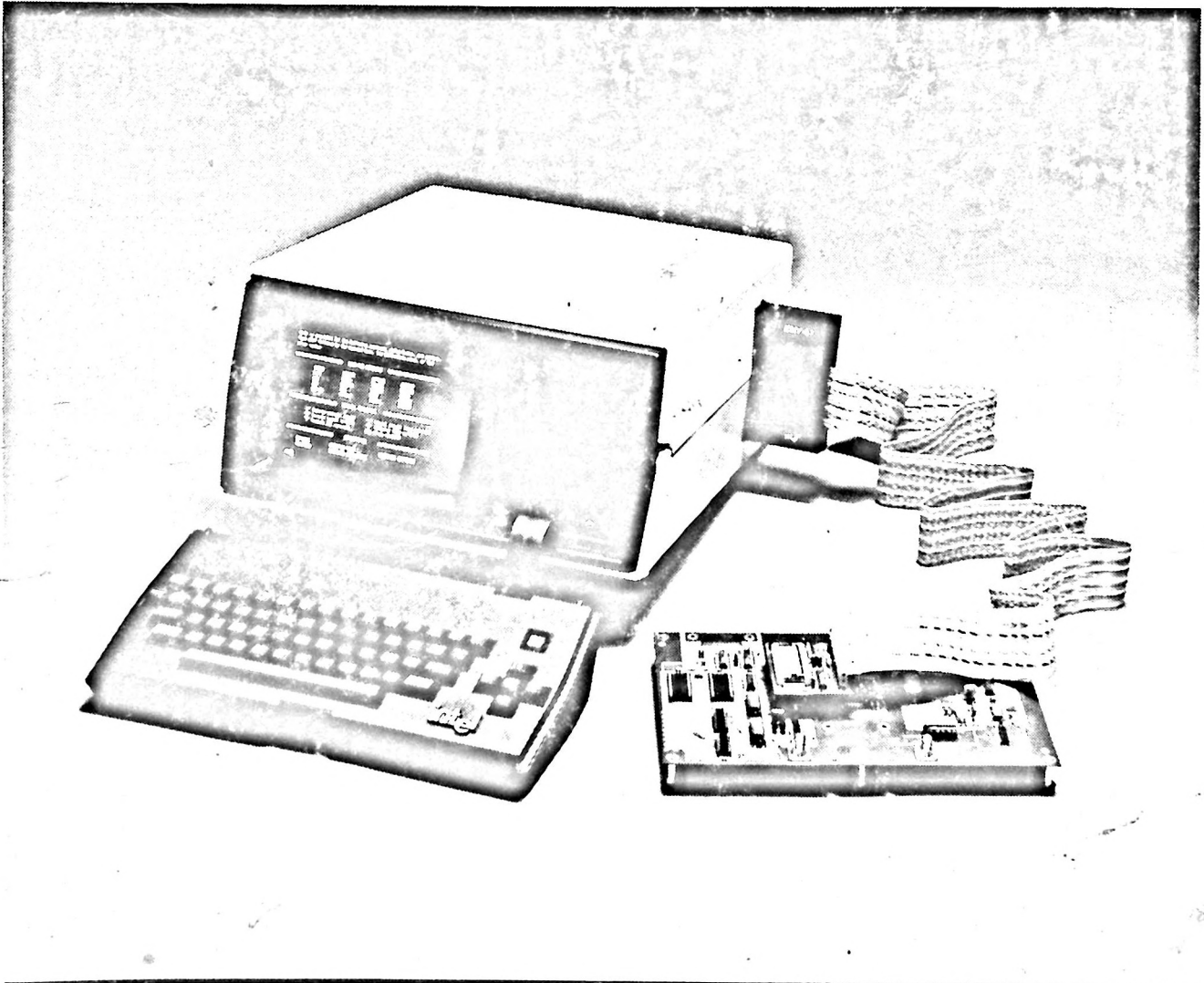


intel

EMV-51 EMULATION VEHICLE USER'S GUIDE



CAUTION

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:

BXP	Insite	IRMX	MULTICHANNEL
CREDIT	Intel	iSBC	MULTIMODULE
i	Television	iSBX	Plug-A-Bubble
i ² -ICE	Intelligent Identifier	iSXM	PROMPT
ICE	Intelligent Programming	Library Manager	RMX/80
iCS	Intellec	MCS	RUPI
iLBX	Intellink	Megachassis	System 20F
im	iOSP	MICROMAINFRAME	UPI
iMMX	iPDS	MULTIBUS	

REV.	REVISION HISTORY	DATE
-001	Original Issue	2/82
-002	Revision and update of manual	10/82



This manual is the user's guide for the EMV-51 Emulation Vehicle. Chapters 1 and 2 contain an overview of the EMV-51. Chapters 3 through 6 provide tutorial sessions to familiarize the initial user with the emulator commands. Chapter 7 contains reference material about the EMV-51 and the MCS-51 family of microprocessors. Chapter 8 contains an alphabetical command dictionary.

Appendix A contains installation instructions for EMV-51. The following is a brief guide to the contents of the manual:

Chapter 1 includes a description of the hardware and software components of the EMV-51. The chapter defines emulation and how emulation is used in the product development cycle.

Chapter 2 contains an overview of the command categories used in the emulator software.

Chapter 3 presents a tutorial session on using the utility commands. It includes a description of each command and gives screen displays showing how each command is used.

Chapter 4 presents a tutorial session on using the display/modify commands. It includes a description of each command and gives screen displays showing how each command is used.

Chapter 5 presents a tutorial session on using the emulation commands. It includes a description of each command and gives screen displays showing how each command is used.

Chapter 6 presents a tutorial session on using the advanced commands. It includes a description of each command and gives screen displays showing how each command is used.

Chapter 7 contains reference material on the MCS-51 family of microprocessors and the EMV-51.

Chapter 8 contains an alphabetical listing of the EMV-51 commands including the format requirements.

Appendix A contains installation instructions and technical reference material about the EMV-51.

Appendix B lists the error and warning messages.

Appendix C contains instructions for running the EMV-51 Confidence Tests.

The reader should reference the following material to obtain additional information about the MCS-51 family of microprocessors:

Microcontroller Applications Handbook, Order Number 210316
Microcontroller User's Manual, Order Number 210359
MCS-51 Macro Assembler User's Guide, Order Number 9800937
Component Data Catalog, Order Number 210298



SERVICE AND REPAIR ASSISTANCE

The best service for your Intel product is provided by an Intel Customer Engineer. These trained professionals provide prompt, efficient, on-site installation, 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 Intel 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:

- a. The date you received the product.
- b. 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 usually stamped on a label.
- c. 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.
- d. The shipping and billing address.
- e. If the Intel Product warranty has expired, a purchase order number is needed for billing purposes.
- f. Be sure to advise the Center personnel of any extended warranty agreements that apply.

Use the following telephone 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 the 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.



CHAPTER 1	PAGE
INTRODUCTION	
Introduction	1-1
EMV-51 Components	1-1
Hardware Components	1-2
Controller	1-2
Emulator	1-2
Software Components	1-2
User Publications	1-3
8751/8051/8031 Architecture Overview	1-3
Designing With The 8051	1-3
Purpose of Emulation	1-3
Definition of Emulation	1-4
Before Emulation	1-4
With Emulation	1-5
Symbolic Debugging	1-6
A Generalized Emulation Session	1-6
Emulation Procedure	1-6
Chapter Preview	1-10

CHAPTER 2	PAGE
USING EMV-51	
Introduction	2-1
Command Categories	2-1
Prompts And Messages	2-1
Comment Lines	2-2
Entering Commands	2-2
Entering the Command Line	2-2
Command Line Execution	2-2
Continuation Lines	2-2
Entry Editing and Display Control	2-3
Command Line Editing	2-3
Controlling The Display	2-3

CHAPTER 3	PAGE
UTILITY COMMANDS	
Introduction	3-1
HELP Information	3-1
HELP Information Screen Displays	3-1
Beginning An Emulation Session	3-2
The LOAD Command	3-2
Loading Multiple Programs	3-3
Copy An Emulation Session	3-3
Setting Up A LIST File	3-3
The LIST Command	3-3
Closing A LIST File	3-4
Load and LIST Screen Display	3-4
Symbolic Reference Commands	3-5
Assigning Symbol Names	3-5

	PAGE
The DEFINE Command	3-3
Manipulating Symbols	3-5
Displaying Symbol Names	3-6
Displaying Individual Names	3-6
Displaying Multiply Defined Names	3-6
The SYMBOLS Command	3-6
Removing Symbol Names	3-6
The REMOVE Command	3-6
Symbolic Reference Screen Displays	3-7
Enabling/Disabling Symbol Display	3-7
The DISABLE Command	3-8
The ENABLE Command	3-8
Evaluating Symbol Values	3-8
The EVALUATE Command	3-8
Evaluate Screen Display	3-8
Number Display Format	3-9
The SUFFIX and BASE Commands	3-9
SUFFIX and BASE Screen Displays	3-9
Saving The User's Program	3-9
The SAVE Command	3-10
Initializing EMV-51 Hardware	3-10
The RESET Command	3-10
Ending An Emulation Session	3-10
The EXIT Command	3-10
Using Utility Commands	3-10

CHAPTER 4	PAGE
DISPLAY/MODIFY COMMANDS	
Introduction	4-1
Using Register Commands	4-1
Displaying Register Contents	4-1
Modifying Register Contents	4-1
Displaying Register Names and Contents	4-2
The REGISTER Command	4-2
Special Function Key 4	4-2
REGISTER Command Display Screen	4-2
Displaying INTERRUPT Register Contents	4-3
The INTERRUPT Command	4-3
Memory Commands	4-4
Displaying Memory Contents	4-4
Modifying Memory Contents	4-5
Displaying Memory Commands	4-6
The MEMORY Command	4-6
Special Function Key 1	4-6
Assembly/Disassembly Commands	4-7
Assembling User's Program	4-7
The ASM Command	4-7
Disassembling User Programs	4-8
The DASM Command	4-8
Using Display/Modify Commands	4-8



CONTENTS (continued)

CHAPTER 5 USING EMULATION COMMANDS

	PAGE
Introduction	5-1
Setting Breakpoints	5-1
Address Breakpoints	5-1
Branch Breakpoints	5-1
Range Breakpoints	5-2
Register Value Breakpoints	5-2
Clearing Breakpoints	5-2
Breakpoint Restrictions	5-2
Using The BREAK Command	5-3
Special Function Key 2	5-3
Breakpoint Screen Displays	5-3
Using Trace Display	5-4
The TD Command	5-4
The TR Command	5-5
The TBn Command	5-5
The TS Command	5-5
Controlling TRACE Display	5-5
The TRn Command	5-6
The TV Command	5-6
The DTRACE Command	5-6
Special Function Key 3	5-7
DTRACE Screen Displays	5-7
Program Emulation Commands	5-8
The STEP Command	5-8
The GO Command	5-9
The P Command	5-9
Emulation Control Screen Displays	5-10

CHAPTER 6 USING ADVANCED COMMANDS

Introduction	6-1
Using Macros	6-1
Defining Macros	6-1
The DEFINE Command	6-1
The EM Command	6-1
Parameter Passing	6-1
Writing Messages	6-2
Macro Definition Screen Display	6-2
Displaying Macro Information	6-2
The DIR Command	6-3
The MACRO Command	6-3
Macro Information Screen Displays	6-3
Executing Macros	6-3
The Macro Execution Command	6-3
Macro Execution Screen Display	6-4
The ENABLE/DISABLE Commands	6-4
Deleting Macros (the REMOVE Command)	6-4
Saving Macros (the PUT Command)	6-5
The INCLUDE Command	6-5
Executing Command Files	6-5
Executing Macro Command Files	6-5

	PAGE
INCLUDE Command Screen Display	6-6
Using Compound Commands	6-7
Setting Up Conditional IF Commands	6-7
Conditional IF Screen Display	6-8
Setting Up the REPEAT Command	6-11
REPEAT Command Screen Displays	6-11
Setting up the COUNT Command	6-12
Using Special Function Keys	6-12
Assigning Function Keys to Macros	6-13
Function Key Precautions	6-13
Displaying Assigned Function Keys	6-13
Advanced Commands Demonstration	6-13

CHAPTER 7 EMV-51 REFERENCE MATERIAL

Introduction	7-1
EMV-51 Symbols	7-1
EMV-51 Operators	7-2
Arithmetic Operators	7-2
Content Operators	7-2
Code Memory (CBYTE)	7-2
CBYTE Restrictions	7-3
On-Chip Data and Register Memory (DBYTE, RBIT, and RBYTE)	7-3
On-Chip Data Memory (DBYTE and RBIT)	7-3
Special Function Register Memory (RBYTE)	7-3
External Data Memory (PBYTE)	7-3
Relational Operators	7-6
Logical Operators	7-6

CHAPTER 8 COMMAND DICTIONARY

	PAGE
Introduction	8-1
Entering Commands	8-1
Entering the Command Line	8-1
Specifying the Workfile	8-1
Command Line Execution	8-2
Continuation Lines	8-2
Entry Editing and Display Control	8-2
Command Line Editing	8-2
Controlling the Display	8-3
Form of EMV-51 Commands	8-3
Notational Conventions	8-3
Command Description Formats	8-5
Special Command Format Terms	8-5
Expression	8-6
String	8-6
Source	8-6



CONTENTS (continued)

	PAGE
Destination	8-6
Command	8-7
Condition	8-7
Alphabetical Listing of Commands	8-7
ACCUMULATOR	8-7
ASM	8-8
B	8-9
BASE	8-10
BC	8-10
BRB	8-11
BREAK	8-11
BRR	8-12
BR<n>	8-13
BV	8-14
CBYTE	8-15
CDUMP	8-16
COUNT	8-17
DASM	8-18
DBYTE	8-19
DDUMP	8-20
DEFINE	8-20
DIR	8-21
DISABLE	8-22
DPTR	8-22
DTRACE	8-23
ENABLE	8-24
EVALUATE	8-24
EXIT	8-25
FUNCTION	8-26
GO	8-27
HELP	8-28
IF...THEN...ELSE	8-28
INCLUDE	8-30
INTERRUPT	8-30
LIST	8-31
LOAD	8-32
MACRO	8-33
MEMORY	8-34
P	8-34
PBYTE	8-35
PC	8-36
PSW	8-37
PUT	8-38
RBIT	8-39
RBS	8-40
RBYTE	8-40
REGISTER	8-41
R<n>	8-42
REMOVE	8-43
REPEAT	8-44
RESET	8-45
SAVE	8-46

	PAGE
SP	8-47
STEP	8-48
SUFFIX	8-49
SYMBOLS	8-50
TM0	8-50
TM1	8-51
WRITE	8-52
(macro execution)	8-52
(symbol handling)	8-53

APPENDIX A INSTALLATION

Introduction	A-1
Installing the EMV-51	A-1
Installation Considerations	A-1
Installation Procedures	A-1
Jumper Configurations	A-5
Selecting the Program Memory	A-5
Selecting Supply Voltage	A-6
Selecting Oscillator Input	A-6
Selecting Reset Input	A-6
User Test Signals	A-6
Port 0 in Stand-Alone Mode	A-7

APPENDIX B ERROR MESSAGES

APPENDIX C CONFIDENCE TEST

Confidence Test	C-1
PCONF Command	C-1
INIT E51CON Command	C-1
Confidence Test Commands	C-2
Clear Command	C-2
Describe Command	C-2
Error Command	C-3
Exit Command	C-3
Ignore Command	C-3
List Command	C-4
Recognize Command	C-4
Summary Command	C-4
Test Command	C-5
Confidence Test Error Messages	C-9



FIGURES

FIGURE	TITLE	PAGE	FIGURE	TITLE	PAGE
1-1	Complete Development System	1-1	A-2	Installation In The Personal Development System	A-3
1-2	Typical Microcomputer Development Cycle ..	1-4	A-3	Installation of Emulator Module in Prototype	A-4
1-3	Development Cycle With EMV-51 Module ..	1-5	A-4	EMV-51 Emulator Jumper Locations	A-6
1-4	Emulator Module Connected to Prototype ...	1-6	A-5	EMV-51 Controller's Terminal Post Location	A-7
1-5	Listing of Demonstration Program	1-11	A-6	Port 0 Pullup Configuration	A-8
7-1	Internal Data Memory Organization	7-4			
8-1	Format of Command Descriptions	8-5			
A-1	EMV-51 Emulator Hardware Components ..	A-2			



TABLES

TABLE	TITLE	PAGE	TABLE	TITLE	PAGE
2-1	Chapter Breakdown by Command Category	2-1	7-5	Bit-Addressable Register Memory Contents ..	7-5
4-1	Register Keyword Names	4-1	7-6	Relational Operators	7-6
4-2	Memory Areas and Keyword Names	4-4	7-7	Logical Operators	7-6
7-1	System Symbols	7-1	8-1	Available Number Bases and Digits	8-6
7-2	Punctuation and Delimiter Keywords	7-1	A-1	EMV-51 Emulator Jumper Descriptions	A-5
7-3	Arithmetic Operators	7-2	A-2	User Test Signals	A-7
7-4	Content Operators	7-2			

Introduction

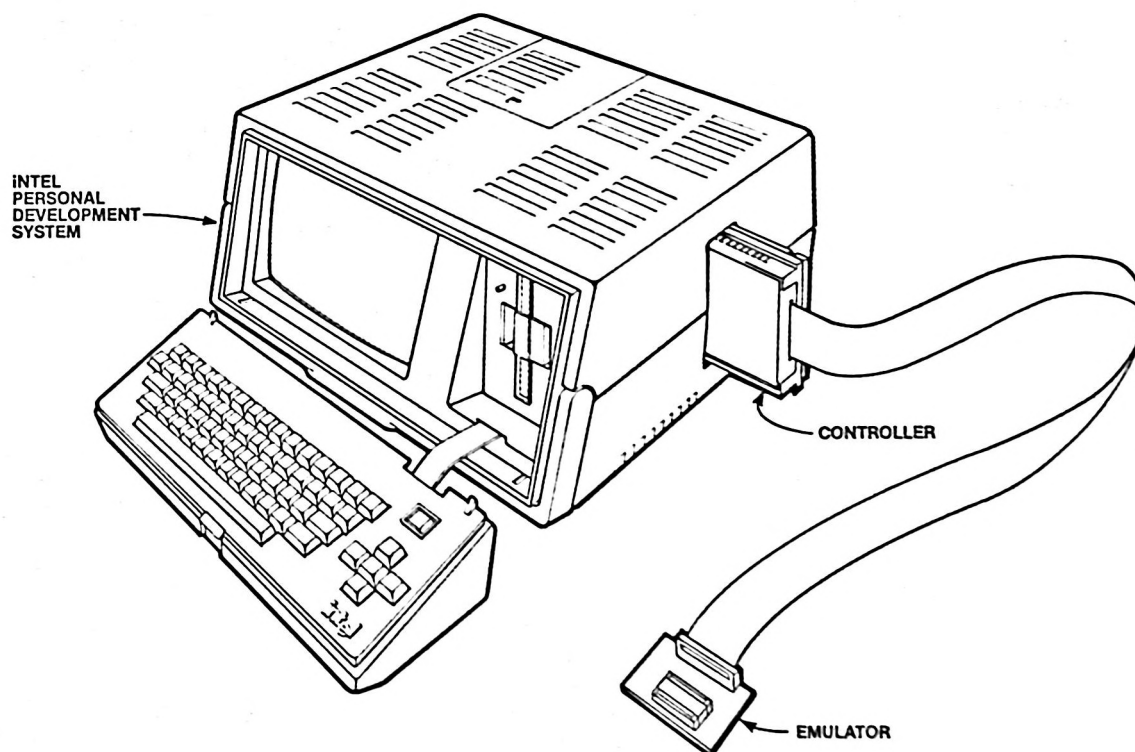
The EMV-51 Emulation Vehicle is an emulation tool for the MCS-51 family used with the Personal Development System (iPDS). This chapter covers the following topics:

- EMV-51 Components
- 8751/8051/8031 Architecture Overview
- Purpose of Emulation
- A Generalized Emulation Session

EMV-51 Components

The EMV-51 emulator is composed of hardware and software components. The hardware includes the controller, the emulator, and the cable connecting the controller and emulator. The software component is the emulator software on a diskette.

