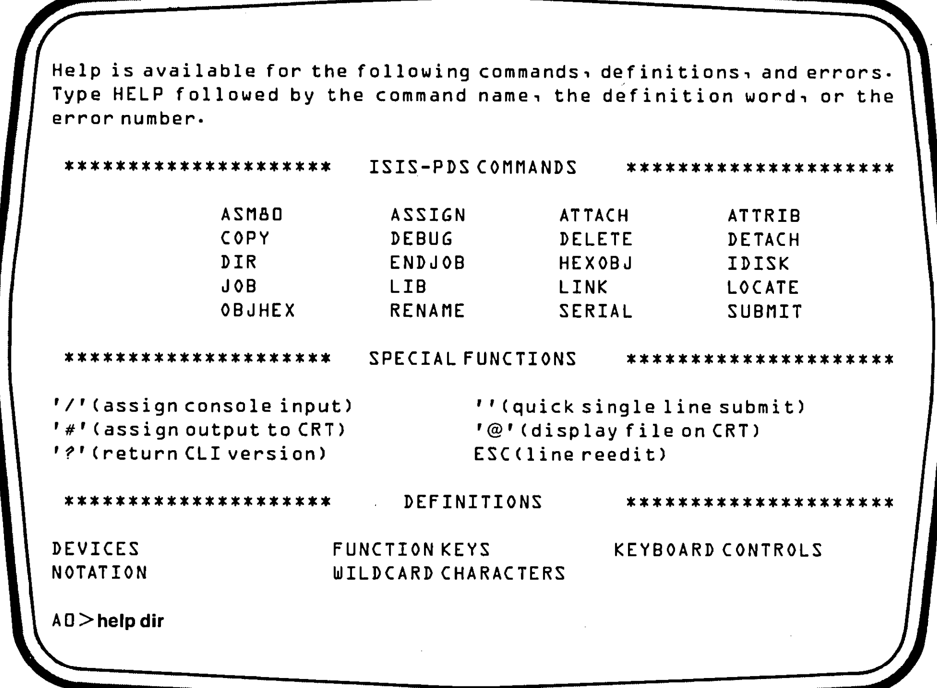
This series of examples illustrates the use of the commands already described.



Key-in Sequence | **Comments**

The ? function displays the version number of the current Command Line Interpreter. Note that ? is one example of a command that does not correspond to a file on the disk. The ? command is always resident in memory.

The HELP command is one of the most important commands for the new user.



Key-in Sequence	Comments
	HELP displays information about the system. Additional information is displayed about any of the names shown in this screen by typing HELP followed by the name.
	Type T while holding down the FUNCT key to switch to typewriter mode on the keyboard. Then, characters typed at the keyboard are lower case unless the SHIFT key is used. The output displayed on the CRT screen by a program is not affected by FUNCT-T.
help dir	Typing help dir returns information about the DIR command.

```
DIR              Displays index of disk files on the specified disk
                 device

    DIR[TO <pn>][FOR <pn>][<n>][I | J | K | L | F | O | P | Z]

      TO <pn>   Device to receive directory listing.
      FOR <pn>  Scope of the directory listing.
      <n>       Logical device form which files are listed.
      I         All files, including those with the invisible attri-
                bute I, are listed.
      J         Only files with User Defined attribute J included.
      K         Only files with User Defined attribute K included.
      L         Only files with User Defined attribute L included.
      F         Fast listing; only filenames and extensions.
      O         Single column listing.
      P         Single drive directory.
      Z         Only summary line is listed.

A0>help esc
```

Key-in Sequence	Comments
	Type S while holding down the FUNCT key to slow down the scrolling on the screen display.
help esc	Typing help esc returns information about the use of the escape key for editing command lines.

```
ESC      Re-edit previous command line or current command line and re-
         execute.

    ESC

After entering command, the following keyboard commands can be used:

    ESC      Execute entire line
    RETURN   Execute line up to current cursor position
    ←        Move cursor left
    →        Move cursor right
    CTRL-A   Encloses characters to be inserted
    CTRL-B   Move cursor to beginning of line
    CTRL-D   Delete character at current cursor
    CTRL-L   Move cursor to end of line.
    CTRL-X   Terminate re-edit and return to ISIS
    RUBOUT   Same as CTRL-D.

A0>HELP FUNCTION
```

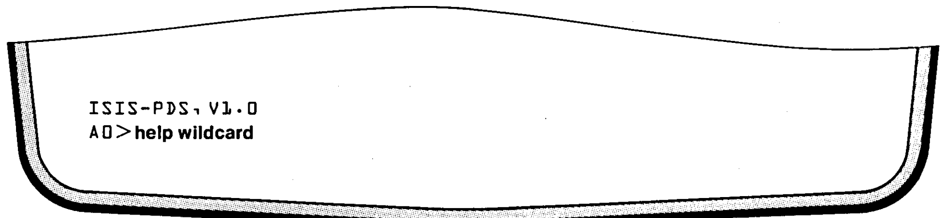
Key-in Sequence	Comments
	This is the resulting screen display showing the ESC key information.

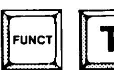
Key-in Sequence	Comments
	The FUNCT-T key combination returns the keyboard to non-typewriter mode.
	Typing FUNCT-S returns the screen to normal scrolling speed. Both of these function key combinations act as switches setting and resetting the function every other time.
HELP FUNCTION	Typing HELP followed by FUNCTION returns information on the user-defined function keys.

```
FUNCTION KEYS
    FUNC-<n>               Pressing the 'FUNC' key and a numerical key,
                           simultaneously, causes console input to be
                           input from the file JOB<n>.CSD. <n>'s value
                           is from 0 to 9.
    FUNC-R                 Pressing the 'FUNC' key and 'R' will cause a
                           software reset. If running under DEBUG, it
                           will cause a break in user program execution.
    FUNC-S                 Toggles the display rate of the CRT.
    FUNC-T                 Toggles the keyboard between upper and lower
                           case.
    FUNC-HOME              Alternately switches the keyboard between
                           processors on a dual processing system. Also,
                           switches the bottom part of the screen with the
                           top part.
    FUNC-↑                 Increases by one line the display size of the
                           bottom half of the CRT screen in a dual process-
                           ing system.
    FUNC-<Down Arrow>      Decreases by one line the display size of the
                           bottom half of the CRT screen in a dual process-
                           ing system.
A0>
```

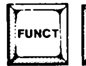
Comments

This is the screen display resulting from the HELP FUNCTION command.



Key-in Sequence	Comments
	Type R while holding down the FUNCT key to reset the processor.
	Switch to typewriter mode.
help wildcard	Type help wildcard to display information on wildcard characters.

```
WILDCARD CHARACTERS

The wildcard characters (* and ?) can appear in a filename with the fol-
lowing meanings. The chart below lists the commands that allow wildcard
characters.

    *    In a filename specifies a match to any characters and any number
         of characters in that position.
    ?    In a filename specifies a match to any single character in that
         position.

The commands that allow wildcard characters are:
       ATTRIB
       COPY (To Transfer), i.e., non-appending form of the command.
       DELETE
       DIR

A0>
```

Key-in Sequence	Comments
	This screen is displays the information concerning wildcard characters.
	Type the FUNCT-T combination to return to non-typewriter mode.

Device Management Commands

Many different devices can be connected to the development system for data storage and input/output operations. The operating system provides commands to control some of these devices. The following device management commands are covered in Chapter 5. The /, #, and FUNCT-<n> commands are not separate files on the disk but are part of the operating system that is resident in memory at all times.

IDISK initially prepares disks and bubble memory for use with the operating system.

ASSIGN displays or assigns the mapping of physical to logical devices.

[#] re-assigns the system output to the CRT display screen.

[FUNCT] [<n>] changes the system input from the keyboard to the file named JOB<n>.CSD where <n> is a one-digit number from 0 to 9. Pressing <n> followed by the RETURN key is the same as pressing FUNCT <n>. This function is useful in executing often used commands such as the DIR command and the HELP command. Examples are in the section "Program Execution Commands" in this chapter.

[/] changes the system input from the keyboard to a file or device which is specified by the user. See the JOB command for related information. An example of this command is given in the "Program Execution Commands" of this chapter.

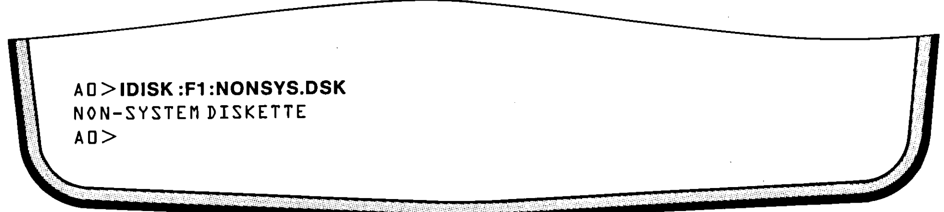
SERIAL initializes the serial I/O port.

ATTACH assigns a row of multimodules to a processor.

DETACH releases a row of multimodules from a processor.

Formatting a Non-System Disk

In this series of examples, a non-system disk is formatted on a multiple drive system and then on a single drive system.

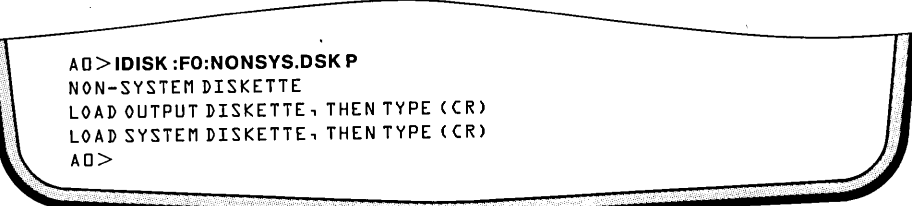


Key-in Sequence | **Comments**

IDISK :F1:NONSYS.DSK

This command illustrates formatting a non-system diskette on a system with more than one drive. The next screen display illustrates formatting of a non-system disk on a single drive system. See Chapter 3 for an example of formatting a system disk.

Place a new diskette (without a write protect tab) in drive 1. Enter the command as shown. When the new disk is formatted, the operating system prompt appears.



Key-in Sequence | **Comments**

IDISK :F0:NONSYS.DSK P

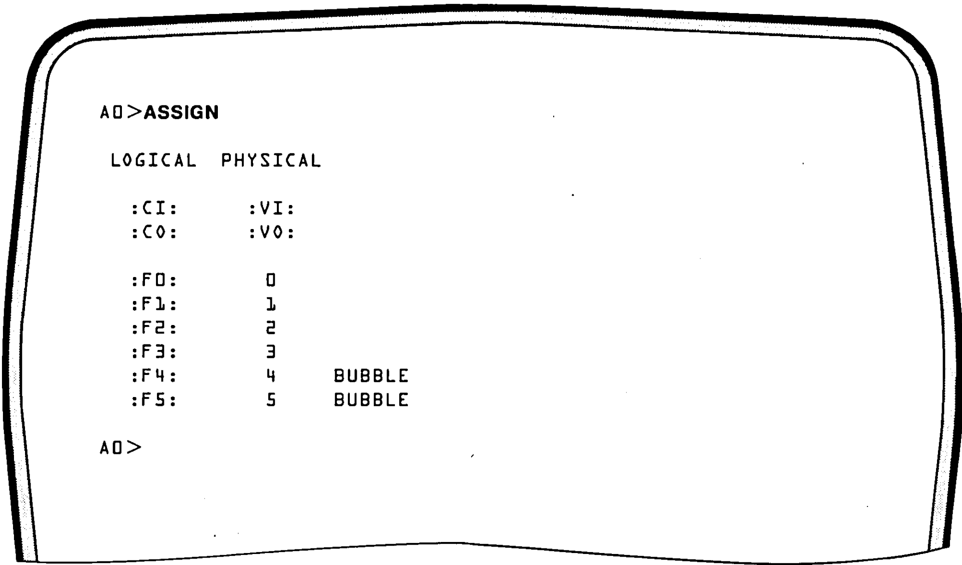
This example shows how to format a non-systen diskette on a single drive system. With the system disk in the drive, enter the command line.

When the prompt appears, remove the system diskette and insert the blank diskette without a write protect tab. Then, press the RETURN key.

When the next prompt appears, remove the diskette being formatted, and insert the system diskette. Then, press the RETURN key. The operating system prompt appears.

Changing the System Input and Output Devices

In the next few examples, the ASSIGN command is illustrated as it would be used to assign logical device names recognized by the operating system to physical devices.

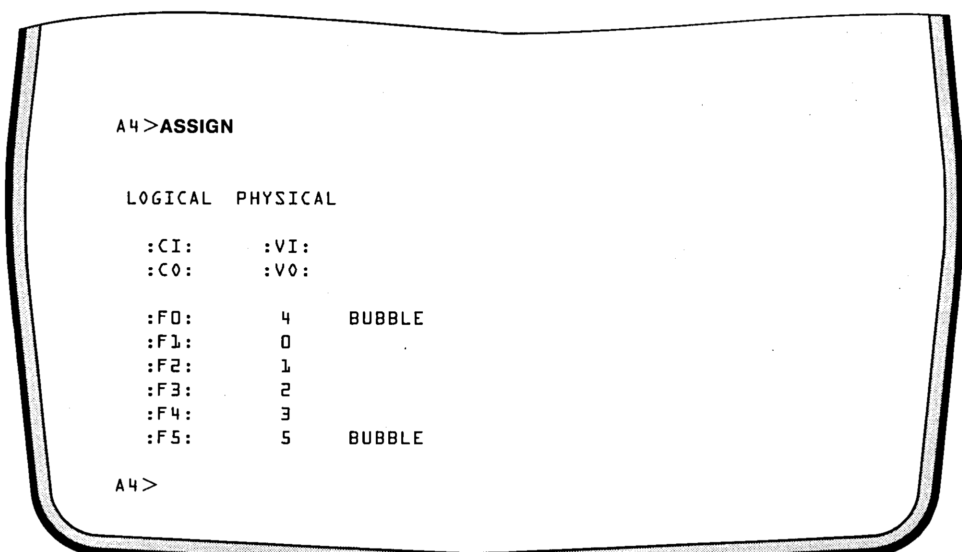


Key-in Sequence **Comments**

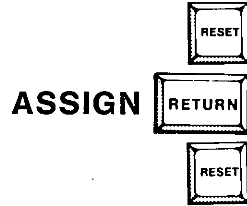
ASSIGN

The ASSIGN command is a multiple purpose command. In the simple form shown, it displays the mapping of ISIS logical device names to the corresponding physical devices of the system. This screen example shows the default assignment after initializing the system from drive 0.

The ISIS logical device names are shown on the left and the physical device names are shown on the right. These names are explained in detail in Chapter 5. However, the digits 0 through 5 refer to the disk devices and bubble memory. The digit 0 is the internal disk drive, 4 is the bubble memory installed at connector J1, and 5 is the bubble memory installed at connector at J3. The digits 1 through 3 are the optional external disk drives.

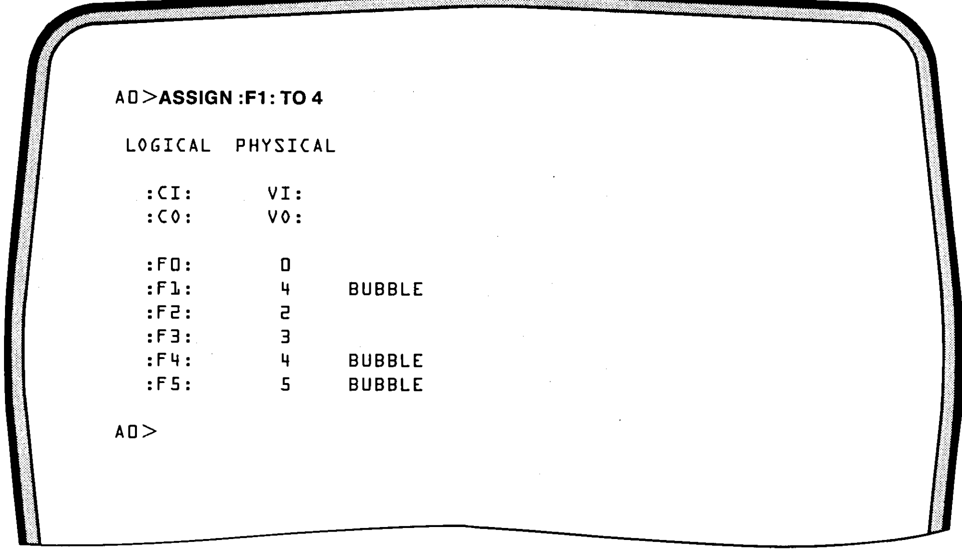


Key-in Sequence | **Comments**



Open the drive door and reset the system. This example shows the default assignment after initializing the system from the bubble memory multimodule.

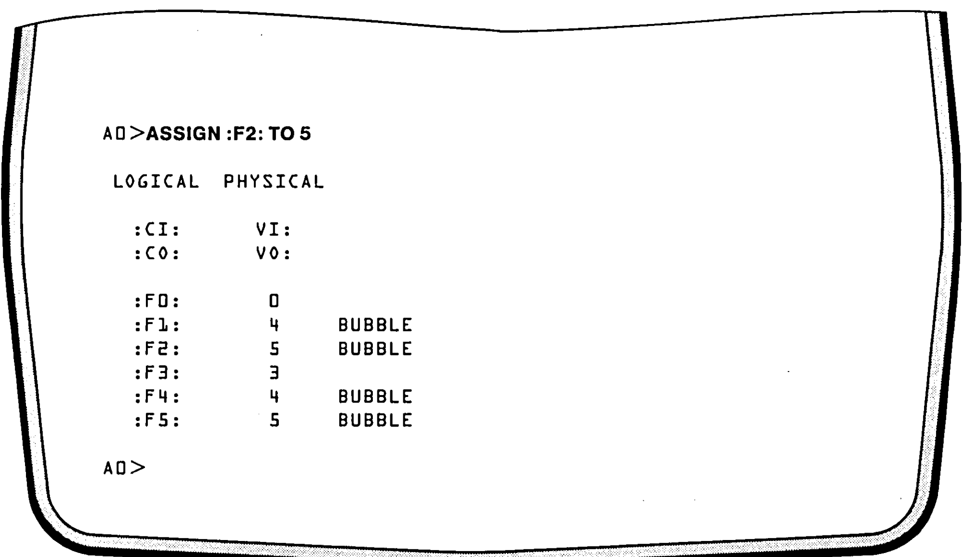
Close the drive door and reset the system.



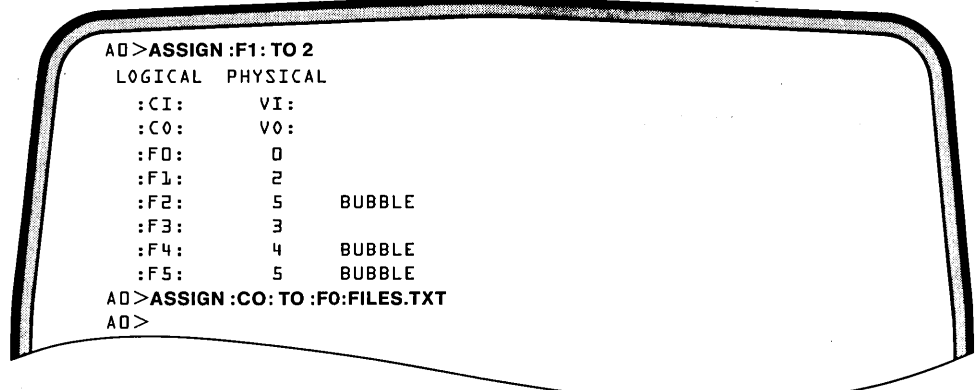
Key-in Sequence | **Comments**

ASSIGN :F1: TO 4

In this example, the bubble memory multimodule installed at connector J1 is assigned to the logical device :F1:. This assignment is appropriate for a system with bubble memory and no external disk drives.



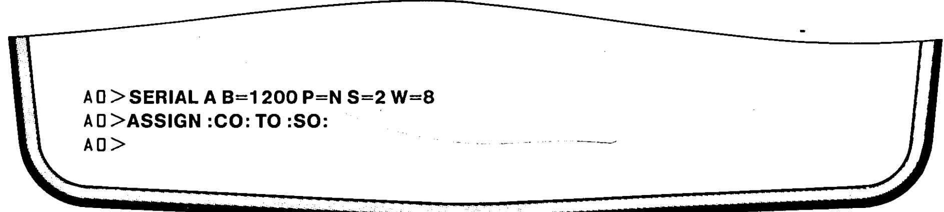
Key-in Sequence	Comments
ASSIGN :F2: TO 5	This example shows how to assign the bubble memory at connector J3 to the logical device :F2:. This assignment is appropriate for a system with bubble memory and no external drives.



Key-in Sequence	Comments
ASSIGN :F1: TO 2	This example shows how to reassign disk drives when a drive is not working. This eliminates the need to reconfigure or re-install the other drives when one drive is not working. Here, programs which expect files to be on the logical device, :F1:, can still be run.
ASSIGN :CO: TO :FO:FILES.TXT RETURN	This command assigns the system output file on on drive 0. The display generated by the ASSIGN command is not on the screen. This display is stored in the file FILES.TXT on drive 0.
DIR	After changing the system output device, type the DIR command. No output appears on the screen. The DIR command and the resulting directory listing are stored in the file FILES.TXT on drive 0.
	After running the DIR command, enter the # command to return the system output to the CRT screen.

Using the Serial Port

The next example shows the SERIAL command used to configure the serial port and then the ASSIGN command to use the serial port in the system.



Key-in Sequence	Comments
SERIAL A B=1200 P=N S=2 W=8	There are two steps in using a serial device: first, configure the system for the device with the SERIAL command and, second, assign the device to one of the system's logical devices with the ASSIGN command. The SERIAL command shown here prepares the system for communication with an asynchronous, serial device at a speed of 1200 baud, with no parity, with two stop bits and with a word length of 8.
ASSIGN :CO: TO :SO:	This command assigns the console output (initially the CRT screen) to the serial device. The output display of the command showing the new assignment of device appears on the serial device. If no serial device is connected, the display is not echoed anywhere. There is no way to way to determine what is actually typed in further commands.
	A quick way to re-assign the system output device to the iPDS CRT screen is with the # command. Type # followed by the return key, and the operating system prompt is displayed on the iPDS CRT screen, indicating that system output is again echoed on the CRT. The # command is another example of a command always resident in memory and not corresponding to a file.
	Reset all the assignments to their defaults by pressing the RESET key to reset the system.

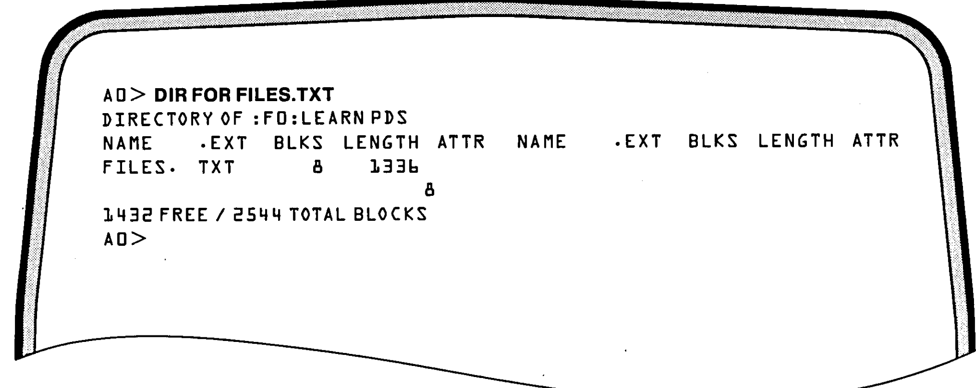
File Management Commands

Two important features of the operating system are its device and file handling capabilities. Several file management commands are provided with the operating system. The following commands are covered in Chapter 5:

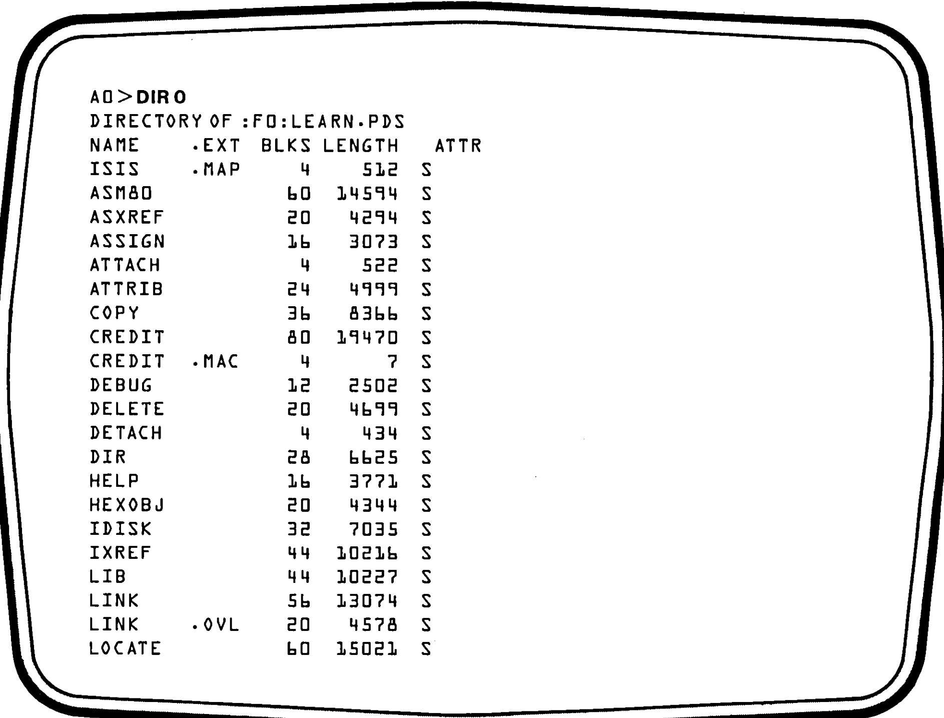
DIR	displays a list of the files stored on a disk or on bubble memory.
ATTRIB	displays and modifies the attributes of a file.
COPY	transfers files and appends files.
DELETE	removes files from the disk.
RENAME	changes the filename and/or extension of a file.
@	displays the contents of a file on the screen. This command is in the part of ISIS-PDS that is resident in memory, and it does not correspond to a command file on the disk.

Displaying a List of Files

The DIR command is used display available files.



Key-in Sequence	Comments
DIR FOR FILES.TXT	This example illustrates a form of the DIR command that can be used to find out if a file exists on a drive or not. Only the file requested in the FOR clause is displayed in the listing. A search of all the files on the disk is not needed. In this screen display, all the directory information on the file is shown: the filename, the number of blocks used by the file, the length of the file in bytes, and any attributes set for the file.

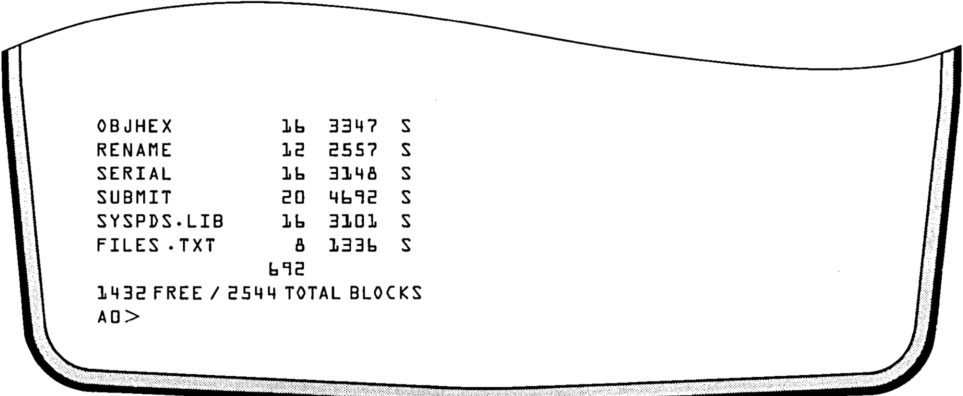


Key-in Sequence **Comments**

DIR O

This sample display results from the DIR command with the O option which lists the files in a single column with all the directory information about each file.

Type the CTRL-S key combination to stop the scrolling of the screen display. The resulting display depends on when the CTRL-S is typed.

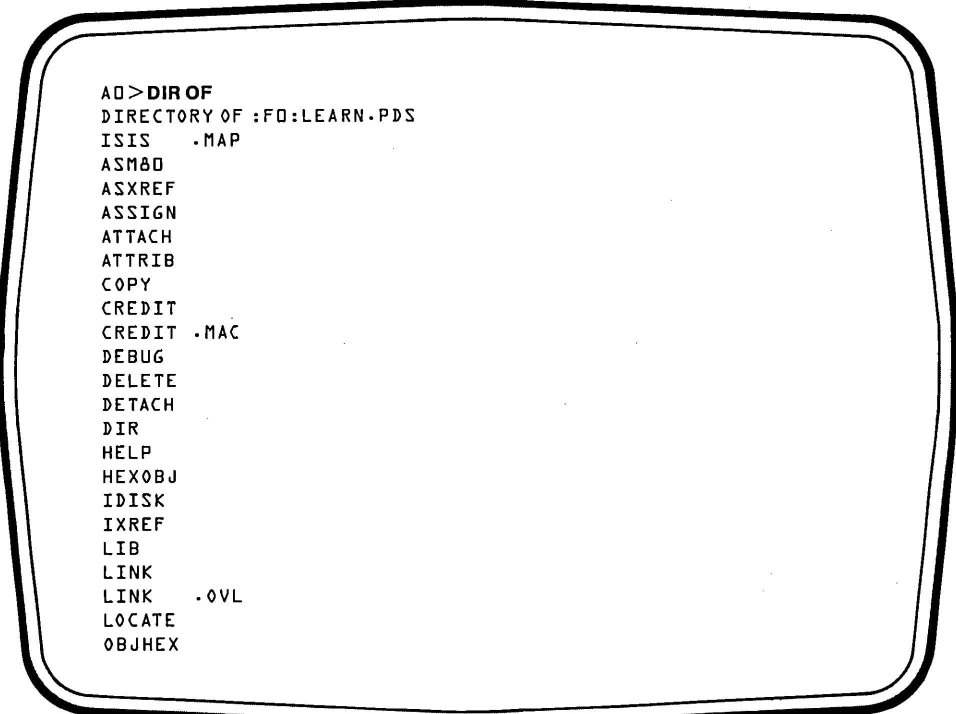


Key-in Sequence **Comments**

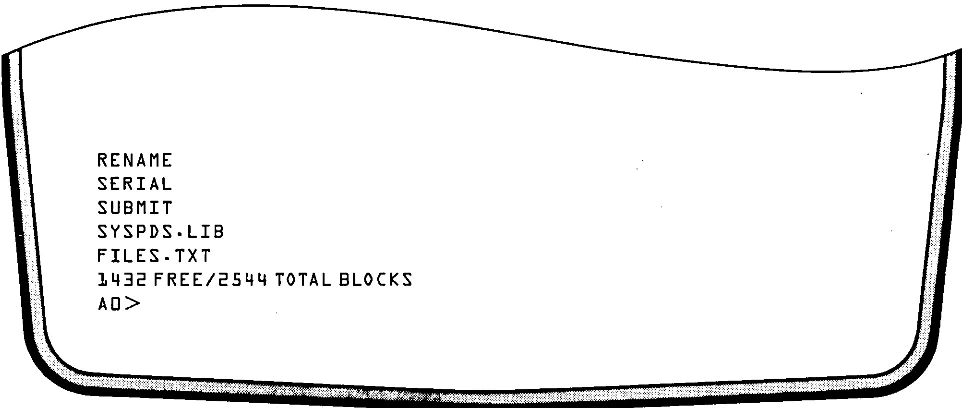
Type the CTRL-Q combination to continue scrolling of the screen display. This example shows the rest of the display from the preceding command.

```
A0>DIR F
DIRECTORY OF :F0:LEARN.PDS
ISIS     .MAP ASM80
ASXREF        ASSIGN
ATTACH        ATTRIB
COPY          CREDIT
CREDIT   .MAC DEBUG
DELETE        DETACH
DIR           HELP
HEXOBJ        IDISK
IXREF         LIB
LINK          LINK.OVL
LOCATE        OBJHEX
RENAME        SERIAL
SUBMIT        SYSPDS.LIB
FILES.TXT
1432 FREE / 2544 TOTAL BLOCKS
A0>
```

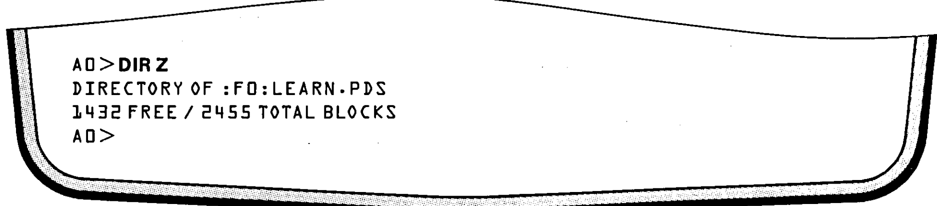
Key-in Sequence	Comments
DIR F	This sample display shows the results from the DIR command with the F option. The F option produces a fast listing without the detailed data of the normal directory listing. The actual display may vary depending on the version of the operating system being used.



Key-in Sequence	Comments
DIR OF	This example shows the results of combining the O and F options.
	Type the CTRL-S key combination to halt scrolling of the screen display.



Key-in Sequence	Comments
	Type the CTRL-Q combination to continue the scrolling of the screen display. This example shows the rest of the display from the preceding command.



Key-in Sequence

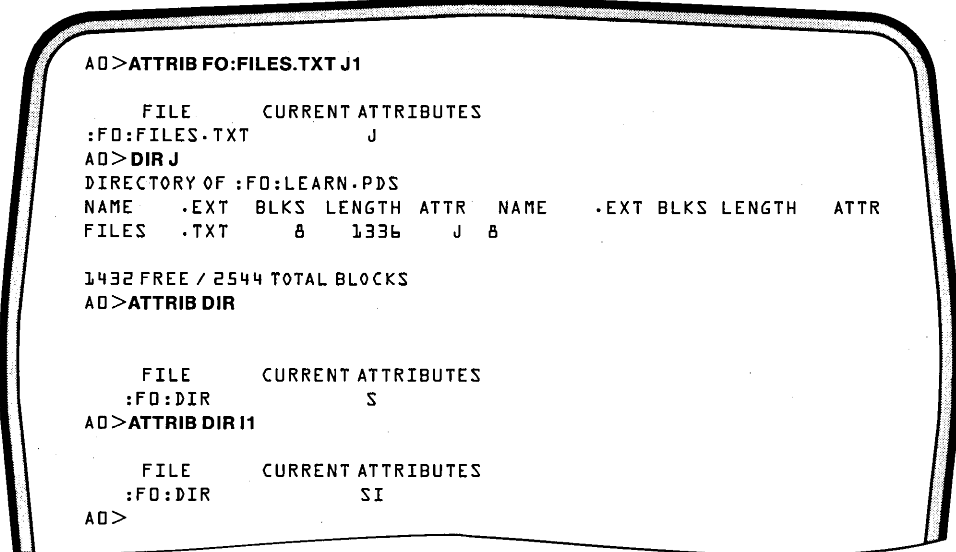
DIR Z RETURN

Comments

This screen display shows sample output from the DIR command with the Z option. The Z option displays only the summary line of the normal directory listing.

Assigning and Removing File Attributes

The following screens illustrate how attributes are assigned and how these attributes are used.



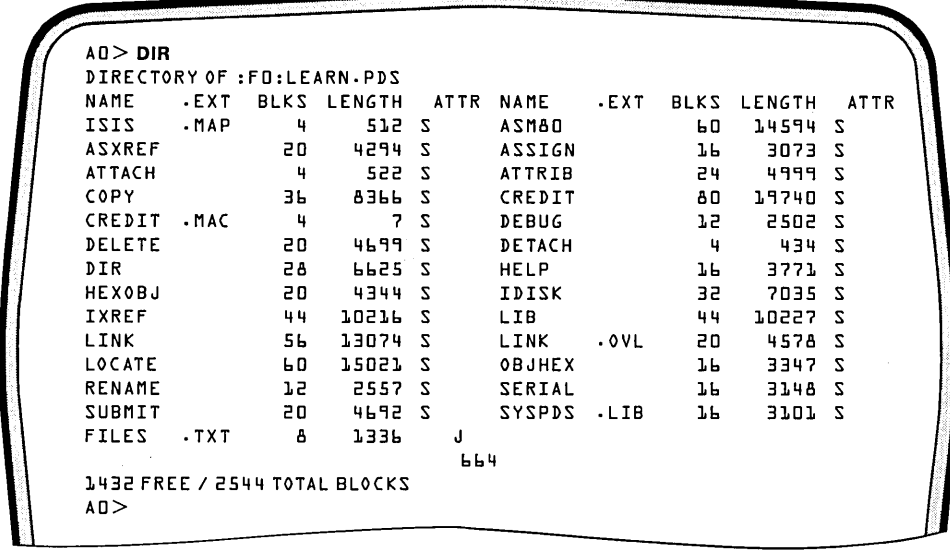
Key-in Sequence

ATTRIB FILES.TXT J1

Comments

The ATTRIB command assigns attributes to files and removes attributes from files. See Chapter 5 for a detailed explanation of attributes. Here, the J attribute is assigned to the file by specifying the file's pathname and the attribute name followed by the digit 1. The digit 1 sets the attributes; 0 resets it. Attributes can be used to limit the scope of commands.

Key-in Sequence	Comments
DIR J	When the DIR command is entered followed by an attribute, the directory listing is produced for only those files having the specified attribute. Attributes can be combined to further limit the scope of files used in the command.
ATTRIB DIR	The ATTRIB command can also be used to display the current attributes of a file.
ATTRIB DIR I1	The Invisible attribute is set for the file DIR. Invisible files are not included in a normal directory listing.



Key-in Sequence	Comments
DIR	Now that the file, DIR, has the Invisible attribute, it no longer appears in the normal directory listing even though it is still on the diskette.

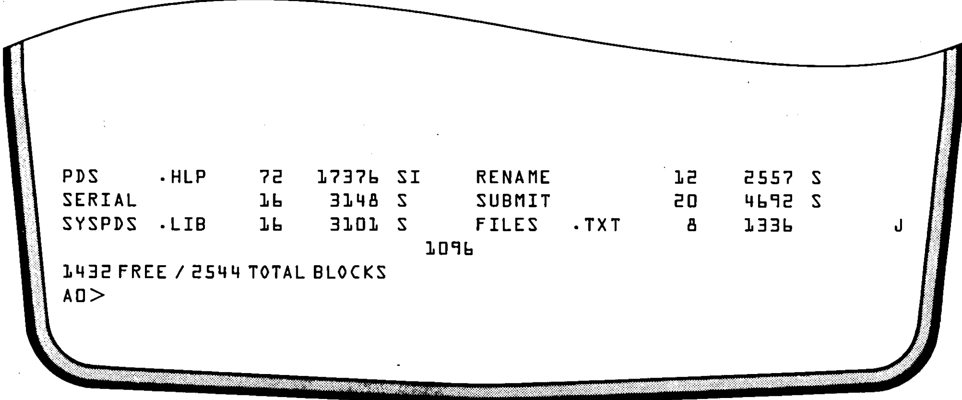
```
A0> DIR I
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT  BLKS  LENGTH   ATTR  NAME    .EXT  BLKS  LENGTH   ATTR
ISIS    .DIR   16    3840   I  F  ISIS    .FRE    4      80   I  F
ISIS    .T0    16    3840   I  F  ISIS    .LAB    4     768   I  F
ISIS    .PDS   52   12088   SI F  ISIS    .CLI   16    3113   SI F
ISIS    .MAP    4     512   S     ASM80          60   14594   S
ASM80   .OV0   12    1847   SI    ASM80   .OV1   12    2108   SI
ASM80   .OV2   12    2115   SI    ASM80   .OV3    8     996   SI
ASM80   .OV4  100   24413   SI    ASM80   .OV5   80   20037   SI
ASXREF         20    4294   S     ASSIGN         16    3073   S
ATTACH          4     522   S     ATTRIB         24    4999   S
COPY           36    8366   S     CREDIT         80   19470   S
CREDIT  .MAC    4       7   S     DEBUG          12    2502   S
DELETE         20    4699   S     DETACH          4     434   S
DIR            28    6625   SI    HELP           16    3771   S
```

Key-in Sequence **Comments**

DIR I RETURN

This example shows the results from the DIR command with the I option. Note that the I option differs slightly from the use of the other attributes as options. When other attributes are used as options, only files with the specified option appear in the directory. When the I option is used, Invisible files are included in addition to the other files. Notice the files near the top of the listing with the F attribute. The F attribute designates system files used in formatting a new disk and should not be assigned or removed.

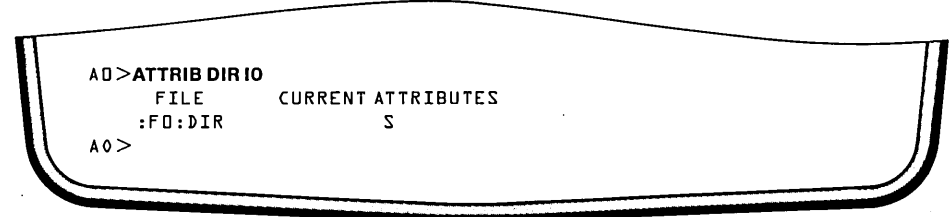
Type CTRL-S to halt the scrolling of the screen.



```
PDS     .HLP   72   17376   SI    RENAME         12    2557   S
SERIAL         16    3148   S     SUBMIT         20    4692   S
SYSPDS  .LIB   16    3101   S     FILES   .TXT    8    1336         J
                           1096
1432 FREE / 2544 TOTAL BLOCKS
A0>
```

Key-in Sequence **Comments**

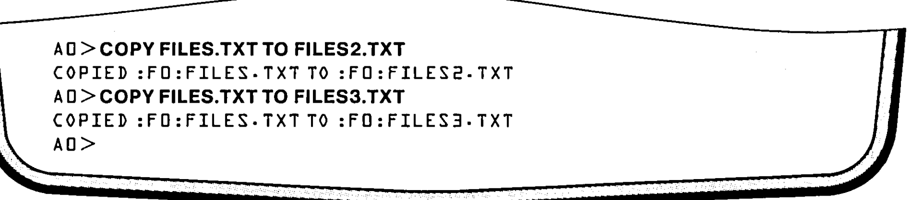
Type the CTRL-Q combination to continue scrolling the screen display. Depending on the version of the operating system used, directory listings may vary slightly, but the listing will be similar to this one.



Key-in Sequence	Comments
ATTRIB DIR I0	To remove an attribute from a file, specify the file's path-name and the attribute name followed by the digit 0 to remove the attribute.

Copying Files

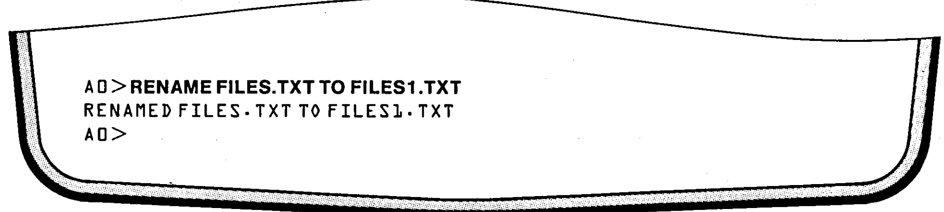
Two files are created with the COPY command in the next example. These files will be used in later examples.

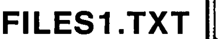


Key-in Sequence	Comments
COPY FILES.TXT TO FILES2.TXT	The file being copied (the source file) was created in a previous example. The COPY command is used here to duplicate the source file in the destination file. After this command is complete, the file, FILES2.TXT, contains a copy of the data in FILES.TXT as indicated in the message displayed.
COPY FILES.TXT TO FILES3.TXT	The file, FILES3.TXT, contains a duplicate of the data in FILES.TXT.

Changing Filenames

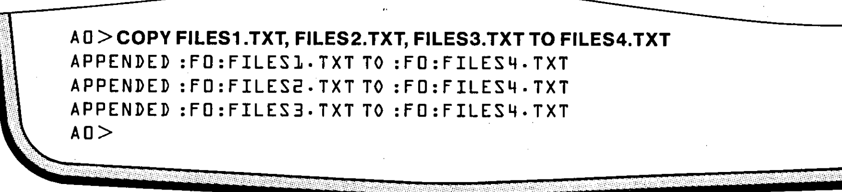
The name of a file can be changed with the RENAME command as shown in the following example.



Key-in Sequence	Comments
RENAME FILES.TXT TO FILES1.TXT 	The RENAME command is used to rename the source file. It does not create a new file nor does it affect the content of the source file. There is now no file named FILES.TXT; it is now named FILES1.TXT.

Appending Files

The COPY command can also be used to append files as shown in the next example.



Key-in Sequence	Comments
COPY FILES1.TXT, FILES2.TXT, FILES3.TXT TO FILES4.TXT	The COPY command can also be used to append files. If more than one source file is specified, each source is appended to the previous source. In this case, the files created in the previous examples are appended. Since FILES1.TXT, FILES2.TXT, and FILES3.TXT all contain the same data, FILES4.TXT will contain three copies of this data.

Displaying a Text File on the CRT

The @ command can be used to display a text file on the CRT. The COPY command can also be used. However, with the @ command, several controls are available that determine the speed of the display. The @ command also provides other controls over the display of the file. Any byte in the file with no corresponding display character is displayed as a blank.

```
A0>@FILES1.TXT

 LOGICAL   PHYSICAL

   :CI:       :VI:
   :CO:       :F0:FILES.TXT

   :F0:       0
   :F1:       1
   :F2:       2
   :F3:       3
   :F4:       4    BUBBLE
   :F5:       5    BUBBLE

A0>DIR
DIRECTORY OF :F0:LEARN.PDS
ISIS    .MAP      4    512  S      ASM80         60   14594  S
ASXREF           20   4294  S      ASSIGN        16    3073  S
ATTACH            4    522  S      ATTRIB        24    4999  S
```

Key-in Sequence	Comments
@FILES1.TXT	The @ command is used to display the contents of a file on the CRT screen. It should be used to display ASCII files. The @ command displays the file until the screen is full and then stops. In this case, the file, FILES1.TXT, is displayed. Remember, the file was created by assigning the system output to the file. Thus, the output from the ASSIGN command appears first in the file. Then, the DIR command was entered. So, the DIR command line and the output from the DIR command appears next in the file.

```
COPY            36    8366 S     CREDIT             80   19470 S
CREDIT  .MAC     4       7 S     DEBUG              12    2502 S
DELETE          20    4699 S     DETACH              4     434 S
DIR             28    6625 S     HELP               16    3771 S
HEXOBJ          20    4344 S     IDISK              32    7035 S
IXREF           44   10216 S     LIB                44   10227 S
LINK            56   13074 S     LINK     .OVL      20    4578 S
LOCATE          60   15021 S     OBJHEX             16    3347 S
RENAME          12    2557 S     SERIAL             16    3148 S
SUBMIT          20    4692 S     SYSPDS   .LIB      16    3101 S
FILES   .TXT     4       0
                                688
1420 FREE / 2544 TOTAL BLOCKS
A0>#

A0>
```

Key-in Sequence | **Comments**

Press any key other than one of the @ commands (E, L, Z, B, F, S, or P) to continue the display. The @ command then displays the next screen of data from the file. The RETURN key is used here.

The display is now at the end of the file. Notice the # which was the command entered to change the output back to the CRT screen in the example where FILES.TXT was created. Press the E key to end the display and return to the operating system.

```
A0>@FILES2.TXT

 LOGICAL  PHYSICAL

   :CI:       :VI:
   :CO:       :F0:FILES.TXT

   :F0:       0
   :F1:       1
   :F2:       2
   :F3:       3
   :F4:       4.    BUBBLE
   :F5:       5     BUBBLE

A0>DIR
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT  BLKS  LENGTH  ATTR    NAME     .EXT  BLKS  LENGTH  ATTR
ISIS    .MAP     4     512  S       ASM80             60   14594  S
ASXREF          20    4294  S       ASSIGN            16    3073  S
ATTACH           4     522  S       ATTRIB            24    4999  S
```

Key-in Sequence **Comments**

@FILES2.TXT

This example show that FILES2.TXT contains the same data as FILES1.TXT. The @ command is another example of a command that is resident in memory. There is not file, @, corresponding to this command.

```
COPY             36    8366  S      CREDIT           8019470   S
CREDIT  .MAC      4       7  S      DEBUG             12    2502  S
DELETE           20    4699  S      DETACH             4     434  S
DIR              28    6625  S      HELP              16    3771  S
HEXOBJ           20    4344  S      IDISK             32    7035  S
IXREF            44   10216  S      LIB               44   10227  S
LINK             56   13074  S      LINK     .OVL     20    4578  S
LOCATE           60   15021  S      OBJHEX            16    3347  S
RENAME           12    2557  S      SERIAL            16    3148  S
SUBMIT           20    4692  S      SYSPDS   .LIB     16    3101  S
FILES   .TXT      4       0
                               688
1420 FREE / 2544 TOTAL BLOCKS
A0>#

A0>
```

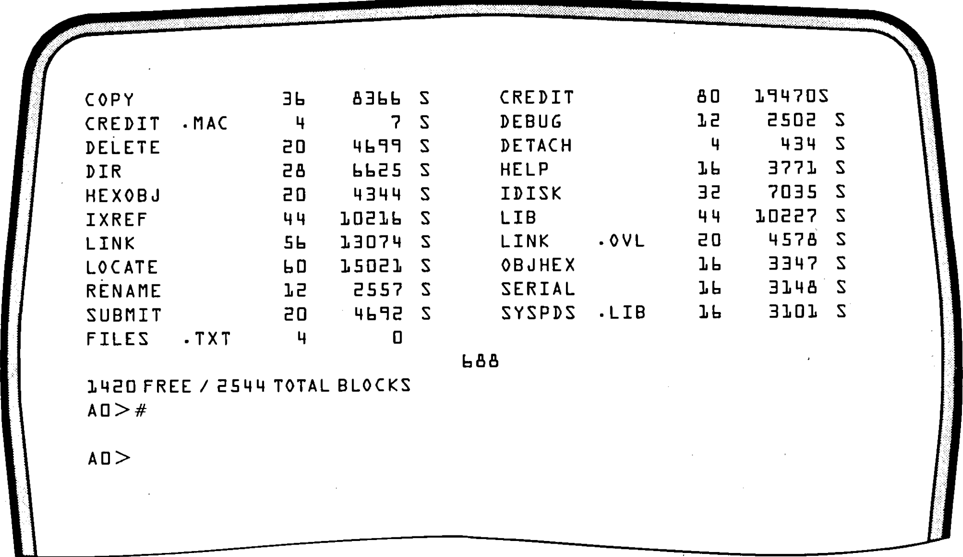
Key-in Sequence **Comments**

Press any key to continue the display.

Again, this is the end of the file. Press the E key to end the display and return to the operating system.

```
A0>@FILES3.TXT

 LOGICAL  PHYSICAL

   :CI:       :VI:
   :CO:       :F0:FILES.TXT

   :F0:       0
   :F1:       1
   :F2:       2
   :F3:       3
   :F4:       4    BUBBLE
   :F5:       5    BUBBLE

A0>DIR
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT  BLKS  LENGTH  ATTR   NAME     .EXT  BLKS  LENGTH   ATTR
ISIS    .MAP     4     512  S      ASM80             60  14594  S
ASXREF          20    4294  S      ASSIGN            16   3073  S
ATTACH           4     522  S      ATTRIB            24   4999  S
```

Key-in Sequence	Comments
@FILES3.TXT RETURN	This example shows that FILES3.TXT contains the same data as FILES1.TXT and FILES2.TXT.

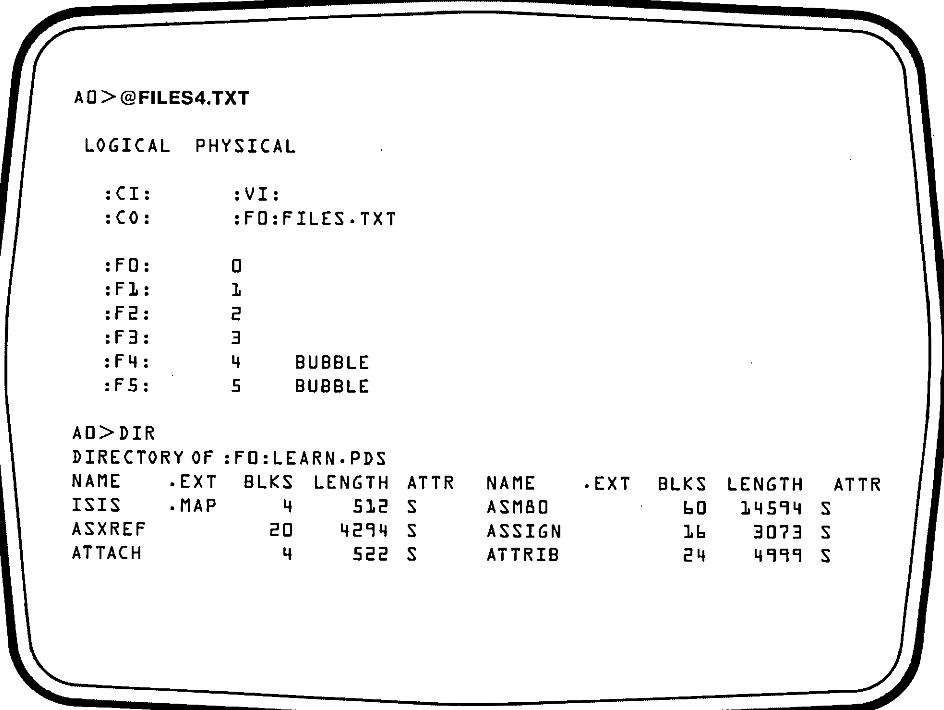


```
COPY             36    8366  S     CREDIT            80  19470S
CREDIT  .MAC      4       7  S     DEBUG             12   2502  S
DELETE           20    4699  S     DETACH             4    434  S
DIR              28    6625  S     HELP              16   3771  S
HEXOBJ           20    4344  S     IDISK             32   7035  S
IXREF            44   10216  S     LIB               44  10227  S
LINK             56   13074  S     LINK      .OVL    20   4578  S
LOCATE           60   15021  S     OBJHEX            16   3347  S
RENAME           12    2557  S     SERIAL            16   3148  S
SUBMIT           20    4692  S     SYSPDS    .LIB    16   3101  S
FILES   .TXT      4       0
                                688
1420 FREE / 2544 TOTAL BLOCKS
A0>#

A0>
```

Key-in Sequence **Comments**

Press any key, except one of the @ command keys, to continue the display.

Press the E key to end the display and return to the operating system.



Key-in Sequence	Comments
@FILES4.TXT	In this example, the output file from the append operation is displayed. This file should contain three copies of the data in the other files.

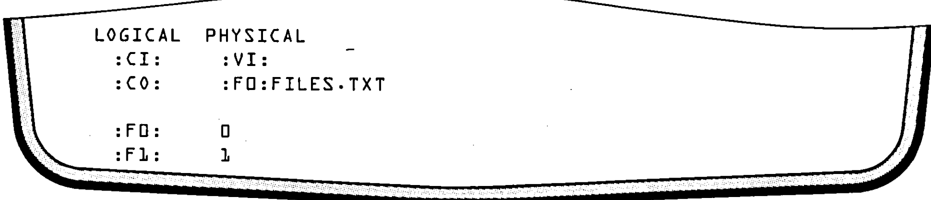
```
COPY              36     8366  S    CREDIT            80   19470  S
CREDIT  .MAC       4        7  S    DEBUG             12    2502  S
DELETE            20     4699  S    DETACH             4     434  S
DIR               28     6625  S    HELP              16    3771  S
HEXOBJ            20     4344  S    IDISK             32    7035  S
IXREF             44    10216  S    LIB               44   10227  S
LINK              56    13074  S    LINK     .OVL     20    4578  S
LOCATE            60    15021  S    OBJHEX            16    3347  S
RENAME            12     2557  S    SERIAL            16    3148  S
SUBMIT            20     4692  S    SYSPDS   .LIB     16    3101  S
FILES   .TXT       4        0
                                  688
1420 FREE / 2544 TOTAL BLOCKS

A0>#

 LOGICAL   PHYSICAL

   :CI:     :VI:
   :CO:     :F0:FILES.TXT
```

Key-in Sequence	Comments
	Press any key to continue the display

```
   :CI:   :VI:
   :CO:   :F0:FILES.TXT

   :F0:   0
   :F1:   1
   :F2:   2
   :F3:   3
   :F4:   4     BUBBLE
   :F5:   5     BUBBLE

A0>DIR
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT  BLKS  LENGTH  ATTR   NAME     .EXT  BLKS  LENGTH  ATTR
ISIS    .MAP     4     512  S      ASM80            60   14594  S
ASXREF          20    4294  S      ASSIGN           16    3073  S
ATTACH           4     522  S      ATTRIB           24    4999  S
COPY            36    8366  S      CREDIT           80   19470  S
CREDIT  .MAC     4       7  S      DEBUG            12    2502  S
DELETE          20    4699  S      DETACH            4     434  S
DIR             28    6625  S      HELP             16    3771  S
HEXOBJ          20    4344  S      IDISK            32    7035  S
IXREF           44   10216  S      LIB              44   10227  S
LINK            56   13074  S      LINK     .OVL    20    4578  S
```

Key-in Sequence	Comments
	Press any key, except one of the @ command keys, to continue the display. Continue pressing a key to continue the display until the end of the file. At the end of the file, the data has appeared three times and pressing a key to continue the display has no effect.

Key-in Sequence	Comments
	The @ command has several subcommands in addition to E. Press the L key. This causes the command to enter line mode where the file is displayed line by line instead of page by page.
	Press the B key several times or hold it down until the screen is blank. This causes the @ command to back up 1024 characters (1K bytes) at a time. Eventually, the beginning of the file is reached. Since the command is in line mode, only the first line of the file is displayed. It is a blank line, so the screen display is blank.

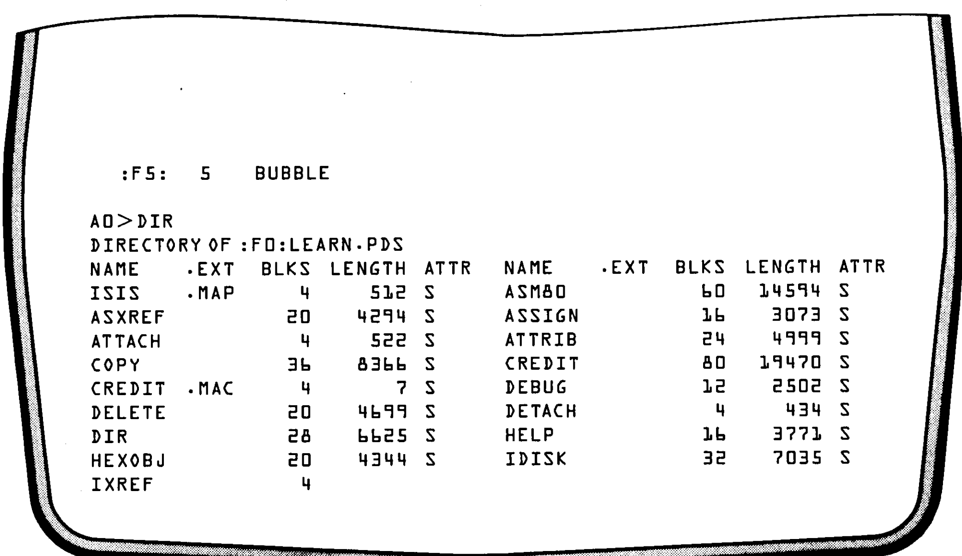


Key-in Sequence	Comments
	Press any key (RETURN is shown here) until the first line of the file appears. Press any key several times to display the file one line at a time.

```
:CO:    :FO:FILES.TXT

:FO:    0
:F1:    1
:F2:    2
:F3:    3
:F4:    4    BUBBLE
:F5:    5    BUBBLE

A0>DIR
DIRECTORY OF :FO:LEARN.PDS
NAME    .EXT   BLKS  LENGTH  ATTR    NAME    .EXT   BLKS  LENGTH  ATTR
ISIS    .MAP      4     512  S       ASM80            60   14594  S
ASXREF           20    4294  S       ASSIGN           16    3073  S
ATTACH            4     522  S       ATTRIB           24    4999  S
COPY             36    8366  S       CREDIT           80   19470  S
CREDIT  .MAC      4       7  S       DEBUG            12    2502  S
DELETE           20    4699  S       DETACH            4     434  S
DIR              28    6625  S       HELP             16    3771  S
HEXOBJ           20    4344  S       IDISK            32    7035  S
IXREF            44   10216  S       LIB              44   10227  S
LINK             56   13074  S       LINK    .OVL     20    4578  S
LOCATE           60   15021  S       OBJHEX           16    3347  S
```

Key-in Sequence **Comments**

Press P to switch back to Page mode and display the file a page at a time.



```
:F5:    5    BUBBLE

A0>DIR
DIRECTORY OF :FO:LEARN.PDS
NAME    .EXT   BLKS  LENGTH  ATTR    NAME    .EXT   BLKS  LENGTH  ATTR
ISIS    .MAP      4     512  S       ASM80            60   14594  S
ASXREF           20    4294  S       ASSIGN           16    3073  S
ATTACH            4     522  S       ATTRIB           24    4999  S
COPY             36    8366  S       CREDIT           80   19470  S
CREDIT  .MAC      4       7  S       DEBUG            12    2502  S
DELETE           20    4699  S       DETACH            4     434  S
DIR              28    6625  S       HELP             16    3771  S
HEXOBJ           20    4344  S       IDISK            32    7035  S
IXREF             4
```

Key-in Sequence **Comments**

S

S

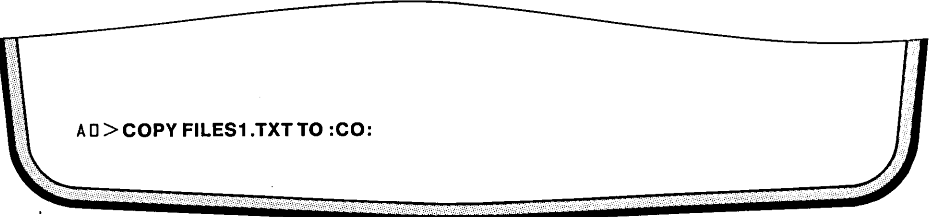
Press S to enter a Slow scroll mode. This causes the file to be displayed slowly without stopping at the end of the page. Press the CTRL-S to stop the display at any time.

```
   :F5:   5   BUBBLE

AO>DIR
DIRECTORY OF :FO:LEARN.PDS
NAME     .EXT  BLKS  LENGTH  ATTR   NAME     .EXT  BLKS  LENGTH  ATTR
ISIS     .MAP     4     512  S      ASM80            60   14594  S
ASXREF           20    4294  S      ASSIGN           16    3073  S
ATTACH            4     522  S      ATTRIB           24    4999  S
COPY             36    8366  S      CREDIT           80   19470  S
CREDIT   .MAC     4       7  S      DEBUG            12    2502  S
DELETE           20    4699  S      DETACH            4     434  S
DIR              28    6625  S      HELP             16    3771  S
HEXOBJ           20    4344  S      IDISK            32    7035  S
IXREF            44   10216  S      LIB              44   10227  S
LINK             56   13074  S      LINK     .OVL    20    4578  S
LOCATE           60   15021  S      OBJHEX           16    3347  S
RENAME           12    2557  S      SERIAL           16    3148  S
SUBMIT           20    4692  S      SYSPDS   .LIB    16    3101  S
FILES    .TXT     4       0
                                688
1420 FREE / 2544 TOTAL BLOCKS
AO>#
AO>
```

Key-in Sequence	Comments
	Press any key (the RETURN key is used here) to continue the slow scroll display.
	Press the F key to speed up the display. This causes the file to be displayed in a Fast scroll mode until the end of the file.

```
ASXREF          20    4294  S     ASSIGN           16    3073  S
ATTACH           4     522  S     ATTRIB           24    4999  S
COPY            36    8366  S     CREDIT           80   19470  S
CREDIT  .MAC     4       7  S     DEBUG            12    2502  S
DELETE          20    4699  S     DETACH            4     434  S
DIR             28    6625  S     HELP             16    3771  S
HEXOBJ          20    4344  S     IDISK            32    7035  S
IXREF           44   10216  S     LIB              44   10227  S
LINK            56   13074  S     LINK     .OVL    20    4578  S
LOCATE          60   15021  S     OBJHEX           16    3347  S
RENAME          12    2557  S     SERIAL           16    3148  S
SUBMIT          20    4692  S     SYSPDS   .LIB    16    3101  S
FILES   .TXT     4       0
                                688
1420 FREE / 2544 TOTAL BLOCKS
A0>#

A0>
```

Key-in Sequence	Comments
	Press Z to display the last 1024 characters (1K bytes) of the file.
	Press E to end the display and return to the operating system.



Key-in Sequence	Comments
COPY FILES1.TXT TO :CO:	The COPY command can also be used to display the contents of a file on the CRT screen. However, the display cannot be controlled as with the @ command. The specification :CO: is the pathname of the console output device. This command copies the file to the device currently assigned to receive system output. Note that displaying a non-ASCII file can have unpredictable results.

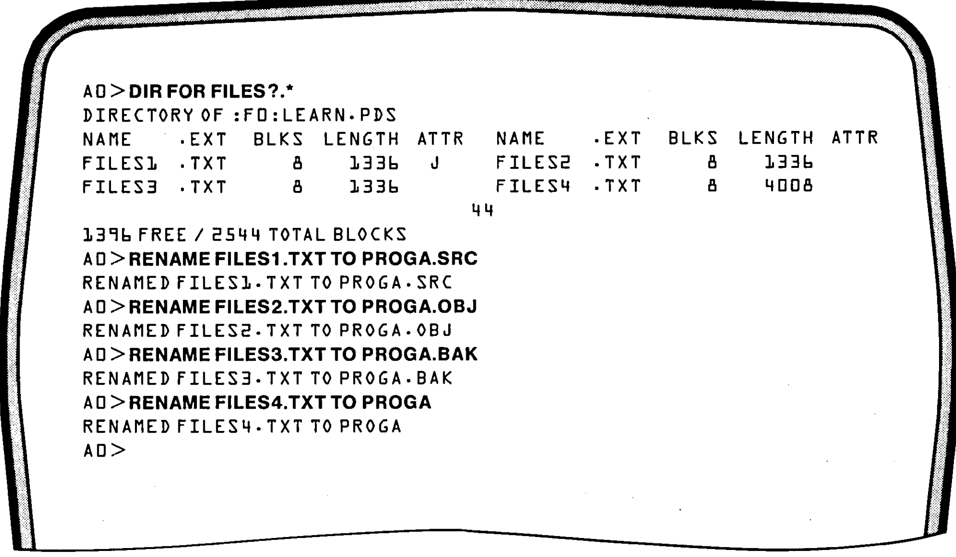
```
  :F4:   4    BUBBLE
  :F5:   5    BUBBLE

A0>DIR
DIRECTORY OF :F0:LEARN.PDS
NAME     .EXT  BLKS  LENGTH  ATTR    NAME     .EXT  BLKS  LENGTH  ATTR
ISIS     .MAP     4     512  S       ASM80             60   14594  S
ASXREF           20    4294  S       ASSIGN            16    3073  S
ATTACH            4     522  S       ATTRIB            24    4999  S
COPY             36    8366  S       CREDIT            80   19470  S
CREDIT   .MAC     4       7  S       DEBUG             12    2502  S
DELETE           20    4699  S       DETACH             4     434  S
DIR              28    6625  S       HELP              16    3771  S
HEXOBJ           20    4344  S       IDISK             32    7035  S
IXREF            44   10216  S       LIB               44   10227  S
LINK             56   13074  S       LINK     .OVL     20    4578  S
LOCATE           60   15021  S       OBJHEX            16    3347  S
RENAME           12    2557  S       SERIAL            16    3148  S
SUBMIT           20    4692  S       SYSPDS   .LIB     16    3101  S
FILES    .TXT     4       0
                                 688
1420 FREE / 2544 TOTAL BLOCKS
A0>#
COPIED :F0:FILES1.TXT TO :C0:
A0>
```

Comments

This screen shows the results from entering the previous command.

Using Wildcard Characters

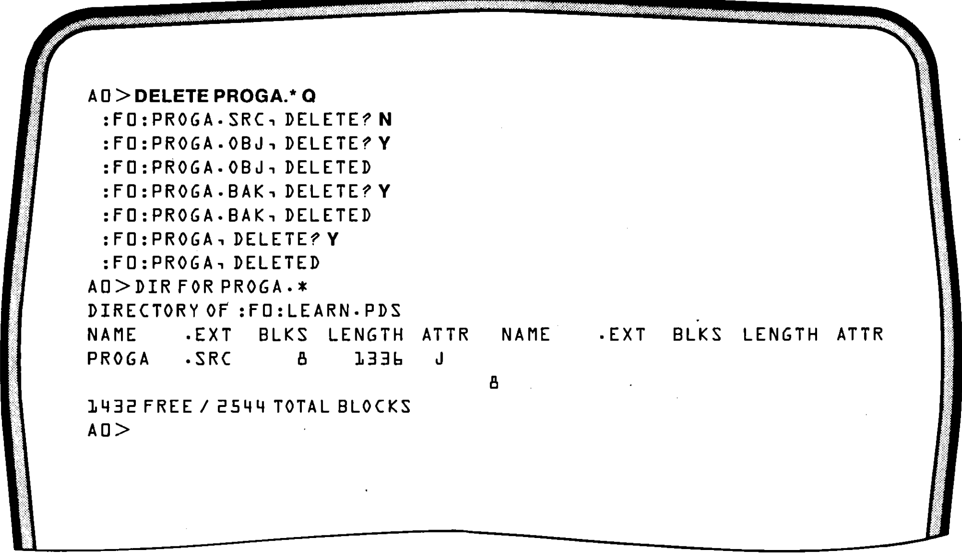
Wildcard characters are placeholders that can be used in filenames to hold the place of the actual filename character. For example, the * wildcard character takes the place of any number of characters in a filename. The following examples show the use of wildcard characters.

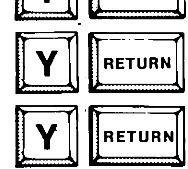


Key-in Sequence	Comments
DIR FOR FILES?.*	This example shows the use of the wildcard characters: ? and *. The ? character can be given as part of a filename. It substitutes for any single character in the position where it appears. The * character can be given as part of a filename and substitutes for any number of characters starting at the position where it appears. Thus, the DIR command in this example is for a directory of any file beginning with the characters F I L E S, with any character in the sixth position and with any extension. The result is shown in the screen.
RENAME FILES1.TXT TO PROGA.SRC. RETURN	The file, FILES1.TXT is renamed to PROGA.SRC
RENAME FILES2.TXT TO PROGA.OBJ RETURN	FILES2.TXT is renamed to PROGA.OBJ.
RENAME FILES3.TXT TO PROGA.BAK RETURN	FILES3.TXT is renamed to PROGA.BAK.
RENAME FILES4.TXT TO PROGA RETURN	FILES4.TXT is renamed to PROGA.

```
A0>DIR FOR FILES?.*
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT   BLKS  LENGTH  ATTR   NAME     .EXT   BLKS  LENGTH  ATTR
                                                 0
1396 FREE / 2544 TOTAL BLOCKS
A0>DIR FOR PROGA.*
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT   BLKS  LENGTH  ATTR   NAME     .EXT   BLKS  LENGTH  ATTR
PROGA   .SRC      8    1336    J    PROGA    .OBJ      8    1336
PROGA   .BAK      8    1336    J    PROGA              8    4008
                                         44
1396 FREE / 2544 TOTAL BLOCKS
A0>ATTRIB PROGA.* J1
      FILE        CURRENT ATTRIBUTES
  :F0:PROGA.SRC           J
  :F0:PROGA.OBJ           J
  :F0:PROGA.BAK           J
  :F0:PROGA               J
A0>COPY *.* TO :F1:*.* J
COPIED :F0:PROGA.SRC TO :F1:PROGA.SRC
COPIED :F0:PROGA.OBJ TO :F1:PROGA.OBJ
COPIED :F0:PROGA.BAK TO :F1:PROGA.BAK
COPIED :F0:PROGA TO :F1:PROGA
A0>
```

Key-in Sequence	Comments
DIR FOR FILES?.*	Verify that all the files are renamed.
DIR FOR PROGA.*	The results of this command show that all the files are renamed.
ATTRIB PROGA.* J1	Backing up files means saving a copy of the files on a second disk. Saving back-ups of important files is a good practice to adopt to avoid losses. The first step in backing up a group of files is to assign a unique attribute to all the files to be backed up. For this example, the ATTRIB command is used to assign the J attribute to the files to be backed up. The ATTRIB command accepts wildcard filenames.
COPY *.* TO :F1:*.*J	The second step in backing up a group of files is to copy the files with the unique attribute. This example assumes a multiple drive system with a formatted disk in drive 1. Use the non-system disk created in a previous example. Also shown is the use of wildcard filenames with the COPY command. The unique attribute limits the scope of the wildcard file specification. Here, all files with the J attribute set are copied.



Key-in Sequence	Comments
DELETE PROGA.* Q RETURN	This example shows the use of wildcard characters to delete a group of files. The Q option is used so that the command prompts before deleting each file.
	Because N is entered in response to the first prompt, the file, PROGA.SRC, is not deleted. Do not delete this file. It is used in a later example.
	Delete the other files specified.
DIR FOR PROGA.*	The DIR command for PROGA.* verifies that all the files except PROGA.SRC were deleted.

File Operations With a Single Drive System

Special considerations must be made to handle files efficiently on a single drive system. The following examples illustrate the techniques of using a single drive system. Appendix E contains some additional techniques and procedures for use with single drive systems.

```
A0>COPY DIR TO DIR P
LOAD SOURCE DISKETTE, THEN TYPE (CR)
LOAD OUTPUT DISKETTE, THEN TYPE (CR)
COPIED :F0:DIR TO :F0:DIR
LOAD SYSTEM DISKETTE, THEN TYPE (CR)
A0>DIR FOR DIR P
LOAD SOURCE DISKETTE, THEN TYPE (CR)
DIRECTORY OF :F0:NONSYS.DSK
NAME    .EXT  BLKS  LENGTH  ATTR   NAME    .EXT  BLKS  LENGTH  ATTR
DIR            28    6625
                                 28
2476 FREE / 2544 TOTAL BLOCKS
LOAD SYSTEM DISKETTE, THEN TYPE (CR)
A0>
```

Key-in Sequence / **Comments**

COPY DIR TO DIR P

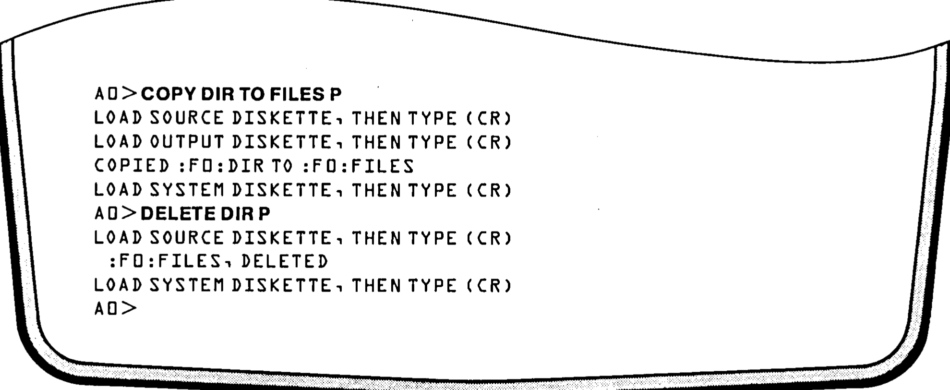
This screen is an example of copying to disk other that the system disk on single drive systems. The P option is used to pause while the disks are alternately removed and inserted.

Press RETURN to begin the copy, since, the source file is on the system disk. When the source file is not on the system disk, the system disk should be removed and source disk inserted before pressing the RETURN key.

Wait until the red LED on the drive is off and motor has stopped, indicating that the drive is not being accessed. Remove the source diskette and insert the output diskette. Press the RETURN key to continue the copy. Use the non-system diskette created in a previous example as the output diskette. The disk used for output must have been formatted with the IDISK command first.

(continued)

Key-in Sequence	Comments
	When the file is copied, a message is displayed and a prompt is issued to insert the system disk. Remove the output diskette and insert the system diskette when the disk drive motor is off. Press RETURN to end the copy and return to the operating system. When copying groups of files, there may be several alternations of loading the source and output diskette before inserting the system diskette and completing the copy. Be careful to insert the correct diskette at each prompt to ensure the correct copying of data.
DIR FOR DIR P	This is how to display a directory for a disk other than the current system disk. The P option causes the system to pause to insert and remove diskettes.
	When the prompt appears, and the disk drive motor is off, remove the system diskette and insert the diskette for which a directory is required. Use the non-system diskette from the previous example.
	A prompt is given after the directory is displayed. Remove the non-system diskette and insert the system diskette. Press the RETURN key to return to the operating system prompt.



Key-in Sequence — **Comments**

COPY DIR TO FILES P RETURN

Rename a file named DIR to a file named FILES on a non-system disk in a single drive system. The RENAME command does not have the P option and can only be used to rename files on the system disk. Use the COPY command followed by the DELETE command to rename files that are not on the system disk.

Remove the system disk and insert the disk from which the file is to be deleted. Press RETURN to delete the file.

Remove the system disk and insert the disk with the file to be renamed. Press the RETURN.

When the prompt appears to load the output diskette, do not remove the disk. Press RETURN and the file is duplicated with the new name specified.

DELETE DIR P

Delete the file DIR.

Remove the system disk and insert the disk from which the file is to be deleted. Insert the non-system disk and press RETURN to delete the file.

A message is displayed when the file is deleted and a prompt is given to load the system disk. Insert the system disk. Press RETURN to return to the operating system. Run the DIR command to verify the deletion. The command line is DIR FOR DIR P.

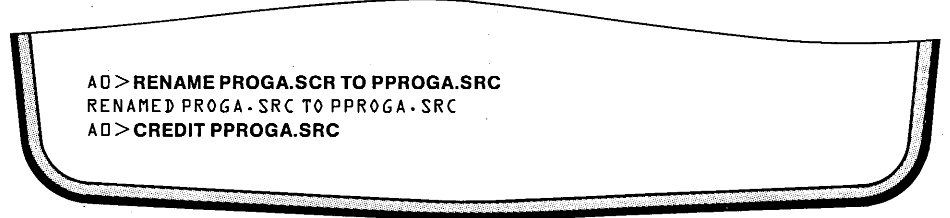
Text Editing Commands

The operating system provides a text editor to create and modify text files interactively or through a command file. A text file can be a source program to be used as input to an assembler or a language translator, or it can be a text document like a letter or a manual. The text editor supplied with the operating system is:

CREDIT interactively creates and modifies text files. See chapter 6 for a complete description of the macro editing file CMACRO provided on the iPDS system diskette. Chapter 6 also explains the primary differences between text editing with the iPDS system and text editing on other Intel development systems. See the *ISIS CREDIT™ CRT-Based Text Editor User's Guide,* for complete details on text editing.

Editing Text Files

In the first series of editing examples, a file created in a previous example is edited after first being renamed. A source program used in a later example is entered into this file. There must be space on the disk for two times the size of the file being edited.



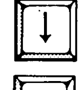
Key-in Sequence	Comments
RENAME PROGA.SRC TO PPROGA.SRC RETURN	The file named PROGA.SCRC is renamed to PPROGA.SRC This file is used in the next set of examples.
CREDIT PPROGA.SRC RETURN	The next set of examples illustrate some simple editing techniques for use with the CREDIT text editor.

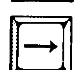
```
ISIS-II CRT-BASED EDITOR V2.1
OLD FILE  SIZE=1336 CHARACTERS

 -----
@
 LOGICAL PHYSICAL↑

↑
  :CI:     :VI:↑
  :CO:     :F0:FILES.TXT↑
↑
  :F0:     0↑
  :F1:     1↑
  :F2:     2↑
  :F3:     3↑
  :F4:     4   BUBBLE↑
  :F5:     5   BUBBLE↑
↑
A0>DIR↑
DIRECTORY OF :F0:LEARN.PDS↑
NAME    .EXT  BLKS  LENGTH  ATTR    NAME    .EXT  BLKS  LENGTH   ATTR↑
ISIS    .MAP     4     512  S       ASM80           60   14594  S    ↑
ASXREF          20    4294  S       ASSIGN          16    3073  S    ↑
ATTACH           4     522  S       ATTRIB          24    4999  S    ↑
COPY            36    8366  S       CREDIT          80   19470  S    @
```

Key-in Sequence | **Comments**

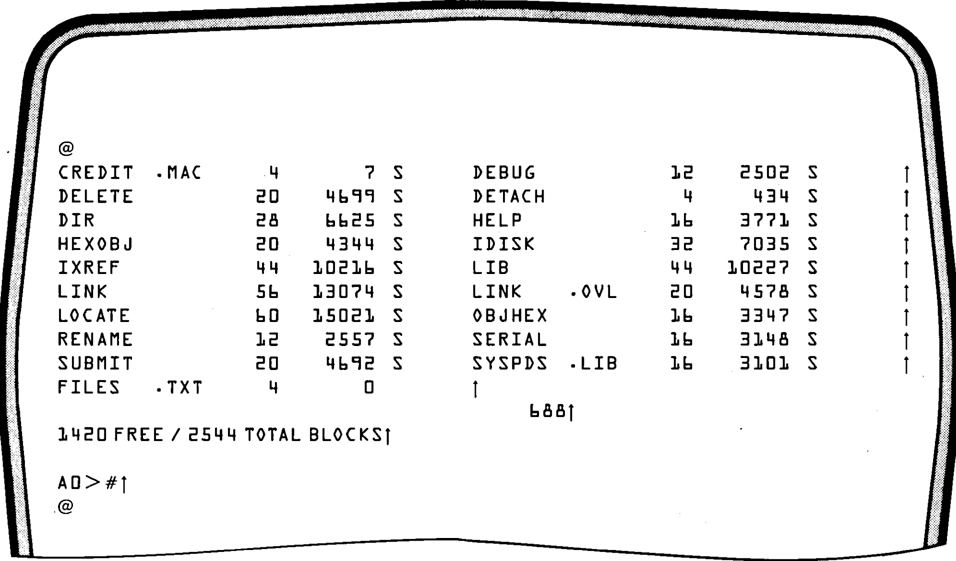
When the CREDIT command line is entered, the screen is cleared and the data from the file to be edited is displayed. The file used was created in a previous example and contains the system output of an ASSIGN command and a DIR command from that example. The up arrow character at the end of each line is the end of line character.

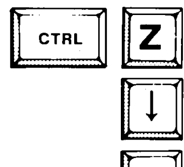
The CTRL-Z function is used to delete text. The first CTRL-Z typed causes an @ character to appear in the position where the cursor was. This @ marks the beginning of the text to be deleted.





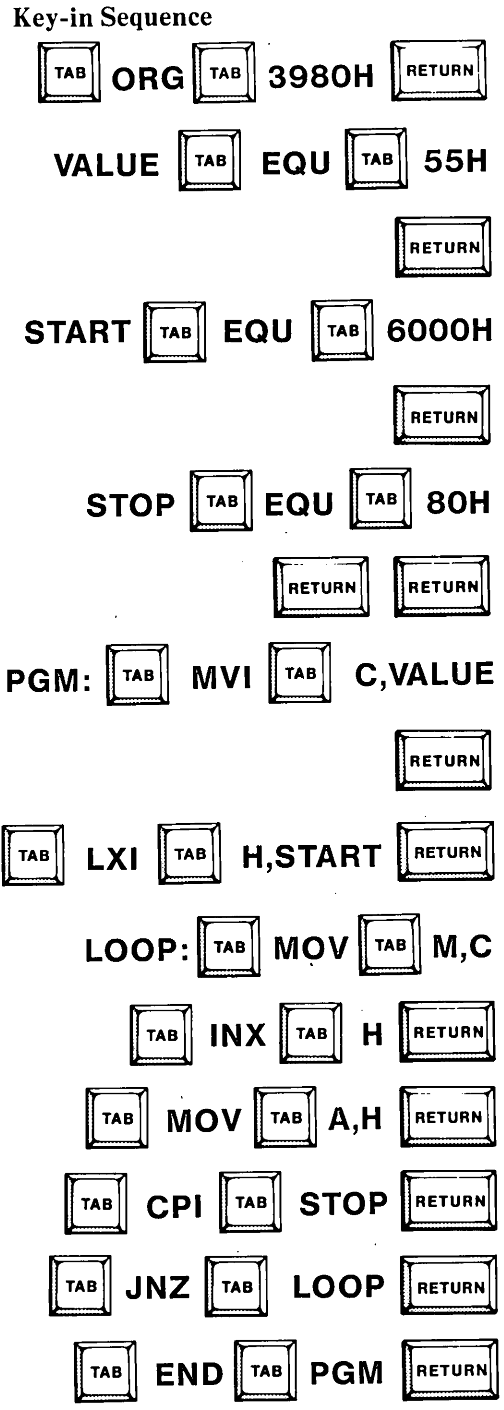
Hold the down arrow key. The first @ remains at the top of the screen. A second @ appears and moves down the screen a line each time the down arrow is pressed. Hold down the → key to move the @ to the right side of the screen so it lines up over the last up arrow on the the screen. The second @ marks the end of the text to be deleted. Press CTRL-Z again and the text appearing on the screen between the two @ markers is deleted from the file. The remainder of the file now appears on the screen taking the place of the deleted text. The vertical bar at the bottom of the screen is the end of file character.



Key-in Sequence	Comments
	To delete this text, enter a CTRL-Z and use the cursor control keys to move the second marker to the end of file character. The cursor keys are the keys labeled with arrows on the right side of the keyboard. Each key moves the cursor in the direction of the arrow on the key cap.
	Then, type CTRL-Z again to complete the deletion.

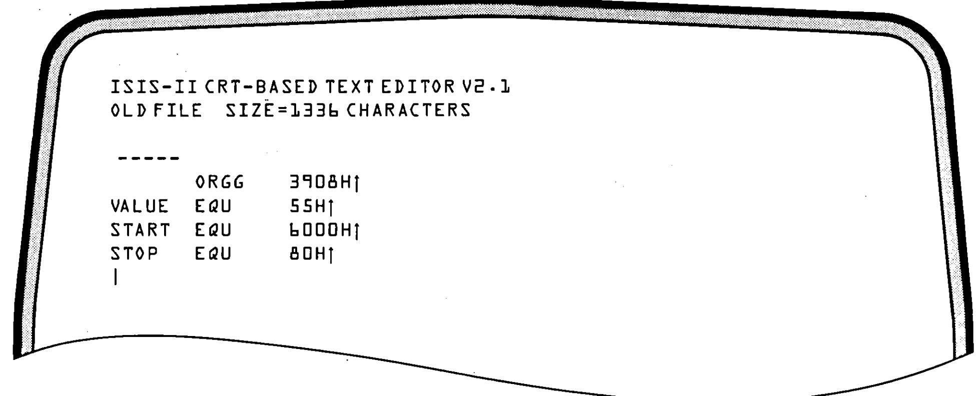
```
ISIS-II CRT-BASED TEXT EDITOR V2.1
OLD FILE  SIZE=1336 CHARACTERS

 -----
        ORG     3980H↑
VALUE   EQU     55H↑
START   EQU     6000H↑
STOP    EQU     80H↑
↑
PGM:    MVI     C,VALUE↑
        LXI     H,START↑
LOOP:   MOV     M,C↑
        INX     H↑
        MOV     A,H↑
        CPI     STOP↑
        JNZ     LOOP↑
        END     PGM↑
```

Key-in Sequence

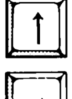


Comments

The part of screen under the dashed line is called the text area. The the text area is used to display the contents of the files as text is entered,, Only the vertical bar, end of file character, appears in the text area now. Enter the data as shown. As each character is typed, the end of file marker is moved one character to the right. The TAB key moves the cursor eight spaces the tight. The following examples show how to correct errors that might occur in typing this data.

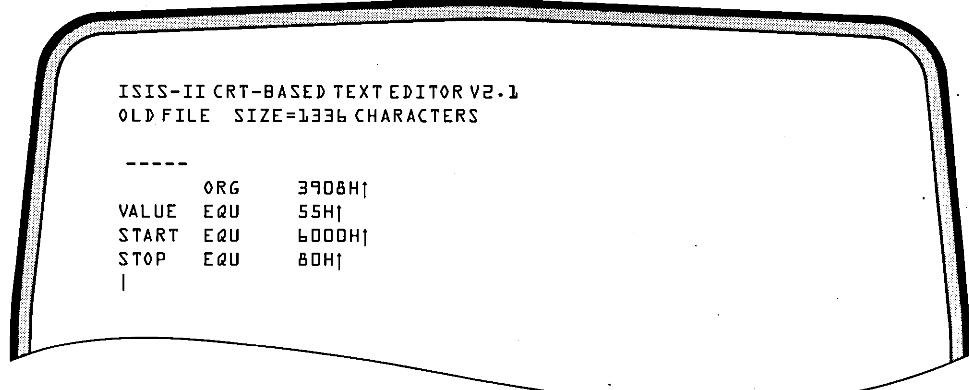


Key-in Sequence | **Comments**



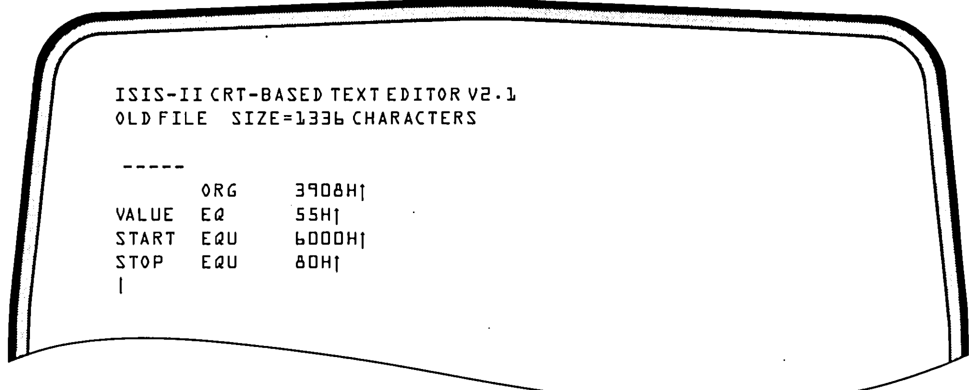
→

The extra G in ORG was not noticed until several lines had been typed. Use the cursor control keys to move the cursor to the extra G.