Figure 1-1 shows the controller connected to the iPDS and the cable connecting the controller and emulator. Appendix A describes the procedure for preparing the emulator for operation.



0151

Figure 1-1 Complete Development System

Hardware Components

The hardware components are:

- Controller
- Emulator
- Cable Assembly

Controller

The controller plugs into the side of the iPDS. It connects to the emulator via an 80 inch 50-conductor flat cable.

The controller's user accessible terminal posts supply the test signals listed below to the user's test equipment. Appendix A contains additional information on these signals.

- Instruction Fetch Clock (CLK)
- Address latch Enable (ALE)
- Program Store Enable (PSENE/)
- Emulation (EMUL)
- External Break Input (EXTBRK)
- Ground (GND)

Emulator

The emulator replaces the microcontroller in the user's product. It contains the following parts:

- 8051 microcontroller and integrated circuit buffers
- 12 MHz crystal
- Jumper connections for power, clock, source, reset input, and the EA/pin
- 40 pins exactly like the 8051/8031/8751 microcontroller

The emulator receives power from the iPDS development system or the user's product. The clock signal is supplied by the user's product or by a 12 MHz crystal located on the emulator circuit board. The user selects the source of power and clock signals via jumper connections, described in Appendix A.

The reset jumper enables the user to select the reset input to the emulator's 8051. Either the prototype system reset or a 10uf reset capacitor on the emulator circuit board may be selected. The choice between internal emulator memory or external user memory is determined by the state of the EA/ pin. Additional information about reset input and the EA/ pin is contained in Appendix A.

Software Components

The emulator software is furnished on a 5 1/4" diskette under the file name EMV51. A demonstration program is also furnished under the file name EMVDEM.OBJ. Figure 1-5 shows a listing of the program.

The software is divided into three files. One file controls the iPDS and EMV-51 controller, while the second and third files contain error and general information messages.

Chapter 2 describes the procedure to load the software from the diskette.

User Publications

The EMV-51 package includes instructional and reference manuals to assist the user in operating the system. These materials include the following:

- EMV-51 User's Guide
- EMV-51 Pocket Reference listing each emulator command in alphabetical order with a summary of its format and usage.

8751/8051/8031 Architecture Overview

The MCS-51 family consists of the 8031, the 8051, and the 8751 microcontrollers. The 8031 contains no internal program memory. The 8051 contains 4K of internal Read-Only-Memory (ROM) that is programmed during the chip manufacture. The 8751 contains 4K of internal Eraseable Programmable Read-Only-Memory (EPROM) which the user programs with a PROM Programmer.

In addition, the MCS-51 family of chips contains 32 general purpose registers, 32 I/O lines, two 16-bit timer/event counters, a multiple source, nested interrupt structure, a serial I/O channel, and an on-chip oscillator and clock circuit. The microcontrollers can address 64K bytes of program memory and 64K bytes of external data memory.

The *Microcontroller User's Manual*, manual order number 210359, contains additional information about the MCS-51 family of chips.

Designing With the 8051

Due to the complexity of the MCS-51 microcontrollers, traditional design procedures using the oscilloscope and logic analyzer cannot be used. The 8051 (used collectively to indicate the 8051, 8751, and 8031) is a computer-on-a-chip requiring advanced design techniques. The engineer needs the capability of emulating new product designs with another computer to develop and debug microcontroller/microcomputer products.

Purpose of Emulation

The tools required for developing a microcontroller or microcomputer based product differ from the tools used in the development of electronic products not using micros.

For microcontroller and microcomputer based products, traditional electronics development tools alone are not sufficient for two reasons. First, they do not support software development as required in microcontroller and microcomputer based products. 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.

When used with the iPDS; the EMV-51 emulation vehicle provides the designer with the tools necessary to design, develop, and debug microcontroller/microcomputer products.

Definition of Emulation

Emulation is the controlled execution of the prototype software in an artificial hardware environment that duplicates the microcontroller of the prototype. The EMV-51 creates the artificial hardware environment for all applications using the MCS-51 family of microcontrollers.

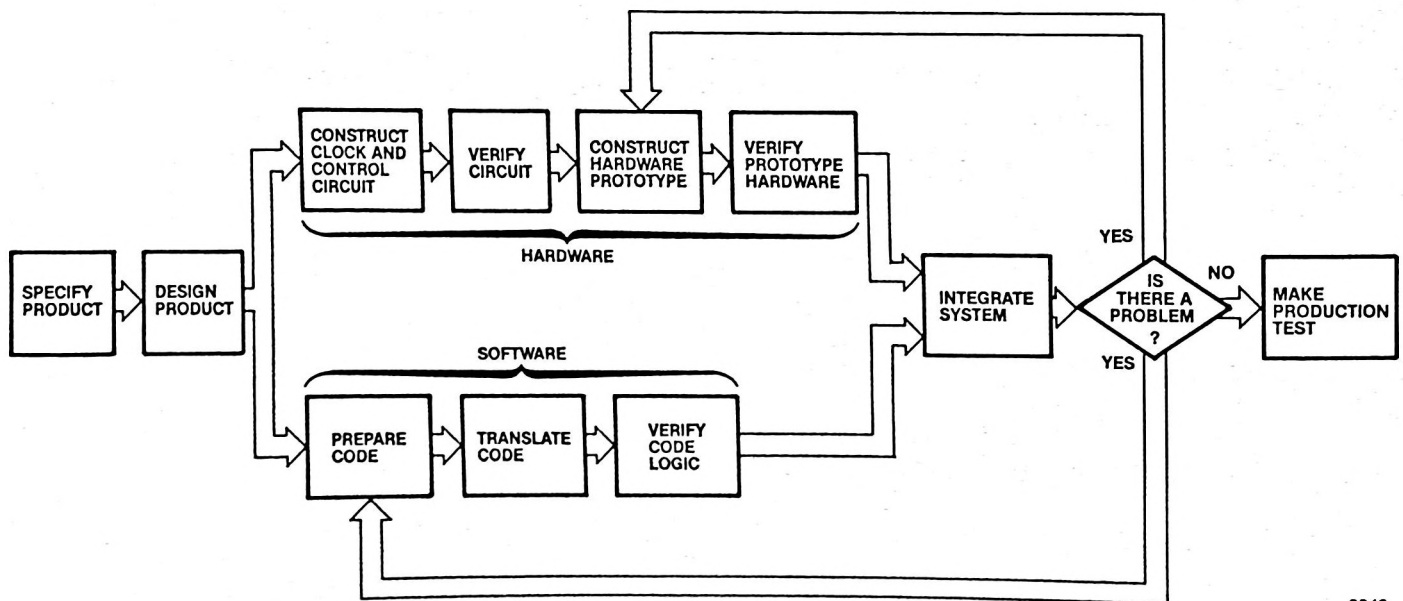
Emulation allows the design engineer to interactively debug a product's microcontroller-based program code. Emulators permit reading and writing of microcontroller registers and system memory, program disassembly, and interactive control of program execution. One value of emulation is the ability to externally control program execution while operating in the user's prototype hardware.

The EMV-51 incorporates an 8051 microcontroller that replaces the microcontroller in the user's product, thus providing a window into the user's prototype system. With one tool, the engineer watches the hardware and software interact within the prototype system, pinpointing problems in minutes that otherwise takes days to find.

The EMV-51 connects between the iPDS and the user's prototype and provides complete control over the prototype. The EMV-51 contains 4K of RAM that can replace the user's ROM during emulation and permits the engineer to debug the prototype hardware and software as a unit. This procedure is described in more detail in the section titled WITH EMULATION.

Before Emulation

Prior to using emulators, the design of a new product involved a time-consuming iterative process as illustrated in figure 1-2.



0248

Figure 1-2 Typical Microcomputer Development Cycle

As figure 1-2 indicates, each phase of the design progressed separate from the other. The engineer developed and verified the hardware prototype, then developed and verified the software to control that prototype. The software could not be combined with the prototype hardware until both components were completely verified. If problems arose during the integration of the two, the costly design and verification phase had to be repeated until the problem was resolved.

WITH EMULATION

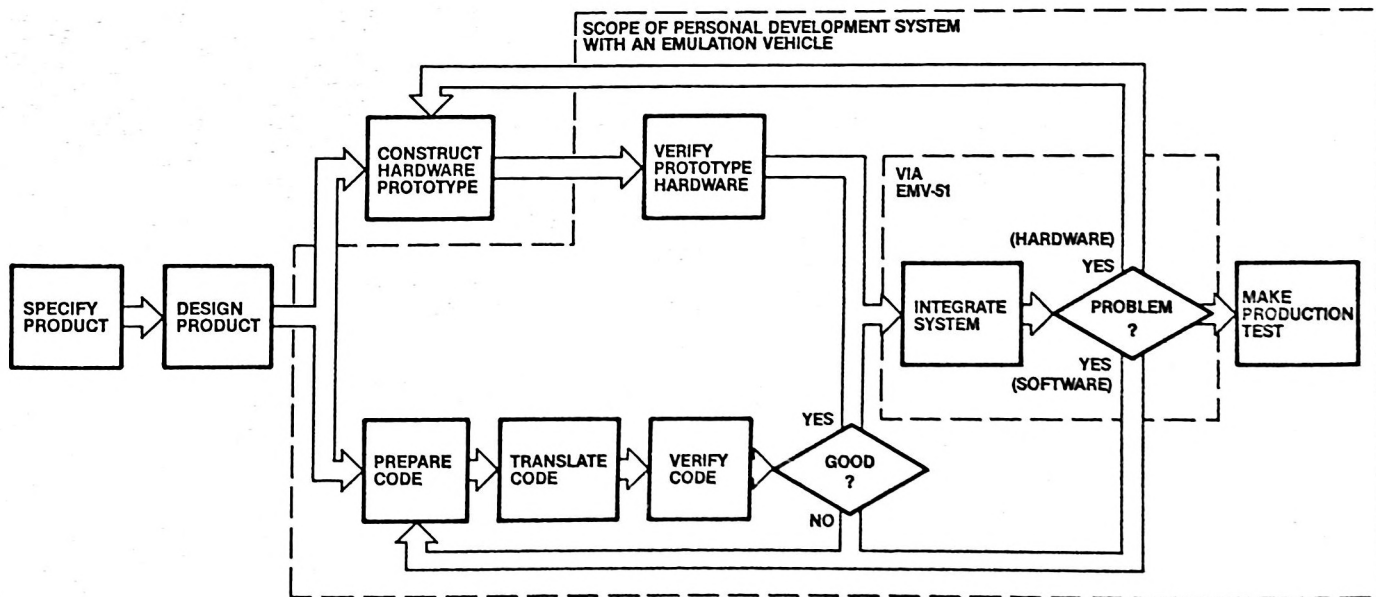
When using an emulator, the product design procedure is shortened considerably. Figure 1-3 illustrates the design procedure using emulation.

As figure 1-3 illustrates, the design procedure with emulation eliminates four design steps. After designing the product, the engineer proceeds immediately to constructing the prototype hardware. Another advantage of emulation, then, is that it permits the engineer to begin testing and debugging the software for a product without waiting for the hardware prototype to be built.

When using emulation techniques, the engineer eliminates the clock and control circuit construction and verification phase. At the same time as the hardware prototype is being constructed, the program code is developed and translated using the development system.

After the hardware prototype is constructed and the program code is developed, the engineer connects the emulator between the development system and the prototype and verifies the hardware and software as a unit. This phase eliminates the separate hardware verification, software verification, and system integration steps.

The emulator provides immediate recognition of problems between the hardware and software. When a software problem is identified, the engineer corrects the problem with the development system and continues with product emulation.



0249

Figure 1-3 Development Cycle With EMV-51 Module

Symbolic Debugging

During emulation, the engineer can refer to physical addresses and data values with symbolic names. This procedure, called symbolic debugging or symbolic reference, improves the engineer's ability to debug the software by removing the need to remember physical numbers. If the contents of the program change, the symbolic names indicate the same addresses or data values.

Chapter 3 contains additional information about symbolic debugging. The next section presents a generalized emulation session that demonstrates some of the emulator's power.

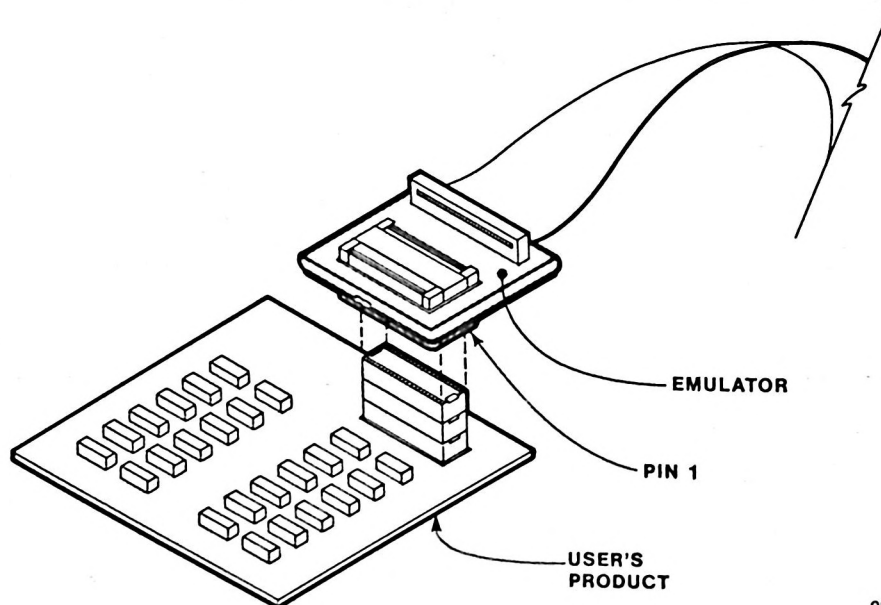
A Generalized Emulation Session

This section describes the main steps in an emulation session. The user should do each step as described to gain familiarity with EMV-51. Other system features are discussed in Chapters 3 through 6.

Emulation Procedure

1. Turn off iPDS development system power.
2. Plug the EMV-51 controller into the iPDS system (described in Appendix A).
3. If any prototype hardware is used, remove the socket protector from the emulator and attach the emulator to the prototype hardware via the 40-pin socket. If the prototype contains other sockets around the 40-pin socket, insert additional 40-pin sockets between the emulator and the prototype. Figure 1-4 shows the emulator connected to the prototype.

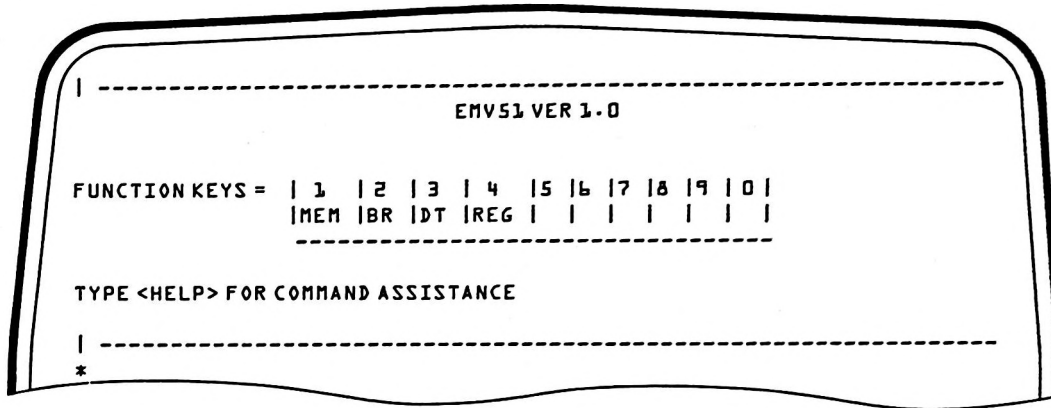
If the prototype hardware is not used, leave the socket protector attached to the emulator. The emulator then functions as a stand-alone 8051.



0252

Figure 1-4 Emulator Module Connected to Prototype

4. Enter the EMV51 command and obtain the sign-on message from the EMV-51 emulator software.



Key-in Sequence

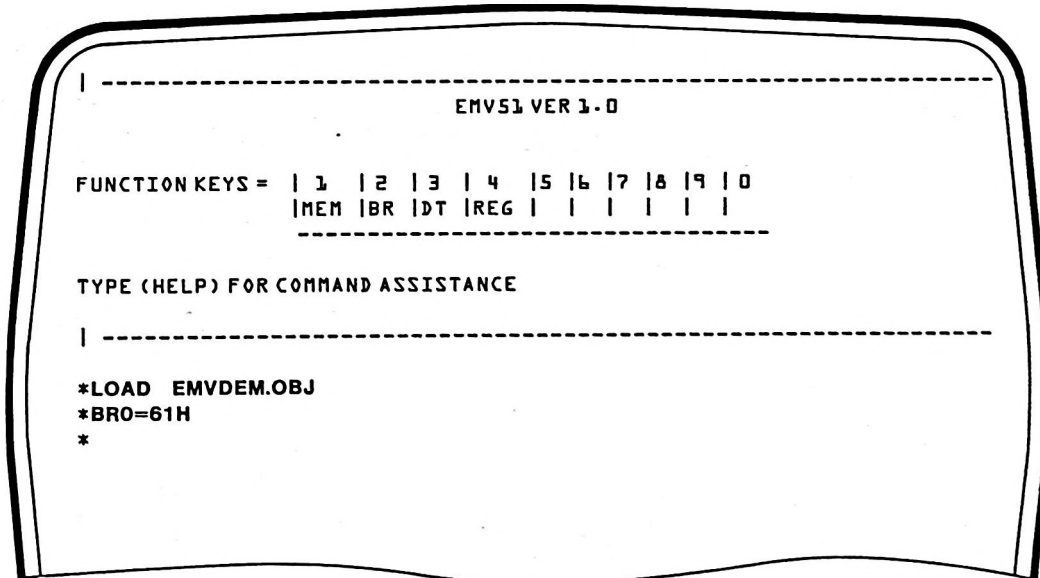
Comments

EMV51

Enter EMV51. The ISIS-PDS operating system loads and executes the EMV-51 software. The software displays the sign-on message and asterisk prompt shown in this display.

5. Load the prototype program from diskette into the EMV-51 memory, using the LOAD command.

6. Prepare the system for emulation by setting the emulation breakpoints. A breakpoint is a specific location in a program where execution of the program should be halted. A breakpoint can be an address, a range of addresses, a register value, or a program execution branch.



Key-in Sequence

Comments

LOAD EMVDEM.OBJ

The LOAD command gets a program from a disk file and loads it into EMV-51 program memory. In this example, the program labeled EMVDEM.OBJ is loaded into memory. The emulator software responds with the asterisk prompt when the LOAD command is completed.

BR0=61H

Set a breakpoint at address 61H

The BREAK (BR) command displays a table containing a list of all the breakpoints. The following screen display illustrates the BREAK command being entered and the address value breakpoint specified.

```

*BR
-----
                BREAKPOINT SETTINGS                |                TYPE
-----|-----
| BR0 = 0061H   BR1 = OFF  BR2 = OFF  BR3 = OFF   | location
| BRR =  OFF                                         | range
| BRB =  OFF      (go mode only)  BC disables all | branch
| BV =  OFF      (step mode only)  breakpoints    | value
-----|-----
*
-----

```

Key-in Sequence

Comments

BR 

In this example, when the user entered the BREAK command and the emulator software displays the breakpoint table, one address breakpoint is set from a previous operation. Breakpoint BR0 is set at address 0061H.