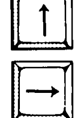


Key-in Sequence | **Comments**

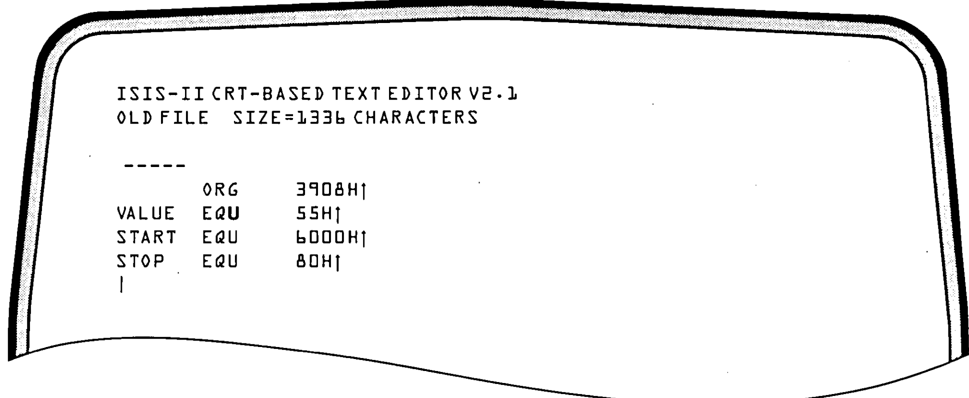
Enter CTRL-D to delete the extra character. The CTRL-D function deletes the single character at the position of the cursor.



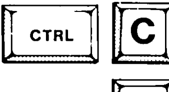
Key-in Sequence **Comments**



The U on the EQU on the second line was left off. Use the cursor control keys to move the cursor to the space following the Q.



Key-in Sequence **Comments**

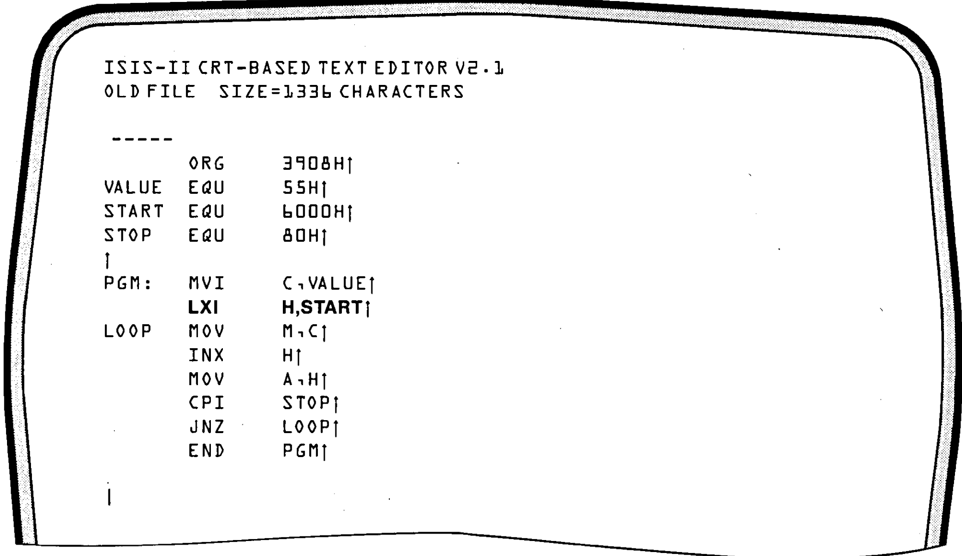


To insert a single character, enter CTRL-C and, enter the character to be inserted.

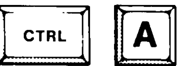
```
ISIS-II CRT-BASED TEXT EDITOR V2.1
OLD FILE  SIZE=1336 CHARACTERS

-----
        ORG     3908H↑
VALUE   EQU     55H↑
START   EQU     6000H↑
STOP    EQU     80H↑
↑
PGM:    MVI     C,VALUE↑
LOOP    MOV     M,C↑
        INX     H↑
        MOV     A,H↑
        CPI     STOP↑
        JNZ     LOOP↑
        END     PGM↑
|
```

Key-in Sequence **Comments**

[↑] The line LXI H,START was omitted. Use the cursor control keys to move the cursor to the L of LOOP.

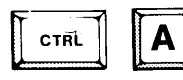


Key-in Sequence **Comments**



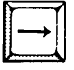
Enter CTRL-A to begin an insert of more than one character. The first CTRL-A causes the screen to clear at the point where the insert is to be made.

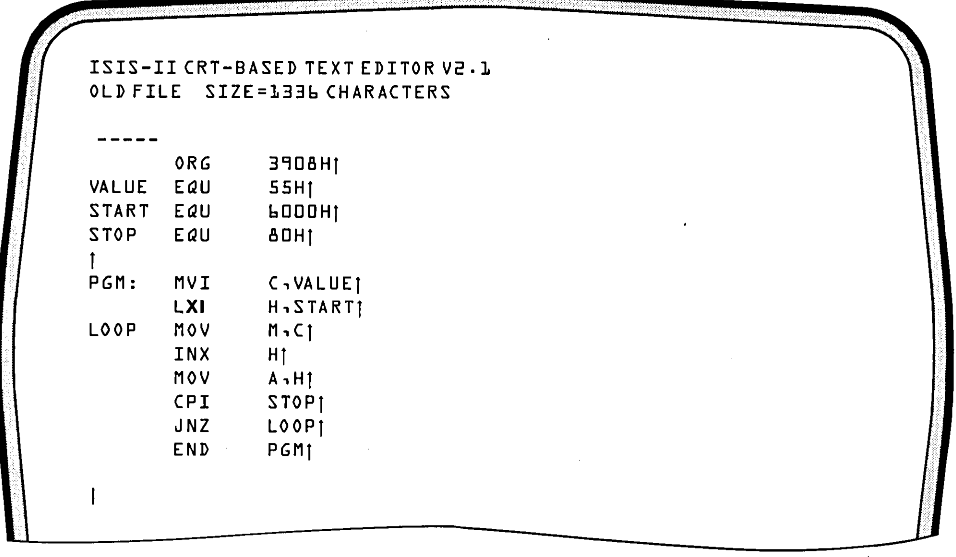
[RETURN]



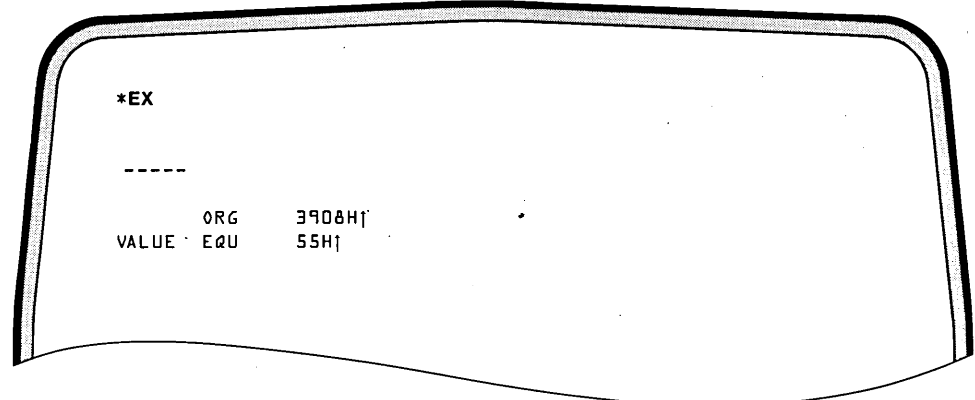
Type the line that was omitted. It appears on the screen as it is typed. Enter CTRL-A again to end the insert.

```
ISIS-II CRT-BASED TEXT EDITOR V2.1
OLD FILE  SIZE=1336 CHARACTERS

 -----
           ORG      3908H↑
VALUE      EQU      55H↑
START      EQU      6000H↑
STOP       EQU      80H↑
↑
PGM:       MVI      C,VALUE↑
           LIX      H,START↑
LOOP       MOV      M,C↑
           INX      H↑
           MOV      A,H↑
           CPI      STOP↑
           JNZ      LOOP↑
           END      PGM↑

|
```

Key-in Sequence	Comments
↑ 	Here, LIX was typed instead of LXI. Use the cursor control keys to move the cursor to the I of LIX.



Key-in Sequence	Comments
XI	Type XI to replace the characters I and X.

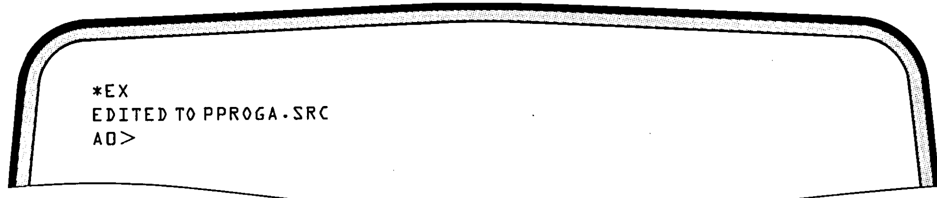


Key-in Sequence **Comments**

Press the HOME key to enter the command line mode of the CREDIT text editor. All the examples until now have illustrated the screen mode of editing. The screen mode allows interactive editing of text. The command line mode allows commands to be entered to modify text. In command mode, a prompt is displayed at the top of the screen and commands are entered there. This area of the screen is called the command area. More details on the command line mode can be found in Chapter 6 and the *ISIS Credit CRT-Based Text Editor User's Guide.*

EX

The EX command exits from the CREDIT text editor back to the operating system. The file containing the edited text is updated to contain the new data.



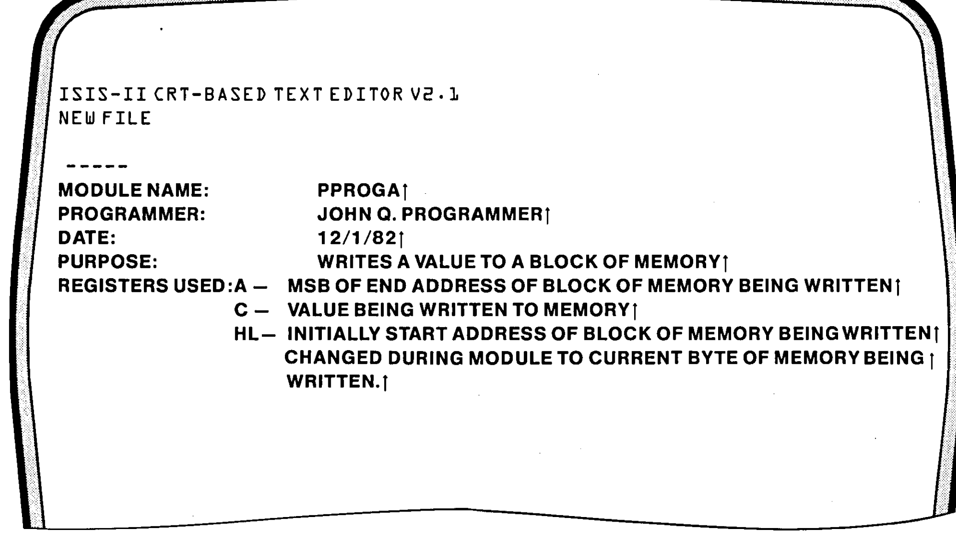
Comments

This screen shows the results of the EX command. The screen is cleared and the operating system prompt is displayed.

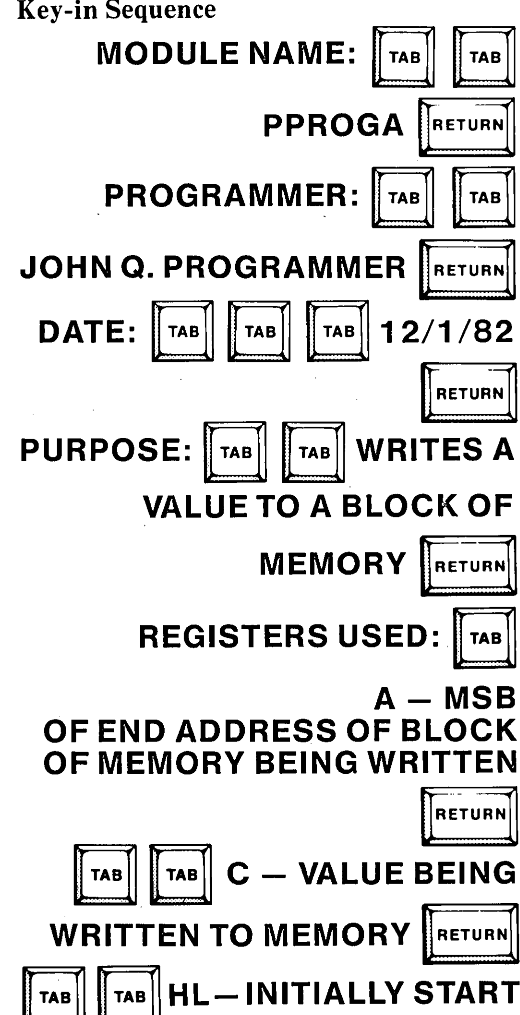
Creating a Source Program

In this example, a new file is created with the editor and text is input.

Key-in Sequence	Comments
CREDIT DPROGA.DOC	A new file is created named DPROGA.DOC, and documentation describing the program is entered.



Key-in Sequence



Comments

The screen is cleared and the text area is blank. The sign-on message appears in the command area. Enter the data as shown using any editing commands necessary to correct errors.

Key-in Sequence **Comments**

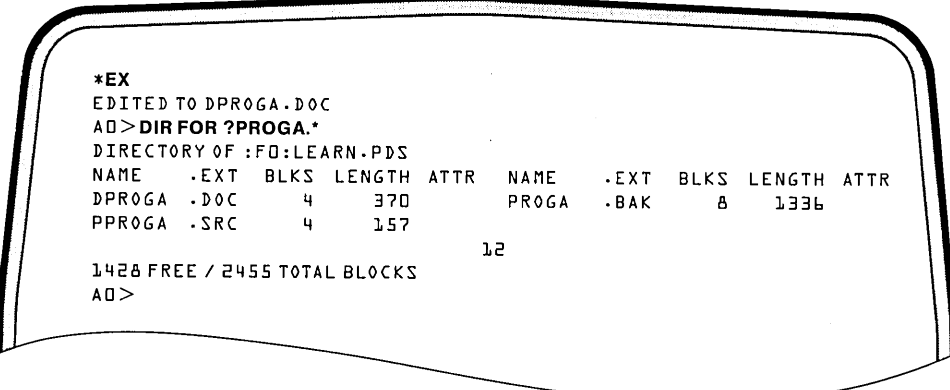
ADDRESS OF BLOCK OF MEMORY BEING WRITTEN

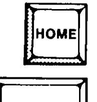
SPACE SPACE SPACE

SPACE SPACE SPACE **CHANGED DURING MODULE TO CURRENT BYTE OF**

MEMORY BEING

WRITTEN.



Key-in Sequence	Comments
 **EX**	Press the HOME key to enter the command line mode, and enter the EX command to return to the operating system.
DIR FOR ?PROGA.*	The DIR command is used to confirm the existence of the two files just edited. More information on the use of the CREDIT text editor can be found in the *ISIS-II CREDIT™ CRT-Based Text Editor User's Guide,* order number 9800902

Program Development Commands

The two main aspects of developing a computer-based product is the program or software development and hardware development. The operating system offers a number of languages for developing programs. In addition to language translators, the following commands are provided to aid in software development. These commands are described in the manuals indicated.

LIB allows the user to manage a library of MCS-80/85 program modules. With the LIB command, libraries can be created, modules can be added or deleted from them, and a list of the modules in a library can be displayed. *MCS™-80/85 Utilities User's Guide,* order number 121617.

LINK combines a number of object modules into a single object module in an output file. *MCS™-80/85 Utilities User's Guide,* order number 121617.

LOCATE converts relocatable object programs into absolute object programs by supplying memory addresses throughout the program. *MCS™-80/85 Utilities User's Guide, order number 121617.*

HEXOBJ converts a program from hexadecimal file format to absolute object format. *MCS™-80/85 Utilities User's Guide,* order number 121617.

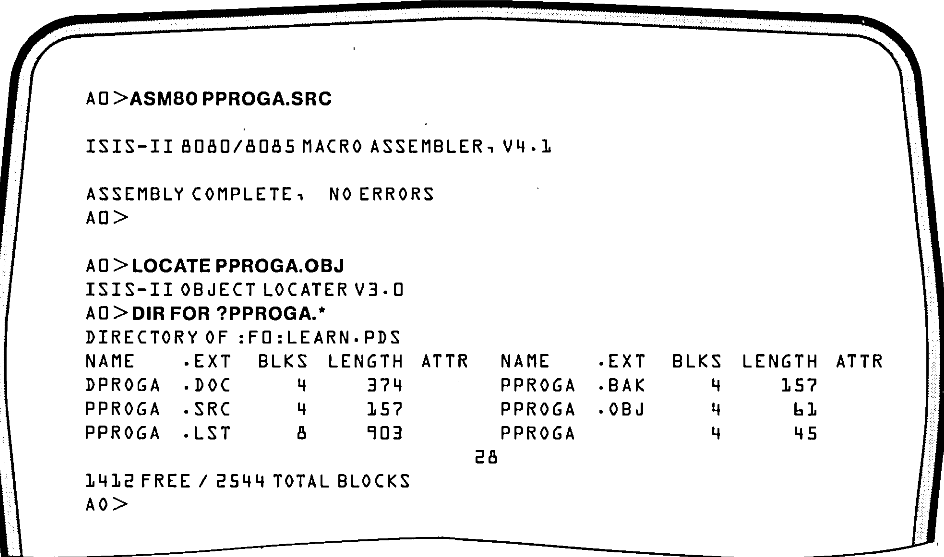
OBJHEX converts a program from absolute object format to hexadecimal file format. *MCS™-80/85 Utilities User's Guide,* order number 121617.

DEBUG provides a minimum set of debugging commands. Chapter 7.

IPPS is used with the PROM Personality Adapters to control the programming of EPROMs and E^2PROMs. Chapter 10 and the *iUP-200/201 Universal Programmer User's Guide,* order number 162613.

Creating an Object File

The next two example uses the program entered into the file created previously by the CREDIT text editor.



Key-in Sequence | **Comments**

ASM80 PPROGA.SRC

The next examples illustrate a simple case of program development on the iPDS. Assemble the previously entered program PPROGA.SRC. The program PPROGA fills a block of memory with a constant. The output file created by the assembler is named PPROGA.OBJ. This file must be processed by the LOCATE command before it can be run. A list file is also created by the assembler under the name PPROGA.LST.

LOCATE PPROGA.OBJ

The program does not have any external references, so it can be located without being linked first. The LOCATE utility assigns absolute memory addresses, where needed, in the program. The program is located in memory starting at address 3980H as determined by the ORG statement in the program. The output file created by the LOCATE utility is a file named PPROGA with no extension. This file can be loaded into memory and executed.

DIR FOR ?PROGA.*

The DIR command shows the files created in the development of the program PPROGA. The file PPROGA.LST was created by the assembler and contains a listing of the program. It can be displayed with the @ command.

Debugging a Program

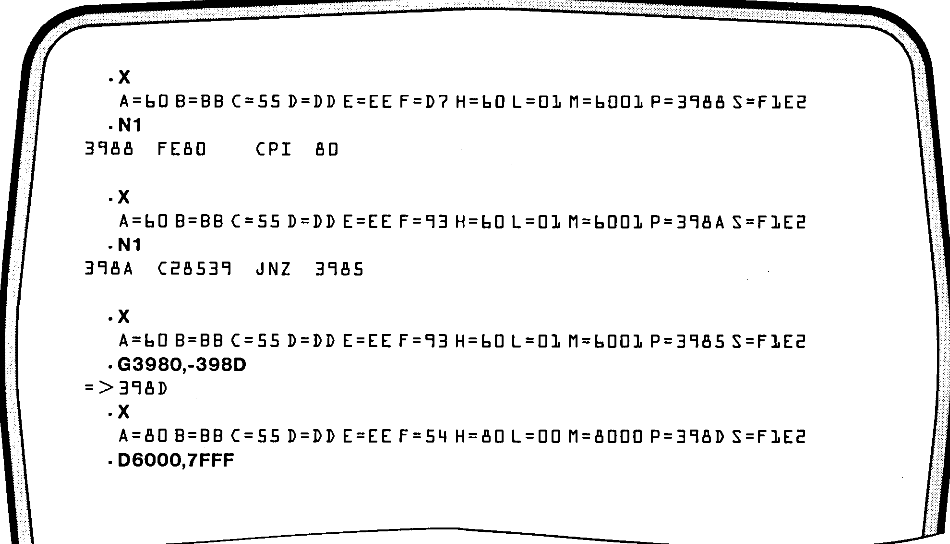
In the next series of examples, the object file created in the previous example is debugged using the DEBUG command.

```
A0>DEBUG PPROGA

PDS DEBUGGER V.10
=>3980
.C3980,9
3980  0E55     MVI  C,55
3982  210060   LXI  H,6000
3985  71       MOV  M,C
3986  23       INX  H
3987  7C       MOV  A,H
3988  FE80     CPI  80
398A  C28539   JNZ  3985
398D  38       DB   38
398E  3A8047   LDA  4780

.X
A=AA B=BB C=CC D=DD E=EE F=FF H=12 L=34 M=1234 P=3980 S=F1E2
.N1
3980  0E55     MVI  C,55

.
```

Key-in Sequence	Comments
DEBUG PPROGA	The DEBUG command loads a program into memory and allows the program to be run under controlled conditions. The program just assembled and located is run. The DEBUG program displays a sign on message and then displays the starting address of the program being debugged. A DEBUG command can be entered whenever the DEBUG prompt, the period (.), appears.
C3980,9	The C command disassembles instructions starting at the address specified for a count of the number of instructions specified. Here, the starting address is 3980H and the number of instructions to be disassembled is 9. The first column contains the address, the second column shows the hexadecimal value of the instruction starting at that address, and third column gives the mnemonic for the instruction starting at the address.
X	The X command displays the 8085 registers. All values are hexadecimal. The program counter, P, contains the address of the first instruction to be executed in the program.
N1	The N command executes the specified number of instructions starting at the address currently in the program counter and then halts program execution. The address, the hexadecimal opcode and operands, and the instruction mnemonics are displayed for the instructions executed. The N command can be used to single step through a program, executing only one instruction at a time. This technique for debugging is illustrated here.

```
.X
A=AA B=BB C=55 D=DD E=EE F=F7 H=12 L=34 M=1234 P=3982 S=F1E2
 .N1
3982  210060  LXI  H,6000

 .X
 .A=AA B=BB C=55 D=DD E=EE F=F7 H=60 L=00 M=6000 P=3985 S=F1E2
 .N1
3985  71        MOV  M,C

 .D6000,6000
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
6000  55_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ U ______________
 .N1
3986  23        INX  H

 .X
A=AA B=BB C=55 D=DD E=EE F=D7 H=60 L=01 M=6001 P=3987 S=F1E2
 .N1
3987  7C        MOV  A,H
.
```

Key-in Sequence	Comments
X	The value in the C register has changed to 55H, and program counter P has changed to 3982H.
N1	One more instruction is executed, and the program is halted. The address 6000H is the start address of the block of memory to be filled with the constant 55H.
X	The registers are displayed again.The program counter contains the value 3985H, the address of the next instruction in the program.
N1	One more instruction is executed. The value in the C register is moved to memory, currently 6000H.
D6000,6000	The D command displays the contents of the memory locations requested. The underlines appear because only one location was requested. They act as placeholders for the adjacent 15 locations that would appear on the same line if requested. The right end of the display contains the ASCII character corresponding to the value in the memory location. The underline also appears if there is no displayable ASCII character for the value at the specified memory location.
N1 RETURN	Another instruction which increments the HL register pair is executed.
X	The registers are displayed.
N1 RETURN	This instruction checks the memory address so that the loop can be ended when the end of the block of memory being filled is reached.



Key-in Sequence **Comments**

X

The registers are displayed showing that the A register also contains the value 60H.

N1

This instruction compares the value in the A register with 80H. This value is the MSB of the end of the block of memory being filled. The loop no longer executes when the program reaches 80H.

X

The flag register, shown as F, now contains 93H. The zero flag is the sixth bit in the register and is set to a value of 0.

N1

Another instruction, which jumps to address 3985H if the zero flag is off, is executed. Since the zero flag was reset in the previous compare instruction, the jump should take place.

X

When the registers are displayed, the program counter contains the address 3985H for the next instruction to be executed. This is the address of the beginning of the loop. The loop is executed again.

G3980,398D

The G command executes a program starting at the instruction specified. A breakpoint is set at address 398DH. The program halts when the entire block of memory is filled and the loop no longer repeats.

X

The registers are displayed showing that register A contains the value 80H, the end value for the loop.

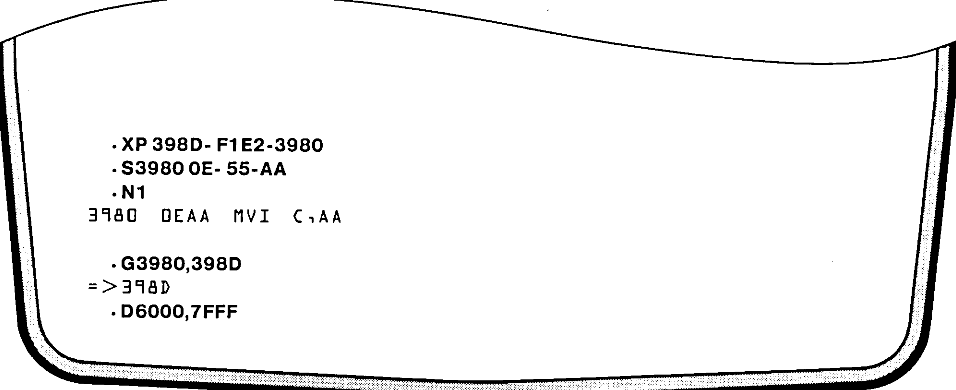
D6000,7FFF

The block of memory from 6000H to 7FFFH is displayed.

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
6000 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
6010 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
6020 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
6030 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
6040 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
6050 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
                                  .
                                  .
                                  .
                                  .
7F80 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7F90 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FA0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FB0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FC0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FD0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FE0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
7FF0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
.
```

Comments

The memory contains the value 55H which is the ASCII code for the character U.



Key-in Sequence

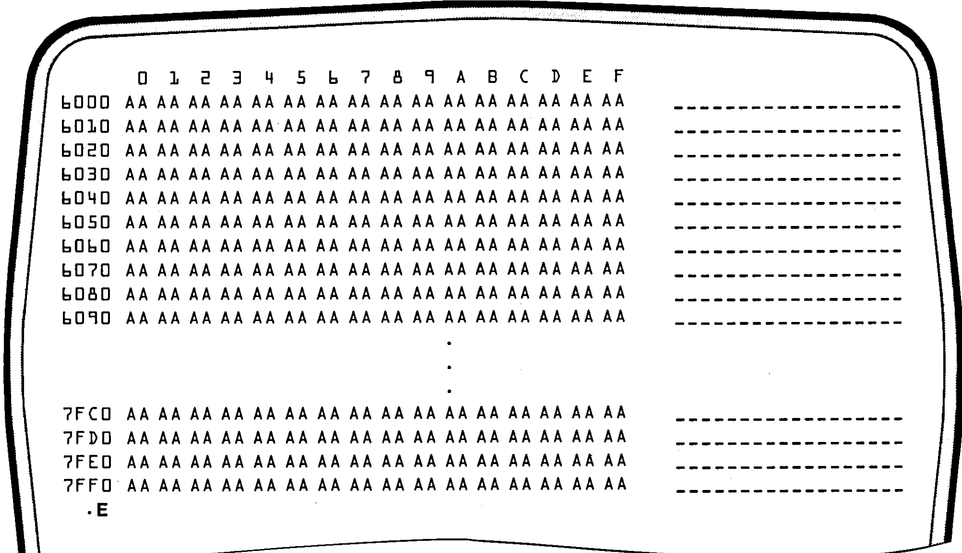
XP 3980

Comments

The X command can also be used to change the contents of registers. The XP form of the command displays the current value of the P register and allows that value to be changed. It is changed to 3980H, the starting address of the program. Thus, the next N command executes the program from its starting address.

(continued)

Key-in Sequence	Comments
S3980 *SPACE* **AA** *SPACE*	The constant value that is written to memory is changed from 55H to AAH. This is done with the S command which substitutes new values in memory. The memory location containing the constant is the operand of the first instruction. Since, the opcode is not changed, press the space bar to increment to the next address. Enter AA and the value is changed in memory. The constant is not changed in the file containing the program.
N1 RETURN	A single instruction is executed. Notice that the value moved into the C register is AAH this time.
G3980,398D RETURN	The G command begins executing the program at 3980H with a breakpoint set at 398H which is the address of the instruction following the loop.
D6000,7FFF RETURN	The block of memory from 6000H to 8000H is displayed again. It contains the value AAH.



Key-in Sequence	Comments
	The E command exits from DEBUG and returns to the operating system.

Program Execution Commands

Programs can be executed under the control of the operating system in two ways:

- Interactively by typing the command line for each program to be executed
- Automatically (not requiring operator intervention) through the SUBMIT or JOB capabilities of the operating system

SUBMIT files are implemented with the SUBMIT command and the . command. The . command differs from the SUBMIT command in that only one line from the SUBMIT file is read and executed and no intermediate file is created. Parameter substitution is allowed in both cases.

Jobfiles are implemented with the following features in the operating system: the JOB command, the ENDJOB command, the ASSIGN command, the / command, FUNCT-<n>, and the ABOOT.CSD and BBOOT.CSD files. A jobfile differs from a SUBMIT file in that no intermediate disk file is created and no parameter substitution is allowed.

The operating system offers the following features, described in Chapter 5, for executing commands.

<filename> loads and executes the object program named <filename>.

SUBMIT reads an input SUBMIT file, creates a command file containing ISIS commands, and executes commands in sequence from the file created.

. is a fast form of the SUBMIT command. One command line is read from the SUBMIT file, transformed into an ISIS command in memory, and executed. No intermediate file is created.

FUNCT-<n> is also considered a device management command. It reads command lines from a file named JOB<n>.CSD, where <n> is a single digit from 0 to 9. Pressing <n> followed by the RETURN key is the same as pressing FUNCT-<n>.

ASSIGN is also considered a device management command. However, a form of ASSIGN can be used to run commands from a file.

/ reads ISIS commands from a disk file and executes them in sequence. The / command is also considered a device management command.

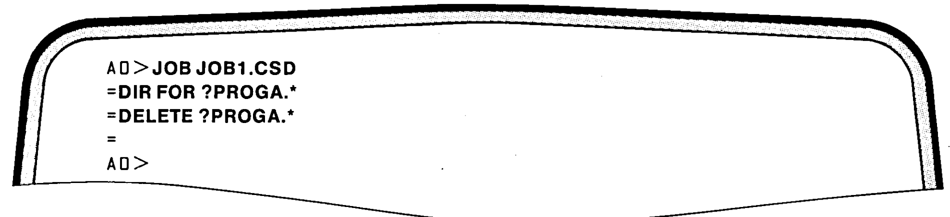
JOB stores a sequence of frequently used ISIS commands in a file as they are entered from the keyboard without executing them until the sequence is completely entered. Then, the commands can be executed in order, or they can be saved in the file and executed later with the / command. Two jobfiles, ABOOT.CSD and BBOOT.CSD, deserve special mention. If either of these files is present (ABOOT.CSD for Processor A and BBOOT.CSD for Processor B) when the system is initialized, commands are automatically executed from the file. This feature can be used to configure a system. An example is given later in this chapter.

ENDJOB stops the automatic execution of commands from a JOB file and returns control to the keyboard. The ENDJOB command is automatically inserted after a sequence of commands is entered under the control of the JOB command. If a jobfile is created with the text editor, the user must enter the ENDJOB command.

ESC edits the previously entered or the current command line and allows the new command line to be executed.

Using the JOB Command

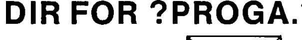
A jobfile is one form of running programs automatically. In the next example, a jobfile is created but not executed.



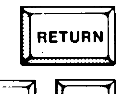
Key-in Sequence **Comments**

JOB JOB1.CSD

There are several ways to enter commands and run programs, other than under control of the operating system: editing a command line to run a different program (shown at the end of Chapter 3), running several programs with the JOB command and running several programs with the SUBMIT command. When the command line is entered the equal sign (=) prompt is displayed. A file is created with the name specified on the JOB command line. Here, the name is JOB1.CSD.



DELETE ?PROGA.*

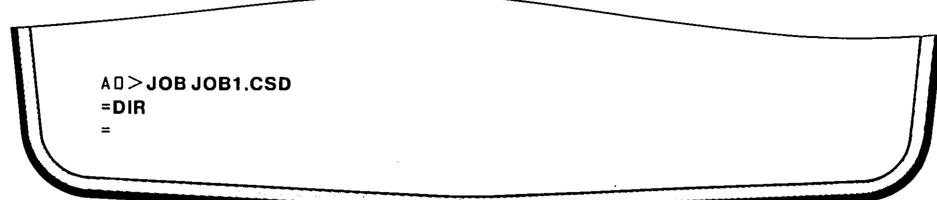


CTRL Z

Enter the command lines shown. These commands are not executed. Instead, the JOB prompt is displayed after each command to allow another command to be entered. There are three ways to exit from the JOB command: CTRL-Z, the RETURN key, and the ESC key. When a CTRL-Z is entered, the JOB is cancelled. The commands entered are not saved and the job file is not created. Control is returned to the operating system.

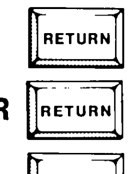
Automatic Job Execution

This section of examples shows several techniques for running jobfiles.

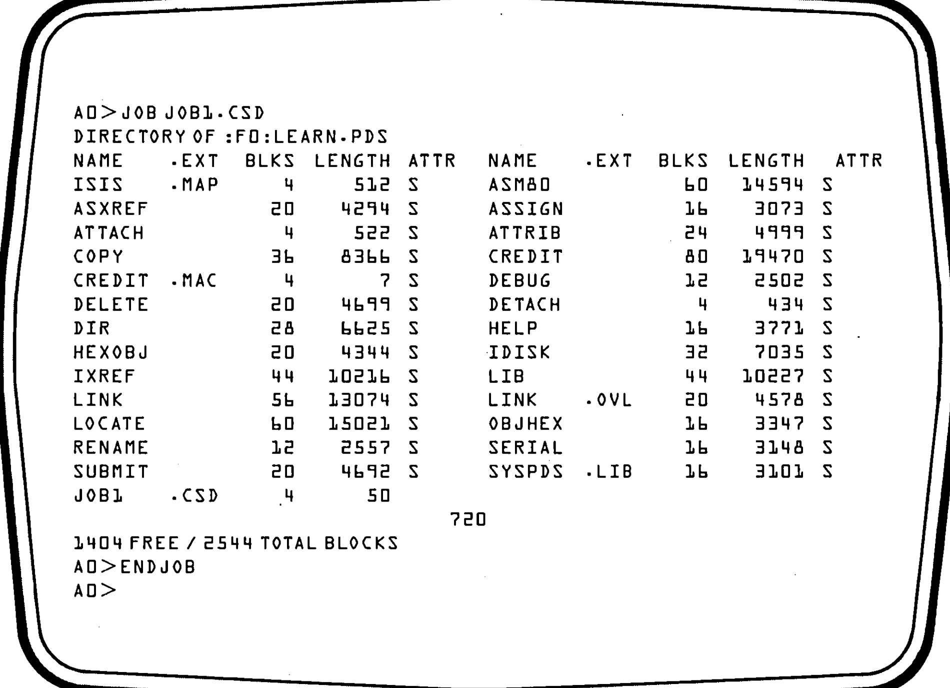


Key-in Sequence **Comments**

JOB JOB1.CSD

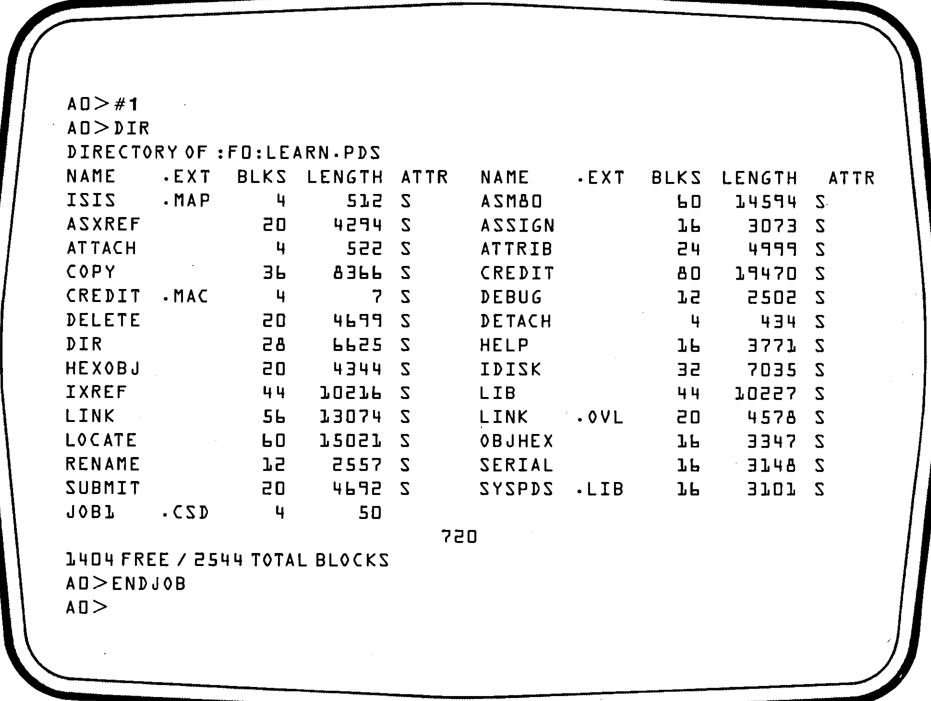


This example shows the second way to exit from the JOB command. Press the RETURN key at the beginning of a job line (after the = prompt) and the JOB command ends. The commands entered are saved in the specified file and are immediately executed in the sequence in which they were typed. The JOB command appends an ENDJOB command line at the end of the file, so that when all the commands are executed, control is returned to the operating system. The job file remains on the disk and can be executed again later.



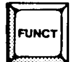
Comments

This screen shows the result of the previous entries. The third way to end a JOB command is to press the ESC key at the beginning of a job line. The commands entered are saved in the specified file, and the ENDJOB command line is appended to the file. However, the commands are not immediately executed.



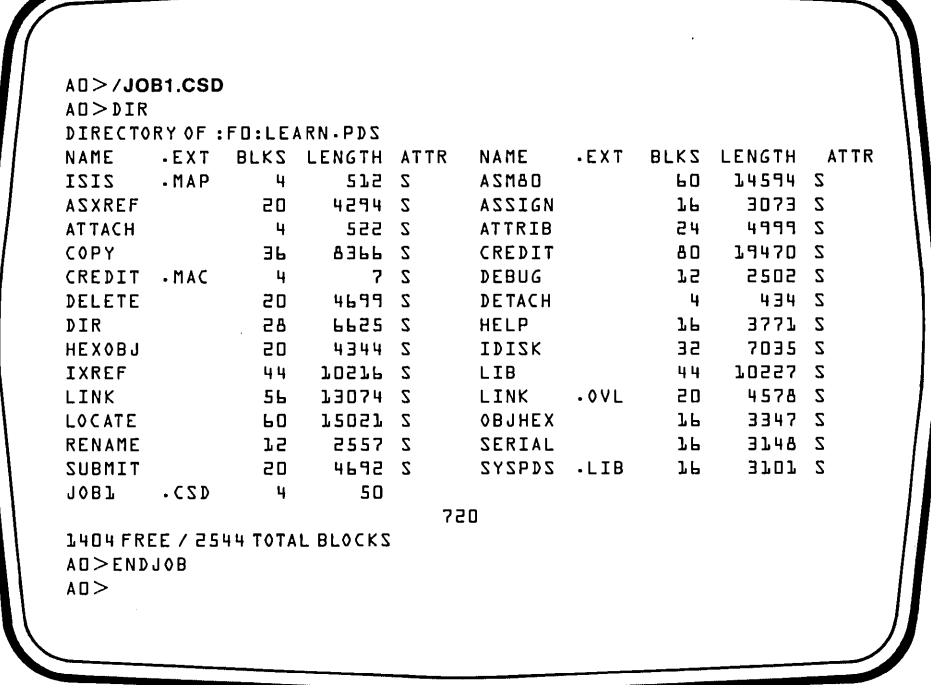
```
A0>#1
A0>DIR
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT  BLKS  LENGTH  ATTR  NAME    .EXT  BLKS  LENGTH  ATTR
ISIS    .MAP     4     512  S     ASM80           60   14594  S
ASXREF          20    4294  S     ASSIGN          16    3073  S
ATTACH           4     522  S     ATTRIB          24    4999  S
COPY            36    8366  S     CREDIT          80   19470  S
CREDIT  .MAC     4       7  S     DEBUG           12    2502  S
DELETE          20    4699  S     DETACH           4     434  S
DIR             28    6625  S     HELP            16    3771  S
HEXOBJ          20    4344  S     IDISK           32    7035  S
IXREF           44   10216  S     LIB             44   10227  S
LINK            56   13074  S     LINK    .OVL    20    4578  S
LOCATE          60   15021  S     OBJHEX          16    3347  S
RENAME          12    2557  S     SERIAL          16    3148  S
SUBMIT          20    4692  S     SYSPDS  .LIB    16    3101  S
JOB1    .CSD     4      50
                              720
1404 FREE / 2544 TOTAL BLOCKS
A0>ENDJOB
A0>
```

Key-in Sequence

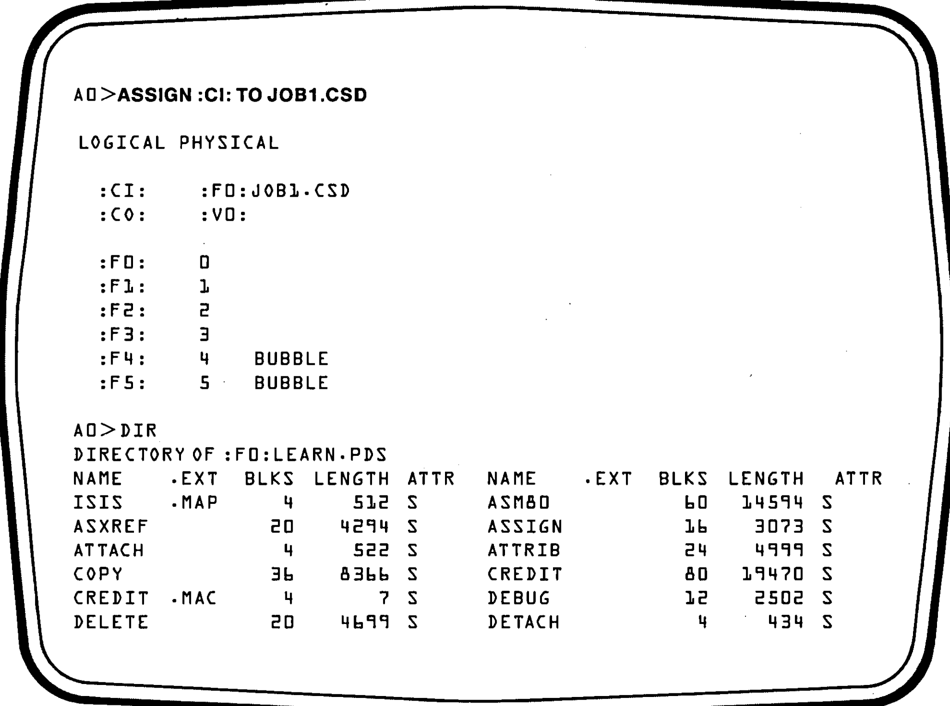


Comments

There are four ways to run the commands in a job file after exiting the JOB command: use the FUNCT key, use the / command, use the ASSIGN command, and use the automatic configuration feature of the operating system. This example illustrates using the FUNCT key. This method only works if the filename of the job file is in the form JOB<n>.CSD where <n> is a digit from 0 to 9. Press FUNCT<n> where <n> is the digit in the filename to run the job file.



Key-in Sequence	Comments
/JOB1.CSD	The / command is a second method of running the commands in the job file. Type / followed by the filename of the job file, and the commands are executed. When the ENDJOB command is reached, control is returned to the operating system.



Key-in Sequence

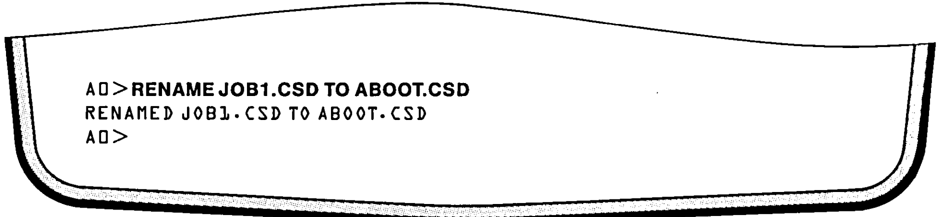
ASSIGN :CI: TO JOB1.CSD

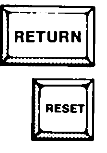
Comments

A third way to run the commands in a job file is to assign the system input device to the job file as shown in this example. The operating system reads the job file for command input until it encounters the ENDJOB command. At this time, the system is returned to the keyboard.

Configuring a User System Automatically

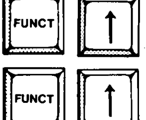
The next example shows a jobfile being used as a configuration file. A configuration file is run automatically when the system is initialized. For example, the ASSIGN command could be run to set up the required logical to physical device mapping for a system. Alternately, the SERIAL command could be used followed by the ASSIGN command to first configure the serial port and then map the serial device to the ISIS logical console device. The ATTACH command could be occur in a configuration file for a system containing multimodules. Finally, the confidence tests could be executed automatically from a configuration file.



Key-in Sequence	Comments
RENAME JOB1.CSD TO ABOOT.CSD 	When there is file named ABOOT.CSD (BBOOT.CSD for the optional processor) on the system disk used to initialize the system, this file is automatically run as a job file when the system is reset. Rename the job file containing the DIR command to ABOOT.CSD. Then, press the RESET key. As soon as the system is initialized, this file is read and executed as a job file.

```
ISIS-PDS, V1.0
A0>DIR
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT  BLKS  LENGTH  ATTR   NAME    .EXT  BLKS  LENGTH  ATTR
ISIS    .MAP     4     512  S      ASM80            60   14594  S
ASXREF          20    4294  S      ASSIGN           16    3073  S
ATTACH           4     522  S      ATTRIB           24    4999  S
COPY            36    8366  S      CREDIT           80   19470  S
CREDIT  .MAC     4       7  S      DEBUG            12    2502  S
DELETE          20    4699  S      DETACH            4     434  S
DIR             28    6625  S      HELP             16    3771  S
HEXOBJ          20    4344  S      IDISK            32    7035  S
IXREF           44   10216  S      LIB              44   10227  S
LINK            56   13074  S      LINK    .OVL     20    4578  S
LOCATE          60   15021  S      OBJHEX           16    3347  S
RENAME          12    2557  S      SERIAL           16    3148  S
SUBMIT          20    4692  S      SYSPDS  .LIB     16    3101  S
ABOOT   .CSD     4      50
                                720
1404 FREE / 2544 TOTAL BLOCKS
A0>ENDJOB
A0>
```

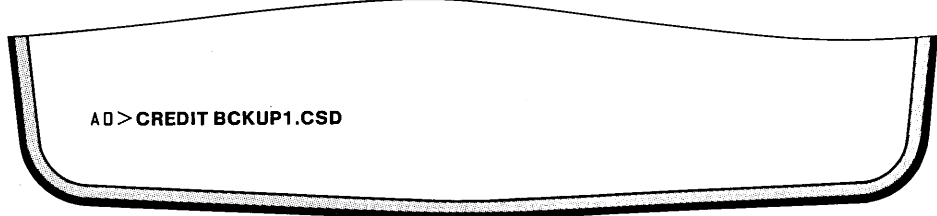
Key-in Sequence | **Comments**



This screen shows the result of entering the previous command. A good application for an automatic configuration file is in a dual processor system with a terminal connected to the serial port as the base processor's console device. The first command in the file could be the SERIAL command required to configure the terminal. This could be followed by the ASSIGN commands necessary to assign the system input and output (:CI: and :CO:) to the terminal. Another good application is in a system using a multimodule. The configuration file could contain the ASSIGN and ATTACH commands and the other commands necessary to load the I/O driver for the multimodule.

Using the SUBMIT Command

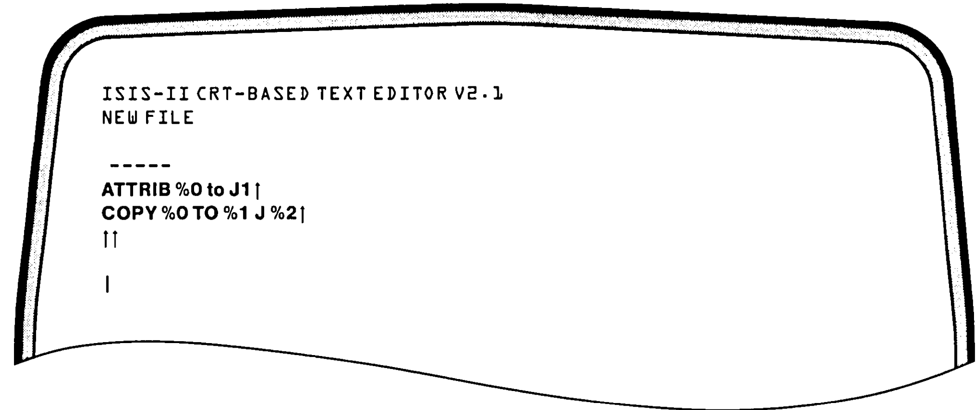
In the next series of examples, the SUBMIT command is illustrated. Notice the differences between SUBMIT files and jobfiles as this example is presented.



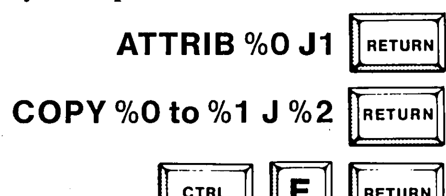
Key-in Sequence | **Comments**

CREDIT BCKUP1.CSD

In addition to entering operating system commands a single command line at a time and entering them through a job file, SUBMIT files can also be used. A SUBMIT files is a text file containing operating system commands and created with an editor. Here, CREDIT is used to create a SUBMIT file.



Key-in Sequence

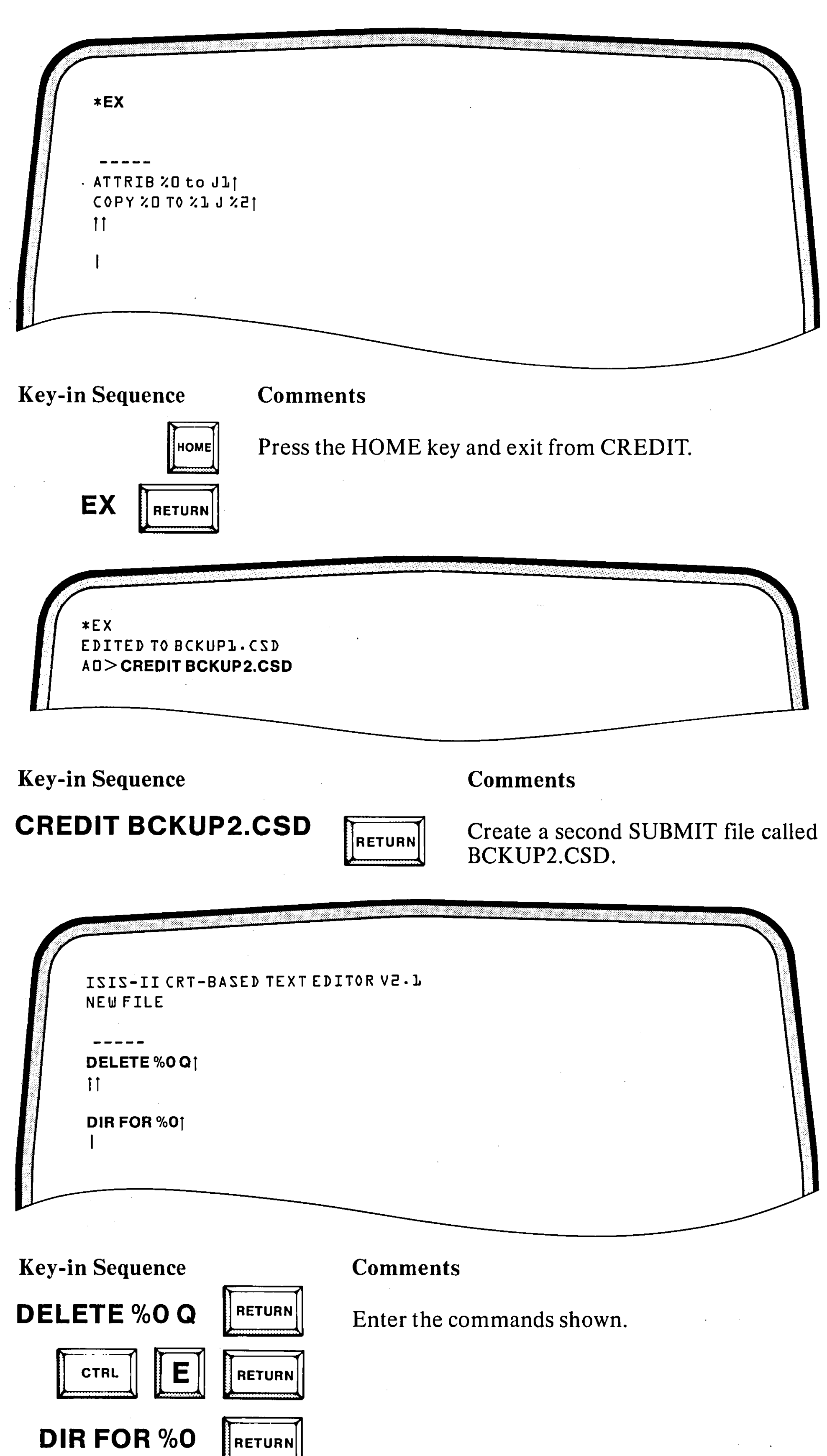


Comments

Enter the commands shown. The differences between a SUBMIT file and a job file are as follows:

Parameters can be passed to a SUBMIT file. The %0, %1, and %2 in the file act as placeholders and can be assigned different values each time the SUBMIT file is run. SUBMIT files can be interrupted to allow interactive input. The CTRL-E in the file shown illustrates this feature. As soon as the line with CTRL-E is run, the submit file halts and allows input from the keyboard. The next CTRL-E returns to the correct place in the SUBMIT file for further input. SUBMIT files can be nested. Thus, the SUBMIT command can appear in a SUBMIT file. There is no ENDJOB command at the end of a SUBMIT file. Therefore, two SUBMIT files can be appended to create one larger SUBMIT file.

A job file can be edited to delete the ENDJOB command. Then, it can be run as a SUBMIT file. A SUBMIT file with no nesting, no interactive input, and no parameters can be edited to add an ENDJOB command. Then, it can be run as a job file.

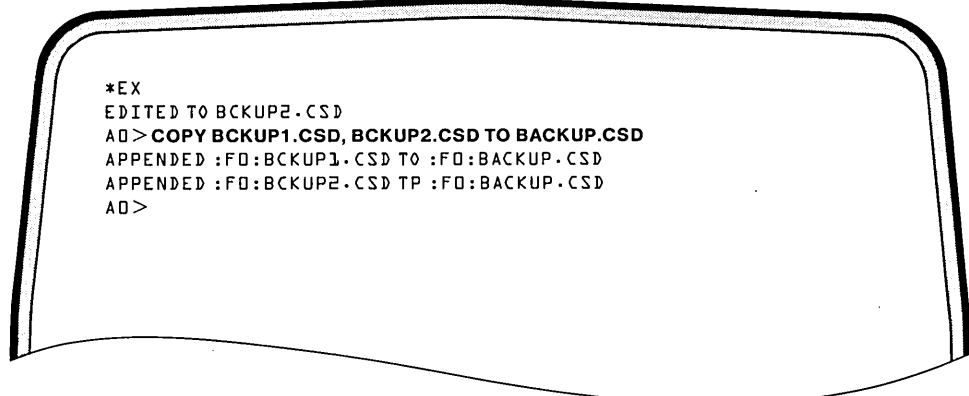


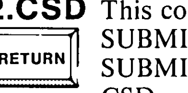
Key-in Sequence	Comments
HOME	Press the HOME key and exit from CREDIT.
EX RETURN	

Key-in Sequence	Comments
CREDIT BCKUP2.CSD RETURN	Create a second SUBMIT file called BCKUP2.CSD.

Key-in Sequence	Comments
DELETE %0 Q RETURN	Enter the commands shown.
CTRL E RETURN	
DIR FOR %0 RETURN	

Key-in Sequence	Comments
EX RETURN	Exit from CREDIT. There are two ways these two SUBMIT files can be run together: BCKUP1.CSD can be edited to add a SUBMIT command at the end to SUBMIT BCKUP2.CSD, or the two files can be appended as shown in the next example.



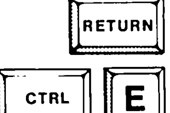
Key-in Sequence	Comments
COPY BCKUP1.CSD, BCKUP2.CSD TO BACKUP.CSD 	This command appends the two SUBMIT files to create a single SUBMIT file named BACKUP.-CSD.

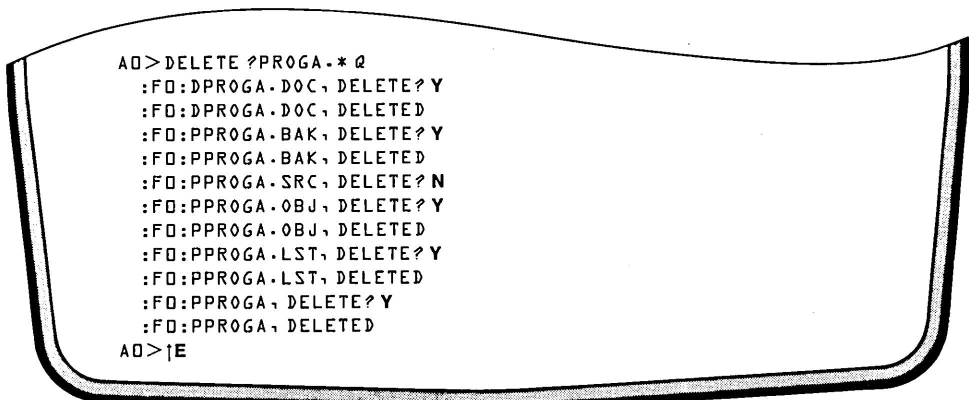
Running the SUBMIT File

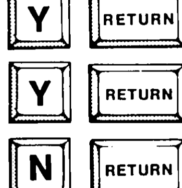
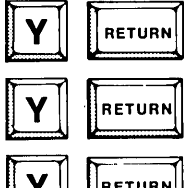
In this example, the same SUBMIT file that was just entered is run first on a single drive system and then on a multiple drive system.

```
A0>SUBMIT BACKUP(?PROGA.*,?PROGA.*,P)
A0>ATTRIB ?PROGA.* J1
       FILE        CURRENT ATTRIBUTES
  :F0:DPROGA.DOC          J
  :F0:PPROGA.BAK          J
  :F0:PPROGA.SRC          J
  :F0:PPROGA.OBJ          J
  :F0:PPROGA.LST          J
  :F0:PPROGA              J
A0>COPY ?PROGA.* TO ?PROGA.* J P
LOAD SOURCE DISKETTE, THEN TYPE (CR)ua E
LOAD OUTPUT DISKETTE, THEN TYPE (CR)
COPIED :F0:DPROGA.DOC TO :F0:DPROGA.DOC
COPIED :F0:PPROGA.BAK TO :F0:PPROGA.BAK
COPIED :F0:PPROGA.SRC TO :F0:PPROGA.SRC
COPIED :F0:PPROGA.OBJ TO :F0:PPROGA.OBJ
COPIED :F0:PPROGA.LST TO :F0:PPROGA.LST
COPIED :F0:PPROGA TO :F0:PPROGA
LOAD SYSTEM DISKETTE, THEN TYPE (CR)
A0>↑E
```

Key-in Sequence	Comments
SUBMIT BACKUP.CSD (?PROGA.*, ?PROGA.*,P) RETURN	This example shows how to run the SUBMIT file just created. The file has three parameters. All three parameters are specified. Wildcard filenames are given for the first two parameters (%0 and %1) and the P option is specified for the third parameter. The P option is appended to the COPY command line causing a copy for a single drive system to take place. The E is the display given when the CTRL-E is read from the file. The SUBMIT command stops reading from the submit file and takes input from the keyboard.
	The files to be copied are on disk currently in the drive. Press the RETURN key and COPY command begins. When the files have been read, the prompt to load the output diskette appears. Remember, the system is still accepting input from the keyboard.

Key-in Sequence	Comments
	Remove the system diskette and insert another diskette. Use the non-system diskette created in a previous example. Press RETURN. The files are copied with a message displayed for each file. When the last copy is complete, the system prompts to load the system diskette. The system is still accepting input from the keyboard.
	Remove the non-system diskette and insert the system diskette. Press RETURN. Enter a CTRL-E to return to the correct place in the SUBMIT file now that the interactive input is ended.



Key-in Sequence	Comments
  Y RETURN CTRL E	The next command that runs from the SUBMIT file is the DELETE command. The ↑E is displayed as soon as the CTRL-E from the file is read. The system halts to allow interactive input from the keyboard in response to each prompt. PPROGA.SRC is not deleted, while all the other files are. The files may not be deleted in the order shown. The actual order depends on previous examples. As soon as the CTRL-E is entered at the keyboard, the system returns to the correct place in the SUBMIT file.

```
  :F0:PPROGA, DELETE? Y
  :F0:PPROGA, DELETED
A0>↑E
A0>DIR FOR ?PROGA.*
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT  BLKS  LENGTH  ATTR   NAME    .EXT  BLKS  LENGTH  ATTR
PPROGA  .SRC     4     157   J
                                     4
1412 FREE / 2544 TOTAL BLOCKS
A0>:F0:SUBMIT RESTORE :F0:BACKUP.CS(:VI:)
A0>
```

Comments

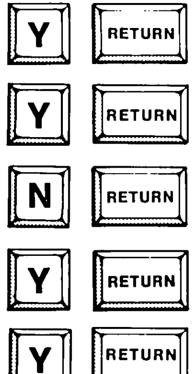
The rest of the SUBMIT file is run without interruption. The final command line run before returning to the operating system is a special form of the SUBMIT command generated by the SUBMIT command when the end of the SUBMIT file is reached. It restores the keyboard as the system input device.

```
A0>SUBMIT BACKUP (?PROGA.*,:F1:?PROGA.*,,)
A0>ATTRIB ?PROGA.* J1
       FILE        CURRENT ATTRIBUTES
  :F0:DPROGA.DOC           J
  :F0:PPROGA.BAK           J
  :F0:PPROGA.SRC           J
  :F0:PPROGA.OBJ           J
  :F0:PPROGA.LST           J
  :F0:PPROGA               J
A0>COPY ?PROGA.* TO :F1:?PROGA.* J
COPIED :F0:DPROGA.DOC TO :F0:DPROGA.DOC
COPIED :F0:PPROGA.BAK TO :F0:PPROGA.BAK
COPIED :F0:PPROGA.SRC TO :F0:PPROGA.SRC
COPIED :F0:PPROGA.OBJ TO :F0:PPROGA.OBJ
COPIED :F0:PPROGA.LST TO :F0:PPROGA.LST
COPIED :F0:PPROGA TO :F0:PPROGA
A0>↑E↑E
```

Key-in Sequence	Comments
SUBMIT BACKUP (?PROGA.*, :F1:?PROGA.*,,) RETURN	The same SUBMIT file is run for multiple drive systems. Insert the non-system diskette created in a previous example in drive 1. Enter the SUBMIT command as shown. Notice the second parameter contains a drive specification. The third parameter is not given. The COPY command is run without the P option.
CTRL E	The SUBMIT file is run until the CTRL-E is read. Since no keyboard input is required at this point in the example, enter a CTRL-E from the keyboard to immediately switch back to the file.

```
A0>DELETE ?PROGA.* Q
  :F0:DPROGA.DOC, DELETE? Y
  :F0:DPROGA.DOC, DELETED
  :F0:PPROGA.BAK, DELETE? Y
  :F0:PPROGA.BAK, DELETED
  :F0:PPROGA.SRC, DELETE? N
  :F0:PPROGA.OBJ, DELETE? Y
  :F0:PPROGA.OBJ, DELETED
  :F0:PPROGA.LST, DELETE? Y
  :F0:PPROGA.LST, DELETED
  :F0:PPROGA, DELETE? Y
  :F0:PPROGA, DELETED
A0>↑E
```

Key-in Sequence



Comments

The system returns to the SUBMIT file and runs the next command. The second CTRL-E is read from the file and the system halts again allowing keyboard input. Enter the responses to the prompts as shown. The order of the files to be deleted may vary depending on the previous examples run. Enter a CTRL-E to switch back to the SUBMIT file.

```
  :F0:PPROGA, DELETE? Y
  :F0:PPROGA, DELETED
A0>↑E
A0>DIR F0R ?PROGA.*
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT  BLKS  LENGTH  ATTR   NAME    .EXT  BLKS  LENGTH  ATTR
PPROGA  .SRC     4     157   J
                                    4
1412 FREE / 2544 TOTAL BLOCKS
A0>:F0:SUBMIT RESTORE :F0:BACKUP.CS(:VI:)
A0>
```

Comments

The SUBMIT file is completed without further interruption. Further information on SUBMIT can be found in Chapter 5.

CHAPTER 5 COMMAND DICTIONARY

Notational Conventions

Because of the many different ways that a single command can be entered, it is not possible or desirable to list every correct entry. Instead, the general format of the command is described using special symbols or notational conventions.

Notational conventions are symbols that have been adopted to help describe operating system commands. These symbols are not part of the command itself but are used to precisely describe the format of the command.

The special characters used in these conventions have no significance to the operating system and are only meaningful in describing a class of correct command entries. For example, items enclosed in brackets are optional parts of a command. The brackets themselves would never be entered on a command line, but the item within the brackets could optionally be included.

UPPERCASE — Characters shown in upper case must be entered exactly as shown. Uppercase is used to denote command keywords as shown in the following example:.

RENAME <filename 1> TO <filename 2>

<class name> — Angle brackets denote general terms that must be replaced by a specific member of the class referenced. For example, <filename> would be replaced by a valid ISIS-PDS filename and <address> would be replaced by a valid address. The commonly used general terms are discussed below. Often, a numeric suffix is added to distinguish different items of the same class. For example, <filename 1> and <filename 2> refer to two different filenames.

[<option>] — Brackets enclose optional material that may or may not be included on the command line. For example, [<switch>] is an optional item that may be appended to the COPY command if certain actions are desired.

... — Ellipses indicate that the preceding item can be repeated.

{<item>
<item>} — Braces indicate that one and only one of the enclosed entries must be selected. If the items are also enclosed by brackets, they are optional and no choice is required. For example,

{Y
N}

indicates a choice must be made to enter either Y or N. The enclosed choices are printed in a vertical column.

{<item>} ...
{<item>}

Braces followed by ellipses indicate that at least one of the enclosed items must be selected. If the items are also enclosed by brackets, they are optional and no choice is required. The items may be used in any order unless otherwise specified. For example,

{A} ...
{B}
{C}
{D}

indicates that a choice must be made to include one or more of the items A, B, C, or D.

punctuation

Punctuation other than ellipses, braces, and brackets must be entered as shown. For example, the commas and parentheses in the following command must be entered:

```
SUBMIT <command name> (<parm1>,<parm2>,<parm3>)
```

Special Command Format Terms

In addition to notational conventions, the command format descriptions in this manual contain the following general terms that are common to many commands:

<device name> <source> <jobfile>
<filename> <destination>
<extension> <n>
<pathname> <a>

These terms are described in detail in the following sections before they are used in command descriptions.

Device Names

Device names are system-assigned names in the form:

:<device name>:

<digit>

The :<device name>: form is one form recognized by ISIS commands. Device names are recognized for physical devices and logical devices. Physical device names can be assigned to system-defined devices or to user-defined devices. Logical device names are system defined.

Physical Devices

Physical devices are hardware units that are separate from the processor itself and are used for input and output of data for the processor. The operating system recognizes and handles a wide range of devices, such as disk drives, bubble memory devices, a printer, a keyboard, and a CRT display screen.

Some of these devices, called system-defined devices, have I/O drivers included in the operating system. Other devices, called user-defined devices, can be defined by the user, and the user-written I/O driver can be added to the operating system.

System-Defined Devices. System-defined devices are devices which can be accessed without modifying the operating system. That is, system-defined devices are predefined; I/O driver routines are provided by ISIS-PDS for accessing them. The names assigned for system-defined devices are:

:VI: Video terminal keyboard (input only)
:VO: Video terminal screen (output only)
:SI: Device connected to Serial port (input only)
:SO: Device connected to Serial port (output only)
:LP: Device connected to Line Printer port (output only)
:TR: Teletype Paper Tape Reader connected to Serial port (input only)
:TP: Teletype Paper Tape Punch connected to Serial port (output only)
:HR: High Speed Paper Tape Reader connected to Serial port (input only)
:HP: High Speed Paper Tape Punch connected to Serial port (output only)

The TR, TP, HR, and HP devices are the same as the SI and SO devices. These device names were added to maintain compatibility with previous versions of the ISIS operating system.

The operating system also provides I/O support for six disk devices: four disk drives and two bubble memory multimodules. However, the names assigned to these physical devices vary from the form of the preceding physical device names. The physical disk devices are named as follows:

0 Physical disk drive 0 (input and output)
1 Physical disk drive 1 (input and output)
2 Physical disk drive 2 (input and output)
3 Physical disk drive 3 (input and output)
4 Bubble memory multimodule 1 (input and output)
5 Bubble memory multimodule 2 (input and output)

Disk drive 0 is the internal disk drive, and drives 1 through 3 are external drives in the order in which they are connected to the system. Bubble memory multimodule 1 is the bubble memory device installed in connector J1 and bubble memory multimodule 2 is the bubble memory device installed in connector J3. See Appendix A for installation instructions.

User-Defined Devices. User-defined devices are devices for which the user must provide customized I/O routines. The system recognizes the following names for user-defined devices:

:L1: User list device
:I1: User console input device
:O1: User console output device
:R1: User reader input device 1
:P1: User punch output device 1
:R2: User reader input device 2
:P2: User punch output device 2

The R1, P1, R2, and P2 devices were incorporated to maintain compatibility with previous versions of ISIS. Refer to Chapter 8 for further information on generating custom I/O drivers for user-defined devices and on adding these drivers to the operating system.

Logical Devices

Logical devices do not exist physically as a printer or disk drive physically exists. They are symbolic device names that the operating system recognizes to provide flexibility for input or output of data. These devices are assigned by the ASSIGN command to one of the physical devices described above. Logical devices recognized by the operating system are the logical disk devices, the console input device, the console output device, and the byte bucket.

The logical disk devices can be assigned to one of the physical disk drives or to one of the bubble memory multimodules or to the byte bucket. They are named as follows:

:F0: Logical disk device 0
:F1: Logical disk device 1
:F2: Logical disk device 2
:F3: Logical disk device 3
:F4: Logical disk device 4
:F5: Logical disk device 5

Logical disk device 0 (:F0:) is always the system default disk device, meaning that, if no disk device is specified, the system assumes the device :F0:. :F0: is initially assigned to the boot device, drive 0 or bubble memory 4. By ASSIGNing :F0: to some other disk device, any physical disk drive or bubble memory multimodule can become the system default disk device.

The console provides interactive control over the system. It is the device from which commands are entered and to which system messages are sent. The logical console device names are as follows:

:CI: Console input
:CO: Console output

The ASSIGN command can be used to assign any physical input device (including a disk file) as the console input and any physical output device (including a disk file) as the console output.

:CI: is always a line edited file; :CO: is its associated echo file. A line edited file is a temporary buffer in memory that contains the command line characters as they are keyed in at the keyboard. These characters can be edited using the editing control characters described in Chapter 3. An echo file is a file containing the echoed characters from the line edited file. As data is input to :CI:, it is echoed on :CO:. Both files are always open, i.e., accessible at all times.

The keyboard (:VI:) and screen (:VO:) are initially the :CI: and :CO: respectively. As characters are typed at the keyboard, they are echoed on the screen. However, some other physical device, such as a disk file or a user-defined device, may be assigned as the console I/O. A printer can also be assigned as the console output. Whenever an end of file, generated by the ENDJOB command, is encountered on the :CI: device, :CI: is automatically changed back to the keyboard (:VI:).

The byte bucket is a logical I/O device which acts as an infinite sink for bytes when written to and a file of zero length when read from. It is used for data that is not to be saved or displayed. In software development, a write only device can be useful in simulating I/O and also in isolating a bad file without creating more bad data by copying the suspected file to the write only device. The system name assigned to the byte bucket is:

:BB: Byte bucket

Filenames

A major purpose of the operating system is to ease the programming task of implementing files on disk devices. Many programs operate on files or produce files as their output. Programs themselves are contained in files and are executed under the operating system by entering the name of the file.

A file is a sequence of 8-bit bytes. Programs receive information by reading from files and transmit information by writing to files. Each file must be fully contained on one physical device. Usually, files are thought of as disk files or bubble memory files. However, non-disk devices can also be thought of as single file devices that can be opened, written, and read.

Filenames provide a standard way of identifying and accessing files. All system files come with system assigned names. The user assigns names to files created with commands such as CREDIT or COPY. The filename for a file on a non-disk device is blank.

The term filename refers to both the name of the file and its extension, if any. Each file on a disk must have a unique filename. The general format for a filename is:

<name>.<extension>

where

<name>	is a one to six character name assigned to a file. The characters may be alphabetic or numeric.
<extension>	is a one to three character modifier created for a name. An <extension> is optional when the file is created, but if .<extension> is specified, it must always be used when referencing the file.

Examples of valid filenames are:

REPORT.TXT	PROG.OBJ	P3987.V1	DIR
SYMBOL.SRC	A.B	COPY	RENAME

Default extensions are predefined extensions that certain programs assume when no extension is provided. Default extensions are designed to save the time when entering commands.

Examples of default extensions are:

.OBJ	Output from translator program
.CS	Output from SUBMIT program
.BAK	Output from CREDIT program
.TMA	Temporary output from PLM80, ASM80, and CREDIT program on Processor A
.TMB	Temporary output from PLM80, ASM80, and CREDIT program on Processor B

Default extensions are explained further under the individual commands which assign and use them. It is recommended that such extensions not be assigned to user created files. The extension .TMP cannot be used under ISIS-PDS. If .TMP is assigned to a file by the user, it is automatically converted to .TMA, and it may then conflict with one of the defaults.

Wildcard Filenames

A wildcard filename uses wildcard characters to specify multiple files which share characters in their filenames. Several operating system commands allow the wildcard characters to replace the standard characters in a filename or extension. Entering a command with a wildcard filename has the same effect as entering the command more than once with a single filename each time.

The two wildcard characters are:

* * Asterisk specifies a match to any number of characters.
* ? Question mark specifies a match to a single character. It does not match a blank character.

For example, the asterisk can be used to match any name or any extension in the disk directory:

ABC.* matches any filename with the name ABC and any or no extension.
*.PLM matches any filename with the extension .PLM, such as MYPROG.PLM.

The asterisk can also be used to match the names or extensions with the same first characters:

AB*.HEX matches any filename with AB as the first two characters of the name and HEX as the extension. This example would match ABC.HEX, ABXYZ.HEX, or AB.HEX.

An asterisk preceding the initial character in a name or extension is not valid, for example, *B.HEX and *.*B are not a valid wildcard filenames. However, *.* is a valid wildcard filename.

Each question mark substitutes for a single character that can be a wildcard match. For example,

A?B.HEX matches any filename beginning with A and with B as the third character and with an extension of .HEX.

A??.* matches any filename with a three character name beginning with A and with an extension of any length.

Pathnames

A pathname uniquely specifies a file to be used in an operating system command. It consists of the device name directly followed by the filename (if there is a filename) without separating spaces. Single file devices, such as the line printer, do not have filenames. Pathname is sometimes abbreviated <pn> in command descriptions. The format for a disk file pathname is:

[:F<n>:]<filename>

where

<n> is the logical disk drive number from 0 to 5 as described previously. If no device is specified, the default disk device (:F0:) is assumed.

<filename> follows :F<n>: with no intervening space and consists of two parts: a <name> followed by an <extension>.

The format for non-disk files is:

:<device name>:

where

<device name> is any valid non-disk device name as described previously.

The following examples illustrate actual pathnames of disk files as well as a common use of extensions.

:F1:PROGA.SRC for the source code for a program
:F1:PROGA.LST for the listing from the translator
:F1:PROGA.OBJ for the object code
:F1:PROGA.LNK for the linked object code
:F1:PROGA for the code located at absolute addresses

Note that all these files have the same name and are distinguished only by their extensions. Extensions allow the different file types associated with a given program to be distinguished.

The following examples show the pathnames for some of the single file devices in the system:

:CO: for the current console output file
:VI: for the keyboard file
:SI: for the serial input file

Source

The term <source> in a command line refers to the input to the command. The <source pn> is the input file.

Destination

The term <destination> in a command line refers to the output for the command. The <destination pn> is the output file.

N and A

The lower case letter <n> refers to a number usually used as a suffix for some other part of the command.

W<n>

means W followed by a number. The range of values that <n> can take on is specified in the command where it is used.

The letter <a> refers to an alphanumeric string of characters. The range of values that <a> can take on is defined in the command where it is used.

Jobfile

The term <jobfile> refers to a file which contains operating system command lines and which is processed by commands such as JOB to execute a series of programs in batch mode.

Command Description Formats

In addition to the conventions described above, a standard form is used in this chapter to describe each command. This helps to access the reference information at a glance.

The operating system commands appear in alphabetical order as a reference guide for the experienced user. Each command begins on a new page with the command keyword at the top outside margin on the page; the syntax is followed by a brief description of the items required in the command; and short examples are given as illustrations of the command in use. This format is shown in figure 5-1.

Some of the examples use the logical drives :F1:, :F2:, and :F3:. These examples can usually be run on a single drive system by assigning all logical drives to drive 0.

```
                         COMMAND KEYWORD
                                  brief phrase
                                  describing command.

COMMAND FORMAT

KEYWORD <parameter 1 through parameter n>

<parameter 1>          ................................
                       ......... Description ..........
                       ................................
                                  .
                                  .
                                  .
<parameter n>          ................................
                       ......... Description ..........
                       ................................

COMMENTS

Comments related to operating the command.

EXAMPLES

A few brief illustrations of the command line.
```

0161

Figure 5-1. Format of Command Descriptions

Functional Summary of Commands

The command categories discussed in Chapter 4 are repeated below for reference. Only categories and commands described in this chapter are listed.

The System Management Group

HELP
?

The File Management Group

ATTRIB	DIR
COPY	RENAME
DELETE	@

The Device Management Group

ASSIGN
ATTACH
DETACH
IDISK
SERIAL
#
/
FUNCT <n>

The Program Execution Group

ASSIGN
ENDJOB
JOB
SUBMIT
.
/
ESC
FUNCT <n>

ASSIGN

Maps logical devices
to physical devices

Command Format

```
        { :CI:                   TO  <console input device>  }
ASSIGN  { :CO:                   TO  <console output device> }
        { <logical disk device>  TO  <physical disk device>  }
```

where

<console input device> specifies a physical device to be used for console input. Valid devices are :VI:, :SI:, :I1:, :BB:, or the pathname of a disk file.

<console output device> specifies a physical device to be used for console output. Valid devices are :VO:, :SO:, :O1:, :BB:, or the pathname of a disk file.

<logical disk device> specifies one of the ISIS-PDS logical disk device names. Valid names are :F0:, :F1:, :F2:, :F3:, :F4:, and :F5:.

<physical disk device> specifies one of the ISIS-PDS physical disk device names. Valid names are 0, 1, 2, 3, 4, or 5.

Comments

The ASSIGN command is used to change the assignment of logical devices to physical devices. It also displays the new assignment after the change is made. Entering the ASSIGN command without any parameters displays a listing of the current device assignments.

After initializing the system from drive 0, the assignments are as follows:

LOGICAL	PHYSICAL	
:CI:	:VI:	
:CO:	:VO:	
:F0:	0	
:F1:	1	
:F2:	2	
:F3:	3	
:F4:	4	BUBBLE
:F5:	5	BUBBLE

The logical devices that can be assigned are shown in the left column and the physical devices to which they are currently assigned are shown in the right column. The console input and console output are initially assigned to the keyboard and CRT display respectively. The logical disks :F0: through :F3: are initially assigned to the four disk drives, 0 through 3, respectively; and logical disks :F4: and :F5: are initially assigned to the two bubble memories, 4 and 5, respectively.

After initializing the system from the bubble memory multimodule, the assignments are as follows:

LOGICAL	PHYSICAL	
:CI:	:VI:	
:CO:	:VO:	
:F0:	4	BUBBLE
:F1:	0	
:F2:	1	
:F3:	2	
:F4:	3	
:F5:	5	BUBBLE

The only difference is with the disk drive assignment. The disk :F0: is assigned to bubble memory 4, :F1: is assigned to disk drive 0, :F2: to disk drive 1, :F3: to disk drive 2, and :F4: to disk drive 3.

Disk drive 0 is the internal drive and disk drives 1, 2, and 3 are the optional external drives. Bubble memory 4 is the bubble memory multimodule installed at connector J1 of the multimodule adapter board and bubble memory 5 is the bubble memory multimodule installed at connector J3 of the multimodule adapter. See Appendix A for installation instructions for these devices.

The system default disk device is always :F0:. If no disk device is specified in a pathname, the drive to which :F0: is currently assigned is used. The ISIS-PDS system prompt in the form Ad> (or Bd> for the optional processor) shows the number of the physical drive, d, to which :F0: is currently assigned.

Since :F0: is the system default device, it must always be assigned to a physical device that contains a system disk with all the commands needed, or the system does not operate correctly. For example, assume that :F0: is assigned to drive 0 and :F3: is assigned to drive 3 and that the two drive assignments are to be switched. If :F0: is re-assigned first to drive 3, drive 3 must contain an ISIS-PDS system disk. Otherwise, the second ASSIGN command can never be made.

Examples

The commands

```
ASSIGN :F3: TO 0
ASSIGN :F0: TO 3
```

switch the assignment for :F0: and :F3:.

The commands

```
ASSIGN :CO: TO :SO:
ASSIGN :CI: TO :SI:
```

change the console input and output devices to the device connected to the serial port. These two commands allow a terminal connected to the serial port to provide

interactive control over the system. The primitive calls :CI: and :CO: still go to the iPDS keyboard. See the "A Command" in chapter 8.

The command

```
ASSIGN :CI: TO :F1:CMDFIL.TXT
```

changes the console input assignment to the file, :F1:CMDFIL.TXT. The system immediately begins reading this file and executing commands from it. The last command in the file must be an ENDJOB command to return control to the keyboard, :VI:, at the end of the file.

The command

```
ASSIGN :CO: TO :F1:LOGFIL.TXT
```

changes the console output assignment to the file, :F1:LOGFIL.TXT. The system immediately stops echoing user input and command output messages on the CRT display and begins saving them in the file. The first text in the file is the output from the ASSIGN command:

```
LOGICAL      PHYSICAL

 :CI:        :VI:
 :CO:        :F1:LOGFIL.TXT

 :F0:          0
 :F1:          1
 :F2:          2
 :F3:          3
 :F4:          4        BUBBLE
 :F5:          5        BUBBLE
```

After this command, anything typed at the keyboard would have to be typed blind since the characters are no longer echoed on the CRT screen. This command would be useful as the first command in a jobfile. Commands would be executed automatically from the jobfile and a record of the output would be saved in the log-file for later examination.

ATTACH

Assigns multimodule row to a processor

Command Format

ATTACH <multimodule row>

where

<multimodule row> specifies which multimodule row to attach: 0 to attach the first row and 1 to attach the second row.

Comments

The ATTACH command only applies to systems with the optional iSBX Multimodule Adapter board. The adapter board has four connectors for up to four multimodule boards: two in row 0 which correspond to multimodule connectors J1 and J2 and two in row 1 which correspond to multimodule connectors J3 and J4. See Appendix A for a figure showing the location of these connectors.

To use non-bubble multimodules, the multimodule row that contains the device must be attached to the processor. Once the row has been attached to a processor, that processor can access either multimodule on the row. On systems with a single processor, the ATTACH command assigns the specified row to the processor. See Chapter 9 for information on running commands on the optional processor.

Some multimodules use one multimodule connector and cover the other connector in the same row. In this case, attaching a row only makes one multimodule device available to the processor.

The operating system considers the bubble multimodule to be a sharable device like a disk drive. Bubble multimodules need not be ATTACHed to a processor before accessing them through the operating system. Attempting to ATTACH a multimodule row that contains bubble memory results in an error. The error message displayed in this case is:

61 MODULE ALREADY ASSIGNED TO BUBBLE

If a multimodule pair is already attached to a processor and the ATTACH command is run again, the following message is displayed:

60 MODULE ALREADY ASSIGNED

See Chapter 8 for information on using multimodules.

Examples

The command

ATTACH 1

attaches the multimodules in Row 1 (J3 and J4) to the processor.

ATTACH 0

attaches the multimodules in Row 0 (J1 and J2).

ATTRIB

Displays and modifies attributes of disk files

Command Format

```
                        [ { W<n> }       ]
                        [ { I<n> }       ]
                        [ { S<n> }       ]
ATTRIB <pathname>       [ { F<n> }  ...  ] [Q]
                        [ { J<n> }       ]
                        [ { K<n> }       ]
                        [ { L<n> }       ]
```

where

<pathname> specifies the file whose attributes are being displayed or modified. Wildcard characters are valid.

<n> is a numeric suffix following the particular attribute that specifies whether that attribute is set or reset. If <n> is 0, the attribute in question is reset, i.e., turned off. If <n> is 1, the attribute is set.

W is the write protect attribute. A file cannot be written to with this attribute set.

I is the invisible attribute. An invisible file is not displayed in the normal listing of files on the disk. See the DIR command.

S is the system attribute. A file with this attribute is an integral part of the operating system and should be present on any System Disk.

F is the format attribute. This attribute is used by the ISIS-PDS operating system and the IDISK command. Normally, the user should not assign or remove this attribute from a file.

{ J K L } are user defined attributes. They can represent any type of file the user wishes.

Q is the Query option which causes the ATTRIB command to prompt before actually changing or displaying the attributes.

<pathname>, MODIFY ATTRIBUTES?

The Q option is used if a wildcard filename is entered on the command line. This option causes ATTRIB to prompt for each specific pathname. Answering Y or y displays and changes the attributes. Any other response leaves that file's attributes unmodified and continues processing.

Comments

The command may be entered with a pathname containing wildcard characters so that attributes can be displayed or modified for a family of files at one time.

If no attributes are entered, the current settings are displayed for all the attributes. If attributes are changed, the new settings for all attributes are displayed after the change is made.

Any combination of attributes can be specified on a single command. If the same attribute is specified more than one time, the last occurrence of that attribute is the one used by the command. All other occurrences of the same attribute are ignored.

An error occurs when the pathname does not specify a disk file or when the disk file specified does not exist.

The ATTRIB command cannot be used in a single drive system to change the attributes of a file that is not on the system disk currently in the drive.

Examples

The command

```
ATTRIB :F1:MYFILE.TXT W0 I0 S0 J1 K1 L1
```

sets the J, K, and L attributes while turning off the W, I, and S attributes for MYFILE.TXT on the physical drive to which :F1: is assigned. The new attributes are displayed for the file.

```
ATTRIB :F1:*.* W0 I0 S0 J0 K0 L0
```

turns off the W, I, S, J, K, and L attributes for all files on the drive to which :F1: is currently assigned. The new attributes are displayed.

The command

```
ATTRIB :F4:PROG.SRC W1 I1 S1 J0 K0 L0 Q
```

prompts as follows before setting the W, I, and S attributes and turning off J, K, and L.

```
:F4:PROG.SRC, MODIFY ATTRIBUTES? Y
```

Typing Y causes the attributes to be modified. The new attributes are displayed for the file.

COPY

Transfers files

Command Format

```
COPY <source pn> TO <dest pn> [{S}] [{B}] [{C} ...] [{J} ...]
                              [{N}] [{U}] [{P}    ] [{K}    ]
                                          [{Q}    ] [{L}    ]
```

where

<source pn> is the pathname of the input file, the file being copied. Wildcard characters are allowed.

TO <dest pn> is the pathname of the output file. If wildcard characters are used in the source pathname, they must also be used in the destination pathname. The destination filename and extension default to the source filename and extension. However, if allowed to default, the destination device name must be different from the source device. Otherwise, the P option is assumed and a single drive copy sequence is run. See P below.

S specifies that only files with the S attribute set and the F attribute not set is included in the COPY. This option is used to copy system files after a system disk has been initialized. See the ATTRIB command for further information on attributes.

N specifies that only files with the S and F attribute not set is included in the COPY. Only non-system and non-format files will be copied. See the ATTRIB command for further information on attributes.

B suppresses the prompt that is normally displayed if the destination file exists.

<dest pn> ALREADY EXISTS, DELETE?

The existing destination file is deleted and a new destination file is created with a copy of the source file.

U is the same as B (the prompt is suppressed) except that the existing file is not deleted first. The new destination file is copied over the existing destination file.

C copies the attributes (except for the F attribute) that are set for the source file. If the destination file already exists and has any attributes set, they are retained in addition to the source file attributes. Without the C option, the destination file only has attributes that are retained from the existing destination file if there are any. The F attribute is not copied even with the C option.

P is used on single drive systems to copy a file from one disk to another disk using the same drive. The program halts and prompts

LOAD SOURCE DISKETTE, THEN TYPE (CR)

Load the disk containing the file to be copied and press the RETURN key. When it is necessary to switch disks, the system halts and prompts

LOAD OUTPUT DISKETTE, THEN TYPE (CR)

The two prompts alternate until the file is copied. Then, the program halts and prompts

LOAD SYSTEM DISKETTE, THEN TYPE (CR)

to terminate the COPY and return to the operating system. P is automatically assumed as a default and need not be specified if the source name and the destination name are the same, including the device specification.

Q causes the program to display the prompt:

COPY <source pn> TO <destination pn>?

A response of "Y" or "y" causes the copy to take place. Otherwise, no copy is made for the specific file displayed. Q is used on wildcard copies to allow a decision on a file by file basis whether or not to perform the copy.

{J K L} specifies that only files with the J, K, or L attribute set are included in the COPY. If J, K, and L are combined, they are ANDed to determine which files are copied. For example, if J and K are both specified, only files with both J and K set are copied. If J, K, and L are specified, only files with all three attributes set are copied. See the ATTRIB command for more information on these attributes.

Comments

This form of the COPY command copies one file to another. See the following command description for another form of the COPY command that appends a series of source files.

If the source file is specified with wildcard characters, there are only two ways to specify the destination file. In the first way, the destination filename and extension is the same as the source filename and extension. Only the device is specified.

```
COPY :F1:*.TXT TO :F3:
```

is the same as:

```
COPY :F1:*.TXT TO :F3:*.TXT
```

The second way of specifying the destination name when the source contains wildcard characters allows the files to be renamed while they are being copied. The

destination file is specified with the same mask as the source file. There are three rules that determine if the source and destination names have the same mask.

1. For every position in the source wildcard name which contains an *, the corresponding position in the destination wildcard name must contain an * also.

2. For every position in the source wildcard name which contains a ?, the corresponding position in the destination wildcard name must contain either an * or a ? wildcard character.

3. For every position in the source wildcard name which contains no wildcard character, the corresponding position in the destination wildcard name must contain no wildcard character.

In following these rules, the command

```
COPY :F2:P?O?1.* TO :F4:A?O?1.*
```

is valid while the command

```
COPY :F2:P?O?1.* TO :F4:SKILL
```

is not valid. To summarize, the parts of the source and destination names that are explicitly entered must be the same length, and wildcard tokens used in the source name must be used in the same position in the destination name.

The N and S options only affect wildcard copies. They have no effect if non-wildcard filenames are used. The J, K, and L options affect both wildcard and non-wildcard copies.

On a single drive copy sequence (using the P option), after the source diskette has been inserted one time, the system remembers its identification. If the wrong source is inserted after subsequent LOAD SOURCE DISKETTE prompts, an error message is given and the user can insert the correct diskette.

```
WRONG DISKETTE IN DRIVE <n>
LOAD SOURCE DISKETTE, THEN TYPE (CR)
```

The default for the COPY command with wildcard characters is to copy only non-format files, that is, files without the F attribute set. The S, N, J, K, and L options define a different scope to limit the files copied.

The message

```
COPIED <source pn> TO <destination pn>
```

is displayed as each file is copied.

A COPY from the console input (:CI:) to a port or a file requires special consideration. This copy command is not terminated until the buffer space (40 Kbytes) is exceeded, or an end-of-file character (CTRL-Z) is detected. Be sure to terminate COPY commands in this category with CTRL-Z.

Examples

The command

```
COPY :CI: TO :SO: <cr>
Any text
CTRL-Z
```

copies the console input (Any text <cr>) to the serial output port. This command is not completed until the end-of-file character (CTRL-Z) is sent.

The command

```
COPY MYFILE.TXT TO :F2:OLD.TXT
```

copies the file MUFILE.TXT from :F0: to :F2: and renames it to OLD.TXT.

The command

```
COPY :F3:*.* TO :F2:
```

copies all non-format files from :F3: to :F2:. None of the files are renamed.

```
COPY :F3:*.* TO :F2: NCQ
```

copies only non-system, non-format files (files without the F or S attribute set) from :F3: to :F2:. This command copies the files as well as any attributes for those files and prompts before each copy so specific files can be skipped and not copied. Notice the space between :F2: and the options NCQ. Without this space, the system would interpret NCQ as the destination filename.

Appends files

Command Format

```
COPY <source pn 1>,<source pn 2>[,<source pn 3>,...,<source pn n>]

     TO <destination pn>[{B|U}] [P]
```

where

<source pn 1>	specifies the first input file and cannot contain any wildcard characters.
<source pn 2>	specifies the second input file to be appended to the first and cannot contain any wildcard characters.
<source pn 3>	specifies the third input file to be appended to the second and cannot contain any wildcard characters.
<source pn n>	specifies the last input file to be appended and cannot contain any wildcard characters.

TO <destination pn> specifies the output file that contains all the inputs in the order they are entered. The <destination pn> must not be the same as any of the sources. It cannot contain any wildcard characters.

B suppresses the prompt that is normally displayed when the destination file already exists.

<destination pn> ALREADY EXISTS, DELETE?

The existing destination file is deleted and a new destination file is created copied from the source file.

U is the same as B (the prompt is suppressed) except that the existing file is not deleted first. The source file is copied over the existing destination file without deleting the file first.

P is used on single drive systems to copy a file from one disk to another disk using the same drive. The program halts and prompts

LOAD SOURCE DISKETTE, THEN TYPE (CR)

The disk containing the file to be copied is then loaded the and the RETURN key pressed . When it is necessary to switch disks, the system halts and prompts

LOAD OUTPUT DISKETTE, THEN TYPE (CR)

The two prompts alternate until the file is copied. Then, the program halts and prompts

LOAD SYSTEM DISKETTE, THEN TYPE (CR)

to terminate the COPY and return to the operating system. P is assumed as a default and need not be specified if the source name and the destination name are the same, including the device specification.

Comments

This form of the COPY command appends each source file to the previous source file instead of making a copy of a single file as with the previous form of the COPY command.

If a comma (,) appears in the source pathname for the COPY command, it is assumed that the source files is appended to one another in the order in which they are entered.

None of the pathnames can contain wildcard characters. The destination pathname must be different from any of the source files.

The command terminates and returns to the operating system if any source file is missing.

The message

APPENDED (source pathname> TO (destination pathname>

is displayed on the console output device after the copy is complete for each source file.

Examples

The command

COPY FILE1,FILE2,FILE3 TO FILE4

first copies FILE1 to FILE4 and then appends FILE2 and FILE3 to the output in FILE4. All files have a blank extension and are on the physical drive to which :F0: is assigned.

COPY :F2:FILE1,:F4:FILE2,:F3:FILE3 TO :F0:FILE4

performs the same action except that the input files are located on different logical drives.

DELETE

Removes files from the disk

Command Format

DELETE <pathname 1>[pathname 2>,...,<pathname n>] [{Q | P} ...]

where

<pathname 1> thru <pathname n> — specifies the file or files to be removed from the disk. These specifications can contain wildcard characters.

Q — causes the program to display the prompt

<pathname>, DELETE?

for each file before the file is deleted. A response of "Y" or "y" causes the file to be deleted. Otherwise, the file is not deleted and processing continues. Q is used on wildcard deletes to allow a decision on a file by file basis whether or not to delete the file. It can also be used on single file deletes. If a sequence of files is specified, the prompt is only for the last file specification.

P — is used on single drive systems to delete a file from a disk other than the system disk using one drive. The program halts and prompts

LOAD SOURCE DISK, THEN TYPE (CR)

The the disk containing the file to be deleted is then loaded and the RETURN key is pressed . When the delete is done, the program stops and prompts

LOAD SYSTEM DISK, THEN TYPE (CR)

to terminate the delete and return to the operating system.

Comments

As each file is deleted, the message

<pathname>, DELETED

is displayed on the console output device.

If the pathname does not specify a disk file or if the disk file specified does not exist, an error message is displayed on the console output device.

Examples

The command

```
DELETE FILE1.TXT
```

removes FILE1.TXT from the drive to which :F0: is currently assigned.

```
DELETE :F4:PROG?.SRC Q
```

removes all files from :F4: with an extension of .SRC and a filename beginning with the letters PROG and with any character in the fifth position. This command also prompts before deleting any files. Specific files that should not be deleted can be skipped.

For example,

```
:F4:PROGA.SRC, DELETE? Y
:F4:PROGA.SRC, DELETED
:F4:PROGB.SRC, DELETE? Y
:F4:PROGB.SRC, DELETED
:F4:PROGZ.SRC, DELETE? N
```

By responding with a Y to the first two prompts and an N to the third prompt, the first two files are deleted and the third file remains on the disk.

The command

```
DELETE :F4:*.CSD, :F4:*.BAK Q
```

prompts for the :F4:*.BAK files but does not prompt for the :F4:*.CSD files.

DETACH

Releases multimodule row from processor

Command Format

DETACH <multimodule row>

where

<multimodule row> is 0 to detach the first row of multimodules and 1 to detach the second row of multimodules.

Comments

This command only applies to systems with the optional Multimodule Adapter board. The adapter board has four connectors for up to four multimodule boards: two in row 0 which correspond to multimodule connectors J1 and J2 and two in row 1 which correspond to multimodule connectors J3 and J4. See Appendix A for the location of these connectors. Once the row has been attached to a processor, that processor can access either multimodule on the row.

The DETACH command releases the specified row from the processor on which the command runs. See Chapter 9 for information on running commands on the optional processor.

The operating system considers the bubble multimodule as a sharable device like a disk drive. Bubble multimodules need not be attached to or detached from a processor. Attempting to DETACH a multimodule row that contains bubble memory results the following error message being displayed:

61 MODULE ALREADY ASSIGNED TO BUBBLE

Examples

The command

DETACH 0

detaches the multimodules in Row 0 (J1 and J2) from the processor executing the command. See Chapter 9 for more information on dual processing.

DIR

Displays index of disk files

COMMAND FORMAT

DIR [TO <pn>] [FOR <pn>] [<n>] [I] [{ J | K | L }...] [{ F | O | P | Z }...]

where

TO <pn> specifies the device or file pathname to receive the output of the directory listing. The default is the current console device (:CO:). The TO clause may only appear once.

FOR <pn> displays the directory listing only for the file or files specified. The <pn> may contain wildcard characters, thereby specifying more than one file. The <pn> must specify a complete filename; a logical drive specification is not allowed. See the <n> option below for specifying a logical drive. The FOR clause may only appear once.

<n> is a digit from 0 to 5 which specifies the logical disk device being indexed. The default for <n> is 0 for :F0:. Only one drive can be specified. If more than one <n> is entered on the command line, the last one entered is used and the others are ignored. If <n> is specified and the FOR clause also gives a device name, <n> overrides the device given in the FOR clause.

I specifies that files with the Invisible attribute set are to be included in the list with the other files.

{ J | K | L } specifies that only files with the user-defined J, K, or L attribute are to be listed. If J, K, and L are combined, they are ANDed to determine which files are displayed. For example, if J and K are specified, only files with both J and K attributes set are included in the directory listing. See the ATTRIB command for more information on these attributes.

F specifies a Fast listing. Only the filenames, extensions, and the summary line are given.

O specifies a single column listing of files. Otherwise, the output is double column.

P is used on single drive systems to stop the program after it is loaded into memory, allowing another disk to be inserted. The prompt

LOAD SOURCE DISK, THEN TYPE (CR)

is issued. The the disk requiring a directory listing is then loaded and the RETURN key is pressed. When the directory is finished, the system prompts

LOAD SYSTEM DISK, THEN TYPE (CR)

Remove the source disk, insert the system disk, and press the RETURN key.

Z specifies a listing of only the summary line. The information on files is omitted. The Z option overrides other options given.

Comments

The DIR command lists information about the files on a disk to the output device specified in the TO clause. If no device is specified, the current console device (:CO:) is assumed.

Each entry in the directory listing contains the following information:

- Filename and extension
- Number of bytes in the file
- Number of disk blocks allocated to the file
- Attributes currently set for the file

The last line is a summary containing the number of blocks available on the disk and the total number of blocks available on the disk. A directory listing is illustrated in Chapter 4.

Examples

The command

```
DIR 1
```

displays a listing of the files on logical disk :F1: except files with the I attribute set.

The command

```
DIR 1 I
```

performs the same action except that files with the I attribute set are also displayed resulting in a listing of all files, i.e., all files are displayed from the disk.

```
DIR TO :LP:
```

lists the files currently on :F0: on the line printer.

```
DIR FOR :F0:PROG?.SRC JKL
```

displays a list of files on :F0: which meet the following criteria:

- Have the J, K, and L attributes set
- Have an extension of .SRC
- Have a filename beginning with PROG and with any character as the fifth letter.

The command

DIR Z

displays only the summary line for :F0: showing the number of blocks in use on the disk.

DIR 0 OF

lists only the filenames and extensions in single column format for files on :F0:.

ENDJOB

Terminates a file used as console input

Command Format

ENDJOB [<comment>]

where

(comment) is user defined. For example, the comment can contain the job name.

Comments

The ENDJOB command is used to terminate a file currently being used as console input, so that an end of file error (See ISIS error 29 in Appendix B.) does not occur. An end of file error causes a software reset and the operating system is reloaded.

ENDJOB is automatically appended to the end of the file created by the JOB command. It should also be added as the last command of any job file not created by the JOB command. For example, if the user creates a file with the CREDIT text editor to be run by the / command, ENDJOB should be the last command in the file.

The ENDJOB command is an example of an ISIS-PDS command that is always resident in memory. There is no file that corresponds to this command, so it need not be loaded into memory to be run.

Examples

The command

ENDJOB JOB9

terminates the JOB file with the comment showing that the file terminated is JOB9.

ENDJOB

appears as the last command in a JOB file to terminate that file.

HELP

Displays help
information for ISIS-PDS

Command Format

```
     [ { <n>              } ]
HELP [ { <command name>   } ]
     [ { <topic>          } ]
```

where

<n>	is the number, up to three digits, of an ISIS error message.
<command name>	is the name of a valid ISIS command.
<topic>	is a term for which help is available.

Comments

Entering the HELP command without specifying any parameters causes every ISIS-PDS command and topic for which help is available to be displayed.

Entering the command with a valid error number causes the error message to be displayed on the screen.

Entering the command with a valid ISIS command name causes the command format and related information to be displayed on the screen.

Entering the command with a valid topic term causes information about that topic to be displayed on the screen.

Examples

The command

HELP COPY

displays the format and a description of the COPY command.

HELP 29

displays the text of error message 29.

HELP

displays a list of all topics and commands for which help is available.

An example showing the output of the HELP command is given in Chapter 4.

IDISK

Initializes a disk for ISIS-PDS use

Command Format

IDISK :F<n>:<volume id>[.<vol ext>][{P | S} ...]

where

:F<n>:	specifies the logical disk to be initialized.
<volume id>.<vol ext>	specifies the volume name to be assigned to the disk being initialized as a one- to six- character name. The extension (one- to three-characters) is not required. No blank spaces are allowed between the device name and the filename.
P	is the pause option for initializing a disk on a single drive system. The following prompt is displayed LOAD OUTPUT DISKETTE, THEN TYPE (CR) After inserting the disk to be initialized, press the RETURN key. When required, the prompt LOAD SYSTEM DISKETTE, THEN TYPE (CR) is displayed. Place the system disk back in the drive and press the RETURN key to return to the operating system.
S	causes a system disk to be created. The files ISIS.LAB, ISIS.FRE, ISIS.DIR, ISIS.TO, ISIS.PDS, and ISIS.CLI are put on a system disk. Otherwise, a non-system disk is created. When a non-system disk is initialized, ISIS.PDS and ISIS.CLI are omitted.

Comments

Single drive mode is specified by the P option, but is also assumed if :F0: is specified.

A non-system disk can be made into a system disk without reinitializing it by using the COPY command with the C option to copy ISIS.PDS and ISIS.CLI from a system disk. The C option does not copy the F attribute for these two files. The ATTRIB command should be used to assign the F attribute to the files.

To IDISK from a drive other than physical drive 0, first use the ASSIGN command to change the assignment of :F0: to the physical drive desired as the IDISK input. Then, the IDISK command uses the alternate physical drive as the input for files required.

Examples

The command

```
IDISK :F3:NEWVOL S
SYSTEM DISKETTE
```

initializes a system disk on :F3: named NEWVOL. The message SYSTEM DISKETTE indicates that the S option was used.

```
IDISK :F2:VOL2
NON-SYSTEM DISKETTE
```

initializes a disk on :F2: called VOL2 as a non-system disk.

The command

```
IDISK :F0:VOL3
SYSTEM DISKETTE
```

initializes a disk on the logical disk device :F0: assuming the P option. The following message is displayed:

```
LOAD OUTPUT DISKETTE, THEN TYPE (CR)
```

At the end of the initialization, the following message is displayed:

```
LOAD SYSTEM DISKETTE, THEN TYPE (CR)
```

Remove the new system disk and insert the old system disk and press RETURN or simply press RETURN since the newly created disk is also a system disk.

JOB

Batches commands and executes from a file

Command Format

JOB [<jobfile>]

where

<jobfile> is the pathname of the file that contains the commands specified by the user. If no pathname is specified, :F0:JOB<a>.CSD is the default where <a> is A for Processor A and B for Processor B.

Comments

After entering the JOB command, the equal sign prompt (=) appears. Commands can be entered as usual; however, they are saved in memory and are not executed immediately.

There are three ways to end this mode of entering commands that are saved in memory:

- Press the RETURN key two times in a row to save the commands in the file specified by <jobfile>. An ENDJOB command is appended to the last command entered, and the commands are executed from <jobfile> in the order in which they were entered.

- Press CTRL-Z to return to standard input mode and delete any commands already entered. No job file is created.

- Press the ESC key to save the commands previously entered in the file specified by <jobfile>. An ENDJOB command is appended to the last command entered, but the file is not executed. The user is returned to the standard input mode.

If a job file is created, it can be executed later with the ASSIGN command or the / command, or it can be edited first and then executed. If the file is named in the form

JOB<n>.CSD

where <n> is a digit from 0 to 9, the file can be executed as a user defined function. See FUNCT <n>.

If the commands entered overflow the available memory, the effect is the same as if ESC had been typed. The commands are saved in the file specified by <jobfile> with the ENDJOB command appended to the end of the file, and the system is returned to the standard input mode.

The JOB command is ignored if it appears in a command file, i.e., nesting of JOB commands is not allowed. Note that the SUBMIT command allows nesting.

An error occurs if the system cannot open the file after the RETURN or ESC key is pressed and the user is prompted to enter a new filename.

The JOB command is another command that is always in memory. It does not have to be loaded from a disk file to be run.

Examples

The command

```
JOB :F1:JOB1.CSD
```

accepts keyboard input in batch mode to be saved in a file on :F1: named JOB1.CSD.

```
JOB
```

accepts keyboard input in batch mode to be saved in a file on :F0: named JOBA.CSD if run on Processor A and JOBB.CSD if run on Processor B. Processor A is the processor that comes with the system, and Processor B is the optional processor that can be added. See Chapter 9 for more information on dual processing. If the same command is run again, it overwrites the existing JOBA.CSD.

```
JOB JOB9.JOB
```

accepts keyboard input in batch mode to be saved in a file on :F0: named JOB9.JOB.

```
JOB :F4:JOB2
```

accepts keyboard input in batch mode to be saved in a file on :F4: named JOB2 with a blank extension.

The command

```
JOB ABOOT.CSD
```

creates a jobfile on :F0: that is executed every time that Processor A is initialized. A jobfile named BBOOT.CSD is executed every time Processor B is initialized. ABOOT.CSD and BBOOT.CSD must contain an ENDJOB command so that control is returned to the keyboard after the configuration jobfile is run.

RENAME

Changes the filename or extension of a disk file

Command Format

RENAME <old pathname> TO <new pathname>

where

<old pathname>	specifies the old name of the file to be changed. Wildcard characters are not allowed. The file being renamed must not be write protected. Use the ATTRIB command to remove write protection.
<new pathname>	specifies the new name for the file. The <new pathname> must specify the same physical device as the source. Wildcard characters are not allowed.

Comments

The two pathnames must specify the same physical disk device; however, the logical disk device can differ. For example, :F1: can be used for the old pathname and :F2: for the new pathname as long as :F1: and :F2: are assigned to the same physical device. If the new pathname specifies an existing file (new pathname already exists), the program prompt:

<new pathname> ALREADY EXISTS, DELETE?

Type "Y" or "y" to delete the existing file. Otherwise, the program terminates, returning the system prompt.

The RENAME command cannot be used on a non-system disk in single drive systems. To rename a file on a non-system disk in a single drive system, use the COPY command with the P option. Then, use the DELETE command with the P option to remove the original file. An example is given in Chapter 4 for this procedure.

The message

RENAME <old filename> TO <new filename>

is displayed when each file is renamed.

Examples

The command

```
RENAME PROGA.SRC TO PROGB.SRC
RENAMED PROGA.SRC TO PROGB.SRC
```

renames the file PROGA.SRC to PROGB.SRC on :F0:.

```
RENAME :F3:NEWDOC.TXT TO :F3:OLDDOC.TXT
RENAMED :F3:NEWDOC.TXT TO :F3:OLDDOC.TXT
```

renames the file NEWDOC.TXT to OLDDOC.TXT on :F3:.

SERIAL

Configures the 8251 USART and the 8253 Timer for the serial output port

Command Format

There are two formats for the SERIAL command: one for synchronous mode and one for asynchronous mode. For synchronous mode, the format is:

SERIAL S [P=<a> W=<n1> E=0 C=<n3> I=<n4>]

where

S — specifies the synchronous mode of data transfer. P, W, E, C, and I are the only valid parameters in synchronous mode.

P=<a> — specifies the parity. The value of <a> can be

E for even parity
O for odd parity
N for no parity

The default is P=N.

W=<n1> — specifies the word size. The value of <n1> can be

5 for a 5-bit word size
6 for a 6-bit word size
7 for a 7-bit word size
8 for an 8-bit (one byte) word size

The default is W=8.

E=0 — specifies the source of the synchronization character. The value of 0 is for internal synchronization. The iPDS system does not support external synchronization.

C=<n3> — specifies the use of a synchronization character. The value of <n33> can be

0 for double synchronization character
1 for single synchronization character

The default is C=0, and the two default synchronization characters are both 50H. The command prompts for the value of the character or characters to be used for synchronization.

I=<n4> — specifies an 8251 USART command as a numeric value. If no value is specified, the default command is 37H. See the Component Data Catalog for a description of the commands.

For asynchronous mode, the format is:

SERIAL A [P=<a> S=<n1> B=<n2> W=<n3> I=<n4>]

where

A — specifies the asynchronous mode of data transfer. P, S, W, and B are the only valid parameters in asynchronous mode.

P=<a> — specifies the parity. The value of <a> can be:

E for even parity
O for odd parity
N for no parity

The default is P=N.

S=<n1> — specifies the number of stop bits. The value of <n1> can be:

1 for one stop bit
1.5 for one and a half stop bits
2 for two stop bits

The default is S=2.

B=<n2> — specifies the baud rate. The value of <n23> can be:

110 for a 110 baud rate
150 for a 150 baud rate
300 for a 300 baud rate
600 for a 600 baud rate
1200 for a 1200 baud rate
2400 for a 2400 baud rate
4800 for a 4800 baud rate
9600 for a 9600 baud rate
19200 for a 19200 baud rate

The default is B=9600.

W=<n3> — specifies the word size. The value can be:

5 for a 5-bit word size
6 for a 6-bit word size
7 for a 7-bit word size
8 for an 8-bit (one byte) word size

The default is W=8.

I=<n4> — specifies an 8251 USART command as a hexadecimal value. If no value is specified, the default command is 37H. See the Component Data Catalog for a description of the commands.

Comments

The SERIAL command programs the 8251 USART and the 8253 Timer chips for synchronous or asynchronous transmission or reception of data. The user should understand the 8251 and the 8253 before using this command. Refer to the current edition of the Intel Component Data Catalog for further information on the 8251 and the 8253.

In some applications, the SERIAL command could be in the ABOOT.CSD file so that the USART is automatically configured whenever the system is initialized. Two ASSIGN commands could also appear to assign the ISIS-PDS console device (:CO: and :CI:) to :SO: and :SI:.

Any parameters that are left off the command line take on the default values as indicated previously.

Numeric values for the B and I parameters and for synchronization characters can be entered in any number base: binary (B), octal (O or Q), decimal (D), or hexadecimal (H) by appending the base suffix shown in parentheses to the numeric value. Hexadecimal values beginning with the digits A to F should be entered with leading zeroes. The default base, if no suffix is appended, is decimal.

Examples

The command

```
SERIAL S P=E W=5 E=0 C=0
```

configures the serial port to synchronous mode with even parity, a word length of 5 bits, external synchronization on output, and two bytes used for synchronization. The command prompts twice for the two synchronization characters to use. If C=1 were specified, only one prompt would be given.

```
INPUT SYNC CHAR (NUMERICAL VALUE) => 20H
INPUT SYNC CHAR (NUMERICAL VALUE) => 20H
```

The ASCII code for the space character (20H) is entered for both characters.

The command

```
SERIAL A P=N S=2 W=8 B=1200
```

configures the port for asynchronous mode, with no parity, 2 stop bits, a word length of 8 bits, and a baud rate of 1200.

SUBMIT

Executes commands
from a disk file

Command Format

SUBMIT <pn> [(<parameter 0>,<parameter 1>,...,<parameter 9>)]

where

<pn>	is the pathname of the file containing the command lines to be executed automatically. If no extension is supplied, SUBMIT assumes an extension of .CSD. A file with a blank extension can be used by typing the filename followed by a period (.). More details on the content of this file follow.
<parameter 0> thru <parameter 9>	are the values (up to 31 characters) assigned to formal parameters in the command file. Parameters are discussed in more detail in the following section.

Comments

To run the SUBMIT command in the most straightforward way:

1. First, use CREDIT to create a text file containing an operating system command on each text line. The command should be typed exactly as it would normally be entered from the keyboard including the RETURN key at the end of the line. If any keyboard responses are expected by the command, these should also be entered in the order they would normally occur.

2. Type the SUBMIT command line with the pathname of the text file containing the commands. The commands in the text file are run in the order in which they appear in the file.

The following sections describe the operation of the SUBMIT command in more detail.

Chapter 4 contains an example of the SUBMIT command and the JOB command where both are compared.

The Input File

The input file specified in the SUBMIT command line defines the sequence of ISIS-PDS commands to be executed. It is referred to as the Command Sequence Definition file and has a default extension of .CSD. Some ISIS commands have restrictions when used in a .CSD file. For example, the DEBUG command cannot be run under SUBMIT.

The .CSD file must contain commands in the exact sequence they are to run. The commands cannot be out of order. Also, any keyboard responses required by a specific command must follow that command line and must be in the order expected by that command.

The commands in the .CSD file may contain placeholders which name constant or variable values, i.e., formal parameters.

The SUBMIT command reads the .CSD file specified in the command line and copies it to a temporary file substituting the actual values of parameters. This file is referred to as the Command Sequence file and has the same filename as the .CSD file with an extension of .CS. The .CS file is deleted when the SUBMIT command is finished.

Parameters

The SUBMIT command allows up to 10 formal parameters to appear in the .CSD file. Each formal parameter appears in place of an actual value of the form:

%<n>

where <n> is a digit 0 to 9 and no spaces separate the % and the digit.

The actual parameters (the values for the formal parameters) are specified in the SUBMIT command line as <parameter 0> through <parameter 9>. Each actual parameter can be up to 31 characters. The actual parameters are enclosed in parentheses and are separated from one another by a comma.

To use a comma, a space, or a parentheses in a parameter, enclose the parameter in single quotes. For example,

('DATE: May 15, 1981','TIME: 10:00')

The two spaces after DATE:, the space after MAY, the comma and space after 15 are treated as part of the parameter while the comma after 1981' is a delimiter and separates the first parameter from the second. The value that appears in the .CS file for the first formal parameter is:

DATE: May 15, 1981

The value that appears in the .CS file for the second formal parameter is:

TIME: 10:00

To use quotes in a parameter, type a pair of extra single quotes. One single quote appears as part of the parameter.

For example, the parameter:

'TIME'

should be entered as:

'''TIME'''

To skip one of the formal parameters and supply no actual value for it, type two adjacent commas in the parameter list on the SUBMIT command line.

For example, assume that the file COPY.CSD, the input to the SUBMIT command, contains the following:

```
ATTRIB :F1:%0 WO
DELETE :F1:%0
COPY %0 TO :F1:%0
ATTRIB :F1:%0 W1
```

This example assumes that :F0: and :F1: are assigned to different physical drives. A SUBMIT command line to execute this file is:

```
SUBMIT COPY(PROGA)
```

The file COPY.CS, created by the SUBMIT program, would then contain:

```
ATTRIB :F1:PROGA WO
DELETE :F1:PROGA
COPY PROGA TO :F1:PROGA
ATTRIB :F1:PROGA W1
:FO:SUBMIT RESTORE :FO:COPY.CS (:VI:)
```

The last command in the sequence, inserted by SUBMIT at run time, terminates the SUBMIT command and returns to keyboard input mode. It is discussed in following sections.

Interactive Usage

If CTRL-E appears in a .CSD file, the current console input device (the .CSD file) is changed to the keyboard. Then, the user may enter input from the keyboard. Typing CTRL-E at the keyboard restarts the SUBMIT command sequence.

To enter CTRL-E in the .CSD file, either the literalizing feature or the hexadecimal entry feature of the CREDIT text editor must be used. See the *ISIS CREDIT™ CRT-Based Text Editor User's Guide* for instructions on entering characters literally or entering hexadecimal values. The hexadecimal value for CTRL-E is 05H.

Do not edit the .CSD file when entering data at the keyboard during a CTRL-E portion of SUBMIT.

Example

Assume that the input file to SUBMIT contains the following command lines:

```
CREDIT :F3:PROGA.SRC
CTRL-E
ASM80 :F3:PROGA.SRC
LOCATE :F3:PROGA.OBJ
:F3:PROGA
```

The SUBMIT command begins executing commands read from the file. When the CTRL-E is read, the SUBMIT program stops reading the file for input and switches to keyboard input. The user can interactively edit the source program text file at this time. Only command line editing is allowed when the CREDIT text editor is run under SUBMIT. After the editing session is ended, the user enters CTRL-E at the keyboard to switch back to reading the file for operating system input. SUBMIT resumes reading at the third line and assembles, locates, and runs the program.

Advanced Usage

This section describes how SUBMIT processes the input .CSD file. Because of the way that SUBMIT processes the command file, SUBMIT commands can be nested to any level in the .CSD file. Processing of a nested .CSD file is illustrated in the example.

In general, the steps in executing a .CSD file are:

1. SUBMIT creates a .CS file and copies the .CSD file to it, substituting actual values for formal parameters.

2. SUBMIT generates a special command (a RESTORE form of the SUBMIT command) for the end of the .CS file that points to the most previous console input device. Thus, when SUBMIT finishes processing the .CS file, it can return either to the keyboard or to the correct line of the next higher nested SUBMIT file.

These two steps are described in detail in the rest of this section.

When SUBMIT first begins, it copies the .CSD file, the input file created by the user, to the .CS file substituting actual parameters for formal parameters. Also, when creating the .CS file, SUBMIT attaches a command at the end of the file of the form:

SUBMIT RESTORE <pn) (<previous input>[,<block>,<byte)])

where

<pn> is the pathname of the current console input device, i.e., the name of the current .CS file.

(<previous input>) is the pathname of the previous console input device enclosed in parentheses. If SUBMIT was run from the keyboard, this would be :VI: to return to the keyboard for further input. If SUBMIT was nested (run from another .CS file), the previous input would be the pathname of the .CS file that invoked SUBMIT.

,<block> is only specified if the previous input is a .CS file. The <block) identifies the block in the previous .CS file where SUBMIT left off. Thus, SUBMIT can return to the command following the nested SUBMIT command. Blocks are 128 bytes each starting at block 0.

,<byte> is only specified if the previous input is a .CS file. The <byte3> identifies the byte within the block specified where SUBMIT left off. Thus, SUBMIT can return to the command following the nested SUBMIT command. The byte count starts at byte 0 to byte 127 for each block.

When the RESTORE version of the SUBMIT command is executed from a .CS file, that .CS file is deleted, and control returns to the previous console input device.

The execution sequence of nested SUBMIT files is summarized as follows:

1. Read the SUBMIT command line nested in the current .CS file.

2. Create the .CS file for the nested SUBMIT .CSD file substituting actual values for formal parameters.

3. Place the RESTORE version of the SUBMIT command at the end of the new .CS file to return to the command after the nested SUBMIT.

4. Change the console input to the new .CS file.

5. Execute the new .CS file until the RESTORE version of the SUBMIT command is run.

6. Delete the new .CS file and return to the old .CS file to finish execution.

The highest level of the .CS file returns control to the keyboard by running the RESTORE version of SUBMIT with :VI: as the previous input.

Normally, the RESTORE version is generated by the SUBMIT command and is never entered by the user. However, if a SUBMIT command is terminated early because of an error, it can be restarted after the error is corrected by entering the RESTORE version of the command from the keyboard. The user must calculate the correct block and byte of the .CS file to continue executing in the correct place.

For example, if the SUBMIT command terminates because the drive specified for a file to be copied is offline, the command can be restarted at the COPY command after the drive is made available.

The following is an example of processing a nested SUBMIT command. For this example, the file COPY.CSD from the example in the section "Parameters" is used. This example assumes that :F0: and :F1: are ASSIGNed to different physical devices. It contains the following text:

```
ATTRIB :F1:%0 WO
DELETE :F1:%0
COPY %0 TO :F1:%0
ATTRIB :F1:%0 W1
```

To execute this .CSD file, a SUBMIT command must be issued passing the actual value of the formal parameter %0. For this purpose, the file BACKUP.CSD is created containing the following two SUBMIT commands:

```
SUBMIT COPY(PROGA)
SUBMIT COPY(PROGB)
```

The SUBMIT command to start execution is:

```
SUBMIT BACKUP
```

The .CSD extension is assumed in this command line. The SUBMIT command creates the file BACKUP.CS containing the following text:

```
SUBMIT COPY(PROGA)
SUBMIT COPY(PROGB)
:F0:SUBMIT RESTORE :F0:BACKUP.CS(:VI:)
```

The first SUBMIT command is run creating the file COPY.CS as follows:

```
ATTRIB :F1:PROGA W0
DELETE :F1:PROGA
COPY PROGA TO :F1:PROGA
ATTRIB :F1:PROGA W1
:F0:SUBMIT RESTORE :F0:COPY.CS(:F0:BACKUP.CS,0,19)
```

When the last command in this file (the RESTORE version of SUBMIT) is run, COPY.CS is deleted and control returns to the 19th byte in block 0 of the file BACKUP.CS, i.e., the second SUBMIT command in the file BACKUP.CS. SUBMIT calculates the correct byte and block for return and generates the RESTORE version of the SUBMIT command.

The second SUBMIT command creates a second COPY.CS file as follows:

```
ATTRIB :F1:PROGB W0
DELETE :F1:PROGB
COPY PROGB TO :F1:PROGB
ATTRIB :F1:PROGB W1
:F0:SUBMIT RESTORE :F0:COPY.CS(:F0:BACKUP.CS,0,39)
```

When the last command in this file is run, COPY.CS is deleted again and control returns to the 39th byte in block 0 of the file BACKUP.CS, i.e., the third command line in the file BACKUP.CS:

```
:F0:SUBMIT RESTORE :F0:BACKUP.CS(:VI:)
```

The third command is the RESTORE version of SUBMIT for the highest level of nesting. It returns control to the keyboard, :VI:.

Technical Information

Any program running under the ISIS-PDS operating system and receiving its input from the :CI: device can be run by SUBMIT, if there is sufficient buffer space for all the open files.

The ISIS-PDS operating system allows six disk files to be open at a time. Each open file requires two to three 128-byte buffers from the user's memory addresses 3180H to 3980H. See Chapter 8 for a detailed discussion of buffer space requirements.

Regardless of the number of nested SUBMITs, the SUBMIT command itself requires only one open file at a time and, thus, 128 bytes from the user's buffer space. However, some programs such as CREDIT or IPPS can open additional files. When running these programs under SUBMIT, the six open file limit can be exceeded creating an error.

Error Messages

The SUBMIT command produces three error messages in addition to those produced by the ISIS-PDS operating system. All three are fatal errors.

ILLEGAL SUBMIT PARAMETER — occurs when the parameter contains illegal characters. For example, parameters are not enclosed in single quotes when the quotes are required.

ARGUMENT TOO LONG	occurs when the parameter on the command line is longer than 31 characters.
TOO MANY PARAMETERS	occurs when more than 10 values are specified on the command line.

In addition, the use of the pause option (P) for the COPY, DIR, and DELETE commands is prohibited when these commands are run from a SUBMIT file. No error or warning message is issued; however, use of the option can destroy files on one or more of the disks.

Examples

The command

```
SUBMIT SRCOBJ(PROGA)
```

runs the commands in file SRCOBJ.CSD on :F0:. The first parameter in this file takes on the value PROGA.

See Chapter 4 for an example of developing and creating a SUBMIT file.

?

Displays the version
of the command line interpreter

Format

```
?
```

Comments

The ? command displays the version number of the Command Line Interpreter (CLI). The display is of the form:

```
CLI Vn.m
```

where n.m is replaced by the actual version number.

The ? command is always present in memory. It does not correspond to a disk file containing the ? program like the COPY command.

The ? command must be followed by the RETURN key.

Examples

If the 1.0 version of the CLI is running the display is:

```
?
CLI V1.0
```

Displays the contents of a file on the screen

Command Format

@ <pathname> [4]

where

<pathname>	specifies a file to be displayed on the screen.
4	specifies that tabs in the text file is displayed as 4 spaces instead of 8.

Comments

After entering the command, the first 19 lines of the file are output to the screen and a pause occurs. Any of the following characters can then be entered at the keyboard:

P	switches to page mode and continues. The file is displayed 21 lines at a time. Pressing any character causes the next 21 lines to be displayed and then halt.
S	switches to slow scroll mode and continues. The file is displayed continuously scrolling at a slow speed.
F	switches to fast scroll mode and continues. The file is displayed continuously scrolling at a fast speed.
E	exits back to the operating system.
L	switches to line-by-line mode and continues. The file is displayed a line at a time, pausing after each line. Press any character to continue.
B	backs up 1024 characters and continues. The B command can be pressed repetitively to return to the beginning of the file.
Z	prints the last 1K bytes of the file, sets the mode to F, and halts.
CTRL-S	pauses the display. Press any character to continue.
<a>	any other character continues the display after a halt from a CTRL-S, the end of a page on Page mode, or the end of a line on Line mode.

Any commands can be entered at any time, even during a paused display.

When the end of the file is reached, the E command must be entered to exit back to the operating system.

Lines longer than 77 characters automatically wrap around to the next physical line on the console output device.

The @ command is an example of a command that is always present in memory. There is no corresponding file containing the @ program that must be loaded into memory to run the @ command.

Examples

The command

```
@MYDOC.TXT
```

displays the contents of the file MYDOC.TXT.

```
@PROG.SCR 4
```

displays the contents of PROG.SRC with any tabs in the file displayed as 4 spaces.

/

Assigns a file as the console input device

Command Format

```
/<pathname>
```

where

<pathname> is the pathname of a jobfile or device to be used as console input device. If no extension is specified, .CSD is assumed. If <filename> is followed by a dot (.) but no extension follows, the dot is ignored and <filename> with a blank extension is used.

Comments

The / command is a shorthand form of the ASSIGN :CI: TO <pathname> command. It can be used to change the assignment of the console input device to a jobfile or to a device. The jobfile can be a command file created with the JOB command or with the CREDIT text editor and with the ENDJOB command at the end. The / command is always present in memory. There is no corresponding / file to be loaded to run the / command.

The last command in the jobfile should be the ENDJOB command. Otherwise, an ISIS error 29 is generated and the system is reinitialized.

The / command can appear in a SUBMIT file or a JOB file, but, after the / command is complete, the SUBMIT or JOB command is not resumed.

If a CTRL-E appears in the command file, input switches to the keyboard. Type CTRL-E to resume input from the file.

The / command is faster than SUBMIT because it does not have to create an intermediate file as SUBMIT does. An intermediate file is not required because parameters cannot be passed.

Examples

The command

```
/:F1:CMDFIL
```

takes the console input from the file CMDFIL.CSD on :F1: instead of from the keyboard.

The command

```
/:F3:CMDFIL.
```

takes the input from the file on :F3: with the name CMDFIL and with a blank extension.

The command

```
/:SI:
```

switches the input to the serial input device.

See Chapter 4 for examples of the SUBMIT command, the JOB command, and the / command.

# #

Re-assigns console output
to the CRT screen

Command Format

```
#
```

Comments

The # command is a shorthand form of the ASSIGN :CO: TO :VO: command. It restores the console output device to :VO: which is the CRT display screen. This command is used after the ASSIGN command has assigned the console output to some other physical device. The # command is always present in memory and does not correspond to a file that must be loaded to run the command.

The # command must be followed by a RETURN.

Examples

The command

```
#
```

switches the console output from a file, the printer, or some other output device back to the CRT screen.

■

Fast single line
SUBMIT command

Command Format

.<pathname> [(<parameter 0>,<parameter 1>,. . .,<parameter 9>)]

where

<pathname>	is the pathname of a jobfile containing a single command line. The default extension is .CSD. If a filename is followed by a dot (.) but no extension is specified, the filename with a blank extension is used. More details on the content of this file are described in the SUBMIT command.
<parameter 0> thru <parameter 9>	are the values (up to 31 characters) assigned to formal parameters in the .CSD file.

Comments

The . command reads a single line from the .CSD file, substitutes actual values for any formal parameters in the file, and executes the resulting command. Only 122 characters for the command line are allowed after all substitutions are made. The . command operates the same as SUBMIT except that no intermediate file (.CS file) is created, only one command is read and executed, and the command file may contain a blank extension. All substitutions for formal parameters are made in memory. Thus, the . command is faster than SUBMIT for a single command line. Nesting of . commands is not allowed.

See the SUBMIT command for further details on parameters.

Examples

The command

```
.:F1:CMDFIL (15,25)
```

performs the SUBMIT command with CMDFIL.CSD on :F1: as the job file. The value 15 is substituted for any %0 formal parameters, and 25 is substituted for any %1 formal parameters.

```
.:F1:CMDFIL. (:F2:CMD.FIL)
```

performs the SUBMIT command with the file CMDFIL on :F1: as the job file, substituting the value :F2:CMD.FIL for all occurrences of the formal parameter %0.

FUNCT <n>

Assigns the file JOB<n>.CSD
as the console input device

Command Format

FUNCT-<n>

where

<n> is a digit from 0 to 9. The digit specifies the JOB file to be used as console input.

Comments

FUNCT 0 through FUNCT 9 are user defined function keys. Typing a digit from 0 to 9 while holding down the FUNCT key causes the file named JOB<n>.CSD to be used as the console input. The file should contain operating system commands. The operation of the function keys is similar to the operation of the / command except that a default pathname is used for the input file and it need not be specified.

Pressing the number <n> followed by the RETURN key is the same as pressing FUNCT <n>.

The job file must have been previously created with CREDIT or with the JOB command. If CREDIT is used to create the file, the last command in the file must be ENDJOB. If the JOB command is used to create the file, the ENDJOB command is automatically appended to the file by the JOB command.

See the JOB command for further information on creating job files.

Examples

The command

FUNCT 0

switches the console input to the file JOB0.CSD on :F0:. The number sign and digit 0 are displayed on the screen (#0).

FUNCT 1

switches the console input to the file JOB1.CSD on :F0:, the system default logical disk. The number sign and the digit 1 are displayed on the screen (#1).

ESC

Re-edits and re-executes
the most previous command line

Command Format

ESC

Comments

Instead of entering a command line at the operating system prompt, the ESC key can be pressed to edit the most recent command line. The ESC key can also be pressed during a command line entry to edit the command line entered so far. Then, the entire command line is displayed as it is stored in the command line interpreter line editing buffer. The following keys can be used to modify and re-execute the command line.

CTRL-A	CTRL-A inserts any number of characters before the current cursor position. Pressing CTRL-A the first time enters insert mode. Then, any characters typed are inserted before the cursor. Pressing CTRL-A a second time ends the insert.
CTRL-B	CTRL-B moves the cursor to the beginning of the line.
CTRL-D	CTRL-D deletes the character at the current cursor position unless the cursor is at the end of the line. Then, the character preceding the end of the line is deleted.
CTRL-L	CTRL-L moves the cursor to the end of the line.
CTRL-X	CTRL-X terminates the re-edit without executing the command line and returns to ISIS for another command.
ESC	Press ESC a second time to execute the entire command line.
RETURN	Press RETURN to execute the command line up to the current cursor position.
RUBOUT	Pressing the RUBOUT key is the same as pressing CTRL-D.
<—	The left arrow, cursor control key moves the cursor to the left.
—>	The right arrow, cursor control key moves the cursor to the right.

Only command lines of six or more characters, including spaces, are saved for re-editing.

The ESC key can also be used to repeatedly execute a command.

Examples

Entering the command line:

RENAME MYFILE.TXT TO OLDFIL.TXT

After this command has run, typing the ESC key causes the command line to be displayed for editing and re-execution.

ESC
RENAME MYFILE.TXT TO OLDFIL.TXT

Then, the following editing steps can be taken to change the command and re-execute it.

1. Use the left arrow to move the cursor to the M of MYFILE.TXT.
2. Press CTRL-A and type:

   :F1:

3. Press CTRL-A to complete the insert.
4. Use the right arrow to move the cursor to the O of OLDFIL.TXT.
5. Press CTRL-A and type:

   :F1:

6. Press CTRL-A to complete the insert.
7. Press CTRL-D three times to delete the OLD of OLDFIL.TXT.
8. Press CTRL-L to move the cursor to the end of the line.
9. Press the ESC or RETURN key to execute the entire command line. If step 8 were not done, the ESC key would execute the entire command line and the RETURN key would execute the command line down to the :F1: of the destination file resulting in an invalid command.

CHAPTER 6
TEXT EDITING

Introduction

A text editor is a program that aids in creating and modifying text files. Text files are files containing alphanumeric characters, i.e., each byte in the file is interpreted as a character according to the ASCII code. The byte values and corresponding characters for ASCII codes are in Appendix C.

With the CREDIT text editor, text is entered by typing characters at the keyboard. The text is stored in a file that can later be modified by CREDIT editing commands or can be processed by other commands. For example, when the text file contains the source code for a program, a language translator can process it to create machine code.

A tutorial session illustrating the use of text editing is given in Chapter 4.

Getting Started with the CREDIT™ Text Editor

This chapter includes information, specific to the iPDS system, for using the CREDIT editor. The *ISIS CREDIT™ CRT-Based Text Editor User's Guide,* order number 9800902 describes the simplified, intermediate, general, and advanced command formats. Screen mode and command mode editing functions, CREDIT text editor features, and screen and command line mode editing commands are covered. Included in the CREDIT manual is, creating macros with macro commands, entering and correcting editing commands, using delimiters, and tutorial sessions illustrating all aspects of editing using the CREDIT text editor.

Screen Mode Features

The CREDIT™ Display

The iPDS screen displays 24 lines. When the command line has been entered, the screen clears and divides it into two parts as illustrated in figure 6-1. In screen mode, the bottom 20 lines, called the text area, display text from the file. In the remaining lines at the top, the sign-on message, error messages, and status messages are displayed. In screen mode, all operations are performed in the text area, and the message area at the top of the screen not accessible to the user. The text area and message area are separated by a line of five dashes.

In the screen mode, any ASCII code with an associated character is displayed on the screen as that character. A code with no associated character is displayed as an up arrow (↑). Codes displayed as an up arrow can be pointed to with the cursor and replaced or deleted like any other character.

The end of the file is displayed as a vertical bar (|).

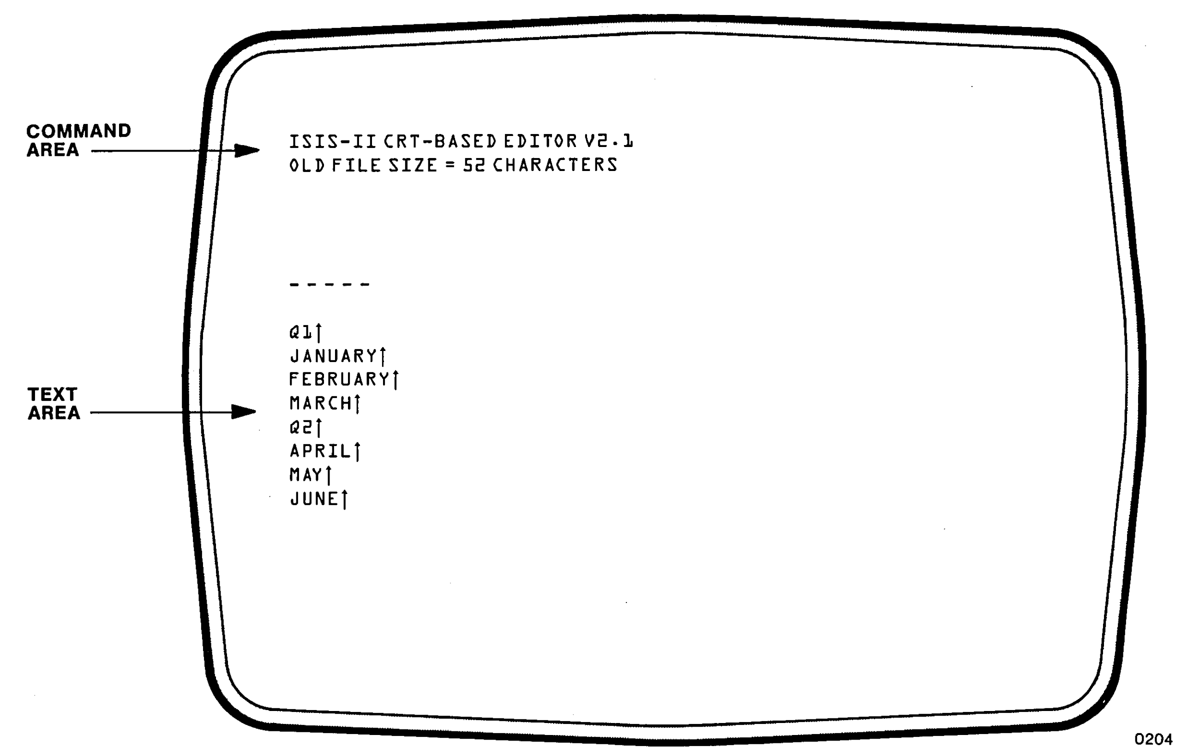
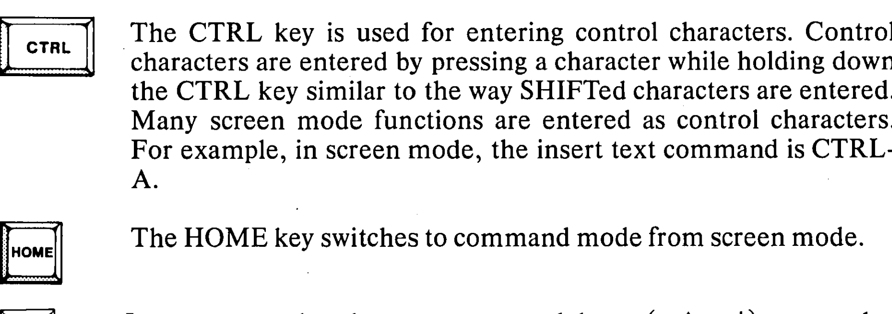


Figure 6-1 The CREDIT™ Display

The Keyboard

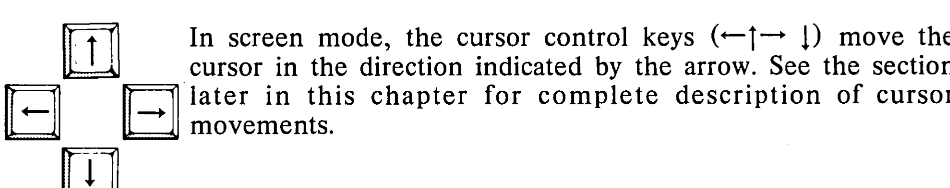
All characters typed at the keyboard are read by the editor. However, some of these characters have a special meaning. They do not simply represent text data, and are not written to the text file. When editing in screen mode, text is entered through the development system keyboard to be saved in a disk file. Commands are also entered through the keyboard, but are not saved in the file.

Figure 6-2 shows the keyboard. Character codes generated by the keys are interpreted as ASCII codes by the editor. The keys that perform special functions and are not normally entered as data into the file are listed below.



The CTRL key is used for entering control characters. Control characters are entered by pressing a character while holding down the CTRL key similar to the way SHIFTed characters are entered. Many screen mode functions are entered as control characters. For example, in screen mode, the insert text command is CTRL-A.

The HOME key switches to command mode from screen mode.



In screen mode, the cursor control keys (←↑→ ↓) move the cursor in the direction indicated by the arrow. See the section later in this chapter for complete description of cursor movements.

ESC

In either screen or command mode, the ESC key terminates commands. When ESC is pressed, <BREAK> is displayed in the message area of the screen.

RUB OUT

The RUBOUT key deletes the previously entered character when inserting text in screen mode. Otherwise, the RUBOUT key moves the cursor one position to the left without deleting the character.

TAB

The TAB key positions the cursor to the next tab set on the line. Its operation is similar to a typewriter tab. The default for tab settings is every 8 characters. Tab settings can be changed using the Alter command described in the section "Advanced Editing Techniques" in the *ISIS CREDIT™ CRT-Based Text Editor User's Guide.*

\

The backslash (\) is the default literalizing character. It allows characters that normally perform some function to be entered into the file as data (literalized) instead. The character following the backslash is taken as data. The backslash itself can be entered as data in a file by typing two backslashes in a row. The second one is literalized and is entered as data in the file. The literalizing character can be changed from the backslash to any other character by using the Alter command. See the section on "Alter Commands" in the *ISIS CREDIT™ CRT-Based Text Editor User's Guide.*

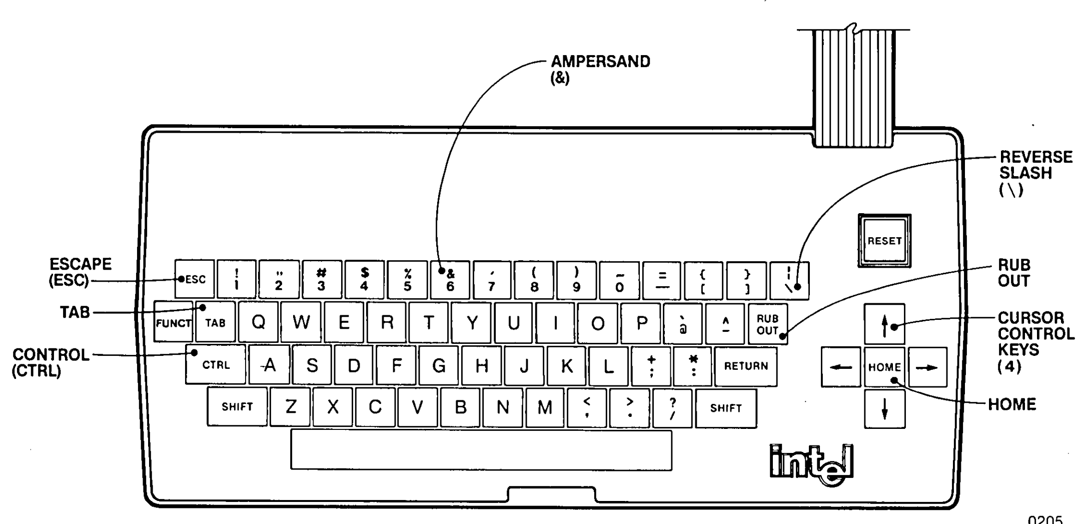


Figure 6-2 The Keyboard

All other characters are accepted as data and are entered in the text file, or they are invalid. Any invalid character causes a warning beeper to sound and no action to occur. An example of an invalid character is a command character entered in the middle of an insert or delete command.

The RETURN key is accepted as data and is entered into the file as a pair of characters, and it also acts as a line terminator. A line of text consists of a character string terminated by a carriage return-linefeed. This pair of characters, called the line terminator, is entered in the file as a 0DH and 0AH when the RETURN key is pressed. Lines are not limited to 80 characters (the width of the display), but it is generally easier to work with a file when each text line fits on a display line.

The line terminator is displayed as one character on the screen, the up arrow (↑). Most screen editing functions treat the terminator as one character.

The Cursor

The CREDIT editor maintains a pointer that marks a character in the text file. Changes are made relative to this pointer. For example, deleting a character erases the character designated by the pointer. Insertions are made immediately preceding the pointer.

In the screen editing mode, the cursor, the reverse video block, reflects the current position of the pointer.

In screen mode, when the cursor is pointing to an area of the screen that does not contain any characters, no edit commands are accepted. The warning beeper sounds when an attempt is made to enter commands with the cursor pointing to an area containing no characters. However, the CTRL-Z command to delete characters can be completed with the cursor pointing to an area containing no characters.

There are no characters between the line terminator and the next line or beyond the end of file marker.

Command Mode Features

The CREDIT™ Display

When the command line is first entered under the ISIS-PDS operating system, the CREDIT text editor clears the screen and divides it into two parts as shown previously in figure 6-1. The CREDIT editor initially enters screen mode. Pressing the HOME key switches to command line mode.

In command line mode, the top area of the screen, called the command area, is the only area accessed by the user. In fact, the text area is erased as commands are entered. The asterisk prompt displayed in the command area indicates that a command can be entered.

When the command line mode is first entered, the text area contains the residual display of the file left over from previous screen editing operations. As commands are entered at the keyboard, they are displayed in the command area. As soon as the command area exceeds the top three lines, the entire text area is erased allowing commands to fill the screen. Once the screen is full of commands, it scrolls up one line at a time as new commands are entered.

In the command line mode, ASCII codes with an associated graphics character are displayed as an up arrow (↑). The up arrow character is displayed as two up arrows (↑↑) to distinguish it from codes with no associated graphics character.

The text area is not used in command line mode.

The Keyboard

When editing in command line mode, commands are entered at the development system keyboard to indirectly modify the text in a file. Data to be added to the text file is entered as a parameter to a command. Data is not directly entered into the file as in screen mode.

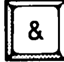
Figure 6-2 shows the keyboard. Some of the keys perform special functions in the command line mode of editing as listed below.

The CTRL key is used for entering control characters. Control characters are entered by pressing a key while holding down the CTRL key. Some commands are entered using control keys. For example, CTRL-V switches from command line editing to screen editing.

In either mode, the ESC key aborts commands. When ESC is pressed, <BREAK> is displayed in the command area of the screen.

The RUBOUT key deletes the previous character when in command line mode.

Many of the commands in command line mode require a string of characters as a parameter. The string of characters must be delimited by a valid delimiter character. CTRL-B is a special delimiter character that causes the string to be interpreted as hexadecimal values rather than as ASCII codes. This character is discussed in the *ISIS CREDIT™ CRT-Based Text Editor User's Guide.*



The ampersand is used as a continuation character for command lines. This character is discussed in the *ISIS CREDIT™ CRT-Based Text Editor User's Guide.*

The semicolon is used to separate multiple commands entered on a single command line. This character is discussed in the *ISIS CREDIT™ CRT-Based Text Editor User's Guide.*

The HOME key, the cursor control keys, the TAB key, and the backslash key do not perform any special command line editing function.

In command line mode as in the screen mode, the RETURN key is entered as two characters in the file (carriage return, 0DH, and linefeed, 0AH).

Disk File Use

The CREDIT editor stores text in disk files and loads the text into memory for editing. Usually, only a part of the file is loaded into memory at a given time, since the entire file usually does not fit in memory. Often, files in addition to the one containing the text are needed during an editing session. These files are temporary files created by the editor, backup files created by the editor, and files used by different CREDIT commands. See figure 6-3

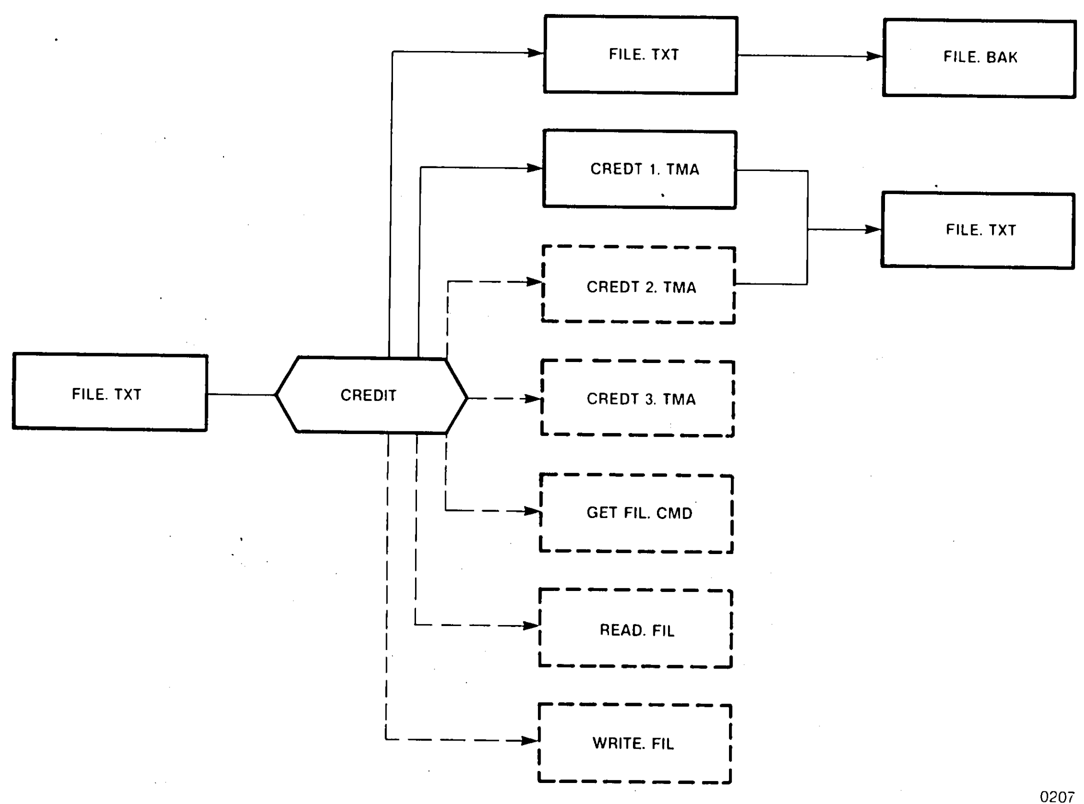


Figure 6-3 Disk File Use

Temporary Files

In addition to the old edit file containing the source text data, the following temporary files are created by the text editor during an editing session:

- An output file called CREDT1.TMA contains the modified text data during the editing session. When the session is ended with the EXIT command, CREDT1.TMA is renamed. When no name is supplied on the CREDIT command line (as part of the TO clause), the old file is renamed with the extension of .BAK, and CREDT1.TMA is renamed to the old edit file. When the TO clause is supplied, CREDT1.TMA is renamed to the file specified as part of the TO clause on the command line.

- A temporary file called CREDT2.TMA is created only when a part of the file no longer resident in memory is edited.

- A temporary file called CREDT3.TMA may also be created to store the modified text data during an editing session.

CREDT1.TMA, CREDT2.TMA, and CREDT3.TMA are reserved filenames and should not be assigned to files. When the TO clause is used on the CREDIT command line, they are created on the same drive as the file specified in the TO clause. Otherwise, they are created on the same drive as the old file being edited. None of the temporary files appear in the directory unless the operating system is reloaded (for example, when RESET is pressed) before the editing session is terminated. However, the temporary files can be viewed in the directory on a dual processing system by running the DIR command on one processor while an editing session is in progress on the other processor. See figure 6-3.

Backup Files

When an existing file is edited and no TO clause is specified, it is renamed with the same filename and an extension of .BAK when the editing session is ended with the EX command. Thus, after changes are made, the previous version of the file is still available.

When a backup file already exists from previous editing, it is automatically deleted and replaced by the version of the file prior to the current editing session. See figure 6-3.

Several rules must be followed to successfully use the backup feature:

- The .BAK version of the file should not be deleted.
- The .BAK version of the file should not be edited.
- The .BAK version of the file should not be write protected; don't set the write attribute to 1.

If the .BAK version is deleted, no backup is available.

If the .BAK version is edited, the changes made are not reflected in the original; the original version is copied to the backup version, not vice versa. The first time the original is accessed through the editor, the .BAK version is replaced by the current version, wiping out any changes made in the backup file.

If the .BAK version is write protected, the editing session cannot be ended with the EX command unless a filename other than the source file for the output is specified. Only EQ or EX with a filename parameter is accepted. When EQ is used, all changes from the editing session are lost.

Files Used by CREDIT™ Commands

In command line mode, the XC and XM commands use a temporary file named CREDT3.TMA. Some of the advanced CREDIT commands use additional files. The section on "Advanced Editing Techniques" in the *ISIS CREDIT™ CRT-Based Text Editor User's Guide* describes file use in more detail. See figure 6-3 for an illustration of disk file use by the CREDIT editor.

Limits on Disk File Use

The ISIS-PDS operating system allows a maximum of six files to be open at any one time. This leaves three files for user applications after allowing for the three files that the CREDIT editor can open. Normally, this number is not exceeded. However, the user should exercise judgment in opening files for access. Files should be closed when not being accessed.

Editing under the control of the SUBMIT program uses one of the available user files. SUBMIT file requirements must be considered when using the CREDIT editor with SUBMIT. When more than six files are opened at a time, a fatal error occurs, and the iPDS operating system is re-initialized.

Performance and File Size

The size for CREDIT files is limited only by the storage device. There must be enough space available on the diskette or bubble to hold the file, the backup file, and the temporary files that the editor uses. The free space on the disk must be two times as great as the size of the file being edited.

The CREDIT text editor works best when files are restricted to 20K bytes or less (the size of the text buffer in memory). Files less than 20K bytes can be loaded into memory, and all editing functions can be performed in memory with a minimum of disk accesses. A file with 20K bytes is about four 8 1/2 x 11 pages of written text.

CMACRO.MAC

The CMACRO File

The file named CMACRO.MAC contains several macro definitions provided with the CREDIT editor to aid in editing text. The macros are described briefly in this section. Some of the macros are illustrated in the previous editing examples. They are listed according to the following functional areas:

- Cursor Movement Macros
- Text Control Macros
- Block Transfer Macros
- File Formatting Macros
- Data File Macros

The CMACRO.MAC file can be loaded with the G command or with the MACRO option on the CREDIT command line. In either case, the macros are available after they are loaded. The G command to be entered in the command line mode to load the CMACRO file is:

```
G CMACRO.MAC
```

The CREDIT command line to use to load the macro definitions when invoking the editor is:

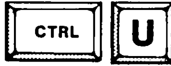
```
CREDIT <pathname1> [TO <pathname2> MACRO(CMACRO.MAC)]
```

Cursor Movement Macros

The cursor movement macros provide for fast positioning of the cursor in the text file. They perform the same function as holding down the cursor control keys. All the cursor control macros can be invoked in the screen mode of editing by entering the control character that is the name of the macro.

CTRL-B returns the cursor to the first character of the line where the cursor is located. This first character of a line is the first character that follows the most recent line terminator.

CTRL-L moves the cursor to the last character of the line where the cursor is located. The last character of a line is the line terminator character. If the current line does not have a line terminator, a NOT FOUND message is displayed. If the cursor is currently at the line terminator, it is moved to the next line terminator.



CTRL-U moves the cursor to the line terminator of the previous line.

CTRL-W searches for the next space character in the file. Usually, the effect is to move the cursor to the next word in the file. However, there are exceptions. For example, there is usually no space between the last word on a line and the first word on the next line. Therefore, CTRL-W at the last word on a line skips the first word of the next line.

CTRL-H searches for the next period character (.) in the file. Usually, the effect is to move the cursor to the next sentence in the file. However, there are exceptions. For example, sentences that end in a ? or ! are skipped. Also, periods do not indicate the end of a sentence, e.g., decimal points.

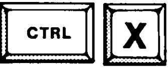
Text Control Macros

The text control macro (c) is invoked in screen editing mode by entering the macro execution command, CTRL-F, followed by the name of the macro, c.

The c macro centers any line that is less than 80 characters long on a single 80 character screen line. The macro should be used at the end of the file after entering the line to be centered. Position the cursor at the space following the last character of the line to be centered. Do not type a RETURN at the end of the line. Invoke the macro by entering CTRL-F followed by c, the macro name. If used within the text file instead of at the end of the file, position the cursor at the space character between the last character on the line and the line terminator. An extra line terminator is put into the file and must be deleted.

Block Transfer Macros

The block transfer macros allow block transfers of text within screen mode of editing. There are four macros that are used to transfer the text. All the macros are invoked by entering the control character that is the name of the macro.



CTRL-X sets tag 1 at the cursor location. If tag 1 was previously set, it is reset by this macro. Tag 1 marks the beginning of the text to be transferred. The character at tag 1 is included in the transfer.

CTRL-Y sets tag 2 at the cursor location. If tag 2 was previously set, it is reset by this macro. Tag 2 marks the end of the text to be transferred. The character at tag 2 is not included in the transfer.

CTRL-Q moves the block of text defined by the CTRL-X and CTRL-Y macros. The text is deleted from its current location and inserted at the location immediately preceding the cursor. The cursor is then moved to the first character of the inserted block of text.

CTRL-R copies the block of text defined by the CTRL-X and CTRL-Y macros. The text is inserted immediately preceding the cursor. The cursor is then moved to the first character of the inserted block of text. The source block of text is not affected by the copy.

File Formatting Macros

The file formatting macros are used to reformat files for printing. These macros are invoked from command line mode by entering the MF command followed by the name of the macro.

f The f macro formats a file for pagination and printing. It cannot be used on a file that is already formatted for pagination and printing (i.e., a file that f has already been run on). The file being formatted can contain no tilde characters. Do not interrupt this macro after it has started processing the text. This macro allows files to be edited as a single block of text and formatted and paged separately for printing when editing is complete. The macro moves the cursor to the beginning of the file and inserts an initial form feed character. The macro than advances 58 lines and sets up a page ending decision by displaying lines 49 through 58, followed by the page ending prompt ("_____TOP OF FORM _____"), followed by lines 58 through 69. The macro then queries

Do you want to start a new page here?

If the response is no (N), the top of form indicator is moved up one line and the query

How about here?

is made. Each N response moves the top of form indicator up one line. At line 48 or at the first yes (Y) response, the macro inserts a form feed character in place of the top of form indicator. The macro then advances another 58 lines and repeats the process. This process continues until there are less than 58 characters in the file. The macro then returns to the start of the file and inserts two lines after every form feed character. The first of the two inserted lines contains the word PAGE and a tilde as a position holding character for the page number. See the p macro to assign page numbers. This line is the top line on every page. The second of the two inserted lines is blank separating the page number from the text.

r The r macro removes the formatting characters from a file. Do not interrupt the r macro after it has started processing a file. The macro jumps to the start of the file and searches for the form feed characters. When a form feed is found, it deletes three lines to remove the pagination. Then, the file can be edited without disturbing the formatting. It can be reformatted prior to printing again.

p(<n>) The p(<n>) macro replaces each tilde character inserted by the format character with the page number specified by <n>. The macro must be invoked once for each page to be numbered. The macro jumps to the start of a file and searches for the position holding character. It substitutes the page number specified for the position holding character. When no more position holding characters remain, the macro displays the message NOT FOUND on the command line.

Data File Macros

The data file macros include one macro to read a data file and one macro to write a data file. These macros are invoked in command line mode with the MF command followed by the name of the macro. These macros use the OR, OW, R, and W commands described in the *ISIS CREDIT™ CRT-Based Text Editor User's Guide.*

i(<pathname>,<n>) The i macro inserts the specified number of lines <n> from the specified file <pathname> into the file being edited. All reads start from the first line of the data file. The number of lines to be read can exceed the number of lines in the data file, in which case the entire file is read. The data is inserted in the file being edited at the character preceding the cursor. The data file is closed after it is read. The cursor is moved to the first character of the data inserted into the file being edited unless the insert is made at the beginning of the file. Then, the cursor is moved to the end of the block of text inserted.

o(<pathname>,<n>) The o macro writes all or part of the file being edited to another disk file specified as <pathname>. The file being written need not exist. When the file does exist, its contents are lost. The <n> specifies the number of lines to write. It can be either positive or negative. If positive, the macro writes the number of lines specified from the cursor. If negative, the macro writes the number of lines specified preceding the cursor. The number of lines specified can exceed the number of lines in the file being edited in which case the entire file from the cursor is written. The text in the file being edited is not affected by the write.

NOTE

All further information needed to use the CREDIT text editor is covered in the *ISIS CREDIT™ CRT-Based Text Editor User's Guide,* included in the iPDS system's literature kit.

CHAPTER 7
DEBUG COMMANDS

Software Debugging and the Development Task

Writing programs is an essential part of the development task for microprocessor-based products. The software engineer requires tools to help verify program modules and isolate errors that can occur in software routines. The isolation and correction of errors in a program is called debugging.

As a minimum aid to debugging software, is stopping program execution at specified points, called breakpoints, and displaying the status of the machine. By comparing the actual machine status with the expected status at the breakpoint, errors in the software can be isolated.

There are a number of tools available that satisfy this minimum requirement. At one end of the spectrum, emulators provide breakpoint and display commands as well as a wide range of other features to control and monitor the hardware and software of a microprocessor-based product. As a software debugging aid, emulators are typically powerful enough for debugging complex programs in high level languages.

The emulator itself consists of both hardware and software separate from the hardware and software of the user system being debugged and, resulting in no overhead on the user system. Symbolic debugging (where memory locations are referenced as symbols defined by the programmer instead of as addresses), trace data collection (where sequences of machine states are stored for later examination), and advanced control structures (to control the operation of the target machine) are only some of the features found in emulators.

More information on emulation and emulators can be found in Chapter 2 of this manual and in the manuals on specific emulators.

While not offering all the features of an emulator, software debugging tools such as the DEBUG commands are adequate for isolating errors in assembly language programs running on existing hardware. The DEBUG commands provide breakpoints, display commands, and other features to aid in debugging software written for the MCS-85, the iPDS processor.

Debug Features

The DEBUG utility provides the following features:

- Loads an MCS-85 program into the iPDS memory from a file
- Executes the program with breakpoints
- Steps through the program executing a specified number of instructions at a time
- Displays and modifies the iPDS memory
- Displays and modifies the iPDS I/O ports
- Displays and modifies the iPDS processor registers

- Disassembles instructions from memory
- Configures custom I/O drivers for the system

With these features, it is possible to monitor the internal state of the processor during program execution. The programmer can verify that the actual processor state and memory contents match the expected state and contents at specific points during the execution of the program. The programmer can also save time testing possible corrections by modifying program memory from within DEBUG.

Debug Command

This section describes the initial loading and operation of the DEBUG command, a utility program that runs under the control of the ISIS-PDS operating system.

Since the DEBUG command runs on the same machine as the software to be debugged, it cannot occupy the same memory space as the software to be debugged. The DEBUG command uses locations from EE50H to F6C0H (the top of user memory). These locations are not available to the user program being debugged. However, the DEBUG command does modify the value of the top of user memory returned by the MEMCK system call, so any user routines that use high memory locations as an offset of the value returned by MEMCK still work properly. The MEMCK system call returns a value of ECC0H when DEBUG is present in memory.

Command Format

DEBUG [<command line>]

where

<command line> is an optional parameter. If specified, DEBUG loads an executable program to be debugged; otherwise, no program is loaded. The <command line> is the valid ISIS-PDS command required to run the executable program. A <command line> is of the form:

<pathname> [<parameters>]

where <pathname> is a valid ISIS pathname for the file containing the program to be debugged, and <parameters> are any parameters required on the command line by the program.

Comments

As soon as the DEBUG command is entered, it signs on with the message:

iPDS DEBUGGER Vm.n

where m.n is replaced by the actual version number.

If an executable 8080/8085 program is specified, the DEBUG command loads that program, displays the contents of the iPDS CPU program counter preceded by the character:

```
=>
```

and then prompts for a debugging command:

```
.
```

Any debugging command can be entered after the period prompt character appears. An input line can be terminated without execution by typing the ESC key. For example, the G command begins execution of the program at the specified starting address with up to two breakpoints. All the commands are described in this chapter.

Any time the program halts at a breakpoint, the next entry point is displayed preceded by the character:

```
=>
```

The DEBUG prompt (.) appears on the following line so that any debugging command can be entered. Other debugging commands display and modify the contents of iPDS memory or the iPDS CPU registers.

To return to the operating system, do one of the following:

- Enter the debugging command:

  ```
  E
  ```

- Execute an EXIT system call to return to the ISIS-PDS operating system (See Chapter 8 for an explanation of system calls)

- Press RESET key to reinitialize the system

Examples

In the following example, the DEBUG command loads a program named LIST with a starting address of 3680H. The file FILE.TXT is a parameter required by the LIST program. The A0> is the ISIS prompt. The G command executes the LIST program.

```
A0>DEBUG LIST FILE.TXT
iPDS DEBUGGER V1.0
=>3680
.G
```

The following commands execute the same program setting a breakpoint at address 36A0H and returning to the operating system with the E command as soon as the breakpoint is reached.

```
A0>DEBUG LIST FILE.TXT
iPDS DEBUGGER V1.0
=>3680
.G,-36A0
=>36A0
.E
A0>
```

In the following example, the DEBUG command is invoked without loading a program to debug.

```
A0>DEBUG
iPDS DEBUGGER V1.0
```

Overview of the Debugging Commands

The debugging commands perform the following functions:

- Configure the I/O interface to all standard peripheral devices except disks
- Aid in the software development of 8080/8085-based programs

I/O Interface

A physical device is an actual peripheral device connected to the system, e.g., a line printer, a terminal, or a modem. The term logical device refers to a symbolic device name assigned to a physical device. This name is recognized by the operating system to provide flexibility for input and output of data.

The DEBUG command recognizes four logical devices: a console device, a reader device, a punch device, and a list device. However, the I/O routines for these devices are different from the I/O routines used by the ISIS-PDS operating system for the console, reader, punch, and list devices.

By assigning a specific physical device to one of the logical devices, the corresponding data stream is routed to the specified peripheral. Debugging commands are provided to configure a system by assigning physical devices to logical devices.

The DEBUG program does not handle disk I/O.

Software Development

Debugging commands are provided to help debug MCS-80/85-based programs. These commands allow the user to:

- Display and modify the iPDS memory and iPDS CPU registers
- Disassemble MCS-80/85 instructions in iPDS memory
- Initiate execution of an MCS-80/85 program on the iPDS CPU
- Insert breakpoints in an MCS-80/85 program before execution
- Step through a program, stopping after a specified number of instructions
- Access user written I/O routines
- Directly input and output data to iPDS I/O ports

Entering Debugging Commands

Debugging commands can be entered at the keyboard anytime that the DEBUG prompt (a period) is displayed at the left side of the screen followed by the cursor. All commands are single alphabetic characters.

Some of the commands require parameters; for others the parameters are optional. Parameters can be either alphabetic or numeric as specified for each command. All numeric parameters are entered as 1 to 4 hexadecimal digits (0-9 and A-F). Do not append the letter H to the numeric value.

Normally, commands are executed after the RETURN key is pressed. Any exceptions to this rule are explained in the individual command descriptions. To terminate a command during execution or during command entry, use the ESC key.

Command Format for Debugging Commands

The general format of a debugging command is:

<command>[<parameters>]

where

<command> is the single alphabetic character for the command.

<parameters> are one or more values that vary from command to command. For example, two addresses are required as parameters for the D command.

Parameters can be alphabetic or numeric. Numeric parameters can be entered as 1 to 4 hexadecimal digits. If more than 4 digits are entered, only the last 4 digits entered are used by the command. For example, the value 123456 is treated as 3456 in hexadecimal by the command. A comma or a space can be entered in place of a comma shown in the format for a command. If a comma is not shown in the command format, do not enter a space or a comma in the command line. For example, in most cases, a space is not allowed after the command letter.

Entry Errors

The debugging commands check for the following error conditions:

- Invalid characters
- Address value errors
- Parameter errors

Invalid Characters

The DEBUG utility checks the validity of each character entered at the keyboard. As soon as an invalid character is encountered, a number sign (#) is displayed and the command is terminated. The DEBUG prompt is displayed on the following line and another command can be entered. In the following example, 4 is rejected because it is not a valid command:

.4#

The first character entered must be a valid command; otherwise, it is rejected. All addresses must be entered in hexadecimal. Any character other than 0-9 and A-F is rejected. In the next example, G is not a valid digit.

```
.D1000,1FFG#
```

In the following example, the space after the command character X is rejected because the space is not allowed.

```
.X #
```

Address Value Errors

Many commands require two addresses where the first address is lower than the second. If the first address given is higher than the second, the operation is performed on the single address specified first.

For example, if the following command is entered to fill memory from address 900H to 1000H, the DEBUG utility would place an 0FFH in address 1000H and do nothing else.

```
.F1000,900,FF
```

No error message is given to indicate that only a single byte was filled instead of 100H bytes.

The valid range of address is 0000H through 0FFFFH. If addresses higher than 0FFFFH are entered, only the last four digits are used by the command. For example, if 10000H is entered instead of 1000H, the address is evaluated as 0000H. In the following example, the actual command that is executed is F0000,9000,FF filling 9000H bytes of memory with the constant 0FFH.

```
.F10000,9000,FF
```

This command erases the memory used by the operating system and by the DEBUG command. The system must then be reinitialized to run.

Parameter Errors

If the correct number of parameters is not entered, the debugging command replaces the DEBUG prompt with a number sign (#) and displays the DEBUG prompt on the following line to accept a new command.

For example, the D command requires two parameters. If only one is given, DEBUG replaces the prompt on the D command line with a number sign (#) and returns the DEBUG prompt on the following line.

```
.D011
```

becomes

```
#D011
.
```

Categories of Debugging Commands

The debugging commands are grouped into the following categories:

- Program execution commands
- I/O configuration commands
- I/O control commands
- Memory control commands
- Register commands
- Utility commands

Program Execution Commands

The program execution commands are used to run the program to be debugged. They provide breakpoints to halt the program at a specified address. Then, other DEBUG commands can be used to monitor the machine status. For example, CPU registers, memory locations, and I/O ports can be checked at a breakpoint to verify that they contain the expected data. The commands are:

G (Execute) Transfers control to the loaded program and optionally sets one or two breakpoints.

N (Step) Executes a specified number of instructions starting at the address currently in the Program Counter. The disassembled instructions are displayed as they are executed.

FUNCT-R Manually stops program execution. This function can be used to interrupt a program that is in an infinite loop.

I/O Configuration Commands

The DEBUG utility recognizes four logical devices:

- Console (CO, CI)
- Reader (RI)
- Punch (PO)
- List (LO)

The terms Reader and Punch were chosen to maintain compatibility with earlier systems that supported paper tape readers and paper tape punches.

The I/O configuration commands select the physical device that receives the logical device data.

The DEBUG commands that control the system I/O configuration are:

A (Assign) Assigns a physical device to a logical device

Q (Query) Displays the devices currently assigned

The characteristics of the physical device must match the characteristics of the logical device to which it is assigned. Therefore, only a subset of the available peripherals can be validly assigned to a given logical device. The characteristics of the four logical devices are as follows:

- The Console (CO, CI) is an interactive, character-oriented input and output device.
- The Reader (RI) is a serial input device.
- The Punch (PO) is a serial output device.
- The List device (LO) is a character-oriented output device that accepts a character from the calling program and outputs it to an external medium in alphanumeric characters that can be read by the user.

An I/O driver routine is required for each physical device before it can be assigned to a logical device. The DEBUG software provides I/O driver routines for the following physical devices:

- Internal iPDS video terminal, CRT and keyboard (can be assigned to any logical device)
- Serial device attached to the serial I/O connector and configured with the SERIAL command (see Chapter 5) (can be assigned to any logical device)
- Line printer attached to the parallel I/O connector (can be assigned to the List logical device)
- Batch device where the currently assigned Reader logical device is also assigned as the Console input and the currently assigned List logical device is also assigned as the Console output (can be assigned to CO and CI)

The batch device permits a non-interactive mode where commands are input from the Reader and executed by the operating system. A file containing ISIS commands must be prepared on the Reader device. In preparing the command file, enter commands in exactly the same way as if the system were in interactive mode. Each command should end with a carriage return/linefeed pair of characters. The prompt character should not appear as part of the command file. The last command in the file should re-assign the Console to prevent DEBUG commands from reading off the end of the file.

The user must provide I/O driver routines for the following physical devices:

- User Defined Device 1 where the user provides the I/O driver routines for the device (can be assigned to the Console, Reader, Punch, or List logical device as long as the characteristics of the physical device match the characteristics of the logical device)
- User Defined Device 2 where the user provides the I/O driver routines for the device (can be assigned to the Reader or Punch logical device as long as the characteristics of the physical device match the characteristics of the logical device)

User defined I/O drivers are accessed by configuring them with the A command. See the A command for more information.

I/O Control Commands

The I/O control commands allow data to be read from and written to the iPDS I/O ports one byte at a time. These commands can be used to verify that the I/O ports contain the correct data at a breakpoint during the execution of a program.

If the port does not contain the expected data or if the part of the program that writes to I/O ports is not available yet, the correct data can be written to the port with these commands, so that debugging can continue on the parts of the program that are available. There are two commands for accessing the iPDS I/O ports:

I (Input)	Reads and displays a single byte from the specified input port
O (Output)	Writes the specified byte to the specified output port

Memory Control Commands

There are six commands for accessing the iPDS memory. The commands that read memory can be used for RAM as well as PROM and ROM. The commands that write to memory can only be used with RAM but no error is given if writing to PROM or ROM occurs. All of these commands with the exception of the S command can be terminated while running by pressing the ESC key.

The memory control commands can be used to verify that a memory location contains the correct data at a breakpoint during the execution of a program.

If the memory location does not contain the expected data or if the part of the program that writes to memory is not available yet, the correct data can be written to the memory location with these commands, so that debugging can continue on the parts of the program that are available.

These commands can also be used to disassemble program memory and modify the program in memory (patch the program). The ability to patch a program in memory allows minor programming errors to be corrected without having to exit from DEBUG, modify the source program, and retranslate it. The memory control commands are:

D (Display)	Displays a specified range of memory
F (Fill)	Fills a specified range of memory with a specified constant value
M (Move)	Copies the contents of a specified range of memory into another area of memory
S (Substitute)	Modifies memory on a byte-by-byte basis
C (Disassemble Code)	Displays specified memory range as MCS-85 instructions
T (Disassemble Code)	Displays the specified number of MCS-85 instructions starting at the MCS-85 Program Counter (PC)

Register Commands

The register commands can be used to verify that the CPU registers contain the correct data at a breakpoint during the execution of a program. If a register does

not contain the expected data or if the part of the program that writes to the register is not available yet, the correct data can be written to the register with these commands, so that debugging can continue on the parts of the program that are available.

The register commands can also be used to set the program counter, so that the program being debugged can be run from a controlled starting location. The register commands are:

X (Display Form) Displays the contents of all registers

X (Modify Form) Changes the contents of a single register

Utility Commands

The utility commands are:

E (Exit) Returns to the ISIS-PDS operating system

H (Hexadecimal) Adds and subtracts two hexadecimal numbers

Sample Debugging Session

The following sample debugging session illustrates most of the debugging commands.

However, before this sample can be run, an object program must be created to debug. The program loaded and run under DEBUG control is the sample program (PROGA.SRC) entered with the CREDIT text editor in the *ISIS CREDIT™ CRT-Based Text Editor User's Guide.*

The first few examples show how to create an object file that can be loaded and run under the control of DEBUG.

The source code for the program is given below for convenience. It should be entered under the CREDIT editor with the command line:

```
CREDIT PROGA.SRC
```

The source listing for the file that should be entered into PROGA.SRC follows:

```
        EXTRN  ISIS
        EXTRN  CO
        EXTRN  CI

        ORG    4000H
EXIT    EQU    9

EBLK:   DW     ESTAT
ESTAT:  DS     2

START:  MVI    B,1AH

LOOP:   CALL   CI

        MOV    C,A
        CALL   CO
```

(continued)

```
CMP     B
JNZ     LOOP
MVI     C,EXIT
LXI     D,EBLK
CALL    ISIS

END     START
```

```
A0>ASM80 PROGA.SRC

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1

ASSEMBLY COMPLETE,   NO ERRORS
A0>LINK PROGA.OBJ,SYSPDS.LIB TO PROGA.LNK
ISIS-II OBJECT LINKER V3.0
A0>LOCATE PROGA.LNK
ISIS-II OBJECT LOCATER V3.0
A0>DEBUG PROGA

PDS DEBUGGER V1.0
=>4004
```

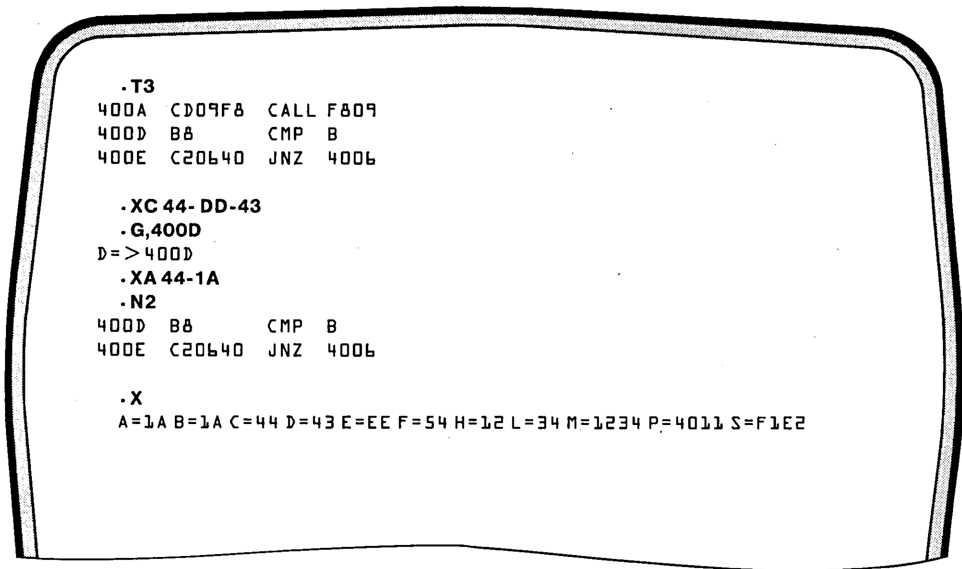
Key-in Sequence	Comments
ASM80 PROGA.SRC	Assemble the program.
LINK PROGA.OBJ,SYSPDS.LIB TO PROGA.LNK	Since the program contains external references to system calls, link it with the system library as shown in this command line.
LOCATE PROGA.LNK	Locate the program to assign memory locations where needed before running.
DEBUG PROGA	Use the DEBUG command line to load the program to be debugged. The program start address is displayed below the sign-on message.