7. Display the demonstration program to verify that it was loaded correctly. The DASM command disassembles the program and displays it on the screen.

```

*DASM.DEMO TO .BUFFER
.DEMO
| 0040H=MOV    R1,#.BUFFER  DASM
| 0042H=MOV    R2,#03H      DASM
| 0044H=CLR    .CY           DASM
| 0046H=PUSH   .RDX         DASM
.LOOP
| 0048H=MOV    A,@R0        DASM
| 0049H=INC    R0           DASM
| 004AH=INC    R0           DASM
| 004BH=INC    R0           DASM
| 004CH=ADD    A,@R0       DASM
| 004DH=DA     A           DASM
| 004EH=PUSH   .ACC        DASM
| 0050H=ANL   A,#0FH       DASM
| 0052H=ORL   A,#30H       DASM
| 0054H=MOV   @R1,A        DASM
| 0055H=INC   R1           DASM
| 0056H=POP   .ACC        DASM
| 0058H=SWAP  A           DASM
| 0059H=ANL   A,#0FH       DASM
| 005BH=ORL   A,#30H       DASM
| 005DH=MOV   @R1,A        DASM
| 005EH=INC   R1           DASM
| 005FH=DEC   R0           DASM
| 0060H=DEC   R0           DASM

```

Key-in Sequence

Comments

DASM .DEMO TO .BUFFER



The DASM command disassembles the contents of program memory and produces the mnemonic equivalents of the instructions. In this example, memory from symbolic address .DEMO to .BUFFER is disassembled. The example continues on the following screen display.



To halt scrolling of the screen display, press the CTRL key and the S key simultaneously.



To continue, press the CTRL-Q Key combination.

```

      | 0061H=DJNZ   R2,.LOOP   DASM
      | 0063H=JNC   .CLRMSD    DASM
      | 0065H=MOV   @R1,#31H   DASM
      | 0067H=SJMP  .DONE      DASM
.DCLRMSD | 0069H=MOV   @R1,#30H   DASM
.DONE    | 006BH=POP  .RDX      DASM
.BUFFER  | 006DH=SJMP .DONE      DASM
*        | 006FH=XCH  A,R4      DASM
    
```

8. Enter the GO command to begin continuous emulation. The EMV-51 software begins emulation at the address specified with the command entry or at the current address in the 8051 program counter. Emulation continues at the full clock rate until a breakpoint is encountered.
9. When a breakpoint is encountered, the EMV-51 hardware stops emulation. At this point, instruct the emulator software to display the previous two instructions emulated and determine if the instructions are performing the desired operation.
10. By alternating between setting breakpoints and emulation of the program, the user can quickly test and debug the prototype program.
11. At the end of the emulation session, the debugged code can be saved on an ISIS-PDS diskette file, using the EMV-51 SAVE command.

```

*GO FROM .DEMO
| 0061H=DJNZ R2,.LOOP GO-BREAK
*P
| 0060H=DEC R0 DASM
| 0061H=DJNZ R2,.LOOP DASM
*SAVE:F0:EMVDEM.V10
*

```

Key-in Sequence**Comments****GO FROM .DEMO**

The GO command emulates the user's program from an address entered with the command, or from the address contained in the program counter. The emulation continues until a breakpoint is reached. When a breakpoint is reached, the emulator software displays the cause of the breakpoint and the address where it occurred.



After an emulator is performed, enter the P command to display the last two instructions emulated. In this example, the instructions DEC and DJNZ are the last two instructions emulated with the GO command.

SAVE:F0:EMVDEM.V10 Save the prototype program in a disk file prior to terminating the session. In this case the program is saved in a file labeled EMVDEM.V10.



This introduction shows some of the scope and power of the EMV-51 emulator in operation, and suggests how this integrated software/hardware design aid can fit into the user's development cycle.

In summary, emulation provides the following advantages:

- Controlled execution of software so the program can be monitored while being run,
- Ability to start software development prior to completion of a working hardware prototype,
- Use of symbolic references to avoid a need to remember physical addresses.

Chapter Preview

Chapter 2 presents an overview of the emulator software command categories. Chapters 3 through 6 describe command usage and present tutorial sessions involving the demonstration program. Chapter 7 contains 8051 and EMV-51 reference material, including memory maps of the 8051 and key definitions used throughout the emulator software. Chapter 8 is a command dictionary containing command format notations and an alphabetical listings of EMV-51 commands, including command formats and examples of commands usage. Appendix A contains installation instructions for the EMV-51 emulator.

Figure 1-5 is a listing of the demonstration program used in following chapters.


```

;
;           EMV-51 DEMONSTRATION PROGRAM
; THIS PROGRAM ADDS TWO NUMBERS IN PACKED BCD FORMAT, AND
; CONVERTS THE SUM TO AN ASCII STRING. THE TWO NUMBERS ARE
; STORED IN CONSECUTIVE MEMORY BYTES AS:

```

```

; R0 ->  FIRST BYTE =    LEAST AND NEXT LEAST SIGNIFICANT DIGITS OF
;           SECOND BYTE = NEXT TWO MOST SIGNIFICANT DIGITS OF AUGEND
;           THIRD BYTE =  MOST AND NEXT MOST SIGNIFICANT DIGITS OF
;           FOURTH BYTE = \
;           FIFTH BYTE =  > SAME FOR ADDEND
;           SIXTH BYTE =  /

```

```

; THE RESULT IS RETURNED IN A STRING POINTED TO BY R1.
;*****

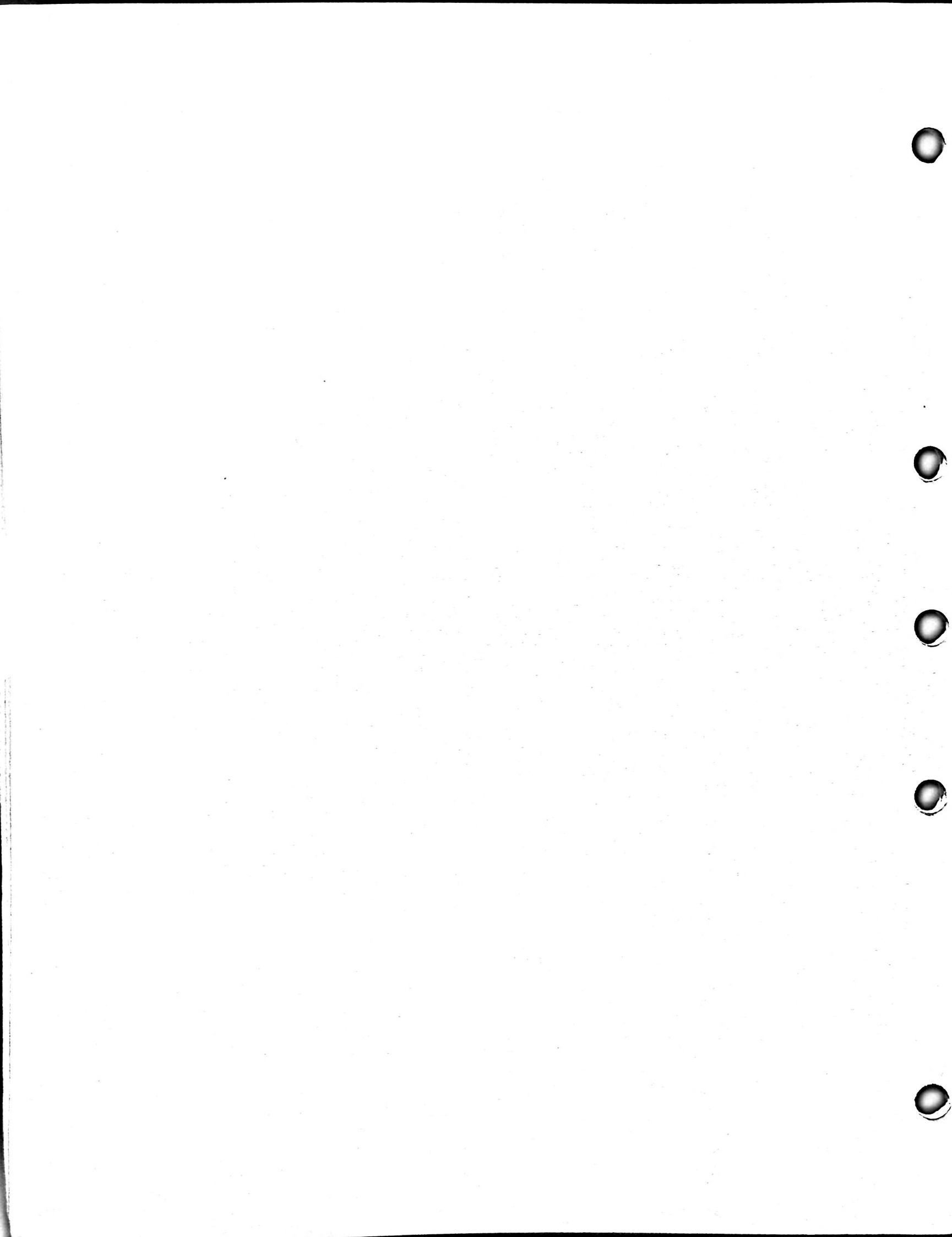
```

```

;*****
;
; DEMO:  ORG    0040H
;        ROX    EQU 00H    ;PSEUDONYM FOR R0
;        MOV    R1,#BUFFER ;GET POINTER TO RESULTS AREA
;        MOV    R2,#03    ;SET BCD DIGITS COUNTER
;        CLR    CY        ;GET READY FOR ADDITION ACTION
;        PUSH   ROX       ;SAVE PARAMETER POINTER
; LOOP:  MOV    A,@R0     ;GET TWO BCD DIGITS FROM AUGEND
;        INC    R0        ;POINT TO CORRESPONDING DIGITS IN
;                   ADDEND
;        INC    R0        ; ("INC" DOESN'T AFFECT CARRY FLAG)
;        INC    R0
;        ADDC  A,@R0     ;ADD THE TWO TOGETHER
;        DA    A          ;ADJUST FOR DECIMAL ARITHMETIC
;        PUSH  ACC       ;SAVE RESULT
;        ANL   A,#0FH    ;GET LOWER DIGIT
;        ORL   A,#30H    ;CONVERT LOWER DIGIT TO ASCII
;        MOV   @R1,A     ;PUT INTO RESULT BUFFER
;        INC   R1        ;POINT TO NEXT RESULT BUFFER LOCATION
;        POP   ACC       ;GET BACK RESULT OF ADDITION
;        SWAP  A         ;GET UPPER DIGIT
;        ANL   A,#0FH    ;
;        ORL   A,#30H    ;CONVERT TO ASCII
;        MOV   @R1,A     ;PUT INTO RESULT BUFFER
;        INC   R1        ;POINT TO NEXT RESULT BUFFER LOCATION
;        DEC   R0        ;POINT TO NEXT TWO BCD DIGITS IN AUGEND
;        DEC   R0        ; ("DEC" DOESN'T AFFECT CARRY FLAG)
;        DJNZ  R2,LOOP   ;LAST PAIR OF BCD DIGITS ADDED?
;                   ; NO—ADD TWO MORE
;                   ; YES—CHECK LAST DIGIT OF SUM
;                   ; CARRY OUT OF 6 DIGIT ADDITION?
;                   ; YES-PUT AN ASCII "1" IN RESULT BUFFER
;
;        JNC   CLRMSD    ;
;        MOV   @R1,#31H  ;
;        SJMP  DONE      ;
; CLRMSD: MOV   @R1,#30H  ; NO-PUT AN ASCII "0" IN RESULT BUFFER
; DONE:   POP   ROX      ; RESTORE PARAMETER POINTER
;        SJMP  DONE
;
; BUFFER: DS   10H      ; BUFFER TO HOLD ASCII RESULT STRING
;        END

```

Figure 1-5 Listing of Demonstration Program





Introduction

This chapter presents an overview of the command categories. It describes how to enter commands and defines basic terms used throughout the manual. The following topics are covered:

- Command Categories
- Prompts and Messages
- Comment Lines
- Entering Commands
- Entry Editing and Display Control

Command Categories

Table 2-1 lists the four categories of EMV-51 commands and the chapters describing each category. The paragraphs following the table give a brief description of each category.

Table 2-1 Chapter Breakdown by Command Category

CATEGORY	USAGE CHAPTER	DEFINITION CHAPTER
Utility	3	8
Display/Modify	4	8
Emulation	5	8
Advanced	6	8

The utility commands are used to load user's programs from diskette, establish symbolic names, save user's programs when the emulation session is completed, and operate on the emulator hardware.

The Display/Modify commands are used to display or change register and memory contents. This category includes commands to assemble new instructions into code memory and to disassemble the contents of code memory.

The Emulation commands are used to start and stop emulation of the user's program. This category includes commands to set up various emulation controls.

The Advanced commands are used to build macro definitions and to set up conditional control structures. This category includes commands that define macros, load and save macro definitions, and assign special function keys to macros.

Prompts and Messages

The EMV-51 software displays an asterisk (*) at the beginning of a line to indicate software readiness for entry of command lines. If a macro is being defined (described in Chapter 6), the EMV-51 software displays a special prompt (.*) during the macro definition. Entering the Write command (described in Chapter 6) with an embedded RETURN, causes the EMV-51 software to display a double asterisk (**) prompt until the Write command is completed.

The emulator software displays error or warning messages indicating the status of operations. The messages are only displayed when an operation generates erroneous results. Appendix B contains a complete listing of the error and warning messages.

Comment Lines

Comments may appear anywhere in the command line. Comment lines are preceded by a semicolon and ended with a carriage return. The emulator software ignores all text between the semicolon and carriage return.

Entering Commands

EMV-51 operations are controlled by command lines. The emulator software receives command lines from one of four sources:

1. Commands entered from the keyboard.
2. Previously defined command files containing valid EMV-51 commands. After loading the EMV-51 software, the command file is loaded and executed using the INCLUDE command (described in Chapter 6).
3. Macros, defined as a sequence of related commands, stored in a disk file. After loading the EMV-51 software, the macro definition is loaded and executed using the INCLUDE command (macros and the INCLUDE command are described in Chapter 6).
4. Previously defined files loaded and executed using the SUBMIT command. Refer to the ISIS-PDS User's Guide for additional information on the SUBMIT command.

Entering the Command Line

An EMV-51 command line is a sequence of one or more command words separated by spaces and ending with a carriage return. Normally, the command line is limited to 122 characters. In most cases, the command words can be abbreviated to one, two, or three characters. Chapter 8 contains a list of valid command words and their abbreviations.

Command Line Execution

With six exceptions, the carriage return causes the command to be interpreted and, if no format errors are detected, executed immediately. The exceptions are macro commands, continuation lines, the REPEAT command, COUNT command, IF command, and WRITE command. Macro, REPEAT, COUNT, IF and WRITE commands are described in Chapter 6; continuation lines are described in the following section.

Continuation Lines

If a command line exceeds 122 characters, the line must be separated into multiple lines, known as continuation lines. Continuation lines are formed by entering an ampersand character (&) and a carriage return prior to the 122nd character.

When entering the ampersand and carriage return, the emulator software responds by displaying a double asterisk prompt (**). The double prompt indicates the EMV-51 is accepting a command line that exceeds the 122 character maximum length. The command line is not executed until termination of a continuation line with just a carriage return. At this point, the emulator returns to displaying a single asterisk prompt character.

Entry Editing and Display Control

The emulator software recognizes control characters that permit editing of command line entries and control the display generated by command execution. These control characters are described in the following sections.

Command Line Editing

The control characters listed below are used to correct errors in command line entry. The control characters consist of one or two keyboard characters. If a character is preceded by the word CTRL, press the specified character while holding down the CTRL key. Otherwise, the indicated key is entered.



Aborts any operation that is executing or aborts any command being entered.



Signifies the end of the command line.



Pressing the RUBOUT key deletes the last character typed. One character is deleted each time the key is pressed.



Gets the next keyboard entry and places the character in the command line buffer without interpreting it.



Echo command line being entered.



Deletes the current line only, not any previous lines. If several continuation lines have been entered, the CTRL X deletes only the current portion.



Deletes an entire command line, including all continuation lines.

CONTROLLING THE DISPLAY

The emulator software displays information on the screen by displaying a line, scrolling the displayed line upward one line, and displaying another line. The speed of the display can be controlled with the characters listed below.



Stops the output of new information to the screen.



Continues the output of new information to the screen.



Slows down or speeds up screen display. Pressing FUNCT S once slows the display down, while pressing it again speeds up the display.





Introduction

The Utility commands perform the emulator control functions for accessing the iPDS resources and display information about the emulator. This chapter describes what the Utility commands are, where they are used, and how they are used through screen displays.

Help Information

The HELP command displays information about the emulator commands and keywords. It operates in two modes: general help mode where a list of available commands and keywords is provided; specific mode where format and requirements about specific commands and keywords are displayed.

Help Information Screen Displays

The following screen displays show the two forms of the HELP command. The first form, without parameters, displays a list of all available commands and keywords. This form is useful for a quick reference of further available help.

```

*HELP
Help is available for the following commands and definitions. Type
HELP followed by the full command name or (definition).

utility      - BASE, DEFINE, DISABLE, ENABLE, EVALUATE, EXIT, LIST,
              LOAD, REMOVE, RESET, SAVE, SUFFIX, SYMBOL
display      - ASM, CBYTE, CDUMP, DASM, DBYTE, DDUMP, INTERRUPT, MEM
              ORY, PBYTE, RBIT, RBYTE, REGISTER
evaluation   - BREAK, DTRACE, GO, PREVIOUS, STEP
advanced     - COUNT, FUNCTION, DIR, IF, INCLUDE, PUT, REPEAT, WRITE
(definition) - <ADDRESS>, <CHANGE>, <CPU#KEYWORD>, <DISPLAY>,
              <EXPR><IDENTIFIER>, <INSTRUCTION>, <MACRO#DISPLAY>,
              <MACRO#INVOCATION>
              <NUMERIC#CONSTANT>, <PARTITION>, <STRING>,
              <STRING#CONSTANT> <SYMBOL#REF>, <USER#SYMBOLS>

Keyboard - RUBOUT = delete char  ESC = cancel          CTRL R = echo line
Controls - CTRL X = delete line  CTRL P = literal char  CTRL Z = delete line

Function Keys are invoked by simultaneously pressing FUNCTION and a
number.
  1 = MEMORY      3 = DTRACE      [5 to 9 and 0 are user defined]
  2 = BREAK       4 = REGISTER
*

```

Key-in Sequence

HELP

Comments

When the user enters the word HELP, the emulator software responds by displaying a list of all commands and keywords used by the software.

The second example demonstrates HELP LOAD which responds with the information about the LOAD command. Typing in HELP followed by one or more commands or keywords produces information about each command or keyword.

```

*HELP LOAD
LOAD - Command to load object code and/or its symbols. The
symbols are added to the end of the EMV-51 symbol table.
  (1) To load the object code and its symbols, type:
      LOAD :Fn:<filename>
      where n is the drive number.
      (EX: LOAD :F2:MYPROG.OBJ)
  (2) To load only the object code, type:
      LOAD :Fn:<ISIS filename>NOSYMBOL
      (EX: LOAD :F2:MYPROG.OBJNOSYMBOL)
  (3) To load only the symbols, type:
      LOAD :Fn:<ISIS filename>NOCODE
      (EX: LOAD :F2:MYPROG.OBJNOCODE)
Note: The LOAD command does not check for duplicate symbols. Thus, two
loads of the same program will duplicate all symbols twice. Also,
duplicate symbols can be accessed by entering the module name
first.
Example: CBYTE <.module.symbol> <cr>
*
```

Key-in Sequence	Comments
HELP LOAD <input type="button" value="RETURN"/>	The user can obtain additional information about a command by entering the word HELP followed by the desired command or keyword. This example requests help with the LOAD command.
	The emulator software displays this information when the user requests help with the LOAD command.