```
. .C4004,9
4004  061A     MVI   B,1A
4006  CD03F8   CALL  F803
4009  4F       MOV   C,A
400A  CD09F8   CALL  F809
400D  B8       CMP   B
400E  C20640   JNZ   4006
4011  0E09     MVI   C,09
4013  110040   LXI   D,4000
4016  CD4000   CALL  40

 .X
 A=AA B=BB C=CC D=DD E=EE F=FF H=12 L=34 M=1234 P=4004 S=F1E2
 .G,-4009
=>4009
 .X
 A=4D B=1A C=CC D=DD E=EE F=10 H=12 L=34 M=1234 P=4009 S=F1E2
 .N1
4009  4F       MOV   C,A
```

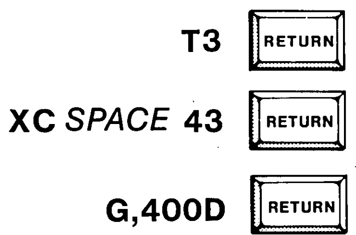
Key-in Sequence	Comments
C4004,9	The C command disassembles the number of instructions specified. The second, forth, and ninth instructions are system calls.
X	Display the registers. Notice that the program counter, P, is set to 4004H, the beginning address of the program.
G,4009	Use the G command to start the program from the address specified or from the program counter if no address is specified. A breakpoint is set at location 4009H, the address of the instruction following the system call.
	The first system call inputs a character from the keyboard. Here, the character entered is M. As soon as the character is entered, the program returns from the system call to execute the next instruction at address 4009H. However, a breakpoint is set here, so the program halts.
X	While the program is halted, any DEBUG command can be entered. Enter the X command to display registers. The program counter is pointed at 4009H. The ASCII code for M, 4DH, is in register A.
N1	Execute the number of instructions specified. The N command also displays the disassembly information for the instructions. This instruction moves the value in the A register to the C register.

```
.X
A=4D B=1A C=4D D=DD E=EE F=10 H=12 L=34 M=1234 P=400A S=F1E2
.G,–400D
M=>400D
.X
A=4D B=1A C=4D D=DD E=EE F=B0 H=12 L=34 M=1234 P=400D S=F1E2
.N1
400D  B8       CMP  B

.X
A=4D B=1A C=4D D=DD E=EE F=14 H=12 L=34 M=1234 P=400E S=F1E2
.N1
400E  C20640   JNZ  4006

.X
A=4D B=1A C=4D D=DD E=EE F=00 H=12 L=34 M=1234 P=4006 S=F1E2
.G,-4009
=>4009
.N1
4009  4F       MOV  C,A

.
```

Key-in Sequence	Comments
X	Display the registers again. This time both the A and C registers contain 4DH, the ASCII code for M.
G,400D	Issue the G command is again with a breakpoint set at 400DH. This system call displays the character just typed at the keyboard. Notice the M displayed on the screen. Use the G command with a breakpoint set at the instruction following the system call to DEBUG within a system call.
X N1 RETURN	Display the registers. This instruction compares the value keyed with the value in the B register. If the two are the same, the program ends. Otherwise, another character can be entered and displayed.
X	The flag register (F) has changed. The sixth bit in the F register should have been set to zero from the compare instruction.
N1	Execute a single instruction. The program loops if the zero flag is zero.
X	The program counter is set to 4006H to loop through the program again.
G,4009 RETURN D	Give the G command with a breakpoint at address 4009H to halt the program on returning from the system call. Enter the character D while the program waits for keyboard input.
N1	Single step to execute the instruction to move the value in the A register to the C register.



Key-in Sequence **Comments**



T3 RETURN

The T command disassembles the number of instructions specified starting with the instruction at the program counter.

XC *SPACE* 43 RETURN

The X command can also be used to change the registers. First, type XC and press the space bar. The current value of the C register is displayed. Type 43 to change this to the ASCII code for the character C. Now, when the program is run the character C is displayed instead of D.

G,400D RETURN

Issue the G command with a breakpoint at address 400DH. The character, C, is displayed by the system call.

XA RETURN

1A RETURN

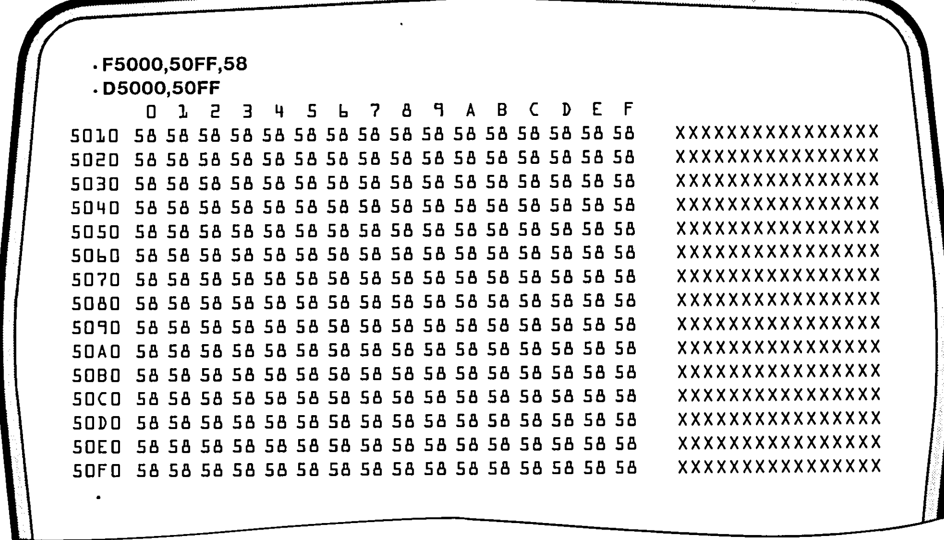
Change the value in the A register to 1AH. This is the value used to end the program loop.

N2 RETURN

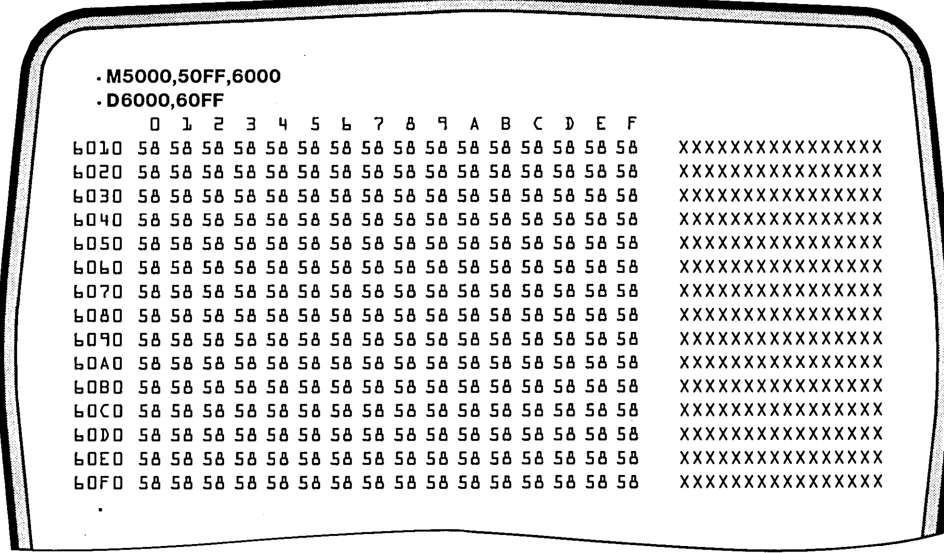
Step through two instructions with the N command. The zero flag should be set to one after the compare and the Jump should not be made.

X RETURN

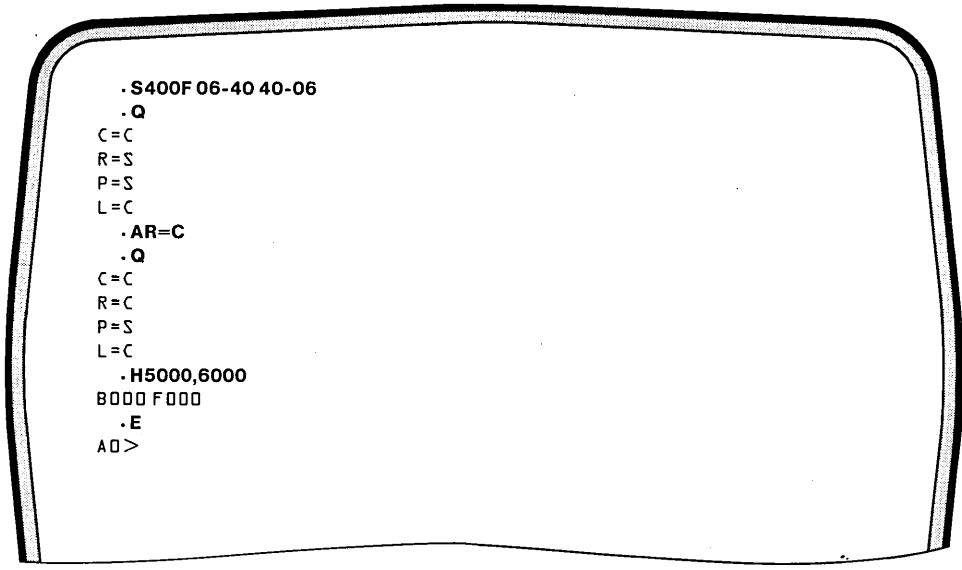
Display the registers to verify that the jump is not taken. Notice the flag register has the value 54H. The sixth bit, the zero flag is set to one. The program counter has the value 4011H, so the program will not run through the loop again.



Key-in Sequence	Comments
F5000,50FF,58	Sometimes it is useful to fill memory with a constant when debugging. This screen shows an example of the F command which fills the block of memory specified with the constant specified.
D5000,50FF	Display a block of memory with the D command. Here, the block filled with the constant 58H, the ASCII code for X, is displayed.



Key-in Sequence	Comments
M5000,50FF,6000	A specified block of memory can also be moved to a specified destination with the M command.
D6000,60FF	Display the destination block to verify the move.



Key-in Sequence | **Comments**

S400F *SPACE* **40** *SPACE* **06** RETURN

Memory can be interactively changed with the S command. Type S followed by the address of the first byte of memory to be changed. The content of this location is displayed followed by a dash. To change this byte, type a hexadecimal value. To leave it unchanged press the space bar. Press the space bar at any time to continue displaying the next memory location. Press the RETURN key to end the interactive memory editing and return to DEBUG.

The Q command displays the current assignment of physical devices to the DEBUG logical devices (different from the ISIS logical devices).

Q RETURN

The A command changes the default assignment of devices. A physical device must be properly connected before the assignment of the device has any meaning.

It is often useful to compute hexadecimal values when debugging assembler language programs. The H command computes the sum and difference of the two hexadecimal values specified. The first result displayed is the sum and second is the difference.

Exit from DEBUG and return to the ISIS operating system.

Debugging Commands in Alphabetical Order

The rest of this chapter describes the individual debugging commands in a reference format using the notational conventions explained in Chapter 5. The commands appear in alphabetical order.

A

Assign logical device to physical device

Command Format

A<logical device>=<physical device>

where

<logical device> specifies the logical device to which a physical device is assigned.

<physical device> specifies which physical device is to be assigned.

Comments

The possible values for <logical device> are shown in Table 7-1.

Table 7-1 Possible values for <logical device>.

Single Letter Symbol for <logical device>	Device
C	Console
R	Reader
P	Punch
L	List

Table 7-2 gives the possible values for each <physical device> and the valid matches with logical devices.

Table 7-2 Possible values for <physical device>

Logical Device	Single Letter Symbol for <physical device>	Device
CONSOLE	S	Serial I/O Device
	C	CRT Terminal
	B	Batch Mode Device
	1	User Defined Device 1
READER	S	Serial I/O Device
	C	CRT Terminal
	1	User Defined Device 1
	2	User Defined Device 2
PUNCH	S	Serial I/O Device
	C	CRT Terminal
	1	User Defined Device 1
	2	User Defined Device 2
LIST	S	Serial I/O Device
	C	CRT Terminal
	L	Line Printer
	1	User Defined Device 1

The default assignments are:

```
C=C
R=S
P=S
L=C
```

The Batch Device is a non-interactive mode of operation where the currently assigned Reader Device is used as the Console input device and the currently assigned Punch Device is used as the Console output device.

The User Defined Devices 1 and 2 require a user-written I/O driver program. This program must be added to the ISIS operating system with the IOSET system call. See Chapter 8 for instructions on adding a user written I/O driver to the operating system.

Examples

To assign a serial I/O device as the Console:

```
.AC=S
```

To assign a user defined device as the Reader:

```
.AR=1
```

To assign a serial I/O device as the List Device:

```
.AL=S
```

Given the previous assignments, the following command assigns the User Defined Device 1 (the current Reader Device) as the Console input and the Serial Device (the current List Device) as the Console output:

```
.AC=B
```

To reassign the Console I/O to the CRT Terminal, the following command must be entered from the current console input, User Defined Device 1:

```
.AC=C
```

C

Disassemble code at specified memory locations

Command Format

C<start address>,<n>

where

<start address>	is the beginning of the memory range to be disassembled. The <start address> must be given in hexadecimal. Do not append the letter H to the start address value.
<n>	is the hexadecimal number of instructions to be disassembled.

Comments

Both the <start address> and the <n> are required. They must be separated by a space or a comma.

The instructions at the specified addresses are displayed on the current List device as MCS-80/85 mnemonics.

The display is listed in the format shown in the following example.

Example

To disassemble four instructions starting at memory location 4004H, enter the following command:

```
.C4004,4
4004 061A   MVI    B,1A
4006 CD03F8 CALL   F803
4009 4F     MOV    C,A
400A CD09F8 CALL   F809
```

D

Display specified memory range

Command Format

D<start address>,<end address>

where

<start address> is the beginning of the memory range to be displayed. The <start address> must be given in hexadecimal and must be less than or equal to the <end address>. Do not append the letter H to the start address value. If <start address> is greater than or equal to <end address>, the single byte located at the <start address> is displayed.

<end address> is the end of the memory range to be displayed. The address must be given in hexadecimal. Do not append the letter H to the hexadecimal value.

Comments

Both the <start address> and the <end address> are required. They must be separated by a space or a comma.

The contents of the specified addresses are displayed on the current List device in hexadecimal and ASCII.

The memory display is listed in the format shown in the following example.

- The address at the left of each line is the address of the first byte on that line.
- Sixteen bytes are displayed on each line in hexadecimal, followed by the sixteen ASCII characters represented by each byte.
- An underline appears at any position in the display that is not in the specified range or for any non-printable ASCII byte.
- If the start address is not on a sixteen byte boundary, the first line contains underlines from the previous sixteen byte boundary to the first character in the range specified for display.
- If the end address is not on a sixteen byte boundary, the last line contains underlines from the last character in the range specified to the next sixteen byte boundary.
- If the entire range is less than sixteen bytes, the entire range appears on a single line with underlines in positions not included in the range specified.

Examples

To display the contents of memory locations C09H through C2AH:

```
      0 1   2 3   4 5   6 7   8 9   A B   C D   E F
0C09  __ __ __ __ __ __ __ __ __ 40 30 CD A4 2D 22 47 - - - - - - - - - - @ 0 _ _ - " G
0C10  30 3E  09 CD  80 2D  D2 1E  0C 21  4A 30  36 37  3A 4A 0 > _ _ _ - _ _ _ ! J 0 6 7 : J
0C20  20 11  47 30  CD E3  2C EB  3E 04  __ :_  ____  ____ 0 - G O - - ' - > -  # _ _ _ _ _
```

The value 40H that appears under column 9 is the value at address 0C09. Double underlines precede this value because the previous addresses are not part of the range specified. The 16 corresponding ASCII characters follow column 0FH. Underlines appear for out of range values and for non-ASCII codes.

To display the contents of the single location 0100H:

```
.D0100,0100
          0 1   2 3   4 5   6 7   8 9   A B   C D   E F
0100  01 __ ____ ____ ____ ____ ____ ____ ____   - - - - - - - - - - - -
```

Note that the command:

```
.D0100
```

produces an error because the second required parameter is not specified. The prompt (.) is replaced with the number sign (#), and the DEBUG prompt is displayed on the following line so a command can be entered. Both the start and end address must be specified.

```
#D0100
.
```

E

Exit to the ISIS-PDS operating system

Command Format

```
E
```

Comments

The E command returns controls to the ISIS-PDS operating system as indicated by the operating system prompt.

Example

To return to the ISIS operating system enter:

```
E
```

F

Fill memory with constant

Command Format

F<start address>,<end address>,<constant>

where

<start address>	is the beginning of the memory range to be filled with the <constant>. The <start address> should be given in hexadecimal and should be less than or equal to the <end address>. Do not use the letter H for hexadecimal values. If <start address> is greater than or equal to <end address>, the single byte located at the <start address> is filled with the <constant>.
<end address>	is the end of the memory range to be filled with the <constant>. The address must be given in hexadecimal.
<constant>	is the byte to be written to the specified address range. The <constant> must be given in hexadecimal.

Comments

All three parameters are required. The execution of this command may be terminated by typing the ESC key. This command should not be used to write over locations below 3BFDH where ISIS routines are located.

Example

To initialize memory locations 4000H through 402FH with 00H, enter the command:

```
.F4000,402F,00
```

FUNCT-R

Return to DEBUG

Command Format

FUNCT-R

Comments

This function key performs a manual interrupt during the execution of the G command. This command can be used in case the program being debugged begins executing an infinite loop.

Example

If the G command has been given to run a program, type FUNCT-R

=>xxxx

.

halts the program and returns the DEBUG prompt. The actual address where the program halts appears in place of xxxx.

G

Execute program with breakpoints

Command Format

G[<start address>][,<breakpoint 1>[,<breakpoint 2>]]

where

<start address> is an optional parameter that specifies the address to be placed in the program counter. The program loaded begins executing at this address. The address must be entered as a hexadecimal value. If <start address> is not specified, the address currently in the program counter is used.

<breakpoint 1> specifies an instruction address where the program stops executing and return to the DEBUG prompt. The address must be entered as a hexadecimal value. If <breakpoint 1> is not specified, the program does not halt.

<breakpoint 2> specifies a second instruction address where the program stops executing and return to the DEBUG prompt. The address must be entered as a hexadecimal value. If neither breakpoint is specified, the program runs without halting.

Comments

A breakpoint is the address of the first byte of an instruction within the program. When a breakpoint is specified, the first byte of the instruction at the breakpoint is replaced by the 1-byte software interrupt instruction, RST 1. Then, when the breakpoint is reached during program execution:

- The program stops executing.
- The single byte RST 1 instruction is replaced by the original instruction at the breakpoint address.
- DEBUG displays the address of the breakpoint (now containing the original instruction) as the next program entry point for subsequent execution.
- The Program Counter contains the breakpoint address (now containing the original instruction).
- The DEBUG prompt is displayed.

Then, debugging commands can be used to check the contents of registers or memory in the program.

When the breakpoint is reached, the instruction at the breakpoint is not executed before returning to the DEBUG prompt. The address of the breakpoint instruction becomes the next entry point. This instruction is then executed when the G command is given again unless the Program Counter is altered before the G command is given.

To specify breakpoints with the G command:

1. Enter G optionally followed by the start address
2. Type a comma or a space
3. The G command displays a dash.
4. Enter the first breakpoint address.
5. If a second breakpoint is not desired, press the RETURN key to execute the command.
6. To enter a second breakpoint, type a comma or space; the G command displays a dash; enter the second breakpoint address followed by the RETURN key.

If the command contains a syntax error, no breakpoints are set. The command must be re-entered and the breakpoints specified again.

Both breakpoints are eliminated the first time that the system halts. To resume execution with one or both of the same breakpoints, re-enter the command with the breakpoints.

Unpredictable results occur when breakpoints are set within an ISIS-PDS system call routine. When debugging programs containing ISIS system calls, set the breakpoints at the instruction before or after the system call.

Examples

To begin executing at the address currently in the program counter:

```
.G
```

To execute a program whose entry address is 4000H:

```
.G4000
```

To execute a program whose entry address is 4000H and to set a breakpoint at 40CFH:

```
.G4000,-40CF
=>40CF
```

The characters => indicate the next entry point of the program.

To execute a program whose entry point is 4000H and to set two breakpoints at 40CFH and 5000H:

```
.G4000,-40CF,-5000
=>5000
```

Here, the instruction at 40CFH was never executed, so when the instruction at address 5000H was fetched, the program was interrupted.

H

Hexadecimal
add and subtract

Command Format

H<number 1>,<number 2>

where

<number 1> is the first number to be added. The value supplied for <number 2> is subtracted from this value. The number must be entered as a hexadecimal value. Do not append the letter H to this value.

<number 2> is the second number to be added. This value is subtracted from <number 1>. The number must be entered as a hexadecimal value. Do not append the letter H to this value.

Comments

The numbers can contain a maximum of four hexadecimal digits. Negative numbers must be entered in twos complemented form.

The command displays two four-digit hexadecimal values as the result. The first is the sum of the two numbers and the second is the difference between the two numbers. Negative numbers are displayed in twos complement form.

If more than four digits are entered, the command uses the rightmost four digits. The leading digits are lost.

Example

To add E49H and 111H and to subtract 111H from E49H:

```
.HE49,111
OF5A OD38
```

I

Input byte
from iPDS port

Command Format

I<port address>

where

<port address> is the iPDS port address to be read. The port address must be given in hexadecimal. Do not append the letter H to the address. The single byte read at the port address specified is displayed on the current list device in hexadecimal.

Comments

The display returned shows the port address followed by the value read at that port. The MCS-85 port address assignments on the iPDS system are listed in Chapter 8 in the section "I/O Address Space."

Example

The command

```
.IOCO
CO=>OD
```

reads the value at the Keyboard/CRT data port (I/O port address 0C0H). The value at this address is the last value typed in at the keyboard, a carriage return. Thus, the ASCII code for carriage return, 0DH, is displayed.

M

Move block
of memory

Command Format

M<start address>,<end address>,<destination address>

where

<start address> is the address of the first byte to be moved. The <start address> must be given in hexadecimal and must be less than or equal to the <end address>. Do not append the letter H to the address. If <start address> is greater than or equal to <end address>, the single byte located at the <start address> is moved.

<end address>	is the address of the last byte of memory to be moved. The address must be given in hexadecimal. Do not append the letter H to the address.
<destination address>	is the address to which the first byte is moved. Each subsequent byte is moved to the location one higher than the previous byte.

Comments

The data is moved on a byte-by-byte basis. The first byte is moved, then the second byte, and so on. The data in the original location is not destroyed. Any data at the destination address is overlaid.

Because the command works on a byte-by-byte basis, the destination address should not be within the range of the source addresses. If it is within the range of the source addresses, the operation is attempted. By the time the command reaches the end of the block, the source data has been overlaid by the first data moved. No error indication is given.

The move command can be terminated while in progress with the ESC key.

Examples

To move data currently at address 4000H through 4100H to address 5000H through 5100H:

```
.M4000,4100,5000
```

To move the data currently at address 4000H through 4FFFH to address 4800H through 57FFH:

```
.M4800,4FFF,5000
.M4000,47FF,4800
```

If this move were done with a single command, the second 7FFH bytes would be a copy of the first 7FFH bytes because 4800H through 4FFFH would be overlaid by the first 7FFH bytes before they could be copied.

N

Execute a specified number of instructions

Command Format

N<step count>

where

<step count> specifies the number of instructions to execute starting at the program counter. The value is given in hexadecimal. Do not append the letter H to the value.

Comments

The N command executes the specified number of instructions and then stops. The disassembly information is displayed for each instruction executed. Unpredictable results occur if the N command is used to execute instructions within an ISIS system call routine. To debug a program containing ISIS system calls, use the N command up to the system call; then, use the G command with a breakpoint set at the instruction following the system call.

The ESC key can be used to return to DEBUG when stepping through the program with the N command.

Example

To execute four instructions starting at the address in the program counter of 400DH, enter the following command:

```
.N4
400D B8     CMP B
400E C20640 JNZ 4006
4011 0E09   MVI C,09
4013 110040 LXI D,4000
```

O

Output byte to I/O port

Command Format

O<port address>,<databyte>

where

<port address> is the iPDS port address to be written. The port address must be given in hexadecimal. Do not append the letter H to the address.

<databyte> specifies the single byte of data in hexadecimal that is to be written to the I/O port specified.

Comments

The MCS-85 port address assignments on the iPDS system are listed in Chapter 8 in the section "I/O Address Space." The command displays the byte output followed by the port to which it was written.

Example

To output the byte 37H (a command to initialize the 8251 Serial USART chip) to the 8251 Command/Status Port (91H), enter the following command:

```
.O91,37
37=>91
.
```

Q

Query current devices assigned

Command Format

Q

Comments

The Query command displays the status of the system I/O devices. It displays a list of the logical devices and the physical devices currently assigned to them. No parameters are allowed with this command.

Table 7-3 shows the four logical devices.

Table 7-3 Logical Devices

Single Letter Symbol for <logical device>	Device
C	Console
R	Reader
P	Punch
L	List

Table 7-4 gives the possible values for each physical device assigned.

Table 7-4 Possible Values for Physical Device

Logical Device	Single Letter Symbol for <physical device>	Device
CONSOLE	S	Serial I/O Device
	C	CRT Terminal
	B	Batch Mode Device
	1	User Defined Device 1
READER	S	Serial I/O Device
	C	CRT Terminal
	1	User Defined Device 1
	2	User Defined Device 2
PUNCH	S	Serial I/O Device
	C	CRT Terminal
	1	User Defined Device 1
	2	User Defined Device 2
LIST	S	Serial I/O Device
	C	CRT Terminal
	L	Line Printer
	1	User Defined Device 1

The default assignments are:

```
C=C
R=S
P=S
L=C
```

The Batch Device is a non-interactive mode of operation where the currently assigned Reader Device is used as the Console input device and the currently assigned Punch Device is used as the Console output device.

The User Defined Devices 1 and 2 require a user-written I/O driver program as do the High Speed Paper Tape Reader and Punch.

Example

To list the current assignments of physical devices to logical devices, enter:

```
.Q
```

The following assignments could be displayed indicating that a CRT terminal is assigned as the Console and a user defined device is assigned to the Reader and Punch while a line printer is assigned as the List Device.

```
C=C
R=1
P=2
L=L
```

S

Substitute memory interactively

Command Format

S<address>,[<databyte>][,[<databyte>] ...

where

<address> specifies an address. This value must be given in hexadecimal. Do not append the letter H to the value.

<databyte> specifies the single byte of data in hexadecimal that is to replace the byte currently at the location specified by <address>. This parameter is optional. No changes are made if <databyte> is left off.

Comments

The S command is operated as follows:

1. Enter the command and the address followed by a comma or space.
2. The current contents of the address are displayed followed by a dash.
3. Do one of the following:
   - Modify the contents by entering a new byte in hexadecimal.
   - Look at the next sequential byte of data by entering a comma or space.
   - End the command without modifying the data by pressing the RETURN key.
   - Repeat any combination of the first two choices ending with the RETURN key.

Examples

The following command illustrates the byte by byte replacement of data in memory. The dash character (-) is displayed by the system following the current content of each sequential location of memory.

```
.S4000,3A-,56-00,49-
```

The value 3AH at address 4000H is not changed; the value 56H at address 4001H is replaced with 00H; the value 49H at address 4002H is not replaced; the RETURN key is pressed to terminate the command and return to the DEBUG prompt. The characters typed by the user are:

```
S4000,,00,
```

T

Disassemble code
relative to Program Counter

Command Format

T<n>

where

<n> is the number of instructions relative to the Program Counter to be disassembled. It is specified in hexadecimal. Do not append the letter H to the value.

Comments

Memory is disassembled starting at the location of the Program Counter. The number of instructions specified are displayed on the current List device as MCS-80/85 mnemonics.

The display is listed in the format shown in the following example.

Example

To disassemble the contents of 4 instructions starting at the program counter (in this example, 4004H), enter the command:

```
.T4
4004 061A   MVI  B,1A
4006 CD03F8 CALL F803
4009 4F     MOV  C,A
400A CD09F8 CALL F809
```

X

Display/modify registers

Command Format

Display Form

X

Modify Form

X<register>,[<data>][,<data>]...

where

<register> is the single character register name.

<data> specifies one or two bytes of data to be placed in the register. The <data> must be entered in hexadecimal.

Comments

The display form of the command displays the contents of all the registers. The modify form displays and optionally changes the contents of the registers one at a time.

The modify form of the command functions the same as the Substitute command. It operates as follows:

1. Enter the command and a single letter symbol for a register.

2. The contents of the specified register is displayed, followed by a dash.

3. Do one of the following:

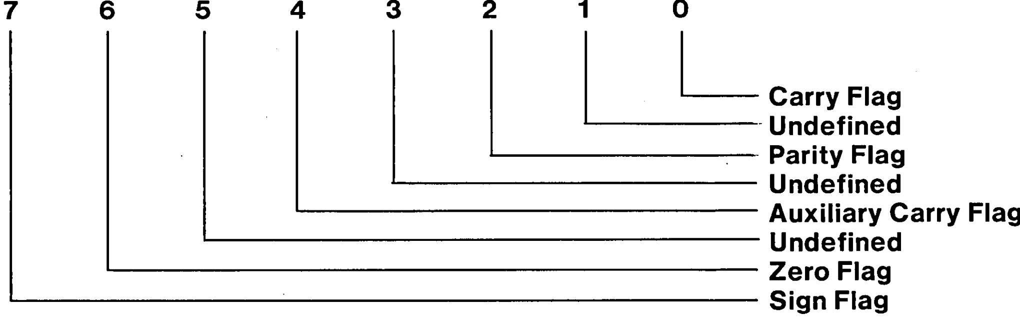
   - Modify the contents by entering new bytes in hexadecimal. If the register is a two byte register, enter two bytes (four digits). For single byte registers, enter a single byte (two digits).
   - Look at the contents of the next sequential register by entering a comma. See the following list for the sequence of registers.
   - End the command without modifying the data by pressing the RETURN key.
   - Repeat any combination of the first two choices ending with the RETURN key.

Table 7-5 lists the single character symbol for the registers that can be modified in the sequence in which they are displayed.

Table 7-5 Character Symbols for Register Modification

Symbol	Register	Size of Register
A	CPU A Register	1 Byte
B	CPU B Register	1 Byte
C	CPU C Register	1 Byte
D	CPU D Register	1 Byte
E	CPU E Register	1 Byte
F	CPU Flag Byte	1 Byte
H	CPU H Register	1 Byte
L	CPU L Register	1 Byte
M	CPU H and L Registers Combined	2 Bytes
P	CPU Program Counter	2 Bytes
S	CPU Stack Pointer	2 Bytes

The Flag Byte is displayed in the following 8-bit format:



Examples

The following example shows the use of the display form of the X command:

```
.X
A=22 B=0D C=0D D=D7 E=00 F=02 H=F2 L=D7 M=F2D7 P=F2D9 S=F1C6
```

The flag byte F has the value 02 which is 0000 0010 in binary. Thus, no flags are set.

In the next example the registers starting with the C register are modified.

```
.XC 0D-,D7-,00-40,02-,F2-,D7-,F2D7-4CCC,F2D9-420C,F1C6-,
.
```

The contents of each register are displayed in sequence starting with the specified register. To change the register contents, a hexadecimal value is entered. To go to the next sequential register, a comma or space is entered.

CHAPTER 8
SYSTEM PROGRAMMER'S REFERENCE

Operating System Considerations

An operating system is a group of programs, or software routines, that provide the software (as opposed to the hardware) environment in which user programs run. The ISIS-PDS operating system provides two levels of software environment: both an operating and a programming environment. See figure 8-1.

The operating environment is the external environment of the iPDS system. It is the environment with which an operator interacts when running programs on the system.

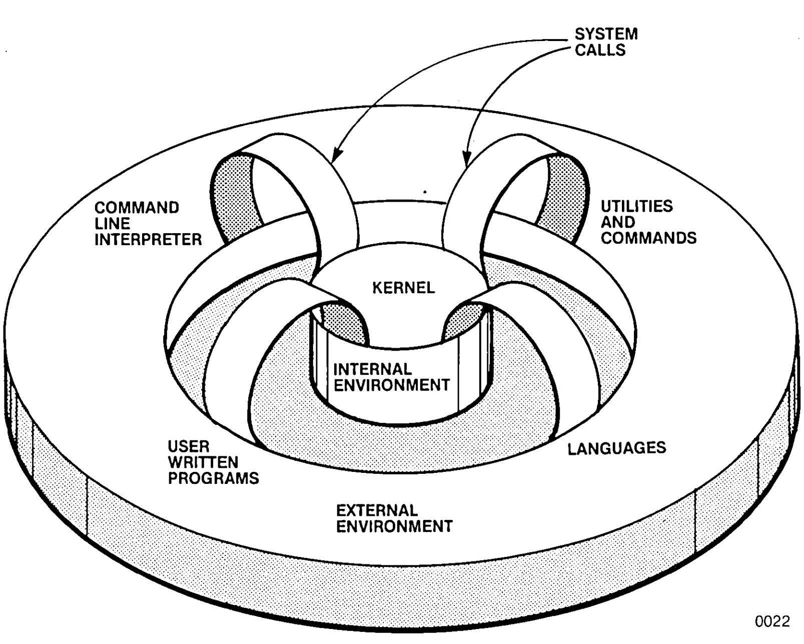


Figure 8-1 Internal and External Environment

The external environment is provided by the Command Line Interpreter (CLI). The CLI interprets command lines entered from the console (usually the keyboard) and then loads and executes the corresponding program from a disk file.

Chapter 3 describes the characteristics of this environment, i.e., how to enter commands. Chapters 4-7 and Chapter 10 describe some of the command programs supplied with the system. Other commands are described in separate manuals.

The programming environment is the internal environment of the iPDS system. It is the environment in which the programs interact as they run on the system. It is the environment which a systems programmer sees when writing programs to run on the system.

The internal environment of the iPDS system is provided by a set of system calls. The system calls activate routines within the ISIS-PDS internal environment. This internal environment is described in this chapter.

Needed Functions

Operating systems typically provide features that aid in the software development task as shown in figure 8-2.

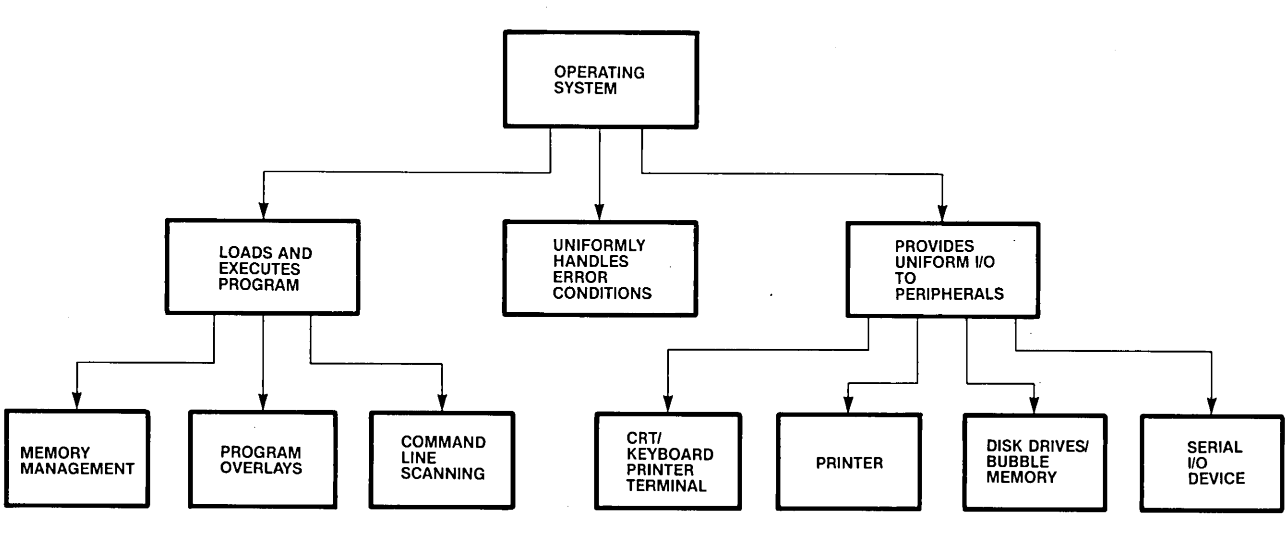


Figure 8-2 Needed Capabilities

Most operating systems control the execution of programs, such as language translators, utilities, or application programs. This involves several functions, such as managing the memory resources of the system, scanning command lines, and managing overlays.

The typical functions of an operating system also include the management of peripheral devices attached to the system hardware. Devices supported often include a console device such as a keyboard and CRT display, a hardcopy output device such as a line printer, and mass storage devices such as flexible disk drives or bubble memory.

Many operating systems also provide a uniform method of handling exception and error conditions detected during the execution of a program.

Features of the ISIS-PDS Operating System

The ISIS-PDS operating system provides all the features needed in an operating system through a collection of routines called the KERNEL. These routines, referred to as system calls, form the internal environment which a programmer sees when writing software to run on the iPDS system.

System Calls

This section describes the system calls in detail in terms of their operations and the parameters required from the user program. First, an overview is given of the system calls and their functions.

Overview of System Calls

System call routines are part of the ISIS operating system. They can be called by user-written programs to perform I/O and other system services on the Personal Development System. They free the programmer from rewriting routines, such as I/O routines, that are already embedded in the operating system, and they also provide a standard interface for all modules and systems developed.

Under the ISIS-PDS operating system, a number of MCS-80/85 language translators can be used to build program modules. The resulting programs then run under ISIS on the iPDS system. In this environment, the program may call upon ISIS routines for a variety of services which already exist as part of the operating system. This frees the programmer from having to code routines that are already available.

An overview of the steps involved in making a system call from a PL/M module and from an ASM module is given next. Detailed descriptions of how to make system calls are provided later in the chapter.

When a system call is used in a PL/M program module, an external procedure is declared for it. Then, within the PL/M module, a procedure call is made to the externally declared procedure.

In an ASM-80 program, an external symbol is declared for the entry point to the system call. (In some cases, the external symbol is ISIS; in others, it is the name of the system call.) Then, in the assembly module a CALL instruction to the external symbol is given.

In either case, with PL/M-80 or with ASM-80, the relocatable module is linked with the file SYSPDS.LIB which provides the correct address of the system call routine.

There are 27 system call routines available. These are grouped into two broad categories: high level routines and primitive routines. The high level routines perform I/O operations and related maintenance services at the file level; they also provide program execution services. The ISIS primitive routines perform I/O operations and related services at the byte level. The high level system calls are:

- File I/O operations for disk and other peripherals. OPEN, CLOSE, READ, WRITE, SEEK, RESCAN, SPATH
- Disk directory maintenance. ATTRIB, DELETE, RENAME
- Console device assignment. CONSOL, WHOCON
- Error message output. ERROR
- Program loading and execution and return to CLI. LOAD, EXIT
- Multimodule Sharing ATTACH, DETACH

The primitive routines are:

- Peripheral I/O routines. CI, CO, RI, PO, LO
- System status routines. CSTS, IOCHK, IOSET, MEMCK
- Custom I/O driver extension. IODEF

Functional Categories of System Calls

A summary of the high level and primitive system calls organized by function follows.

High Level System Calls

The high level system calls perform I/O operations at the file level. They also provide program loading and execution services as well as error handling and console I/O.

File I/O Operations. Seven system calls are available for controlling file I/O for disk and other peripherals: OPEN, CLOSE, READ, WRITE, SEEK, RESCAN, and SPATH. These routines open files for read or write operations, move the pointer in an open file, and close files when they are finished.

With these calls, a program can transfer variable length blocks of data between standard peripheral devices and a memory buffer area in the user program. To clarify the effect of system calls on files, two integer quantities, LENGTH and MARKER, are associated with each file in this description.

LENGTH is the number of bytes in the file. For some files, such as the keyboard input, the LENGTH is potentially infinite. For other files, such as the input from a serial device, the LENGTH is unknown until the file is completely read in. The LENGTH of a file increases as the file is written.

MARKER is the number of bytes already read from or written to the file. It is only associated with open files. The range of MARKER is zero to LENGTH. MARKER is a pointer to the next byte to be read from or written to the file. The value of MARKER can be changed with the SEEK and RESCAN system calls.

Disk Directory Maintenance. There are three system calls in this group: ATTRIB, DELETE, and RENAME. They perform maintenance functions on disk directories. ATTRIB changes the attributes of a file in the directory, DELETE removes a file from the directory, and RENAME assigns a new name to a file in the directory.

Console Device Assignment. There are two system calls in this group: CONSOL and WHOCON. They perform control functions for the logical console device. CONSOL assigns the console devices to physical devices and WHOCON returns the name of the current console device.

Error Message Output. The ERROR system call allows a program to send a message to the console.

Program Loading and Execution. The LOAD and EXIT system calls run programs from disk files and return control to the CLI. The LOAD system call can be used to transfer control to an overlay and then back to the main program.

I/O Driver Extensions. Two system calls aid in adding user defined I/O drivers to the system. ATTACH and DETACH are used to assign non-bubble multimodule devices to one of the processors before the processor can access those devices.

Primitive System Calls

The ISIS primitives perform I/O operations at the byte level. They also perform system status checking and provide for adding I/O drivers to the operating system. The ISIS-PDS primitive system calls are the equivalent of monitor calls on the Series II and the Intellec 800 development systems. The calls are used to talk to the chips that interface to the device driver.

Peripheral I/O Routines. The five I/O routines provide standard I/O interface to the console (CI and CO), list device (LO), and a serial device (RI and PO).

System Status Routines. The four system status routines allow the user program to check the console status (CSTS), to check the assignment of I/O devices (IOCHK), and to check the top of user memory (MEMCK). Additionally, the IOSET routine assigns I/O devices.

I/O Driver Extensions. The IODEF system call adds I/O drivers to the system.

Differences Between High Level and Primitive System Calls

The physical devices used with a high level call are the logical devices for the primitive system call. Bubble memory and disk drives do not have primitive system calls.

System Call Format and Use

All system calls alter the CPU registers; save any values it needed later. None of the system calls use the stack.

The ISIS-PDS system routines can be called from programs written in a number of MCS-80/85 languages. If a system call is made in a program, that program must be linked with the file SYSPDS.LIB using the LINK program to supply the absolute addresses of the system routines to the calling program.

Some of the system call primitives are used as functions. In PL/M, they are declared as a typed procedure and can be invoked as part of an expression or a parameter list rather than being CALLed. The calling sequence given for each system routine indicates whether it is invoked by a CALL or is used as a function. In assembly language, the functions return values to a register.

PL/M Calls

To use ISIS system calls in a PL/M program:

1. Declare a procedure as an external with the name of the system call desired. Include the variable declarations for all formal parameters used by the system call.

2. Before the procedure is called, declare and assign the proper values to the actual parameters to be passed to the procedure.

3. Invoke the procedure, passing the parameters.

A PL/M program interfaces to ISIS by performing calls to procedures. The PL/M program must include external procedure declarations so that the proper procedures (declared as public procedures) from SYSPDS.LIB will be included in the program by LINK. For most system calls, the procedure declaration is untyped; but some system calls serve as functions and, therefore, are typed as either BYTE or ADDRESS.

The external procedure declaration must also include variable declarations for the parameters to be passed to the procedure. The parameters can be of type ADDRESS or of type BYTE, depending on the system call.

Each system call description gives the calling sequence and an example.

Assembly Language Calls

To use ISIS system calls in assembly language programs:

1. Define the system call entry point as an external.

2. Make any EQUs, DWs, or DSs needed for parameters to be passed to the routine.

3. Load the value of the parameters to be passed into the proper register or memory location.

4. Use the assembly language CALL instruction to call the system routine entry point, previously defined as an external.

The interface between the MCS-80/85 Assembly Language Program and ISIS is accomplished by declaring an external symbol for the entry point to the system routine and then using the CALL instruction to that externally defined label. The symbol to define as an external for each system call is given in the description of that system call.

Parameters are passed from an assembly language program using the processor registers and memory as required by the routine. These are specified in the description of the particular system call.

Assembly language calls to system calls change the contents of the processor registers. If the value of a register must be preserved, it should be saved before the system call.

Each system call description includes the calling sequence and gives an example.

Assembly Language Calls to High Level System Routines. Assembly language system calls differ from PL/M in several respects. In PL/M, each system routine has a corresponding procedure that is declared and called. In assembly language, all the high level system routines are accessed by a CALL to a single entry point labelled ISIS. (The remaining ISIS primitives use a mnemonic for the routine as the external label for the entry point. These mnemonics are described in the discussion of the particular call.)

For all high level system calls, the C register must contain the number of the system call desired and the DE register pair must contain the address in memory of a table of parameters required by the system call. The detailed steps follow:

1. Define ISIS as an external label:

   ```
   EXTRN ISIS
   ```

2. Use an EQU statement to equate the number of the desired system call with a symbol according to the following table:

SYSTEM CALL	NUMBER
OPEN	0
CLOSE	1
DELETE	2
READ	3
WRITE	4
SEEK	5
LOAD	6
RENAME	7
CONSOL	8
EXIT	9
ATTRIB	10
RESCAN	11
ERROR	12
WHOCON	13
SPATH	14
ATTACH	15
DETACH	16

3. Use DS and DW statements to set up a template for a table to hold the required parameters.

4. With assembly language instructions, store the appropriate values for the parameters in the table.

5. Store the number of the system call in the C register.

6. Store the address of the table of parameters in the D and E register pair.

7. Insert the following instruction in the program:

   ```
   CALL ISIS    ;Call the operating system.
   ```

The file SYSPDS.LIB must be linked to the program and provides the absolute address of the ISIS entry point.

All the high level system calls require two parameters. The first is a number, passed in register C, that identifies the system call; the second is an address, passed in register pair DE, specifying the memory location of the additional parameters required by the system call.

Usually, the system call numbers are defined in EQU statements before they are referenced in the program, so that they can be referenced symbolically. The table shown in step two above lists the high level system calls by number. These equates are assumed in the assembly language examples given throughout this chapter. Only the specific calls used in a program need to be defined in that program.

Assembly Language Calls to Primitive System Routines. Calls to ISIS primitives are similar to other system calls. The difference is that each ISIS primitive has its own mnemonic label for its entry point instead of the single ISIS label. These labels are defined as externals. The parameters required by ISIS primitives vary, and they are stored in the processor registers before the call instruction is given. The CALL is made to the label for the entry point and any values returned are stored in the processor registers.

The mnemonics used for the entry points are:

```
CI
CO
LO
RI
PO
CSTS
IOCHK
IODEF
IOSET
MEMCK
```

For example, the call to output a character to the console is:

```
EXTRN CO ;ENTRY POINT FOR CO SYSTEM CALL
      .
      .
      .
MOV C,M  ;LOAD CHARACTER TO BE DISPLAYED INTO THE C
          REGISTER
CALL CO  ;OUTPUT THE CHARACTER
```

Error Handling

Most of the high-level system calls return fatal and non-fatal error numbers in a status byte that can be tested by the calling program.

If a non-fatal error occurs, no action is performed by the system call and control returns to the calling program. The error number is returned as a two-byte value to the calling program.

If a fatal error occurs, a message containing the error number is displayed on the CRT screen and the CLI is reloaded.

The error numbers returned by each system routine are listed in the description of that routine. Error numbers are described in Appendix B.

System Calls in Alphabetical Order

In this section, each system routine is described. Certain conventions are followed in the descriptions so the information can be easily accessed. For every description, notational conventions, general format terms, and a similar format are used.

Notational Conventions

The notational conventions used here are consistent with the conventions used throughout the manual as described in Chapter 5.

General Format Terms

The parameters passed to system calls assume certain uses. The name chosen for each parameter indicates its use. All the general terms used for parameter names are listed in this section after a brief discussion that highlights some of them.

A name used by every routine, is <status$ptr>. This is the address of a location which contains a non-zero error code if the system call could not complete its task normally.

Two other parameter names are <conn> and <conn$ptr>. ISIS maintains a list of twelve devices or files that a program can access during its execution, i.e., a table of file access connections.

The connection is a number, named and declared in the user program, that corresponds to a file to be accessed from the program. In other words, it connects the file to the user program. The user program also supplies the pathname of the device or file as an ASCII string. The ASCII string must conform to the format given for pathnames:

:<device name>:<filename>.<extension>

However, it may have leading spaces (ASCII code 20H). The pathname cannot be terminated by a letter, a digit, a colon (:), or a period (.); but a space may be used. The OPEN system call maintains this pathname and its connection number as an entry in a table. Thereafter, the connection specifies the file or device that the user program can read, write, seek, rescan or close.

This table of connections is also called an Active File Table, and the entries in it are Active File Table Numbers (AFTNs). Only files with AFTNs, i.e., that are in the Active File Table, can be used for I/O operations. Reads, writes, opens, and all other file operations refer to the connection or AFTN rather than the device and file name. During execution, the program can access multiple files, but only six may be open at one time (not counting the console input and console output). When I/O actions for a given file are complete, it can be closed so that another file can be opened.

Be careful not to confuse the AFTN with the PL/M construction .AFTN. The period (.) specifies the address of the memory location where the AFTN is stored.

To reduce the potential confusion, the term <conn> is used to refer to the connection number and <conn$ptr> to refer to the address of the connection number (the pointer to the connection number). The <$ptr> is appended to another term to indicate a pointer to the other item. In PL/M examples, the dot operator (.) precedes names to indicate the address of the name.

Similarly, <path$ptr> represents the address of a memory location containing the pathname string. In PL/M calls to the system routines, simply use the dot operator (.) to provide the address of the variable declared for use in the system routine.

In order to READ and WRITE a file, several questions must be answered:

1. How many bytes are to be transferred?
2. To (or from) where?
3. From (or to) what memory locations?

In the system call descriptions, (1) is supplied as the parameter <count>; (2) is supplied as the parameter <conn> described previously; and (3) is supplied as the parameter <buffer$ptr>, the address of the locations to be read from or written to.

The following chart lists the parameters in alphabetical order and lists the system calls in which they are used.

Parameter Name	Routines Using Parameter and Brief Definition
<access>	OPEN 2-byte number telling how the file is to be used, i.e., read or write or update.
<actual$ptr>	READ 2-byte pointer to the actual number of bytes successfully read.
<atrb>	ATTRIB 2-byte number indicating which attribute to change.
<block$ptr>	SEEK 2-byte pointer to the block number.
<buf$ptr>	READ, WHOCON, WRITE 2-byte pointer to the area declared for reading from (or writing to) a file; for read and write, it should be at least COUNT bytes long or undefined results will occur.
<byte$ptr>	SEEK 2-byte pointer to the byte number.
<char>	CO, LO, PO Byte value output to the console, the serial device, or the printer.
<ci$path$ptr>	CONSOL 2-byte pointer to ASCII string containing the pathname of the console input device.

Parameter Name	Routines Using Parameter and Brief Definition (continued)
<config$byte>	IOSET Byte value used to assign I/O devices.
<conn>	CLOSE, READ, RESCAN, SEEK, WHOCON, WRITE 2-byte connection number to a file or device.
<conn$ptr>	OPEN 2-byte pointer to the connection number.
<control$sw>	LOAD 2-byte value indicating where to transfer control after the load.
<co$path$ptr>	CONSOL 2-byte pointer to ASCII string containing the path-name of the console output device.
<count>	READ, WRITE 2-byte value that specifies the number of bytes to read from or write to a file.
<echo>	OPEN 2-byte connection number for the echo file when a line edited file is opened.
<entry$point>	IODEF 2-byte address of the entry point of the user written I/O driver.
<entry$ptr>	LOAD 2-byte address of the location to which the loaded program should return after execution.
<errnum>	ERROR 2-byte error number to output to the console.
<function$code>	IODEF Byte value that identifies which I/O driver is being added.
<info$ptr>	SPATH 2-byte pointer to the memory area containing file description data.
<load$offset>	LOAD 2-byte offset value added to the load address causing the program to load at the adjusted address.
<mode>	SEEK 2-byte value representing the direction and type of the seek operation.
<mmio$row>	ATTACH, DETACH 2-byte value specifying which multimodule row is being attached or detached.

Parameter Name	Routines Using Parameter and Brief Definition (continued)
<newpath$ptr>	RENAME 2-byte pointer to the new pathname of the file being renamed.
<oldpath$ptr>	RENAME 2-byte pointer to the old pathname of the file being renamed.
<onoff>	ATTRIB 2-byte number indicating whether the attribute is to be set or reset.
<path$ptr>	ATTRIB, DELETE, LOAD, OPEN, SPATH 2-byte pointer to the pathname of the accessed file.
<status$ptr>	All routines but EXIT and the primitives. 2-byte pointer to the error numbers generated during the system call.

Description Formats

In addition to the conventions just described, each of the system calls is described using the same format to aid in accessing the needed information at a glance. This format is shown in figure 8-3. The system calls are described in alphabetical order. Each description begins on a new page with the system call keyword on the top outside margin of the page.

1. First, the purpose of the system call is given.

2. Second, the parameters are described in the order required by the system call. Two pieces of information are supplied for every parameter.

   "Input parameter." or "Output parameter." is the first information given for each parameter. All the parameters listed are supplied by the calling program to the system call. In this context, output parameter refers to the destination address of a value returned by the system call.

   The second piece of information provided for every parameter is the parameter size. All parameters are either one-byte or two-byte. All high level system calls use two-byte parameters.

3. Third, the error numbers are listed. These are the ISIS error numbers that can be returned for that system call. The error numbers are explained in Appendix B.

4. Fourth, a PL/M section shows the format of the PL/M procedure declaration and the format of the PL/M procedure call and gives an example.

5. Fifth, an assembly language section shows shows the calling sequence in an ASM-80 program and gives an example.

NOTE

The examples provided use symbolic names, labels, and variable names. If several of the example calling sequences are combined into a single program, the symbols, labels, and variable names may have to be modified to avoid duplicate symbol errors.

SYSTEM CALL NAME

brief phrase
describing routine

Description

A brief paragraph or two describing use of the system call.

Parameters

Parameter 1	.... Description ....
. . .	
Parameter n	.... Description ....

Error Numbers

Fatal:	n1, n2, n3, ..., nm
Non-Fatal:	n1, n2, n3, ... , nm

PL/M Calling Sequence

Explanation and example of the calling sequence.

Assembly Language Calling Sequence

Explanation and example of the calling sequence.

0224

Figure 8-3 Format of System Call Descriptions

ATTACH

Assigns multimodule row to processor

Description

The ATTACH call assigns a row of multimodules to a processor. The program supplies the multimodule row, and the ATTACH call returns the status. See the DETACH call.

Parameters

Two parameters are required by ATTACH in the following order. They can be passed in a PL/M procedure or through the C register and the DE register pair from an assembly language program.

<mmio$row> Input parameter. Two byte. Value can be:

0 - I/O Connectors J1 and J2.
1 - I/O Connectors J3 and J4.

<status$ptr> Output parameter. Two byte. Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 33.

Non-fatal: 60, 61.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
ATTACH:
  PROCEDURE (<mmio$row>,<status$ptr>) EXTERNAL;
    DECLARE (<mmio$row>,<status$ptr>) ADDRESS;
  END ATTACH;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE MULTIMODULE ADDRESS;
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL ATTACH(MULTIMODULE,.STATUS);
IF STATUS < > 0 THEN ...
```

Notice that a variable was declared for the status, and then the dot operator was used to pass the address of this value as required by the system call.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled ABLK) is stored in the DE register pair before the CALL instruction.

```
; ATTACH
;
          EXTRN  ISIS       ; LINK TO ISIS ENTRY POINT
ATTACH    EQU    15         ; SYSTEM CALL IDENTIFIER
                            ; SAVE REGISTERS...
          MVI    C,ATTACH   ; LOAD IDENTIFIER
          LXI    D,ABLK     ; ADDRESS OF PARAMETER BLOCK
          CALL   ISIS
          LDA    ASTAT      ; TEST ERROR STATUS
          ORA    A
          JNZ    EXCEPT     ; BRANCH TO EXCEPTION ROUTINE
                            ; REST OF PROGRAM
ABLK:                       ; PARAMETER BLOCK FOR ATTACH
ATTROW:   DS     2          ; MULTIMODULE ROW
ASTAT:    DS     2          ; ATTACH STATUS
```

ATTRIB

Change the attributes of a file

Description

The ATTRIB call allows a program to change an attribute of a disk file. The program supplies the name of the file, the attribute to be changed, and its new value. The ATTRIB call returns a status code.

Parameters

Four parameters are required by ATTRIB in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<path$ptr> Input parameter. Two byte. Pointer to the ASCII string (maximum of 15 bytes) containing the name of the file. The ASCII string can contain leading spaces but no embedded spaces. It must be terminated by a character other than a letter, digit, colon (:), or period (.). A space can be used.

<atrb> Input parameter. Two byte. Number indicating which attribute is to be changed:

0 - invisible attribute
1 - system attribute
2 - write protect attribute
3 - format attribute
4 - user defined attribute
5 - user defined attribute
6 - user defined attribute

<onoff> Input parameter. Two byte. Value indicating whether attribute is to be set (turned on) or reset (turned off). The value is stored in the low order bit of the low order byte. A value of 1 specifies the attribute to be set, and a value of 0 specifies that it be reset.

<status$ptr> Output parameter. Two byte. Address in memory of the two-byte error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 1, 24, 30, 33

Non-fatal: 4, 5, 13, 23, 26, 28

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
ATTRIB:
  PROCEDURE (<path$ptr>,<atrb>,<onoff>,<status$ptr>)
  EXTERNAL;
    DECLARE (<path$ptr>,<atrb>,<onoff>,<status$ptr>)
    ADDRESS;
  END ATTRIB;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE FILE(15) BYTE;
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing the actual parameters.

```
CALL ATTRIB(.FILE,2,0,.STATUS);
IF STATUS <> 0 THEN . . .
```

Notice that a variable was declared for the filename and the status, and then the dot operator was used to pass the addresses of these values as required by the system call. This example clears the Write Protect attribute of the file.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled ABLK) is stored in the DE register pair before the CALL instruction.

```
; ATTRIB
;
         EXTRN  ISIS          ; LINK TO ISIS ENTRY POINT
ATTRIB   EQU    10            ; SYSTEM CALL IDENTIFIER
                              ; SAVE REGISTERS ...
         MVI    C,ATTRIB      ; LOAD IDENTIFIER
         LXI    D,ABLK        ; ADDRESS OF PARAMETER BLOCK
         CALL   ISIS
         LDA    ASTAT         ; TEST ERROR STATUS
         ORA    A
         JNZ    EXCEPT        ; BRANCH TO EXCEPTION ROUTINE
                              ; REST OF PROGRAM
ABLK:                         ; PARAMETER BLOCK FOR ATTRIB
         DW     FILE3         ; ADDRESS OF FILENAME
ATRB:    DW     0             ; ATTRIBUTE IDENTIFIER
VALUE:   DW     I             ; ATTRIBUTE VALUE
         DW     ASTAT         ; POINTER TO STATUS
;
ASTAT:   DS     2             ; STATUS (RETURNED)
FILE3:   DB     'OPSYS.CLI '
```

CI

Input character
from console

Description

The CI call reads a character entered at the system console device and returns it as a byte variable in PL/M or in the A register in an assembly language program. Once called, the routine loops until a character is entered. The character entered is not echoed on the console output device. The actual device from which the character is read depends on the value set by the IOSET system call. The CI system call is *not* used by the ISIS operating system to read the :CI: device.

Parameters

None.

Error Numbers

Fatal: None.

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
CI:
   PROCEDURE BYTE EXTERNAL;
   END CI;
```

Notice that this is a typed procedure or function and can be used in an expression or parameter list. There are no actual parameters to declare, so the function is called without any further declarations. Assuming an array has been set up to receive a sequence of characters, the following program receives those characters into the array.

```
DO WHILE BUFFER(INDEX) <> CR;
     INDEX = INDEX + 1
     BUFFER(INDEX) = CI;
     END;
```

Assembly Language Calling Sequence

From an assembly language program, the label CI is defined externally instead of the label ISIS, because CI is a primitive ISIS routine. The character is returned in the A register.

```
; CI
     EXTRN  CI  ; ENTRY POINT FOR CI IN ISIS
     CALL   CI  ; GET CHARACTER
                ; CHARACTER RETURNED IN "A"
                ; REGISTER
```

CLOSE

Terminate I/O operations to a file

Description

The CLOSE call removes a file from the Active File Table (AFT) and releases the buffers allocated for it by the OPEN command. Each file should be closed whenever I/O processing is complete. The program supplies the connection number of the file to be closed and the CLOSE call returns an error code. The EXIT system call also closes all open files except the console.

Parameters

Two parameters are required by CLOSE in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<conn> Input parameter. Two byte. Connection number (AFTN) returned for a random access file when it was opened.

<status$ptr> Output parameter. Two byte. Pointer to two-bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 33

Non-fatal: 2

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
CLOSE:
  PROCEDURE (<conn>,<status$ptr>) EXTERNAL;
    DECLARE (<conn>,<status$ptr>) ADDRESS;
  END CLOSE;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE AFT$IN ADDRESS;
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL CLOSE(AFT$IN,.STATUS);
IF STATUS <> 0 THEN ...
```

Notice that a variable was declared for the status, and then the dot operator was used to pass the addresses of this value as required by the system call.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled CBLK) is stored in the DE register pair before the CALL instruction.

```
; CLOSE
;
        EXTRN ISIS       ; LINK TO ISIS ENTRY POINT
CLOSE   EQU   1          ; SYSTEM CALL IDENTIFIER
        .                ; SAVE REGISTERS . . .
        MVI   C,CLOSE    ; LOAD IDENTIFIER
        LXI   D,CBLK     ; ADDRESS OF PARAMETER BLOCK
        CALL  ISIS
        LDA   CSTAT      ; TEST ERROR STATUS
        ORA   A
        JNZ   EXCEPT     ; BRANCH TO EXCEPTION ROUTINE
; ...
CBLK:                    ; PARAMETER BLOCK FOR CLOSE
CAFTN:  DS    2          ; FILE IDENTIFIER
        DW    CSTAT      ; POINTER TO STATUS
;
CSTAT:  DS    2          ; STATUS (RETURNED)
```

CO

Output character
to console

Description

The CO call sends a single character to the system console device. From PL/M, a byte variable is used for the character. From an assembly language program, the character must be in the C register. The actual device to which the character is written is determined by the value set by the IOSET system call. The CO system call is *not* used by the ISIS-iPDS operating system to write to the :CO: device.

Parameters

The parameter required by CO can be passed in a PL/M procedure call or through the C register from an assembly language program.

<char> Input parameter. Byte. The character to be output.

Error Numbers

Fatal: None.
Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
CO:
  PROCEDURE (<char>) EXTERNAL;
    DECLARE (<char>) BYTE;
  END CO;
```

The actual parameter must be declared in the program prior to making the call.

```
DECLARE CHAR BYTE;
```

A value is then assigned to the declared variable which is used as an actual parameter. Finally, the procedure is called passing the actual parameter.

```
CALL CO(CHAR);
```

Assembly Language Calling Sequence

From an assembly language program, the label CO is defined externally instead of the label ISIS, because CO is a primitive ISIS routine.

```
;CO
                 ; SAVE REGISTERS
     EXTRN CO    ; ENTRY POINT FOR CO ISIS
     MOV   C,M   ; LOAD CHARACTER TO BE OUTPUT INTO "C"
                 ; REGISTER
     CALL  CO    ; OUTPUT THE CHARACTER
```

CONSOL

Change
console device

Description

The CONSOL call allows a program to change the console input and output devices, i.e., reassign the physical device named by :CI: and :CO:. The program supplies the name of the new console I/O device. The CONSOL call returns an error code.

Parameters

Three parameters are required by CONSOL in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<ci$path$ptr> Input parameter. Two byte. Pointer to the ASCII string (15-bytes maximum) containing the name of the file to be used for console input. The ASCII string can contain leading spaces but no embedded spaces. It must be terminated by a character other than a letter, digit, colon (:), or period (.). A space can be used. If the specified file cannot be opened, a fatal error occurs. If :CI: is specified for the input file, the current input assignment is not changed.

<co$path$ptr> Input parameter. Two byte. Pointer to the ASCII string (15-byte maximum) containing the name of the file to be used for console output. The ASCII string can contain leading spaces but no embedded spaces. It must be terminated by a character other than a letter, digit, colon (:), or period (.). A space can be used. If the specified file cannot be opened, a fatal error occurs. If :CO: is specified for the output file, the current output assignment is not changed.

<status$ptr> Output parameter. Two byte. Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 1, 4, 5, 12, 13, 14, 22, 23, 24, 28, 30, 33

Non-fatal: None; all errors are fatal.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
CONSOL:
  PROCEDURE (<ci$path$ptr>,<co$path$ptr>,<status$ptr>)
  EXTERNAL;
    DECLARE (<ci$path$ptr>,<co$path$ptr>,<status$ptr>)
    ADDRESS;
  END CONSOL;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE INFILE(6) BYTE;
DECLARE OUTFILE(6) BYTE;
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL CONSOL(.INFILE,.OUTFILE,.STATUS);
IF STATUS <> 0 THEN ...
```

Notice that variables were declared for the filenames and status, and then the dot operator was used to pass the addresses of these values as required by the system call.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (CBLK) is stored in the DE register pair before the CALL instruction.

```
;
CONSOL
;
        EXTRN ISIS         ; LINK TO ISIS ENTRY POINT
CONSOL  EQU   8            ; SYSTEM CALL IDENTIFIER
                           ; SAVE REGISTERS ...
        MVI   C,CONSO      ; LOAD IDENTIFIER
              L
        LXI   D,CBLK       ; ADDRESS OF PARAMETER BLOCK
        CALL  ISIS
;...
CBLK:                      ; PARAMETER BLOCK FOR CONSOLE
        DW    INPUT        ; POINTER TO NAME OF CONSOLE
                           ; INPUT FILE
        DW    OUTPUT       ; POINTER TO CONSOLE OUTPUT
                           ; FILE
        DW    CSTAT        ; POINTER TO STATUS
;
CSTAT:  DS    2            ; STATUS (RETURNED)
INPUT:  DB    ':SI: '      ; INPUT CONSOLE FILE
OUTPUT: DB    ':SO: '      ; OUTPUT CONSOLE FILE
```

CSTS

Return console
input status

Description

The CSTS call tests the console device to determine if a character is ready for input. It returns a value of 00H if no character is ready for input since the last console status and a value of FFH if a character is ready. The status is returned as a byte variable in PL/M or in the A register in an assembly language program. The console device and console status routines can be changed by using the IOSET system call.

Parameters

None.

Error Numbers

Fatal: None.

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
CSTS:
   PROCEDURE BYTE EXTERNAL;
   END CSTS;
```

Notice that this is a typed procedure or function and can be used in an expression or parameter list.

There are no actual parameters to declare, so the function is called without any further declarations. In the following example, the status is used as a test condition for another routine.

```
IF CSTS THEN . . .
```

Assembly Language Calling Sequence

From an assembly language program, the label CSTS is defined as an external instead of the label ISIS, because CSTS is a primitive ISIS routine. The status is returned in the A register.

```
; CSTS
          EXTRN  CSTS  ; ENTRY POINT FOR CSTS IN ISIS
                       ; SAVE REGISTERS
          CALL   CSTS  ; GET CHARACTER
                       ; CHARACTER RETURNED IN "A"
                       ; REGISTER
```

DELETE

Delete a file from the disk directory

Description

The DELETE call removes a specified file from its disk. The file must not be open. The disk space allocated to the file is released and can be used for another file. The program supplies a pointer to the name of the file to be deleted. The DELETE call returns an error code.

Parameters

Two parameters are required by DELETE in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<path$ptr>	Input parameter. Two byte. Pointer to the ASCII string (15-byte maximum) containing the name of the file. The ASCII string can contain leading spaces but no embedded spaces. It must be terminated by a character other than a letter, digit, colon (:), or period (.). A space can be used.
<status$ptr>	Output parameter. Two byte. Pointer to two-bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 1, 24, 30, 33

Non-fatal: 4, 5, 13, 14, 17, 23, 28, 32

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
DELETE:
   PROCEDURE (<path$ptr>,<status$ptr>) EXTERNAL;
      DECLARE (<path$ptr>,<status$ptr>) ADDRESS;
   END DELETE;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE FILENAME(15) BYTE;
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL DELETE(.FILENAME,.STATUS);
IF STATUS <> 0 THEN ...
```

Notice that a variable was declared for the filename and status, and then the dot operator was used to pass the addresses of these values as required by the system call.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled DBLK) is stored in the DE register pair before the CALL instruction.

```
; DELETE
;
        EXTRN ISIS           ; LINK TO ISIS ENTRY POINT
DELETE  EQU   2              ; SYSTEM CALL IDENTIFIER
                             ; SAVE REGISTERS . . .
        MVI   C,DELETE       ; LOAD IDENTIFIER
        LXI   D,DBLK         ; ADDRESS OF PARAMETER BLOCK
        CALL  ISIS
        LDA   DSTAT          ; TEST ERROR STATUS
        ORA   A
        JNZ   EXCEPT         ; BRANCH TO EXCEPTION ROUTINE
; ...
DBLK:                        ; PARAMETER BLOCK FOR DELETE
        DW    DFILE          ; POINTER TO FILENAME
        DW    DSTAT          ; POINTER TO STATUS
;
DSTAT:  DS    2              ; STATUS (RETURNED)
DFILE:  DB    'FILE.EXT '    ; NAME OF FILE TO BE DELETED
```

DETACH

Release multimodule row from a processor

Description

The DETACH call releases a row of multimodules from a processor. The program supplies the multimodule row, and the DETACH call returns an error. See the ATTACH call.

Parameters

Two parameters are required by DETACH in the following order. These can be passed in a PL/M procedure or through the C register and the DE register pair from an assembly language program.

<mmio$row> Input parameter. Two byte. Value can be:

0 - I/O Connectors J1 and J2.
1 - I/O Connectors J3 and J4.

<status$ptr> Output parameter. Two byte. Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 33.

Non-fatal: 60, 61.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
DETACH:
   PROCEDURE (<mmio$row>,<status$ptr>) EXTERNAL;
      DECLARE (<mmio$row>,<status$ptr>) ADDRESS;
   END DETACH;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE MULTIMODULE ADDRESS;
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL DETACH(MULTIMODULE,.STATUS);
IF STATUS <> 0 THEN ...
```

Notice that a variable was declared for the status, and then the dot operator was used to pass the address of this value as required by the system call.

Assembly Language Calling Sequence

From an assembly language program, the label DETACH is defined as an external instead of the label ISIS, because DETACH is a primitive ISIS routine. The multi-module row must be in the C register.

```
; DETACH
;
          EXTRN ISIS       ; LINK TO ISIS ENTRY POINT
DETACH    EQU   15         ; SYSTEM CALL IDENTIFIER
                           ; SAVE REGISTERS ...
          MVI   C,DETACH   ; LOAD IDENTIFIER
          LXI   D,DBLK     ; ADDRESS OF PARAMETER BLOCK
          CALL  ISIS
          LDA   DSTAT      ; TEST ERROR STATUS
          ORA   A
          JNZ   EXCEPT     ; BRANCH TO EXCEPTION ROUTINE
;
;...
DBLK:                      ; PARAMETER BLOCK FOR LOAD
DETROWDS        2          ; MULTIMODULE ROW
:
DSTAT:    DS    2          ; DETACH STATUS
```

ERROR

Output error message on system console

Description

The ERROR call allows a program to send an error message to the current console output device. The program supplies the ISIS error number of the message to be displayed. The ERROR call returns a status code.

Parameters

Two parameters are required by ERROR in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<errnum> Input parameter. Two byte. Error number to display on the CRT screen. The error number must be in the low order eight bits of the parameter. Only the numbers 101 to 199 inclusive should be used for user programs. the other numbers are reserved for ISIS. The system displays the message in the form:

ERROR nnn, USER PC mmmmm

where nnn is the error number specified in the call and mmmm is the return address of the calling program.

<status$ptr> Output parameter. Two byte. (for assembly language only). Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 33

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
ERROR:
  PROCEDURE (<errnum>) EXTERNAL;
    DECLARE (<errnum>) ADDRESS;
  END ERROR;
```

Notice that the status parameter is not used in the PL/M call.

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE ENUM ADDRESS
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL ERROR(ENUM);
```

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled EBLK) is stored in the DE register pair before the CALL instruction.

```
; ERROR
;
          EXTRN ISIS       ; LINK TO ISIS ENTRY POINT
ERROR     EQU   12         ; SYSTEM CALL IDENTIFIER
                           ; SAVE REGISTERS ...
          MVI   C,ERROR    ; LOAD IDENTIFIER
          LXI   D,EBLK     ; ADDRESS OF PARAMETER BLOCK
          CALL  ISIS
;...
EBLK:
ERRNUM:   DS    2          ; NUMBER OF ERROR TO BE PRINTED
          DW    ESTAT      ; POINTER TO STATUS
;
ESTAT:    DS    2          ; STATUS (RETURNED)
;
```

EXIT

Terminate program and
return to ISIS

Description

The EXIT call terminates execution and returns to ISIS-PDS. All open files are closed, with the exception of :CO: and :CI:. The current console assignment is not changed, unless the terminated program specifies a change. (See IOSET).

Parameters

From PL/M, no parameters are supplied. In an assembly language program, only the pointer to the location for the error code is supplied.

<status$ptr>	Output parameter. Two byte. (for assembly language only) Pointer to two bytes in memory reserved for the error number. If the error umber is 0, no error occurred.

Error Numbers

Fatal: None.

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
EXIT:
  PROCEDURE EXTERNAL;
  END EXIT;
```

There are no actual parameters to declare, so the call is made without any further steps. For example:

```
CALL EXIT;
```

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled EBLK) is stored in the DE register pair before the CALL instruction.

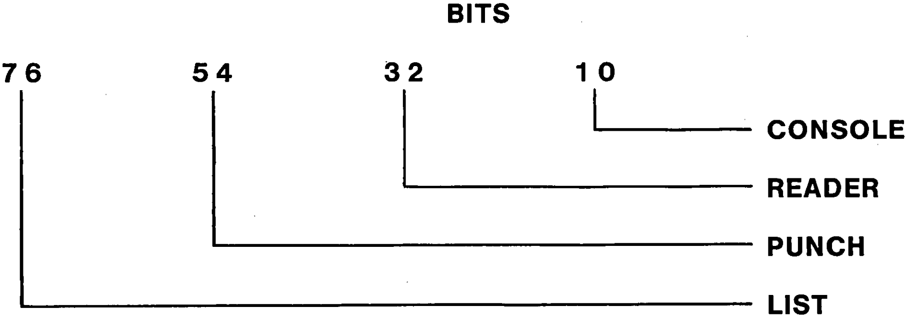
```
; EXIT
;
        EXTRN ISIS      ; LINK TO ISIS ENTRY POINT
EXIT    EQU   9         ; SYSTEM CALL IDENTIFIER
                        ; SAVE REGISTERS . . .
        MVI   C,EXIT    ; LOAD IDENTIFIER
        LXI   D,EBLK    ; ADDRESS OF PARAMETER BLOCK
        CALL  ISIS
; ...
EBLK:
        DW    ESTAT     ; POINTER TO STATUS
; ...
ESTAT:  DS    2         ; STATUS FIELD
```

IOCHK

Check system I/O configuration

Description

The IOCHK call is a function and returns a byte value that describes the current assignment of the physical devices to logical devices. The IOCHK call assignments are used by the low level system calls, not by the high level calls. The Logical devices are: Console, Reader, Punch, and List. The byte is divided into 4 two-bit fields interpreted as follows:



The terms Reader and Punch were used to be consistent with previous versions of ISIS that support paper tape reader and punch devices. Table 8-1 shows the possible values of each field in the byte returned and gives the physical device assignment for that value.

Table 8-1 Field Values and Physical Device Assignment

VALUE	LIST	PUNCH	READER	CONSOLE
00	SO	SO	SI	SI/SO
01	VO	VO	VI	VI/VO
10	LP	UD (4)	UD (2)	BATCH
11	UD (6)	UD (5)	UD (3)	UD (0/1/7)

SI is the serial input; SO is the serial output. VI is the video input (keyboard); VO is the video output (CRT screen). LP is the line printer. UD is a user defined device; the numbers in parentheses refer to the function codes in the IODEF call. BATCH assigns the current reader device as the console input and the current list device as the console output.

To isolate the two-bit field for a particular logical device from the byte returned, logically AND the IOCHK byte with one of the mask values shown in Table 8-2.

Table 8-2 Mask Values

DEVICE	HEX	BINARY
CONSOLE	03H	0000 0011B
READER	0CH	0000 1100B
PUNCH	30H	0011 0000B
LIST	C0H	1100 0000B

To check for a physical device code, AND the IOCHK byte with one of the following mask values:

CONSOLE	SI/SO	00H 0000 0000B	PUNCH	SO	00H 0000 0000B
	VI/VO	01H 0000 0001B		VO	10H 0001 0000B
	BATCH	02H 0000 0010B		UD(4)	20H 0010 0000B
	UD(0/1/7)	03H 0000 0011B		UD(5)	30H 0011 0000B
READER	SI	00H 0000 0000B	LIST	SO	00H 0000 0000B
	VI	04H 0000 0100B		VO	40H 0100 0000B
	UD (2)	08H 0000 1000B		LP	80H 1000 0000B
	UD (3)	0CH 0000 1100B		UD(6)	C0H 1100 0000B

Parameters

None.

Error Numbers

Fatal: None.

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
IOCHK:
   PROCEDURE BYTE EXTERNAL;
   END IOCHK;
```

Notice that this is a typed procedure or function and can be used in an expression or parameter list.

There are no actual parameters to declare, so the function is called without any further declarations.

```
IF IOCHK AND MASK1 = 0 THEN . . .
```

Assembly Language Calling Sequence

From an assembly language program, the label IOCHK is defined as an external instead of the label ISIS, because IOCHK is a primitive ISIS routine. The character is returned in the A register.

```
; IOCHK
          EXTRN  IOCHK ; ENTRY POINT FOR IOCHK IN ISIS
                       ; SAVE REGISTERS . . .
          CALL   IOCHK ; GET STATUS BYTE
                       ; CHARACTER RETURNED IN "A"
                       ; REGISTER
```

IODEF

I/O definition routine

Description

Up to eight user written I/O drivers can be added to extend ISIS I/O capabilities. Each driver can be assigned one of the following function codes.

0 User defined console input
1 User defined console output
2 User defined serial input 1
3 User defined serial input 2
4 User defined serial output 1
5 User defined serial output 2
6 User defined parallel output (list device)
7 User defined console status routine

Only one program can be assigned to any one code at a time. The program can be changed but two cannot be assigned to the same code at the same time. These codes correspond to the numeric assignment codes in the I/O configuration routine. See the IOSET call.

A user defined console device requires three routines: one for input, one for output, and one to check status. The user-written console status routine should return a value of 00H if a character is not ready for input and a value of FFH if a character is ready for input. The value should be returned in the A register for assembly language or in a byte variable for PL/M.

To link user written drivers to the ISIS primitive I/O routines, call IODEF and supply two parameters: the function code from the list above and the entry point of the driver routine. Do not locate drivers in RAM reserved for ISIS. Hardware information to aid in adding the I/O driver to the operating system can be found at the end of this chapter.

When IODEF is called, the driver must be present in memory. The driver can be loaded with the load system call.

Parameters

Two parameters are required by IODEF in the following order. They can be passed in a PL/M procedure or through the C register and the DE register pair from an assembly language program.

<function$code> Input parameter. Byte. Value from 0 to 7 corresponding to the driver being added. For example, 0 is used to add a console input routine. Refer to the previous list for the code.

<entry$point> Input parameter. Two byte. Pointer to the entry point for the I/O driver being added. The I/O driver must be in memory before IODEF is called.

Error Numbers

Fatal None.

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
IODEF:
   PROCEDURE (<function$code>,<entry$point>) EXTERNAL;
      DECLARE (<function$code>) BYTE;
      DECLARE (<entry$point>) ADDRESS;
   END IODEF;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE FCODE BYTE;
DECLARE ENTRY$POINT ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL IODEF(FCODE,ENTRY$POINT);
```

Assembly Language Calling Sequence

From an assembly language program, the label IODEF is defined as an external instead of the label ISIS, because IODEF is a primitive ISIS routine. The character to output must be in the C register.

```
; IODEF
        EXTRN IODEF         ; ENTRY POINT FOR IODEF IN ISIS
                            ; SAVE REGISTERS . . .
        MOV   C,M           ; LOAD FUNCTION CODE
        LXI   D,ADDRESS; LOAD REGISTER DE WITH ADDRESS
                            OF
                            ; I/O ROUTINE
        CALL  IODEF         ; CALL IODEF TO SET UP AN ISIS JUMP
                            TO
                            ; USER DEFINED I/O ROUTINE
```

IOSET

Set system I/O configuration

Description

The IOSET call modifies the system I/O configuration assignments. The program supplies the new configuration assignments as a byte value. Refer to IOCHK for a description of fields in this byte.

Parameters

One parameter is required by IOSET. It can be passed in a PL/M procedure or through the C register from an assembly language program.

<config$byte>	Input parameter. Byte. Value coded to contain the new configuration assignments. See IOCHK for a description of the fields in this byte.

Error Numbers

Fatal: None.

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
IOSET:
   PROCEDURE (<config$byte>) EXTERNAL;
      DECLARE (<config$byte>) BYTE;
   END IOSET;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE CONFIG$BYTE BYTE;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL IOSET(CONFIG$BYTE);
```

Assembly Language Calling Sequence

From an assembly language program, the label IOSET is defined as an external instead of the label ISIS, because IOSET is a primitive ISIS routine. The character to output must be in the C register.

```
; IOSET
            EXTRN IOSET ; ENTRY POINT FOR IOSET IN ISIS
                        ; SAVE REGISTERS . . .
            MOV   C,M   ; LOAD CONFIGURATION CODE
            CALL  IOSET ; ASSIGN NEW DEVICES
```

LO

Output character to list device

Description

The LO call takes a single character and sends it to the current list device for output. From PL/M, a byte variable is used for the character. From an assembly language program, the character must be in the C register. The LO system call is not used by the operating system to write to the :LP: device.

Parameters

One parameter is required by LO. It can be passed in a PL/M procedure or through the C register from an assembly language program.

<char> Input parameter. Byte. The character to be output.

Error Numbers

Fatal: None.

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
LO:
  PROCEDURE (<char>) EXTERNAL;
    DECLARE (<char>) BYTE;
  END LO;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE CHAR BYTE;
```

A value is then assigned to the declared variable which is used as an actual parameter. Finally, the procedure is called passing the actual parameter.

```
CALL LO(CHAR);
```

Assembly Language Calling Sequence

From an assembly language program, the label LO is defined as an external instead of the label ISIS, because LO is a primitive ISIS routine. The character to output must be in the C register.

```
; LO
    EXTRN LO    ; ENTRY POINT FOR LO ISIS
                ; SAVE REGISTERS ...
    MOV   C,M   ; LOAD CHARACTER TO BE OUTPUT INTO "C"
                ; REGISTER
    CALL  LO    ; OUTPUT THE CHARACTER
```

LOAD

Load an executable program
and transfer control

Description

The LOAD call allows a program to load an absolute object module. After the file is loaded, control is passed to the loaded program, or the calling program. The program supplies the name of the absolute object module to load and also supplies a switch that indicates where to transfer control after loading. The LOAD call returns an error code.

Parameters

Five parameters are required by LOAD in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<path$ptr> Input parameter. Two byte. Pointer to the ASCII string (15-byte maximum) containing the name of the file to be loaded. The ASCII string can contain leading spaces but no embedded spaces. It must be terminated by a character other than a letter, digit, colon (:), or period (.). A space can be used.

<load$offset> Input parameter. Two byte. Value to be added to the load address causing the program to be loaded at an adjusted address. This does not mean that the program is relocatable. Usually, the code cannot be executed at the offset address.

<control$sw> Input parameter. Two byte. Value indicating where to transfer control after the load.

0 - Return control to the calling program.
1 - Transfer control to the loaded program.

If the program is not a main program, its entry point is zero, which causes control to vector through address 0.

<entry$ptr> Input parameter. Two byte. Pointer to an area in memory reserved for the address of the loaded program's entry point. This value is used when <control$sw> is 0. A zero is returned if the program is not a main program.

<status$ptr> Output parameter. Two byte. Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 1, 15, 16, 24, 30, 33

Non-fatal: 3, 4, 5, 12, 13, 22, 23, 28, 34

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
LOAD:
  PROCEDURE
      (<path$ptr>,<load$offset>,<control$sw>,<entry$ptr>,
      <status$ptr>) EXTERNAL;
    DECLARE
      (<path$ptr>,<load$offset>,<control$sw>,<entry$ptr>,
      <status$ptr>) ADDRESS;
END LOAD;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE FILENAME(15) BYTE;
DECLARE ENTRY ADDRESS;
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL LOAD(.FILENAME,0,1,.ENTRY,.STATUS);

IF STATUS < > 0 THEN . . .
```

Notice that a variable was declared for the filename, the entry point, and the status, and then the dot operator was used to pass the addresses of these values as required by the system call.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled LBLK) is stored in the DE register pair before the CALL instruction.

```
;LOAD
        EXTRN ISIS              ; LINK TO ISIS ENTRY POINT
LOAD    EQU   6                 ; SYSTEM CALL IDENTIFIER
                                ; SAVE REGISTERS . . .
        MVI   C,LOAD            ; LOAD IDENTIFIER
        LXI   D,LBLK            ; ADDRESS OF PARAMETER BLOCK
        CALL  ISIS
        LDA   LSTAT             ; TEST ERROR STATUS
        ORA   A
        JNZ   EXCEPT            ; BRANCH TO EXCEPTION ROUTINE
; ...
LBLK:                           ; PARAMETER BLOCK FOR LOAD
        DW    LFILE             ; POINTER TO FILENAME
BIAS:   DW    0                 ; OFFSET ADDRESS
RETSW:  DW    0                 ; RETURN SWITCH
        DW    ENTRY             ; POINTER TO ENTRY
        DW    LSTAT             ; POINTER TO STATUS
;
ENTRY:  DS    2                 ; ENTRY POINT (RETURNED)
LSTAT:  DS    2                 ; STATUS (RETURNED)
LFILE:  DB    'FILE.EXT '       ; SAMPLE FILE NAME
;
```

MEMCK

Return highest RAM
address in user memory

Description

The MEMCK call returns the highest memory address of contiguous memory available to the user. This address is the highest address available below the ISIS resident area 2. On the iPDS system, MEMCK should always return a value of 0F6C0H unless the DEBUG command is running. Then, it returns a value of 0ECC0H.

Parameters

None.

Error Numbers

Fatal: None.

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
MEMCK:
   PROCEDURE ADDRESS EXTERNAL;
   END MEMCK;
```

Notice that this is a typed procedure or function and can be used in an expression or parameter list.

There are no actual parameters to declare, so the function is called without any further declarations.

```
TOP$MEMORY = MEMCK...
```

Assembly Language Calling Sequence

From an assembly language program, the label MEMCK is defined as an external instead of the label ISIS, because MEMCK is a primitive ISIS routine. The address is returned in the HL register pair.

```
; MEMCK
;
          EXTRN   MEMCK ; ENTRY POINT FOR MEMCK IN ISIS
                        ; SAVE REGISTERS...
          CALL    MEMCK ; GET ADDRESS IN HL REGISTER
```

OPEN

Initialize file for I/O operation

Description

The OPEN call initializes the Active File Table and allocates buffers that are required for the I/O processing of the specified file. No input from or output to a file may occur until it has been OPENed. To open a file for line-edited access, use the <echo> parameter. The user supplies the pathname and the mode of access desired for the file after it is opened. The OPEN call returns the connection number and the error code.

Parameters

Five parameters are required by OPEN in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<conn$ptr> Output parameter. Two byte. Pointer to a value which contains the two-byte connection number (AFTN) for the file being opened. All other accesses to the file refer to this connection number. The console output and console input are permanently assigned connection numbers of 0 and 1 respectively. In addition, a program can have six files open at one time.

<path$ptr> Input parameter. Two byte. Pointer to the ASCII string (15 byte maximum) containing the name of the file to be opened. The ASCII string can contain leading spaces but no embedded spaces. It must be terminated by a character other than a letter, digit, colon (:), or period (.). A space can be used.

<access> Input parameter. Two byte. Value indicating the access mode for the file being opened.

A value of 1 specifies that the file is open only for input: READ. MARKER is set to 0 and LENGTH is unchanged. If the file doesn't exist, a non-fatal error occurs.

A value of 2 specifies that the file is open only for output: WRITE. MARKER and LENGTH are set to 0. If the file doesn't exist, a disk file is created with the filename specified at location <path$ptr>, and all attributes of the file are reset: 0. If the file already exists, it will be written over. All information will be lost. If the existing file has the format or write-protect attribute, a non-fatal error occurs.

A value of 3 specifies that the file is open for update: READ and WRITE. MARKER is set to 0. LENGTH is unchanged for existing files and is set to 0 for new files. If the file does not exist, a new file is created with the filename at location <path$ptr>, and all attributes are reset: 0.

Opening a file for an access mode that is not valid causes a non-fatal error. For example, opening a line printer for input.

<echo> Input parameter. Two byte. If the file being opened is for line editing, <echo> is the two-byte connection number (AFTN) of the echo file. If the file being opened is not for line editing, the LSB of <echo> contains 0. If the console output is the echo file, the MSB of <echo> must be non-zero, since the permanently assigned connection number for console output is also 0. For example, use an AFTN of 0FF00H for :CO:. The echo file must be previously opened for output (access value of 2).

<status$ptr> Output parameter. Two byte. Address in memory of the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 1, 7, 24, 30, 33

Non-fatal: 3, 4, 5, 9, 12, 13, 14, 22, 23, 25, 28, 63

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
    OPEN:
      PROCEDURE (<conn$ptr>,<path$ptr>,<access>,<echo>,
<status$ptr>)
          EXTERNAL;
        DECLARE (<conn$ptr>,<path$ptr>,<access>,<echo>,
<status$ptr>)
          ADDRESS;
      END OPEN;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE AFT$IN ADDRESS;
DECLARE FILENAME (15) BYTE DATA (':F1:MYPROG.SRC ');
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL OPEN(.AFT$IN,.FILENAME,1,0,.STATUS);
IF STATUS <> 0 THEN . . .
```

Notice that a variable was declared for the connection number, the filename, and the status, and then the dot operator was used to pass the addresses of these values as required by the system call. In this example, the file is opened in Read Mode as specified by the value 1 following the filename.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled OBLK) is stored in the DE register pair before the CALL instruction.

```
;OPEN
        EXTRN ISIS              ; LINK TO ISIS ENTRY POINT
OPEN    EQU   0                 ; SYSTEM CALL IDENTIFIER
                                ; SAVE REGISTERS . . .
        MVI   C,OPEN            ; LOAD IDENTIFIER
        LXI   D,OBLK            ; ADDRESS OF PARAMETER
                                BLOCK
        CALL  ISIS
        LDA   OSTAT             ; TEST ERROR STATUS
        ORA   A
        JNZ   EXCEPT            ; BRANCH TO EXCEPTION
                                ROUTINE
;...
OBLK:                           ; PARAMETER BLOCK FOR
                                OPEN
        DW    OAFT              ; POINTER TO AFT
        DW    OFILE             ; POINTER TO FILENAME
ACCESS: DW    I                 ; ACCESS, READ = I, WRITE =
                                2,
                                ; UPDATE = 3
ECHO:   DW    0                 ; IF ECHO <> 0,
                                ; ECHO = AFTN OF
                                ; ECHO OUTPUT FILE
        DW    OSTAT             ; POINTER TO STATUS
OAFT:   DS    2                 ; AFTN (RETURNED)
OSTAT:  DS    2                 ; STATUS (RETURNED)
OFILE:  DB    ':FO:FILE.EXT'    ; FILE TO BE OPENED
```

PO

Output character
to punch device

Description

The PO call takes a single character and sends it to the currently assigned punch device for output. The term punch was used to be consistent with earlier versions of ISIS that supported a paper tape punch device. From PL/M, a byte variable is used for the character. From an assembly language program, the character must be in the C register. The PO system call is not used by the operating system to write to :CO: or to :SO: devices.

Parameters

One parameter is required by PO. It can be passed in a PL/M procedure or through the C register from an assembly language program.

<char> Input parameter. Byte. The character to be output.

Error Numbers

Fatal: None.

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
PO:
  PROCEDURE (<char>) EXTERNAL;
    DECLARE (<char>) BYTE;
  END PO;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE CHAR BYTE;
```

A value is then assigned to the declared variable which is used as an actual parameter. Finally, the procedure is called passing the actual parameter.

```
CALL PO(CHAR);
```

Assembly Language Calling Sequence

From an assembly language program, the label PO is defined as an external instead of the label ISIS, because PO is a primitive ISIS routine. The character to output must be in the C register.

```
;PO
        EXTRN PO    ; ENTRY POINT FOR PO ISIS
                    ; SAVE REGISTERS
        MOV   C,M   ; LOAD CHARACTER TO BE OUTPUT INTO "C"
        CALL  PO    ; OUTPUT THE CHARACTER
```

READ

Transfer data from file to memory

Description

The READ call transfers data from an open file to a memory location specified by the calling program. See the section on "Line Edited Files" for further information. The user supplies the connection number of the file and the number of bytes to be read. The READ call returns the count of the bytes actually read, a pointer to the buffer containing the bytes read, and the error status code.

Parameters

Five parameters are required by READ in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<conn> — Input parameter. Two byte. Connection number (AFTN) returned for the file when it was opened. The connection number is always 1 for the console input.

<buf$ptr> — Input parameter. Two byte. Pointer to the buffer in user memory space that will receive data from the open file. The buffer must be at least as long as the <count> described below. If the buffer is too short, memory locations following the buffer are overwritten.

<count> — Input parameter. Two byte. Number of bytes to be transferred from the file to the buffer.

<actual$ptr> — Input parameter. Two byte. Pointer to location in the user memory space reserved for the two byte value of the number of bytes successfully read. The actual number of bytes read is determined by READ and added to the value of MARKER (the file pointer) as well as being returned to the user. The actual number of bytes transferred is never greater than <count>.

For line edited files, the actual number of bytes is never more than the number of bytes in the line edited buffer. When a file is not line-edited, the number of bytes is equal either to COUNT or MARKER whichever is less.

<status$ptr> Output parameter. Two byte. Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 24, 30, 33

Non-fatal: 2, 8

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
    READ:
       PROCEDURE (<conn>,<buf$ptr>,<count>,<actual$ptr>,
<status$ptr>)
            EXTERNAL;
          DECLARE (<conn>,<buf$ptr>,<count>,<actual$ptr>,
<status$ptr>)
            ADDRESS;

       END READ;
```

The actual parameters must be declared in the program prior to making the call. An example of declared actual parameter follows:

```
DECLARE AFT$IN ADDRESS;
DECLARE BUFFER(256) BYTE;
DECLARE ACTUAL ADDRESS;
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL READ(AFT$IN,.BUFFER,256,.ACTUAL,.STATUS);

IF ACTUAL = 0 THEN . . .
```

Notice that a variable was declared for the buffer, the actual number of bytes read, and the status, and then the dot operator was used to pass the addresses of these values as required by the system call.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled RBLK) is stored in the DE register pair before the CALL instruction.

```
; READ
;
         EXTRN ISIS     ; LINK TO ISIS ENTRY POINT
READ     EQU   3        ; SYSTEM CALL IDENTIFIER
                        ; SAVE REGISTERS . . .
         MVI   C,READ   ; LOAD IDENTIFIER
         LXI   D,RBLK   ; ADDRESS OF PARAMETER BLOCK
         CALL  ISIS
         LDA   RSTAT    ; TEST ERROR STATUS
         ORA   A
         JNZ   EXCEPT   ; BRANCH TO EXCEPTION ROUTINE
; ...
RBLK:                   ; PARAMETER BLOCK FOR READ
RAFT:    DS    2        ; FILE AFTN
         DW    RBUF     ; ADDRESS OF INPUT BUFFER
RCNT:    DW    256      ; LENGTH OF READ REQUESTED
         DW    ACTUAL   ; POINTER TO ACTUAL
         DW    RSTAT    ; POINTER TO STATUS
ACTUAL:  DS    2        ; COUNT OF BYTES READ (RETURNED)
RSTAT:   DS    2        ; STATUS (RETURNED)
RBUF:    DS    256      ; INPUT BUFFER
```

RENAME

Change disk filename

Description

The RENAME call allows a program to change the name of a disk file. The program supplies a pointer to the old name and the new name. The RENAME call returns an error code.

Parameters

Three parameters are required by RENAME in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<oldpath$ptr> Input parameter. Two byte. Pointer to the old pathname as an ASCII string (15 byte maximum) in memory. The string can contain leading spaces, but no embedded spaces. It must be terminated by a character other than a letter, a digit, a colon (:), or a period (.). A space (ASCII code 20H) can be used.

<newpath$ptr> Input parameter. Two byte. Pointer to the new pathname as an ASCII string (15 byte maximum) in memory. The device name must be the same as the device name in the old pathname. The string can contain leading spaces, but no embedded spaces. It must be terminated by a character other than a letter, a digit, a colon (:), or a period (.). A space (ASCII code 20H) can be used.

<status$ptr> Output parameter. Two byte. Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 1, 24, 30, 33

Non-fatal: 4, 5, 10, 11, 13, 17, 23, 28

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
RENAME:
  PROCEDURE (<oldpath$ptr>,<newpath$ptr>,<status$ptr>)
  EXTERNAL;
    DECLARE (<oldpath$ptr>,<newpath$ptr>,<status$ptr>)
    ADDRESS;
  END RENAME;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE OFILE(16) BYTE;
DECLARE NFILE(16) BYTE;
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL RENAME(.OFILE,.NFILE,.STATUS);
IF STATUS <> 0 THEN ...
```

Notice that variables were declared for the filenames and the status, and then the dot operator was used to pass the addresses of these values as required by the system call.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (RBLK) is stored in the DE register pair before the CALL instruction.

```
;
RENAME
;
        EXTRN ISIS              ; LINK TO ISIS ENTRY POINT
RENAME  EQU   7                 ; SYSTEM CALL IDENTIFIER
                                ; SAVE REGISTERS . . .
        MVI   C,RENAME          ; LOAD IDENTIFIER
        LXI   D,RBLK            ; ADDRESS OF PARAMETER BLOCK
        CALL  ISIS
        LDA   RSTAT             ; TEST ERROR STATUS
        ORA   A
        JNZ   EXCEPT            ; BRANCH TO EXCEPTION ROUTINE
; ...
RBLK:                           ; PARAMETER BLOCK FOR RENAME
        DW    FILE2             ; POINTER TO OLD FILENAME
        DW    FILEI             ; POINTER TO NEW FILENAME
        DW    RSTAT             ; POINTER TO STATUS
;
RSTAT:  DS    2                 ; STATUS (RETURNED)
FILEI:  DB    'FILE.NEW '       ; NEW NAME OF FILE
FILE2:  DB    'FILE.OLD '       ; OLD NAME OF FILE
```

RESCAN

Position marker to beginning of line

Description

The RESCAN call is used on line-edited files only. It allows your program to move the pointer to the beginning of a logical line that has already been read. Thus, the next READ call starts at the beginning of the last logical line read. The line is not re-echoed because it is echoed only when it is input from the console. The READ does not input the line from the file but only from the line editing buffer in memory. The user supplies the connection number of the file and the RESCAN call returns an error code.

Parameters

Two parameters are required by RESCAN in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<conn> Input parameter. Two byte. Connection number (AFTN) returned for a random access file when it was opened.

<status$ptr> Output parameter. Two byte. Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 33

Non-fatal: 2, 21

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
RESCAN:
   PROCEDURE (<conn>,<status$ptr>) EXTERNAL;
      DECLARE (<conn>,<status$ptr>) ADDRESS;
   END RESCAN;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE AFT$IN ADDRESS;
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL RESCAN(AFT$IN,.STATUS);
IF STATUS <> 0 THEN...
```

Notice that a variable was declared for the status, and then the dot operator was used to pass the address of this value as required by the system call.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled RBLK) is stored in the DE register pair before the CALL instruction.

```
;
RESCAN
;
        EXTRN ISIS          ; LINK TO ISIS ENTRY POINT
RESCAN  EQU   11            ; SYSTEM CALL IDENTIFIER
                            ; SAVE REGISTERS...
        MVI   C,RESCAN      ; LOAD IDENTIFIER
        LXI   D,RBLK        ; ADDRESS OF PARAMETER BLOCK
        CALL  ISIS
        LDA   RSTAT         ; TEST ERROR STATUS
        ORA   A
        JNZ   EXCEPT        ; BRANCH TO EXCEPTION ROUTINE
;...
RBLK:                       ; PARAMETER BLOCK FOR RESCAN
RAFT:   DS    2             ; FILE IDENTIFIER
        DW    RSTAT         ; POINTER TO STATUS
;
RSTAT:  DS    2             ; STATUS (RETURNED)
```

RI

Input character from
serial device

Description

The RI call reads a character entered at the currently assigned reader device and returns it as a byte variable in PL/M or in the A register in an assembly language program. The term reader is used to be consistent with previous versions of ISIS that supported a paper tape reader. The RI system call is not used by the operating system to read from the :CI: or :SI: devices.

Parameters

None.

Error Numbers

Fatal: None.

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
RI:
   PROCEDURE BYTE EXTERNAL;
   END RI;
```

Notice that this is a typed procedure or function and can be used in an expression or parameter list. There are no actual parameters to declare, so the function is called without any further declarations. Assuming an array has been set up to receive a sequence of characters, the following program receives those characters into the array.

```
DO WHILE BUFFER(INDEX)<> CR;
     INDEX = INDEX + 1
     BUFFER(INDEX) = RI;
     END;
```

Assembly Language Calling Sequence

From an assembly language program, the label RI is defined as an external instead of the label ISIS, because RI is a primitive ISIS routine. The character is returned in the A register.

```
; RI
     EXTRN RI  ; ENTRY POINT FOR CI IN ISIS
               ; SAVE REGISTERS...
     CALL  RI  ; GET CHARACTER
               ; CHARACTER RETURNED IN "A"
               ; REGISTER
```

Positions disk file marker

Description

The SEEK call allows a program to determine or to change the value of MARKER (the file pointer). SEEK can only be used with files opened for read or update. See the OPEN call. The MARKER can be changed in four ways: moved forward, moved backward, moved to a specific location, or moved to the end of the file. A non-fatal error is issued if a SEEK is made on a write only file. The user supplies the connection number for the file, the seek mode, and a value that specifies how far to move the MARKER. The SEEK call returns a status code.

Parameters

Five parameters are required by SEEK in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<conn> Input parameter. Two byte. Connection number (AFTN) returned for a random access file when it was opened.

<mode> Input parameter. Two byte. Value from 0 to 4 that indicates what action to take. The block and byte parameters represent the current MARKER position or calculate the desired offset, depending on the value of mode. A detailed discussion follows.

<block$ptr> Input parameter. (Output parameter in mode 0.) Two byte. Pointer to the two-byte value used for the block number. A block is 256 bytes, the same as a disk sector. However, the SEEK system call treats a block as 128 bytes to maintain compatibility with previous versions of the ISIS operating system. Depending on the mode, the block number is one of three values: the address of the block to which MARKER is to be moved, the number of blocks forward or backward to move the MARKER, or the current block address of MARKER.

<byte$ptr> Input parameter. (Output parameter in mode 0.) Two byte. Pointer to a two-byte value used for the byte number. Depending on the mode, the byte number is one of three values: the address of the byte to which MARKER is to be moved, the number of bytes forward or backward to move the MARKER, or the current byte address of MARKER.

<status$ptr> Output parameter. Two byte. Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Return Marker Location: Mode = 0

Under this mode, the system returns a pair of block and byte values (at <block$-ptr> and <byte$ptr> that signify the current position of the MARKER. For example, if the MARKER were just beyond the first block of the file, the system would return the number 1 and 0 in the addresses assigned to block and byte respectively. Block values are counted from 0. Byte values are counted from 0 to 128. The value of MARKER is given by the following formula:

128 * (block number) + byte number

Move.Marker Backward: Mode = 1

If the mode value is 1, the marker is moved back toward the beginning of the file. The block and byte parameters determine how many bytes the marker is moved. For example, if block is equal to 0 and byte is equal to 382, the marker is moved backward 382 bytes. To define an offset of N, use block and byte values according to the following formula:

128 * (block number) + byte number

If N is greater than MARKER, the prescribed action would place the MARKER before the beginning of the file. In this case, MARKER is set to 0 and a non-fatal error occurs.

Move Marker to Specific Location: Mode = 2

In this mode, the marker is moved to a specific position in the file defined by the block and byte parameters. For example, if block is equal to 27 and byte is equal to 63, MARKER will be moved so that the 64th byte of block 27 will be the next one read or written. Similarly, if both block and byte are set to 0, the marker is moved to the beginning of the file. If the file is open for update and the prescribed action would place the marker beyond the end of the file, the new position of the marker becomes the end of the file. Thus, LENGTH becomes equal to MARKER. Note that the system does not guarantee that initialized data areas are skipped over.

Move Marker Forward: Mode = 3

In this mode, the marker is moved ahead toward the end of the file. The block and byte parameters define the offset N according to the following formula:

128 * (block number) + byte number

If the file is open for update and the prescribed action would place the marker beyond the end of the file, the new position of the marker becomes the end of the file. Thus, LENGTH becomes equal to MARKER. Note that the system does not guarantee that initialized data areas are skipped over.

If the extension of a file by SEEK causes an overflow on the disk, a fatal error is reported, either during the SEEK or when the program tries to WRITE to the extended area of the file.

If an attempt is made to extend a file that is open for READ only, the marker is set to the former end-of-file and a non-fatal error occurs.

Move Marker to the End of the File: Mode = 4

If the mode value is 4, the marker is moved to the end of the file. Block and byte parameters are ignored.

NOTE

For a file opened for update, moving the MARKER can allocate more space than LENGTH requires, i.e., than is subsequently written with data. A DIR will shown the allocated locations as being used. However, data can still be written to these locations.

Error Numbers

Fatal: 7, 24, 30, 33

Non-fatal: 2, 19, 20, 27, 31, 35

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
SEEK:
   PROCEDURE (<conn>,<mode>,<block$ptr>,<byte$ptr>,
<status$ptr>)
         EXTERNAL;
   DECLARE (<conn>,<mode>,<block$ptr>,<byte$ptr>,
<status$ptr>)
         ADDRESS;
   END SEEK;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE AFT$IN ADDRESS;
DECLARE BLOCKNO ADDRESS;
DECLARE BYTENO ADDRESS;
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL SEEK(AFT$IN,0,.BLOCKNO,.BYTENO,.STATUS);
IF STATUS <> 0 THEN . . .
```

Notice that a variable was declared for the block number, the byte number, and the status, and then the dot operator was used to pass the addresses of these values as required by the system call.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled SBLK) is stored in the DE register pair before the CALL instruction.

```
; SEEK
;
        EXTRN ISIS      ; LINK TO ISIS ENTRY POINT
SEEK    EQU   5         ; SYSTEM CALL IDENTIFIER
                        ; SAVE REGISTERS . . .
        MVI   C,SEEK    ; LOAD IDENTIFIER
        LXI   D,SBLK    ; ADDRESS OF PARAMETER BLOCK
        CALL  ISIS
        LDA   SSTAT     ; TEST ERROR STATUS
        ORA   A
        JNZ   EXCEPT    ; BRANCH TO EXCEPTION ROUTINE
; ...
SBLK:                   ; PARAMETER BLOCK FOR SEEK
SAFT:   DS    2         ; FILE IDENTIFIER
MODE:   DS    2         ; TYPE OF SEEK
        DW    BLKS      ; POINTER TO BLKS
        DW    NBYTE     ; POINTER TO NBYTE
        DW    SSTAT     ; POINTER TO STATUS
;
BLKS:   DS    2         ; NUMBER OF SECTORS TO SKIP
NBYTE:  DS    2         ; NUMBER OF BYTES TO SKIP
SSTAT:  DS    2         ; STATUS (RETURNED)
```

SPATH

Obtain file information

Description

The SPATH call allows a program to obtain information relating to a specified file. The program supplies a pointer to the name of the file and the address of a 12-byte location in which the system will return the information. In PL/M, the 12-bytes can be handled as an array or a record. The information returned by this call includes the device number, the file name and extension, the device type, and the disk drive type.

Parameters

Three parameters are required by SPATH in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<path$ptr> Input parameter. Two byte. Pointer to the ASCII string (15-byte maximum) containing the name of the file. The ASCII string can contain leading spaces but no embedded spaces. It must be terminated by a character other than a letter, digit, colon (:), or period (.). A space can be used.

<info$ptr> Output parameter. Two byte. Pointer to a 12-byte location in memory reserved for the return information. After the call is complete, the memory buffer will contain the following information.

Byte 0	Logical Device Number
Byte 1-6	Filename
Byte 7-9	Extension
Byte 10	Physical Device Type
Byte 11	Drive Type

More information on these values follows the error numbers.

<status$ptr> Output parameter. Two byte. Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 33

Non-fatal: 4, 5, 23, 28

Information Block

The possible values for byte 0, the device number are:

0 = logical disk device 0
1 = logical disk device 1
2 = logical disk device 2
3 = logical disk device 3
4 = logical disk device 4
5 = logical disk device 5
6 = serial input device
7 = serial output device
8 = CRT input device (keyboard)
9 = CRT output device (display screen)
10 = User defined console input device
11 = User defined console output device
12 = Teletype paper tape reader
13 = High speed paper tape reader
14 = User defined reader device 1
15 = User defined reader device 2
16 = Teletype paper tape punch
17 = High speed paper tape punch
18 = User defined punch device 1
19 = User defined punch device 2
20 = Line printer
21 = User defined printer device 1
22 = Byte bucket (a pseudo output device)
23 = Console input device
24 = Console output device

Bytes 1-6 are the ASCII characters for the ISIS filename.

Bytes 7-9 are the ASCII characters for the ISIS file extension.

There is no period character separating the filename from the extension.

Byte 10 is the device type and defines the type of peripheral with which the file is associated. The values are:

0 = Sequential Input
1 = Sequential Output
2 = Sequential I/O
3 = Random I/O

Byte 11 is the drive type. This byte only has meaning if the device is a disk, that is the device type is 3. Then, the values for the drive type are:

0 = Bubble memory not present
1 = Double density diskette
2 = Bubble device

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
SPATH:
  PROCEDURE (<path$ptr>,<info$ptr>,<status$ptr>)
    EXTERNAL;
  DECLARE (<path$ptr>,<info$ptr>,<status$ptr>) ADDRESS;
  END SPATH;
```

The actual parameters must be declared in the program prior to making the call. In this example of declaring actual parameters, the 12-bytes of information will be in a record.

```
DECLARE FILENAME (15) BYTE;
DECLARE FILINF STRUCTURE (DEVICE$NO BYTE,
                          FILENAME (6) BYTE,
                          FILE$EXT (3) BYTE,
                          DEVICE$TYPE BYTE,
                          DRIVE$TYPE BYTE);
DECLARE STATUS ADDRESS;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL SPATH(.FILENAME,.FILINF,.STATUS);
IF STATUS <> 0 THEN ...
```

Notice that a variable was declared for the filename, the file information, and the status, and then the dot operator was used to pass the addresses of these values as required by the system call.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled SBLK) is stored in the DE register pair before the CALL instruction.

```
; SPATH
;
         EXTRN ISIS          ; LINK TO ISIS ENTRY POINT
SPATH    EQU   14            ; SYSTEM CALL IDENTIFIER
                             ; SAVE REGISTERS ...
         MVI   C, SPATH      ; LOAD IDENTIFIER
         LXI   D,SBLK        ; ADDRESS OF PARAMETER BLOCK
         CALL  ISIS
         LDA   SSTAT         ; TEST ERROR STATUS
         ORA   A
         JNZ   EXCEPT        ; BRANCH TO EXCEPTION ROUTINE
;
SLBK:                        ; PARAMETER BLOCK FOR SPATH
         DW    FILEN         ; POINTER TO FILE NAME
         DW    BARRAY        ; POINTER TO BYTE ARRAY
         DW    SSTAT         ; POINTER TO STATUS
FILEN:   DS    15            ; FILE NAME FIELD
BARRAY:  DS    12            ; ARRAY FOR DATA
SSTAT:   DS    2             ; STATUS (RETURNED)
```

WHOCON

Determine file assigned
as system console

Description

The WHOCON call allows a program to determine what file is assigned as the current system console device. The program requests the information, and the system call returns the filename.

Parameters

Three parameters are required by WHOCON in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<conn> — Input parameter. Two byte. Connection number (AFTN) to indicate whether the console input or console output file is desired. Both have permanently assigned AFTNs. A value of 0 indicates console output; a value of 1 indicates console input.

<buf$ptr> — Input parameter. Two byte. Pointer to a 15-byte buffer reserved for the return information, the pathname of the file currently assigned as the console input or console output. The name is returned as an ASCII string terminated by a space.

<status$ptr> — Output parameter. (for assembly language only) Two byte. Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 33

Non-fatal: None.

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
WHOCON:
  PROCEDURE (<conn>,<buf$ptr>) EXTERNAL;
      DECLARE (<conn>,<buf$ptr>) ADDRESS;
  END WHOCON;
```

Notice that the error status is not returned to a PL/M program.

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameters follows:

```
DECLARE BUFF$IN(15) BYTE;
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters.

```
CALL WHOCON(1,.BUFF$IN);
```

Notice that a variable was declared for the buffer, and then the dot operator was used to pass the addresses of this value as required by the system call.

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The parameters are set up in the correct order with a series of DW, DS, and DB directives. Then, the address of the beginning of the block (labelled WBLK) is stored in the DE register pair before the CALL instruction.

```
; WHOCON
;
          EXTRN ISIS          ; LINK TO ISIS ENTRY POINT
WHOCON    EQU   13            ; SYSTEM CALL IDENTIFIER
                              ; SAVE REGISTERS . . .
          MVI   C,WHOCON      ; LOAD IDENTIFIER
          LXI   D,WBLK        ; ADDRESS OF PARAMETER
                                BLOCK
          CALL  ISIS
; ...
WBLK:
WID:      DS    2             ; I/O IDENTIFIER
          DW    WBUF          ; POINTER TO WORK BUFFER
                              ; (FILLED BY WHOCON)
          DW    WSTAT         ; POINTER TO STATUS
;
WSTAT:    DS    2             ; STATUS (RETURNED)
WBUF:     DS    15
```

WRITE

Transfer data from memory to a file

Description

The WRITE system call transfers data from a specified location in memory to an open file. The user supplies the connection number of the file being written to, a pointer to the buffer containing the data to be written, and a count of the number of bytes to be written. The WRITE call returns an error code.

Parameters

Four parameters are required by WRITE in the following order. These can be passed in a PL/M procedure call or stored in a table in memory pointed to by the DE register pair for an assembly language program.

<conn>	Input parameter. Two byte. Connection number (AFTN) returned for the file when it was opened. The connection number is always 0 for the console output.
<buf$ptr>	Input parameter. Two byte. Pointer to the buffer in user memory space that contains the data to be written.
<count>	Input parameter. Two byte. Number of bytes to be transferred from the file to the buffer. The value of <count> is added to MARKER. If this results in MARKER being greater than LENGTH, then LENGTH is set equal to MARKER, i.e., the file is extended. The number of bytes actually written is equal to <count>. If the buffer length is less than <count>, memory locations outside of the buffer are written to the file.
<status$ptr>	Output parameter. Two byte. Pointer to two bytes in memory reserved for the error number. If the error number is 0, no error occurred.

Error Numbers

Fatal: 7, 24, 30, 33

Non-fatal: 2, 6

PL/M Calling Sequence

The form of the declaration of the external procedure is:

```
WRITE:
  PROCEDURE (<conn>,<buf$ptr>,<count>,<status$ptr>) EXTERNAL;
      DECLARE (<conn>,<buf$ptr>,<count>,<status$ptr>) ADDRESS;
  END WRITE;
```

The actual parameters must be declared in the program prior to making the call. An example of declaring actual parameter follows:

```
DECLARE AFT$OUT ADDRESS;
DECLARE BUFFER(256) BYTE;
DECLARE STATUS ADDRESS;
DECLARE ERR$MESSAGE BYTE (*) DATA ('This is an error
message',0DH,0AH)
```

Values are then assigned to the declared variables which are used as actual parameters. Finally, the procedure is called passing all the actual parameters. Notice that the address specified as the actual parameter for buffer pointer is in the form:

```
.(<string literal>)
```

The period causes the use of the address of the location where the string literal, given inside of parentheses and single quotes, is stored.

```
CALL WRITE (0,.ERR$MESSAGE,LENGTH(ERR$MESSAGE),.STATUS);
IF STATUS <> 0 THEN . . .
```

Assembly Language Calling Sequence

An example that illustrates the calling sequence in assembly language is shown below. The address of the beginning of the parameter block (labelled WBLK) is stored in the DE register pair before the CALL instruction.

```
; WRITE
;
        EXTRN ISIS          ; LINK TO ISIS ENTRY POINT
WRITE   EQU   4             ; SYSTEM CALL IDENTIFIER
                            ; SAVE REGISTERS . . .
        MVI   C,WRITE       ; LOAD IDENTIFIER
        LXI   D,WBLK        ; ADDRESS OF PARAMETER BLOCK
        CALL  ISIS
        LDA   WSTAT         ; TEST ERROR STATUS
        ORA   A
        JNZ   EXCEPT        ; BRANCH TO EXCEPTION ROUTINE
; ...
WBLK:                       ; PARAMETER BLOCK FOR WRITE
WAFT:   DS    2             ; FILE AFTN
        DW    WBUF          ; ADDRESS OF OUTPUT BUFFER
WCNT:   DW    256
        DW    WSTAT         ; POINTER TO STATUS
WSTAT:  DS    2             ; STATUS (RETURNED)
WBUF:   DS    256           ; OUTPUT BUFFER
```

Example Programs Using System Calls

The first three example programs show how to use the system calls to add a custom I/O driver to the system. The first program is written in PL/M-80 and assumes that the I/O driver is for a list device and, thus, is for output only. The function code for a List device is 6; the entry point for the driver is 0E800H.

This program adds the I/O driver for the user-defined list device to the system, and then loads the driver program stored in the absolute object file file :F1:PRINT.DRV. This program must be rerun every time the system is reset. It can be placed into a configuration file (ABOOT.CSD or BBOOT.CSD) as described in Chapters 4 and 5, and then it will be automatically loaded each time the system is reset.

Example 1:

```
PRINT$DRIVER:DO;        /*Beginning of main module*/

      /*Declare external procedures IODEF and LOAD*/

IODEF:PROCEDURE (FCODE,ENTRY$POINT) EXTERNAL;
      DECLARE (FCODE) BYTE;
      DECLARE (ENTRY$POINT) ADDRESS;
END IODEF;

LOAD:PROCEDURE (FILENAME, LOAD$OFFSET, CONTROL,
                ENTRY, STATUS) EXTERNAL;
      DECLARE FILENAME ADDRESS;
      DECLARE LOAD$OFFSET ADDRESS;
      DECLARE CONTROL ADDRESS;
      DECLARE ENTRY ADDRESS;
      DECLARE STATUS ADDRESS;
END LOAD;

      /*Declare variables used in the program*/

DECLARE
      FCODE BYTE,
      ENTRY ADDRESS,
      FILENAME (*) ADDRESS INITIAL (':F1:PRINT.DRV'),
      OFFSET (*) ADDRESS INITIAL (0),
      RETURN$SWITCH (*) ADDRESS INITIAL (0),
      STATUS ADDRESS;

      /*Setup variables and call IODEF to add driver*/

FCODE = 6;
ENTRY = 0E800H;
CALL IODEF (FCODE,ENTRY);

      /*Load the driver from the file*/

CALL LOAD (.FILENAME,OFFSET,RETURN$SWITCH,.ENTRY,.STATUS);

END PRINT$DRIVER;
```

In the next example, an I/O driver for a serial device is added to the operating system. Since the device is both input and output, it is being added as User Reader 1 and User Punch 1. The function codes are 2 and 4, respectively. The entry point for the reader portion of the code is 0E000H and for the punch portion of the code is 0E800H.

This program adds I/O drivers for the User Reader device and the User Punch device to the system, and then loads the driver program stored in the absolute object file :F1:SERIAL.DRV. This program must be rerun every time the system is reset. It can be placed into a configuration file (ABOOT.CSD or BBOOT.CSD) as described in Chapters 4 and 5, and then it will be automatically loaded each time the system is reset.

Example 2:

```
SERIAL$DRIVER:DO;     /*Beginning of main module*/

     /*Declare external procedures IODEF and LOAD*/

IODEF:PROCEDURE (FCODE,ENTRY$POINT) EXTERNAL;
     DECLARE (FCODE) BYTE;
     DECLARE (ENTRY$POINT) ADDRESS;
END IODEF;

LOAD:PROCEDURE (FILENAME, LOAD$OFFSET, CONTROL,
                    ENTRY, STATUS) EXTERNAL;
     DECLARE FILENAME ADDRESS;
     DECLARE LOAD$OFFSET ADDRESS;
     DECLARE CONTROL ADDRESS;
     DECLARE ENTRY ADDRESS;
     DECLARE STATUS ADDRESS;
END LOAD;

     /*Declare variables used in the program*/

DECLARE
     FCODE BYTE,
     ENTRY ADDRESS,
     FILENAME (*) ADDRESS INITIAL (':F1:SERIAL.DRV'),
     OFFSET (*) ADDRESS INITIAL (0),
     RETURN$SWITCH (*) ADDRESS INITIAL (0),
     STATUS ADDRESS;

     /*Setup and call IODEF for the reader portion of the driver*/

ENTRY = 0E000H;
FCODE = 2;
CALL IODEF (FCODE,ENTRY);
     /*Setup and call IODEF for the punch portion of the driver*/
ENTRY = 0E800H;
FCODE = 4;
CALL IODEF (FCODE,ENTRY);

CALL LOAD (.FILENAME,OFFSET,RETURN$SWITCH,.ENTRY,.STATUS);

END SERIAL$DRIVER;
```

In the next example, an I/O driver for a console device is added to the operating system. Since the device is both input and output, it is being added as User Console Input and User Console Output. An additional console status routine must be added as well. The status routine should return a value of 00H if a character is not ready for input and a value of 0FFH if a character is ready for input. The value should be returned in the A register for assembly language and in a byte variable for PL/M.

The function codes are 0, 1, and 7, respectively. The entry point for the console input portion of the code is 0E000H, for the console output portion of the code is 0E800H, and for the console status portion of the code is 0D500H.

This program adds all three I/O routines to the system, and then loads the program with all the routines stored in the absolute object file :F1:CONSOL.DRV. This program must be rerun every time the system is reset. It can be placed into a configuration file (ABOOT.CSD or BBOOT.CSD) as described in Chapters 4 and 5, and then it will be automatically loaded each time the system is reset.

Example 3:

```
CONSOL$DRIVER:DO;   /*Beginning of main module*/

     /*Declare external procedures IODEF and LOAD*/

IODEF:PROCEDURE (FCODE,ENTRY$POINT) EXTERNAL;
     DECLARE (FCODE) BYTE;
     DECLARE (ENTRY$POINT) ADDRESS;
END IODEF;

LOAD:PROCEDURE (FILENAME, LOAD$OFFSET, CONTROL, ENTRY, STATUS) EXTERNAL;
     DECLARE FILENAME ADDRESS;
     DECLARE LOAD$OFFSET ADDRESS;
     DECLARE CONTROL ADDRESS;
     DECLARE ENTRY ADDRESS;
     DECLARE STATUS ADDRESS;
END LOAD;

     /*Declare variables used in the program*/

DECLARE
     FCODE BYTE,
     ENTRY ADDRESS,
     FILENAME (*) ADDRESS INITIAL (':F1:CONSOL.DRV'),
     OFFSET (*) ADDRESS INITIAL (0),
     RETURN$SWITCH (*) ADDRESS INITIAL (0),
     STATUS ADDRESS;

     /*Setup and call IODEF for the console input portion*/

ENTRY = 0E000H;
FCODE = 0;
CALL IODEF (FCODE,ENTRY);

     /*Setup and call IODEF for the console output portion*/

ENTRY = 0E800H;
FCODE = 1;
CALL IODEF (FCODE,ENTRY);

     /*Setup and call IODEF for the console status portion*/

ENTRY = 0D500H;
FCODE = 7;
CALL IODEF (FCODE,ENTRY);

CALL LOAD (.FILENAME,OFFSET,RETURN$SWITCH,.ENTRY,.STATUS);

END CONSOL$DRIVER;
```

Each of the three programs shown must be compiled using the PL/M-80 compiler, linked with SYSPDS.LIB and PLM80.LIB using the LINK command, and then located using the LOCATE command. The output of the LOCATE command should be a file named as specified in each example.

The next two examples illustrate two programs, one written in PL/M and one written in MCS-80/85 Assembly Language, with identical functions. Both programs allow a file to be typed into the :CO: device by specifying:

```
TYPE <filename>
```

rather than

```
COPY <filename> TO :CO:
```

The PL/M program called TYPE must be compiled, linked with SYSPDS.LIB and PLM80.LIB, and located to file "TYPE" before it can be executed. The assembly language program must be assembled, linked with SYSPDS.LIB, and located before it can be executed.

Example 4:

```
TYPE:DO;
     DECLARE BUFFER(128) BYTE;
     DECLARE ACTUAL$COUNT ADDRESS;
     DECLARE STATUS ADDRESS;
     DECLARE AFT$IN ADDRESS;
     DECLARE READ$ACCESS LITERALLY '1';

OPEN:PROCEDURE (AFT,FILE,ACCESS,MODE,STATUS) EXTERNAL;
     DECLARE (AFT,FILE,ACCESS,MODE,STATUS) ADDRESS;
END OPEN;

CLOSE:PROCEDURE (AFT,STATUS) EXTERNAL;
     DECLARE (AFT,STATUS) ADDRESS;
END CLOSE;

READ:PROCEDURE (AFT,BUFFER,COUNT,ACTUAL,STATUS) EXTERNAL;
     DECLARE (AFT,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
END READ;

WRITE:PROCEDURE (AFT,BUFFER,COUNT,STATUS) EXTERNAL;
     DECLARE (AFT,BUFFER,COUNT,STATUS) ADDRESS;
END WRITE;

EXIT:PROCEDURE EXTERNAL;
END EXIT;

ERROR:PROCEDURE (ERRNUM) EXTERNAL;
     DECLARE (ERRNUM) ADDRESS;
END ERROR;

     /*Read the console file to get the parameter string.
       For this example, the command entered is:

          TYPE ASM.LST

       At this point, the console input buffer contains

          A S M . L S T CR LF      */
```

```
CALL READ (1,.BUFFER,128,.ACTUAL$COUNT,.STATUS);
CALL OPEN (.AFT$IN,.BUFFER,READ$ACCESS,0,.STATUS);
IF STATUS > 0 THEN CALL ERROR (STATUS);

     /*The file ASM.LST is now open for input.*/

ACTUAL$COUNT = 1;
DO WHILE ACTUAL$COUNT < > 0;
     CALL READ (AFT$IN,.BUFFER,128,.ACTUAL$COUNT,.STATUS);
     IF STATUS > 0 THEN CALL ERROR (STATUS);
     CALL WRITE (0,.BUFFER,ACTUAL$COUNT,.STATUS);
     IF STATUS > 0 THEN CALL ERROR (STATUS);
END;

CALL WRITE (0,.('COPY COMPLETED',0DH,0AH),16,.STATUS);
CALL CLOSE (AFT$IN,.STATUS);
IF STATUS > 0 THEN CALL ERROR (STATUS);
CALL EXIT;

END;
```

The next example program also called TYPE is written in MCS-80/85 Assembly Language and is equivalent to the preceding PL/M-80 program.

Example 5:

```
;                                   Sample Program
;
OPEN     EQU   0
CLOSE    EQU   1
READ     EQU   3
WRITE    EQU   4
EXIT     EQU   9
ERROR    EQU   12
;
         EXTRN ISIS
;
         CSEG                       ;BEGINNING OF CODE SEGMENT
BEGIN:
         LXI   SP,STCKA + 4
         MVI   C,READ               ;READ THE CONSOLE
         LXI   D,RBLK
         CALL  ISIS
         LDA   STATUS
         ORA   A
         JNZ   ERR
;
         MVI   C,OPEN               ;OPEN THE INPUT FILE
         LXI   D,OBLK
         CALL  ISIS
         LDA   STATUS
         ORA   A
         JNZ   ERR
         LHLD  AFT
         SHLD  CAFT
;
LOOP:
         MVI   C,READ               ;READ THE INPUT FILE
         LXI   D,RBLK
```

```
                CALL    ISIS
                LDA     STATUS
                ORA     A
                JNZ     ERR
                LHLD    ACTUAL
                MOV     A,H
                ORA     L
                JZ      DONE
                MVI     C,WRITE         ;WRITE TO THE CONSOLE
                LXI     D,WBLK
                CALL    ISIS
                LDA     STATUS
                ORA     A
                JNZ     ERR
                JMP     LOOP
DONE:
                MVI     C,CLOSE         ;CLOSE THE INPUT FILE
                LXI     D,CBLK
                CALL    ISIS
                MVI     C,EXIT          ;NORMAL EXIT
                LXI     D,XBLK
                CALL    ISIS
;
ERR:
                MVI     C,ERROR         ;ERROR MESSAGE
                LXI     D,EBLK
                CALL    ISIS
                MVI     C,EXIT          ;ERROR EXIT
                LXI     D,XBLK
                CALL    ISIS
                DSEG                    ;BEGINNING OF DATA SEGMENT
;
;
OBLK:
;
                DW      AFT
                DW      BUFFER
                DW      1               ;READ ACCESS
                DW      0               ;NO ECHO
                DW      STATUS
;
;
CBLK:
CAFT:           DS      2
                DW      STATUS
;
;
RBLK:
;
AFT:            DW      1
                DW      BUFFER
                DW      128
                DW      ACTUAL
                DW      STATUS
;
;
WBLK:
                DW      0
                DW      BUFFER
```

```
ACTUAL: DS      2
        DW      STATUS
;
;
XBLK:
        DW      STATUS
;
;
EBLK:
STATUS: DS      2
        DW      STATUS
;
;
BUFFER: DS      128
;
;
STCKA:  DS      4
;
END     BEGIN
```

Another example of an assembler language program using the CI and CO primitive system calls is given in Chapter 7. Also, an application note (AP-80, ISIS-II System Calls) is available that describes two other Assembly Language programs using ISIS system calls. This application note is available from the Intel Literature Department, order number 121559.

System Architecture

The topic of system architecture is divided into four categories for discussion: memory organization and allocation, I/O address space, peripheral device input and output, and disk structure.

Memory Organization and Allocation

The organization of the development system's memory under the ISIS-PDS operating system is shown in the figure 8-4.

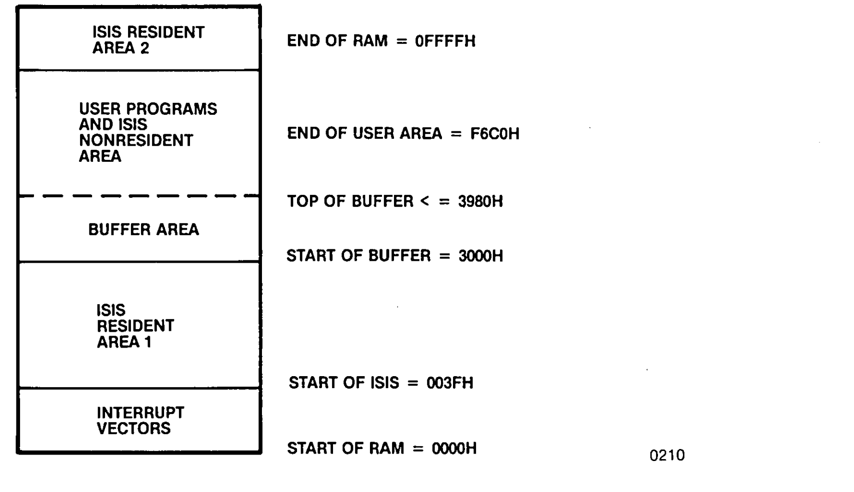


Figure 8-4 Memory Map

Interrupt Vectors

The operating system reserves restart interrupts 5.5, 6.5, and 7.5 for use by the keyboard, the flexible disk drive, and software reset, respectively. Additionally, interrupt 1 is reserved for use by the DEBUG command and software reset 7.5.

Interrupt 0 and interrupts 2 through 7 are available to the user. For example, custom I/O driver routines can use these interrupts.

The interrupt vectors are located at addresses as shown in figure 8-5. These are the only locations below 3000H which should be loaded with user code. Only three bytes per vector are available for user code starting at the address shown in figure 8-5. The operating system does not load the user interrupt vectors; the systems programmer must load the required vectors within the user program.

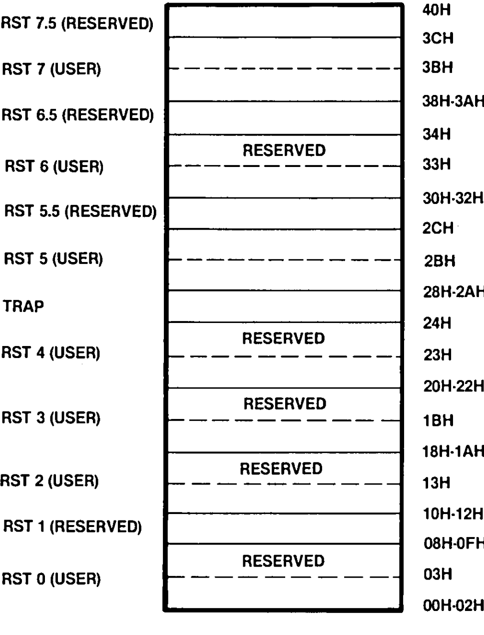


Figure 8-5 Interrupt Vectors

ISIS Resident Area 1

The ISIS resident area 1 is reserved for the kernel, the collection of ISIS routines that are always present in memory. The kernel is protected from a program load operation and cannot be overlaid. However, it is not protected from an executing user program and can accidentally be destroyed by writing to this area of memory.

Buffer Area and ISIS Resident Area 2

The buffer area and ISIS resident area 2 are used for I/O buffers of 256 bytes each.

Buffers are dynamically allocated and deallocated according to the I/O needs of user programs. These requirements come from explicit system calls in the source code or are generated from the source code by the language translator used.

There is one permanent I/O buffer used by ISIS for line edited input from the console. Line edited input is described in later sections of this chapter.

The minimum number of buffers allocated is 3 (including the permanent console buffer); the maximum is 19. Thus, the top of the buffer area varies dynamically. If a user program needs more than two buffers, space may be allocated from the buffer area at the expense of the user program area. The maximum top of the buffer area is 3980H, since only half of the buffer space is allocated from the buffer area.

User Programs and ISIS Nonresident Area

User programs and ISIS nonresident routines, such as the CLI, commands, utilities, and language translators, are loaded into the area above the buffer area and below the ISIS resident area 2.

The origin point of user programs is specified by commands to LOCATE or by the ORG statement in assembly language programs. It should be at least 3180H to allow for the minimum of three buffers. The higher a program's base address, the more buffers can be allocated up to a maximum of 19. In general, more buffer space means that more disk files can be opened simultaneously by the program.

The base address of a user program can either be calculated as described in the following paragraphs, or it can be assigned to 3980H allowing the maximum number of buffers.

To write a program that is independent of the type of device used for files and also independent of how it is called (from SUBMIT or interactively from the keyboard), allow the maximum number of buffers, 19. The program base address is then 3980H.

On the other hand, if memory space is needed by the program, the lowest possible base address can be calculated. To calculate the lowest possible base address for a given program, the number of buffers required by the program must first be established. Use the following rules as a guideline. File I/O is discussed later in this chapter.

1. Each open disk file requires two buffers until the file is closed.

2. An open line edited file (including the permanent console line edited file) requires one buffer until the file is closed. For a disk file used as line edited input, this buffer is in addition to the two buffers required by rule 1.

3. A system call that accesses a disk directory requires two buffers during the processing of the system call. The buffers are released on return to the calling program. The system calls that may access a directory are LOAD, DELETE, RENAME, ATTRIB, and CONSOL when it specifies a disk file.

4. When the CONSOL system call assigns the console input or output device to a disk file, three buffers are required for the console input file and two more are required for the console output file. These buffers are required until the console is re-assigned. If a program is to be called from a SUBMIT file, these file buffers must be taken into account when calculating the buffer area.

The buffer area must be large enough for the maximum number of buffers allocated simultaneously divided by two. Since only half the buffer space comes from the buffer area. The other half comes from ISIS resident area 2. The top of the user area can be obtained by the MEMCK system call.

A formula to determine the required size of the buffer area is:

N * 256/2 or N * 128

where N is the decimal value of the maximum number of buffers allocated simultaneously by the program as determined by the rules listed previously. The value 256/2 or 128 gives the number of bytes for each required buffer that is allocated from the Buffer Area of memory. Remember that there are 256 bytes/buffer, and half the buffer is allocated from the Buffer Area.)

If an attempt is made to LOCATE a program below 3180H (fewer than 3 buffers allocated), an error message is generated by the LOCATE utility.

To calculate the program base address, add the size of the buffer area as calculated above to the start of the buffer area in memory (3000H or 12288 in decimal). This can be done using the following formula:

12288 + (128 * N)

where N is the decimal value for the maximum number of buffers allocated simultaneously by the program as determined by the rules listed previously. Address 12288 (3000H) is the beginning of the buffer area. The value 128 is from the previous formula.

Examples of Calculating the User Program Base Address. A program that has no system calls, does not assign the console to a disk file, and is not called by a command in a disk file only requires the minimum of three buffers. Therefore, it can have a base address of 12672 or 3180H.

12288 + (128 * 3) = 12672

If the program is changed to open one disk file, it needs five buffers, and the base address must be 12928 or 3280H.

12288 + (128 * 5) = 12928

If the same program is called from a SUBMIT file and then defines the console output as a disk file, five more buffers will be needed, requiring an address of 13568 or 3500H:

12288 + (128 * 10) = 13568

Assume a program opens a line edited disk file (3 buffers) and an echo file on disk (2 buffers). Console input is open but is not assigned to a disk file, so the minimum number of three buffers are also required for console and system use. The program base address is 13312 or 3400H.

12288 + (128 * 8) = 13312

I/O Address Space

The iPDS CPU (an MCS-80/85) can address 256 (0 - 0FFH) I/O ports. These ports are assigned as follows:

Port Address		Port Function		
00H -	0FH	Reserved		
10H -	1FH	EMV/PROM Programmer Interface		
20H -	2FH	Reserved		
30H -	3FH	Reserved		
40H -	4FH	Multimodule Chip Select J1		
50H -	5FH	Multimodule Chip Select J2		
60H -	6FH	Multimodule Chip Select J3		
70H -	7FH	Multimodule Chip Select J4		
	80H	8253 Counter 0 (Baud Clock)		
	81H	8253 Counter 1 (Disk Motor-On Timer)		
	82H	8253 Counter 2 (Disk Index Timer)		
	83H	8253 Mode Select		
84H -	8FH	Reserved		
	90H	8251A Serial I/O Data		
	91H	8251A Serial I/O Command Status		
92H -	9FH	Reserved		
A0H -	A1H	Interrupt Controller 8259A Read/Write		
A2H -	AFH	Reserved		
	B0H	FDC 8272 Main Status Register		
	B1H	FDC 8272 Data Register Read/Write		
B2H -	BFH	Reserved		
	C0H	CRT/KYBD 8255A		Port A Data Read/Write
	C1H	CRT/KYBD 8255A		Port B Bit 0 through 7
			PB0	RQFD-B(A) (Disk Request)*
			PB1	RQM0-B(A) (MMIO J1/J2 Request)*
			PB2	RQM1-B(A) (MMIO J3/J4 Request)*
			PB3	Disk Ready
			PB4	MPST0 (MMIO Present J1)
			PB5	MPST1 (MMIO Present J2)
			PB6	MPST2 (MMIO Present J3)
			PB7	MPST3 (MMIO Present J4)
	C2H	CRT/KYBD 8255A		Port C Bit 0 through 7
			PC0	RQFD-A(B) (Disk Request)
			PC1	RQM0-A(B) (MMIO J1/J2 Request)
			PC2	RQM1-A(B) (MMIO J3/J4 Request)
			PC3	KBINT, (Kybd Interrupt)
			PC4	STBA/B (CRT/KYBD Data Strobe)
			PC5	IBF, (Input Buffer Full)
			PC6	ACKA/B (CRT/KYBD Data Acknowledge)
			PC7	OBFA/B (Output Buffer Full)
	C3H	8255A Control Byte		
C4H -	CFH	Reserved		
	D0H	FDC Terminal Count		
D1H -	DFH	Reserved		
E0H -	E2H	Line Printer 8255A		
	E3H	PROM Programming/Emulator Power Control		
E4H -	EFH	Reserved		
	F0H	Boot ROM Disable		
F1H -	FFH	Reserved		

*Signals from the other processor.

In the following descriptions, it is assumed that the user is familiar with I/O programming techniques and with the I/O chips used. Refer to the current *Intel Component Data Catalog* for further information. Refer to Appendix A for the pin assignments and I/O signals.

CRT and Keyboard I/O

This section describes the keyboard characters available and their interpretation by the system. It also describes the graphics characters.

Most of the keyboard characters are sent to ISIS by the CRT/Keyboard controller as the ASCII code corresponding to the character. The following exceptions to this rule may affect user-written programs that read characters input at the keyboard:

1. Triple key operations are undefined. For example, pressing the FUNCTION key, the SHIFT key, and some other key simultaneously does not have a defined effect.

2. The up arrow key generates the code 1EH instead of CTRL-↑. The left arrow key generates the code 1FH instead of CTRL←.

3. The FUNCTION key only has affect with upper case alpha characters and the digits 0-9. The result of these function characters is that the ASCII code for the character with the most significant bit set is sent to the processor (except for FUNCT-R, FUNCT-S, and FUNCT-T). Lower case function characters are converted to the corresponding upper case function character and sent as an upper case function character.

4. The following function characters are processed directly by the CRT/Keyboard controller and are not sent to the processor:

FUNCT-T	Typewriter/Non-typewriter mode switch
FUNCT-S	CRT scrolling speed switch
FUNCT-HOME	Processor keyboard/screen assignment switch
FUNCT-↑	Increase size of lower half of split screen display
FUNCT-↓	Decrease size of lower half of split screen display
FUNCT-R	Interrupt processor currently assigned

Cursor Addressing and Graphics Mode. The cursor location on the CRT screen can be programmed. To control the cursor location from a program, output an ESC (1BH) followed by another ASCII character as defined below. Use the CO system call two times to output the two bytes (1BH, <ASCII code>). Since the cursor location is relative to a full screen; in a split screen with dual processors, the cursor may not appear on the physical screen.

ESC, A Move the cursor up one line.

ESC, B Move the cursor down one line.

ESC, C Move the cursor to the right one character.

ESC, D Move the cursor to the left one character. If the cursor is at the first character of a line, it is wrapped around to the last character of the previous line.

ESC, E Home cursor and clear the screen.

ESC, H Home the cursor.

ESC, J Erase from the current location of the cursor to the end of the screen.

ESC, K Erase the line containing the cursor from the cursor to the end of the line.

The following sequence moves the cursor to a specified address on the screen. The address is given as an x,y coordinate with an offset of 20H. Thus, to move the cursor to the first character (character 0) on the first line (line 0), the address would be 20H,20H.

ESC, Y, <x>, <y> Move the cursor to the address specified in the third and fourth bytes output. Use the CO system call to output the bytes. Add 20H to the absolute values for <x>, and <y>, since the CRT/Keyboard controller subtracts 20H from the value it receives. This offset is used to be compatible with other products.

In addition, the CRT/Keyboard controller is capable of generating a set of graphics characters that can be displayed on the screen by a user-written program.

The steps to follow in writing a program to output graphics characters to the screen are:

1. Enter graphics mode by outputting the sequence ESC, G. Use the CO system call two times to output the two bytes (1BH, 47H).

2. Move the cursor to the desired location by outputting one of the cursor location control sequences to the CRT screen. Use the CO system call to output the cursor control sequences described previously.

3. Use the CO system call to output the code for the desired graphics symbol or ASCII character. The ASCII codes and the codes for graphics symbols are given in Appendix C.

4. Repeat steps 2 and 3 until the entire graphics display is completed.

5. Exit from graphics mode by outputting the sequence ESC, N. Use the CO system call two times to output the two bytes (1BH, 4EH).

The escape sequences to enter and exit graphics mode are:

ESC, G Enter graphics mode. In graphics mode, any control characters (00H-1EH except 02H, alternate escape, or 1EH, escape) that are output will be displayed as the graphics symbol corresponding to the code. The codes and their corresponding graphic symbols are given in Appendix C. Other characters (20H-7EH) will be displayed as the corresponding ASCII characters.

ESC, N Exit from graphics mode.

Serial I/O

ISIS provides an I/O driver and operating system commands for the serial I/O port. The following information is provided for those who wish to write a customized I/O driver.

The 8253 Programmable Interval Timer is used to provide software control for the baud rate on the serial I/O port.

Intended Baud Rate to be Used	Programmable Baud Rate Generator Nominal Output Frequency (KHz)		
19200	19.2	---	---
9600	9.6	153.6	---
4800	4.8	76.8	307.2
2400	2.4	38.4	153.6
1200	---	19.2	76.8
600	---	9.6	38.4
300	---	4.8	19.2
150	---	2.4	9.6
110	---	1.76	7.04

The 8253 Programmable Timer generates the frequencies shown in the preceding table corresponding to the desired baud rate. The frequency required for a given baud rate depends on the 8251 mode instruction (1X, 16X, or 64X). The maximum allowable frequency deviation is ± 1%.

The I/O address assignment for this baud rate generator, the 8253, is:

80H: Load Counter 0 with value to generate frequency corresponding to desired baud rate on an output instruction, Read Counter 0 on an input instruction

83H: Counter 0 Mode Select

The input frequency to the counter is 1.53846 MHz ± 0.1%. This frequency is the value to be divided by the 1X, 16X, or 64X frequency in the preceding chart to generate the value to load into the counter.

The serial I/O port consists of an 8251A USART and RS-232 receivers and drivers. It provides full duplex asynchronous communication from 110 to 19200 baud using 7 bits plus a parity bit.

The port may be jumpered to use the internal 8253 Programmable Timer or an external timer as a clock. The clock can provide a signal either 16 times or 64 times the actual baud rate. See the Installation Instructions in Appendix A for setting the jumper.

The I/O addresses for the 8251 USART are:

90H: Data I/O

91H: Command/Status

Printer I/O

The printer port for Processor A uses an 8255 Programmable Parallel Interface. The 8085 I/O address assignment is:

8085 Port Address	Function
E2H:	Write Data to 8255 Port A (PA0 - PA7)
E0H:	Read 8255 Port C (PC0-PC3) Write 8255 Port C (PC4-PC7)
E1H:	Write Control Byte

The assignments for the bits in Port A and Port C on the 8255 are shown in the following chart:

Port No.	Function	Mode
PA0	DATA0	Output
PA1	DATA1	Output
PA2	DATA2	Output
PA3	DATA3	Output
PA4	DATA4	Output
PA5	DATA5	Output
PA6	DATA6	Output
PA7	DATA7	Output
PC0	SELECT	Input
PC1	BUSY	Input
PC2	$\overline{\text{ACK}}$	Input
PC3	$\overline{\text{FAULT}}$	Input
PC4	$\overline{\text{STB}}$	Output
PC5	$\overline{\text{PRIME}}$	Output
PC6	N/C	
PC7	N/C	

The 8255 control bytes are the Mode Select Byte and the Bit Set/Reset Byte. The Mode Select Byte is output at 8085 port E1H. A value of 87H (1000 0111) selects the following mode assignment for the 8255:

8255 Port A (PA0 - PA7) Output
8255 Port C (PC0 - PC3) Input
8255 Port C (PC4 - PC7) Output
8255 Port B (PB0 - PB7) Output *
* Used for Plug-in Module Adapter Power On/Off Control

The Bit Set/Reset control byte is output at the 8085 port E0H and provides single bit set/reset control for PC4-PC7 of Port C on the 8255. The following chart gives the value that is output at the 8085 port E0H to set and reset bits PC4-PC7 of Port C.

8255 Port	Set	Reset
PC4	09H (00001001B)	08H (00001000B)
PC5	0BH (00001011B)	0AH (00001010B)
PC6	0DH (00001101B)	0CH (00001100B)
PC7	0FH (00001111B)	0EH (00001110B)

Multimodule I/O

Up to four 8-bit iSBX Multimodule™ boards are supported with some restrictions. Sixteen bit Multimodules are not supported. Only non-DMA mode I/O is supported. Only limited power consumption boards are supported. See the power supply specification in Appendix A. See the *Intel iSBX® Bus Specification*, order no. 142686, for further information on the multimodule bus interface.

Most of the information required for using Multimodules is contained in the Hardware Reference Manual for each Multimodule. Some of the additional precautions which should be followed when using Multimodules with the iPDS system are as follows:

- Multimodule interrupts are returned to the INTR line of the CPU (both base and optional processor). These interrupts are maskable.

- Multimodule interrupts are sent to both processors in a dual processor system. Mask off the other processors interrupts. The base processor should enable only those interrupts ISIS does not allow. (See the ATTACH call).

- Data cannot be transferred between a Multimodule and a processor unless the proper semaphores are first set up.

- The ATTACH and DETACH commands (chapter 5) must be uset to communicate with Multimodules in the iPDS system.

Communication between the Multimodule Adapter board and wither the base or optional processor board is controlled by an 8255 Programmable Peripheral Interface (PPI) chip on each processor board. The following I/O port address list covers the ports needed to use Multimodules in the iPDS system.

Port Address	Port Function	Comments
40H - 47H	Multimodule Chip Select J1	Functions selected by A processor for MMIO at J1
48 - 4FH	Multimodule Chip Select J1	Functions selected by B processor for MMIO at J1
50H - 57H	Multimodule Chip Select J2	Functions selected by A processor for MMIO at J2
58H - 5FH	Multimodule Chip Select J2	Functions selected by B processor for MMIO at J2
60H - 67H	Multimodule Chip Select J3	Functions selected by A processor for MMIO at J3
68H - 6FH	Multimodule Chip Select J3	Functions selected by B processor for MMIO at J3
70H - 77H	Multimodule Chip Select J4	Functions selected by A processor for MMIO at J4
78H - 7FH	Multimodule Chip Select J4	Functions selected by B processor for MMIO at J4
A0H	Interrupt Controller R/W	Used to set up the 8259A Interrupt Controller for the Control Register. See Table 8-3 for pin assignments.
A1H	Interrupt Controller R/W	Used to set up the 8259A Interrupt Controller for the Data Register. See Table 8-ff for pin assignments.
C1H	PPI Port B, bits 0 through 7	
	PB0 - RQFD	Disk semaphore (Semaphore available=1)
	PB1 - RQM0	MMIO J1/J2 semaphore (Semaphore available =1)
	PB3 - Disk Ready	
	PB4 - MPST0	Multimodule present at J1
	PB5 - MPST1	Multimodule present at J2
	PB6 - MPST2	Multimodule present at J3
	PB7 - MPST3	Multimodule present at J4
C2H	PPI Port c, bits 0 through 2	
	PC0 - RQFD	Set disk request bit and turn disk motor on (Motor ON=0)
	PC1 - RQM0	Enable Multimodules at J1/J2 (MMIO Enable =0)
	PC2 - RQM1	Enable Multimodules at J3/J4 (MMIO Enable=0)

C3H PPI Control Byte, bits 0 through 2

- bit 0 - Request the MMIO semaphore (Request semaphore=1)
- bit 1 - Request Multimodules J1/J2 (Request MMIO=1)
- bit 2 - Request Multimodules J3/J4 (Request MMIO=1)

Table 8-3 shows the pin numbers of the interrupt lines from the MMIO to the 8259A interrupt controller chips.

Table 8-3 Interrupt Line Pin Numbers

MMIO Pin Number	U-5 and U-6 8259A Pin Number
J1 Pin 12	IR0-0 Pin 18
J1 Pin 14	IR1-0 Pin 19
J2 Pin 14	IR2-0 Pin 20
J2 Pin 12	IR3-1 Pin 21
J3 Pin 14	IR0-1 Pin 22
J3 Pin 12	IR1-1 Pin 23
J4 Pin 14	IR2-1 Pin 24
J4 Pin 12	IR3-1 Pin 25

Peripheral Device I/O Operations

One of the most important features of an operating system is its I/O capabilities. The ISIS-PDS operating system provides a simple and uniform method of identifying each peripheral device and each file. By assigning symbolic names to devices and files, system resources can be accessed without requiring the user to remember the physical addresses of each device and each file.

The following sections describe file I/O, dynamic file control, line edit files, and disk file types.

File I/O

A device is a peripheral connected to the system hardware. A device name is of the form:

:<device>:

where <device> is a two-character mnemonic assigned by the system to a supported peripheral. A complete list of ISIS-PDS devices and names is given in Chapter 5 as well as a discussion of logical and physical device names, filenames, and pathnames.

A file is an abstraction of a peripheral device and is a collection of information in machine readable form. The term file refers to the data stored on a peripheral device. A file is formally defined as a sequence of 8-bit values, or bytes.

No file can exist on more than 1 device. In particular, a disk file must reside entirely on one diskette.

A filename is of the form:

<name>.<extension>

where <name> is a sequence of one to six characters. The initial character must be a letter or a digit. The <extension> is a sequence of one to three characters: letters or digits.

For all non-disk devices, the filename is blank. Thus, the device named :LP: is identical to the file with the pathname :LP:. Pathnames are described in Chapter 5. All disk devices must have a non-blank filename.

User programs perform I/O by making calls to the ISIS kernel, i.e., system calls. All I/O occurs to and from files, not devices, and is status driven rather than interrupt driven. (Interrupts 1, 5.5, 6.5, and 7.5 are reserved for ISIS and must not be masked or altered by user programs.) Programs receive information by reading from an input file and transmit information by writing to an output file. Files are accessed by their pathnames; see Chapter 5 for a discussion of pathnames.

ISIS-PDS usually does not interpret the byte values of a file as having any special meaning. The exception is with line edited files described in a later section of this chapter. However, the CLI, as well as some commands and user-written programs, may expect a certain kind of value to be present.

For example, the CLI expects the byte values of a file to represent a sequence of machine language instructions that can be loaded into memory and executed. If a text file is executed, unpredictable results may occur.

See later sections of this chapter for information on disk file types and disk file formats.

Four files (disk files, the byte bucket, the console input file, and the console output file) deserve further attention.

One of the major purposes of ISIS-PDS is to implement files on diskettes and bubbles. A diskette is the recording medium for 5-1/4" flexible disks while a bubble is the recording medium for bubble memory.

A drive is the mechanism on which the recording medium is mounted. A disk is the drive together with a mounted diskette or bubble.

The term disk is often used in place of diskette and bubble to refer to either recording medium.

Disk files are discussed in greater detail later in this chapter.

The byte bucket is a virtual I/O file with a pathname of :BB:. The byte bucket acts as an infinite sink for bytes when written to and as a file of zero length when read from. The :BB: device can be used as the line printer and serial I/O devices on the optional processor, since the optional processor cannot access line printer and serial devices. Multiple opens of :BB: are allowed. Each open is treated as the open of a different file and returns a different connection number (Active File Table Number). See the OPEN system call for a description of connection numbers (AFTNs).

The operating system supports a virtual console which is implemented as two files, an input file with a pathname of :CI: and an output file with a pathname of :CO:. (Since these are non-disk files, the nine character filename is blank, and the pathname is the same as the device name.) These two files are always open. :CI: is always a line edited file; :CO: is its associated echo file. Line edited files and echo files are discussed in a later section.

Both :CI: and :CO: are pseudonyms for the file corresponding to an actual physical device. At initialization, :CI: and :CO: refer to the video terminal (:VI: and :VO: respectively), i.e., the keyboard and CRT. User programs can change the two halves of the console to different physical devices.

The ASSIGN command can also be used to change the assignment of the console. The :CI: assignment is automatically redefined to the keyboard whenever an end of file is encountered.

The CLI always obtains its command lines from the physical device currently assigned to the console.

Dynamic File Control

Dynamic file control means that file access (I/O operations) are under program control at runtime. With ISIS-PDS, a list of twelve files or devices to be used by a program can be maintained while physical access is restricted to a smaller group of six files actually in use at any one time. The existence and maintenance of the list of devices is accomplished through ISIS system calls and can make I/O operations more efficient. See the discussion of the Active File Table and the OPEN system call for further information.

Line Edited Files

Data read from input files can be filtered through an ISIS module called the line editor. Files read in this way are called line edited files. They are the only files whose byte values are interpreted specially by ISIS.

ISIS assumes that the byte values of a line edited file represent ASCII characters. Three groups of ASCII characters are given special meaning by ISIS: characters that terminate a line, editing characters, and function characters. Function characters are described in Chapter 9. Terminating characters and editing characters are described in later sections of this chapter.

Line edited files are provided for (but not restricted to) the case of a user typing characters at a keyboard. They allow the input to be edited until correct prior to being read by the READ system call. Editing is accomplished by entering characters, called editing characters, that are interpreted as having special meaning by ISIS.

By supplying a parameter in the OPEN system call, line edited files can be read. Thus, a file is a line edited file by virtue of its access method, not because of any intrinsic attribute of the file.

Every line edited file has an echo file associated with it. The echo file reflects the current contents of the line edited file. The previously opened echo file is also specified in the OPEN system call. If no echo is desired, the byte bucket (:BB:) can be opened as the echo file.

Line edited files are partitioned into segments, called logical lines, according to the following rules. Linefeed and escape are referred to as break characters.

1. A linefeed (LF) is inserted following every carriage return (CR). Any LF that follows a break character is removed from the buffer and ignored.

2. A logical line is defined as all the characters between two break characters plus the terminating break character (the line terminator).

3. If all logical lines are shorter than 122 uncancelled characters, the partitioning is complete. Lines longer than 122 uncancelled characters are further partitioned into two segments. The first segment is the longest proper substring that is less than 121 uncancelled characters; the second is the remaining characters.

4. Rule 3 is applied until all long lines are eliminated.

Uncancelled characters include only those characters remaining after the line has been edited. The editing characters do not count as uncancelled characters. Terminating characters and function characters do count as uncancelled characters. A READ to a line edited file obtains a maximum of one line at a time.

Terminating Characters

While a line is being entered from an input device, it is accumulated into a 122-character line editing buffer. While still in the buffer, the line can be changed using the editing characters described below. No data is transferred to the program reading the line edited file until the line is terminated. The line editing buffer can be terminated in one of three ways:

- A linefeed character (ASCII code 0AH) is entered. Linefeed is automatically appended to the carriage return (ASCII code 0DH) when the RETURN key is pressed
- A non-editing character is entered as the 122nd character

A linefeed which is the first (and therefore the only) character in a line has no effect. It is ignored and there is no echo. This feature permits disk files with CR LF terminators to be used as line edited files. The CR generates a LF automatically yielding a CR LF LF, but the second LF is removed from the buffer and ignored.

The ESCAPE character is echoed as a dollar sign ($).

Editing Characters

A READ system call transfers no characters from a line until the line has been terminated. During the physical input, the line is accumulated in the line editing buffer and can be modified with the editing characters described in Chapter 3.

Reading From the Line Editing Buffer

When the line has been terminated, the next (i.e., pending) READ system call transfers the specified number of bytes from the line editing buffer to the requesting program buffer. When the number of bytes entered to the buffer is greater than the number requested by the program, ISIS keeps track of the characters read and returns the remaining bytes in response to subsequent READs.

For example, if the line editing buffer contains 100 characters and a READ system call is issued with a count of 50, the first 50 characters are transferred to the requesting program buffer. The next READ system call transfers characters starting at the 51st character. The term MARKER is used in later discussions to refer to the position of the next byte to be processed in a file.

If the READ system call requests 100 bytes and the line editing buffer only contains 50 bytes, only 50 bytes are transferred.

A READ system call returns bytes from only one logical line at a time. This means no more than 122 characters can be read. If the READ system call requests 200 bytes, only 122 bytes are transferred.

When all the characters in the line editing buffer have been read, the buffer pointer is positioned after the last character. The buffer contents are not destroyed. In fact, the RESCAN system call can be used to reposition the buffer pointer to the beginning of the buffer and the line can be read again.

When the buffer has been completely read, with the pointer after the last character, a new READ system call will transfer new input from the line edited file into the line editing buffer. When the line is terminated, the number of characters requested by the READ are transferred to the program.

Reading a Command Line

Reading a command line from the console input device is a special case of reading a line edited file.

When a command is entered at the console, it is collected in the line editing buffer and is not available to the CLI until it is terminated. The CLI reads only the command name (down to the first space) and then loads the disk file corresponding to that name. The file should be an absolute MCS-80/85 object module.

The line editing buffer pointer is positioned after the command name. Thus, the loaded program can issue a READ system call to transfer the first parameter, or it can issue a RESCAN system call to position the pointer to the beginning of the buffer and re-read the command name.

For example, suppose the following command has been entered:

```
A0>COPY :F1:PROGA TO :F0:PROGB
```

The line editing buffer for the console input file contains 29 characters as follows. CR represents the carriage return (ASCII code 0DH) and LF represents the line-feed (ASCII code 0AH).

```
COPY :F1:PROGA TO :F0:PROGBCRLF
12345678901234567890123456 7 8 9
         10        20
```

The CLI reads the characters COPY and loads the file with the pathname :F0:COPY. The buffer pointer is left pointing to the fifth character, a space following COPY. The file :F0:COPY is a program that issues a READ system call to read in the parameters, the names of the source and destination files to be copied.

When the buffer pointer passes the CR LF characters, the line is terminated and the next READ system call inputs new data from the console input to the line editing buffer.

Remember that when control is passed from the CLI to the loaded program, the buffer pointer is positioned after the command name, not after the CR LF. If there are no parameters for the command, the next READ system call returns to the requesting program the CR LF left over from the previous command line.

For example, suppose the following command line is entered to run a user program:

```
A0>PROGA
```

and the line editing buffer contains:

```
P R O G A CR LF
1 2 3 4 5  6  7
```

When the program is loaded into memory, the line editing pointer is at the CR. If subsequent input is expected by the user written program PROGA, an extra READ system call must be issued to clear the buffer of the CR LF terminating characters.

If the program does not request any console input, the remaining CR LF terminator is cleared by ISIS before a new command line is read by the CLI.

Disk File Types

With the exception of line edited files described in a previous section, the operating system makes no assumptions about and places no interpretation on the bytes in the data blocks of files. Data blocks are described in a later section. However, certain programs assume the data to be formatted and interpret the data in certain ways.

The CLI expects the file it loads into memory to be an MCS-80/85 absolute object file. Intel language translators, linkers, and locators produce files of the expected format.

An absolute object file contains a form of machine language instructions and data that permit the file to be loaded into memory for execution. In addition, it may contain control information that may govern the loading process. An absolute object file can be used to program a ROM or a PROM.

Fields in absolute object files are all reserved for use by Intel even if they are not described. Any use of or modification to these fields may interfere with the proper functioning of the program. The MCS-80/85 absolute object file format is summarized below for convenience.

Notation Used to Describe Records. The record format diagrams use the conventions shown in figure 8-6:

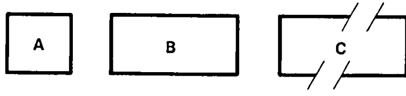


Figure 8-6 Record Format Conventions

The first rectangle represents a single-byte field, the second rectangle represents a two-byte field, and the third broken rectangle represents a field of a variable number of bytes.
Some records contain a field or a series of fields that may be repeated. These are indicated by the REPEATED or RPT brackets in the diagrams.

Any field that contains a name has the following internal structure: the first byte contains a number from 1 to 255 inclusive which indicates the number of the remaining bytes in the field. These remaining bytes are interpreted as a byte string. Most translators constrain these values to ASCII codes of printing characters.

Any field with an X through it contains unspecified information and is not relevant to the translator.

MCS-80/85 Absolute Object File Format. The format of an absolute object file contains three record formats:

Module Header Record
Content Record
Module End Record

A proper absolute object file contains these record formats in the following order:

- One Module Header Record
- One or More Content Records
- One Module End Record

It may also contain other records between the Module Header Record and the Content Record. These fall into two categories:

- Extraneous records containing symbolic debug information which will be ignored by the CLI
- Erroneous records containing relocation information which indicates that the information is still in relocatable form. The file will be rejected by the CLI

The Module Header Record, Content Record, and Module End Record all share the following organization: the first byte identifies the record and is called the RECORD TYPE, the next two bytes contain a number called RECORD LENGTH, the last byte in every record is the CHECKSUM field. The RECORD LENGTH is the total number of bytes in the record exclusive of the first three bytes, the RECORD TYPE and the RECORD LENGTH. The CHECKSUM contains the two's complement of the sum, modulo 256, of all the other bytes in the record.

Module Header Records are RECORD TYPE number 02H; Content Records are RECORD TYPE number 06H; and Module End Records are RECORD TYPE number 04H. Extraneous records have the RECORD TYPE numbers of 08H, OEH, 10H, 12H, 16H, 18H, or 20H depending on how they were created. Erroneous records have all other numbers.

In addition to the common fields, the module header record (see figure 8-7) contains the module name field. Every module has a name. A valid module name contains between 1 and 31 characters each of which must be an uppercase letter (A, B, C, . . ., Z), a digit (0, 1, 2, . . ., 9), a question mark (?), or the at sign (@). The first character may not be a digit.

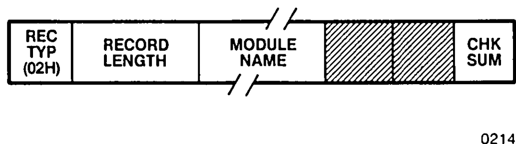


Figure 8-7 Module Header Record

The Content Record (see figure 8-8) provides contiguous data, from which a memory image may be constructed for a portion of memory.

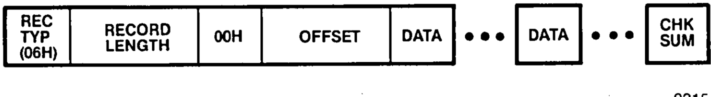


Figure 8-8 Content Record

The OFFSET field specifies the absolute location of the first data byte.

Following the OFFSET are one or more DATA bytes. Thus, this record provides N consecutive bytes of a memory image from OFFSET through OFFSET + N - 1, inclusive.

The Module End Record (see figure 8-9) has a MODULETYPE (MODTYPE) field with a value of 0 or 1. If the value is 1, the module is a main program. If the value is 0, the module is not a main program.

If the module is a main program, the OFFSET field specifies the module's execution start address. Otherwise, this field has no significance, but it must be present.

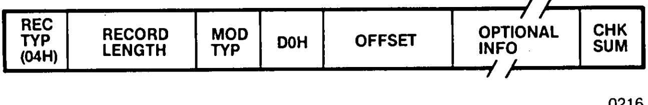


Figure 8-9 Module End Record

The OPTIONAL INFORMATION field may not be present depending on the language translator used. It contains debug information.

Disk Structure

This section describes the structure of disks and disk files at the byte level. The information is not necessary to use the system calls described previously. Disk devices are discussed in Chapter 5. This section deals with the disk media and the structure of the files recorded on it.

Bubble memory multimodules are treated as virtual disk devices. Both flexible disks and bubble memory are organized into tracks and sectors. All flexible disks contain 80 tracks which are divided into 32 sectors of 256 bytes each. (Sixteen sectors are on side 0 and sixteen on side 1.) Bubble memory contains 16 tracks with 32 sectors of 256 bytes each. Disk capacities are given in the following chart.

	Diskette	Bubble Memory
Tracks/Disk	80	16
Sectors/Track	32	32
Sectors/Disk	2560 *	512
Bytes/Sector	256	256
Bytes/Disk	655,360	131,072

* Only 2544 sectors available to the user.

General Disk File Structure

Each disk contains a number of files. Each file is made up of 256-byte blocks. Each block corresponds to one disk sector, which is a hardware addressable unit. See figure 8-10. ISIS-PDS system files and commands occupy about 400 sectors on a system disk and 50 sectors on a non-system disk.

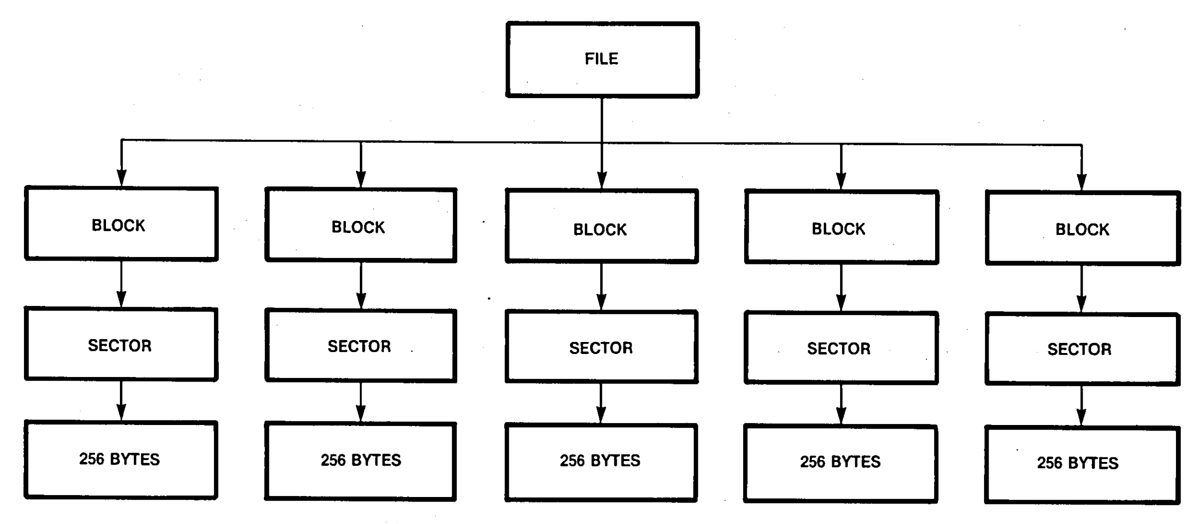


Figure 8-10 Disk File Components

Each sector on a disk has a unique address by which it can be accessed. The address consists of a one-byte track number and a one-byte sector (block) number. Tracks are numbered 0-79 on a diskette and 0-15 on bubble memory. The sectors on a track are numbered 1-32 on both diskettes and bubble memory. The address of a block is also referred to as a pointer to that block. Related blocks are linked together by pointers. That is, two of the bytes in a block may contain the address of a related block.

Blocks. A block is the data in one sector. There are two types of blocks in a file: pointer blocks and data blocks. Pointer blocks contain nothing but pointers to other blocks as shown in figure 8-11.

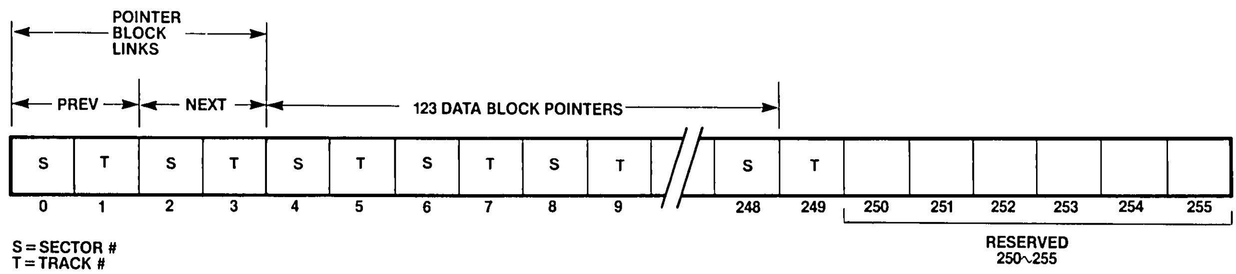


Figure 8-11 Pointer Block

All files begin with a pointer block that is called the header block. If the file contains fewer than 123 data blocks, the header block is the only pointer block in the file. If there are more than 123 data blocks, there is an additional pointer block for every 123 data blocks. For example, a file of 300 data blocks contains 3 pointer blocks, including the header block.

The first two pointers in a pointer block are links to other pointer blocks in the file. The first link contains the address of the previous pointer block. The header block always contains zeros in this field because there is no previous pointer block in the file. The second link contains the address of the next pointer block in the file. The last pointer block in the file has zeros in this field.

Following the links to other pointer blocks are 123 pointers to the data blocks in the file followed by six reserved bytes. If a pointer contains zeros, then no data block has been allocated for the pointer. A zero pointer does not necessarily mark the end of the file.

Data blocks, as shown in figure 8-12, have no particular format, since they contain user data.

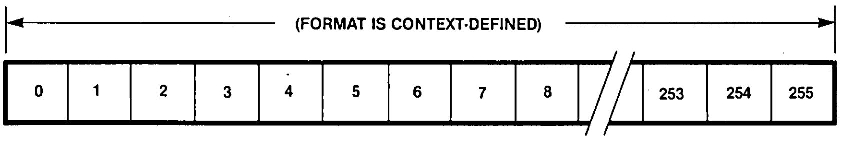


Figure 8-12 Data Block

Data blocks are fundamentally different from pointer blocks. Data blocks are visible to users; they contain the information that is transferred by read and write operations. Pointer blocks, on the other hand, are invisible to users; the data they contain is of interest only to the system. Data blocks can be thought of as destinations with pointer blocks as paths to those destinations. To access user data in a file, ISIS follows a path of pointers to a data block.

The relationship of pointer and data blocks is shown in figure 8-13.

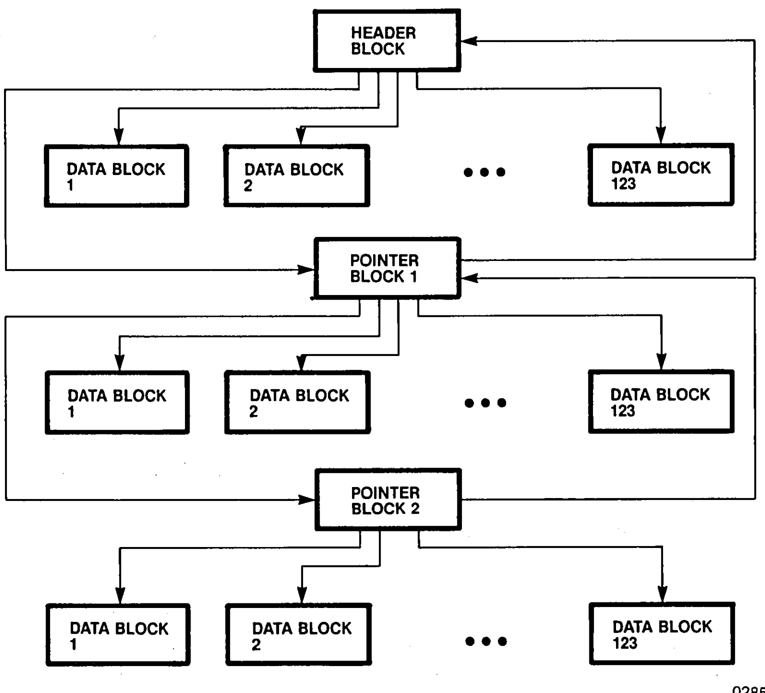


Figure 8-13 Relation of Data and Pointer Blocks

Figure 8-14 shows an example file which consists of data in five blocks: DATA 1 through DATA 5. The diagram in figure 8-14 is simplified in that it shows only four sectors per track instead of 32 and only two data pointers per pointer block instead of 123. The file begins at the header block which contains pointers to the first two data blocks, DATA1 and DATA2.

The header block is linked to a second pointer block at sector 3 of track 8. The second pointer block contains pointers to DATA3 and DATA4. It is linked back to the header block and forward to the last pointer block at sector 1 of track 9.

The third pointer block contains the last data pointer in the file. Because it is the last pointer block, it contains a backward link to the second pointer block but no forward link.

Notice that it is the data pointers which order the data blocks for sequential access. The physical locations of the data blocks and the pointer blocks are irrelevant. Because of this ability to scatter files on the disk, the system can make efficient use of available space. Note also the data capacity is reduced by the number of pointer blocks on a disk.

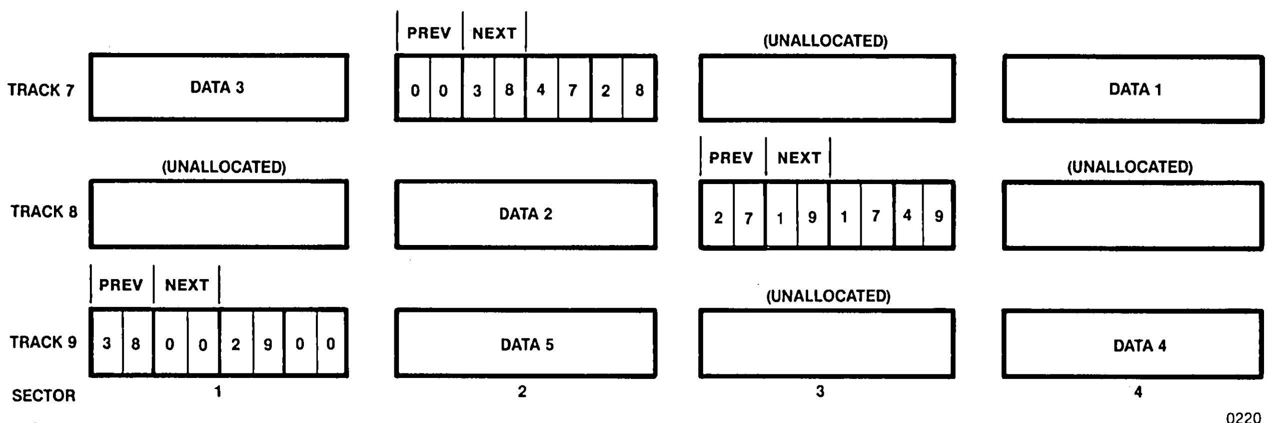


Figure 8-14 Pointer and Data Blocks in a File

Interleaving Factors. Interleaving factors are used to speed up the sequential access of blocks on the same track. Often a program reads a block, processes that block, reads the next block, and so on. If the blocks were stored in physically adjacent sectors, the disk drive read/write head would pass by the second sector while the program was processing the first sector. The program would have to wait one full disk revolution for the read/write head to seek the second sector again. The effect of an interleaving factor of 3 is shown in figure 8-15.