Beginning an Emulation Session

The next sections explain how to begin an emulation session.

The Load Command

To begin an emulation session, load a program into the EMV-51 emulator. The LOAD command loads the specified 8051 program from an external device (either diskette or magnetic bubble memory) into the EMV-51 program memory. Programs can only be loaded into the 4K bytes of EMV-51 program memory. The command can load hex or binary files. In the following example, a program named EMVDEM.OBJ is loaded:

```
LOAD :F1:EMVDEM.OBJ
```

The :F1: indicates the program is contained on disk drive number 1, but it may be any disk drive or magnetic bubble memory currently installed on the iPDS.

During program assembly, the ASM-51 Assembler builds a symbol table containing the addresses of all user assigned labels within the program. The symbol table may be included in the object file produced by the assembler. The object file consists of the actual machine language equivalents of the mnemonic instructions.

When the symbol table is included in the object file, the LOAD command optionally loads the program, the symbol table created with the assembler, or both the program and the symbol table. If the format shown above is used, the emulator software loads the program and its symbol table into the emulator's memory. The symbol table, preceded by the program name, is appended to the end of the emulator's internal symbol table.

When the keyword NOSYMBOL is added, the command loads the program without the symbol table. The format is shown below:

```
LOAD :F1:EMVDEM.OBJ NOSYMBOL
```

By replacing the keyword NOSYMBOL with NOCODE, the command loads the symbol table without the program:

```
LOAD :F1:EMVDEM.OBJ NOCODE
```

Loading Multiple Programs

EMV-51 software permits the user to load multiple programs into its program memory. The programs must be addressed so they do not overlap when loaded. The addressing is accomplished with the ORG statement during program assembly with ASM-51 assembler.

When loading multiple programs and symbol tables into EMV-51, each symbol table is appended to the end of the emulator software's internal symbol table with the respective program name preceding the first symbol.

Copy an Emulation Session

In normal operation, the emulator software expects user input from the keyboard and displays results on the screen. Instructing the emulator software to put a copy of keyboard input and screen output in a separate file, is described in the following sections.

Setting Up a List File

The following paragraphs describe how to set up a list file to receive a copy of information sent from the keyboard or to the screen.

The LIST Command

The LIST command instructs the emulator software to open a file to receive a copy of all information sent to the display screen or coming from the keyboard. This copy can be analyzed thoroughly after the emulation session to isolate errors. To open a list file named EMVDEM.SMP on drive 1, the command is:

```
LIST :F1:EMVDEM.SMP
```

After the command is entered, all subsequent keyboard and display information is copied to the file EMVDEM.SMP. Any information previously in the file specified with the LIST command is destroyed.

Closing a LIST File

When the user is ready to close the LIST file, enter the command:

LIST :CO:

The command instructs the emulator to close the list file and display information on the screen only. The list file now contains a complete copy of the emulation session, providing a means of examining the session at a later time.

If the user terminates an emulation session prior to closing the LIST file, the emulator software closes the file before returning control to ISIS-PDS.

Load and LIST Screen Display

The following screen display demonstrates loading a demonstration program, opening a LIST file, performing several commands, then closing the LIST file.

```
*LOAD EMVDEM.OBJ
*LIST :FO:SAVE.TXT
*HELP SAVE
SAVE - Command to save code memory and/or symbols in an 8051
object file.
  (1) To save the last partition loaded and the symbol table, type:
      SAVE <filename> (EX: SAVE :F1:SAVE.IT)
      where <filename> is defined as the name of an ISIS file. For
      example: :F1:MYPROG.OO1.
  (2) To save only the last partition loaded type:
      SAVE <filename> NOSYMBOL (EX: SAVE :F1:F00 NOSYMBOL)
  (3) To save only the symbol table, type:
      SAVE <filename> NOCODE (EX: SAVE :F2:SYMBOL NOCODE)
  (4) To save a specified part of code memory and the symbol table,
      type:
      SAVE <filename> <partition> (EX: SAVE :F1:T00 5 TO 45H)
  (5) To save a specified part of code memory with no symbols, type:
      SAVE <filename> <partition> NOSYMBOL
      (EX: SAVE :F3:CODE.SAV 100H TO 200H NOSYMBOL)
*SAVE :FO:EMV.PAR NOCODE
*LIST :CO:
*
```

Key-in Sequence

```
LOAD EMVDEM.OBJ 
LIST :FO:SAVE.TXT 
HELP SAVE 
SAVE :FO:EMV.PAR NOCODE

LIST :CO: 
```

Comments

The LOAD command gets a program from a disk file and loads it into EMV-51 program memory. In this example, the program labeled EMVDEM.OBJ is loaded into memory. The emulator software responds with the asterisk prompt when the LOAD command is completed.

This display illustrates how a LIST file is opened and closed. The first command opens the file, while the last command closes the file. The material between the two commands, including the command LIST :CO:, is displayed on the screen and simultaneously copied into the LIST file.

Symbolic Reference Commands

Symbolic reference is a powerful feature of the EMV-51 emulator. The EMV-51 software permits the user to reference memory address values with the symbols from the program symbol table. This table was created when the program was originally assembled or compiled. The user can also add new names to the symbol table to reference additional addresses or values.

The user can assign addresses to meaningful symbolic names, then refer to the symbolic names rather than trying to keep track of the addresses of key locations within the program. Numeric values can also be assigned symbolic names, which can then be manipulated in expressions as variables.

The following topics are explained in this section:

- Assign values to symbol names
- Display user assigned symbol names
- Remove symbol names from the symbol table
- Enable symbolic name display
- Disable symbolic name display

Assigning Symbol Names

The DEFINE command is used to assign expression values to symbol names as described in the following section.

The DEFINE Command

The DEFINE command assigns an expression value to a symbol name. The expression can be a numeric value or an address. User defined symbol names must have a period as the first character. Once assigned, the user can enter the symbol name rather than the expression. Several command examples are:

DEFINE .BAKER=2000T	Assigns the numeric decimal value 2000 to the symbolic name .BAKER.
DEFINE .ABLE=23T*100T/110T	Assigns the result of the expression 23T*100T/110T to the symbolic name .ABLE

Manipulating Symbols

The user can manipulate the values assigned to symbolic names by entering the symbol name followed by an expression or value. Several examples are:

.ABLE=.ABLE+1	Increments the current contents of the symbol name .ABLE.
.BAKER=(.ABLE/256T)+30H	Assigns the value of the expression (.ABLE/256T)+30H to the symbol name .BAKER.

Displaying Symbol Names

The following sections describe how to display individual, multiply defined, or all symbol names.

Displaying Individual Names

The user can display a symbol name and its corresponding value by entering the name preceded by a period.

.ABLE Display the value assigned to the symbol name .ABLE.
 .BAKER Display the value assigned to the symbol name .BAKER.

Displaying Multiply Defined Names

When several program modules are loaded into EMV-51 program memory, a symbol name might be defined more than once. When this happens, a specific definition can be displayed by preceding the name with the module name that contains the desired definition:

.EMVDEM.ABLE Displays the value assigned to the symbol name .ABLE used in the program EMVDEM.
 .EMVTWO.ABLE Displays the value assigned to the symbol name .ABLE used in the program EMVTWO.

The SYMBOLS Command

The SYMBOLS command displays all user symbol names currently residing in memory and their corresponding address values. The command is:

SYMBOLS

Removing Symbol Names

The method for removing symbol names is described in the next section.

The REMOVE Command

The REMOVE command deletes one or more symbol names that the user assigned from the emulator software's symbol table. Several examples are:

REMOVE .BAKER Removes the user-assigned symbolic name .BAKER from the symbol table.
 REMOVE .APPLE, .BAKER Removes the user-assigned symbolic names .APPLE and .BAKER from the symbol table.
 REMOVE SYMBOLS Removes all user assigned symbol names from the symbol table.

Symbolic Reference Screen Displays

```
*DEFINE.BAKER=63
*.BAKER
.BAKER=0063H
*
```

Key-in Sequence

Comments

DEFINE.BAKER=63

The user can define new symbolic names with the DEFINE command. In this example, the symbolic name .BAKER is defined and the value 63 is assigned to .BAKER.

.BAKER

Symbolic names have a value assigned to them. The user can display the name and the assigned value by entering the symbolic name. In response to the command .BAKER, the emulator software displays the value assigned to the name (in this case, the value is 63H).

```
*SYMBOLS
.EMVDEM=0200H
.BUFFER=006FH
.CLRMSD=0069H
.DEMO=0040H
.DONE=0068H
.LOOP=0048H
.RDX=0000H
.EMVDEM=0200H
.BUFFER=006FH
.CLRMSD=0069H
.DEMO=0040H
.DONE=0068H
.LOOP=0048H
.RDX=0000H
.BAKER=0063H
*REMOVE.BAKER
*
```

Key-in Sequence

Comments

SYMBOLS

The emulator software displays all user-assigned symbolic names and their assigned values in response to the SYMBOLS command.

REMOVE.BAKER

The REMOVE command deletes one or more user symbolic names. In this example, the name .BAKER is deleted.

Enabling/Disabling Symbol Display

The emulator software commands display results as either numeric data or symbolic names. The commands look at an internal software switch before displaying symbolic names. This switch is automatically controlled with the ENABLE and DISABLE commands described below. Its default condition is enabled to display symbolic names.

The DISABLE Command

The DISABLE command resets the internal switch, forcing the software to use numeric addresses. The command is:

DISABLE SYMBOLIC

The ENABLE Command

The ENABLE command sets the internal switch permitting the display of symbolic names rather than numeric addresses. When the software finds a symbol in the symbol table that matches the numeric data, it displays the symbol rather than the data. The command is:

ENABLE SYMBOLIC

Evaluating Symbol Values

The following sections explain how to evaluate symbol names and presents a screen display demonstrating the EVALUATE command.

The EVALUATE Command

The EVALUATE command displays the value of a previously defined symbol name or calculates the value of an expression involving symbols and mathematical operators. In either case, the value is displayed in hexadecimal, decimal, octal, ASCII, and as an offset from the nearest symbol name that is less than the value. Several command entry examples are:

EVALUATE .ABLE Display the numeric value or address currently assigned to .ABLE.

EVALUATE 100*23+4 Calculate the value of the expression and display the result.

Evaluate Screen Display

```
*DEFINE .ABLE=23
*EVA .ABLE*21/2
1001000001Y 11010 577T 241H 'A' .EMVDEM+0041H
*
```

Key-in Sequence

Comments

DEFINE .ABLE=23

EVA .ABLE*21/2

The EVALUATE command evaluates an expression or symbol name entered with the command. The emulator software displays the results of the command in Binary, Octal, Hexadecimal, and ASCII. In addition, the result is displayed as an offset from the nearest symbol name that is less than the value.

Number Display Format

The EMV-51 software accepts numeric values entered from the keyboard and displays numeric values on the screen in one of four number bases: hexadecimal (H), decimal (T), binary (Y), or octal (Q). The user specifies the desired number base for input and display with two commands, SUFFIX and BASE.

Any number entered without an explicit suffix is interpreted using the current suffix. An explicit suffix overrides the suffix set by the SUFFIX command.

The SUFFIX and BASE Commands

The number base for keyboard entry is specified with the SUFFIX command and the number base for display output with the BASE command.

The SUFFIX command displays the current keyboard entry number base or changes the current number base to the specified one, as shown below:

SUFFIX Displays the current keyboard entry number base.
 SUFFIX = H Changes the current keyboard entry number base to Hexadecimal.

The BASE command displays the current display number base or changes the current number base to the specified one, as shown below:

BASE Displays the current display number base.
 BASE = Q Changes the current display number base to Octal.

The keyboard entry and display number bases may differ, as the above examples show. Generally, the display is more readable when both bases are identical.

SUFFIX and BASE Screen Displays



Key-in Sequence

Comments

SUFFIX The emulator software displays the current radix for input numbers when the user enters the SUFFIX command.

SUFFIX = T The user can change the input number radix as shown in this screen display.

Saving the User's Program

The command for saving a user program is described in the following paragraph.

The SAVE Command

The SAVE command writes the user's program to a disk file. The user specifies whether the program, its symbol table, or both the program and symbol table should be written to the file.

Examples:

SAVE :F1:EMVDEM.SAV	Saves the program object code and symbol table in the file labeled EMVDEM.SAV.
SAVE :F1:EMVDEM.SAV NOSYMBOL	Saves the program object code in the file labeled EMVDEM.SAV, but does not save the symbol table.
SAVE :F1:EMVDEM.SAV NOCODE	Saves the program symbol table only. The object code is not saved.
SAVE :F1:EMVDEM.SAV 10 TO 90	Saves a range of program object code from address 10H through 90H and symbol table in the file labeled EMVDEM.SAV.

Initializing EMV-51 Hardware

The next section describes how to initialize the EMV-51 hardware.

The RESET Command

The RESET command initializes the breakpoints and tracepoints to the off state, puts the 8051 microcontroller in its reset state, and issues prompts for additional commands.

Ending an Emulation Session

The EXIT command, described in the next section, is used to end an EMV-51 emulation session.

The EXIT Command

The EXIT command terminates the EMV-51 emulation session and returns control to the ISIS-PDS operating system, where the ISIS-PDS prompt character is displayed.

Using Utility Commands

The following screen displays demonstrate several of the Utility commands by exercising various portions of the demonstration program. In addition, the DASM command (described in Chapter 4) is shown to demonstrate the ENABLE/DISABLE commands. The demonstration program example is continued in Chapters 4, 5, and 6.

```

*ENABLE SYMBOLIC
*DASM 40H TO 45H
.DEMO          0040H=M0V      R1,#.BUFFER      DASM
               | 0042H=M0V      R2,#03H          DASM
               | 0044H=CLR      .CY                DASM

*DISABLE SYMBOLIC
*DASM 40H TO 45H
               | 0040H=M0V      R1,#6FH          DASM
               | 0042H=M0V      R2,#03H          DASM
               | 0044H=CLR      D7H                    DASM

*SAV :F0:EMVDEM.V10
*EXIT

AD>
    
```

Key-in Sequence

Comments

ENABLE SYMBOLIC

When symbolic display is enabled, the emulator software displays addresses and data values as symbolic names if the names exist in the user symbol table.

DASM 40H TO 45H

Display disassembled instructions from 40H to 45H.

DISABLE SYMBOLIC

Enter the DISABLE SYMBOLIC command, the emulator displays all addresses and values as numeric quantities.

DASM 40H TO 45H

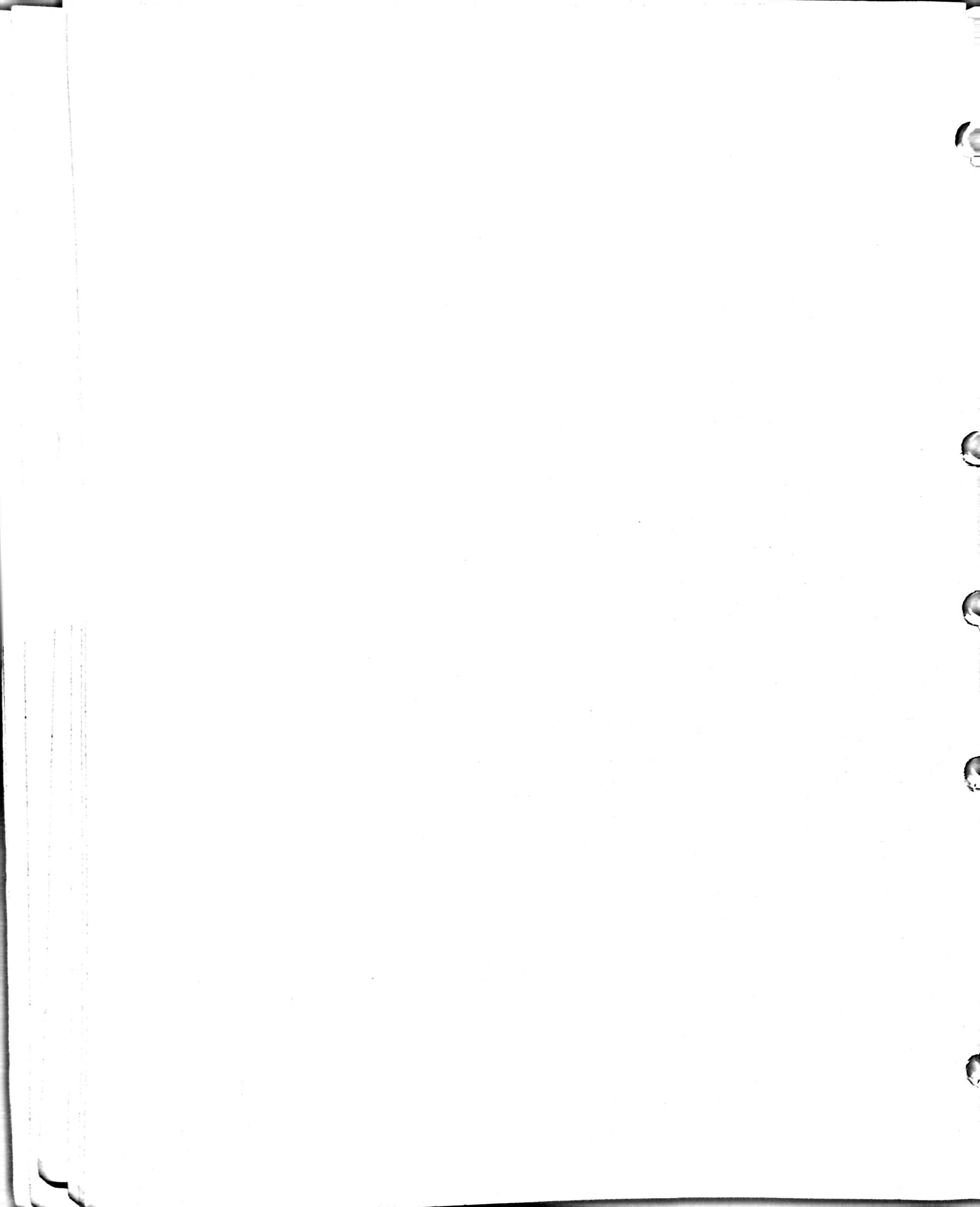
Display disassembled instructions from 40H to 45H.

SAV :F0:EMVDEM.V10

Save the prototype program in a disk file prior to terminating the session. In this case the program is saved in a file labeled EMVDEM.V10.

EXIT

The EXIT command terminates the emulator software and returns control to the ISIS-PDS operating system.





Introduction

The DISPLAY/MODIFY commands include register and memory handling commands. This chapter describes what the commands are, where they are used, and provides examples of how they are used.

Using Register Commands

The MCS-51 family of microcontrollers contain temporary data storage locations, called registers. The user accesses a register by entering the keyword associated with that register. Table 4-1 lists the registers and the associated keyword names.

Table 4-1 Register Keyword Names

KEYWORD	FUNCTION
PC	Program Counter
SP	Stack Pointer
DPTR	Data Pointer
ACC	Accumulator
B	Multiplication Register
TMO	Counter/Timer 0
TM1	Counter/Timer 1
PSW	Program Status Word
RBS	Register Bank Select
R0	General Purpose Register 0
R1	General Purpose Register 1
R2	General Purpose Register 2
R3	General Purpose Register 3
R4	General Purpose Register 4
R5	General Purpose Register 5
R6	General Purpose Register 6
R7	General Purpose Register 7

Displaying Register Contents

The emulator software displays the content of a register when the keyword associated with the particular register is entered. Several command examples are:

PC Displays the current address value in the program counter.
SP Displays the current 8-bit address value in the stack pointer.
DPTR Displays the current address value in the data pointer.