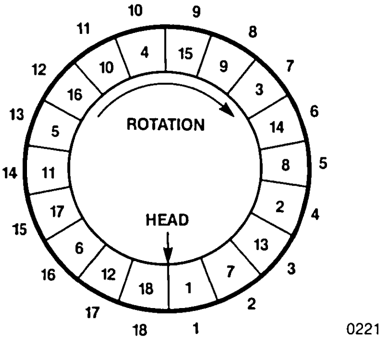


Figure 8-15 Sector Interleaving

Physical sector addresses are shown outside the track (which is simplified for the drawing to show only 18 sectors). Logical sector addresses, which are the basis for accessing blocks stored in the sectors, are shown on the track.

With interleaving, a program which reads and processes every block in logical sequence has a processing window equivalent to the time it takes for two sectors to pass by the head.

Assuming that processing each block takes slightly less time than is available in this window, all 18 blocks can be processed in three revolutions of the disk. Without interleaving, 18 revolutions would be required.

The ISIS-PDS operating system uses an interleaving factor of 4 except for Track 0, Sectors 1-16 which has an interleaving factor of 1. Track 0 contains the file ISIS.T0 which is formatted at 128 bytes/sector and contains information needed to initialize the system. The interleaving information is used by the IDISK command when disks are formatted.

System Disk Files

All ISIS non-system disks contain four system files: ISIS.T0, ISIS.LAB, ISIS.DIR, and ISIS.FRE. These are created automatically when the disk is initialized.

The location of these files is fixed as shown in table 8-4. The FROM and THRU values are given in the form T,S where T is the track number and S is the sector number. The values are given in hexadecimal.

Table 8-4 System File Locations

File Name		Double Density Mini-Diskette		Bubble Memory	
		FROM	THRU	FROM	THRU
ISIS.T0	(Header)	00H,11H	00H,11H	00H,11H	00H,11H
	(Data)	00H,12H	00H,20H	00H,12H	00H,20H
ISIS.LAB	(Header)	01H,01H	01H,01H	01H,01H	01H,01H
	(Data)	01H,02H	01H,04H	01H,02H	01H,04H
ISIS.DIR	(Header)	27H,01H	27H,01H	00H,01H	00H,01H
	(Data)	27H,02H	27H,10H	00H,02H	00H,04H
ISIS.FRE	(Header)	27H,11H	27H,11H	00H,05H	00H,05H
	(Data)	27H,12H	27H,14H	00H,06H	00H,08H

On a system disk, 6 files are reserved for the operating system. These are: ISIS.PDS, ISIS.CLI, ISIS.TO, ISIS.LAB, ISIS.DIR, and ISIS.FRE. Note that four of these appear on a non-system disk as well. In addition to these six files, there are a number of command files containing programs and a library file named SYSPDS.LIB.

ISIS.PDS

This file contains the ISIS kernel, that is, the resident system routines.

ISIS.CLI

This file contains the command line interpreter which occupies part of the user program area of memory.

ISIS.T0

This file contains a program called T0BOOT. When the RESET button is pressed, this file is read in from the disk. Once it is loaded into memory, T0BOOT begins executing. This program reads the contents of ISIS.PDS and displays the ISIS sign on message. If an attempt is made to initialize the system from a non-system disk, T0BOOT displays the message:

NON-SYSTEM DISKETTE

on the screen. When running from a hardware reset, T0BOOT then returns control to the initialization PROM. If running from a FUNCT-R (a software reset), T0BOOT stops after attempting to initialize from the system diskette.

ISIS.LAB

The first nine bytes of this file contain the disk label stored as nine ASCII characters with a six-character name and a three-character extension.

The rest of the bytes are undefined except for the last 256 bytes (corresponding to Track 1, Sector 4) which are filled with repetitions of the ASCII characters:

DIAGNOSTICSECTOR

These bytes are used by diagnostic programs described in Appendix A.

ISIS.DIR

Each disk contains one directory. This file contains 15 data blocks (3 for bubble memory); each block has room for 16 directory entries. One entry is used for each file on the disk, so there is room in the directory for 240 files (48 for bubble memory). Each directory entry is 16 bytes long and is formatted as shown in figure 8-16.

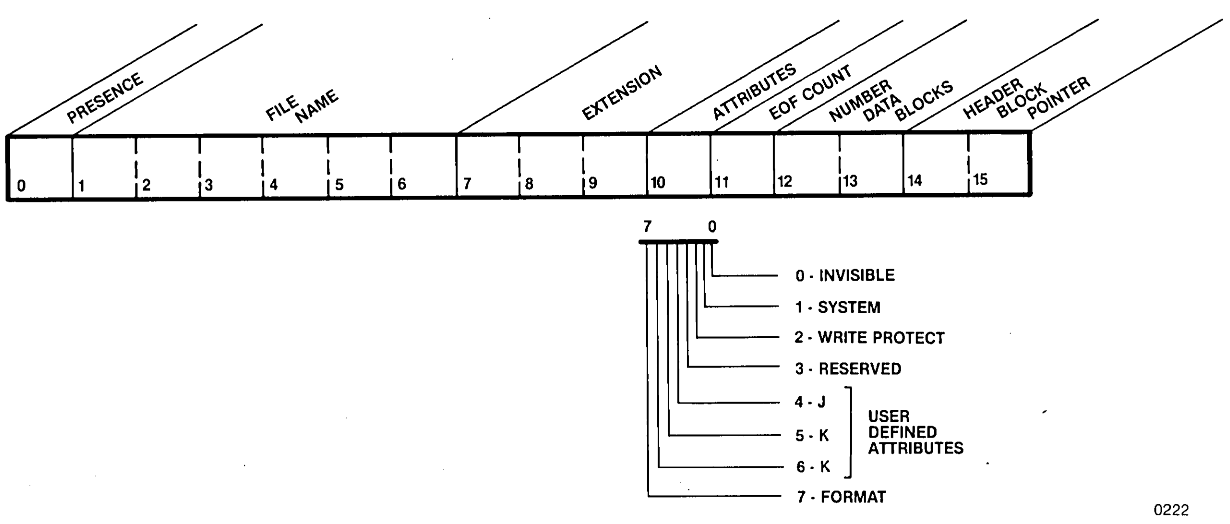


Figure 8-16 Directory Entry

The following chart explains the field names used in figure 8-16.

PRESENCE is a flag which can contain one of three values:

- 00H: The file associated with this entry is on the disk.
- 7FH No file is associated with this entry; the content of the rest of the entry is undefined. The first entry with its flag set to 7FH marks the current logical end of the directory; directory searches stop at this entry.
- FFH The file named in this entry once existed on the disk, but is currently deleted. The next file added to the directory will be placed in the first entry marked FFH. This flag cannot, therefore, be used to find a file that has been deleted, unless no other files have been created or written since the deletion. A value of FFH should be thought of as marking a free directory entry.

FILENAME is a string of up to 6 non-blank ASCII characters specifying the name of the file associated with the directory entry. If the filename is shorter than six characters, the remaining bytes contain 00H. For example, the name ALPHA would be stored as 41H 4CH 50H 48H 41H 00H.

EXTENSION is a string of up to three non-blank ASCII characters that specify an extension to the filename. Extensions often identify the type of data in the file such as OBJ for object module or PLM for PL/M source module. As with filename, unused positions in the extension field are filled with zeroes.

ATTRIBUTES are bits that identify certain characteristics of the file. A 1 bit indicates that the file has the attribute, while a 0 bit means that the file does not have the attribute. The bit positions and their corresponding attributes are listed below (bit 0 is the low order or rightmost bit, bit 7 is the leftmost bit):

- 0: Invisible. Files with this attribute are not listed by the DIR command unless the I option is used. All system files are invisible.
- 1: System. Files with this attribute are copied to any disk being initialized as a system disk.
- 2: Write Protect. Files with this attribute cannot be opened for output or for update, nor can they be deleted or renamed.
- 3: Reserved.
- 4-6: J, K, and L. User defined attributes.

7: Format. Files with this attribute are treated as though they are write protected. Some of the system files have this attribute. It should not be given to other files.

Attributes can be written with the ATTRIB command or the ATTRIB system call.

EOF COUNT
: contains the number of the last byte in the last data block of the file minus 1. If the value of this field is 80H, for example, the last byte in the file is byte number 129 in the last data block.

NUMBER OF DATA BLOCKS
: is a two-byte variable indicating the number of data blocks currently used by the file. To calculate the current number of bytes in the file, use the following formula:

(NUMBER OF DATA BLOCKS) * 256 + EOF COUNT + 1

HEADER BLOCK POINTER
: is the address of the file's header block. The low order byte in this field is the sector number, and the high order byte is the track number. The system finds a disk file by searching the directory for the name, then using the header block pointer to seek the beginning of the file.

ISIS.FRE

This file contains a bit map of the disk, with each bit position representing one cluster of the disk. A cluster is 4 blocks or 4 sectors of the disk. Since there are 32 sectors/track, each byte of the bit map represents one track on the disk. For diskettes, the bit map is 80 bytes long and for bubble memory the bit map is 16 bytes long.

If a bit in the bit map is 1, the corresponding cluster is allocated, that is, in use as a pointer block and/or as data blocks. If a bit in the bit map is 0, the corresponding cluster is free space on the disk. When a file is deleted, the bits that correspond to the clusters it previously occupied are reset to 0.

Table 8-5 shows the values of the bit maps for the system files located at tracks 00H, 01H, and 27H on the mini-diskette and tracks 0 and 1 on the bubble memory.

Table 8-5 Values of System File Bit Maps

	Track 00H	Track 01H	Trace 27H
Mini-diskette	0FFH ISIS.T0	01H ISIS.LAB	01FH ISIS.FRE ISIS.DIR
Bubble Memory	0F3H ISIS.T0 ISIS.DIR ISIS.FRE	01H ISIS.LAB	

ISIS uses a pre-allocation scheme to allocate disk space to requesting programs. A pre-allocation table is maintained in memory containing a list of available clusters. These clusters have already been set on in the bit map, so they will not be allocated by any other program.

If a program requests disk space and there are no clusters available in the pre-allocation table, ISIS gets 5 clusters and sets the bit map in ISIS.FRE for these 5 clusters. One of these clusters is made available to the requesting program. The other four are saved in memory in a pre-allocation table for future requests.

This technique saves time since ISIS only has to access the disk one time to allocate 5 clusters. Thus, if the system is reset before clusters 2-5 are used, they will remain allocated but will never be usable again.

Disk File Structure Summary

Figure 8-17 provides an overall view of the most important elements in the file structure. Some simplifications have been made for clarity. There are only four directory blocks, pointer blocks contain only four data block addresses, and so on. However, the key relationships of file elements are shown.

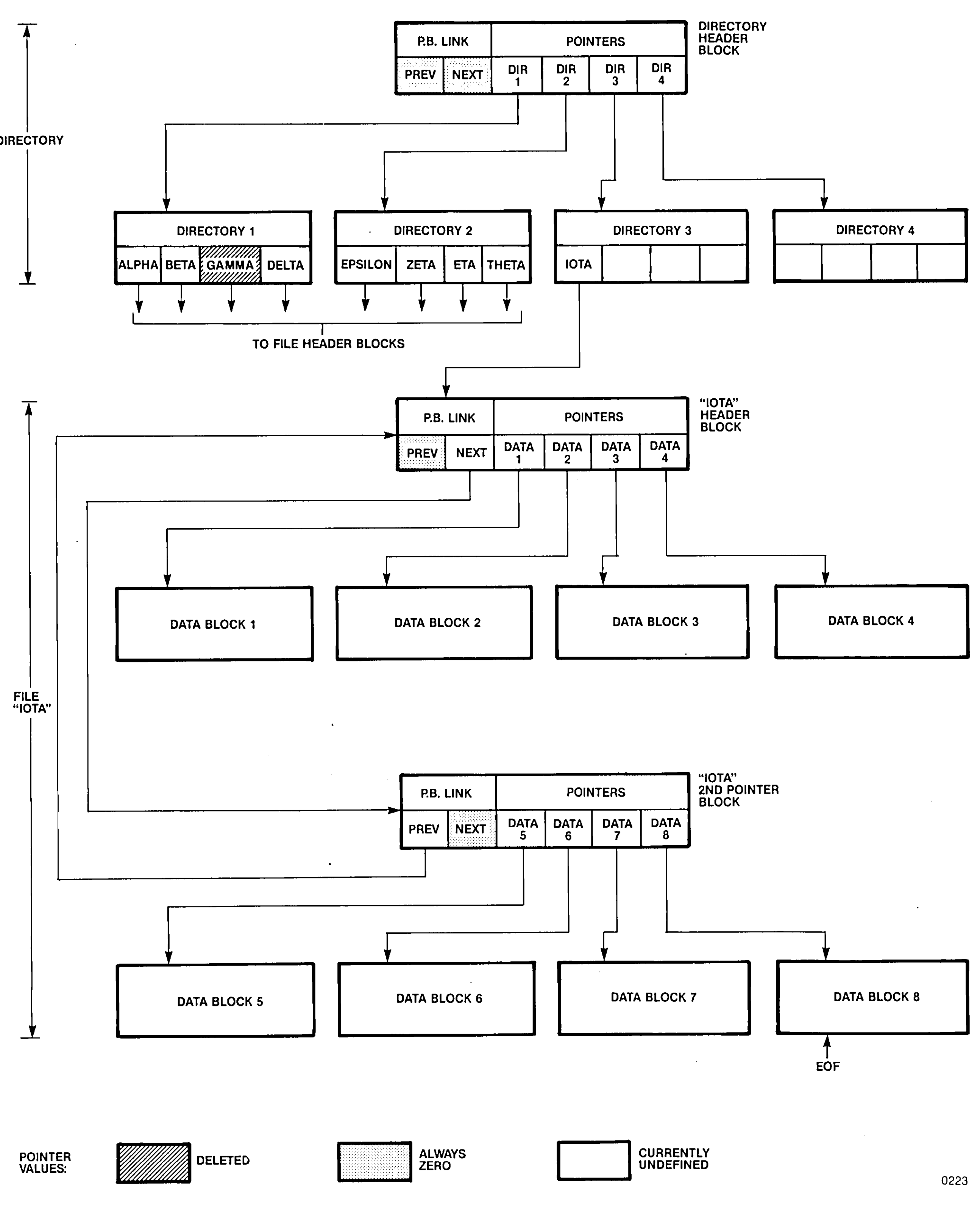


Figure 8-17 Disk File Structure Summary

CHAPTER 9
DUAL PROCESSING

Introduction

Dual processing is a capability which increases the processing power of the iPDS system. It requires adding an optional processor board to the system. Controlled by software that is already included in the ISIS-PDS operating system, a different program can be run on each processor at the same time providing greater processing throughput. More jobs can be run during the same time on a dual processor system than on a single processor system.

For example, at the same time one processor is compiling one file of source code, the other processor can be used to edit another file. The result is an increase in throughput with more jobs processed during the same time period.

The processor supplied with the system is referred to as the base processor or Processor A. The add-on processor is referred to as the optional processor or Processor B.

Both the base processor and the optional processor are functionally equivalent 8085 CPUs each with 64K bytes of dynamic RAM and an additional 2K bytes of bootstrap ROM. Both processors share the keyboard, the CRT display unit, the disk drives, and the multimodules. The built-in serial and parallel interfaces are available only to the base processor. If serial and parallel I/O ports are required for the optional processor, they can be added through multimodules.

Operating a Dual Processing System

Operating a dual processing system is similar in most respects to operating a system with only the base processor. Each processor runs the ISIS-PDS operating system and ISIS programs using its own 64K byte memory space, independent of the other processor.

Programs are run the same on the base processor as on the optional processor with few exceptions. Commands that use the :LP:, :SO:, or :SI: devices cannot be run on the optional processor. These I/O ports can only be accessed from Processor A. For example, the SERIAL command returns an error if run on Processor B since the serial port is only part of Processor A.

Once the keyboard is assigned to a given processor, commands can be entered to and run on that processor as described in previous chapters of this manual. The keyboard is initially assigned to Processor A (the base processor). The special function, FUNCT-HOME, is used to switch the keyboard between the two processors.

Special operating procedures that apply only to dual processing systems fall into the following categories:

- Initializing the system
- Sharing the keyboard
- Sharing the CRT display

- Sharing disk drives
- Sharing multimodules
- Sharing files

Each category is described in the following sections.

The function key (FUNCT) is used for many of the special operations required in dual processing systems. It is used with other keys to perform a special function. For example, FUNCT used with the up arrow key is used to increase the screen size for the processor to which the keyboard is currently assigned. To perform a special function, hold down the FUNCT key while pressing the other key and then release both keys. This operation is similar to the way that the SHIFT key is used in typing.

Sharing the Keyboard

The single keyboard on the system is assigned to only one processor at a time. Initially, it is assigned to Processor A. It can be switched to Processor B using the special function, FUNCT-HOME. Subsequent use of FUNCT-HOME alternates the keyboard assignment between Processor A and Processor B.

Commands are entered at the keyboard as described in previous chapters of this manual and are run on the processor currently controlling the keyboard. Command lines entered at the keyboard are echoed on the lower part of the display screen. The prompt characters on the display screen indicate the current processor to which the keyboard is assigned.

The prompt:

```
An>
```

is for Processor A while the prompt:

```
Bn>
```

is for Processor B. The letter n represents the drive number of the physical disk device currently assigned to the :F0: logical disk device, the system drive. Initially, n can be either 0 if the system is initialized from disk or 4 if the system is initialized from bubble memory. Logical and physical devices are discussed in Chapter 5.

The FUNCT-R combination, allows the processor currently assigned the keyboard to be reset independently of the other processor. The operating system is loaded from the physical drive currently assigned to :F0: to the memory of the processor currently controlling the keyboard when FUNCT-R is typed.

Several keys are interpreted and processed by the CRT/Keyboard controller rather than by one of the two processors. These keys, such as CTRL-S, FUNCT-S, FUNCT-T, will affect the keyboard and display for both processors even if entered when the keyboard is assigned to only one processor.

Sharing the CRT Display

The iPDS display screen is divided between the two processors as shown in figure 9-1. The bottom part of the screen is assigned to the processor which currently

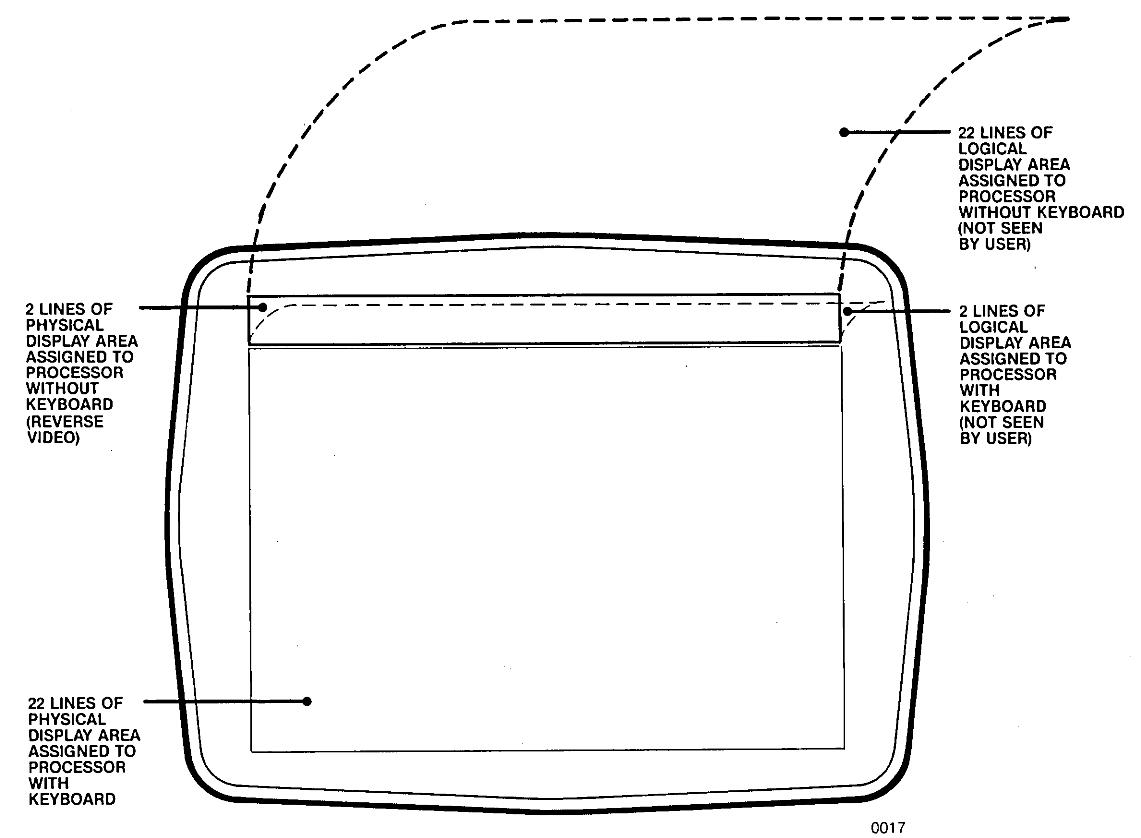
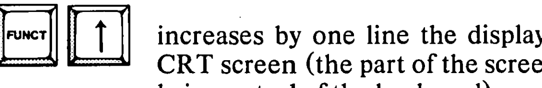


Figure 9-1 Split Screen Display

controls the keyboard. The top part of the screen, displayed on the screen in reverse video, is assigned to the other processor. The two parts of the screen are switched each time FUNCT-HOME is used to change the keyboard assignment.

Each processor has a full 24 lines of logical display. However, less than 24 lines may actually appear on the physical screen. The number of lines appearing on the screen for each processor is controlled by the user through special function keys.



increases by one line the display size of the bottom part of the CRT screen (the part of the screen used by the processor currently in control of the keyboard).

FUNCT ↓ decreases by one line the display size of the bottom part of the CRT screen (the part of the screen used by the processor currently in control of the keyboard).

When the system is initialized, the bottom portion of the screen contains 22 lines, and the top portion of the screen contains 2 lines.

The top two drawings in figure 9-2 represent the 24 logical lines of display output for each processor. The portion of each output enclosed in dashed lines will actually appear on the CRT display screen, if Processor A controls the keyboard and 22 lines are assigned to it. This is shown in the bottom left drawing. The bottom two drawings illustrate the actual lines displayed when Processor A controls the keyboard and when Processor B controls the keyboard and 22 lines are assigned to the processor controlling the keyboard.

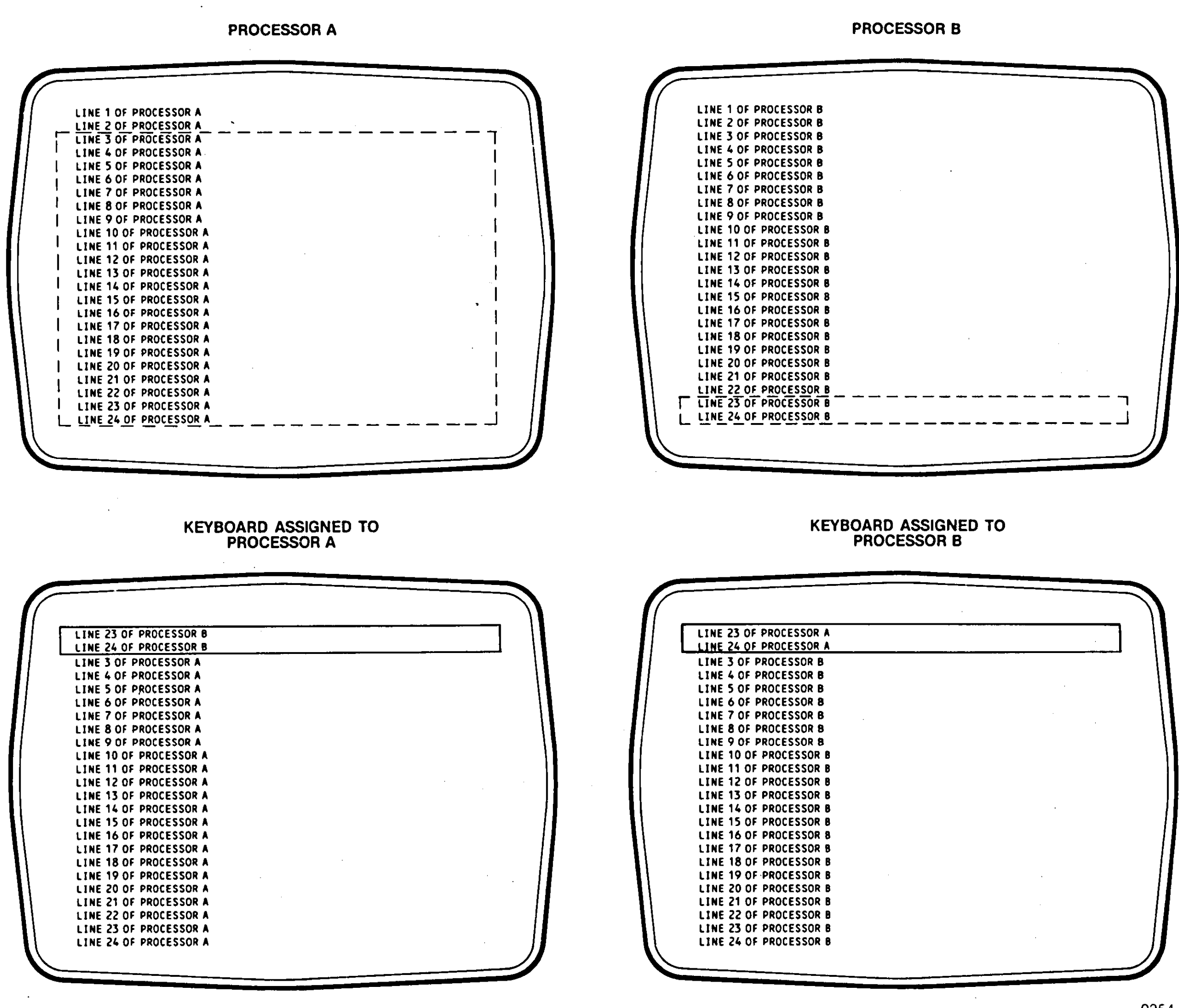


Figure 9-2 Logical and Physical Screens

Commands that are entered through the keyboard are echoed on the bottom part of the screen and are run on the processor to which the keyboard is assigned. The operating system prompt characters indicate which processor will receive the keyboard input: Processor A or Processor B.

The cursor character appears as a reverse video prompt on the lower part of the screen and indicates the position where the next character will appear. The cursor is invisible on the reverse video part of the screen.

Sharing Disk Drives

Disk drive sharing is transparent to the user operating a dual processing system. Only one processor can access a drive at a time. The internal operating system software uses a semaphore to enforce the access rights of the processor.

Even though operating procedures are not affected by the sharing of disks, the user can experience a slowdown in operations if both processors are constantly using the same drive. One processor must wait while the other processor completes its disk access. The drive contention between the two processors also results in undesirable head thrashing on the drive itself.

Therefore, it is recommended that dual processing systems use at least two disk drives, one for each processor. The ASSIGN command described in Chapter 5 is used to assign a different drive as the system device for each processor. Each processor can still access any physical device by specifying the logical device name that is currently assigned to it. However, the physical device to which :F0: is assigned is always used as the default for implicit disk requests. Thus, assigning :F0: to different physical devices for each processor can greatly reduce contention between the two processors for the drives.

Sharing Multimodules

The four connectors on the multimodule adapter board are labelled J1, J2, J3, and J4. The technique used to share the four Multimodules is to group two connectors into a single multimodule row. Row 0 contains connectors J1 and J2, while Row 1 contains connectors J3 and J4. Each row can then be assigned to only one processor at a time. While one processor accesses one row, the other processor can access the other row. However, both rows can also be assigned to the same processor.

For a processor to access a multimodule, the multimodule row must be attached to the processor with the ATTACH command described in Chapter 5. (The ATTACH system call described in Chapter 8 achieves the same result within a program.) Attaching a row sets the semaphore associated with the row, so the processor can access either multimodule on the row without interference from the other processor and without having to set and clear the semaphore on each access.

When the processor is through with the multimodule row, the row should be detached with the DETACH command described in Chapter 5 to clear the semaphore. (The DETACH system call described in Chapter 8 achieves the same result within a program.)

The ATTACH command assigns the specified row to the processor currently in control of the keyboard. The DETACH command removes the row from the processor currently in control of the keyboard. Both commands are entered from the keyboard. The corresponding system calls are executed from an assembly language or PL/M program.

NOTE

The bubble memory multimodule is treated as a disk device and does not need to be attached or detached before it is accessed. An attempt to attach or detach bubble memory generates an error message. See the previous section on the sharing of disk devices for more information.

Sharing Files

Only one processor can open a file for write or update at any one time with the exceptions of the byte bucket, :BB:, and the console assignments :CI:, and :CO:. This restriction is enforced by the operating system and is transparent to the user. A

non-fatal error message is issued by the operating system if the user attempts to write to the same file from both processors at the same time. Both processors can read from the same file at the same time.

Temporary Files

Some of the programs that use temporary work files allow the user to specify the names for these files. If no names are given, the program uses a set of default names. In this case, the user can avoid any conflicts by letting the program assign unique names to the work files.

However, the restriction on writing files does not limit the same program from running on both processors at the same time, even in the case of programs that create temporary files. If a program creates a temporary file, the temporary file is assigned an extension of .TMA if it is created by a program running on Processor A or .TMB if it is created by a program running on Processor B. Thus, each temporary file has a unique filename and does not conflict with the temporary files created by the same program running on the other processor.

Because of the way the operating system assigns names for temporary files, the user must be careful when creating files with .TMP-like extensions.

- If a file is created by a program running on Processor A with a .TMP extension, it will actually appear with a .TMA extension. On Processor B, the file will appear with a .TMB extension.
- A file that is renamed from Processor A with a .TMP extension will actually appear with a .TMA extension. On Processor B, the extension will appear .TMB.
- Further, a .TMA file cannot be renamed to an extension of .TMP from Processor A. The RENAME command will attempt to delete the .TMA file. If deleted, the file can no longer be renamed. If a .TMA file is renamed to a .TMP file from Processor B, it will actually be renamed to a .TMB file.
- A .TMB file cannot be renamed to an extension of .TMP from Processor B. The RENAME command will attempt to delete the .TMB file. If deleted, the file can no longer be renamed. If a .TMB file is renamed to a .TMP file from Processor A, it will actually be renamed to a .TMA file.
- If a .TMP extension is specified for a file to be deleted from Processor A, the same file with the .TMA extension is actually deleted. On Processor B, the file with a .TMB extension is actually deleted.

A file with a .TMB extension can be deleted from Processor A by specifying the filename and the extension .TMB or .TMP. A file with a .TMA extension can be deleted from Processor B by specifying the filename and a .TMA or .TMP.

Data Files

The same data file cannot be written by both processors at the same time. For example, if the attempt is made to write to the same destination file from both processors, one processor will open the file and write to it while the other processor waits. Then, when the first processor closes the file, the other processor will be able to access the file and write to it overwriting the previous version.

When copying or renaming files from both processors or when performing other operations that involve writing to files from both processors, consideration should be given to the effects of writing to the same file from both processors.

Additionally, when using some ISIS commands like COPY and RENAME, the command accesses the directory. Thus, the other processor must wait until the entire command is completed and the directory is released before it can perform its COPY or RENAME on the same disk device.

Initializing the System

The operating system is contained in several files stored on disk or in bubble memory. These files must be loaded into the development system's memory in a process called initialization. During initialization, either the internal disk drive (drive 0) or the bubble memory device (drive 4) contains the operating system files. The device from which the system files are loaded is called the system drive.

In a dual processing system, both processors can be initialized from the same system drive, or one can be initialized from drive 0 while the other is initialized from drive 4.

The hardware reset (RESET key) initializes both processors. The software reset (FUNCT-R) resets only the processor to which the keyboard is currently assigned. This processor is reset from its current system default drive and no assignments are changed.

To reset both processors from the internal disk drive, follow the initialization procedure given in Chapter 3 for initializing Processor A from the internal disk drive. Then, switch to the optional processor by pressing the FUNCT-HOME key. Repeat the same procedure to complete the initialization for Processor B.

To reset both processors from the bubble memory multimodule, follow the procedure given in Chapter 3 for initializing Processor A from bubble memory. Then, switch to Processor B with the FUNCT-HOME key and repeat the procedure to complete the initialization for Processor B.

To reset the two processors from different devices, start the procedure given in Chapter 3 for initializing Processor A from bubble memory. When the BOOT FROM BUBBLE? prompt appears, do one of the following steps:

- To initialize Processor A from bubble memory and Processor B from the disk in drive 0, answer the BOOT FROM BUBBLE? prompt on Processor A with a Y. Switch to Processor B with FUNCT-HOME. Place a system disk in drive 0 and answer N to the BOOT FROM BUBBLE? prompt that appears on Processor B.

- To initialize Processor B from bubble memory and Processor A from the disk in drive 0, insert a system disk in drive 0 and answer the BOOT FROM BUBBLE? prompt on Processor A with an N. Switch to Processor B with FUNCT-HOME. Remove the system disk in drive 0 and answer Y to the BOOT FROM BUBBLE? prompt that appears on Processor B.

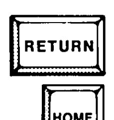
Sample Dual Processing Session

The following screens show some typical uses of an optional processor.

```
ISIS-PDS, V1.0
B1>

*OR

 -----
         EXTRN  ISIS↑
         EXTRN  CO↑
         EXTRN  CI↑
↑
         ORG    4000H↑
EXIT     EQU    9↑
↑
EBLK:    DW     ESTAT↑
ESTAT:   DS     2↑
↑
START:   MVI    B,1AH↑
↑
LOOP:    CALL   CI↑
↑
         MOV    C,A↑
         CALL   CO↑
↑
         CMP    B↑
```

Key-in Sequence

CREDIT PROGA.SRC RETURN



HOME

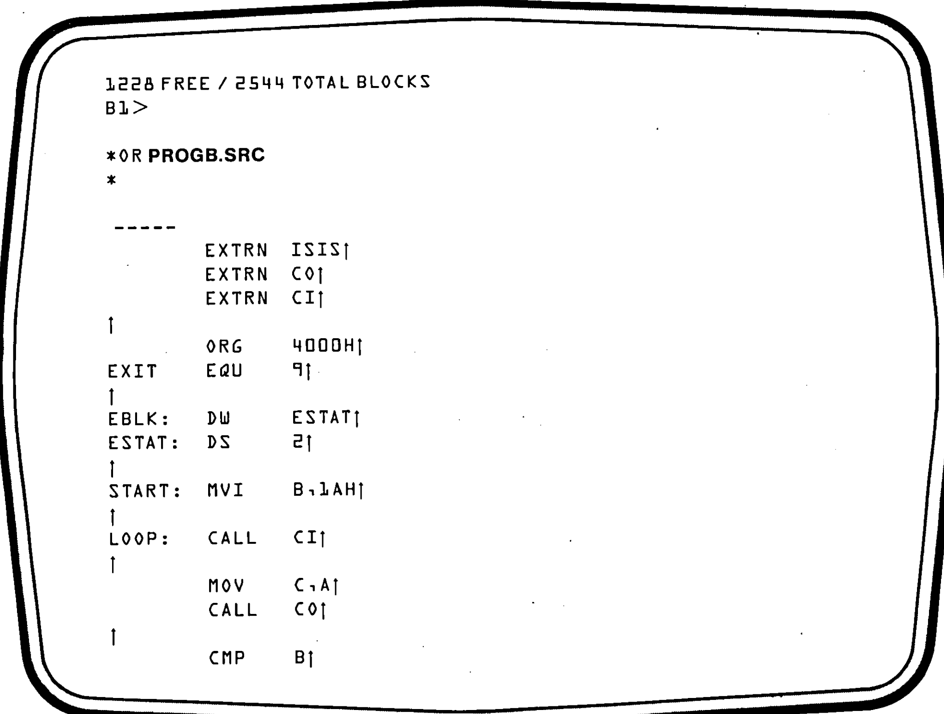
OR *SPACE*

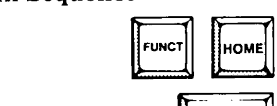
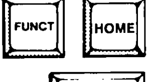
Comments

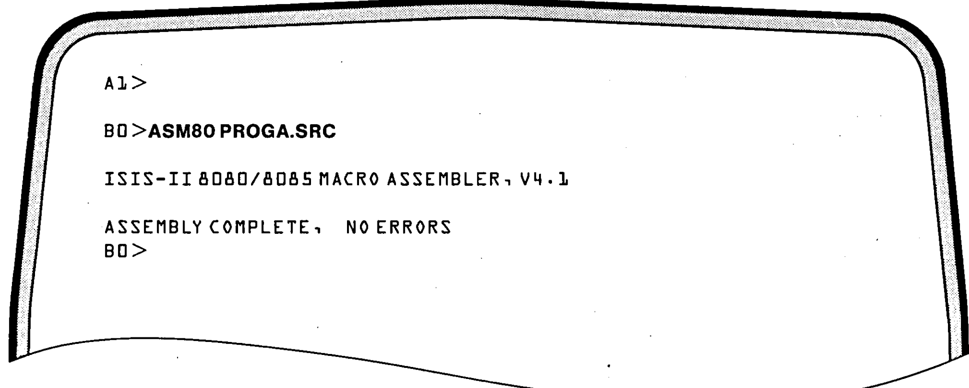
The file PROGA.SRC is being edited. A file on the disk contains a block of code to be added to PROGA.SRC, but the name of the file is unknown.

```
*OR

B1>DIR 0 FOR *.SRC
DIRECTORY OF :F0:LEARN.PDS
NAME    .EXT .BLK  LENGTH ATTR   NAME    .EXT  BLKS LENGTH ATTR
               S
PROGA   .SRC    8     937        PROGB   .SRC     4    388
                               12
1228 FREE / 2544 TOTAL BLOCKS
B1>
```

Key-in Sequence	Comments
DIR 0 FOR *.SRC RETURN	To find the name without exiting from CREDIT, switch to the other processor by pressing FUNCT-HOME, and run a DIR command on Processor B. The default system device for Processor B is drive 1 (B1>) while it is drive 0 for Processor A. Because of disk contention when both processors use the same default drive, it is recommended that dual processor system have at least two disk devices. The ASSIGN command should be used to set the system default device to different disks for each processor.



Key-in Sequence	Comments
  **PROGB.SRC** RETURN	Type FUNCT-HOME again to switch back to Processor A and finish entering the OR command. Exit from the editor with the EX command.

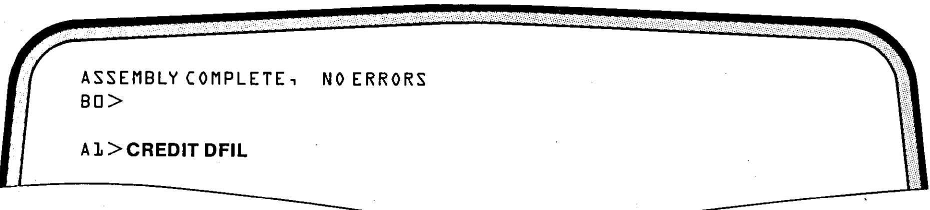


Key-in Sequence

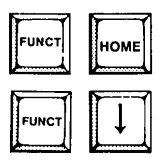
ASM80 PROGA.SRC RETURN

Comments

Another application for dual processor systems is to start an assembly on one processor, and switch to the other processor to edit a module. Here a new example is started where the system device for Processor A is drive 1 and for Processor B is drive 0.



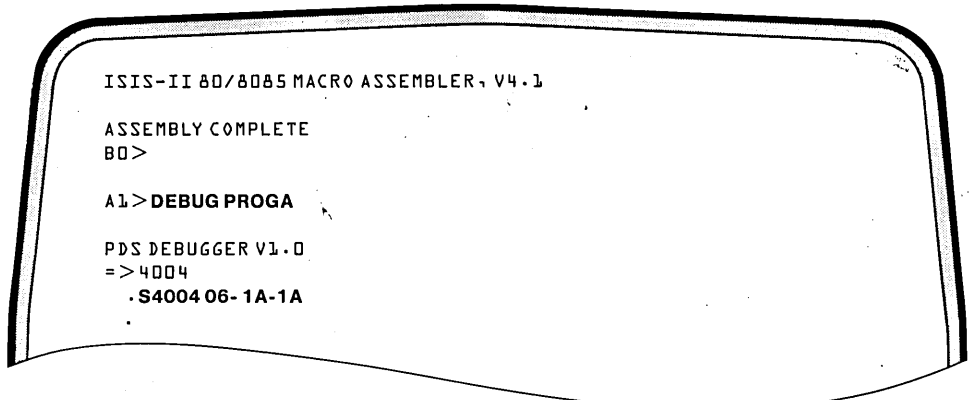
Key-in Sequence



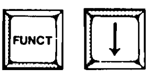
CREDIT DFIL RETURN

Comments

Switch to Processor A; use the FUNCT- ↓ to display several lines from Processor B so the assembly output can be monitored. Enter the CREDIT command to monitor a module on Processor A.



Key-in Sequence

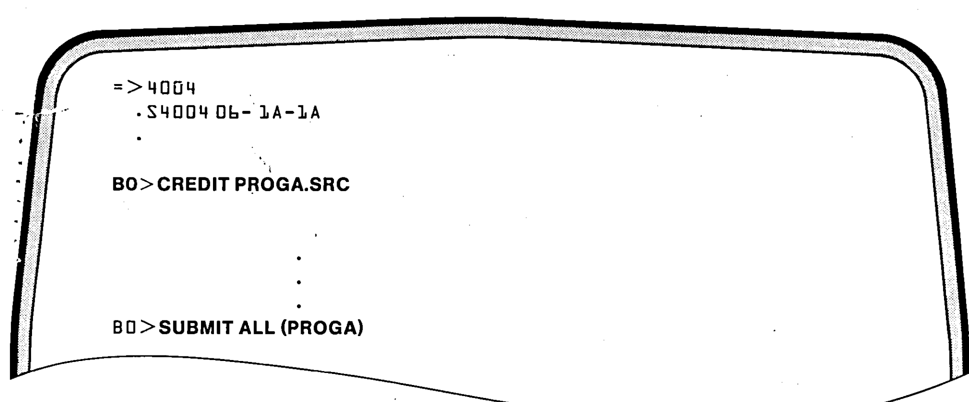


DEBUG PROGA

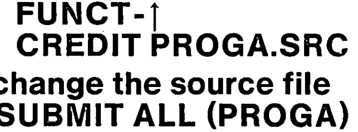
S4004 *SPACE SPACE* **1A**

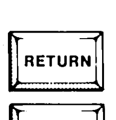
Comments

PROGA is being debugged on Processor A when an error is discovered. This error can be quickly corrected by patching memory within DEBUG. However,the file containing the program is not changed. With a dual processor system, the original source program stored on a different drive (drive 0) can be corrected on the other processor without exiting from DEBUG.



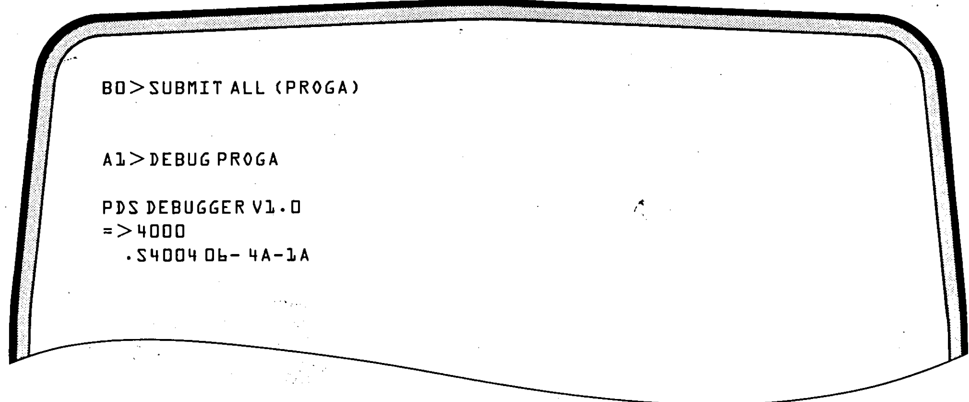
Key-in Sequence

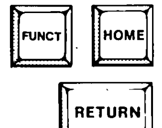


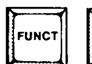


Comments

On Processor B, change the source program with CREDIT and then run a SUBMIT file named ALL that reassembles, links, and locates the source file.



Key-in Sequence	Comments
	Return to Processor a and continue debugging the object code that was patched in memory while the assembly, link, and locate run on Processor B.

Key-in Sequence	Comments
	While on Processor A, press FUNCT-R to reset Processor A without disturbing the programs running on Processor B.

Programming on a Dual Processing System

When programming on a dual processing system, special consideration must be given to the techniques used to share certain system resources (the keyboard, the CRT display, disk drives, and multimodules). The techniques used by the operating system to share the system resources are described in this section.

A program can determine which processor it is executing on by doing a RIM instruction in assembly language (or an R$MASK instruction in PL/M-80). The RIM instruction reads eight bits of data. The Most Significant Bit (MSB) of the data read is from the SID pin of the MCS-85 processor chip. If the MSB is not set (i.e., is zero), the program is running on Processor A. If the MSB is set (i.e., is one), the program is running on Processor B.

Shared Resources

The sharing of resources between the two processors must be handled with care to prevent collisions where both processors are trying to use the same peripheral for different jobs at the same time. Collisions can result in loss of data and processing time. The the ISIS-PDS operating system provides built in mechanisms to prevent collisions.

The keyboard is assigned to either the base processor or the optional processor at a given time.

The CRT display unit is shared between the two processors through a split screen mode. The screen is split horizontally into two sections with the top section displaying output from the processor without a keyboard and the bottom section displaying output from the processor with the keyboard. The number of lines allocated to each processor can be changed by the user dynamically from 0 to 24 lines.

No matter how many lines are allocated to each processor, the processor itself believes it has a full 24 lines of display. Lines that do not appear on the screen are stored in memory.

The two processors use a semaphore to share the disk drives and multimodules.

Semaphores

To ensure that only one processor uses a shared disk or multimodule at a time, there is a hardware semaphore associated with the disk and with each of the two multimodule rows (connectors J1/J2 and connectors J3/J4). Before trying to use a disk or multimodule, the processor checks the semaphore to see if the device is available. If the semaphore is 1, the device is available. The processor then clears the semaphore, uses the device, and resets the semaphore when it is finished.

If the device is busy (the semaphore is set to 0), the processor waits until the device is available. The checking of semaphores is done internally by the system calls involved and is transparent to the programmer using the system calls.

Shared Multimodules

The multimodule adapter board allows up to four multimodules to be added to the system. The four multimodule connectors are grouped into two rows of two multimodules each. See figure A-19 for the location of the multimodule connectors. Connectors J1 and J2 form row 0 and connectors J3 and J4 form row 1. Each row is associated with a semaphore flag as discussed above, so the processor accesses both devices on a row by attaching the row. The ATTACH system call is used within a program to set the semaphore. The other row may then be accessed by the other processor. When finished, the processor must detach the row to clear the semaphore with the DETACH system call within the program.

A single bubble memory multimodule takes the space of both connectors on a row; so each row can support only one bubble memory multimodule or two non-bubble multimodules. Bubble memory multimodules contain files just as disk drives and are subject to the mechanism controlling access to files.

Shared Files

To ensure that only one processor can open a given disk or bubble memory file at a given time for a write, a list of all the files currently open is maintained by the Keyboard/CRT controller. Processor A and Processor B can only open a file after getting permission from the controller.

For example, the controller would not give permission to Processor A to open a file for reading if Processor B had the same file open for write operations. A file could only be opened for write operations if the other processor did not have the file open at all. Both processors can open the same file for reading at the same time.

When a processor closes a file, it notifies the Keyboard/CRT controller so the file can be removed from the open file list.

All interactions with the Keyboard/CRT controller are handled by the ISIS-PDS system calls transparently to the systems programmer using the system calls.

CHAPTER 10
PROM PROGRAMMING

Firmware Development

Microcomputer based systems are a central element in the design of electronic products. A typical development effort for a system consists of repeated design cycles:

- First, the design is implemented in a hardware prototype
- Then, the prototype is tested to determine if it meets the needs of the application

Any failures during testing lead to repetitions of the cycle until the performance of the product is adequate.

Development cycles for microcomputer systems are characterized by the development of software—both data and programs. See figure 10-1. Programs are sequences of coded machine instructions held in the microcomputer memory. The machine executes the instructions to fulfill the needs of the particular application. The application is developed by integrating the hardware and the software.

Once the software is perfected, it is often installed permanently into a type of Read Only Memory (ROM). Software or data frozen in ROM is referred to as firmware. Firmware is used in systems design because of its relative low cost, high speed, and data non-volatility (firmware is retained even when power to the system is turned off).

Intel provides a number of ROM memory components for the firmware development effort including two general types of ROM: mask programmable and electrically programmable.

Firmware for mask programmable ROM is fabricated into the ROM during manufacture. This is often the cost effective method for including firmware in a mass produced product.

Programmable ROM (PROM) devices allow the firmware to be electrically programmed when the device is used rather than when the device is manufactured. Electrical programming is accomplished by specifying a particular address with data and then applying the required voltage to the appropriate pin.

There are three kinds of PROM: non-erasable bipolar PROMs, ultraviolet Erasable PROMs (EPROMs), and Electrically Erasable PROMs (E^2PROMS). Bipolar PROMs can be electrically programmed only once and, thereafter, permanently retain their data. However, EPROMs can be erased by exposure to ultraviolet light. E^2PROMs are electrically erased similar to the way they are electrically programmed.

EPROMs and E^2PROMs allow flexibility during the firmware development cycle because they can be repeatedly erased and reprogrammed. Some components, such as the 8751, have built-in EPROM in addition to other logic.

For design convenience, Intel components containing EPROM have pin-compatible counterparts containing mask programmable ROM instead of EPROM. This allows the re-programmable EPROM version to be used during prototype development and the masked ROM version to be used for the final mass production.

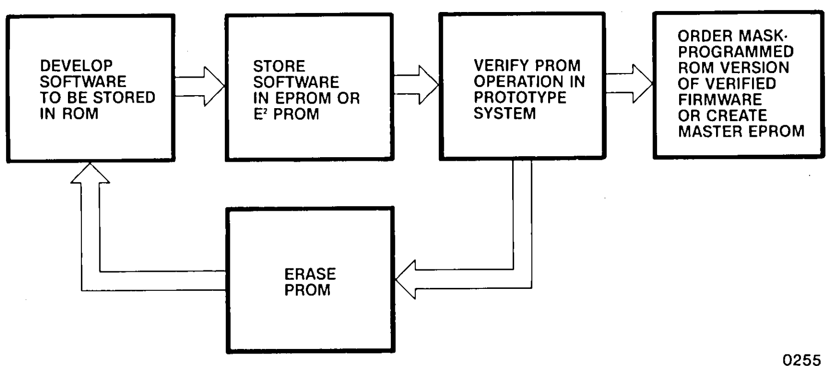


Figure 10-1 Firmware Development Cycle

Eprom Erasure

In some cases, sections of non-blank EPROMs are programmed; however, EPROMs to be programmed are normally in their blank (erased) state.

EPROMs are erased by exposing the integrated circuit to ultraviolet light through a window provided on the chip package. Erasure occurs when the exposure light has a wavelength shorter than approximately 4000 Angstroms.

Sunlight and certain types of florescent lamps have wavelengths in the 3000-4000 Angstrom range. Constant exposure to room level florescent lighting could erase the typical EPROM in about 3 years, while it would take approximately 1 week to erase when exposed to direct sunlight.

If the EPROM device is exposed to these lighting conditions for extended periods of time, the device window should be covered with an opaque label (available from Intel) to prevent unintentional erasure.

The optimum light for erasing EPROMs has a wavelength of 2537 Angstroms. The integrated dose (UV intensity X exposure time) for erasure should be a minimum of 15 W-sec/cm^2. The erasure time is approximately 15 to 20 minutes using an ultraviolet lamp with 12,000 uW/cm^2 power rating. The EPROM should be within 1 inch of the lamp tubes during erasure.

The EPROM should not be powered up during erasure. If the EPROM device is powered up during erasure, the internal current paths effectively cancel the energy being provided by the UV light, and the device is not erased.

Consult the section on "PROM and ROM Programming Instructions" in the *Intel Component Data Catalog* for further information on erasing EPROM components. Information on erasing E^{2}PROMs can also be found in the catalog. Individual device specifications contain further erasure information.

Overview of Prom Programming on the System

The Intel Personal Development System supports programming of E²PROMs, EPROMs, and other circuits that have built-in EPROM or E²PROM through the use of the following hardware and software:

- Personality Module
- Plug-in Module Adapter Board
- Intel PROM Programmer Software (iPPS)

Personality Modules

Personality Modules are small units that are installed in the side of the iPDS system. They plug into a connector on the Plug-in Module Adapter board and allow the development system to program a specific family of PROM devices.

Some PROM families consist of a single unique PROM while others comprise many specific PROMs. Consult the current *Intel Component Data Catalog* or *Intel Systems Data Catalog* for further information on the Personality Module required for specific PROMs. To program a given PROM device, the appropriate Personality Module must be installed.

Each Personality Module is shipped with a User's Guide which contains detailed information on that particular module. The information covered includes the following:

- PROM devices supported by the Personality Module
- Installation of PROM devices in the module

Plug-In Module Adapter Board

The Plug-in Module Adapter Board is used for both PROM Personality Modules and Emulators. It provides the hardware interface between the Plug-in Modules and the rest of the system.

iPPS SOFTWARE

The iPPS software is a utility that runs under the ISIS-PDS Operating System. The iPPS software provides a foundation for programming and verifying all Intel components that contain EPROM and E²PROM. A variety of these components exist including memory components and other integrated circuits that have built-in PROM.

The iPPS commands control the operation of the Personality Module installed in the development system. They provide the following capabilities needed to program PROMs:

- Reading and writing data to and from disk files
- Mapping data for a particular PROM word size
- Modifying the data in the memory Buffer

- Interleaving data for different addressing schemes
- Programming the contents of a particular PROM device
- Reading back the contents of the PROM device
- Verifying the contents of the PROM device by comparing it to the contents of the memory buffer

In general, the steps for programming a PROM device are:

1. Produce a file containing the data or the program (object code version) to be stored in the PROM device.
2. Install the appropriate Personality Module for the PROM device to be programmed.
3. Install the PROM device in the Personality Module.
4. Run the iPPS software under the ISIS-PDS operating system.
5. Use the iPPS commands necessary to transfer the data or program to the PROM device.

PROM Programming Subsystem

The hardware and software used to program PROMs on the iPDS system can be viewed as a iPDS subsystem for PROM programming. Figure 10-2 illustrates the system data flow and logical devices in this subsystem. All data transfers are handled by the iPPS software and the firmware uploaded from the Personality Module board.

The three major system logical devices are shown with arrows indicating the directions of data flow between these devices. The development system contains the Buffer and File devices while the Personality Module contains the PROM device.

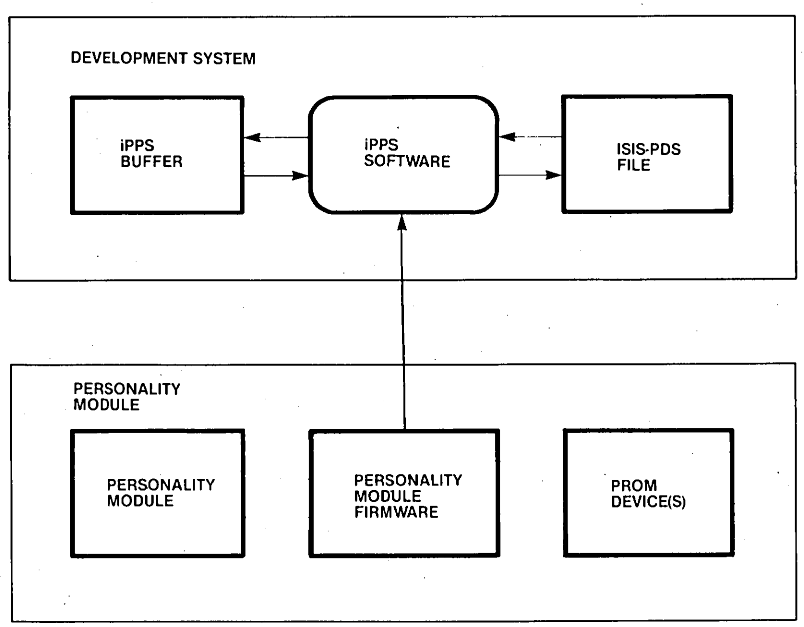


Figure 10-2 PROM Programming Subsystem

iPPS Software

This section covers the operation of the Intel PROM Programming Software (iPPS). This software runs as a command under the ISIS-PDS operating system. The iPPS provides a set of subcommands, called iPPS commands, to program PROM devices.

iPPS Initialization

The iPPS software can be initialized from a command line entered at the keyboard or through the SUBMIT job facility.

The following files are necessary for normal iPPS operation and should be on a currently accessible disk:

- IPPS is the main command file that is loaded and executed by the ISIS-PDS operating system.
- IPPS.ERR is the error message file containing strings for the iPPS error messages.
- IPPS.HLP is the help message file containing text explanations of the iPPS commands.
- IPPS.OV0 is an overlay file containing part of the iPPS software that is loaded during normal operations.
- IPPS.OV1 is an overlay file containing part of the iPPS software that is loaded during normal operations.
- IPPS.OV2 is an overlay file containing part of the iPPS software that is loaded during normal operations.

Once initialized, a number of subcommands called iPPS commands are available to program PROMS. Some examples of iPPS commands are the TYPE command, the WORKFILES command, the FORMAT command, the COPY command, and the EXIT command. For IPPS command descriptions see the *iUP-200/201 Universal Programmer User's Guide*, order number 162613.

Command Line Invocation

The iPPS software is invoked as a command under ISIS-PDS with the following format:

[:F<n>:]IPPS

where :F<n>: is used to specify the drive on which the iPPS files are located. For example, if the iPPS files are located on disk device 1 then the iPPS software would be invoked by entering the command:

:F1:IPPS

followed by the RETURN key.

The notational conventions used to describe commands are covered in Chapter 5 and in the section "Notation for Format Descriptions" in the iUP-200/201 Universal Programmer User's Guide.

The iPPS software expects all IPPS files to be on the same drive. For example, if iPPS is run from drive 1, it expects IPPS.ERR and IPPS.HLP to be on drive 1.

If the drive containing the iPPS files is the system default drive, the command line is:

```
IPPS
```

loads and executes the iPPS program from the current system default drive.

After the IPPS command is keyed in, the iPPS software is loaded and executed. The following sign-on message is displayed:

```
INTEL PROM PROGRAMMING SOFTWARE (PDS) Vn.m
INTEL COPYWRIGHT 1980,1981,1982,1983
PPS>
```

Vn.m is the current version (n) and revision (m) of the software. PPS> is the main iPPS prompt and indicates that the iPPS software is ready to receive commands. The EXIT command returns control to ISIS-PDS.

Invocation Via a SUBMIT File

The iPPS software can also run under the control of a SUBMIT file. SUBMIT is an ISIS-PDS command that allows a disk text file to be used as input for further ISIS commands or as command inputs to utilities running under ISIS. Thus, a SUBMIT file can contain the ISIS command line to invoke iPPS and then a sequence of commands for iPPS itself.

The SUBSTITUTE and ALTER commands cannot be run from a SUBMIT file since they require extensive interactive input. The iPPS software recognizes comments in a SUBMIT control file as any characters following a semicolon (;). For more information on the SUBMIT command, consult Chapter 5 of this manual.

iPPS General Operation

This section describes the major functions of iPPS commands, the logical devices recognized by iPPS, and the operation of iPPS commands.

Major Functions

The purpose of iPPS is to program PROMs on the Personal Development System. iPPS recognizes a compact set of commands for doing this. The commands operate on three major storage devices: PROM, Buffer, or File.

The major functions of iPPS are:

- To read and write data on any of the three major storage devices (PROM, Buffer, or File)
- To modify data in the Buffer
- To display or print the data stored in any of the major storage devices on the console or list on the printer

- To check for an unprogrammed PROM and to allow the data in the Buffer to be overlayed with the bits in the PROM device. This determines whether the PROM can be programmed even though it is not blank.

- To format data for different programmable devices (ie., selecting different PROM word sizes and using interleaving techniques)

- To accommodate the following Intel absolute file formats during any operations that require files: 8080 Hex ASCII, 8080 Absolute Object, 8086 Hex ASCII, 8086 Absolute Object, and 286 Absolute Object. Intel assemblers and compilers produce files in these formats. The 8080 Absolute Object format is described in Chapter 8. The 8051 assembler produces files in 8080 Absolute Object format.

iPPS Storage Devices

iPPS transfers data between any two of the three logical storage devices: PROM, Buffer, or File. The data flow relationship among these logical devices is illustrated in figure 10-2. These devices are defined in the following sections.

PROM Device The PROM device is plugged into a socket on the Personality Module.

During iPPS operations, the size of the PROM device varies based on the most recent TYPE command. The default address boundaries for the PROM device are 0 to the (PROM size - 1). The PROM device is not recognized by iPPS until a TYPE command is issued. The TYPE command sets the appropriate size according to the type of PROM device specified. See the TYPE command description for further details.

Buffer Device The Buffer device is a section of development system memory (created and maintained by the iPPS software). It provides a temporary area where data can be held and modified. Its boundaries can exist anywhere in a virtual address range from 0 to 16777215 (0 to 2^{24}-1).

When the iPPS software is initialized, the Buffer start address is set to 0 and the Buffer end address is set to 8K-1. This implies an initial Buffer size of 8K bytes (the default Buffer size when no PROM type is specified).

During subsequent iPPS operations, the size and boundaries can vary as determined by specific iPPS commands. The Buffer start address is determined by the most recent command that changed the lower boundary of the Buffer. The Buffer end address is determined by the following expression:

(Buffer Start address) + (Buffer size - 1)

The TYPE command affects both the size and location of the Buffer in important ways. For example, the TYPE command always resets the Buffer start address to 0. The most recent TYPE command determines the size of the Buffer. See the TYPE command description for further details.

The size of the Buffer is determined as follows:

- It is equal to the size of the PROM if the PROM type has been specified by the TYPE command.

- It is initialized to 8K bytes when iPPS is invoked.

The need for a virtual Buffer on user disk arises when the PROM size exceeds 8K bytes. Then, iPPS software creates a virtual Buffer area using temporary file space on disk. iPPS software allows the user to specify the device on which the virtual Buffer is created.

Two temporary work files are used to accomplish this virtual Buffer. Their file names are IPPS.BUF and IPPS.TMA. These temporary files are created on the disk device specified by the most recent WORKFILES command. If no previous WORKFILES command was issued and a temporary work file is required, iPPS displays the prompt:

```
WORKFILES (:FX:) ?, X=
```

This prompt requests a number identifying the disk device to use for temporary work files. During subsequent Buffer operations, iPPS automatically swaps data in and out of development system memory to the work files on the specified device.

The work file device remains the same until the next WORKFILES command. See the WORKFILES command for more details. The temporary files are deleted either when the EXIT command is executed or when a new PROM type is selected that is less than 8K in size.

File Device The File device is a logical device containing data stored in an ISIS-PDS disk file. The File device is specified within iPPS commands by the ISIS-PDS file specification format:

```
:<device>:<filename>.<extension>
```

See Chapter 5 for more information on ISIS-PDS device and file specifications. The data is stored in the disk file in one of three Intel formats: 8080 Hexadecimal, 8080 Object, 8086 Hexadecimal, 8086 Object, or 286 Object. iPPS can read any of these formats as input. However, all iPPS commands that output to a File destination produce files of the 286 format.

The iPPS file device has address boundaries that exist in the range from 0 to 16777215 (0 to 2^{24}-1). These boundaries are determined as follows:

- The file's lowest address is the lowest address encountered while reading in the file.
- The file's highest address is the highest address encountered while reading in the file.

If the file was created by iPPS (i.e., it was a destination device in an iPPS command) these boundaries are determined by the specific command issued.

When the specified address range is not present in the file being read, the missing range is written in the PROM with the blank state of the currently selected PROM. If the destination device is the Buffer, those non-existent sections are not affected in the Buffer.

With commands that use the File device as a source, iPPS only reads the actual data from the file and ignores any other information in the file. The nature of the extra information depends on how the file was produced. For example, if the file was produced by a compiler, the file can contain special information used later for debugging.

Since the information is ignored when read by iPPS, if iPPS is then used to copy the data back to a file, the debug information is not in the newly written file. If the data is written back to the original file, the debug information is permanently lost.

Command Entry

iPPS commands consist of a command keyword to identify the command followed by other keywords and their associated parameters to make up the arguments of the command. Arguments are separated from each other by at least one space (extra spaces are ignored).

A command line is terminated by the RETURN key. No action is taken on the command until the RETURN key is detected unless the ESC key is pressed, in which case, the command is terminated. The command line is then verified for the correct format and executed if correct. If the command syntax (format) is illegal the following error message is displayed:

—SYNTAX ERROR—<specific error>.

If a required keyword is omitted, iPPS prompts for the keyword and its associated parameters. If the keyword is entered but its parameters are omitted, either a default value is assumed if applicable, or an error message is displayed if no default is applicable. Some commands also assume default keywords. All commands can be entered in either lower or upper case ASCII.

Input lines may be longer than the 80 character screen line. If the input line exceeds the screen display line, it automatically wraps around to the next screen line until 127 characters are entered or the RETURN key is pressed.

Commands that are longer than one input line (greater than 127 characters) can be continued in successive lines. The continued line must contain an ampersand (&) as the last non-blank character preceding the RETURN.

Command inputs are accumulated character-by-character in a line input buffer. The maximum continued command line is 255 characters (including ampersands, spaces, and RETURNs). The maximum non-continued line size is 127 characters (including the final RETURN).

iPPS keywords can be entered in their entirety or as any unique abbreviation (normally, only the first character is required). For example, command keywords of C, CO, COP, and COPY are all interpreted as the COPY command.

The command keyword can be entered without arguments, in which case iPPS prompts for further needed input; the command keyword and its first argument can be entered, in which case iPPS prompts for the remaining input arguments required for that command.

iPPS accepts numeric entries in any one of four number bases: Binary (Y), Octal (O or Q), Decimal (T), or Hexadecimal (H). Numbers can be entered in any of these bases by appending the appropriate single letter identifier to specify the base; for example, 11111111Y, 377Q, 255T, or FFH. An explicit number base identifier overrides the default number base (which is initially hexadecimal).

To illustrate these command entry features, the examples below show different ways in which the same COPY command can be entered.

```
COPY PROM(0,FF) TO BUFFER(0)

C P(0,11111111Y) TO B
; Here valid abbreviations are used for keywords.
; Binary is used for the number base.
;Note: The default offset value above is 0.

;Command entered by prompts FROM? and TO?:
C
FROM? PR(0,FF)
TO? BU

copy pr(0,7FF)
TO? b
```

Command Entry Editing

A new command in the line editing buffer can be edited using ISIS-PDS command line editing features as described in Chapter 5.

Form of iPPS Commands

Descriptions of notation, special iPPS format terms, the general command format, command switches, a functional summary of commands, each iPPS command, and sample PROM programming sessions are found in the *iUP-200/201 Universal Programmers Users Guide*, order number 162613.

NOTE

The iUP-200/201 Users Guide is intended for use with the Intel iUP-200/201 Universal Programmer to store programs and data into programmable read-only memory (PROM) devices. Disregard the sections of that manual referring to off-line operation, and URAM installation and operation.

APPENDIX A
INSTALLATION INSTRUCTIONS

Installation Considerations

The physical characteristics (width, height, depth, weight) of the system are given in table A-4. Be sure that the installation work area accommodates the system and supports the weight of the basic system and options.

A minimum of six inches (15.24 cm) of clearance is recommended on all sides of the iPDS system to allow proper cooling of the unit. Keep the iPDS air vents clear of any obstructions.

The power cords for the system plug into three-conductor power outlets. The round pin is the safety power ground. If three-conductor power outlets are not available, do not use three-prong to two-prong adapters. Have a qualified electrician rewire the power outlets to accommodate the third wire.

The power plugs shipped with the system are for 120 Vac, three-prong outlets found in the United States and Canada. If the power outlets in a region differ, follow the instructions in the next section for changing the power plugs.

To minimize the system's sensitivity to static electricity, i.e., electrostatic discharge (ESD):

a. Maintain a relatively high humidity (> 60%).

b. Use antistatic mats in the work area.

Save the shipping carton and shipping material in case it is needed later for shipping the system.

Initial Installation Procedures

The following step-by-step procedure is for the initial installation of the basic system. The installation of options is covered in later sections.

The system is shipped fully assembled and tested. Once the system is removed from the shipping carton, proceed as follows.

1. Place the system in operating position as shown in figure A-1.

   a. Lower the bail to raise the unit. The bail is the metal bar on the bottom of the unit.

   b. Lower the handle until it rests against the base of the system.

   c. Place the unit on the work surface so that it rests on the bottom feet and the bail.

   d. Press the keyboard latch to release the keyboard. Lower the keyboard to operating position.

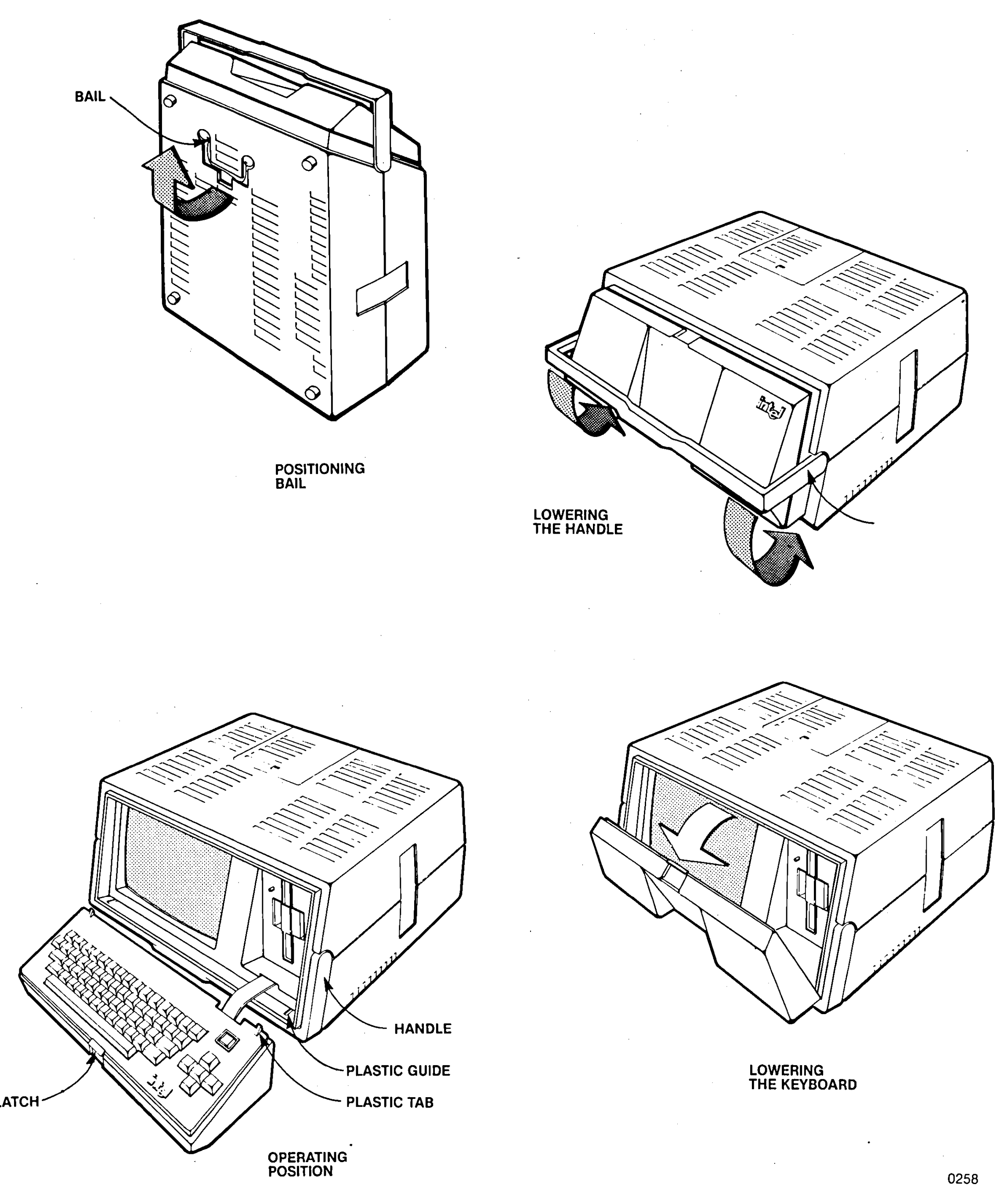


Figure A-1 Lowering the Keyboard to Operating Position

CAUTION

Do not operate the system in carrying position. The system must be opened and in operating position to dissipate the heat properly. A minimum of six inches (15.24 cm) of clearance is recommended on all sides and the air vents must be clear of any obstructions, to allow proper cooling of the unit.

2. Remove the cardboard insert from the disk drive by opening the drive door as shown in figure A-2. Pull the drive door release out and to the right. The insert protects the drive while transporting the system. Use it whenever moving the system. See figure A-2.

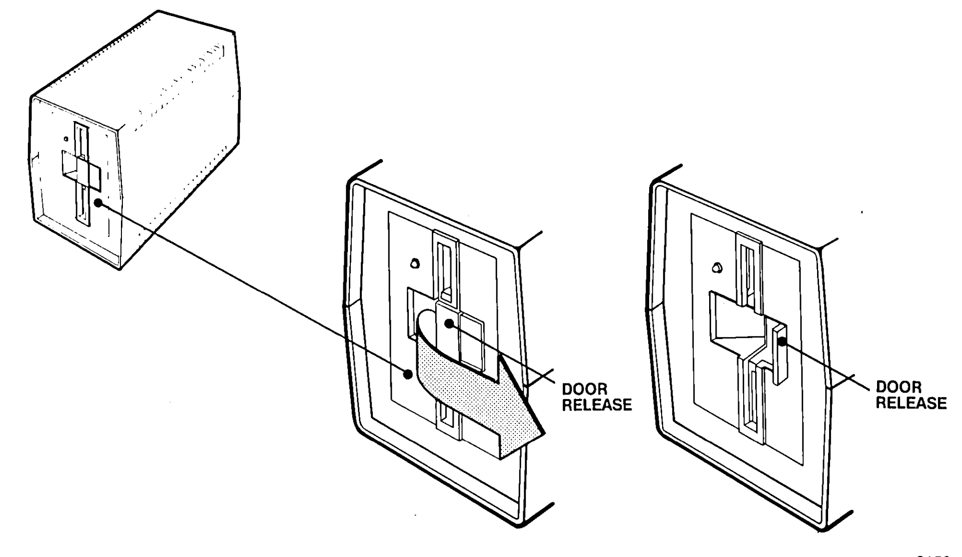


Figure A-2 Door Release on Disk Drive

3. If the power outlets in the region do not match the power plugs supplied with the iPDS system, cut off the power plug supplied with the system and install the appropriate power connector for the area. See figure A-3. The power cable is a three-conductor cable. The brown wire is the power line, the blue wire is neutral, and the green wire with the yellow stripe is the ground wire in conformance with the IEC color coding standard. Do not connect the system to the power outlet yet.

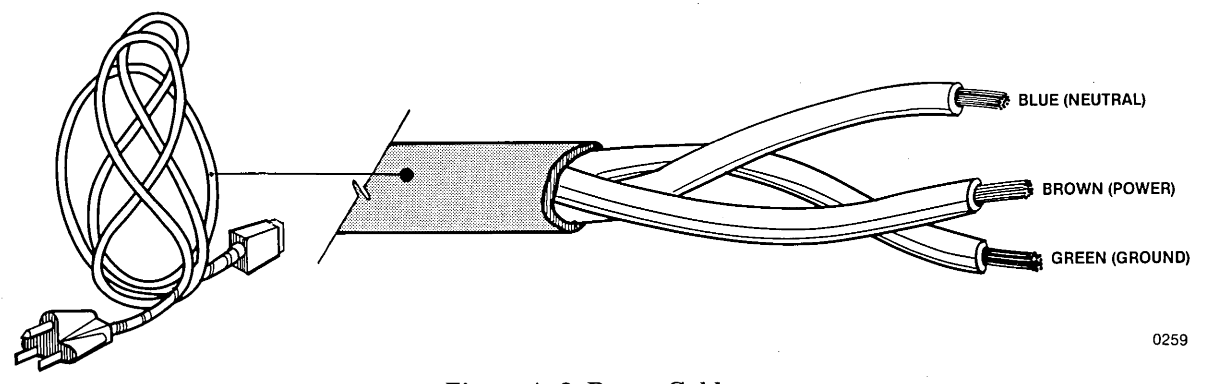
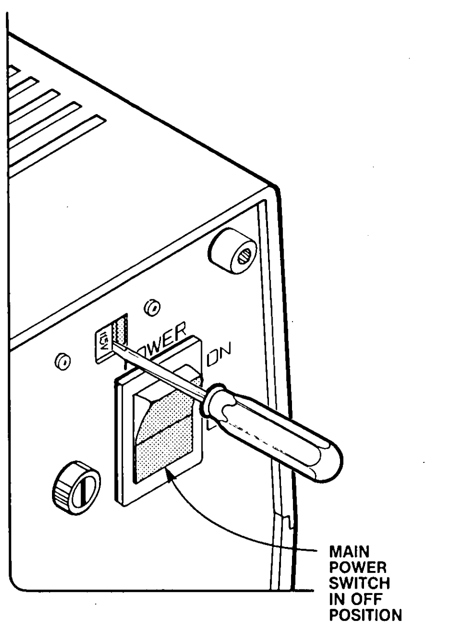


Figure A-3 Power Cable

WARNING

Changing the power cord involves hazardous voltage and current levels. To avoid the risk of electric shock and fire, the power cord should be changed only by qualified technical personnel. Turn the power off.

4. Use a small object, such as a screwdriver, as a lever to set the line voltage switch to the appropriate value. See figure A-4. Slide the switch to the right (labelled 115 V) for 90-132 Vac operation or to the left (labelled 230 V) for 198-264 Vac operation.



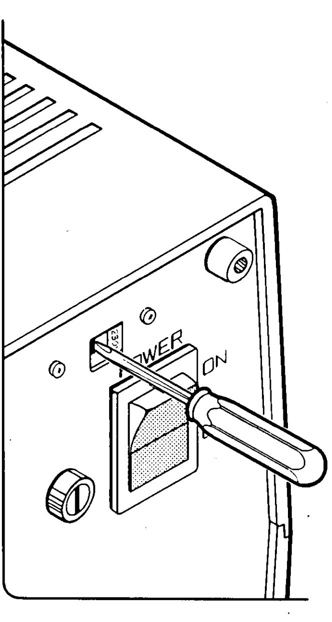


Figure A-4 Line Voltage Switch

5. If the line voltage in the region is between 198 and 264 Vac, replace the 3A fuse installed in the system with the 1.6A 250 Vac fuse provided in the plastic accessories package. Instructions for replacing the fuse are in the following section.

6. Turn the power off as shown in figure A-5. Connect the power cable to the AC power outlet and to the power connector on the lower right corner of the rear panel.

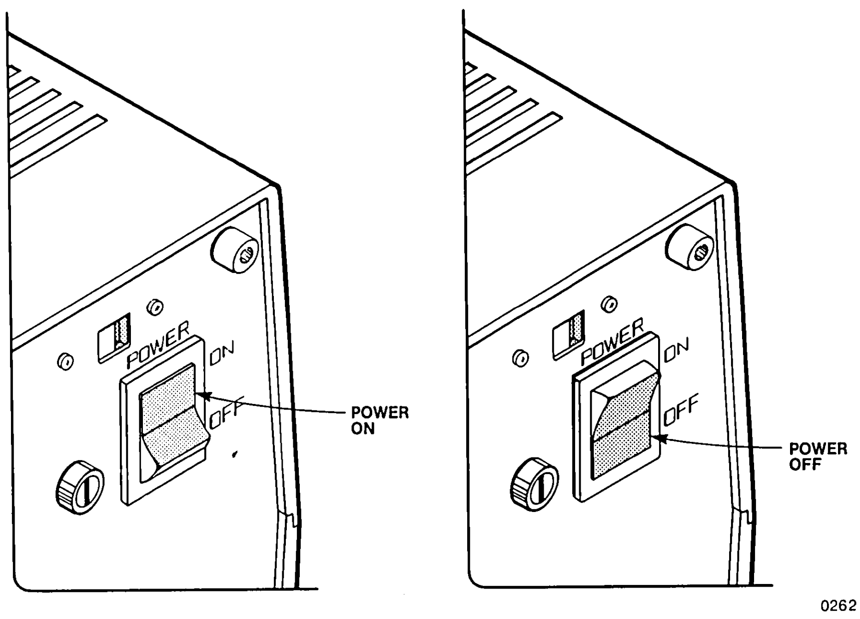


Figure A-5 Power Switch

7. Press the power switch to turn on the system as shown in figure A-5.

Unless options are to be installed, the installation procedure is now complete.

As soon as the system is on, a diagnostic program stored in a system PROM is automatically run to check the system components. Failures detected by the diagnostic program are described in Appendix B. If all the components pass the test, the system can be initialized as described in Chapter 3. After the system is initialized, back up the system disk as described in Chapter 3. The confidence test also described in Chapter 3 (described in detail in Appendix B) should be run as part of the initialization procedure.

Changing the Fuse

The fuse must always be replaced by another fuse with the proper value for the line voltage. For a 90-132 Vac line voltage, use a 3 ampere slow blow fuse. For a 198-264 Vac line voltage, use a 1.5 or 1.6 ampere slow blow fuse.

To replace the fuse:

1. Turn the power off and unplug the main power plug.
2. Using a flat blade screwdriver or a small coin, turn the fuse holder counterclockwise as shown in figure A-6.
3. Once loosened, slide the fuseholder and fuse out as shown.

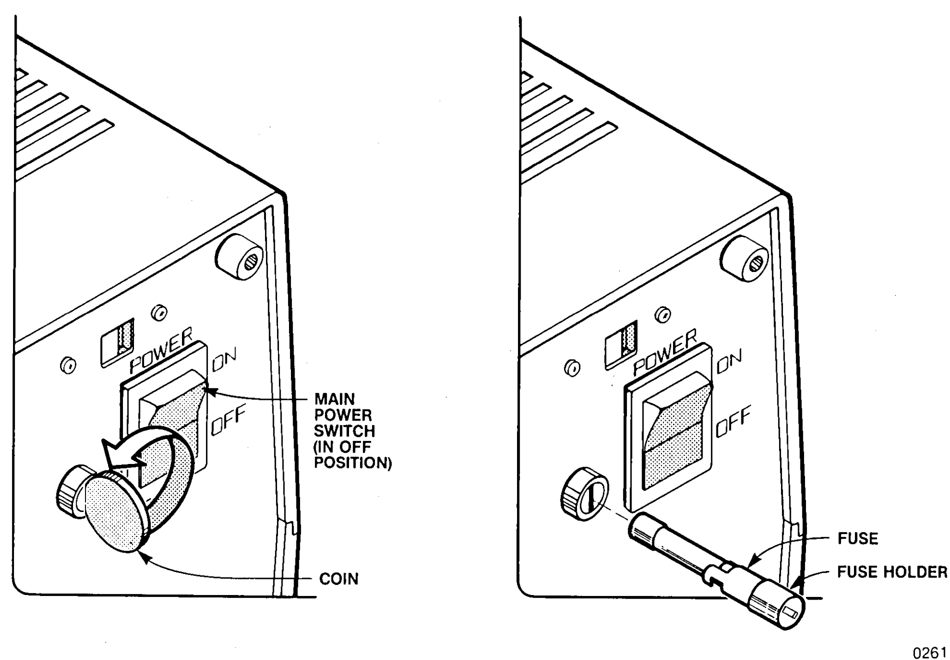


Figure A-6 Changing the Fuse

4. Replace the fuse in the holder with a new fuse of an amperage rating and size corresponding to the line voltage normally selected for the machine. These amperage ratings are specified as follows:

   Fuse Protection: 90-132 Vac, use 3 ampere slow blow fuse
   198-264 Vac, use 1.5 or 1.6 ampere slow blow fuse

5. Slide the fuse and fuseholder back in and tighten by pressing in and turning the fuseholder clockwise at the same time.

6. The main power plug can now be plugged back into the power source. The system can be turned on.

Never remove the top cover. There is a risk of electric shock or fire from high voltage. Repairs should be performed by qualified service personnel only.

Installing Options

This section provides installation instructions as well as hardware specifications for the iPDS options.

WARNING

Installation of some of the options involves working with hazardous voltage and current levels. To avoid the risk of electric shock and fire, options should be installed only by qualified technical personnel.

Removing the I/O Panel

Installation of some of the options requires removal of the I/O panel on the rear panel of the iPDS unit.

1. Turn the system off and disconnect the power plug.
2. Remove the four retaining screws from the rear I/O panel. See figure A-7.
3. Pull the I/O panel/board assembly out about six inches from the rear of the unit. The circuit boards are connected to the rear panel and slide out as the I/O panel is removed.
4. Disconnect the keyboard cable (labelled J7), the disk drive cable (labelled J3), and the plug-in adapter board cable (labelled J6 if installed) by moving the connector locks on either side of the connector toward the PC board; away from the cable connectors. Moving the connector locks pushes the cable out of the connector. See figure A-7 for locations of the cables. See figure A-8 for details about connector locks.
5. Slide out the panel/board assembly about six inches additional and disconnect the power cable (labelled J10) and the CRT cable (labelled J8). The power and CRT cables do not have connector locks.
6. Slide out the assembly the rest of the way.

To replace the I/O panel:

1. With the system off, align the circuit boards with the card guides as shown in figure A-9.
2. Slide the boards into the chassis about six inches and reconnect the power cable (labelled J10) and the CRT cable (labelled J8).
3. Slide the boards in an additional six inches and reconnect the keyboard cable (labelled J7), the disk drive cable (labelled J3), and the plug-in module adapter board cable (labelled J6 if installed). To reconnect these cables, push the connector locks back up to nearly the lock position. Plug cables into the cable connectors. Push the connector locks into the lock position. See figure A-8.
4. Slide the boards in the rest of the way until the I/O panel/board assembly is flush with the rest of the rear panel.
5. Replace the four retaining screws.
6. Connect the power cord.
7. Power the system back on.