Modifying Register Contents

The content of a register can be modified by entering the register keyword followed by an expression. These examples illustrate several types of expressions:

ACC=88H Changes the contents of the accumulator to 88 hex.
DPTR=4444T Changes the contents of data pointer to 4444 base 10.
R0=.BCC+22H*2 Changes the contents of general purpose register R0 to the value of the expression, .BCC+22H*2.

Displaying Register Names and Contents

The REGISTER command, special function key 4, and a REGISTER command display screen are given in the following sections.

The REGISTER Command

The REGISTER command displays all register keywords and contents on the screen. The contents of the registers are displayed in the current number base unless the number base is binary. If the current number base is binary, the contents of the registers are displayed in hexadecimal.

The display remains on the screen until the user enters a non-register keyword (i.e., HELP). Thus, the user can change several registers and the display is updated to reflect the new values. If the BASE is changed, all register contents change to reflect the new base. All register contents are entered according to the current number suffix.

Special Function Key 4

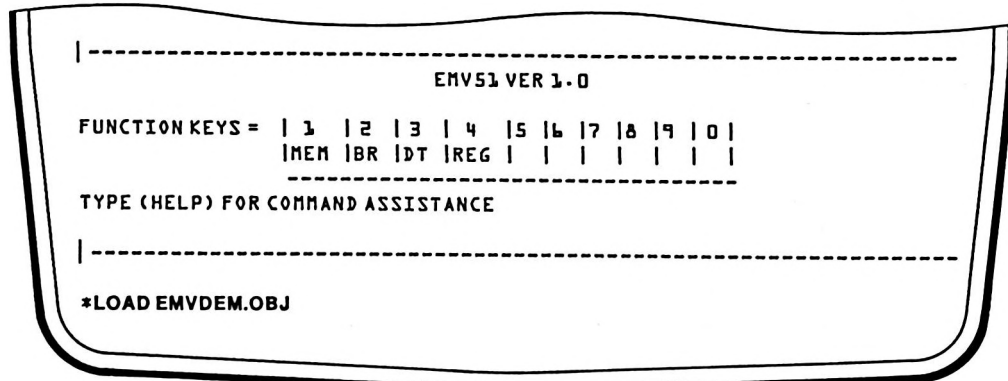
Rather than enter the command REGISTER, the user can enter the function key shown below to obtain the same display. The numeral 4 key is pressed while holding down on the FUNCT key.



Displays some register keywords and their contents on the screen.

REGISTER Command Display Screen

These screens demonstrate the REGISTER command and resulting display.



Key-in Sequence

Comments

EMV51

When the user enters EMV51, the ISIS-PDS operating system loads and executes the EMV-51 software. The software displays the sign-on message and asterisk prompt shown in this display.

LOAD EMVDEM.OBJ



The LOAD command gets a program from a disk file and loads it into EMV-51 program memory. In this example, the program labeled EMVDEM.OBJ is loaded into memory. The emulator software responds with the asterisk prompt when the LOAD command is completed.

```

*REG
-----
| PC  =000H   TM0=0000H   RBS  =0   R0= 86H   R4= A7H
| SP  =07H   TM1=000H    BASE  =H   R1= 66H   R5= EAH
| DPTR =0000H SUFFIX =H   R2= 58H   R6= 66H
| ACC =00H   PSW= 00000000Y   R3= 3AH   R7= 86H
-----

*ACC=88
-----
    
```

Key-in Sequence

Comments

REG

The emulator software displays a table containing the 8051 registers when the REGISTER command is entered. The command can be abbreviated to REG or R. The asterisk prompt above the short line indicates the software is in the register display mode.

ACC=88

Change the contents of an 8051 register as shown in this screen. The value 88 is interpreted as hexadecimal since the suffix is H.

```

*REG
-----
| PC  =000H   TM0=0000H   RBS  =0   R0= 86H   R4= A7H
| SP  =07H   TM1=000H    BASE  =H   R1= 66H   R5= EAH
| DPTR =0000H SUFFIX =H   R2= 58H   R6= 66H
| ACC =88H   PSW= 00000000Y   R3= 3AH   R7= 86H
-----

*
    
```

Comments

This display shows the accumulator updated to its new value and the EMV-51 software still in the register display mode.

Displaying INTERRUPT Register Contents

The INTERRUPT command is explained in the next section.

The INTERRUPT Command

The INTERRUPT command displays the Interrupt Enable (IE), Interrupt Priority (IP) registers, and the status of Interrupts in Progress for priority 0 (IIP0) and priority 1 (IIP1). The command and resulting display are shown in the following screen displays.

*INT	EA	SINT	TIMER1	EXTI1	TIMERO	EXTIO
IIP0		0	0	0	0	0
IIP1		0	0	0	0	0
IE	0	0	0	0	0	0
IP		0	0	0	0	0

Key-in Sequence

Comments

INT 

The INTERRUPT command displays a table showing the status of all 8051 interrupts.

The column headers stand for:

- EA Enable all Interrupts (IE register only)
- SINT Serial Port Interrupt
- TIMER1 Timer 1 Interrupt
- EXTI1 External Interrupt 1
- TIMERO Timer 0 Interrupt
- EXTIO External Interrupt 0

Memory Commands

The MCS-51 microcontroller family contains separate program, data, and special function register memory areas. The user accesses a memory area by entering the keyword associated with it. The keyword also serves as the command for modifying and displaying that memory area. Table 4-2 lists the memory areas and their keyword names, while Chapter 7 contains a breakdown of the contents of each memory area. The *Microcontrollers User's Manual*, manual order number 210359, contains additional information on the architecture of the microcontrollers.

Table 4-2 Memory Areas and Keyword Names

Keyword	Memory Area
CBYTE DBYTE PBYTE RBYTE RBIT	Program Memory (0-4K internal/4K-64K external) Internal 128 byte Data Memory External 64K byte Data Memory Internal Special Register Memory (ports, counter) Internal Bit-Addressable Memory
CDUMP DDUMP	Formatted Display of CBYTE Memory Formatted Display of External Data Memory

Displaying Memory Contents

The contents of a memory area are displayed when the user enters the keyword name and an address within that area. The address consists of single or multiple locations. Several examples of memory display commands are:

- CBYTE 03FFH Display the content of a single memory location.
- DBYTE 0 TO 20H Display the contents of a range of internal data memory locations from 0 through 20H.
- PBYTE 37H LEN 9 Display the contents of external data memory locations from 37H to 3FH.

Optionally, the contents of program or external data memory can be displayed in both ASCII and Hex numeric format. Two examples of these memory display commands are:

- CDUMP 100H TO FFFH Displays the contents of a range of program memory locations as ASCII characters and hexadecimal numbers.
- DDUMP 0 TO 100H Displays the contents of a range of external data memory locations as ASCII characters and hexadecimal numbers.

NOTE

Code memory from 0 to 0FFFH may be internal or external to EMV-51 if the emulator is plugged into a user prototype. If so, the EA/ input to the emulator determines where the memory is located: EA/ = 1 means the memory from 0H to 0FFFH is internal to EMV-51, while EA/ = 0 means the memory is external to EMV-51. The memory data displayed by CBYTE or CDUMP is determined by EA/. The user can strap EA/ high or let the user prototype determine the the state of EA/.

Modifying Memory Contents

The contents of a memory location can be changed by entering the memory keyword, address, and an expression. Thus, the command

CBYTE 03FFH=33H

changes the current contents of program memory location 03FFH to 33H in EMV-51 program memory. Several examples of memory modification commands are:

- PBYTE 03FFH=55H Changes the content of external data memory location 03FFH to 55H.
- DBYTE 0=2*3+.ABLE Changes the content of data memory location 0 to the result of the expression 2*3+.ABLE.
- RBYTE 80H TO 83H=22H Changes the contents of register memory locations 80 through 83 to the value 22H.

NOTE

CBYTE only modifies the EMV-51 program memory, which is located from 0 to 0FFFH. If EMV-51 is plugged into a user prototype with external memory at these addresses (EA/ asserted), the memory may be examined, but not modified.

Displaying Memory Commands

The methods for displaying memory commands and demonstration screen displays are explained in the next sections.

The MEMORY Command

The MEMORY command displays a table containing memory keywords and their format requirements.

Special Function Key 1

Rather than enter the command MEMORY, the user can enter the function key shown below to obtain the same display. The number 1 is pressed while holding down the FUNCT key.



1

Displays all memory keywords and their format requirements on the screen.

*M			
MEMORY COMMANDS			
CBYTE (code memory)		T0 Location	= value
D BYTE (data memory)			
R BYTE (registers)	Location	LENGTH n	
R BIT (bit flags)			
P BYTE (ext. data)			
CDUMP (code dump)	Location T0 Location		
DDUMP (data dump)			
*			

Key-in Sequence

M



Comments

When the MEMORY command (abbreviated to M) is entered, the emulator software displays a table containing the memory reference commands.

```

*M
-----
|          MEMORY COMMANDS          |
|-----|-----|-----|-----|
| CBYTE (code memory ) |      | T0 Location | = value |
| DBYTE (data memory ) |      |      |      |
| RBYTE (registers ) | Location | LENGTH n |      |
| RBIT (bit flags ) |      |      |      |
| PBYTE (ext. data ) |      |      |      |
|-----|-----|-----|-----|
| CDUMP (code dump) | Location T0 Location |
| DDUMP (data dump) |      |      |      |
|-----|-----|-----|-----|
*CBYTE 40H LEN 69H
.DEMO
0040H=79H 6FH 7AH 03H C2H D7H C0H 00H
.LOOP
0048H=E6H 08H 08H 08H 36H D4H C0H E0H 54H 0FH 44H 30H F7H 09H D0H E0H
0058H=C4H 54H 0FH 44H 30H F7H 09H 18H 18H DAH E5H 50H 04H 77H 31H 80H
0068H=02H
.CLRMSD
0069H=77H 30H
*

```

Key in Sequence**Comments****CBYTE 40H LEN 69H**

The CBYTE command displays the contents of a specified section of program memory. The contents are displayed according to the current Base. In this example, the current Base is hexadecimal. If an address is assigned to a symbol name and the name is in the symbol table, the EMV-51 software displays the symbol name followed by the address.

Assembly/Disassembly Commands

The EMV-51 software contains two commands, ASM and DASM, that assemble program instructions in program memory and disassemble program object code into mnemonic equivalents. These commands are covered in the following sections.

Assembling User's Program

The following section explains the command that displays and changes the current contents of the assembly location counter, and assembles one instruction into program memory.

The ASM Command

The ASM command performs three functions:

- Displays the current contents of the assembly location counter.
- Changes the current contents of the assembly location counter.
- Assembles one instruction into program memory.

The emulator software maintains an assembly location counter that indicates the next available location in program memory where new instructions can be placed. The current contents of the assembly location counter can be displayed by entering:

ASM Displays the current content of the assembly location counter.

New instructions are also assembled into program memory with the **ASM** command. The new instructions are stored in memory at the current location pointed to by the assembly location counter. The assembly location counter is incremented to the next available location and the new location is displayed. Several examples are:

ASM ORG 03FFH Changes the contents of the assembly location counter to 03FFH.
ASM MOV A,R0 Assembles the instruction **MOV A,R0** into program memory at the location contained in the assembly location counter.

After a new instruction is assembled into program memory, the **ASM** command updates the assembly location counter and displays the new value.

NOTE

The **ASM** command only operates between memory addresses 0 and 0FFFH, inclusive, in EMV-51 internal program memory.

Disassembling User Programs

To produce and display mnemonic equivalents of instructions, use the command described in the next section.

The DASM Command

The **DASM** command disassembles user programs, producing the mnemonic equivalents of the machine language instructions. The command disassembles a single instruction or a range of instructions. Several examples are:

DASM 100H TO 150H Disassembles contents of memory from address 100H through 150H, producing the mnemonic instruction equivalents.

DASM .ABLE TO .BAKER Disassembles the contents of memory from symbolic address **.ABLE** through symbolic address **.BAKER**, producing the mnemonic instruction equivalents.

Using Display/Modify Commands

In Chapter 1, a short example program was described. This section uses the Display/Modify commands to exercise portions of that program.

```
*ASM
0000H
*ASM ORG .BUFFER+10
007FH
*
```

Key-in Sequence

Comments

ASM

By entering the command ASM, the emulator software responds with the current value of the Assembly Location Counter (ALC). The ALC points to the place in Program memory where the next instruction is assembled. The following screen display shows the results of the ASM command.

ASM ORG .BUFFER+10 When the ALC is set to a new value (via the ORG command), the emulator software displays the new location immediately following the command.

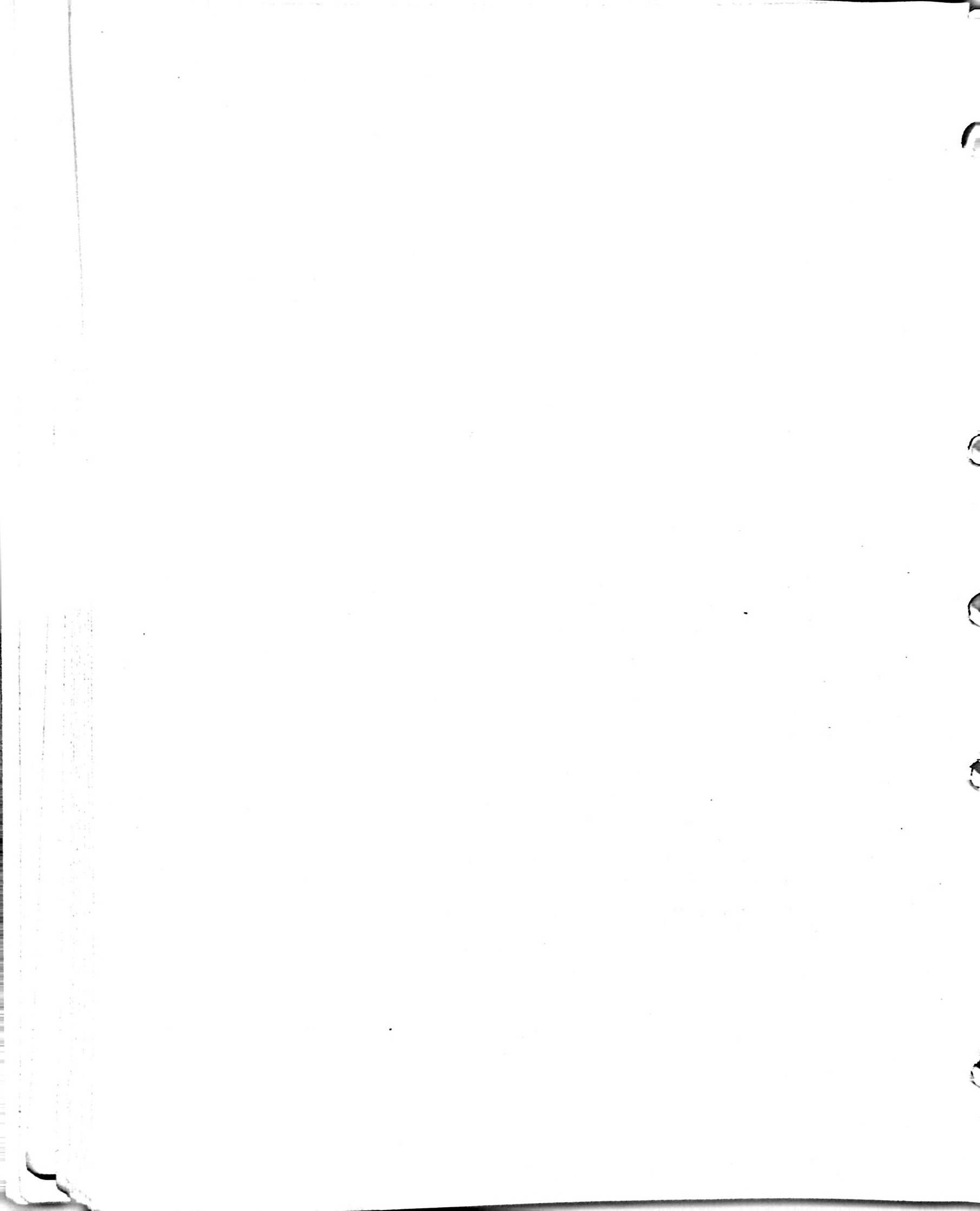
```
*DASM 40H TO 50H
.DEMO          0040H=MOV      R1,#.BUFFER      DASH
                | 0042H=MOV      R2,#03H          DASH
                | 0044H=CLR      .CY              DASH
                | 0046H=PUSH     .R0X             DASH
.LOOP          0048H=MOV      A@R0              DASH
                | 0049H=INC      R0              DASH
                | 004AH=INC      R0              DASH
                | 004BH=INC      R0              DASH
                | 004CH=ADD     A,@0            DASH
                | 004DH=DA      A              DASH
                | 004EH=PUSH     .ACC           DASH
                | 0050H=ANL     A,@0FH         DASH
*
```

Key-in Sequence

Comments

DASM 40H TO 50H

The DASM command disassembles the contents of memory and produces the mnemonic equivalents of the instructions. In this example, memory from 40H to 50H is disassembled.





Introduction

This chapter describes the breakpoint, software trace, and emulation command groups. The following sections describe the commands that comprise each group, discuss how the commands are used, and give command entry examples. The following topics are discussed:

- Using Breakpoint Commands
- Using Software Trace Commands
- Using Program Emulation Commands
- Using Emulation Controls

Setting Breakpoints

A breakpoint functions as a switch that stops program execution after a specified event occurs. By using breakpoints, interactive control of program operation is possible.

During program execution, if a breakpoint is encountered, the emulator stops program execution. Then, the user can selectively display and modify register and memory contents.

The emulator software supports address, branch, range, and register value breakpoints. These breakpoint types are described in the following paragraphs.

Address Breakpoints

The address breakpoint (BRn, where n=0 to 3) stops program execution if control passes to a specified address. The address must be the first byte of an instruction; otherwise, the breakpoint is ignored and program execution continues. Up to four address breakpoints can be specified. Several command entry examples are:

BR0=100H	Sets address breakpoint 0 for address 100H.
BR1=.ABLE	Sets address breakpoint 1 for address specified by the symbol name .ABLE.
BR2=.ABLE+5	Sets address breakpoint 2 for the address located 5 bytes after the address specified by the symbol name .ABLE.
BR3=100H+.BAKER*2	Sets address breakpoint 3 for the address specified by the expression 100H+.BAKER*2.

Branch Breakpoints

A branch breakpoint (BRB) stops program execution when a jump instruction is executed, i.e., a Jump, Call, Return, or Return from Interrupt instruction is executed. The branch breakpoint is only used in the GO (or real time) emulation mode. Branch breakpoint control can be enabled or disabled. Two command entry examples are:

BRB=ON	Enables the branch breakpoint control.
BRB=OFF	Disables branch breakpoint control.

Range Breakpoints

A range breakpoint (BRR) stops program execution when program control passes to an address within a specified range of addresses. The beginning address must be the first byte of an instruction; otherwise, the breakpoints at these addresses are ignored and program execution continues. Several command entry examples are:

BRR=100H TO 120H Sets a range breakpoint for the address range 100H to 120H.
 BRR=.ABLE TO .ABLE+100H Sets a range breakpoint for the address range specified by .ABLE as the lower limit and .ABLE+100H as the upper limit.

Register Value Breakpoints

The register value breakpoint (BV) stops program execution if a register contains a specified value. The register value breakpoint is only used in the STEP (or non-real time) emulation mode (the emulation modes, STEP and GO, are described later in this chapter under Using Program Emulation Commands). The user can specify one of the general purpose registers (R0 through R7) or the accumulator (ACC). The specific bank of general purpose registers depends on the contents of RBS, which specifies register banks 0, 1, 2, or 3. Several command entry examples are:

BV=R0 44H Stops emulation when general purpose register R0 contains the value 44H.
 BV=ACC .ALPHA Stops emulation when the accumulator contains the least 8 bits of the value represented by the symbol .ALPHA.