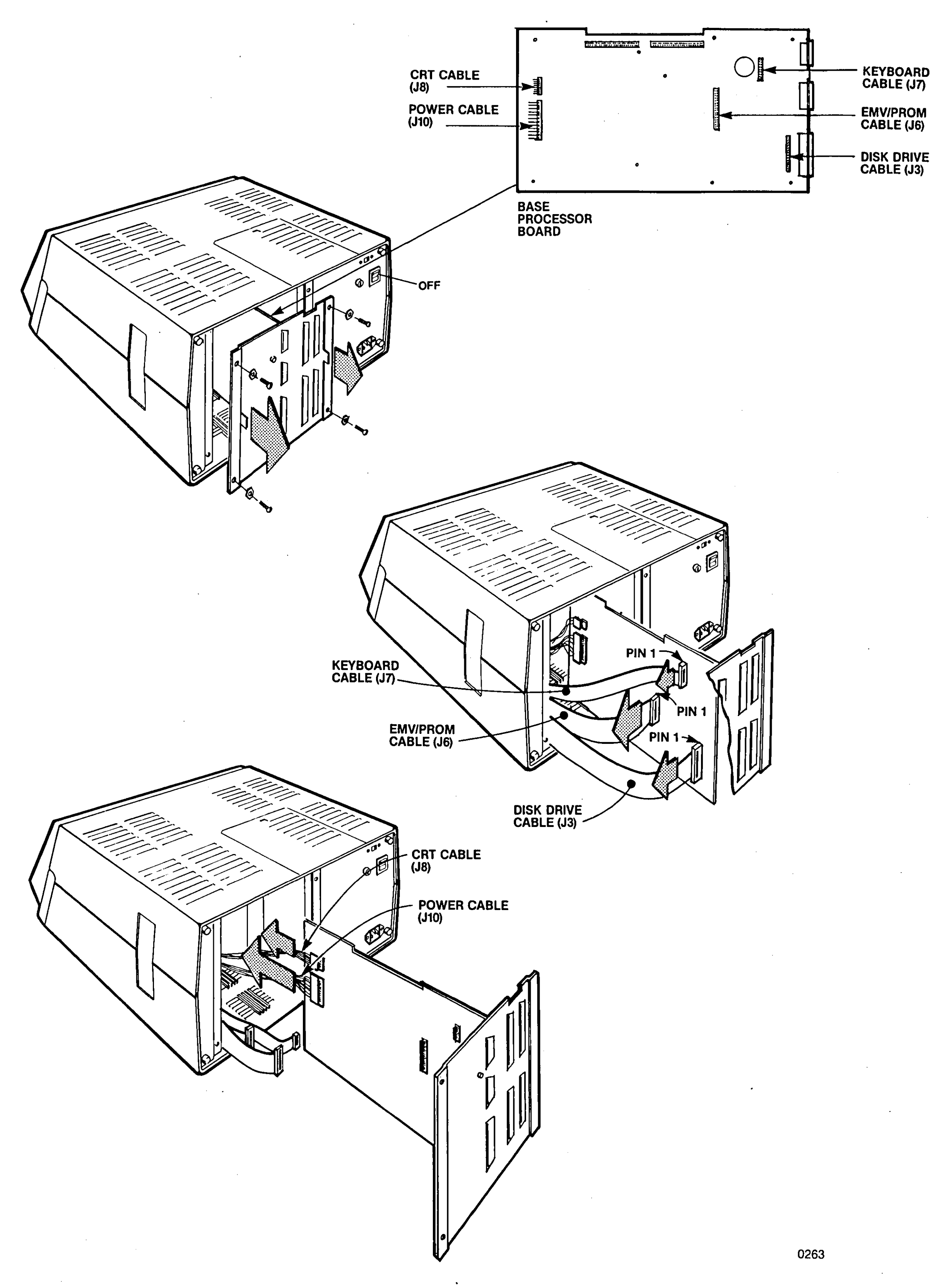


Figure A-7 Removing the I/O Panel

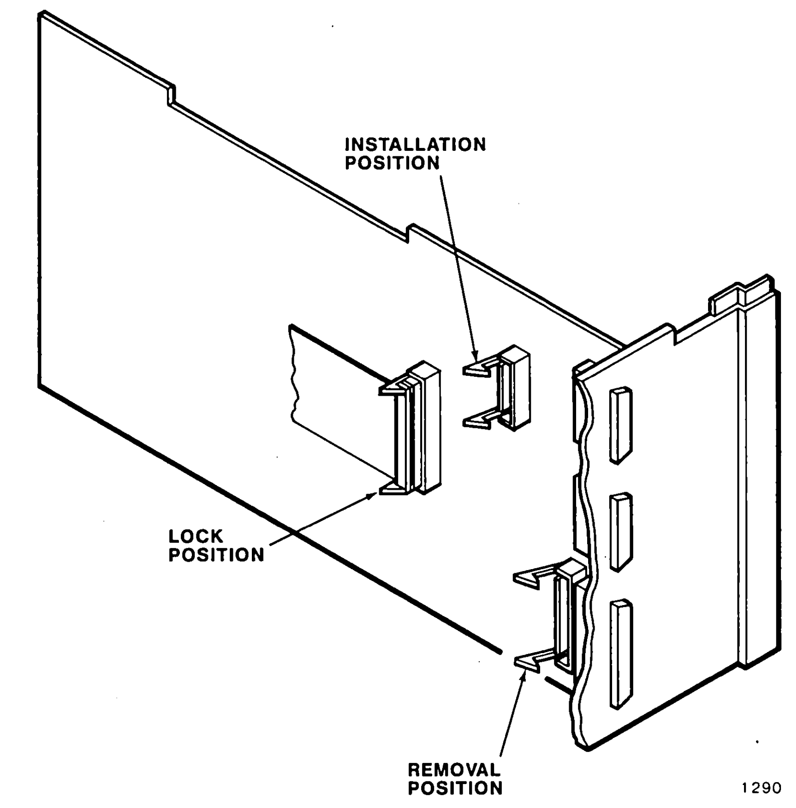


Figure A-8 Using the Connector Locks

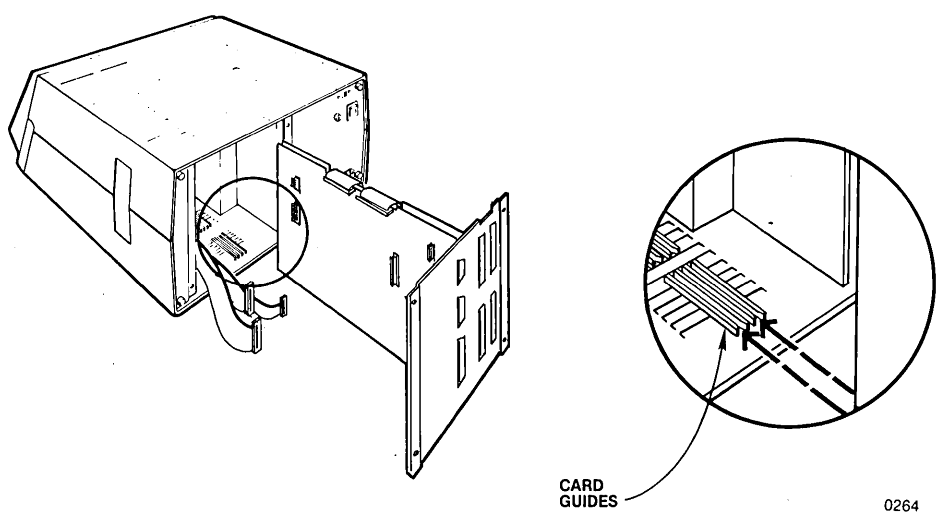


Figure A-9 Replacing the I/O Panel

Connecting a Serial Device

The system provides an EIA RS-232-C interface for asynchronous data transfer by using the 8251A USART controller chip. Several jumpers on the base processor board allow the user to reconfigure the iPDS serial I/O connector to be compatible with devices attached to the serial port. The iPDS system is shipped with jumpers installed to make the serial port appear as a Data Communication Equipment (DTE) device to be connected to its serial port when serial port and the jumpers controlling the serial port configuration.

- The RTS (Request To Send) signal from the 8251A USART is shipped connected to the CTS (Clear To Send) signal from the device to the USART. The CTS signal from the USART is shipped connected to the RTS signal from the device to the USART. In effect, this configuration jumpers the RTS and CTS lines for local communication for example, with a terminal. The user can reconfigure these lines so that the 8251A RTS is connected to the external RTS line and the 8251A CTS is connected to the external CTS line by changing the plug-in type jumper. This configuration provides for remote communications, for example, using a modem. See figure A-10.

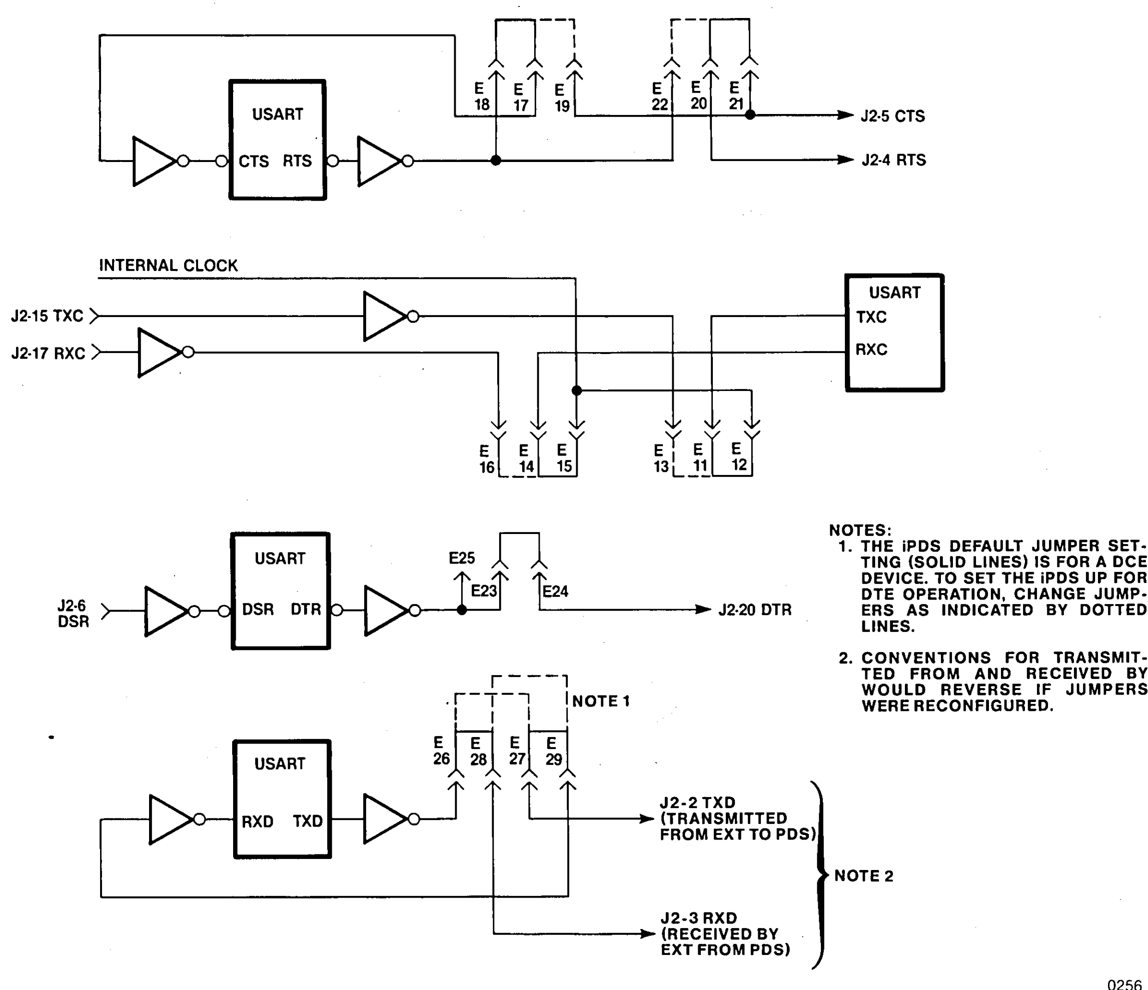


Figure A-10 Schematics for the Serial I/O Interface

- The TXC (Transmit Clock) signal and the RXC (Receive Clock) signal to the 8251A USART are each connected with a plug-in type jumper to an internal clock line. The user can disconnect the internal clock and substitute an external clock signal by changing the two jumpers. The clock signal is used to generate the baud rate for the transmission. See figure A-10.

- The DTR (Data Terminal Ready) signal from the 8251A USART is shipped jumpered to the DTR line on the serial I/O connector on the rear panel. If the DTR signal is not needed for the serial device connected, the user can disconnect this line by changing the plug-in type jumper. See figure A-10.

- The TXD (Transmit Data) signal from the 8251A USART is shipped connected to the RXD (Receive Data) line on the serial I/O connector on the rear panel. The RXD (Receive Data) signal to the 8251A USART is shipped connected to the TXD (Transmit Data) line on the serial I/O connector on the rear panel. In effect, this configuration crosses the transmit and receive data lines for compatibility with many terminals. The user can reconfigure these lines so that the 8251A TXD signal is connected to the external TXD line and the 8251A RXD signal is connected to the external RXD line by changing the plug-in type jumpers. See figure A-10.

Configuring the CTS and RTS Lines

To connect the 8251A CTS signal with the external CTS line and the 8251A RTS signal with the external RTS line, change two jumpers as follows:

1. Turn the system off and disconnect the power cable.

2. Remove the I/O panel and slide out the panel/board assembly about 4 or 5 inches as described previously. Do not remove any cables.

3. Locate pins E17 (CTS signal to the 8251A), E18 (RTS signal from the 8251A), E19 (CTS line on the external serial connector), E20 (RTS line on the external serial connector), E21 (CTS line on the external serial connector), and E22 (RTS signal from the 8251A) on the base processor board. Pins E17 and E18 are connected with a plug-in type jumper when shipped from the factory as are pins E20 and E21. See figure A-11.

4. Unplug the two jumpers and plug back in to connect pin E17 with E19 and to connect pin E20 with E22. See figure A-12.

5. Replace the I/O panel/board assembly as described previously.

Configuring the RXC and TXC Lines

To use an external clock signal to generate the baud rate, change the two jumpers that connect RXC and TXC to an internal clock signal as follows:

1. Unplug the system is unplugged and disconnect the power cable.

2. Remove the I/O panel and slide out the panel/board assembly about 4 or 5 inches as described previously. Do not remove any cables.

3. Locate pins E11 (TXC signal to the 8251A), E12 (internal clock signal), E13 (external transmit clock signal), E14 (RXC signal to the 8251A), E15 (internal clock signal), and E16 (external receive clock signal) on the base processor board. Pins E11 and E12 are connected with a plug-in type jumper when shipped from the factory as are pins E14 and E15. See figure A-10.

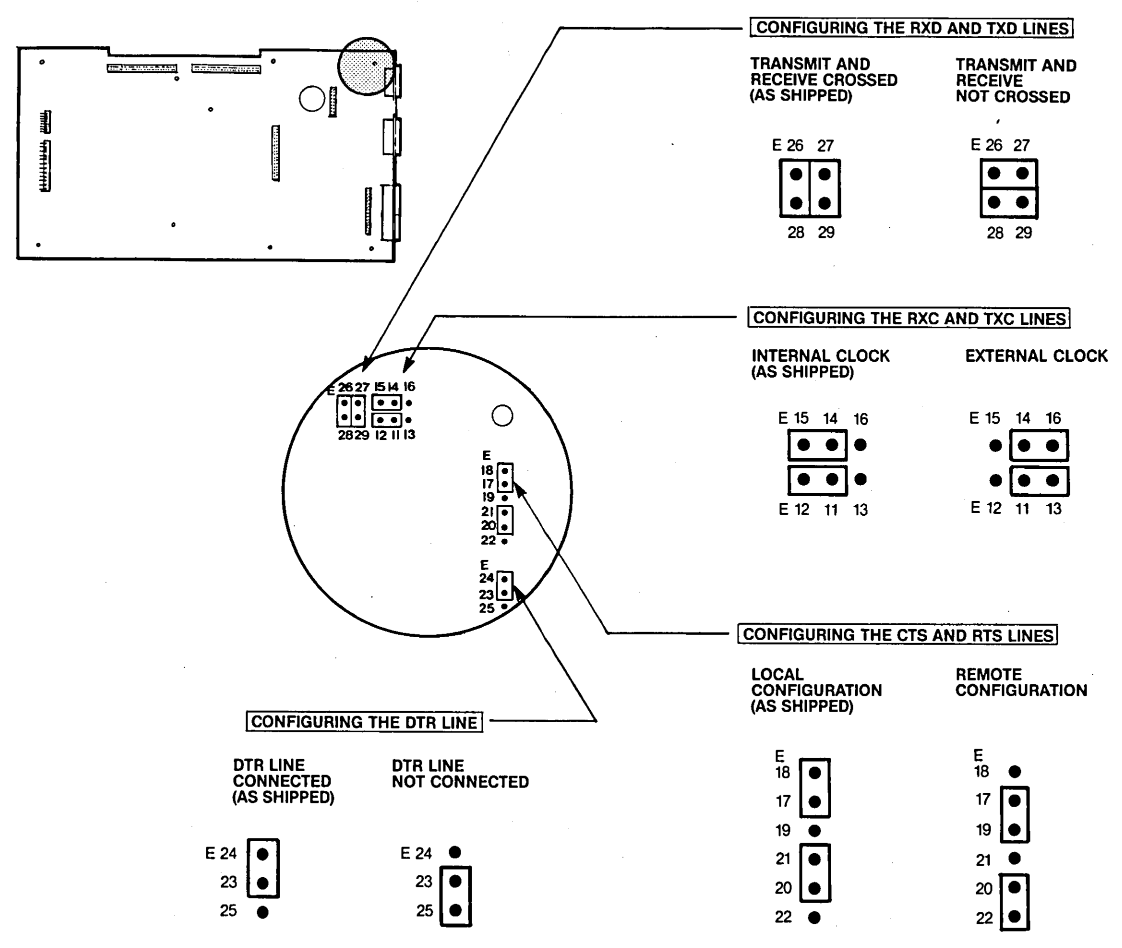


Figure A-11 Removable Jumper Location and Configuration

REMOVING

REPLACING

0267

Figure A-12 Removing and Replacing the Plug-in Type Jumpers

4. Unplug the two jumpers and plug back in to connect pin E11 with E13 and to connect pin E14 with E16. See figure A-11.

5. Replace the I/O panel/board assembly as described previously.

Configuring the DTR Line

To disconnect the DTR line from the serial I/O connector on the rear panel:

1. Unplug the system and disconnect the power cable.

2. Remove the I/O panel and slide out the panel/board assembly about 4 or 5 inches as described previously. It is not necessary to remove any cables.

3. Locate pins E23 (DTR signal from the 8251A), E24 (DTR line on the external serial connector), and E25 (DTR signal from the 8251A, same as E23) on the base processor board. Pins E23 and E24 are connected with a plug-in type jumper when shipped from the factory. See figure A-11.

4. Unplug the two jumpers and plug back in to connect pin E23 with E25. See figure A-12.

5. Replace the I/O panel/board assembly as described previously.

Configuring the RXD and TXD Lines

To connect the RXD line to the RXD signal from the 8251A and the TXD line to the TXD signal from the 8251A, change the two jumpers that connect the RXD signal and TXD signal to the data lines as follows:

1. Unplug the system and disconnect the power cable.

2. Remove the I/O panel and slide out the panel/board assembly about 4 or 5 inches as described previously. Do not remove any cables.

3. Locate pins E26 (TXD from the 8251A), E27 (TXD line on the external serial connector), E28 (RXD line on the external serial connector), and E29 (RXD signal to the 8251A) on the base processor board. Pins E26 and E28 are connected with a plug-in type jumper when shipped from the factory as are pins E27 and E29. See figure A-11.

4. Unplug the two jumpers and plug back in to connect pin E26 with E27 and to connect pin E28 with E29. See figure A-12.

5. Replace the I/O panel/board assembly as described previously.

Connecting a Serial Device

After configuring the serial interface for the device to be connected, proceed as follows:

- Attach the cable from the serial device to the serial I/O connector on the iPDS rear panel. The I/O connector is a D-subminiature 25 pin female connector. The required mating connector is a D-subminiature 25 pin male, AMP205208-1.

Serial Interface Specifications

Table A-1 gives the specifications for the serial I/O connector. The serial interface conforms to the EIA RS-232-C electrical standards. Pins 2, 3, 4, 5, 15, 17, and 20 are user controlled lines as previously described.

Table A-1 Serial Interface Specifications

Pin	Signal	Function
1	CHASSIS GND	Chassis Ground
2	TXD	Transmitted Data Out
3	RXD	Received Data In
4	RTS	Request To Send
5	CTS	Clear To Send
6	DSR	Data Set Ready
7	GND	Signal Ground
8		Not Used
9		Not Used
10		Not Used
11		Not Used
12		Not Used
13		Not Used
14		Not Used
15	TXC	Transmit Clock
16		Not Used
17	RXC	Receive Clock
18		Not Used
19		Not Used
20	DTR	Data Terminal Ready
21		Not Used
22		Not Used
23		Not Used
24		Not Used
25		Not Used

Optional Processor

The optional processor board is mounted on the base processor board between the base processor and the CRT. To install the optional processor board, follow these steps:

1. Unplug the system and disconnect the power cable.
2. Remove the I/O panel/board assembly as described previously.
3. Find the four mounting locations on the base processor board. See figure A-13.
4. Secure the four 1/4 inch stand offs to each of the mounting locations on the base processor board with the 1/4 inch x 6/32 inch screws. See figure A-14.
5. Align the optional processor board mounting holes over the four spacers installed on the base processor board. See figure A-15.
6. Secure the optional processor board to the top of the spacers with the nylon nuts. See figure A-14.

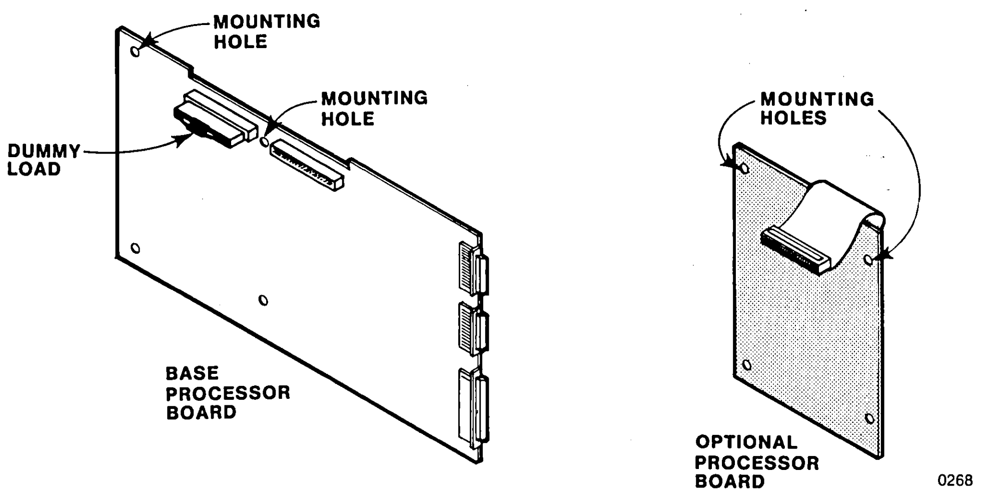


Figure A-13 Mounting Locations for Optional Processor

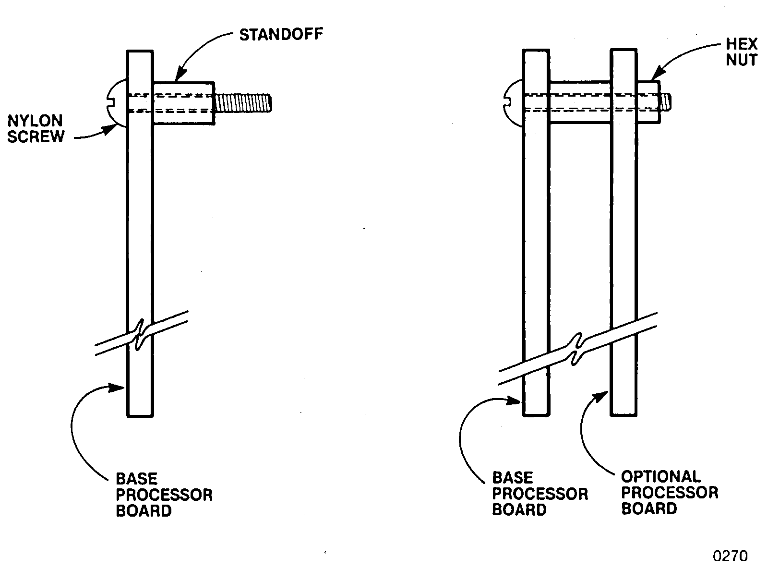


Figure A-14 Mounting Technique for Optional Processor

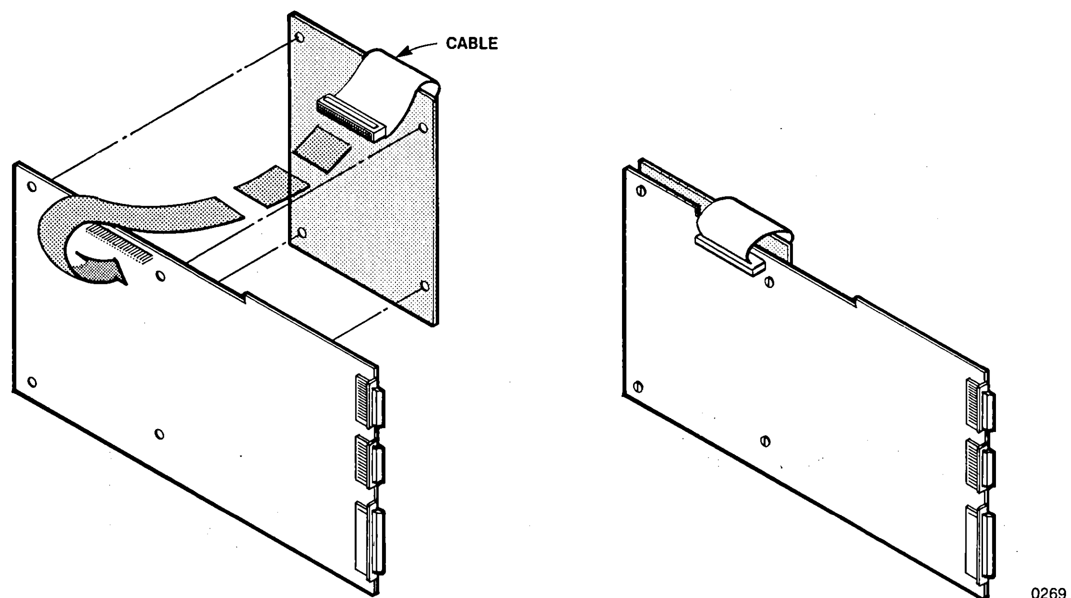


Figure A-15 Aligning the Optional Processor Board

7. Remove the dummy load from J4. This dummy load is shipped installed in the system. See figure A-13. The dummy load is required for running the system without the optional processor

8. Attach the 50-conductor ribbon cable from the connector on the optional processor board to connector J4 on the base processor board. See figure A-15 for the location of the connector.

9. If a multimodule adapter has been installed, connect the 34-conductor ribbon cable from the multimodule adapter board to connector J2 on the optional processor board. See figure A-16.

10. Replace the I/O panel/board assembly as described previously.

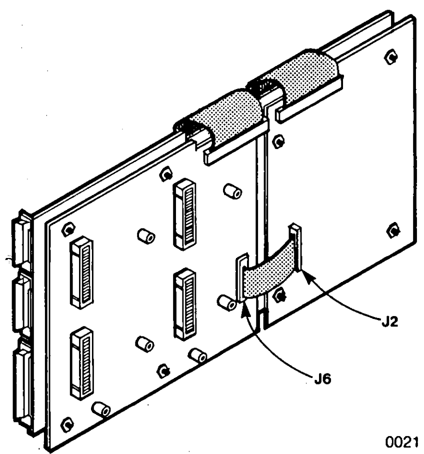


Figure A-16 Optional Processor Connection to Multimodule Adapter

Multimodule Adapter

The multimodule adapter board is installed like the optional processor board. It is mounted on the base processor board between the base processor and the storage area.

To install the multimodule adapter board, follow these steps:

1. Unplug the system and disconnect the power cable.

2. Remove the I/O panel/board assembly as described previously.

3. Find the five mounting locations (holes A and C) on the base processor board. See figure A-17.

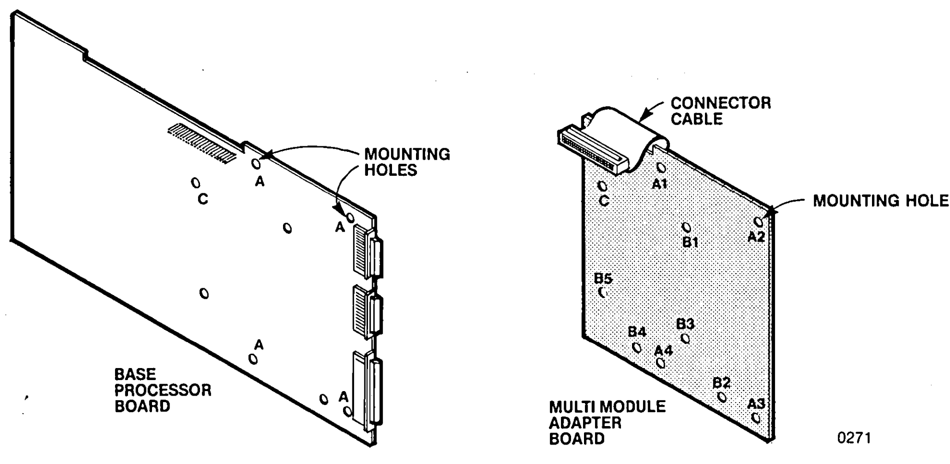


Figure A-17 Mounting Locations for Multimodule Adapter

4. Secure the five 1/4 inch spacers to each of the mounting locations on the base processor board with 6-32 x 0.25 inch screws. See figure A-18.

5. Find the six multimodule mounting locations (holes B and C) on the multimodule adapter board. See figure A-17.

6. Secure the five 1/2 inch nylon standoffs to the five locations labelled B in figure A-17 using the nylon 6-32 x 0.25 inch screws.

7. Align the multimodule adapter board mounting holes over the five spacers installed on the base processor board. See figure A-19.

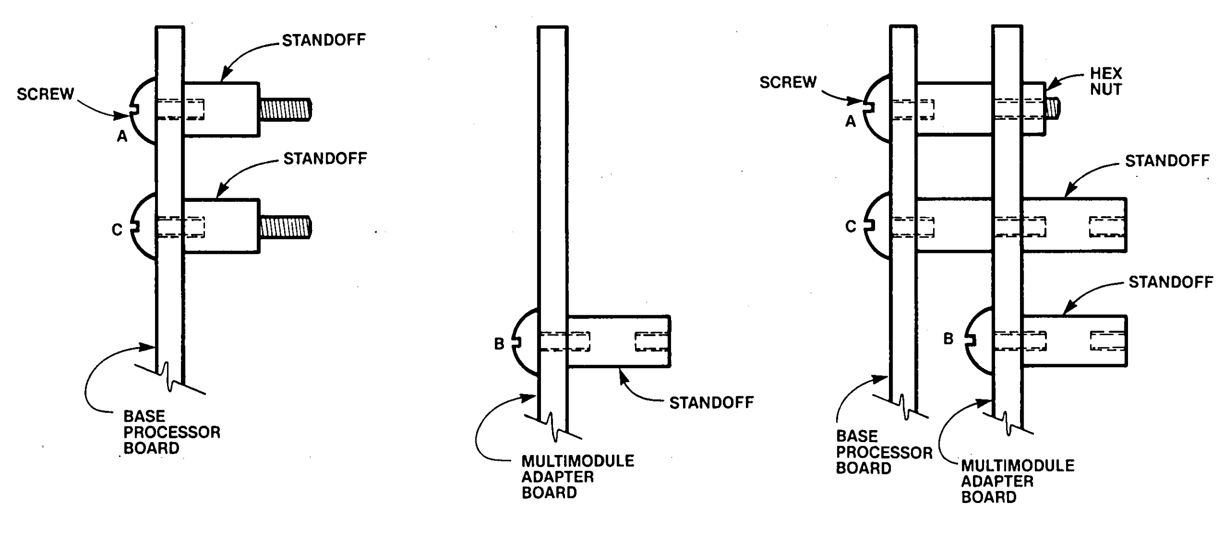


Figure A-18 Mounting Technique for the Multimodule Adapter Board

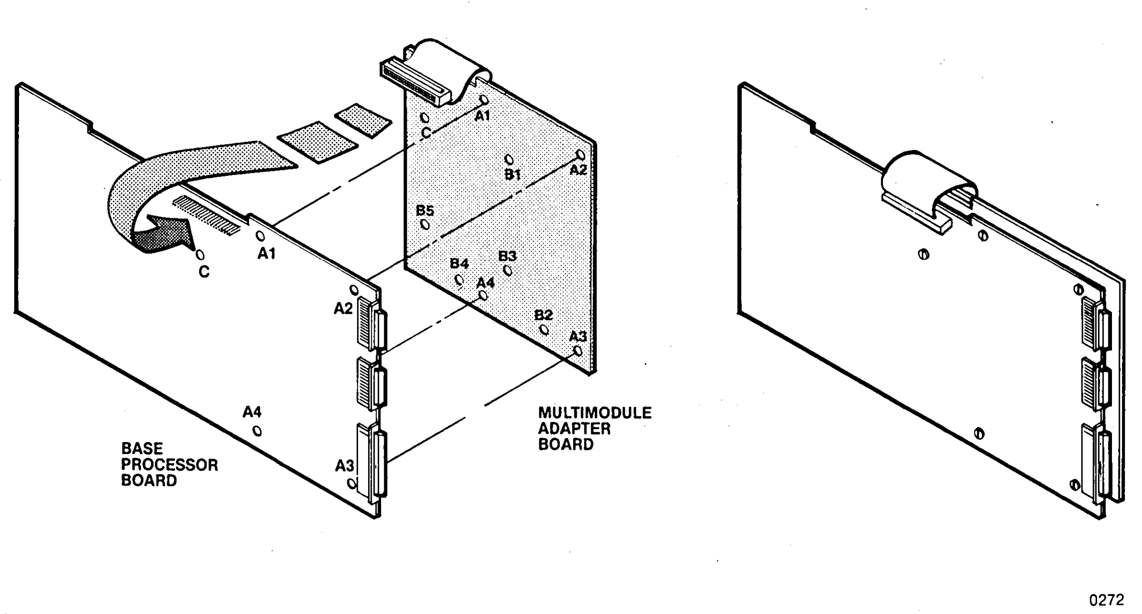


Figure A-19 Aligning the Multimodule Adapter Board

8. With the remaining 1/2 long nylon standoff, secure the multimodule adapter board to the spacer projecting through the sixth mounting location on the multimodule board (labelled C in figure A-17).

9. Secure the multimodule adapter board to the top of the remaining standoffs with the nylon nuts (the holes labelled A). See figure A-18.

10. Attach the 50-conductor ribbon cable from connector on the multimodule adapter board to connector J5 on the base processor board. See figure A-18 for the location of the connectors.

11. If an optional processor has been installed, connect the 34-conductor ribbon cable from the connector on the multimodule adapter board to connector J2 on the optional processor board. See figure A-16.

12. Replace the I/O panel/board assembly as described previously.

Multimodule

The Multimodule Adapter Board contains four iSBX connectors labeled J1 through J4, each of which accepts one, single wide, multimodule board. The four connectors are divided into two groups of two connectors each. Multimodules, such as the iSBX 251 Bubble Memory Multimodule, are double wide and take the space of two boards although they only use one connector. thus, only two bubble memory multimodules can be installed on the adapter board.

The instructions for installing a double wide multimodule board such as the bubble memory multimodule on the adapter board follow.

1. Disconnect the system power cable and remove the I/O panel/board assembly as described previously.

2. Find the mounting location on the multimodule adapter board. See figure A-20.

3. Locate pin 1 of the iSBX connector (P1) on multimodule board and on the multimodule adapter board. Use connector J1 on the adapter board if the bubble memory is used for physical disk device 4 and J3 if the bubble memory is used for physical disk device 5. See figure A-20.

4. Align pin 1 of connector P1 with pin 1 of connector J1 or J3. See figure A-21.

5. Align the mounting holes on the multimodule board with the spacers already attached to the multimodule adapter board. See figure A-21.

6. Secure the multimodule board to the top of the spacers with three 1/4 inch x 6-32 inch screws. See figure A-22.

7. Discard the remaining spacers and screws.

8. Replace the I/O panel board assembly as described previously.

Installation instructions for a single wide multimodule are similar to the instructions for a double wide multimodule given above. Only one mounting location is used per single wide multimodule board. See figure A-23.

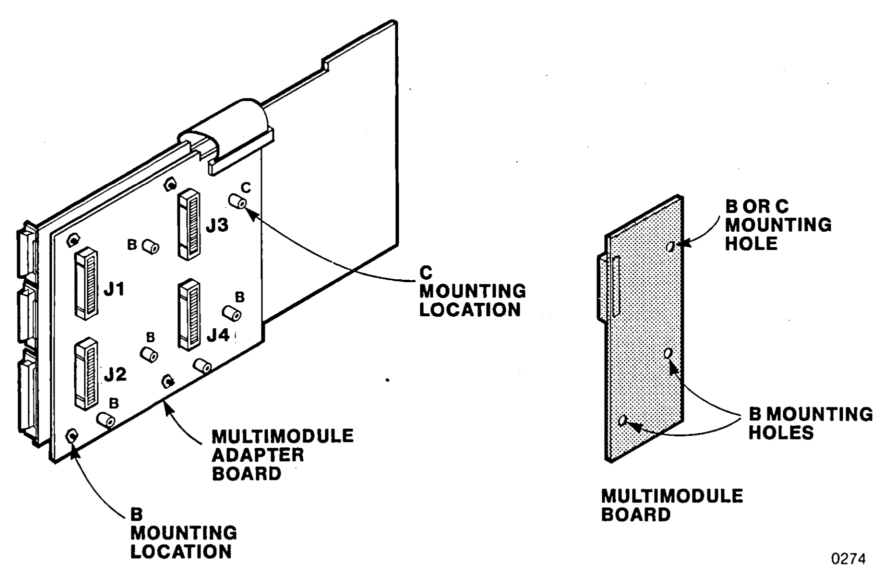


Figure A-20 Mounting Locations for Double Wide Multimodule Boards

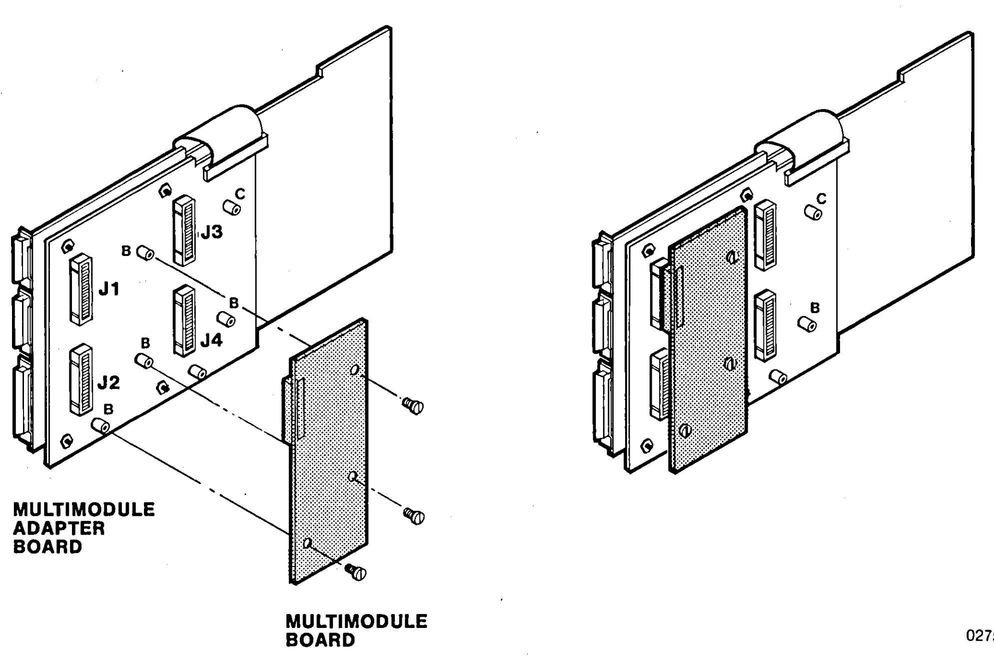


Figure A-21 Aligning Double Wide Multimodule Boards

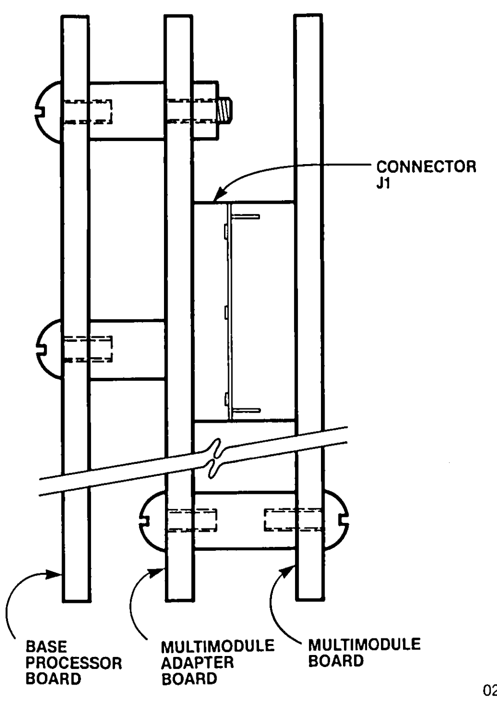


Figure A-22 Mounting Technique for Multimodule Boards

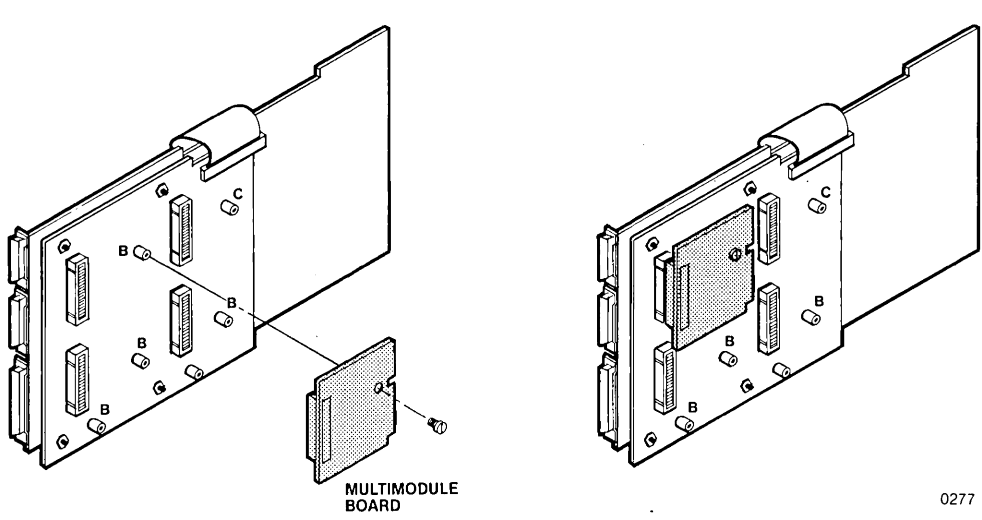


Figure A-23 Mounting a Single Wide Multimodule Board

For some multimodules (the parallel I/O multimodule, for example), a cable must be run from the multimodule board to a mounting hole for the I/O connector on the rear panel. The cable is provided with the multimodule board. (No cable is required for some multimodules, such as the bubble memory multimodule.) Four cutouts are provided on the rear I/O panel for this purpose. Use a drafting knife or a similar tool to remove one of the panels (see figure A-24).

1. Attach the connector plate on the I/O connector using the hardware provided. See figure A-25.
2. Attach the connector to the mounting hole using the screws, lock washers a nuts. See figure A-25.
3. Attach the other end of the cable to the multimodule board.
4. Slide the assembly back into the chassis replacing the rear panel.

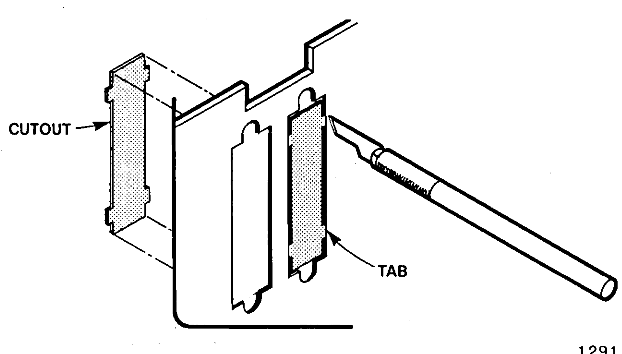


Figure A-24 Removing Rear Panel Cutouts

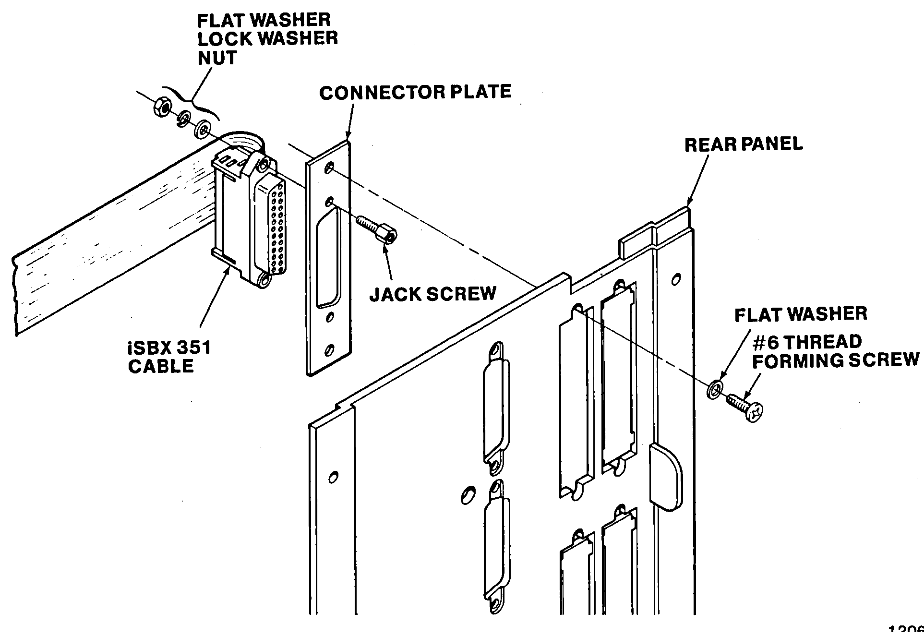


Figure A-25 Connecting a Cable to the Rear Panel Cutouts

Plug-In Module Adapter

The Plug-in Module Adapter board is installed in the main chassis between the base processor and the plug-in module door on the right side of the system. To install this board, proceed as follows:

1. Disconnect the system power cable and remove the I/O panel/board assembly as described previously.
2. Find the mounting locations for the adapter board assembly on the support bracket. See figure A-26.
3. Insert the adapter board assembly into the card guides and slide it fully into the unit.
4. Attach the adapter board assembly to the support bracket as shown in figure A-26.
5. Slide the boards a few inches into the system and attach the 50-conductor ribbon cable from the connector on the plug-in module adapter board to connector J6 on the base processor board. See figure A-26 for the location of the connectors.
6. Replace the I/O panel/board assembly as described previously. See figure A-27.

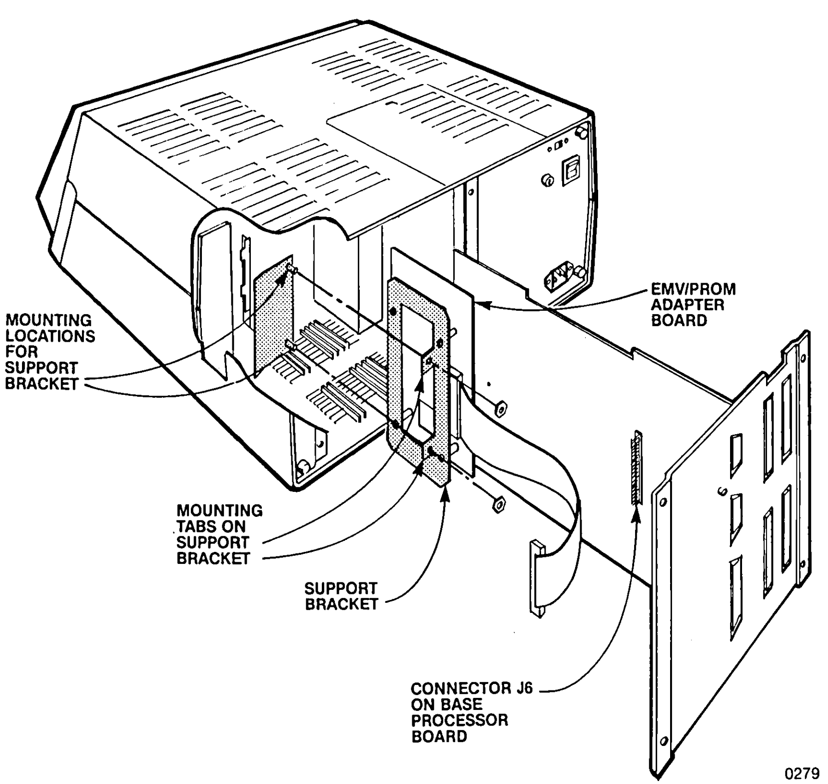


Figure A-26 Installing the Adapter Board Assembly

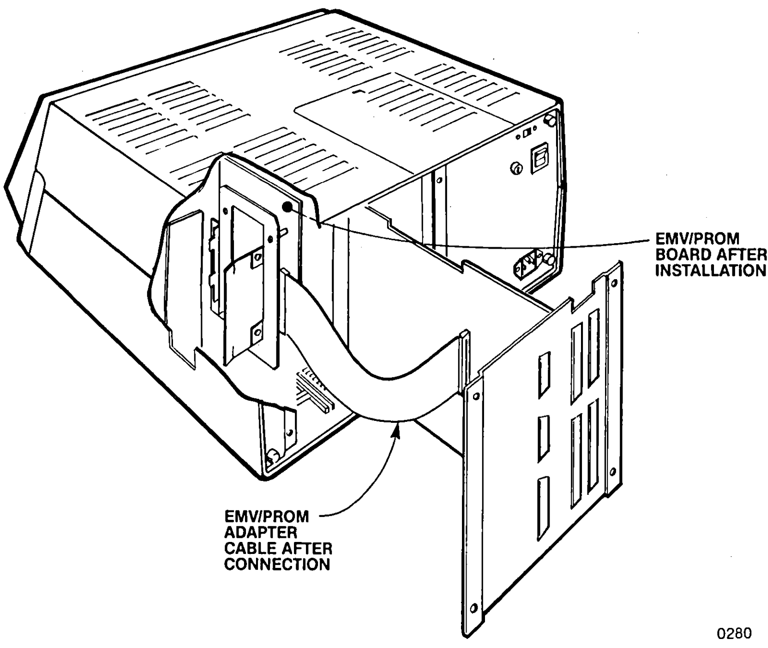


Figure A-27 Cable Connection for Adapter Assembly

Plug-In Module

The plug-in module slot breaches the electrical shielding of the iPDS system. There is a chance of electro-static discharge (ESD) passing, via the plug-in module, to the internal circuitry of the iPDS system and causing system RESETs, disk file damage, or component damage. Ensure that the iPDS system is turned OFF before inserting or removing any plug-in module.

The plug-in module (either an emulator or a PROM personality adapter) is inserted and removed from the slot on the right side of the iPDS system. Align the plastic guides and slide the module in and out as shown in figure A-28.

Connecting a Line Printer

To connect a line printer to the system:

1. Attach the printer cable connector to the D-Subminiature 25-pin female connector on the rear I/O panel. Use an Intel Centronics* cable with a D-Subminiature male AMP205208-1, or equivalent as the mating connector.

Line Printer Interface Specifications

Tables A-2, and A-3 summarize the electrical specifications and pin assignments for the printer connector on the iPDS rear panel. The standard Centronics* printer interface is used.

*Centronics is a trademark of Centronics, Inc.

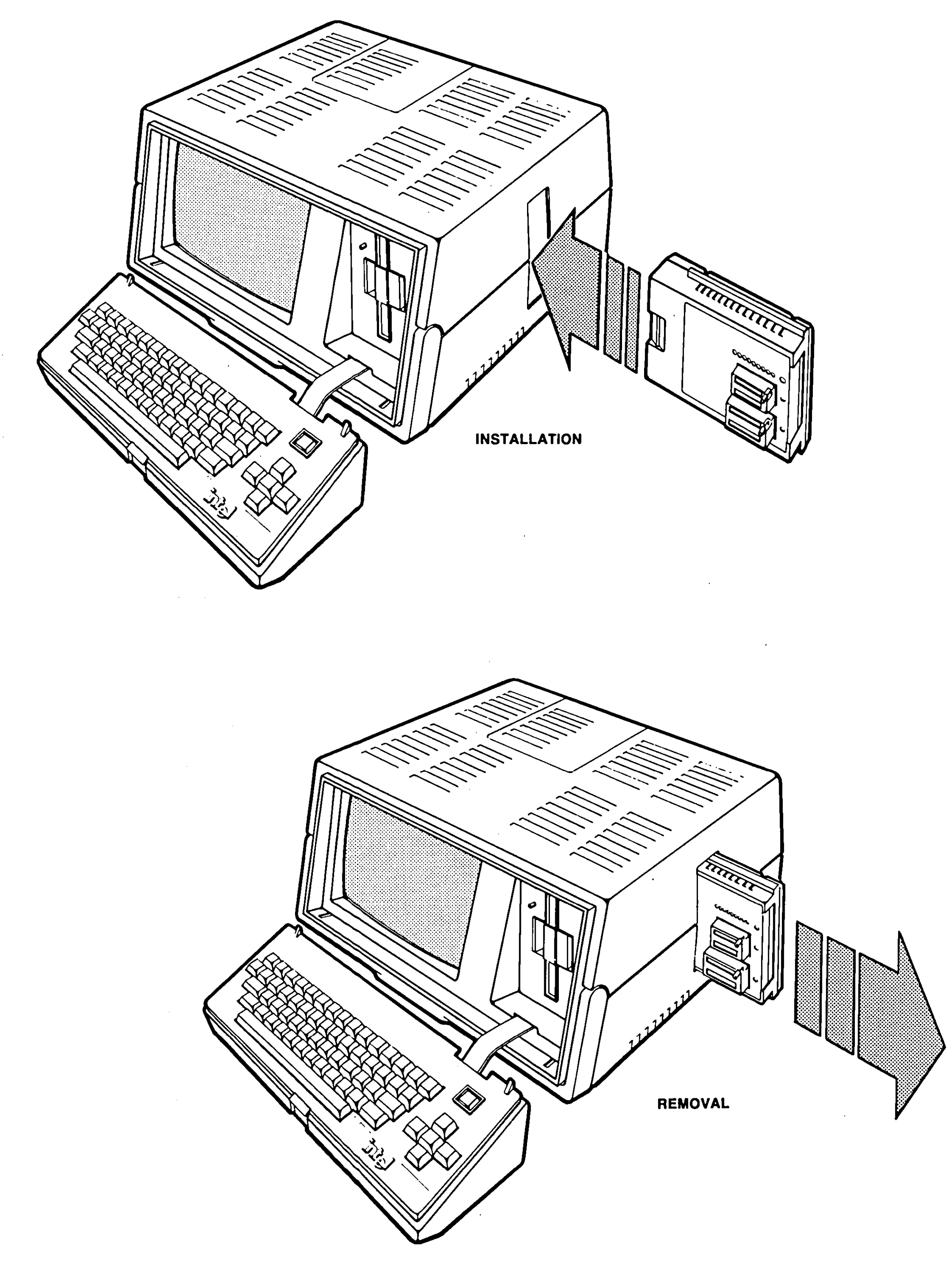


Figure A-28 Installing Plug-in Module

Table A-2 Printer Interface Specifications

Pin	Signal	Function
1	DATA 0	Data Line 0
2	DATA 1	Line 1
3	DATA 2	Data Line 2
4	DATA 3	Data Line 3
5	DATA 4	Data Line 4
6	DATA 5	Data Line 5
7	DATA 6	Data Line 6
8	DATA 7	Data Line 7
9	GND	Protective Ground
10	GND	Protective Ground
11	GND	Protective Ground
12	GND	Protective Ground
13	FAULT/	Indicates a Printer Fault Condition
14	STB/	Clock for Data to Printer
15	GND	Protective Ground
16	ACK/	Indicates the Printer Received Character
17	BUSY	Indicates the Printer Is Not Ready
18	GND	Protective Ground
19	PRIME/	Resets the Printer Logic
20		Not Used
21	GND	Protective Ground
22	SELECT	Indicates that Printer is Ready
23	GND	Protective Ground
24	GND	Protective Ground
25	CHASSIS GND	Chassis Ground

Table A-3 Electrical Specifications for Printer Interface

Inputs				
Signal	VIL max.	VIH min.	IIL max.	Termination
SELECT	.8 V	2.0 V	−12 ma	470 ohms to 5 V
BUSY	.8 V	2.0 V	−12 ma	470 ohms to 5 V
ACK/	.8 V	2.0 V	−12 ma	470 ohms to 5 V
FAULT/	.8 V	2.0 V	−12 ma	470 ohms to 5 V

Output					
Signal	Type	IOL max.	IOH max.	VOL max.	VOH min.
DATA 0-7	Totem Pole	12 ma	−10 ma	0.4 V	2.4 V
STB/	Open Collector	16 ma	---	0.4 V	2.4 V
PRIME/	Open Collector	16 ma	---	0.4 V	2.4 V

Functional Description

In this section, a functional description is given for the components of the iPDS development system. Refer to the *iPDS Field Service Manual*, order no. 143861, for further details on the theory of operation of the system.

System Chassis

The system chassis is 8"H x 16"W x 18"L and contains the CRT, the power supply, a double density double sided 5 1/4" disk drive, the base processor board installed in a card guide, three additional card guides (for the optional processor, the plug-in module adapter board option, and the multimodule adapter board option), and cables.

The rear panel includes connectors for the serial I/O channel, a line printer, the external disk drives, and the optional multimodule boards.

A separate ASCII keyboard connects to the front of the main chassis with a flat ribbon cable. The keyboard can be attached to the front of the main chassis covering the CRT and integral disk drive.

A handle is attached to the front of the chassis for carrying the unit.

A storage compartment is located at the rear of the chassis and opens from the top to allow storage of two plug-in modules with cables.

Base Processor Board

The base processor board occupies a card slot parallel to and at the left of the disk drive. The main processor on the board is an 8085A-2 processor running at 5.0 MHz operating frequency, a 200 ns clock period. Included on the board are 64K bytes of read/write memory (RAM), 2K bytes of read only memory (ROM), a serial I/O port, a line printer port, a plug-in module interface, an interrupt system, a programmable timer, a keyboard/CRT controller, a disk controller, and a multimodule interface.

Keyboard

There are no electronics in the keyboard enclosure which is 2"H x 15"W x 8"D. The keyboard can be attached or detached from the main chassis and is connected with a flat, jacketed, shielded cable to the keyboard/CRT controller on the base processor board. The keyboard contains the standard typewriter keys plus a control key, a function key, cursor control keys, and a reset key.

Integral CRT

The CRT is a 9-inch, raster scan monitor displaying 24 lines of 80 characters each. The horizontal scan rate is 15.6 KHz; the vertical scan rate is 60 Hz. The bandwidth is 18 MHz.

Integral Disk Drive

The integral disk drive is a 5 1/4", 96 tracks per inch unit with two read/write heads. There are 80 tracks on each side of the diskette, each with sixteen 256-byte sectors. The formatted capacity of the unit is 640K bytes. A signal cable connects the unit to the disk controller on the base processor, and a power cable from the power supply provides the +5 Vdc and the +12 Vdc required by the unit.

Power Supply

The power supply is a switching type power supply with a maximum input of 130 watts and an output of 88 watts with four regulated DC voltages.

User Controls

The user controls the system through the keyboard and the switches and controls on the rear panel. A rocker switch is used as the power on/off switch. A two position switch selects the input voltage of 120 Vac or 230 Vac. The CRT contrast control is mounted on the base processor and projects through the rear panel.

Optional Processor Board

The optional processor board occupies a card slot parallel to the base processor and between the base processor and the CRT. It is half the size of the base processor board, but it is functionally equivalent to the base processor with an 8085A-2 CPU, 64K bytes of RAM, and 2K bytes of ROM. It has a connector for the multimodule adapter board and a connector for the base processor board. It does not contain a serial I/O port, a line printer port, a plug-in module port, a programmable timer, a keyboard/CRT controller, or a disk controller. The keyboard, CRT, disk drives, and multimodules are shared between the processors.

Optional Multimodule Adapter Board

The multimodule adapter board occupies a card slot parallel to the base processor and directly behind the optional processor. It is about half the size of the base processor board. It provides system expansion by allowing a maximum of four multimodule boards to be connected. The adapter board is connected to each processor with a cable that provides signals and power. The multimodule adapter board supports 8-bit multimodules only and does not support DMA mode of I/O. Double width multimodules are supported. They use one connector on the adapter board, but the adjacent connector cannot be used.

Optional Plug-In Module Adapter Board

The optional plug-in module adapter board occupies a card slot parallel to the base processor and between the base processor and the plug-in module door on the right side of the main chassis. It is directly behind the disk drive and is about half the size of the base processor board. It is connected to the base processor board and contains the interface between the base processor an any plug-in module that is used.

Specifications

Tables A-4, A-5, A-6, A-7, and A-8 summarize the environmental, physical, and electrical characteristics of the system.

Table A-4 Intel Personal Development System Specifications

Environmental Characteristics	
Operating Temperature: Non-operating Temperature: Operating Relative Humidity: Non-operating Relative Humidity: Cooling:	5°C to 35°C −40°C to 62°C 20% to 80% 5% to 95% convection through vents and forced air cooling on disk drive.
Physical Characteristics	
Main Chassis:	Width: 16" (40.64 cm) Height: 8.15" (20.70 cm) Depth: 17.5" (44.45 cm) Weight: 27 lb (12.22 kg)
Keyboard:	Width: 15" (38.1 cm) Height: 2" (5.08 cm) Depth: 7.5" (19.05 cm) Weight: 2 lb (0.91 kg)
AC Requirements:	
90-132 Vac, 47-63 Hz, single phase 180-264 Vac, 47-63 Hz, single phase	

Table A-5 External Disk Drive Power Supply

AC Requirements:
90-132 Vac, 47-63 Hz, single phase 180-264 Vac, 47-63 Hz, single phase

Table A-6 External Disk Drive Physical Characteristics

Height	Depth	Width	Weight
7 inches 17.78 cm	12 inches 30.48 cm	5 inches 12.70 cm	7 pounds 3.17 kg

Table A-7 Power Supply

Voltage Programmer	Multimodule Board	Emulator/PROM
+5 V	2.35 A	-----
+12 V	0.8 A	-----
−12 volts	0.1 A	-----
+VCC (+5 V)	-----	2.5 A
+VSW (+5.7 V) *	-----	0-135 ma
+VHSW (+8 to +27 V) *	-----	0- 50 ma
−VLSW (−12 V) *	-----	0-200 ma

* Under Program Control

Table A-8 Option Electrical Requirements

Option Electrical Requirements (Max. in Amperes)									
Power Supply Voltage	Optional Processor	EMV/PROM Adaptor	Multimodule Adaptor	iSBX 350	iSBX 351	iSBX 251	iSBX 488	EMV	iUP
+5 volts	1.0	0.3	0.6	0.62	0.53	0.37	0.6	2.5	0.7
+12 volts	–	0.16	–	–	0.03	0.4	–	–	0.85
−12 volts	–	0.05	–	–	0.03	–	–	–	0.4
Note: Maximum option power requirements must no exceed 33.6 watts for any configuration.									

APPENDIX B
ERROR INDICATIONS

Command Entry Error Messages

When an error is made in entering a command line, the Command Line Interpreter (CLI) displays a message based on the error number returned by ISIS routines. Error numbers are explained in the next section. The CLI then returns the ISIS prompt so another command can be entered.

The general form of error messages is:

<string>, <message>

where

<string> is the part of the command line in which the error was detected.

<message> is the error message.

Error messages are listed below in alphabetical order in a standard format as follows:

LINE 1 IS THE <message> AS IT APPEARS ON THE DISPLAY SCREEN.
: Indented below the message are possible reasons for the error occurring, actions taken by the system due to the error, and possible actions open to the user to recover from the error.

The <string> part of the display is not shown in the list below. It precedes the <message> and varies depending on what was entered. For other messages not listed below, see the description of the particular command in which the message occurred.

ILLEGAL DISK LABEL
: The label supplied is not valid for a disk label. See the IDISK command in Chapter 5 for rules on disk labels.

ILLEGAL SUBMIT PARAMETER
: An error was made in the actual parameter to be substituted for a formal parameter in a command sequence file. See the SUBMIT command in Chapter 5.

INCORRECTLY SPECIFIED FILE
: The pathname entered was not in the correct format. Check the format for pathnames in Chapter 5.

INVALID SYNTAX
: There is an error in the command as entered. Check the command format in Chapter 5.

MODE IS MISSING
: On the SERIAL command, the parameter that specifies the mode, A for asynchronous or S for synchronous, is missing.

MODULE NOT AVAILABLE
An attempt was made to ATTACH a multimodule row that was already attached.

MULTI-MODULE ROW MUST BE 0 OR 1
The row specified on the ATTACH or DETACH command was not 0 or 1.

NO SUCH FILE
The file specified was not found in the directory for the disk specified.

PARAMETER TOO LONG
The parameter on the SUBMIT command line exceeded the maximum of 31 characters. See the SUBMIT command in Chapter 5.

TOO MANY PARAMETERS
More actual parameters were supplied than were defined or the maximum of 10 actual parameters on the SUBMIT command line was exceeded. See the SUBMIT command in Chapter 5.

UNRECOGNIZED SWITCH
Certain options (switches) can be used depending on the ISIS command entered. Check the specific command format in Chapter 5 for valid options.

ISIS-PDS Exception and Error Handling

One of the capabilities that an operating system provides is a uniform method of handling error conditions. Resident ISIS routines, i.e., system calls, detect two types of errors: fatal and non-fatal. In addition, console interface routines detect errors in command entries. Every error is designated by a decimal number as follows:

1-99 inclusive	Resident ISIS routine error numbers
100-199 inclusive	Reserved for User Programs
200-255 inclusive	Console interface routine error numbers

These numbers are listed and explained in a later section of this appendix.

Non-Fatal Errors

A non-fatal error is one from which recovery is possible. A non-fatal error results in the appropriate error number being returned to the program in which the error was found. The program can then take the proper corrective action, i. e., display a message, close all open files, etc.

An example of a non-fatal error is a pathname typed in an invalid format. The program requiring the pathname can repeatedly prompt until it receives a valid pathname or it can display an error message and return to the ISIS prompt so further commands can be entered.

Non-Fatal Error Numbers Returned to Programs by ISIS System Calls

OPEN	3, 4, 5, 9, 12, 13, 14, 22, 23, 25, 28, 63
READ	2, 8
WRITE	2, 6
SEEK	2, 19, 20, 27, 31, 35

RESCAN	2, 21
CLOSE	2
DELETE	4, 5, 13, 14, 17, 23, 28, 32
RENAME	4, 5, 10, 11, 13, 17, 23, 28
ATTRIB	4, 5, 13, 23, 26, 28
CONSOL	None; all errors are fatal
WHOCON	None
ERROR	None
LOAD	3, 4, 5, 12, 13, 22, 23, 28, 34
EXIT	None
SPATH	4, 5, 23, 28

Fatal Errors

A fatal error is one from which no recovery is possible. A fatal error results in the program in which the error was detected being terminated and a message being displayed on the cold start console output device. If possible, the CLI is reloaded and executed. Otherwise, the system must be hardware reset by pressing the RESET key. An example of a fatal error is a disk hardware error.

The format of the fatal error message displayed by ISIS is:

```
ERROR <nnn>, USER PC <xxxx>
```

where

<nnn> is the error number displayed in decimal.

<xxxx> is the contents of the program counter when the error occurred displayed in hexadecimal.

As a general rule, if it is possible to reload the CLI, the ISIS prompt is given on the display line following the error message. The only exception to this rule is error 24 which displays a further message. See error 24 in the following section for details.

To receive help in diagnosing a fatal error, type HELP <n> where <n> is the error message number that is displayed. See the HELP command in Chapter 5 for further information.

Fatal Error Numbers Returned as Messages by ISIS System Calls

OPEN	1, 7, 24, 30, 33
READ	24, 30, 33
WRITE	7, 24, 30, 33
SEEK	7, 24, 30, 33
RESCAN	33
CLOSE	33
DELETE	1, 24, 30, 33
RENAME	1, 24, 30, 33
ATTRIB	1, 24, 30, 33
CONSOL	1, 4, 5, 12, 13, 14, 22, 23, 24, 28, 30, 33, 63
WHOCON	33
ERROR	33
LOAD	1, 15, 16, 24, 30, 33
EXIT	None
SPATH	33
ATTACH	33, 60, 61
DETACH	33, 60, 61

Console Interface Errors

If an error is made in entering a command line, the CLI displays a message based on the decimal error number returned by the console interface routines. The CLI then returns the ISIS prompt, so another command can be entered.

The error numbers returned by the console interface routines are listed and explained in the following section.

Error Messages in Numeric Order

In the following list, the error numbers preceded by an uppercase F are always fatal errors. If the error number is not preceded by an uppercase F, it is a non-fatal error except when it is returned by the CONSOL system call. All CONSOL errors are fatal errors. See the charts on the preceding pages.

Resident ISIS Routines

0 No error was detected by the ISIS resident routine.

F 1 NO FREE BUFFER. The memory area from 3000H to program origin is used for I/O buffers. Too few buffers were allocated to meet the current request in addition to earlier requests or the number of buffers allocated exceeded the maximum limit of 19 allowed buffers. See Chapter 8 for information on how to allocate buffers.

2 ILLEGAL AFTN ARGUMENT. Illegal Active File Table Number (AFTN) argument supplied. For example, the file being read was not opened. AFTNs are described in Chapter 8.

3 TOO MANY FILES OPEN. At most, six files can be active at one time. One of the six files currently opened must be closed before another can be opened.

4 INCORRECTLY SPECIFIED FILE. For example, the number of characters entered for the filename might have exceeded the maximum of six allowed characters. Filename conventions are discussed in Chapter 5.

5 UNRECOGNIZED DEVICE NAME. For example, the device name :PR: might have been entered for printer instead of :LP:. Device names are discussed in Chapter 5.

6 ATTEMPT TO WRITE TO INPUT DEVICE. Only output devices can be written to. See Chapter 5.

F 7 INSUFFICIENT DISK SPACE. The disk is full. Make sure that the intended disk was actually specified.

8 ATTEMPT TO READ FROM OUTPUT DEVICE. Some devices are output only and cannot be read. See Chapter 5.

9 DISK DIRECTORY FULL. There is no room on the specified disk's directory to add a filename. The limit is 240 files for a diskette directory and 48 files for a bubble memory directory.

10 NOT ON SAME DISK. Pathnames do not specify the same disk. The RENAME system call requires two pathnames on the same device. See Chapter 8 for information on the RENAME system call.

11 FILE ALREADY EXISTS. A file cannot be given a name already in use. Check the spelling of the filename specified. See Chapter 8 for information on the RENAME system call.

12 FILE IS ALREADY OPEN. Only the Console Input and the Console Output can be opened more than one time. Check the spelling of the pathname specified. See Chapter 8 for information on the OPEN system call.

13 NO SUCH FILE. The specified filename could not be found in the directory of the disk in the specified drive. Make sure that the drive and the filename were correctly entered.

14 WRITE PROTECTED. A write protected file was encountered on WRITE, RENAME, DELETE. The write protect attribute was set on a file to be written.

F 15 CANNOT LOAD INTO ISIS AREA. Attempt to load memory reserved for ISIS, i.e., below 3000H. This operation is not allowed.

F 16 ILLEGAL FORMAT RECORD. An attempt was made to load a file that was not in the absolute module format required. The filename may have been mistyped.

17 NOT A DISK FILE. An attempt was made to RENAME or DELETE a non-disk file. Disk pathnames are discussed in Chapter 5.

18 ILLEGAL ISIS COMMAND. This error results when an ISIS system call is made with an illegal command number. See Chapter 8 for information on system calls.

19 ATTEMPTED SEEK ON A NON-DISK FILE. Seeks on physical devices other than a disk drive or the Byte Bucket are not allowed. See Chapter 8 for information on the SEEK system call.

20 ATTEMPTED BACK SEEK TOO FAR. MARKER is already set to zero indicating the beginning of the file. See Chapter 8 for information on the SEEK system call.

21 CAN'T RESCAN. An attempt was made to RESCAN a file not opened for line editing. See Chapter 8 for information on the OPEN and RESCAN system calls.

22 ILLEGAL ACCESS MODE TO OPEN. Only 1, 2, and 3 are valid for input (read), output (write), and update (both read and write) respectively. See Chapter 8 for information on the OPEN system call.

23 MISSING FILENAME. A filename was expected but was not supplied. See Chapter 8.

F 24 DISK ERROR. When error number 24 occurs, a second message is displayed (in addition to the ERROR 24 USER PC <xxxx> message) as follows:

DRIVE <n> STATUS = <a> <b> <c> <d> <e> <f> <g>

where

<n> represents the drive number of the disk with the hardware error.

The status bytes <a> - <g> are displayed only for disk drives. They are taken from data stored on the 8272 Disk Controller chip. Refer to Table B-1 for further information on the status registers.

<a> Contents of status register 0. See Table B-1.

<b> Contents of status register 1. See Table B-1.

<c> Contents of status register 2. See Table B-1.

<d> The cylinder number being accessed on the disk. A cylinder corresponds to the two tracks on opposite sides of the disk surface that can both be accessed without moving the read/write head of the disk drive. The cylinder number corresponds to the track number.

<e> The head number being accessed. The head number can be either 0 or 1 depending on which side of the disk is being accessed.

<f> Sector number. See Table B-1.

<g> The number of bytes per sector. A value of 0 represents 128 bytes/sector and 1 represents 256 bytes/sector.

For bubble memory, only one byte of status is displayed from the 7220 Bubble Memory Controller chip.

<a> Contents of the status register. See Table B-2 for more information.

Table B-1 8272 Status Registers

Main Status Register

BIT NUMBER	DESCRIPTION
0	If set, FDD (Flexible Disk Drive) number 0 is in the Seek mode.
1	If set, FDD number 1 is in the Seek mode.
2	If set, FDD number 2 is in the Seek mode.
3	If set, FDD number 3 is in the Seek mode.
4	If set, a read or write command is in progress.
5	If set, the FDC (Flexible Disk Controller) is in the non-DMA mode. This bit is set only during the execution phase in non-DMA mode. Transition to 0 state inidicates that execution phase has ended.
6	Indicates direction of data transfer between FDC and Data Register. If set, then a transfer is from the Data Register to the Processor. If not set, then a transfer is from the Processor to the Data Register.
7	Indicates Data Register is ready to send or receive data to or from the Processor.

Table B-1 8272 Status Registers (continued)

8272 Status Register 0

BIT NUMBER	DESCRIPTION
0	Used with Bit 1 to indicate the Drive Unit Number, 0-3, at the interrupt.
1	Used with Bit 0 to indicate the Drive Unit Number, 0-3, at the interrupt.
2	Indicates the state of the head at the interrupt.
3	When the FDD is in the not-ready state and a read or write command is issued, this flag is set. If a read or write command is issued to Side 1 of a single sided drive, then this flag is set.
4	When a fault Signal is received from the FDD, or it the Track 0 Signal fails to occur after 77 step pulses (recalibrate command) then this flag is set.
5	When the FDC completes the SEEK command, this flag is set to 1 (High).
6	bit 6=1 and bit 7=1 Abnormal termination because during command execution, the ready signal from the FDD changed state. bit 6=0 and bit 7=1 Invalid Command which was issued was never started.
7	bit 6=1 and bit 7=0 Abnormal termination of command. Execution of command was started, but was never successfully completed. bit 6=0 and bit 7=0 Normal termination of command. Command was completed and properly executed.

Table B-1 8272 Status Registers (continued)

8272 Status Register 1

BIT NUMBER	DESCRIPTION
0	When the FDC cannot detect the ID Address Mark after encountering the index hole twice, this flag is set. When the FDC cannot detect the Data Address Mark or the Deleted Data Address Mark, this flag is set. Also, at the same time, the Missing Address Mark in the Data Field of Status Register 2 is set.
1	During execution of WRITE DATA, WRITE DELETED DATA, or Format A Cylinder Command, if the FDC detects a write protect signal from the FDD, then this flag is set.
2	During execution of READ DATA, WRITE DELETED DATA, or SCAN Command, if the FDC cannot find the Sector specified in the IDR Register, this flag is set. During execution of the READ ID Command, if the FDC cannot read the ID field without an error, then this flag is set. During execution of the READ A Cylinder Command, if the starting sector cannot be found, then this flag is set.
3	Not used. This bit is always 0 (low).
4	When the FDC is not serviced by the main system during data transfers within a certain time interval, this flag is set.
5	When the FDC detects a CRC error in either the ID field or the data field, then this flag is set.
6	Not used. This bit is always 0 (low).
7	When the FDC tries to access a sector beyond the final sector of a cylinder, this flag is set.

Table B-1 8272 Status Registers (continued)

8272 Status Register 2

BIT NUMBER	DESCRIPTION
0	When the data is read from the medium, if the FDC cannot find a Data Address mark or a Deleted Data Address Mark, then this flag is set.
1	This bit is related to bit 2 of Status Register 1. When the content of C on the medium is different from that stored in the IDR and the content of C is FF, then this flag is set.
2	During execution of the SCAN Command, if the FDC cannot find a sector on the cylinder which meets the condition, then this flag is set.
3	During execution of the SCAN Command, if the condition of "equal" is satisfied, this flag is set.
4	This bit is related to bit 2 of Status Register 1. When the contents of C on the medium is different from that stored in the IDR, this flag is set.
5	When the FDC detects a CRC error in the data field, then this flag is set.
6	During execution of the READ DATA or SCAN Command, if the FDC encounters a Sector which contains a Deleted Data Address Mark, this flag is set.
7	Not used. This bit is always 0 (low).

Table B-2 7220 Status Registers

BIT NUMBER	DESCRIPTION
0	The FIFO register on the 7220 has data available to be read or written.
1	When set, a parity error has been detected in the last data byte sent to the 7220.
2	When set, an uncorrectable error has been detected in the last data block transferred.
3	When set, a correctable error has been detected in the last block of data transferred.
4	When set, indicates that a timing error has occurred or that the host system has failed to keep up with the 7220. Also, indicates that no bootloop sync word was found during initialization or a write bootloop command was issued when the WRITE BOOTLOOP ENABLE bit was not set.
5	When set, the last command was unsuccessful and was not completed.
6	When set, the last command was successfully completed.
7	When set, indicates that the 7220 is in the process of executing a command. When not set, the 7220 is ready to accept a command.

An example of a disk error 24 is given to show how to interpret these error codes. Suppose that, after running the DIR 1 command to display the directory of drive 1, the following error message is displayed on the screen.

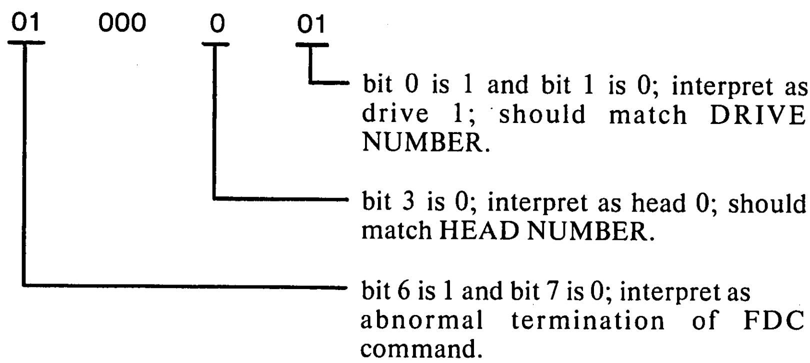
```
ERROR 24   USER PC 4EDD
DRIVE  1   STATUS= 41   1 1 27 0 1 1
```

From the first line of display, the error is error number 24 and the program counter is pointing to address 4EDDH. The second line of display can be summarized as follows:

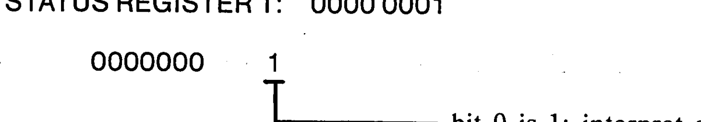
PHYSICAL DRIVE NUMBER:	1
STATUS REGISTER 0:	41H
STATUS REGISTER 1:	1
STATUS REGISTER 2:	0
TRACK NUMBER:	27H
HEAD NUMBER:	0
SECTOR NUMBER:	1
FORMAT:	256 bytes/sector

The three status registers can be written in binary with the following interpretations of the bits. Bits not needed for the interpretation are not described.

STATUS REGISTER 0: 0100 0001



STATUS REGISTER 1: 0000 0001



bit 0 is 1; interpret as either missing Sector (ID) Address Mark *or* as missing Data Address Mark.

STATUS REGISTER 2: 0000 0001

0000000 1

bit 0 is 1; interpret as missing Data Address Mark.

If there is a missing Data Address Mark, it is possible that the disk has not been formatted as in this case.

25 ILLEGAL ECHO FILE. An echo file must have an AFTN between 0 and 255 and must already be open for output. See Chapter 8 for information on echo files.

26 ILLEGAL ATTRIBUTE IDENTIFIER. This error refers to the second parameter in the ATTRIB system call. Only 0, 1, 2, or 3 are valid meaning invisible, system, write protect, or format attributes respectively. See Chapter 8 for more information on the ATTRIB system call.

27 ILLEGAL SEEK COMMAND. This error refers to the MODE parameter in the SEEK system call. See Chapter 8 for more information.

28 MISSING EXTENSION. An expected file extension was not supplied.

F 29 EOF ON CONSOLE INPUT. Premature End of File (EOF) detected on the console input device.

F 30 DRIVE NOT READY. The drive specified was not ready. For example, a DIR command was given and no disk was in the drive. This is a fatal error and the system must be reinitialized.

31 CAN'T SEEK ON WRITE ONLY FILE. SEEKs can only be executed on read or update files. See Chapter 8 for further information on the SEEK system call.

32 CAN'T DELETE OPEN FILE. A file must be closed before it can be deleted.

F 33 ILLEGAL SYSTEM CALL PARAMETER. A parameter specified in a system call was outside the valid range for that parameter.

34 BAD RETURN SWITCH ARGUMENT TO LOAD. The return switch parameter in the LOAD system call was not set to 0, 1, or 2. See Chapter 8 for further information on the LOAD system call.

35 SEEK PAST EOF. An attempt was made to extend a file opened for input by seeking past the End of File (EOF). See Chapter 8 for further information on the SEEK system call.

60 MODULE ALREADY ASSIGNED. An attempt was made to attach a multimodule row that is already attached to other processor.

61 MODULE ALREADY ASSIGNED TO BUBBLE. An attempt was made to attach a multimodule row that is contains a bubble memory multimodule.

62 ILLEGAL TRACK ADDRESS. An illegal track address was given on a direct disk access.

63 FILE OPEN FOR WRITE OR UPDATE BY OTHER PROCESSOR. An attempt was made to open or write to a file currently being written by the other processor.

Console Interface Routines

201 UNRECOGNIZED SWITCH. Certain options (switches) can be used depending on the ISIS command entered. Check the specific command format in Chapter 5 for the valid options.

202 UNRECOGNIZED DELIMITER. A character was encountered that was not valid for a name or a delimiter.

203 INVALID SYNTAX. There is an error in the command as entered. Check the command format in Chapter 5.

206 ILLEGAL DISKETTE LABEL. The label supplied is not valid for a disk label. See the IDISK command in Chapter 5 for rules on disk labels.

207 NO END STATEMENT. The job file is missing an end statement.

208 CHECKSUM ERROR. There may be a format error in the file read.

209 RELO FILE SEQUENCE ERROR. An illegal record sequence was detected in an object module file.

210 INSUFFICIENT MEMORY. The amount of RAM required is not available.

211 RECORD TOO LONG. A record longer than allowed was encountered.

212 ILLEGAL RELO RECORD. A bad record was encountered in an object module.

213 FIXUP BOUNDS ERROR. An illegal fixup record was specified in an object module file.

214 ILLEGAL SUBMIT PARAMETER. An error was made in the actual parameter to be substituted for a formal parameter in a command sequence file. See the SUBMIT command in Chapter 5.

215 ARGUMENT TOO LONG. The number of characters in an actual argument for a SUBMIT file cannot exceed 31 characters. See the SUBMIT command in Chapter 5.

216 TOO MANY PARAMETERS. More actual parameters were supplied than were defined in the SUBMIT file. See the SUBMIT command in Chapter 5.

217 OBJECT RECORD TOO SHORT. This error may be caused by an I/O error in the file to be loaded.

218 ILLEGAL RECORD FORMAT. The record format did not match the Intel standard for object module records.

219 PHASE ERROR. The expected input for a step in the translation process was not correctly supplied for the LINK command.

220 NO END-OF-FILE. There is an error in the internal format of the specified object module file. Retranslate and relink the source module.

221 SEGMENT OVERFLOW. The output segment cannot be greater than 64K bytes. Relink the modules.

222 UNRECOGNIZED RECORD. There is an error in the internal format of the specified object module file. Retranslate and relink the source module.

223 BAD FIXUP RECORD POINTER. There is an error in the internal format of the specified file. Retranslate and relink the source module.

224 ILLEGAL RECORD SEQUENCE. There in an error in the internal format of the specified object module file. Retranslate and relink the source module.

227 COMMAND REQUIRES '('. There is a missing left parenthesis in the command line. Re-enter the command line correctly.

230 DUPLICATE SYMBOL FOUND. An attempt was made to add a symbol that already exists.

231 FILE ALREADY EXISTS. The file specified in a CREATE command already exists.

232 UNRECOGNIZED COMMAND. Check to make sure that the command was correctly entered.

Diagnostic Errors

During system initialization, errors are indicated in two possible ways:

- By the four diagnostic LED indicators
- By a message on the display screen

LED Indicators

The four LED indicators are shown in figure B-1. All four indicators are lit when the system is initially powered on. Then, they are turned off one at a time as the component being tested passes the test. If the system fails to initialize, the problem can be determined by checking to see which LED indicators are still lit.

The LEDs are located on the base processor circuit board. By holding open the swinging door for the plug in module, these indicators can be seen as shown in figure B-1 if the Plug-in Adapter Board is installed. If the Adapter Board is not installed, the LEDs will appear the same except that the cutout in the Adapter Board is not present.

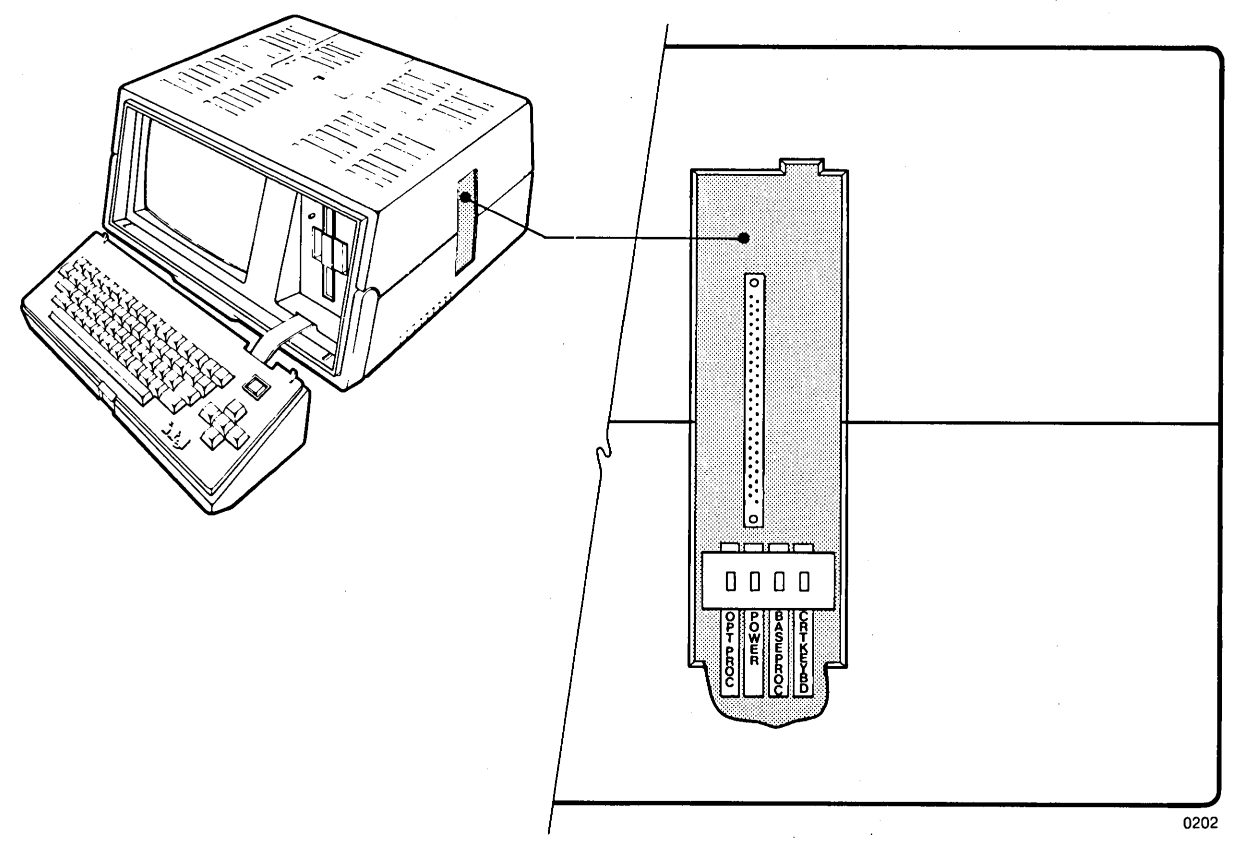


Figure B-1 Diagnostic LED Indicators

From left to right, they are the optional processor indicator, the power on indicator, the base processor indicator, and the CRT/Keyboard indicator. The following chart summarizes the possible error conditions.

OPT PROC	POWER	BASE PROC	CRT/ KYBD	**Failure**
OFF	OFF	OFF	OFF	Power Supply
ON	ON	ON	ON	CRT/Keyboard Section, Main Board
ON	ON	ON	OFF	Processor Section, Main Board
ON	ON	OFF	OFF	Optional Processor Board
OFF	ON	OFF	OFF	Failure indicated by error message on CRT screen.

Diagnostic Error Messages

The following error messages may occur during initialization of the system.

BUBBLE BOOT FAILED

After ten retries, Track 0 of the bubble memory device could not be read.

FDD FAILED TO BOOT

After ten retries, Track 0 of drive 0 could not be read successfully.

FDD FAILED TO RECAL
After two recalibrate commands were issued, the flexible disk drive failed to turn in the proper status.

NO BOOT DEVICE
There is no disk in drive 0, and no bubble memory controller is installed in connector J1 of the Multimodule Adapter Board. The system cannot be initialized.

NON-SYSTEM DISK
The disk in drive 0 is not a system disk; there are no further disk operations.

BOOT FROM BUBBLE? (Y OR N)
If there is not a disk in drive 0 and a bubble memory controller is available in connector J1 of the Multimodule Adapter Board, the system prompts with this message to initialize from the bubble memory device. Type Y on the keyboard to continue. Type N to retry the disk. Any other response causes the warning buzzer to sound.

RAM ERROR <addr>/<ee>/<aa>
The program memory failed a pattern test at address <addr> where <ee> is the expected data and <aa> is the actual data displayed in hexadecimal.

ROM CHECKSUM ERROR <ee>/<aa>
The ROM pattern check failed where <ee> is the expected pattern and <aa> is the actual pattern displayed in hexadecimal.

WAITING FOR CONTROL OF FDC
The processor has timed out in the semaphore wait loop. If nothing else happens, there is a problem with the semaphore. Another message is displayed if the semaphores are working. This message normally appears briefly during a system initialization.

WAITING FOR CONTROL OF MMIO-0
The processor has timed out in the semaphore wait loop. If nothing else happens, there is a problem with the semaphore. Another message is displayed if the semaphores are working.

WAITING FOR CONTROL OF MMIO-1
The processor has timed out in the semaphore wait loop. If nothing else happens, there is a problem with the semaphore. Another message is displayed if the semaphores are working.

Confidence Test

This section describes the operation of the confidence test for the iPDS system. It is not necessary to run this test prior to using the system. However, it is recommended that the test be run when the system is initially installed. See the hands-on initialization session in Chapter 3 for an example of running the confidence test.

The confidence test aids in troubleshooting the system if problems occur. Successful execution of the test demonstrates the complete operation of the system.

The confidence test assumes that ISIS-PDS is loaded and running. The test runs as a utility under the operating system and provides a set of subcommands that help verify the system.

If ISIS-PDS cannot be successfully loaded, refer to the description of the power on diagnostic errors in a previous section of this appendix. The power on diagnostics provide an 85% confidence check of the system.

The confidence test verifies the following parts of the main processor board:

- CPU
- Program Memory
- Keyboard/CRT Interface
- Programmable Timer
- Line Printer Interface
- Serial I/O Interface
- Disk Controller
- Disk Drives
- Semaphores (used in dual processing systems)

The confidence test also verifies the following parts of the optional processor board:

- CPU
- Program Memory
- Keyboard/CRT Interface
- Disk Controller
- Disk Drives
- Semaphores (used in dual processor systems)

Also, the confidence test checks:

- Magnetic Bubble Memory Multimodules
- PROM Programmer Plug-in Modules

The test for each emulator plug-in module is included with the emulator. It is described in the appropriate emulator user manual.

To perform the confidence test:

1. Load the confidence test command under ISIS-PDS by entering the PCONF command line.
2. Initialize the confidence test with the INIT CONPDS command.
3. Enter any of the nine confidence test commands to perform the 17 different tests.

These steps are described in detail in the following sections. Also, see the hands-on demonstration of the confidence test in the initialization session in Chapter 3. Notational conventions used for command descriptions are explained in Chapter 5.

PCONF Command

The PCONF command is shown in the following:

```
A0>PCONF
ISIS-PDS PCONF Vx.y
*
```

The PCONF command is entered when the ISIS-PDS prompt is displayed indicating that the operating system will accept a command. It loads the confidence test program which displays a sign-on message and an asterisk prompt allowing the user to enter the INIT CONPDS command.

INIT CONPDS Command

The INIT CONPDS command is shown in the following:

```
*INIT CONPDS
iPDS CONFIDENCE TESTS, Vx.y
USER RETURN
```

After the INIT CONPDS command is entered, another sign-on message and asterisk prompt are displayed allowing the user to enter any of the nine confidence test commands.

Confidence Test Commands

The nine confidence test commands allow the user to specify the test sequence for the 17 confidence tests and to control the reporting of results from the tests. The nine commands are:

Command	**Abbreviation**	
CLEAR	CLE	
ERROR	ERR	
DESCRIBE	DES	
EXIT		EXI
IGNORE	IGN	
LIST		LIS
RECOGNIZE	REC	
SUMMARY	SUM	
TEST		TES or T

Any command consisting of four or more letters can be abbreviated to the first three letters. Additionally, the TEST command can be abbreviated to T. The TEST command is the command used to run the confidence tests.

Each of these commands is described in one of the following sections. The 17 confidence tests are described under the TEST command.

Numeric values required in these commands can be given in hexadecimal (H), decimal (T), octal (Q), or binary (Y) by attaching the appropriate suffix to the value. The default base (if no suffix is given) is hexadecimal.

CLEAR Command

The format of the CLEAR command is:

```
CLEAR  {[<test number> [,<test number>]] ...}
       {<test number> TO <test number>      }
```

where

<test number> specifies one of the 17 confidence tests by number (00H through 10H). Enter the hexadecimal value of the test number.

The CLEAR command sets the execution count and error count to zero for the test or tests specified. The execution count is the number of times the specified test or tests have been run and the error count is the number of errors detected. If no tests are specified, the CLEAR command clears the counts for all the tests. The CLEAR command does not affect the status of a test. The status of a test indicates whether the test is ignored or recognized. See the IGNORE and RECOGNIZE commands.

To clear the execution count and error count for Tests 3, 4, and 5, enter:

```
CLEAR 3,4,5
```

DESCRIBE Command

The format of the DESCRIBE command is:

```
DESCRIBE  {[<test number> [,<test number>]] ...}
          {<test number> TO <test number>      }
```

where

<test number> specifies one of the 17 confidence tests by number (00H through 10H). Enter the hexadecimal value of the test number.

The DESCRIBE command displays the name and the status of the test or tests specified. The status of a test indicates whether the test is ignored or recognized by the system. See the IGNORE and RECOGNIZE commands. If no tests are specified, the name and status are given for all the tests.

To describe tests 3, 4, 5, and 6, when test 3 is being ignored by the system, enter the following command:

```
DESCRIBE 3 TO 6
00003H  LINE PRINTER TEST            **** IGNORED ****
00004H  SERIAL OUTPUT TEST
00005H  FDC SEMAPHORES
00006H  READY DRIVE DETERMINATION
```

Note that test numbers are displayed in hexadecimal.

ERROR Command

The format of the ERROR command is:

```
ERROR = [<n>]
```

where

<n> is either 0 to display pass/fail messages or 1 to suppress them.

The default setting is 0 to display all pass/fail messages.

To find out the current display status, enter:

```
ERROR
```

To suppress pass/fail messages for the confidence tests, enter the following command before running the tests with the TEST command.

```
ERROR = 1
```

EXIT Command

The format of the EXIT command is:

```
EXIT
```

The EXIT command ends the test session and returns control to ISIS-PDS.

IGNORE Command

```
IGNORE {{<test number> [,<test number>]} ...}
       {<test number> TO <test number>      }
```

where

<test number> specifies one of the 17 confidence tests by number (00H through 10H). Enter the hexadecimal value of the test number.

The IGNORE command allows the user to specify a test or tests to be ignored. The ignored tests will not be run. Tests that are not applicable to the optional processor are automatically ignored when they are run on the optional processor. Also, the tests that check Magnetic Bubble Memory Multimodules and the PROM Programmer Plug-in Modules are automatically ignored by the system. The user must enter the RECOGNIZE command to include the tests for the Bubble Memory and PROM Programmer options.

For example, to run all tests except 5, 6, and 8, enter the command:

```
IGNORE 5,6,8
```

LIST Command

The format of the LIST command is:

```
LIST <pathname>
```

where

<pathname> specifies a valid ISIS-PDS device to receive any output from the tests. See Chapter 5 for an explanation of ISIS pathnames.

The LIST command causes a copy of all subsequent output from the tests to be sent to <pathname>. The output includes prompts, user entered input, and error messages. The output is also displayed at the current console device (on the display screen). If the <pathname> is :CO:, there is no effect, since the console receives all the output anyway.

To copy all output to the line printer, enter the command:

```
LIST :LP:
```

RECOGNIZE Command

The format of the RECOGNIZE command is:

```
RECOGNIZE {[<test number> [,<test number>]} ...}
          {<test number> TO <test number>     }
```

where

<test number> specifies one of the 17 confidence tests by number (00H through 10H). Enter the hexadecimal value of the test number.

The RECOGNIZE command allows the user to declare the specified test or tests to be recognized and run. This command can be used to include the test for the Bubble Memory Multimodule or the PROM Programmer Plug-in Module if these options are available in the system.

Assuming that tests 5, 6, 7, and 8 are currently ignored, enter the following command to recognize 5, 7, and 8. Test 6 will continue to be ignored.

```
RECOGNIZE 5,7,8
```

SUMMARY Command

The format of the SUMMARY command is:

```
SUMMARY {[<test number> [,<test number>]} ...} [EO]
        {<test number> TO <test number>     }
```

where

<test number> specifies one of the 17 confidence tests by number (00H through 10H). Enter the hexadecimal value of the test number.

EO — specifies that only the tests with a non-zero error count will be summarized.

The SUMMARY command displays the following information for each test or tests specified:

- Test number in hexadecimal
- Number of times the test was executed
- Number of times an error occurred during the test
- Status of the test (ignored or recognized)

The information displayed is accumulated since the last INIT CONPDS or CLEAR command. If no test or tests are specified, summaries are displayed for all tests. If EO is specified, a summary is displayed only for those tests with a non-zero error count.

To display a summary of tests 3, 4, and 5, enter the command:

```
SUMMARY 3 TO 5
```

The following display will appear on the CRT display screen.

```
00003H LINE PRINTER TEST EXECUTED 00002T TIMES, 00000T FAILS
00004H SERIAL OUTPUT     EXECUTED 00002T TIMES, 00001T FAILS
00005H FDC SEMAPHORES    EXECUTED 00002T TIMES, 00000T FAILS
```

Note that all test numbers are in hexadecimal and error counts are in decimal.

To display a summary of tests 3, 4, and 5 only if an error is detected, enter the command:

```
SUMMARY 3 TO 5 EO
00004H USART TEST EXECUTED 00002T TIMES, 00001T FAILS
```

TEST Command

The format of the TEST command is:

```
TEST [ { <test number> [,<test number>] ... } ]  [ { ON ERROR        } ]
     [ { <test number> TO <test number>     } ]  [ { ON NOERROR      } ]
                                                 [ { COUNT <nnnn>    } ]
                                                 [ { FOREVER         } ]
```

where

<test number> — specifies one of the 17 confidence tests by number (00H through 10H). Enter the hexadecimal value of the test number.

ON ERROR — specifies that the tests will be executed repeatedly if one or more errors are detected.

ON NOERROR — specifies that the tests will be executed repeatedly until any errors are detected.

COUNT <nnnn> specifies that the tests will be executed in numerical order for <nnnn> times. If <nnnn> = 0 or if COUNT is specified with no <nnnn>, no tests will be executed. The maximum value of <nnnn> is 65,535 in decimal.

FOREVER specifies that the tests will be executed in numerical order and repeat until the user presses the ESC key. Tests will not stop on failures and all errors will be logged in the error table. Errors can be displayed with the SUMMARY command.

The TEST command loads and runs the test or tests specified by the user. The tests are executed in numerical order regardless of the order in which they were entered. If no test number is specified, all currently recognized tests are executed.

To run all tests (0 through 10H), enter the command:

TEST

To run tests 2, 4, 8, and 9, enter the command:

TEST 4,9,2,8

Note that the commands are loaded and run in numerical order.

To run tests 0 through 8 and repeat the failing test only if an error is detected, enter the command:

TEST 0 TO 8 ON ERROR

To run test 7 and repeat until any error is detected, enter the command:

TEST 7 ON NOERROR

To run tests 3 and 5 and repeat forever, enter the command:

TEST 3,5 FOREVER

To run tests 2 and 7 and repeat 5 times, enter the command:

TEST 2,7 COUNT 5

The following tests are available:

Test Number	Processor	Function
0	A, B	CPU Test
1	A, B	CRT Interface Test
2	A	Programmable Timer Test
3	A	Line Printer Interface Test
4	A	Serial Interface Test
5	A, B	Disk Controller Semaphore Test
6	A, B	Disk Drive Recalibrate and Ready Test
7	A, B	Disk Drive Seek and Read Test
8	A	Serial Loopback Test
9	A, B	Disk Format Test
A	A, B	Formatted Disk Data Read Test
B	A, B	Disk Random Seek/Write/Read Test

C	A, B	Keyboard Echo Test
D	A, B	Bubble Memory Seek and Read Test
E	A, B	Bubble Memory Random Seek/Write/Read Test
F	A	PROM Programmer Plug-in Module Test
10	A, B	32K RAM Relocating Random Data Test

The letter A indicates that the test can run on Processor A (the base processor), and B indicates that the test can run on Processor B (the optional processor). The following tests are ignored when the confidence test is first run and must be included with the RECOGNIZE command before they can be run: 3, 8, 9, A, B, C, D, E, F, and 10.

The ignored tests require some user interaction while running. If a response is not received in a predetermined period of time, the test times out and is not executed.

The following tests require the use of the semaphore: 2, 5, 6, 7, 9, A, B, D, and E. If these tests are run on both processors at the same time,one processor will get failures due to semaphore timeouts.

Each of these tests is described in the following sections.

Test 0 - CPU Test

Test 0 verifies the operation of the 8085 microprocessor by executing the instruction set and monitoring the results.

Test 1 - CRT Interface Test

Test 1 first displays a line of all the printable ASCII characters. Then, one line of each character is displayed.

Test 2 - Programmable Timer Test

Test 2 loads the timer counters with a test count value and verifies that it counts down within the prescribed time interval based on a software timing loop. This test is only valid for the base processor, since the optional processor does not have a programmable timer.

Test 3 - Line Printer Interface Test

Test 3 sends a form feed, several lines of incrementing test data, and a line each of ASCII test characters to the line printer port. The printer must be in a ready condition when this test executes. Any other status condition, such as busy, not ready, or no printer attached causes the test to fail. This test only runs on the base processor, since the optional processor does not include a line printer interface.

Test 4 - Serial Interface Test

Test 4 verifies the operation of the 8251 USART controller chip on the base processor board. This test transmits a string of ASCII characters to verify that the transmitter portion of the 8251 USART controller chip is working. The receiver portion of the USART controller is tested in another test. This test only runs on the base processor, since the optional processor does not include a serial interface.

Test 5 - Disk Semaphore Test

Test 5 clears the disk controller semaphore and tests the drive-ready status at the semaphore port for a not true condition. It then sets the semaphore and tests the status for a true condition. Finally, it turns the semaphore off again and tests for a not true condition. The disk controller semaphore allows both processors to share the disk in dual processor systems.

Test 6 - Disk Drive Recalibrate and Ready Test

Test 6 initializes a table indicating which disk drives can be tested. Test 6 must be run once before running test 7. If the condition of a drive is changed during testing, test 6 must be run again to correct the table before running test 7.

Test 6 sets the disk controller semaphore and cycles through all four drives, issuing two recalibrate and sense-interrupt-status command sequences. After the second sequence, the command completion status and track-zero status are checked. If either is invalid, the drive is flagged in the table as not-ready.

NOTE

Tests 6 and 7 require ISIS-PDS formatted disks. The disk should be formatted with the IDISK command. These tests do not destroy any data on the disk.

If the status indicates that the drive is available, a read-ID command sequence is issued to determine if a disk is inserted in the drive. If no disk is inserted, the table entry for that drive is flagged not-ready. If the drive is available and a disk is inserted, the drive is flagged as ready.

After testing all four disk drives, if at least one is ready, the test is passed. An information message is displayed on the CRT indicating the status of the drives. If none of the four drives is ready, the test is failed. The disk controller semaphore is cleared at the end of the test.

Test 7 - Disk Drive Seek and Read Test

Test 7 sets the disk controller semaphore and verifies all drives flagged as ready in the table setup by test 6. Each drive should contain a disk previously initialized with the ISIS IDISK command. Each drive is tested 10 times with the following sequence:

- Seek to the last cylinder
- Issue read-ID command for heads 0 and 1
- Seek to track 1
- Issue a read-data for sector 4, head 0
- Verify that the data read at sector 4 is the data written there when the disk was initialized under ISIS.

Sector 4 of track 1 is the last sector of the file ISIS.LAB. It is not used as part of the disk volume identifier file, but it is initialized by the the IDISK command with sixteen contiguous repetitions of the string:

DIAGNOSTICSECTOR

If no drives are flagged as ready in the table setup by test 6, test 7 fails. At the end of test 7, the disk controller semaphore is cleared. The disk is not destroyed and can be used under ISIS-PDS.

NOTE

Tests 6 and 7 require ISIS-PDS formatted disks. The disk should be formatted with the IDISK command. These tests do not destroy any data on the disk.

Test 8 - Serial Loopback Test

Test 8 is run with the transmit data line of the 8251 USART controller chip tied back to the receive data line. These two lines can be tied with special loopback connector (use a DB25 connector and short Pins 2 and 3 to each other) or with the test switch on a modem.

Test 8 sends an incrementing data sequence from 00H to FFH to the transmit data line and tests the receiver input on the receive data line for the correct data.

Test 9 - Disk Format Test

Test 9 formats a disk on the drive or drives specified. Test 9 prompts the user to enter blank disks in any drive to be tested.

```
** WARNING - DISKS FORMATTED WITH THIS TEST ARE NOT
             COMPATIBLE WITH ISIS. **
- INSERT TEST DISKETTES
- HIT ESCAPE TO ABORT THIS TEST
- HIT ANY OTHER KEY TO CONTINUE
```

As soon as a disk is inserted into the drive to be tested, press any key other than ESCAPE.

Zero to four disks may be formatted at a time. The data written to the disk is the track number.

Note that sector numbers 1-16 (01H through 10H) are on Side 0 of the disk. Sector numbers 17-32 (11H through 20H) are on Side 1 of the disk.

Test A - Formatted Disk Data Read Test

Test A reads the data written to the disk or disks by test 9 using the drives that have diskettes inserted in them. This allows the disk drives to be tested for alignment compatability.

Test B - Disk Random Seek/Write/Read Test

Test B uses the same disk or disks formatted by test 9 for random seek, write, and read testing. The drive to be tested is the same drive used in test 9. A random number generator selects which of the specified drives to test and which cylinder and sector number to test. The random number then becomes the seed for the random data written to the sector. Test B runs until terminated by the user pressing the ESC key. The disk controller semaphore is cleared when the test is completed.

NOTE

The disk format used in tests 9, A, and B is not compatible with Tests 6 or 7 or with ISIS-PDS, and the disks used in these tests must be initialized with the IDISK command before being used under ISIS.

Test C - Keyboard Echo Test

Test C echoes the data entered from the keyboard to the CRT display to allow visual verification of the data. The test continues until the ESC key is pressed two times in a row.

Test D - Bubble Memory Seek and Read Test

Test D determines if either or both Bubble Memory Multimodules are present and sets the appropriate multimodule semaphores. Each bubble multimodule should have been initialized previously with the ISIS IDISK command. Then, each bubble is tested 10 times with the following sequence:

- Seek to page 140 which is the same as track 1 sector 4
- Issue a read-data command sequence for 4 pages (the entire sector 4)
- Verify that the data read at sector 4 is the data written by the IDISK command when the bubble memory was initialized

Sector 4 of track 1 is the last sector of the file ISIS.LAB. It is not used as part of the disk volume identifier file, but it is initialized by the the IDISK command with sixteen contiguous repetitions of the string:

DIAGNOSTICSECTOR

If no multimodule connectors contain bubble memory, test D fails. At the end of test D, the multimodule semaphores are cleared.

Test E - Bubble Memory Random Seek/Write/Read Test

Test E determines if either or both Bubble Memory Multimodules are present and then sets the appropriate multimodule semaphores. Then, it seeks to random sectors on the bubble memory. After seeking to random sectors, it writes and then reads random data at these sectors to verify that the controller and storage medium are operating correctly. This test runs until the user terminates it by pressing the ESC key. The semaphores are cleared when the test is ended. A screen message is displayed indicating read/write and page number test activity.

NOTE

The disk format used in test E is not compatible with Test D or with ISIS-PDS, and the bubble memory must be initialized with the IDISK command before being used under ISIS.

Test F - PROM Programmer Plug-in Module Test

Test F reads the personality program from the PROM Plug-in Module and computes the checksum on the program to verify the operation of the PROM plug-in module.

Test 10 - 32K RAM Relocating Random Data Test

Test 10H tests one half of the 64K bytes of RAM while the test resides in the other half. When one half is tested, the code relocates itself to the half that is already tested and tests the other half. Test 10H continues until the system is reinitialized. Each successful pass through a 32K bank of memory is indicated by printing an asterisk on the screen. Each error is displayed on the screen. The memory is tested with a random data pattern.

Confidence Test Error Messages

CPU TEST FAILED TEST GROUP: <xxx>

While executing the 8085A instruction set test, an invalid result was detected on the instruction set group <xxx>.

TIMER FAILURE - COUNTER <n> EXP: <xxxx>-<xxxx> ACT: <yyyy>

After doing a software timing loop, the actual contents of the timer counter (<yyyy>) did not match the expected contents range (<xxxx>-<xxxx>). The value of <n> can be 0 or 1 to specify which counter was being tested when the error occurred.

LP FAILURE - <cause>

The line printer test failed. Possible values of <cause> are NOT READY, BUSY TIMEOUT, or FAULT.

USART FAILURE - XMTR-RDY TIMEOUT

The transmitter ready signal failed to go high within an allotted time interval.

USART RCVD DATA ERROR

EXP: <ee> ACT: <aa>

The loopback data (<aa>) did not match the transmitted data (<ee>). The value following EXP is the data transmitted; the value following ACT is the data received.

FDC SEMAPHORE FAILED TO <cond> DRIVE-READY

While setting or clearing the disk controller semaphore, the status bit on the semaphore port failed to follow the signal. The value of <cond> can be either SET or CLEAR depending on the action being taken at the time of the failure.

FDC SEMAPHORE TIMEOUT

A test requiring the disk controller semaphore to be set failed, since the semaphore could not be set within 3 seconds.

DRV-0 <stat>, DRV-1 <stat> DRV-2 <stat> DRV-3 <stat>

This is a status message from test 6 indicating which drives were found ready to be tested by test 7. The value of <stat> can be either READY or N RDY for each drive.

NO DRIVES READY

No disk drives were found ready by test 6.

FDD SEEK FAILED ON DRIVE # <d>
EXP STATUS = <aa> <bb>
ACT STATUS = <aa> <bb>

The disk drive seek test failed. The drive number on which the test failed is shown in place of <d>. The expected status and actual status are displayed from status register 0 (<aa>) of the disk controller chip. The present cylinder number (<bb>) is also displayed. Refer to Table B-1 for the interpretation of the status.

```
FDD READ FAILED ON DRIVE #<d>, TRK=<t>, SEC=<s>, HD=<h>
FDC STATUS =<aa> <bb> <cc> <dd> <ee> <ff> <gg>
             STO  ST1  ST2  C    H    R    N
```

The disk drive read test failed. The first line shows the drive number, <d>, the track number, <t>, the sector number, <s>, and the head number <h> where the read was attempted. The second line shows the actual status of the 8272 disk controller chip from status registers 0 (<aa>), 1 (<bb>), and 2 (<cc>). The actual cylinder number (<dd>), head number (<ee>), ready status (<ff>), and number of bytes/sector (<gg>) are also displayed. Refer to Table B-1 for the interpretation of the status.

```
FDD WRITE FAILED ON DRIVE #<d>, TRK=<t>, SEC=<s>, HD=<h>
FDC STATUS =<aa> <bb> <cc> <dd> <ee> <ff> <gg>
             STO  ST1  ST2  C    H    R    N
```

The disk drive write test failed. The first line shows the drive number, <d>, the track number, <t>, the sector number, <s>, and the head number <h> where the read was attempted. The second line shows the actual status of the 8272 disk controller chip from status registers 0 (<aa>), 1 (<bb>), and 2 (<cc>). The actual cylinder number (<dd>), head number (<ee>), ready status (<ff>), and number of bytes/sector (<gg>) are also displayed. Refer to Table B-1 for the interpretation of the status.

```
FDD FORMAT FAILED ON DRIVE #<d>, TRK=<t>, SEC=<s>,
HD=<h>
FDC STATUS =<aa> <bb> <cc> <dd> <ee> <ff> <gg>
             STO  ST1  ST2  C    H    R    N
```

The disk drive format test failed. The first line shows the drive number, <d>, the track number, <t>, the sector number, <s>, and the head number <h> where the read was attempted. The second line shows the actual status of the 8272 disk controller chip from status registers 0 (<aa>), 1 (<bb>), and 2 (<cc>). The actual cylinder number (<dd>), head number (<ee>), ready status (<ff>), and number of bytes/sector (<gg>) are also displayed. Refer to Table B-1 for the interpretation of the status.

FDD DATA COMPARE FAILED ON DRIVE # <d>
LOC: <xx> EXP: <ee> ACT: <aa> TRK: <tt> SEC: <ss> HD: <hh>

The data read and compare test failed. The test data (<aa>) actually read on drive <d> did not compare to the data expected (<ee>) at the offset specified by (<xx>). The track number <tt>, sector number <ss>, and head number <hh> are also displayed.

BUBBLE DATA ERROR ON DRIVE # <d>
LOC: <xx> EXP: <ee> ACT: <aa> TRK: <tt> SEC: <ss>

The test data (<aa>) actually read at the offset specified by (<xx>) did not compare to the expected data (<ee>). The track number <tt> and the sector number <ss> are also displayed.

PERSONALITY CODE CHECKSUM ERROR
EXP: <eeee> ACT: <aaaa>

The checksum computed on the personality program read from the PROM Programmer Plug-in Module was not correct. The actual (<aaaa>) and expected (<eeee>) checksums are displayed.

<xxxx>/<ee>/<aa>

The random data (<aa>) actually read at the memory location specified by <xxxx> was not the expected data (<ee>). This display appears during the RAM test.

APPENDIX C
REFERENCE TABLES

Hexadecimal to Decimal Conversion

The following table is for hexadecimal to decimal and decimal to hexadecimal conversion. To find the decimal equivalent of a hexadecimal number, locate the hexadecimal number in the correct position and note the decimal equivalent. Add the decimal numbers.

To find the hexadecimal equivalent of a decimal number, locate the next lower decimal number in the table and note the hexadecimal number and its position. Subtract the decimal number shown in the table from the starting number. Find the difference in the table. Continue this process until there is no difference.

MOST SIGNIFICANT BYTE				LEAST SIGNIFICANT BYTE			
DIGIT 4		DIGIT 3		DIGIT 2		DIGIT 1	
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	3 548	E	224	E	14
F	61 440	F	3 840	F	240	F	15
7654		3210		7654		3210	
BYTE				BYTE			

Base Conversions

DEC	BIN	HEX	OCT	DEC	BIN	HEX	OCT
0	0000 0000	00	000	51	0011 0011	33	063
1	0000 0001	01	001	52	0011 0100	34	064
2	0000 0010	02	002	53	0011 0101	35	065
3	0000 0011	03	003	54	0011 0110	36	066
4	0000 0100	04	004	55	0011 0111	37	067
5	0000 0101	05	005	56	0011 1000	38	070
6	0000 0110	06	006	57	0011 1001	39	071
7	0000 0111	07	007	58	0011 1010	3A	072
8	0000 1000	08	010	59	0011 1011	3B	073
9	0000 1001	09	011	60	0011 1100	3C	074
10	0000 1010	0A	012	61	0011 1101	3D	075
11	0000 1011	0B	013	62	0011 1110	3E	076
12	0000 1100	0C	014	63	0011 1111	3F	077
13	0000 1101	0D	015	64	0100 0000	40	100
14	0000 1110	0E	016	65	0100 0001	41	101
15	0000 1111	0F	017	66	0100 0010	42	102
16	0001 0000	10	020	67	0100 0011	43	103
17	0001 0001	11	021	68	0100 0100	44	104
18	0001 0010	12	022	69	0100 0101	45	105
19	0001 0011	13	023	70	0100 0110	46	106
20	0001 0100	14	024	71	0100 0111	47	107
21	0001 0101	15	025	72	0100 1000	48	110
22	0001 0110	16	026	73	0100 1001	49	111
23	0001 0111	17	027	74	0100 1010	4A	112
24	0001 1000	18	030	75	0100 1011	4B	113
25	0001 1001	19	031	76	0100 1100	4C	114
26	0001 1010	1A	032	77	0100 1101	4D	115
27	0001 1011	1B	033	78	0100 1110	4E	116
28	0001 1100	1C	034	79	0100 1111	4F	117
29	0001 1101	1D	035	80	0101 0000	50	120
30	0001 1110	1E	036	81	0101 0001	51	121
31	0001 1111	1F	037	82	0101 0010	52	122
32	0010 0000	20	040	83	0101 0011	53	123
33	0010 0001	21	041	84	0101 0100	54	124
34	0010 0010	22	042	85	0101 0101	55	125
35	0010 0011	23	043	86	0101 0110	56	126
36	0010 0100	24	044	87	0101 0111	57	127
37	0010 0101	25	045	88	0101 1000	58	130
38	0010 0110	26	046	89	0101 1001	59	131
39	0010 0111	27	047	90	0101 1010	5A	132
40	0010 1000	28	050	91	0101 1011	5B	133
41	0010 1001	29	051	92	0101 1100	5C	134
42	0010 1010	2A	052	93	0101 1101	5D	135
43	0010 1011	2B	053	94	0101 1110	5E	136
44	0010 1100	2C	054	95	0101 1111	5F	137
45	0010 1101	2D	055	96	0110 0000	60	140
46	0010 1110	2E	056	97	0110 0001	61	141
47	0010 1111	2F	057	98	0110 0010	62	142
48	0011 0000	30	060	99	0110 0011	63	143
49	0011 0001	31	061	100	0110 0100	64	144
50	0011 0010	32	062	101	0110 0101	65	145

Base Conversions (continued)

DEC	BIN	HEX	OCT	DEC	BIN	HEX	OCT
102	0110 0110	66	146	153	1001 1001	99	231
103	0110 0111	67	147	154	1001 1010	9A	232
104	0110 1000	68	150	155	1001 1011	9B	233
105	0110 1001	69	151	156	1001 1100	9C	234
106	0110 1010	6A	152	157	1001 1101	9D	235
107	0110 1011	6B	153	158	1001 1110	9E	236
108	0110 1100	6C	154	159	1001 1111	9F	237
109	0110 1101	6D	155	160	1010 0000	A0	240
110	0110 1110	6E	156	161	1010 0001	A1	241
111	0110 1111	6F	157	162	1010 0010	A2	242
112	0111 0000	70	160	163	1010 0011	A3	243
113	0111 0001	71	161	164	1010 0100	A4	244
114	0111 0010	72	162	165	1010 0101	A5	245
115	0111 0011	73	163	166	1010 0110	A6	246
116	0111 0100	74	164	167	1010 0111	A7	247
117	0111 0101	75	165	168	1010 1000	A8	250
118	0111 0110	76	166	169	1010 1001	A9	251
119	0111 0111	77	167	170	1010 1010	AA	252
120	0111 1000	78	170	171	1010 1011	AB	253
121	0111 1001	79	171	172	1010 1100	AC	254
122	0111 1010	7A	172	173	1010 1101	AD	255
123	0111 1011	7B	173	174	1010 1110	AE	256
124	0111 1100	7C	174	175	1010 1111	AF	257
125	0111 1101	7D	175	176	1011 0000	B0	260
126	0111 1110	7E	176	177	1011 0001	B1	261
127	0111 1111	7F	177	178	1011 0010	B2	262
128	1000 0000	80	200	179	1011 0011	B3	263
129	1000 0001	81	201	180	1011 0100	B4	264
130	1000 0010	82	202	181	1011 0101	B5	265
131	1000 0011	83	203	182	1011 0110	B6	266
132	1000 0100	84	204	183	1011 0111	B7	267
133	1000 0101	85	205	184	1011 1000	B8	270
134	1000 0110	86	206	185	1011 1001	B9	271
135	1000 0111	87	207	186	1011 1010	BA	272
136	1000 1000	88	210	187	1011 1011	BB	273
137	1000 1001	89	211	188	1011 1100	BC	274
138	1000 1010	8A	212	189	1011 1101	BD	275
139	1000 1011	8B	213	190	1011 1110	BE	276
140	1000 1100	8C	214	191	1011 1111	BF	277
141	1000 1101	8D	215	192	1100 0000	C0	300
142	1000 1110	8E	216	193	1100 0001	C1	301
143	1000 1111	8F	217	194	1100 0010	C2	302
144	1001 0000	90	220	195	1100 0011	C3	303
145	1001 0001	91	221	196	1100 0100	C4	304
146	1001 0010	92	222	197	1100 0101	C5	305
147	1001 0011	93	223	198	1100 0110	C6	306
148	1001 0100	94	224	199	1100 0111	C7	307
149	1001 0101	95	225	200	1100 1000	C8	310
150	1001 0110	96	226	201	1100 1001	C9	311
151	1001 0111	97	227	202	1100 1010	CA	312
152	1001 1000	98	230	203	1100 1011	CB	313

Base Conversions (continued)

DEC	BIN	HEX	OCT	DEC	BIN	HEX	OCT
204	1100 1100	CC	314	230	1110 0110	E6	346
205	1100 1101	CD	315	231	1110 0111	E7	347
206	1100 1110	CE	316	232	1110 1000	E8	350
207	1100 1111	CF	317	233	1110 1001	E9	351
208	1101 0000	D0	320	234	1110 1010	EA	352
209	1101 0001	D1	321	235	1110 1011	EB	353
210	1101 0010	D2	322	236	1110 1100	EC	354
211	1101 0011	D3	323	237	1110 1101	ED	355
212	1101 0100	D4	324	238	1110 1110	EE	356
213	1101 0101	D5	325	239	1110 1111	EF	357
214	1101 0110	D6	326	240	1111 0000	F0	360
215	1101 0111	D7	327	241	1111 0001	F1	361
216	1101 1000	D8	330	242	1111 0010	F2	362
217	1101 1001	D9	331	243	1111 0011	F3	363
218	1101 1010	DA	332	244	1111 0100	F4	364
219	1101 1011	DB	333	245	1111 0101	F5	365
220	1101 1100	DC	334	246	1111 0110	F6	366
221	1101 1101	DD	335	247	1111 0111	F7	367
222	1101 1110	DE	336	248	1111 1000	F8	370
223	1101 1111	DF	337	249	1111 1001	F9	371
224	1110 0000	E0	340	250	1111 1010	FA	372
225	1110 0001	E1	341	251	1111 1011	FB	373
226	1110 0010	E2	342	252	1111 1100	FC	374
227	1110 0011	E3	343	253	1111 1101	FD	375
228	1110 0100	E4	344	254	1111 1110	FE	376
229	1110 0101	E5	345	255	1111 1111	FF	377

Powers of Two and Sixteen

Powers of Two

2^n	n
256	8
512	9
1 024	10
2 048	11
4 096	12
8 192	13
16 384	14
32 768	15
65 536	16
131 072	17
262 144	18
524 288	19
1 048 576	20
2 097 152	21
4 194 304	22
8 388 608	23
16 777 216	24

Conversion Between Powers of 2 and 16

$2^m = 16^n$
$2^0 = 16^0$
$2^4 = 16^1$
$2^8 = 16^2$
$2^{12} = 16^3$
$2^{16} = 16^4$
$2^{20} = 16^5$
$2^{24} = 16^6$
$2^{28} = 16^7$
$2^{32} = 16^8$
$2^{36} = 16^9$
$2^{40} = 16^{10}$
$2^{44} = 16^{11}$
$2^{48} = 16^{12}$
$2^{52} = 16^{13}$
$2^{56} = 16^{14}$
$2^{60} = 16^{15}$
$2^{64} = 16^{16}$

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

ASCII Code List

Decimal	Octal	Hexadecimal	Character	Decimal	Octal	Hexadecimal	Character
0	000	00	NUL	32	040	20	SP
1	001	01	SOH	33	041	21	!
2	002	02	STX	34	042	22	"
3	003	03	ETX	35	043	23	#
4	004	04	EOT	36	044	24	$
5	005	05	ENQ	37	045	25	%
6	006	06	ACK	38	046	26	&
7	007	07	BEL	39	047	27	'
8	010	08	BS	40	050	28	(
9	011	09	HT	41	050	29	)
10	012	0A	LF	42	052	2A	*
11	013	0B	VT	43	053	2B	+
12	014	0C	FF	44	054	2C	,
13	015	0D	CR	45	055	2D	-
14	016	0E	SO	46	056	2E	.
15	017	0F	SI	47	057	2F	/
16	020	10	DLE	48	060	30	0
17	021	11	DC1	49	061	31	1
18	022	12	DC2	50	062	32	2
19	023	13	DC3	51	063	33	3
20	024	14	DC4	52	064	34	4
21	025	15	NAK	53	065	35	5
22	026	16	SYN	54	066	36	6
23	027	17	ETB	55	067	37	7
24	030	18	CAN	56	070	38	8
25	031	19	EM	57	071	39	9
26	032	1A	SUB	58	072	3A	:
27	033	1B	ESC	59	073	3B	;
28	034	1C	FS	60	074	3C	<
29	035	1D	GS	61	075	3D	=
30	036	1E	RS	62	076	3E	>
31	037	1F	US	63	077	3F	?

ASCII Code List (continued)

Decimal	Octal	Hexadecimal	Character	Decimal	Octal	Hexadecimal	Character
64	100	40	@	96	140	60	\
65	101	41	A	97	141	61	a
66	102	42	B	98	142	62	b
67	103	43	C	99	143	63	c
68	104	44	D	100	144	64	d
69	105	45	E	101	145	65	e
70	106	46	F	102	146	66	f
71	107	47	G	103	147	67	g
72	100	48	H	104	150	68	h
73	101	49	I	105	151	69	i
74	102	4A	J	106	152	6A	j
75	103	4B	K	107	153	6B	k
76	104	4C	L	108	154	6C	l
77	105	4D	M	109	155	6D	m
78	106	4E	N	110	156	6E	n
79	107	4F	O	111	157	6F	o
80	100	50	P	112	160	70	p
81	101	51	Q	113	161	71	q
82	102	52	R	114	162	72	r
83	103	53	S	115	163	73	s
84	104	54	T	116	164	74	t
85	105	55	U	117	165	75	u
86	106	56	V	118	166	76	v
87	107	57	W	119	167	77	w
88	100	58	X	120	170	78	x
89	101	59	Y	121	171	79	y
90	102	5A	Z	122	172	7A	z
91	103	5B	[	123	173	7B	{
92	104	5C	\	124	174	7C	|
93	105	5D	]	125	175	7D	}
94	106	5E	∧	126	176	7E	~
95	107	5F	B	127	177	7F	DEL

ASCII Code Definition

Abbreviation	Meaning	Decimal Code
NUL	NULL Character	0
SOH	Start of Heading	1
STX	Start of Text	2
ETX	End of Text	3
EOT	End of Transmission	4
ENQ	Enquiry	5
ACK	Acknowledge	6
BEL	Bell	7
BS	Backspace	8
HT	Horizontal Tabulation	9
LF	Line Feed	10
VT	Vertical Tabulation	11
FF	Form Feed	12
CR	Carriage Return	13
SO	Shift Out	14
SI	Shift In	15
DLE	Data Link Escape	16
DC1	Device Control 1	17
DC2	Device Control 2	18
DC3	Device Control 3	19
DC4	Device Control 4	20
NAK	Negative Acknowledge	21
SYN	Synchronous Idle	22
ETB	End of Transmission Block	23
CAN	Cancel	24
EM	End of Medium	25
SUB	Substitute	26
ESC	Escape	27
FS	File Separator	28
GS	Group Separator	29
RS	Record Separator	30
US	Unit Separator	31
SP	Space	32
DEL	Delete	127

ASCII Code in Binary

LSB	MSB	0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	°	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	<	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L		l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	VS	/	?	O	_	o	DEL

Control Codes

Control characters can be used to generate the ASCII codes from 00H through 1FH from the keyboard.

Character	Code in Hexadecimal
CTRL-@	00
CTRL-A	01
CTRL-B	02
CTRL-C	03
CTRL-D	04
CTRL-E	05
CTRL-F	06
CTRL-G	07
CTRL-H	08
CTRL-I	09
CTRL-J	0A
CTRL-K	0B
CTRL-L	0C
CTRL-M	0D
CTRL-N	0E
CTRL-O	0F

Character	Code in Hexadecimal
CTRL-P	10
CTRL-Q	11
CTRL-R	12
CTRL-S	13
CTRL-T	14
CTRL-U	15
CTRL-V	16
CTRL-W	17
CTRL-X	18
CTRL-Y	19
CTRL-Z	1A
CTRL-<	1B
CTRL-\	1C
CTRL->	1D
CTRL-△	*
CTRL--	**

* - Use ↑ to generate the code 1EH.
** - Use ← to generate the code 1FH.

The iPDS CRT responds to CTRL-G (ASCII Bell), CTRL-H (ASCII Backspace), CTRL-J (ASCII Linefeed), CTRL-M (ASCII Carriage Return), CTRL-[(ASCII Escape).

Function Codes

Function codes are only defined for uppercase alpha characters A-Z and numeric characters 0-9. FUNCT-A through FUNCT-Z and FUNCT-0 through FUNCT-9 (except FUNCT-R, FUNCT-S, and FUNCT-T) are sent to the processor as the ASCII code corresponding to the alphanumeric character with the Most Significant Bit (MSB) set. FUNCT-R, FUNCT-S, and FUNCT-T are used by the CRT/Keyboard controller and are not sent to the processor.

Function codes for lower case alpha characters (FUNCT-a through FUNCT-z) are identical to the corresponding codes for the upper case alpha characters. The lower case is converted to upper case when combined with the FUNCT key. The codes for other FUNCT characters are undefined except for FUNCT-HOME, FUNCT-↑, and FUNCT-↓ which are used in dual processor systems.

NOTE

Sending non-ASCII codes to the CRT, i.e., codes greater than 7FH produces undefined results except in the case of FUNCT-R, FUNCT-S, FUNCT-T, FUNCT-HOME, FUNCT-↑, and FUNCT-↓.

Graphics Codes and Escape Sequences

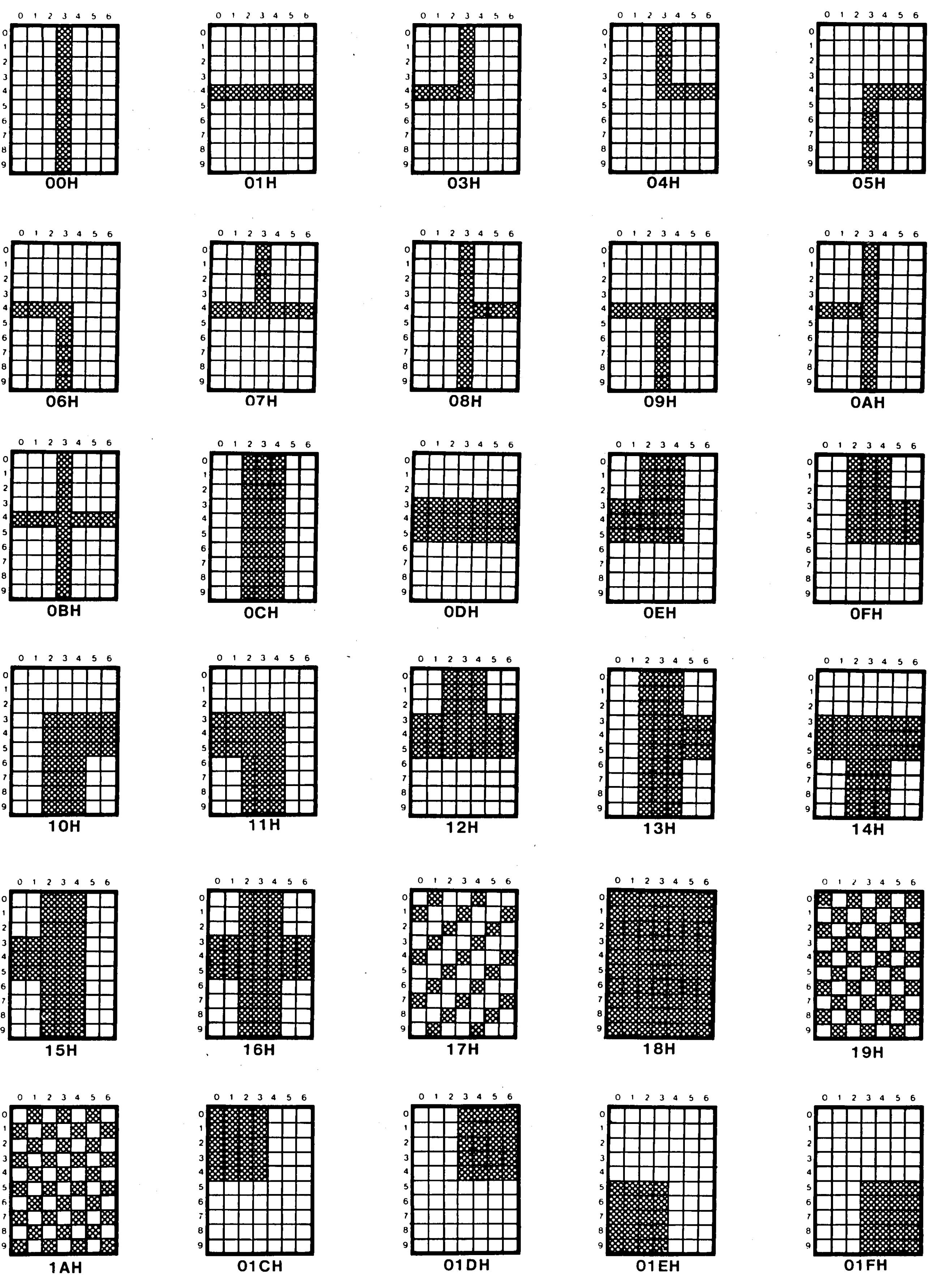
The following chart shows the graphics symbols available and the corresponding hexadecimal value that creates that symbol when output to the CRT screen when in graphics mode. ASCII characters are no more than 5 x 7 on a 7 x 10 background while graphics characters use the full 7 x 10 area. The use of graphics characters is described in Chapter 8.

In graphics mode, any control characters (00H-1FH except 02H and 1BH) that are output to the console will be displayed as the graphics symbol corresponding to the code. The code 1BH is the ESCAPE code and 02H is the alternate ESCAPE code. These two codes are used in graphics mode to generate escape sequences. Other characters (20H-7EH) will be displayed as the corresponding ASCII characters.

The escape sequences are:

ESC, A Move the cursor up one line.
ESC, B Move the cursor down one line.
ESC, C Move the cursor to the right one character.
ESC, D Move the cursor to the left one character.
ESC, E Home the cursor and clear the screen.
ESC, H Home the cursor.
ESC, J Erase from the current cursor location to the end of the screen.
ESC, K Erase the line that contains the cursor from the cursor to the end of the line.
ESC, G Enter graphics mode.
ESC, N Exit graphics mode.

The escape sequences and the methods of programming in graphics mode are described in detail in Chapter 8.


00H
01H
03H
04H
05H
06H
07H
08H
09H
0AH
0BH
0CH
0DH
0EH
0FH
10H
11H
12H
13H
14H
15H
16H
17H
18H
19H
1AH
01CH
01DH
01EH
01FH
0012A

ISIS-PDS and ISIS-II Features

Table D-1 compares ISIS-II features with ISIS-PDS features.

Table D-1. ISIS-II and ISIS-PDS Features

Feature	ISIS-II	ISIS-PDS
Disk Drives	8" Single Side Single Density; 8" Single Side Double Density; 14" Hard Disk	5 ¼" Double Side Double Density and Bubble Memory
Sector Size	128 bytes/sector	256 bytes/sector
Automatic JOB Run at Boot	No	Yes
Select Console at Boot	Yes	No
Monitor I/O Driver	Yes (in ROM)	Yes (as part of ISIS)
Debug Commands	Yes	Yes; includes single line disassembler; step execution; and I/O commands
Reload ISIS	A1 Interrupt switch	FUNCT-R
Return to Monitor	Interrupt 0	FUNCT-R while in DEBUG
HELP Command	No	Yes
FORMAT Command	Yes	No (uses IDISK only)
SERIAL Command	No	Yes
ASSIGN Command	No	Yes
ATTACH Command	No	Yes
DETACH Command	No	Yes

Table D-1. ISIS-II and ISIS-PDS Features (continued)

Feature	ISIS-II	ISIS-PDS
Dual Processors	No	Yes (shared devices)
RAM Used by ISIS	12K bytes	14K bytes (including the monitor I/O driver)
Supports RI, PO primitive I/O drivers	Yes	No
Supports SI, SO primitive I/O drivers	No	Yes
ISIS-II User System Calls	Yes	Yes
User-defined File Attributes	No	Yes
ISIS-II Directory Structure	Yes	Yes (includes user-defined attributes
Command Line Interpreter	Yes	Yes (extended)
Bit Map	Yes	Yes (uses cluster pre-allocation scheme; each bit represents four sectors)

Most programs that are written under ISIS-II will also run under ISIS-PDS.

Use the FTRANS File Transport Utility product to transfer files between other Intel microcomputer development systems and the iPDS Personal Development System.

APPENDIX E
TIPS FOR OPERATING EFFICIENTLY

Single Drive System

A strategy that a user could follow to create large programs with just the single disk drive is described in this section.

Three types of disk are needed:

a) Source disk with system commands, the CREDIT text editor, and language translator.

b) LINK/LOCATE disk (only needed for programs larger than 56K bytes) with system commands (except CREDIT) and the LINK, LOCATE, and LIB commands.

c) Emulator/PROM Programmer disk with the system commands (except CREDIT) and LINK, LOCATE, LIB, EMV software for the emulator being used, and iPPS software for PROM Programming.

The Source disk is used to create and contain the source program, the list files, and the object files created by the language translator.

If the program is less than 56K, the relocatable object files created by the translator should be copied directly to the Emulator/PROM Programmer disk to be linked, located, debugged, and copied to PROMs.

If the program is greater than 56K, the relocatable object modules created by the translator should be copied to the the LINK/LOCATE disk to be linked and located. After being located, the executable object files should be copied to the Emulator/PROM Programmer disk for debugging and PROM programming.

In the rest of this section, an example is considered to illustrate the use of the single drive system.

The following assumptions are made for this example.

- The source disk contains 82K bytes of system files and 99K bytes for an assembler leaving the user with 459K bytes available on the source disk.

- The lengths of the files in bytes are:

  Source file = 1 * x

  List file = 3 * x

  TMA/TMB files = 3.5 * x

  Object files = 0.5 * x (with symbols)

  0.2 * x (without symbols)

  where x is the length of the source file.

- All user source modules are 100 lines.

- Each source line is 25 bytes.
- Listing files are saved on the disk.
- The Emulator/PROM Programmer disk contains 50K bytes for system files (without CREDIT), 96K bytes for EMV software (EMV-51), 36K for iPPS software, and 30K for LINK, LOCATE, and LIB commands. The total of 212K bytes leaves 428K bytes of disk space for the user on the Emulator/PROM Programmer disk.
- The LINK/LOCATE disk contains 50K bytes of system files (without CREDIT) and 30K for the LINK, LOCATE, and LIB commands. The total of 80K bytes leaves 560K bytes of disk space for the user on the LINK/LOCATE disk.

Table E-1 shows the number of modules and size of the object program based on these assumptions. The calculations used are given following the table.

Table E-1 Module Number and Object Program Size

	Free Space	Number of Modules	Executable Object File Size
Source Disk	459K bytes	36 modules	17.5K bytes
LINK/LOCATE Disk	560K bytes	229 modules	111.0K bytes
EMV/iPPS Disk	428K bytes	116 modules	56.0K bytes

About 40 source files of 100 lines each are possible on one source disk, assuming that each source module is 100 lines and each line is 25 bytes. The following calculations can then be made.

100 source lines * 25 bytes/line = 2500 bytes/source module

3.5 * 2500 = 9K for TMA/TMB files

Each source file then requires:

2500 bytes - source file
1250 bytes - object file with symbols (500 bytes actual object file)
7500 bytes - listing files
11250 bytes - total space for each source file through assembly

Subtract the 9K bytes of temporary files from the 459K bytes of user disk space to get 450K bytes of disk space for source files on the source disk. If each source file requires about 11K bytes then:

450K/11K = about 40 source files on each source disk

Then, the following calculations can be made:

40 files * 1250 bytes/object file & symbols = 48K bytes of object file per disk

36 files * 1250 bytes/object file & symbols = 44K bytes of object file per disk[1]

36 files * 500 bytes of actual object code = 17.5K bytes of actual object code per disk

1 Since the COPY command copies 44K bytes at one time, if only 36 files are used, the disk will not have to be swapped to transfer all the object files to the EMV/iPPS disk.

About 116 object modules can be linked and located on the EMV/iPPS disk at one time. This figure is derived from the following calculations.

Assuming that the link file and the execution file are the same as the total for all the object files, then 3x is the total for all three where x is the size of all the object files. Since there are 428K bytes of user space on the EMV/iPPS disk,

3x = 428K bytes of free disk space on the EMV/iPPS disk
x = 142K bytes for the total object files

If there are 1250 bytes/object module (from the previous calculations), then,

142K bytes of object module space/1250 bytes per object module
= 116 object modules

500 bytes of actual object code/module * 116 modules
= 56K actual code

About 152 object modules can be linked and located on the LINK/LOCATE disk. This figure is derived from the following calculations.

Assuming that the link file and the execution file are the same as the total for all the object files, then 3x is the total for all three where x is the size of all the object files. Since there are 560K bytes of user space on the LINK/LOCATE disk,

3x = 560K bytes of free disk space on the LINK/LOCATE disk
x = 186K bytes for the total object files

If there are 1250 bytes/object module (from the previous calculations), then,

186K bytes of object module space/1250 bytes per object module
= 152 object modules

500 bytes of actual object code/module * 152 modules
= 74K actual code

About 229 object modules can be linked on a single LINK/LOCATE disk. However, the object modules would have to be deleted after linking them and the new linked file would be located using the disk space from the object modules.

2x = 560K bytes of free disk space on the LINK/LOCATE disk
x = 280K bytes for the total object files

If there are 1250 bytes/object module (from the previous calculations), then,

280K bytes of object module space/1250 bytes per object module
= 229 object modules

500 bytes of actual object code/module * 229 modules
= 111K actual code

On a single drive system, the user must manually swap disks in and out of the single drive. This swapping is one of the added time factors in using a single drive system. The COPY command copies 44K bytes at each swap. Thus, the number of swaps required to translate and debug a program depends primarily on the size of the program. Table E-2 summarizes the number of swaps required and the number of each type of disk required to generate the software relative to the size of the program. The figures used are based on the previous calculations.

Table E-2 Summary of Number of Disks and Swaps Required

Code Size	Source Disks	Link/Locate Disks	EMV/iPPS Disks	Number of Swaps
1K - 17.5K	1	0	1	1
18K - 35.5K	2	0	1	3
40K - 53.5K	3	0	1	5
54K - 71.5K	4	1	1	10
72K - 89.5K	5	1	1	12-13*
90K -107.5K	6	1	1	16*

* Worst case

Bubble Memory System

Each bubble memory device contains 128K bytes of storage space. It is recommended that the bubble memory be used as the system disk device only if a disk drive is not available. The system files use about TBDK of space and would leave only TBDK for the user.

The access times for bubble memory are faster than for disk drives. For example, copying a 4K file using bubble memory takes 7 seconds while it takes 14 seconds to make the same transfer on disk drives. Therefore, short programs that are run often and data files that are frequently accessed should be stored in bubble memory.

The following are typical applications for the bubble memory device:

- By using the bubble memory for all .TMA or .TMB files when translating the source code to object code, the translation (assembler or compiler) will run faster.

- The bubble memory can also be used to store all work files created by the emulator software (EMV program) and the PROM programming software (iPPS) program. By storing the workfiles in bubble memory, these programs will run faster.

- Short programs that are run often can be stored in bubble memory to reduce the load time of the program. The tradeoff in storing programs on bubble memory is between space (128K available) and speed of access. If a program is not used often, it can be stored on disk.

- The bubble memory can be used to store diagnostic programs that are run for quality control in a manufacturing environment. Here, the programs are frequently loaded and the environment may not be suitable for disks. Both factors make the bubble memory a good alternative.

- The bubble memory device can also be used as the system disk and to store diagnostics for the user's product in field service applications where disks are not easily available.

Dual Processor System

A system with an optional processor installed should have at least two disk drives. One drive should be assigned as the system disk device (:F0:) for one of the processors, while the other drive is assigned as the system disk device (:F0:) for the other processor. Otherwise, disk contention caused by both processors trying to access the same physical device significantly degrades processing speeds on both processors.

The system disk device is the default used to access command files and data files when no explicit drive specification is given. Thus, each drive used as a system disk device should contain a system disk and all the necessary command files that will be used.

The dual processor system can be used in a debugging environment as follows. While debugging a program on Processor A using the EMV software, a bug can be patched in memory. However, the corrected code is not updated in the source file.

Without exiting from the emulator, the user can switch to Processor B, correct the source code using the CREDIT text editor and retranslate the program using a SUBMIT file that assembles/compiles, links, and locates the program.

After the SUBMIT file is started, the user can switch back to Processor A and continue debugging the patched code while monitoring the progress of the SUBMIT file.

While running a program on either processor, if a filename is needed, the DIR command can be run on the other processor to find the correct name. Then, the user can return to the first processor and continue running the program without having entered an incorrect name.

GLOSSARY

absolute object module	an object module that has already been linked and located and contains all the necessary instructions and information so it can be executed on the target microprocessor.
access	the ability to use all or part of the system, for example, to access a file.
address	a location in memory or on a system device.
allocate	to designate a resource, such as memory, for a specific use.
argument	a variable supplied with a command. The result of the command depends on the values of its arguments.
array	a sequence of items, such as a memory component is an array of bits.
ASCII	American Standard Code for Information Interchange. A convention adopted for representing alphanumeric values as byte values.
assembler	a program that translates assembly language source code into object code.
assembly language	a programming language whose instructions are closely related to the instruction set of the target processor.
asynchronous data transmission	a method of transmitting data where the rate of transmission varies. Each byte of data is preceded by one or more start bits and followed by one or more stop bits. Compare with synchronous transmission.
attribute	a characteristic of a disk file, specified in the directory containing the file. The attributes of an ISIS-PDS disk file are invisible, write-protect, format, system, and three user defined attributes.
backup copy	an additional copy of data saved to prevent loss of data in case the original copy is destroyed. It is a good practice to save backup copies of all disk files.
base address	the number to which an offset number is added to produce the actual address.
base processor	the processor originally supplied with the iPDS system. See Processor A and optional processor.
batch processing	a method of running a group of jobs in sequence without requiring operator intervention. The ISIS operating system offers several commands for batch processing. See the SUBMIT and JOB commands.

baud rate	rate of transmission of serial data. On the iPDS system, the baud rate is the number of bits transmitted per second.
binary	referring to the base two number system.
bit	a binary digit (either 0 or 1) or a unit of information storage with only two possible values.
block	a collection of information handled as a single unit. For example, on the iPDS system, a disk block is 256 bytes of data corresponding to a sector on the disk.
bootstrap	to initialize the system to accept programs and data. On the iPDS system, to load the ISIS-PDS operating system. Boot is an abbreviated form of bootstrap.
breakpoint	a specific location in a program where execution of the program should be halted.
bubble memory	a storage device based on magnetic bubbles which can be present or absent, thereby representing binary data. Bubble Memory Multimodules are used as alternate disk devices on the iPDS system.
buffer	an area of memory reserved for expediting input and output operations.
byte	a sequence of eight consecutive binary digits treated as a single value.
byte bucket	a pseudo-device for input or output of data.
CLI	Command Line Interpreter. An ISIS-PDS program that interprets command lines and loads and executes the program specified in the command line.
close file	an operation that removes a file from being accessed. A file can be accessed only when it is opened.
command	an instruction to the operating system or to a program running under the control of the operating system.
command file	a file containing a program that is run when a command line is issued.
command line	the entry made by a user to invoke a command.
compiler	a program that translates high-level language source code into object code.
configuration	a description of a computer system.
connection	a description of how to access a file used by systems programmers.
console	the primary device used for interactive input and output in a system. On the iPDS system, the keyboard is the standard console input device and the CRT screen is the standard console output device.

control character a single character command given at the keyboard by entering the CTRL key and another key simultaneously. Control characters perform such functions as displaying the contents of the line editing buffer, deleting the contents of the line editing buffer, and stopping the console output.

controller a hardware element that enables a processor to manipulate one or more I/O devices of a specific type.

CPU Central Processor Unit. The component of a computer system that controls the rest of the system.

CRT Cathode Ray Tube. A vacuum tube with a screen used for displaying data similar to a TV screen.

cursor a marker used on CRT screens to indicate where data will appear next. On the iPDS system, a reverse video block is used as the cursor.

cylinder on the iPDS system, one cylinder consists of two tracks, one on each surface of the disk. With the bubble memory multimodule, one cylinder is the same total number of bytes as one track.

data block under ISIS-PDS, a sector (256 bytes) on the disk containing actual data of a file. See pointer block.

debug to locate and correct errors in a program. The iPDS system provides the DEBUG command to aid in this process.

default value the value of an input parameter that is assumed by a program if no value is explicitly given.

delimiter under ISIS-PDS programs, a character that separates parts of a command line. A space and a comma are common delimiters.

destination the device used to receive the output from a data transfer.

development system a computer system designed especially to support the development of computer based products, both hardware and software.

device a piece of peripheral equipment, such as a printer, that is attached to a system and can be accessed by the system.

device driver a program used to control an I/O device.

device name a string of characters recognized by the operating system, that identifies a physical input or output device. It consists of four characters: a colon, followed by two alphanumeric characters, followed by another colon. Examples of device names are :F1:, :CI:, :LP:, and :BB:.

directory	a table present on each disk that contains a list of all the files on that disk giving the name, length, location and attribute for each file.
disk	a term designating the 5¼" flexible diskette used as a recording medium in disk drives or the recording medium of the bubble memory.
dual processing	a capability of the iPDS system which allows programs to be run simultaneously on two different processors.
dynamic	the capacity of data, memory, or other entities of being changed while a program is running.
emulator	a module composed of hardware, firmware, and software that aids the designer of a microcomputer system in developing and debugging the hardware and software.
entry point	the first instruction to be executed in a program or subroutine.
environment	the files, memory, and hardware resources available to the system.
EOF	End of File.
EPROM	Erasable Programmable Read Only Memory.
E^2PROM	Electrically Erasable Programmable Read Only Memory.
error	a mistake caused by a program that is currently running.
error handling	the ability of an operating system or user program to deal with error conditions.
exception	error.
extension	the optional part of a filename, consisting of one to three alphanumeric characters preceded by a period.
fatal error	an error in the system that makes it impossible for a program to continue running and causes the system to reinitialize.
file	a collection of information that may be read or written by an operating system command. It may correspond to a physical device such as a printer which has only one file associated with it or it may be one of many files on a multiple file device such as a disk drive.
file name	a character string recognized by the operating system, that identifies a file. It consists of a name of one to six alphanumeric characters, followed by an optional extension. Examples of filenames are PROGA, PROGA.SRC, and ISIS.BIN.

firmware	computer programs stored in a physical device such as ROM that can be used in a machine.
format	the general form of a command that defines the sequences of symbols that produce an acceptable command. See syntax.
graphics mode	a mode in which characters output to the console can be graphics characters.
hardware	the electronic circuits that comprise a computer.
hexadecimal	number system with a base of 16 and with digits of 0-9 and A, B, C, D, E, and F.
initialization	the process of establishing the beginning environment of a computer system including powering on hardware and loading software.
instruction	command in a program that tells the computer what to do.
interface	common boundary between two parts of a system.
interleaving, on disk	the technique of storing consecutive blocks of a file at non-adjacent locations on the disk. This technique enhances the access time.
interleaving, on PROMs	the technique of storing consecutive bytes on different PROM devices to allow two 8-bit PROMs to be connected in parallel to produce a 16-bit wide memory device.
interpreter	a program that directly executes a high-level language source code or intermediate code, so that the source code need not be translated into object code. An example of an interpreter is BASIC-80.
interrupt	a break in the execution of a program such that the program can be resumed after the interrupt is processed.
interrupt level	a priority assigned to an interrupt.
invocation line	the command line used to run a program.
I/O	Input/Output.
I/O system	the collection of routines that handles input and output to peripheral devices.
iteration	a repetition.
job	a program that can be run under the control of the ISIS-PDS operating system.
jobfile	a file containing ISIS-PDS command lines.

K (Kilo)	1024.
library	a file created by the LIB utility that contains one or more object modules that can be accessed by a single ISIS filename. Libraries are used in the LINK utility.
link	the process of combining program modules to run as a single program.
load	to transfer data, usually a program, from a file into memory.
locate	the process of assigning physical addresses in a program.
logical name	a symbolic name assigned to a device by which a user accesses that device. See physical name.
line terminator	characters used to end a line. Under ISIS-PDS, the carriage return followed by a linefeed terminates a line.
machine state	the conditions of all the elements of the processor: the registers, the stack pointer, and the program counter.
macro	a group of commands identified by a single name and executed by entering that single name.
mass storage device	a device used to store large files of data externally.
memory	a component of a microcomputer where data and programs are stored external to the processor.
microcontroller	an LSI component containing all of the necessary parts of a computer: a processing unit, a clock, I/O ports, and memory.
microprocessor	an LSI component containing the CPU of a computer.
monitor	Under ISIS-PDS, the DEBUG program is referred to as a monitor program.
multimodules	small circuit boards provided as options for the iPDS system to add specific capabilities to the system. For example, the bubble memory multimodule provides additional mass storage.
multimodule row	on the iPDS system, four multimodules may be added. They are grouped into two rows of two each.
nesting	the relationship between subroutines such that one subroutine is embedded within another. CREDIT macros can be nested.
non-system disk	Under ISIS-PDS, a disk that does not contain all the system files necessary to initialize the system. It contains the files ISIS.T0, ISIS.LAB, ISIS.DIR, and ISIS.FRE. To make a non-system disk into a system disk, copy the files ISIS.PDS and ISIS.CLI from a system disk to the non-system disk.

notational conventions	conventions adopted throughout a manual to describe a system.
object code	code produced by a compiler or assembler that can be either relocatable or absolute. See relocatable object code and absolute object code.
object file	file containing object code.
octal	number system using a base of 8.
offset	value added to a base to produce the actual value desired.
open file	an operation that allows a file to be accessed. Different programs have limits on the number of files that can be opened at a time.
operating system	a collection of programs that provide the functional (as opposed to the physical) environment in which other programs work.
optional processor	On the iPDS system, an additional processor can be added to the system to increase the processing throughput. See Processor B and base processor.
options	enhancements that can be added to the system. a part of a command line that is not required, but may be entered to modify the operation of the command.
origin point	the address of the first instruction to be executed in a program or subroutine. See entry point.
overlay	a portion of a program that is not loaded from disk immediately when the program is run but is loaded when needed by the program.
parameters	a quantity that can be given a different value to change the operation of a command.
path name	the device name and file name used to identify and access a file.
peripherals	external devices used for input, output, and storage of data.
physical name	the identifier used internally by the system to access a peripheral device.
plug-in modules	small, hand-held units available as options for the iPDS system to add capabilities such as emulation and PROM programming to the system.
pointer	a value used to direct access to a location containing data.
pointer block	a sector in a disk file that contains pointers to data blocks. Also, header block. See data block.

port	an arrangement of circuitry on a microprocessor that allows a byte or word of data to be input from or output to an external device.
procedure	a named block of PL/M code that is not executed at the point where it is written but may be activated from other points in the program by referring to its name.
PROM	Programmable Read Only Memory.
prompt	a sequence of characters displayed on the CRT screen by an interactive program to indicate that the program is ready to accept command input.
Processor A	base processor.
Processor B	optional processor.
RAM	Random Access Memory. RAM can be used to read and write data.
real-time breakpoints	a feature of some debugging tools where breakpoints that are set do not slow down the processing speed.
relocatable object code	object code produced by a compiler or assembler that can be linked with other modules and then located to produce absolute object code.
reset	to restore system to its initial state. On the iPDS system, the RESET key on the keyboard performs a hardware reset to the system.
resource	devices available to the system for processing data.
ROM	Read Only Memory.
sector	256 contiguous bytes on a disk.
semaphore	a bit that used as to signal that a resource is currently being used, so that both processors do not attempt to access the same resource at the same time.
serial device	device that transfers data a bit at a time. On the iPDS system, provisions are available to connect a serial device.
software	programs and data used to control computer hardware.
source	device used to provide input for a data transfer.
source code	high level language or assembler language version of a program.
switch	same as options on a command line. A part of the command line that is not required but may be entered to modify the operation of the command.

symbolic debugging	a debugging environment where symbols can be used to access memory locations (or any other values) rather than physical addresses.
synchronous data transmission	a method of transmitting data where the rate of transmission is a constant. See asynchronous data transmission.
syntax	same as format.
system	a group of components, both hardware and software, designed to perform some task.
system call	a routine in a system that may be called to perform an operation from a program.
system disk	a disk containing all the files necessary to initialize the system. The files are ISIS.T0, ISIS.LAB, ISIS.DIR, ISIS.FRE, ISIS.CLI, and ISIS.PDS.
system files	files containing the ISIS-PDS operating system.
tab	a key on the keyboard that moves the cursor to the next tab stop.
tag	in the CREDIT text editor, a marker that designates a location in the text file.
text editor	a program used to enter and modify and store text in disk files.
throughput	the number of programs that can be processed during a given amount of time. The throughput is increased on a dual processor system because two programs can be run during the same time period.
top down design	a method of designing programs in an organized way in which the program is decomposed into independent modules which can be programmed separately using, for example, the procedure facility in PL/M.
track	a group of sectors on one surface of the disk.
translator	a program that translates program source code into object code.
vectored interrupt	a scheme where an interrupt causes the system to jump to a constant location called the vector.
volatile	characteristic of being erased when power is removed. When applied to memory, RAM is volatile memory. Magnetic Bubble Memory, PROM, EPROM, and ROM are non-volatile.
volume	disk.
volume id	label used to identify disk.

wildcard designation	method used to designate a class of names such as file names with certain character positions variable.
workfile	file used by some programs to temporarily store data during processing. Workfiles are created and maintained by programs when needed.
write protect	characteristic of file such that the file cannot be erased.

INDEX

When more than one page is listed for an item, the italicized page numbers (for example, *3-11*) are primary references; other pages are secondary references or examples.

#, 4-7, 4-12, 4-13, *5-45*
%, 4-68 thru 4-71, *5-38*
&
  IPPS, 10-11
*
  ISIS-PDS wildcard substitution, 4-34 thru 4-36, 4-51, 4-53, 4-60, 4-70, 4-72, *5-6*
.
  DEBUG prompt, 4-54 thru 4-59, *7-3*
  ISIS-PDS command, 5-46
...
  notational convention, 5-1
/, 5-47
:BB:, 5-4
:CI:, 5-4
:CO:, 5-4
:F<n>:, 5-4
:HP:, 5-3
:HR:, 5-3
:I1:, 5-3
:L1:, 5-3
:LP:, 5-3
:O1:, 5-3
:P1:, 5-3
:P2:, 5-3
:R1:, 5-3
:R2:, 5-3
:SI:, 5-3
:SO:, 5-3
:TP:, 5-3
:TR:, 5-3
:VI:, 5-3
:VO:, 5-3
< >, 3-19, *5-1*, 10-12
<n>, 5-6
?
  ISIS-PDS command, 4-1, *5-42*
  wildcard character
    ISIS-PDS, 4-35, *5-6*
@, 4-23, *5-43*
[ ]
  notational convention, *5-1 thru 5-2*,
{ }
mnotational convention, 3-19, *5-1*,
;, 6-5

A command
  DEBUG, 7-17
Absolute Object File Format, *8-86*, 10-10
access system call parameter, 8-10
accessing
  devices, *5-2 thru 5-8*, 8-3, *8-80*
  files, *5-3 thru 5-8*, 8-3, *8-88*
actual$ptr system call parameter, 8-10
address, 7-6, 7-19, 7-20, 7-22, 7-23, 7-26, 7-27, 7-28, 7-31, *8-70*, 8-71, 8-7 thru 8-74, *8-79 thru 8-80*
AFTN, 8-9
An>, *3-21*, 9-2
An+, *3-21*
appending files, 4-21, *5-16*
arrow keys, 3-8, 6-2, 6-4
ASCII
  characters, C-6 thru C-9
  codes, C-6 thru C-9
  files, 8892
ASM-80/85, 1-7, 4-53, 8-6
Assembly language service routine usage, 8-6
  equates for system calls, 8-7 thru 8-8
ASSIGN command
  ISIS-PDS, 4-9, *5-10*
Assign command
  DEBUG, 7-19
atrb system call parameter, 8-10
ATTACH
  ISIS-PDS command, 5-13
  System call, 8-14
ATTRIB
  ISIS-PDS command, 4-18 thru 4-19, *5-14*
  System call, 8-16
attributes, *5-14*, 8-94
Auto configuration, 3-17, 4-60, 4-61, 3-17, 4-60, 4-61, *5-31 thru 5-32*
Auto repeat feature on keyboard, *3-5*

backing up
  files, *3-12*, 4-72 thru 4-74
  disk, 3-15, *3-25*, 3-27, 4-74 thru 4-75
backslash
  CREDIT, *6-3*
base address calculation, 8-72
basic system
  installation, A-1 thru A-5
  introduction, 1-1 thru 1-12
batch program execution, *4-60 thru 4-74*, 5-31, 5-37
baud rate modification, 4-7, 4-12 thru 4-13, *5-36*, 8-76 thru 8-77
binary number base
  conversion tables, C-2
  IPPS, *10-9*
bipolar PROM, 10-1

blank state of PROM, *10-3*
block$ptr system call parameter, 8-10
blocks
  disk, 8-88 thru 8-90
Bn>, 9-2
Bn+, 3-22
boot, *3-16*, B-15 thru B-17
breakpoints
  DEBUG command, 7-1, *7-28*
  software development, 2-5, 2-6
bubble memory multimodule
  drive numbers, A-19
  installation, A-19 thru A-22
  use, 3-14, 3-40 thru 3-44, *5-3, E-5*
buf$ptr system call parameter, 8-12
buffer
  file, 8-46, 8-62, *8-71 thru 8-73*
  IPPS, *10-7 thru 10-8*
byte$ptr system call parameter, 8-10
byte bucket, 5-4

C command
  DEBUG, 7-21
calls
  system, 8-3 thru 8-63
care of disks, 3-11
categories
  DEBUG commands, 7-7
  ISIS-PDS commands, 4-1, 4-2, 4-7, 4-14, 4-40, 4-52, 4-59, *5-9*
  system calls, 8-3 thru 8-4
CAUTION
  flag, 1-11
Change commands
  I/O ports from DEBUG, *7-28, 7-33*
  memory from DEBUG, *7-26, 7-31, 7-34*
  registers from DEBUG, 7-34
char system call parameter, 8-10
CI (console input)
  system call, 8-18
ci$path$ptr, 8-10
CLEAR command, B-18
CLI, *3-15*, 4-1, 5-42, 8-1, 8-81
CLOSE
  system call, 8-19
CO (Console Output)
  system call, 8-21
code conversion commands
  HEXOBJ, 4-52
  OBJHEX, 4-52
command files
  ISIS-PDS, 4-60 thru 4-74, *5-32, 5-41*
command formats
  DEBUG, 7-5 thru 7-6
  ISIS-PDS, 5-1 thru 5-9
command line interpreter, *1-8*, 3-15, 4-1, 5-42, 8-1, 8-81
command line mode
  CREDIT, *6-4 thru 6-5*
command notational conventions, *5-1 thru 5-2*,
command
  defaults
    ISIS-PDS, 3-21
  descriptions
    CREDIT, *6-1 thru 6-12*
    DEBUG, *7-1 thru 7-34*
    IPPS, *10-5*
    ISIS-PDS, 5-10 thru 5-50
  editing, *3-21*, 5-52
  entry
    DEBUG, 7-4 thru 7-5
    IPPS, 10-9
    ISIS-PDS, 3-21 thru 3-23
  format
    DEBUG, 7-5
    ISIS-PDS, 3-19, *5-2 thru 5-9*
  line, 3-19
  syntax
    DEBUG, 7-5
    ISIS-PDS, 3-19, *5-2 thru 5-9*
comments
  IPPS, 10-6
  ISIS-PDS, 3-19
concatenation
  ISIS-PDS COPY command, 5-16
Confidence Tests, 3-42 thru 3-44, *B-15 thru B-29*
config$byte system call parameter, 8-10
configurations, *3-17*, 4-59, 4-65, 5-32, 8-5, 8-64
conn system call parameter, 8-11
conn$ptr system call parameter, 8-11
connections, 8-9 thru 8-10
CONPDS commands, B-16 thru B-27
CONSOL
  system call, 8-22
console
  assignment, 4-9, *5-10*, 8-22
  determination of console device, *5-11*, 8-32
  device, 5-4
control$sw system call parameter, 8-11
control characters, *3-6 thru 3-8*, 3-21, 3-22, 6-2 thru 6-5
control panel, 3-2 thru 3-3
conversion tables, C-1 thru C-5
converting
  number bases, C-1 thru C-5
  files
    absolute object to hexadecimal, 4-52
    hexadecimal to absolute object, 4-52
CO system call, 8-21
co$path$ptr, 8-11
COPY command
  ISIS-PDS, 4-22, *5-16 thru 5-21*
COUNT
  in system call, 8-47
count system call parameter, 8-11
creating a file
  COPY, 5-16
  CREDIT, 6-1
CREDIT command, 6-3

CRT
CREDIT, *6-4*
input, 8-18
ISIS-PDS, 3-10
output, 8-21
CS file, 5-37 thru 5-42
CSD file, 5-37 thru 5-42
CSTS (Console input Status)
system call, 8-26
CTRL key, *3-6 thru 3-7*, 3-22, 3-23, 6-2 thru 6-3
CTRL-A
ISIS-PDS, *3-7*, 3-22, 5-49
CTRL-B, 5-48
CTRL-D
ISIS-PDS, *3-6*, 3-22, 5-49
CTRL-E, 5-38
ISIS-PDS, *3-6*,
CTRL-L, 5-48
CTRL-P
ISIS-PDS, 3-6
CTRL-Q
ISIS-PDS, *3-6*, 3-10, 3-22
CTRL-R
ISIS-PDS, 3-6
CTRL-S
ISIS-PDS, *3-7*, 3-10, 3-23
CTRL-X
ISIS-PDS, 3-7, 5-48
CTRL-Z
ISIS-PDS, 3-7
cursor, *3-10*, 6-4

D command
DEBUG, 7-22
DEBUG command, 4-52, 4-54, *7-2 thru 7-4*
debugging programs, *2-1 thru 2-6*, 4-54 thru 4-55, 7-11 thru 7-21
decimal
IPPS, 10-9
defaults
IPPS, 10-6
ISIS-PDS, 3-20
DELETE
ISIS-PDS command, 4-14, 4-36, *5-21*
system call, 8-25
deleting
ISIS-PDS
command lines, 3-7, 3-19, *5-49*
files, 4-14, 4-35, *5-22*
from a program, 8-25
DESCRIBE command, B-18
design
hardware, 2-2
software, 2-1
design cycle, 2-1
DETACH
ISIS-PDS command, 5-23
system call, 8-26 thru 8-27

determining the console assignment, *5-11*, 8-32 thru 8-33
determining the memory space, 8-40
determining the processor, 9-14
device
management, 4-7
names, 5-3 thru 5-4
diagnostics, B-16 thru B-18
DIR
ISIS-PDS command, 4-14, *5-24*
directory
content, 8-95
listing, 4-14 thru 4-15, *5-24*
disk
addressing, 8-88 thru 8-97
backup, *3-12*, 3-15, 3-25, 3-27
care, 3-11
directory, 4-14, *5-25*, 8-95
drives
configuration, A-29
external, 1-4, *3-10, A-29*
logical, *5-4, 5-10*
installation of, *A-3*
integrated, 1-2, *3-10*
internal, 1-2, *3-10*
operation of, *3-10*
physical, *5-2 thru 5-3*, 5-10
single drive system, *4-37, E-1*
errors, B-2 thru B-5, B-11 thru B-13, B-14 thru B-15, B-27 thru B-29
files, 4-14, *5-5*, 8-80
formatting, 3-25, 3-27, *5-28*
initializing, 3-15, 3-24, 3-39, *5-28*
insertion into drive, 3-12 thru 3-13
media, *3-11*, 8-81
non-system, *3-15*, 4-6, 5-32, 8-92, 8-93
removal from, 3-12 thru 3-13
system, *3-15*, 3-25, 3-27, 5-29, 8-92, 8-93
diskette, See disk
display
CREDIT, *6-1, 6-4*
input, 8-18
ISIS-PDS, 3-10
output, 8-23
documentation, 1-8
dot operator in PL/M, 8-9
drive
configuration, A-26
external, 1-4, *3-10 thru 3-11*
logical, *5-4*, 5-10
installation of, *A-3*
integrated, 1-2, *3-10*
internal, 1-2, *3-10*
operation of, 3-10 thru 3-11
physical, *5-2 thru 5-3*, 5-10
single disk drive system, *4-37, E-1*

E command
DEBUG, 7-24
echo file, *5-4*, 8-81

echo system call parameter, 8-11
editing
command lines, *3-21 thru 3-22*, 5-49
files, 6-1
electrical specifications
line printer interface, A-26
main chassis, A-29
serial interface, A-14
emulation, 1-6, 1-10, *2-5 thru 2-6*
emulators, 1-6, 1-10, *2-5 thru 2-6*
entry$point system call parameter, 8-11
entry$ptr system call parameter, 8-11
EPROM erasure, 10-2
E^2PROM, 10-1
equates
for assembly language system calls, 8-7 thru 8-8
errnum system call parameter, 8-11
ERROR
CONPDS command, B-22
system call, 8-28
error messages
Confidence Test, B-27 thru B-29
CREDIT, 6-159
DEBUG, 7-5 thru 7-6
ISIS-PDS, B-1 thru B-14
Power on Diagnostics, B-15 thru B-17
errors
fatal, 8-8, *B-3*
non-fatal, 8-8, *B-2*
ESC key
CREDIT, *6-3, 6-5*
DEBUG, 7-5
ISIS-PDS, *3-7*, 3-22, 5-48
examining the contents of
memory, 7-22
registers, 7-38
examples
CREDIT, *4-40 thru 4-48*
DEBUG, *4-54 thru 4-58, 7-10 thru 7-18*
ISIS-PDS, *3-24 thru 3-44, 4-1 thru 4-75, 9-9 thru 9-13*
exceptions
See errors
executing programs, *3-17 thru 3-23*, 8-5
execution control, *2-4*, 7-1, 7-27 thru 7-28, 7-32
EXIT
CONPDS command, B-19
system call, 8-30
exiting
CONPDS, B-19
CREDIT, 6-20
DEBUG, 7-24
extension
filename, 5-5
external
disk drive, 1-4, *3-10 thru 3-11*
labels for assembly language system calls, 8-7 thru 8-8
procedure definitions for PL/M system calls, 8-7
EXTRN
assembly directive, 8-7 thru 8-8

fatal errors, 8-8, *B-3*
file
attributes, 4-18 thru 4-19, 4-67, *5-14*, 8-16, 8-94 thru 8-95
backup, *3-12, 3-15*, 3-25, 3-27, 4-68
blocks, 8-88 thru 8-90
console, 8-80 thru 8-81
copying, 4-21, *5-16*
creating, *5-16, 6-1*
deleting, 4-14, 4-36, *5-22*
disk, 4-14, *5-4*, 8-80
echo, *5-4*, 8-81
editing, 6-1
extensions, 5-5
formats, *8-86*
in IPPS
device, 10-8
formats, 10-9
I/O system calls, 8-4
line edited, 8-82 thru 8-84
file management
console commands, 4-12
names, 5-5
object, 8-86
renaming, 4-22, 4-40, *5-33*
running, 3-17 thru 3-23
text, *8-82*, 8-85
types, 8-85
wildcard names, 5-6
filename, 5-5
Fill Memory command
DEBUG, 7-26
Find Command
CREDIT, 6-72
firmware, 10-1
flexible disk
See disk
formatting disks
backup, 3-25, 3-39
IDISK command, 5-28
non-system disks, 3-15, 4-8, 5-28, 8-92, 8-93
system disks, 3-15, 3-25, 3-27, 5-28, 8-92, 8-93
FUNCT key, *3-6*, 4-1 thru 4-5, 4-60, 5-47, 7-29, 9-1 thru 9-5
function keys, *3-7*, 4-1, 4-2, 4-7, 4-59, 5-47, 7-29, 9-1 thru 9-5
function$code system call parameter, 8-11
functional grouping of commands
DEBUG, 7-7
ISIS-PDS, 4-1, 4-2, 4-7, 4-14, 4-40, 4-52, 4-59, *5-9*
fuses
system chassis, A-4 thru A-6

G COMMAND
DEBUG, 7-23 thru 7-24
Go command
DEBUG, 7-23 thru 7-24

PROM
storage device, 10-7
type, 10-7
prompts
DEBUG, 7-3
IPPS, 10-6
ISIS-PDS, 3-21

Q command
DEBUG, 7-29
Query mode
ATTRIB command, 5-14
COPY command, 5-16
DELETE command, 5-21
IDISK command, 5-28

RAM, 1-3
READ
system call, 8-46 thru 8-47
rear panel, 3-2 thru 3-3
RECOGNIZE
CONPDS command, B-20
registers
changing contents, 7-38
use of in system calls, 8-5 thru 8-8
keywords for in DEBUG, 7-34
RENAME
ISIS-PDS command, 4-22, 4-34, *5-33*
system call, 8-48 thru 8-49
repair assistance, ix
repeat last command
ISIS-PDS, 5-49
replacement character
ISIS-PDS, 5-5
RESCAN
system call, 8-50 thru 8-51
reset the system, *3-5*, 3-7, 3-16 thru 3-17, 9-7
RESET
key, *3-5*, 3-7
resident program, 3-22
RETURN key, *3-5*, 3-8, 3-22, 5-51, 6-4, 6-5
reverse slash
CREDIT, *6-3*
RI (Reader input)
system call, 8-52
ROM, 1-3
RS-232
connector, A-14
interface, A-10
RUBOUT key, *3-8*, 3-22, 5-52, 6-3, 6-5
running programs
Confidence Tests, B-17
DEBUG, 7-2
IPPS, 10-6
ISIS-PDS, *3-25*, 3-27, 3-39

S command
DEBUG, 7-36
screen mode
editing, 6-2 thru 6-4
SEEK
system call, 8-53 thru 8-56
self test diagnostic, B-13
SERIAL command
ISIS-PDS, 4-13, *5-33*
serial
connector, A-13 thru A-14
interface, *8-76*, A-10 thru A-11
configuration, 8-76, *A-10 thru A-15*
service information, ix
shutting down the system, 3-4 thru 3-5
single drive system, *4-37*, E-1
software components, 1-8
SPATH
system call, 8-57 thru 8-59
specifications
electrical, *A-14, A-28, A-29, A-30*
physical, A-29
status$ptr system call parameter, 8-11
Step command
DEBUG, 7-32
SUBMIT
with IPPS, 10-6
SUMMARY command, B-20
symbolic debugging, *2-6*, 7-1
syntax
notation, *5-1 thru 5-9*
SYSPDS.LIB, 7-12, *8-7*, 8-67
system
devices, 5-2 thru 5-4
initialization, *3-15*, 3-24
system calls, 8-3 thru 8-63
system description
add-on mass storage, 1-6
basic unit, 1-2
dual processors, 1-5
multimodules, 1-6
plug-in modules, 1-6
software, 1-7
system disk, *3-15*, 3-25, 3-27, 5-29, 8-92, 8-93

T command
DEBUG, 7-37
teletype, *5-3*, 8-59
temporary files
CREDIT, 6-5 thru 6-8
IPPS, *10-8*
terminating commands, *3-21*, 8-82 thru 8-85, 10-9
TEST command, B-25 thru B-31
tests
confidence tests, B-17
power on diagnostics, B-13
text commands
CREDIT, 6-57
text editor, 4-40, *6-1*
TTY, *5-3*, 8-65

Unpacking the system, A-1
usage examples
  commands, 4-1 thru 4-75
  CREDIT, *4-40 thru 4-51,*
  DEBUG, *4-54 thru 4-59, 7-10 thru 7-16*
  IPPS, 10-74 thru 10-90
  ISIS-PDS, *3-24 thru 3-44, 4-1 thru 4-75, 9-9 thru 9-13*
  system calls, 8-72 thru 8-78
user defined devices, 5-4
user written I/O routines, 5-4, *8-34 thru 8-36,*
    8-64 thru 8-70

Video
  display screen, *3-19 thru 3-10*, 6-1, 6-4
  input, 8-20
  output, 8-23
Virtual address range
  buffer, 10-7 thru 10-8
  file, 10-8 thru 10-9
voltage
  changing the voltage
    main chassis, A-4 thru A-5

WARNING flag, 1-11
WHOCON
  system call, 8-61 thru 8-62
wildcard characters
  ISIS-PDS, 5-6
wraparound
  CREDIT, 6-4
WRITE
  system call, 8-62 thru 8-63
write protected files, *5-16*, 8-16

X command
  DEBUG, 7-33 thru 7-34
    display form, 7-33
    modify form, 7-33

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Intel Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

____
____
____
____
____
____

2. Does the publication cover the information you expected or required? Please make suggestions for improvement.

____
____
____
____
____

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

____
____
____
____
____
____

4. Did you have any difficulty understanding descriptions or wording? Where?

____
____
____
____

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). ____________

NAME ______________________________ DATE ____________

TITLE ____

COMPANY NAME/DEPARTMENT ______________________________

ADDRESS ____

CITY ________________ STATE ____________ ZIP CODE ____________
(COUNTRY)

Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS ...

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

NO POSTAGE
NECESSARY
IF MAILED
IN U.S.A.

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
5200 N.E. Elam Young Parkway.
Hillsboro, Oregon 97123

DSHO Technical Publications

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) 987-8080

Printed in U.S.A.

DS001/3K/0383/AP