Clearing Breakpoints

The BC command, shown below, disables all breakpoints.

BC Disables all set breakpoints.

Breakpoint Restrictions

The emulator software permits the user to set four address breakpoints at any given time. The range breakpoint (BRR) counts as three active address breakpoints. Thus, the user can specify one range break and one address break, or up to four address breaks.

When the user specifies two or more address breakpoints and a range breakpoint, the emulator software displays the following error message:

TOO MANY BREAKS

The software does not recognize a range breakpoint when two or more address breakpoints are enabled. To set a range breakpoint in this case, clear one or more of the address breakpoints and then set the range breakpoint.

NOTE

If a range breakpoint and one address breakpoint are set and the user adds another address breakpoint, the range breakpoint is destroyed.

Using the BREAK Command

The preceding breakpoint commands can be entered individually or from within a breakpoint table, as described in the following section.

The BREAK command displays a table containing all breakpoints, their entry requirements, and the current state of each. When the user enters a breakpoint command, the displayed table is updated to reflect the new entry. The breakpoint display remains on the screen until the user enters a non-breakpoint command.

Special Function Key 2

Rather than enter the command BREAK, the user can enter the function key shown below to obtain the same display. The numeral 2 key is pressed while holding down the FUNCT key.



2

Displays all breakpoint keywords and their contents on the screen.

Breakpoint Screen Displays

The following screen displays demonstrate the breakpoint commands and resulting displays.

BREAKPOINT SETTINGS				TYPE
BR0 = OFF	BR1 = OFF	BR2 = OFF	BR3 = OFF	location
BRR = OFF	(go mode only)		BC disables all	range
BRB = OFF	(step mode only)		breakpoints	branch
BV = OFF				value

*

Key-in Sequence

BR 

Comments

When the BREAK or BR command is entered, the emulator software displays a table containing all the breakpoint commands.

BRO=6B 

The breakpoint commands can be entered individually or from within the BREAK table. In this example, the breakpoint command BRO is enabled from within the BREAK table.

BREAKPOINT SETTINGS				TYPE
BRD= 006BH	BR1= OFF	BR2= OFF	BR3= OFF	location
BRR = OFF				range
BRB = OFF	(go mode only)	BC disables all		branch
BV = OFF	(step mode only)	breakpoints		value

*

Comments

The BREAK table is updated when new breakpoint commands are entered.

Using Trace Display

During step (or non-real time) emulation of the user program, up to four lines of emulation information can be automatically displayed after each instruction is emulated. This information, called trace information, consists of disassembled instructions, register and memory contents, and status information.

The user can enable or disable the automatic display of the emulation information depending on program execution addresses or register contents. The emulator software retains the last state of each trace display command. If the user disables a trace command, the command must be re-enabled before it can be used to display trace information. This section describes how to use the following commands:

- TD Command
- TR Command
- TBn Command
- TS Command
- TRn Command
- TV Command

The TD Command

The instruction trace (TD) command enables or disables the display of the disassembled instruction at a breakpoint or during STEP emulation mode. If enabled, the instruction at the breakpoint is disassembled and displayed, along with the cause of the breakpoint, on one line of the display. Command entry examples are:

TD=ON Enables display of disassembled instructions and the cause of the breakpoint.
 TD=OFF Disables the display of disassembled instructions.

The TR Command

The register display (TR) command enables or disables the display of general purpose registers R0 through R7, the accumulator (ACC) at a breakpoint or during STEP emulation mode, and PSW. If enabled, the registers are displayed on one line of the screen. Two Command entry examples are:

- TR=ON Enables the display of general purpose registers R0 through R7, the accumulator (ACC), and PSW.
- TR=OFF Disables the display of general purpose registers R0 through R7, the accumulator (ACC) and PSW.

The TBn Command

The trace bit command (TBn) enables or disables the display of up to four one-bit memory locations within the direct bit-addressable memory space when emulation halts, where n specifies one of four trace bit commands. Bit-addressable memory contains 128 user-defined software flags located between internal data memory addresses 20H through 2FH as well as several bit locations within several special function registers. Chapter 7 contains a memory map showing the physical layout of the bit-addressable memory. Two command entry examples are:

- TB0=4H Displays the contents of the fifth bit in bit-addressable memory as bit display TB0 when a breakpoint occurs or during STEP emulation mode.
- TB2=OFF Disables the display of bit display TB2.

The TS Command

The status display (TS) command enables or disables the display of the program status word (PSW) register when a breakpoint occurs. The register contains seven 1-bit flags that indicate the current state of the following items:

- Carry flag
- Auxiliary Carry flag
- User Flag 0
- Register Bank Select RS0 and RS1
- Overflow Flag
- Parity Flag
- UTL (Reserved flag not accessible to the user)

Command entry examples are:

- TS=ON Enables the display of the program status word register (PSW) on one line of the screen display.
- TS=OFF Disables the display of the PSW.

Controlling Trace Display

The commands described previously enable specific types of information to be displayed as part of the instruction trace display. The instruction trace display can further be controlled by actions occurring within the user's program.

The emulator contains two commands that enable or disable the instruction trace displays depending on the contents of an 8051 register or when program control passes to a specific address. These commands are covered in the following paragraphs.

The TRn Command

The TRn command enables or disables the trace display (as specified with the TD, TRn, TBn, and TS commands) depending on program control passing to a specified memory address. The user can specify up to four program locations that, if located at the same addresses as breakpoints, enable or disable the instruction trace display. The following example demonstrates how to use this command:

TR0=100H ON Enables the trace display if the instruction at address 100H also has a breakpoint set.
TR1=110H OFF Disables the trace display at address 110H .

In this example, the trace information (specified with the trace display commands) is enabled if the instruction at address 100H is also the location of a breakpoint. In STEP mode, the trace information is displayed for each instruction executed until program control reaches address 110H . In GO emulation, the only instructions displayed are those that also have breakpoints set. At address 110H, the trace display is disabled. Once the trace display is disabled, no further trace information is displayed until the user once again enables the display.

The TV Command

The TV command enables or disables the trace display (as specified by the TD, TRn, TBn, and the TS commands) depending on the contents of a specified register. The user can specify any of the registers R0 through R7 or the accumulator ACC. The following example demonstrates how to use this command:

TV=R0 44H ON Enables the trace display if general purpose register R0 contains 44H.
TV=ACC 10H OFF Disables the trace display if the accumulator contains 10H.

In this example, the trace information is displayed if general purpose register R0 contains the value 44H. The trace information is displayed for each instruction executed until the accumulator contains the value 10H. At this point, the trace display is disabled. Once the trace display is disabled, no further trace information is displayed until the user once again enables the display.

NOTE

If the software trace display is turned off with either the TRn or TV commands, the display will remain off permanently until one of the commands which is set to turn on the display is executed, or until EMV-51 software is reset.

The DTRACE Command

The software trace (DTRACE) command displays a table containing all the trace display controls. Thus, the trace control commands can be entered individually or interactively from the DTRACE table. As the user enters instruction trace commands, the table is updated to reflect the new entry. The DTRACE display remains on the screen until the user enters a non-trace display command. The DTRACE table is then replaced by the results of the new command. The DTRACE display is demonstrated in the next section.

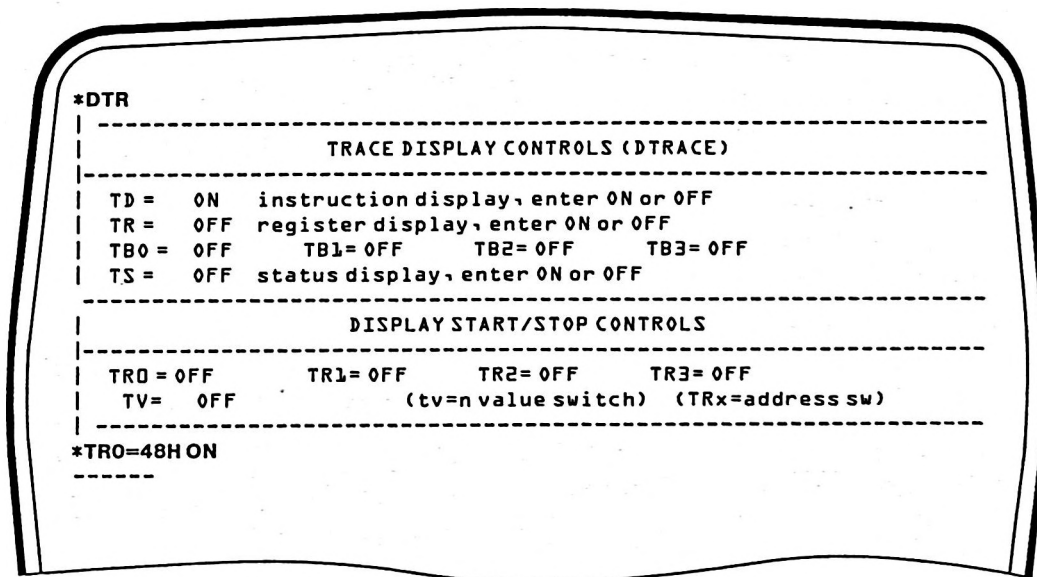
Special Function Key 3

Rather than enter the command DTRACE, the user can enter the function key shown below to obtain the same display. The numeral 3 key is pressed while holding down on the FUNCT key.

FUNCT 3 Displays all trace keywords and their contents on the screen.

DTRACE Screen Displays

The following screen displays demonstrate the instruction trace display commands. The first screen display demonstrates the DTRACE command, while the following displays show the table being updated.



Key-in Sequence	Comments
DTR <input type="button" value="RETURN"/>	Enter the DTRACE command. The emulator software displays a table containing all the tracepoint commands. The DTRACE command can be abbreviated to DTR.
TRO=48H ON <input type="button" value="RETURN"/>	Tracepoint display information can be controlled by actions occurring within the users program. In this example, the tracepoint display is enabled if the contents at address 48H is executed.

*DTR				
TRACE DISPLAY CONTROLS (DTRACE)				
TD =	ON	instruction display, enter ON or OFF		
TR =	OFF	register display, enter ON or OFF		
TB0 =	OFF	TB1 = OFF	TB2 = OFF	TB3 = OFF
TS =	OFF	status display, enter ON or OFF		
DISPLAY START/STOP CONTROLS				
TR0 =	0048H	ON	TR1 = OFF	TR2 = OFF TR3 = OFF
TV =	OFF	(TV=n value switch) (TRx=address sw)		
*				

Comments

When the tracepoint commands are entered from the DTRACE table, the emulator software updates the table and continues displaying the table until a non-tracepoint command is entered.

Program Emulation Commands

The emulator software contains two commands that start emulation of the user's program, the STEP and the GO commands. Both commands check for breakpoints being set, but the STEP command performs the checking in software while the GO command uses hardware breakpoint checking. These commands are described in the following sections.

The STEP Command

The STEP command emulates the user's program one instruction at a time, checking for a trace enable command (TRn or TV) or breakpoint (if enabled) after executing each instruction. The EMV-51 software emulates either a single instruction or a group of instructions. The emulation begins with the instruction pointed to by the program counter, or at the address specified in the STEP command.

During STEP emulation, each instruction is executed and then checked for a trace enable command being set. If software breakpoint checking is enabled, the emulator software also checks for breakpoints being set. If a trace enable command is set, the emulator software displays up to four lines of trace information depending on the trace control commands that are enabled. If a breakpoint is found and breakpoints are enabled, the emulator software stops the emulation and displays the address where the breakpoint occurred. Several command entry examples are:

STEP	Emulates the user program instruction pointed at by the program counter. Breakpoints are not enabled.
STEP FROM 50H	Emulates the user program instruction located at address 50H. Breakpoints are not enabled.

- STEP 50H COUNT=20H** Emulates user program instructions beginning at address 50H and continuing for the next 20H instructions inclusive. Breakpoints are not enabled.
- STEP FROM 50H BR** Emulates user program instructions beginning at address 50H. The BR parameter enables software breakpoint checking.

The GO Command

The GO command emulates the user's program at full speed (real-time) beginning with the instruction pointed to by the program counter. If the user enters an address with the GO command, the emulator software updates the program counter prior to beginning the emulation. The emulation continues until an address, branch, or range break occurs, or the user aborts emulation by pressing the ESC key.

If the user sets address, branch, or range breakpoints prior to entering the GO command, the emulator hardware checks for breakpoints while executing each instruction. If a breakpoint is found, the emulation stops after the instruction executes and EMV-51 displays the last address emulated and the reason for stopping the emulation. Several command entry examples are:

- GO** Begins emulation from the address contained in the program counter. If address, branch, or range breakpoints are set, the emulation stops if a breakpoint is encountered.
- GO FROM 50H** Begins emulation from address 50H after updating the program counter. If address, branch, or range breakpoints are set, the emulation stops if a breakpoint is encountered.

NOTE

The emulator software issues one of the following warning messages if the corresponding conditions occur:

Warning D0 - an interrupt occurs just before the breakpoint to halt emulation.

Warning D1 - a breakpoint occurs on the first instruction of an interrupt service routine.

Warning D2 - when interrupts are enabled, and an 8051 timer is enabled with an initial value of 0 or 1. A spurious timer overflow occurs and an interrupt is generated.

The P Command

The P command displays the previous two instructions emulated by either the STEP or GO commands. The command entry is:

P Displays the last two instructions emulated.

If the command is issued before any instructions are emulated, the emulator software displays the message "NO DATA".

Emulation Control Screen Displays

The demonstration program, introduced in Chapter 3, is used to demonstrate the Emulation commands. In the following display screens, the breakpoint and trace display commands are set up, the STEP and GO emulation commands are demonstrated, and the P command is used to display the two commands last emulated.

```

*BRB=ON
*GO FROM .DEMO      | 0061H=DJNZ   R2,.LOOP      BRB
*

```

Key-in Sequence

Comments

BRB=ON

Enable the branch on breakpoint.

GO FROM .DEMO



The GO command emulates the user's program from an address entered with the command, or from the address contained in the program counter. The emulation continues until a breakpoint is reached. When a breakpoint is reached, the emulator software displays the cause of the breakpoint and the address where it occurred.

```

*GO FROM .DEMO      | 0061H=DJNZ   R2,.LOOP      BRB
*P                   | 0060H=DEC    R0                DASH
*                   | 0061H=DJNZ   R2,.LOOP      DASH
*

```

Key-in Sequence

Comments



After an emulation has been performed, enter the P command to display the last two instructions emulated. In this example, the instructions DEC and DJNZ were the last two instructions emulated with the GO command.


```

*BR
-----
          BREAKPOINT SETTINGS
-----
| BR0= OFF   BR1= OFF  BR2= OFF  BR3= OFF | location
| BRR = OFF                                     | range
| BRB = OFF   (go mode only)  BC disables all | branch
| BV = OFF   (step mode only)  breakpoints    | value
-----
*BR0=61
-----
    
```

Key-in Sequence

BR

Comments

Enter the BREAK command or its abbreviation. The emulator software displays a table containing the breakpoint commands.

BR0= 61

Change the breakpoint commands by entering the desired command followed by the new contents. The emulator updates the BREAK table and continues displaying it as shown in the following screen display.

```

*BR
-----
          BREAKPOINT SETTINGS
-----
| BR0= 0061H  BR1= OFF  BR2= OFF  BR3= OFF | location
| BRR = OFF                                     | range
| BRB = OFF   (go mode only)  BC disables all | branch
| BV = OFF   (step mode only)  breakpoints    | value
-----
*TD=ON
*TR=ON
    
```

Key-in Sequence

TD=ON

Comments

Enable instruction trace.

TR=ON

Enable register trace.


```

*STEP FROM 61 BR
| 0061H=DJNZ R2,.LOOP STEP
ACC=39H PSW=84H R0=04H R1=75H R2=FEH R3=3AH R4=A7H R5=EAH
R6=6EH R7=86H
*

```

Key-in Sequence**Comments****STEP FROM 61 BR**

Enter the STEP command with breakpoints enabled. The EMV-51 software controls the hardware emulation of the user's program and stops the emulation when a breakpoint is encountered. When a breakpoint is detected, the software may display up to four lines of trace information. In this example, the instruction trace (TD) and register trace (TR) were enabled.



Introduction

This chapter describes the macro and conditional command groups. The following sections describe the commands that comprise each group, discuss how the commands are used, and give command entry examples. The following topics are discussed:

- Using Macros
- Using Conditional Commands
- Macro and Conditional Commands Demonstration

Using Macros

A macro is a set of commands that are frequently used. When defined as a macro, these commands are given a name, can be called into use by the single name rather than being entered individually, and can have unique parameter values passed to them each time they are executed.

This section describes the commands that define a macro, save a macro definition, execute a defined macro, and remove a macro.

Defining Macros

The DEFINE Command

The DEFINE command assigns a single name (the macro name) to a group of commands (the macro definition). The name is up to 32 characters long with a colon (:) as the first character. After entering the DEFINE command, the emulator software response is a prompt consisting of a period immediately followed by an asterisk (*).

All user entry, following the prompt, is placed in the new macro definition. The end macro command (described below), terminates the macro definition and the system displays the single asterisk prompt again.

The EM Command

The EM command terminates the macro definition. Entering the EM command, causes the emulator software to close the macro definition and display the single asterisk prompt.

Parameter Passing

Customizing a macro definition is accomplished by passing up to 10 parameters to the commands making up the definition. To indicate the points that receive parameter strings, place a parameter indicator (a percent sign (%)) followed by a

number from 0 to 9) at the desired points. The actual parameter strings are passed to the definition via the macro execution command, described later in this chapter. The following example illustrates how to set up the definition to receive parameter values.

DEFINE :SHOW Defines a new macro definition called SHOW.
DBYTE %0 TO %1 Displays the contents of a range of addresses within data memory. The actual range addresses are passed with the macro execution command.
EM Terminates the macro definition.

NOTE

Commands may be entered to a macro in upper or lower case. However, the system internally converts all the entered characters to upper case.

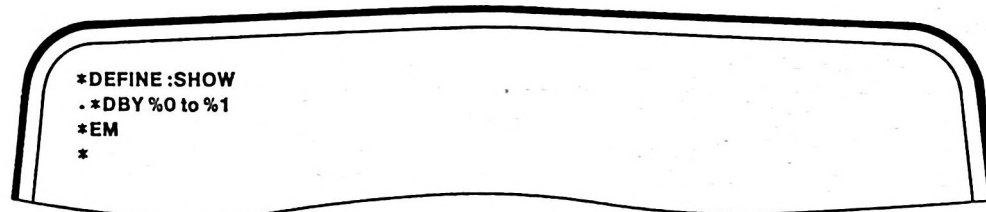
Writing Messages

The WRITE command can be inserted in a macro definition to display messages on the screen. With this command, the result of an expression or a string of characters is displayed. Several examples are shown below:

WRITE 'THE RESULT IS' Displays the message THE RESULT IS when the command is executed.
WRITE .ABLE+100H Displays the result of the expression .ABLE+100H when the command is executed.

Macro Definition Screen Display

The following screen display demonstrates the DEFINE and EM commands. A macro is defined, and receives parameter values when executed.



Key-in Sequence	Comments
DEFINE :SHOW <input type="text" value="RETURN"/>	Macros are defined with the DEFINE command. In this example, the macro :SHOW is defined to display a variable section of memory.
DBY %0 TO %1 <input type="text" value="RETURN"/>	
EM <input type="text" value="RETURN"/>	

Displaying Macro Information

The EMV-51 software contains commands that display the names of macros currently in memory, or the text of specified macro definitions. These commands are described below.

The DIR Command

The DIR command displays a directory of defined macro names currently residing in memory. A macro definition must be defined by the user or loaded from a disk file before the DIR command can display its name. The command is entered as:

DIR Displays a directory of all defined macros.

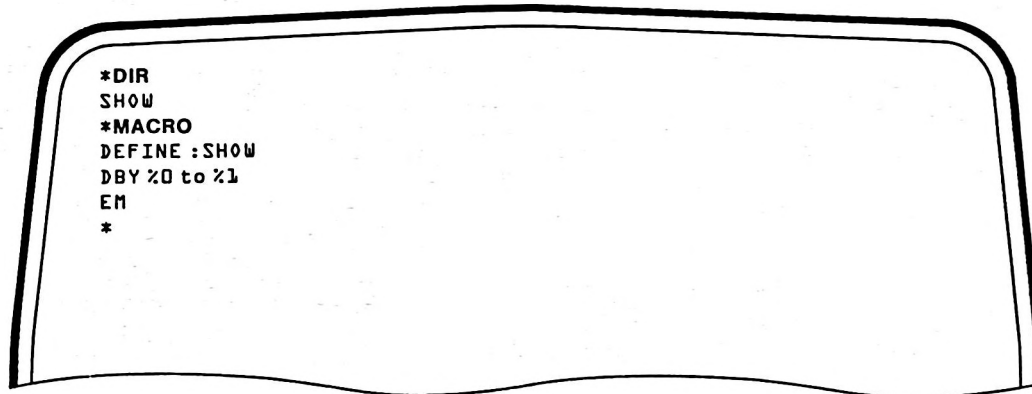
The MACRO Command

The MACRO command displays the definitions of one, several, or all defined macros currently residing in memory. A macro definition must be defined by the user or loaded from a disk file before the MACRO command can display its definition. Several command entry examples are:

- MACRO :NULL Displays the text definition of the macro :NULL.
- MACRO :NULL,:DEMO Displays the text definitions of the two macros :NULL and :DEMO.
- MACRO Displays the text definitions of all defined macros.

Macro Information Screen Displays

The following screen display demonstrates the DIR and MACRO commands by displaying the macros created in the first screen display.



Key-in Sequence	Comments
DIR <input type="button" value="RETURN"/>	The DIR command displays the names of all user-defined macros.
MACRO <input type="button" value="RETURN"/>	The MACRO command displays the names and definitions of all user-defined macros.

Executing Macros

The following sections show how to execute a macro (including a screen display), and list a macro definition.

The Macro Execution Command

The user executes a macro by entering the macro name preceded by a colon (:). If the macro requires parameters for execution, the parameters are entered with the

macro name, separated by commas. The parameter values then replace the parameter indicators (%0 through %9) sequentially.

The following examples demonstrate how to enter a macro name with and without parameter values.:

:DIGIT 3,44	Executes the macro DIGIT and supplies the parameters 3 and 44.
:MULT .MUL1,.MUL2	Executes the macro MULT and supplies the parameters represented by the symbols .MUL1 and .MUL2.
:NULL	Executes a macro called NULL without parameters.
:DEMO	Executes a macro called DEMO without parameters.


Macro Execution Screen Display

The following screen display demonstrates the macro execution command with parameter passing.

```

* :SHOW 0,10H
.R0X
0000H=04H 75H FEH 3AH A7H EAH 6EH 86H 00H 01H 02H 94H 04H 8EH CAH ECH
0010H=6EH
*

```

Key-in Sequence	Comments
:SHOW 0,10H 	The macro :SHOW displays the contents of program memory from address 0 to 10H.

The ENABLE/DISABLE Commands

When a macro definition is executed, the emulator software may or may not list the definition prior to executing it. The default condition is that macro definitions are not listed prior to execution of the macro.

To enable the display of macro definitions before executing the macro, the user enters the command:

ENABLE EXPANSION Enables display of macro definition prior to the execution of the macro definition.

The user can prevent the display of macro definitions by entering this command:

DISABLE EXPANSION Stops display of macro definition prior to the execution of the macro definition.

Deleting Macros (The REMOVE Command)

The REMOVE command deletes macro names from memory. Several command entry examples are:

REMOVE :NULL	Deletes the macro NULL from memory.
REMOVE :NULL,:DEMO	Deletes the macros NULL and DEMO from memory.
REMOVE MACRO	Deletes all macro names from memory.

Saving Macros (The PUT Command)

The PUT command saves a copy of the macro definition, including the DEFINE and EM commands, in a disk file. The macro definition can be retrieved at a later time using the INCLUDE command, described later in this chapter. Several command entry examples are:

PUT :F1:MACRO.SAV :NULL	Saves a copy of the macro :NULL in the file labeled MACRO.SAV on disk drive :F1:.
PUT :F3:MACRO.SAV :DEM,:RUN	Saves a copy of the macros :DEM and :RUN in the file labeled MACRO.SAV on disk drive :F3:.
PUT :F3:SAVE.IT MACRO	Saves a copy of all macros currently in memory in the file labeled SAVE.IT on disk drive :F3:.

NOTE

The PUT command destroys the contents previously in the disk file. If one file is used for all macro definitions, retrieve all macro definitions from the file prior to using the PUT command.

The INCLUDE Command

The INCLUDE command gets a command file from the specified disk file and executes the contents sequentially as they appear in the file. A command file consists of valid EMV-51 commands created with either the ISIS-PDS editor or the DEFINE macro command and stored in a disk file.

Executing Command Files

Command files are loaded with the INCLUDE command and executed sequentially as they appear in the file. For additional information on creating command files, refer to the *iPDS User's Guide*. A command entry example is shown below.

```
INCLUDE :F1:TEST.V1  Loads a command file called TEST.V1 from device
                    :F1:.
```

Executing Macro Command Files

Macro definition commands, previously saved with the PUT command, are loaded with the INCLUDE command and executed. Since the first command in the macro is the DEFINE command, the macro is re-DEFINED in memory and available for future reference. A command entry example is:

```
INCLUDE :F1:MACRO.SAV  Loads a command file called MACRO.SAV from
                      device :F1:.
```

As the macro is loaded into program memory, the emulator software automatically lists the macro definition. The DISABLE EXPANSION command has no effect during execution of the INCLUDE command.

INCLUDE Command Screen Display

```

*ENABLE EXPANSION
*:SHOW 0,10H
.*DBY %0 to %1
*EM
.R0X
0000H=04H 75H FEH 3AH A7H EAH 6EH 86H 00H 01H 02H 94H 04H 8EH CAH ECH
0010H=6EH
*DISABLE EXPANSION
*PUT :F0:MACRO.SAV :SHOW
*

```

Key-in Sequence

Comments

ENABLE EXPANSION

RETURN

The ENABLE EXPANSION command causes macro definitions to be displayed prior to execution of the macro.

:SHOW 0,10 H

RETURN

After the ENABLE EXPANSION command, the macro definition is displayed prior to executing the macro. This display shows the macro command entry.

DISABLE EXPANSION

RETURN

The DISABLE EXPANSION command disables the macro expansion prior to macro execution.

PUT :F0:MACRO.SAV :SHOW

RETURN

The PUT command writes a copy of the macro to the specified file. In this case, the macro :SHOW is written to the file MACRO.SAV on device :F0:.

```

*REMOVE :SHOW
*INCLUDE :F0:MACRO.SAV
*DEFINE :SHOW
.*DBY %0 TO %1
.*EM
*
*

```

Key-in Sequence

Comments

REMOVE :SHOW

RETURN

The Remove command deletes the macro definition show from the symbol table. Each time a macro is INCLUDED, its definition must be REMOVED.

INCLUDE :F0:MACRO.SAV

RETURN

The INCLUDE command retrieves macro definitions from the specified file. In this case, the file is MACRO.SAV. When the macro definitions are retrieved from the file, the definition is displayed on the screen.


```
*WRITE 'THIS IS DUD'S
**TWO (2) LINE MESSAGE'
THIS IS DUD'S
TWO (2)
LINE MESSAGE
*
```

Key-in Sequence**Comments****WRITE 'THIS IS DUD'S****TWO (2) LINE MESSAGE'**

The WRITE command displays a message on the screen. In this example, the message includes an embedded apostrophe and carriage return. When the message is displayed, it is displayed on two lines with the correct single apostrophe.

Using Compound Commands

Compound commands provide the user with conditional testing and looping capabilities. Conditional testing uses the traditional IF..THEN..ELSE commands, while the looping controls use both the REPEAT..WHILE/UNTIL and COUNT..STEP..WHILE/UNTIL commands. The following sections cover:

- Setting Up Conditional IF Command
- Setting Up The REPEAT Command
- Setting Up The COUNT Command

Setting Up Conditional IF Commands

The IF..THEN..ORIF..ELSE command sequence permits conditional testing and execution of groups of EMV-51 commands. The IF command can be nested to any depth. The following command entry examples show several methods of using the IF command.

In the first command entry example, the command STEP is executed if the contents of symbolic name .LOOP is equal to 5. If the test is false, the command GO is executed. Chapter 8 contains additional information about the IF command.

```
IF .LOOP=5 THEN  If .LOOP=5 then execute the command
    STEP        STEP, otherwise execute
ELSE           the command GO.
    GO
ENDIF        The IF..LOOP terminator.
```

In the second command entry example, the value of the symbolic label .T determines the sequence of a series of tests. If the initial value of .T is less than 3, the testing begins. If the value of .T is greater than 1, phase B is performed. If the value of .T equals 1, phase C is performed.

```

IF .T < 3 THEN
  .ABLE = 10 * .T
  .T = .ABLE / 10
  WRITE 'PHASE A PASSED'
  IF .T > 1 THEN
    .BAKER = .ABLE * .T / 4H
    .T = .BAKER / .ABLE
    WRITE 'PHASE B PASSED'
    STEP FROM .T COUNT = .ABLE
    IF .T = 1 THEN
      STEP FROM .T COUNT = .BAKER
      WRITE 'ALL PHASES PASSED'
    ELSE
      WRITE 'PHASE C FAILED'
    END
  END
ELSE
  WRITE 'TEST FAILED'
  STEP FROM .T BR
ENDIF

```

Conditional IF Screen Display

The following screen display demonstrates a typical application of the conditional IF command. The command is used to emulate various sections of a program depending on the contents of a specified register.

```

*DEFINE:WORK2
.*DBY 3E=40
.*DEFINE.Z=1
.*REP
.*WHILE.Z>0
.*STEP FROM (DBY 3E)
.*IF DBY 3E=48 THEN
.* WRITE 'BYTETAKEN'
.* ORIF (DBY 3E)=54
.* WRITE 'ASCII STORED'
.* ORIF (DBY 3E)=61
.* .Z=.Z-1
.* DBY 6F TO 74
.* ENDIF
.*DBY 3E=(DBY 3E)+1
.*END
.*EM
*

```

Key-in Sequence

DEFINE:WORK2

RETURN

DBY 3E=40

RETURN

DEFINE.Z=1

RETURN

REP

RETURN

Comments

This macro demonstrates how the REPEAT, WHILE, STEP, IF...ORIF, and WRITE commands can be combined to emulate the portions of a program. Messages are displayed when various points in the program are emulated. The following screen displays show the results produced by this macro.

Key-in Sequence

```
WHILE .Z > 0   
STEP FROM (DBY 3E)   
IF (DBY 3E) = 48 THEN   
WRITE 'BYTE TAKEN'   
ORIF (DBY 3E) = 54   
WRITE 'ASCII STORED'   
ORIF (DBY 3E) = 61   
.Z = .Z - 1   
DBY 6F TO 74   
ENDIF   
DBY 3E = (DBY 3E) + 1   
END   
EM 
```

```

* :WORK2
.DEMO          | 0040H=MOV      R1, #.BUFFER      DASM
                | 0041H=XRL      A, R7          STEP
                | 0042H=MOV      R2, #03H      DASM
                | 0043H=RR       A           STEP
                | 0044H=CLR      .CY          DASM
                | 0046H=PUSH     .R0X         DASM
                | 0047H=NOP     STEP
                | 0048H=MOV      A, @R0       DASM

.LOOP
BYTE TAKEN    | 0049H=INC      R0          DASM
                | 004AH=INC      R0          DASM
                | 004BH=INC      R0          DASM
                | 004CH=ADD     A, @0        DASM
                | 004DH=DA      A           DASM
                | 004EH=PUSH     .ACC        DASM
                | 004FH=MOVX    A, DPTR      STEP
                | 0050H=ANL     A, #0FH      DASM
                | 0051H=INC     R7          STEP
                | 0052H=ORL     A, #30H      DASM
                | 0053H=JNB     F7H, 005FH   STEP
                | 0054H=MOV     @R1, A       DASM
    
```

Key-in Sequence

Comments

:WORK2 

When the macro :WORK2 is executed, the results shown in the screen displays are produced.

Stop scrolling of screen display.

Continue scrolling of display.

```

ASCII STORED  | 0055H=INC      R1          DASM
                | 0056H=POP     .ACC        DASM
                | 0057H=MOVX    A, @DPTR      STEP
                | 0058H=SWAP    A           DASM
                | 0059H=ANL     A, #0FH      DASM
                | 005AH=INC     R7          STEP
                | 005BH=ORL     A, #30H      DASM
                | 005CH=INC     R7          STEP
                | 005DH=MOV     @R1, A       DASM
                | 005EH=INC     R1          DASM
                | 005FH=DEC     R0          DASM
                | 0060H=DEC     R0          DASM
                | 0061H=DJNZ    R2, .LOOP    STEP

.BUFFER
006FH=30H 30H 30H 7EH 00H 82H
*
    
```

Setting up the REPEAT Command

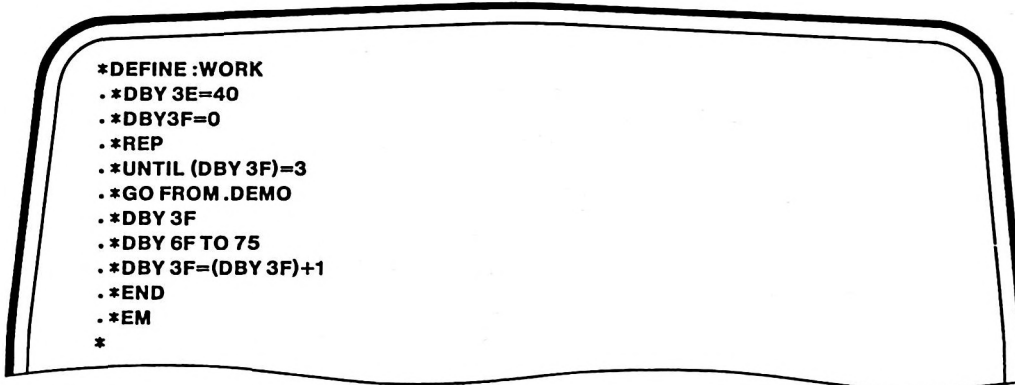
The REPEAT..WHILE and REPEAT..UNTIL command sequences execute a series of commands conditionally. The REPEAT..WHILE sequence executes a series of commands while a specified condition is true. The REPEAT..UNTIL sequence executes a series of commands until a specified condition becomes true. The UNTIL and WHILE clauses can appear once, never, or any number of times, anywhere in the loop.

The following examples illustrate both forms of the REPEAT command.

<pre>REPEAT WHILE DBYTE 0 > 55H STEP FROM 10H COUNT = 10H DBYTE 0 = (DBYTE 0) - 1 END</pre>	<p>Begins a loop that repeats while memory location 0 in data memory is greater than 55H. The loop executes STEP on each pass, then decrements the memory location. Begins a loop that repeats until memory location 0 in data memory equals 55H. The loop executes STEP on each pass, then increments the memory location.</p>
<pre>REPEAT UNTIL DBYTE 0 = 55H STEP FROM 10H COUNT = 10H DBYTE 0 = (DBYTE 0) + 1 END</pre>	

REPEAT Command Screen Displays

The following screen display demonstrates a typical application of the REPEAT command. The command emulates a program until a memory location equals a specified value.



Key-in Sequence	Comments
------------------------	-----------------

DEFINE :WORK

RETURN

DBY 3E=40 RETURN

DBY 3F=0 RETURN

REP RETURN

UNTIL (DBY 3F)=3 RETURN

GO FROM .DEMO RETURN

DBY 3F RETURN

DBY 6F TO 75 RETURN

DBY 3F=(DBY 3F)+1 RETURN

END RETURN

EM RETURN

Macros can reference data memory for function arguments. In this case, the UNTIL and DBY commands get their arguments from data memory. The following screen shows the execution results of this macro.

```

*BRO=.DONE
*:WORK
.DONE          006BH=P0P          .ROX          GO-BREAK
ACC=3FH PSW=80H RD=FFH R1=75H R2=00H R3=FFH R4=E3H R5=01H R6=FFH R7=8CH
  CARRY=1      AUX=0      FLAG=0 RBS=00      OVERFLOW=0 UTL=0 PAR=0

D BYTE .ROX+003FH=00H
. BUFFER
006FH=38H 3FH 36H 3FH 37H 3FH 31H
DONE          006BH=P0P          .ROX          GO-BREAK
ACC=3FH PSW=80H RD=FFH R1=75H R2=00H R3=FFH R4=E3H R5=01H R6=FFH R7=8CH
  CARRY=1      AUX=0      FLAG=0 RBS=00      OVERFLOW=0 UTL=0 PAR=0

D BYTE .ROX+003FH=01H
. BUFFER
006FH=38H 3FH 36H 3FH 37H 3FH 31H
DONE          006BH=P0P          .ROX          GO-BREAK
ACC=3FH PSW=80H RD=FFH R1=75H R2=00H R3=FFH R4=E3H R5=01H R6=FFH R7=8CH
  CARRY=1      AUX=0      FLAG=0 RBS=00      OVERFLOW=0 UTL=0 PAR=0

D BYTE .ROX+003FH=02H
. BUFFER
006FH=38H 3FH 36H 3FH 37H 3FH 31H
*
```

Key-in Sequence	Comments
-----------------	----------

*BRO=.DONE	<input type="button" value="RETURN"/>	Set a breakpoint at the symbolic name .DONE
------------	---------------------------------------	---

*:WORK	<input type="button" value="RETURN"/>	Execute the macro.
--------	---------------------------------------	--------------------

Setting Up the COUNT Command

The COUNT..WHILE and COUNT..UNTIL command sequences execute a series of commands a specified number of times. The WHILE and UNTIL clauses permit conditional termination of the COUNT command sequence depending on conditions within the program.

The following examples illustrate both forms of the COUNT command sequence.

<pre> COUNT 12H WHILE D BYTE 0 > 55H STEP FROM 10H COUNT = 10H END</pre>	<p>Begins a loop that repeats 12H times and while memory location 0 in data memory exceeds 55H. The loop executes STEP on each pass, with the code presumably altering the contents of data memory location 0.</p> <p>Terminates the COUNT command.</p>
<pre> COUNT 12H UNTIL D BYTE) = > 55H STEP FROM 10H COUNT = 10H END</pre>	<p>Begins a loop that repeats 12H times or until data memory location 0 equals or exceeds 55H. The loop executes STEP on each pass, with the code presumably altering the contents of data memory location 0.</p> <p>Terminates the COUNT command.</p>

Using Special Function Keys

The emulator software recognizes ten special function keys numbered 0 through 9. These keys are entered by typing the corresponding number while holding down the FUNCT key similar to the way a SHIFTEd character is typed. Function

keys 1 through 4 are predefined and were described in Chapters 3 through 5. Function keys 0 and 5 through 9 are user-defined. These keys can be assigned to a macro name if the macro does not require parameters passed to it. These predefined and user defined function keys can also be used by pressing only the numbered key followed by a carriage return.

Assigning Function Keys to Macros

The following examples demonstrate the procedure to assign a function key to a macro definition. Macro names associated with functions keys can be from 1 to 9 characters in length.

FUN=0 :DEMO Assigns function key 0 to the macro definition labeled :DEMO.
FUN=5 :NULL Assigns function key 5 to the macro definition labeled :NULL.

Function Key Precautions

The user can assign function keys 0 through 9 to macro definitions, but 0 and 5 through 9 should be assigned first. If function keys 1 through 4 are re-defined, the user can not easily retrieve the system-defined function. The system definition for function keys 1 through 4 can be retrieved by one of the following methods:

1. EXIT the emulation session and re-load the EMV-51 software by entering EMV51. This procedure retrieves the original function key definitions, but destroys all programs currently loaded.
2. Define a macro that duplicates the original function and assign the desired function key to the macro definition.

For example, if the user re-defined function key 4 which generates the register display, the following procedure can be used to retrieve the original definition.

Step 1 Define a macro that duplicates the original function:

DEFINE :REG Defines a macro labeled :REG that duplicates the original function key.
REG
EM Displays the register keywords and their contents when executed.

Step 2 Assign the desired function key to the new macro.

FUN=4 :REG Assigns function key 4 to the macro :REG that re-creates the original function.

Displaying Assigned Function Keys

The user can display all assigned function keys by entering the command:

FUNCTION Displays all assigned function keys.

Advanced Commands Demonstration

This section demonstrates a practical use of the macro and conditional commands by defining a macro that uses conditional commands to emulate the demonstration program generated in Chapter 3.


```

*DEFINE :MESS
.*WRITE'
.**LOAD NEXT PROGRAM
.**'
.*EM
*MESS

LOAD NEXT PROGRAM
*
    
```

Key-in Sequence

Comments

DEFINE :MESS

The WRITE command can be used to display messages on the screen. In this example, the message is "LOAD NEXT PROGRAM", preceded by an embedded carriage return.

WRITE'

LOAD NEXT PROGRAM

When the macro :MESS executes, it writes a message to the display. The screen display shows the message produced by the macro.

,

EM

:MESS

```

*DEFINE :TEST
.*:WORK
.*:MESS
.*EM
    
```

Key-in Sequence

Comments

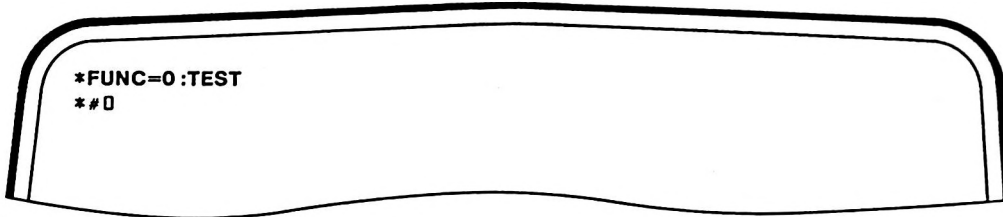
DEFINE :TEST

Macro definitions can include other macro definitions. In this example, the macro :TEST consists of the macros :WORK and :MESS.

:WORK

:MESS

EM



Key-in Sequence

Comments



=0 :TEST

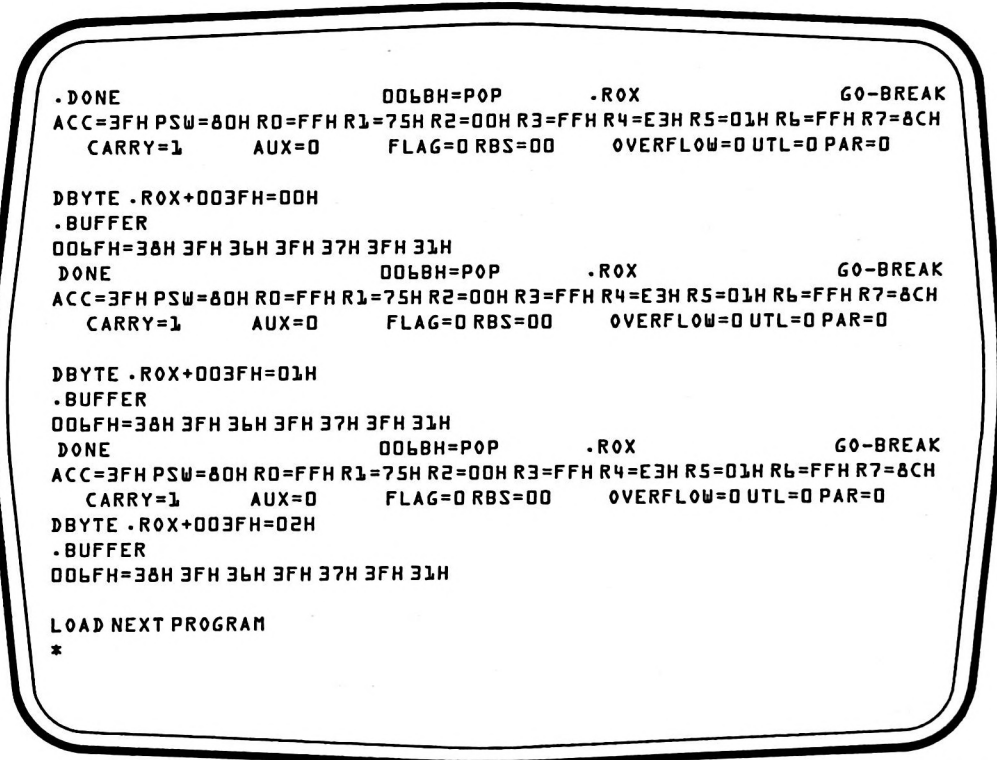
Assign the macro :TEST to function Key 0



When the function key is depressed, the display shown in the next screen is produced.



=0



```

*REMOVE :WORK
*DEFINE :WORK
.*DBY 20T=40
.*COUNT 3T
.*STEP FROM (DBY 20T)
.*IF ACC=0 THEN
.*REG
.*CBY 0 TO 10T
.*DBY 0 TO 10T
.*ELSE
.*CRY R0 TO R0+10
.*DBY R0 TO R0+10
.*ENDIF
.*DBY 20T = (DBY 20T)+1
.*END
.*EM
*

```

Key-in Sequence

Comments

REMOVE :WORK	<input type="button" value="RETURN"/>	The macro must be removed before it can be redefined. This screen Display defines a macro that has an error.
DEFINE :WORK	<input type="button" value="RETURN"/>	
DBY 20T=40	<input type="button" value="RETURN"/>	
COUNT 3T	<input type="button" value="RETURN"/>	
STEP FROM (DBY 20T)	<input type="button" value="RETURN"/>	
IF ACC=0 THEN	<input type="button" value="RETURN"/>	
REG	<input type="button" value="RETURN"/>	
CBY 0 TO 10T	<input type="button" value="RETURN"/>	
DBY 0 TO 10T	<input type="button" value="RETURN"/>	
ELSE	<input type="button" value="RETURN"/>	
CRY R0 TO R0+10	<input type="button" value="RETURN"/>	
DBY R0 TO R0+10	<input type="button" value="RETURN"/>	
ENDIF	<input type="button" value="RETURN"/>	
DBY 20T = (DBY 20T)+1	<input type="button" value="RETURN"/>	
END	<input type="button" value="RETURN"/>	
EM	<input type="button" value="RETURN"/>	

```

*ENABLE EXPANSION
*:WORK
.*DBY 20T=40
.*COUNT 3T
..*STEP FROM (DBY 20T)
...*IF ACC=0 THEN
...*REG
...*CBY 0 TO 10T
...*DBY 0 TO 10T
...*ELSE
...*CRY RD TO RD+10

ERR 80 SYNTAX ERROR
*

```

Key-in Sequence**Comments****ENABLE EXPANSION**

:WORK

When macro expansion is enabled, the emulator software displays a macro definition prior to executing the macro. In this case, the last line displayed contains an error that prevented the macro from executing.

```

*REMOVE :WORK
*DEFINE :WORK
.*DBY 20T=40
.*COUNT 3T
.*STEP FROM (DBY 20T)
.*IF ACC=0 THEN
.*REG
.*CBY 0 TO 10T
.*DBY 0 TO 10T
.*ELSE
.*CBY R0 TO R0+10
.*DBY R0 TO R0+10
.*ENDIF
.*DBY 20T = (DBY 20T)+1
.*END
.*EM
*

```

Key-in Sequence

Comments

REMOVE :WORK	<input type="button" value="RETURN"/>	The macro must be removed before it can be redefined to correct the error. Enter the revised macro.
DEFINE :WORK	<input type="button" value="RETURN"/>	
DBY 20T=40	<input type="button" value="RETURN"/>	
COUNT 3T	<input type="button" value="RETURN"/>	
STEP FROM (DBY 20T)	<input type="button" value="RETURN"/>	
IF ACC=0 THEN	<input type="button" value="RETURN"/>	
REG	<input type="button" value="RETURN"/>	
CBY 0 TO 10T	<input type="button" value="RETURN"/>	
DBY 0 TO 10T	<input type="button" value="RETURN"/>	
ELSE	<input type="button" value="RETURN"/>	
CBY R0 TO R0+10	<input type="button" value="RETURN"/>	
DBY R0 TO R0+10	<input type="button" value="RETURN"/>	
ENDIF	<input type="button" value="RETURN"/>	
DBY 20T = (DBY 20T)+1	<input type="button" value="RETURN"/>	
END	<input type="button" value="RETURN"/>	
EM	<input type="button" value="RETURN"/>	

```

*:WORK
.*DBY 20T=40
.*COUNT 3T
.**STEP FROM (DBY 20T)
.**IF ACC=0 THEN
...*REG
...*CBY 0 TO 10T
...*DBY 0 TO 10T
...*ELSE
...*CBY RD TO RD+10
...*DBY RD TO RD+10
...*ENDIF
.*DBY 20T = (DBY 20T)+1
.**END
.*EM
    
```

Key-in Sequence

:WORK 

Comments

The corrected macro is executed with expansion enabled. The following screen display illustrates the macro execution.

After the macro definition is corrected, the macro is executed and expanded without any errors.

```

.DEMO                                0040H=M0V      R1,#.BUFFER      STEP
ACC=88H PSW=00H RO=00H R1=6FH R2=03H R3=3AH R4=A6H R5=EAH R6=6EH R7=88H
CARRY=0      AUX=0      FLAG=0 RBS=00      OVERFLOW=0 UTL=0 PAR=0

.RDX
0000H=33H 31H C9H CCH 33H 33H CDH CCH 39H 33H CDH C8H 33H 33H C8H C8H
0010H=33H
.RDX
0000H=00H 6FH 03H 3AH A6H EAH 6EH 88H 00H 01H 0FH 4EH 04H 8EH CAH ECH
0010H=6EH

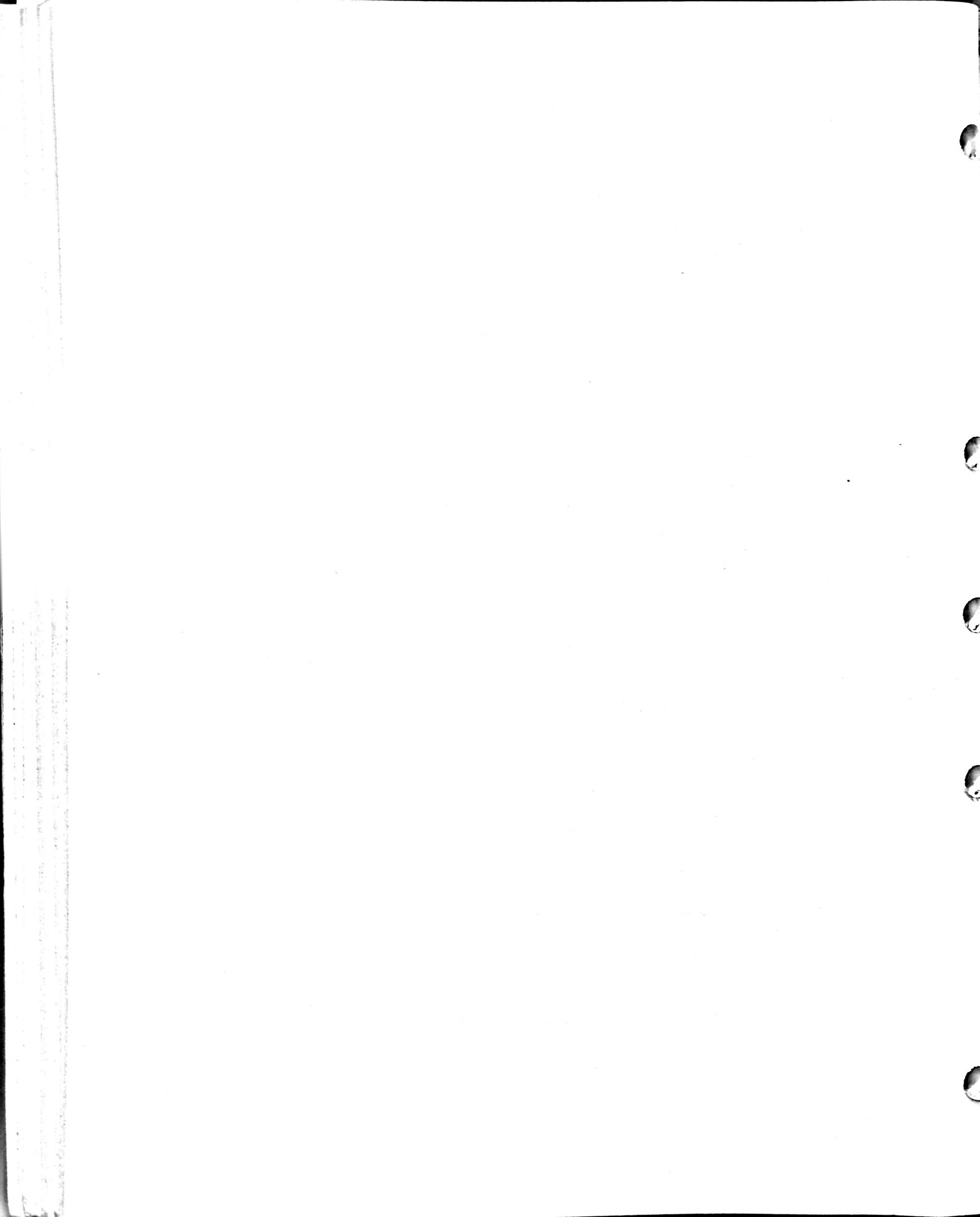
| 0041H=XRL      A, R7      STEP
ACC=00H PSW=00H RO=00H R1=6FH R2=03H R3=3AH R4=A6H R5=EAH R6=6EH R7=88H
CARRY=0      AUX=0      FLAG=0 RBS=00      OVERFLOW=0 UTL=0 PAR=0

.RDX
0000H=33H 31H C8H CCH 33H 33H CDH CCH 39H 33H CDH C8H 33H 33H C8H C8H
0010H=33H
.RDX
0000H=00H 6FH 03H 3AH A6H EAH 6EH 88H 00H 01H 0FH 4EH 04H 8EH CAH ECH
0010H=6EH

| 0042H=M0V      R2,#03H      STEP
ACC=00H PSW=00H RO=00H R1=6FH R2=03H R3=3AH R4=A6H R5=EAH R6=6EH R7=88H
CARRY=0      AUX=0      FLAG=0 RBS=00      OVERFLOW=0 UTL=0 PAR=0
    
```

Comments

When the macro :WORK is executed and the trace display (TD), trace register (TR), and trace status (TS) commands are enabled, the display shown above is produced.





Introduction

This chapter contains reference material on the EMV-51 emulator and the MCS-51 microcontroller family. It lists the keywords used by the emulator software and defines the MCS-51 operators used in the EMV-51 software.

EMV-51 Symbols

The emulator software contains a table of predefined system symbolic names, known as system symbols. Table 7-1 lists the symbols and gives a definition of each.

Table 7-1 System Symbols

KEYWORD	DEFINITION
.ACC	Accumulator
.B	Multiplication Register Address
.P0	Port 0
.SP	Stack Pointer
.DPL	Data Pointer, Low Byte
.DPH	Data Pointer, High Byte
.TCON	Timer/Counter Port 0
.TMOD	Timer Mode
.TL0	Timer 0 Low Byte
.TL1	Timer 1 Low Byte
.TH0	Timer 0 High Byte
.TH1	Timer 1 High Byte
.P1	Port 1
.SCON	Serial Port Control
.SBUF	Serial Port Buffer
.P2	Port 2
.IE	Interrupt Enable
.P3	Port 3
.IP	Interrupt Priority
.PSW	Program Status Word

In addition to the system symbols, the EMV-51 software uses several one-character keywords as punctuation marks. These are listed in Table 7-2.

Table 7-2 Punctuation and Delimiter Keywords

Keyword	Meaning
.	Identifies symbolic reference
,	Separates items in a list
()	Controls order of evaluation in expressions
+	Indicates a positive value or addition
-	Indicates a negative value or subtraction
*	Serves as the EMV-51 prompt character
=	Assigns one value to another
>	The greater than operator sign
<	The less than operator sign
*	Encloses opcode mnemonic constants in expressions
'	Encloses string characters
::	Identifies a macro name
%	Identifies formal parameters in macro definitions

EMV-51 Operators

The EMV-51 software supports four types of operators: arithmetic, content, relational, and logical operators. An operator is a category of commands that perform specific functions. As an example, the CONTENT operator contains the commands that access the various memory regions in the 8051 memory map. The following paragraphs describe the four categories and list the operators in each.

Arithmetic Operators

Arithmetic operators consist of keywords that specify the five types of arithmetic operations recognized by the emulator software. Each arithmetic operator returns a 16-bit value. If the operation involves an 8-bit register as the destination the register receives the low-order eight bits and the upper eight bits are truncated. Table 7-3 lists the arithmetic operators.

Table 7-3 Arithmetic Operators

Operator	Operation
+	Unary Plus
-	Unary Minus (2's complement)
*	Multiplication
/	Integer Division
MOD	Modulo division
+	Addition
-	Subtraction

Content Operators

Content operators consist of keywords that reference specific types of memory addressed by the microcontroller. Table 7-4 lists the content operator keywords, while the following paragraphs describe each memory space.

Table 7-4 Content Operator

Operator	Type of Memory
CBYTE	Code memory
DBYTE	On-chip data memory
RBYTE	Special Function Register memory
PBYTE	External data memory
RBIT	Bit-addressable memory

Code Memory (CBYTE)

The MCS-51 microcontroller family can address 64K bytes of program memory. The 8051 and 8751 microcontrollers separate program memory into 4K bytes internal memory and 60K bytes external program memory, while the 8031 provides 64K bytes of external program memory.

The emulator hardware contains 4K bytes of RAM that can optionally replace the 4K bytes of internal program memory of the 8051/8751 microcontrollers. The user can replace the internal 4K byte memory space in the 8051/8751 microcontrollers with the 4K bytes of RAM within the emulator hardware by setting control jumpers as described in Appendix A.

CBYTE Restrictions

When using the emulator hardware 4K RAM as the microcontroller ROM, the user can emulate and debug a program containing up to 64K bytes of instructions. In the external 60K bytes of user program memory, the user can set breakpoints and emulate programs, but can not modify the program code.

A program longer than 4K bytes can be emulated and debugged in blocks of up to 4K instructions. The memory addresses for each block must be modified to load into the EMV-51's 4K RAM. As each block is emulated and debugged, it can be programmed into a PROM, and inserted into the external 60K of program memory space. In this manner, the entire user program can be emulated and debugged. However, if the user enables external memory from 0 to 4K, the CBYTE command is limited to reading memory contents. It can not modify program instructions in external memory.

On-Chip Data and Register Memory (DBYTE, RBIT, and RBYTE)

The MCS-51 microcontrollers contain 128 bytes of internal data memory organized as 32 general-purpose registers, 16 bytes of bit-addressable memory, and 80 bytes of general-purpose RAM. In addition, the microcontrollers contain a block of memory that consists of input/output registers and various internal control registers. Figure 7-1 illustrates the organization of the internal memory.

On-Chip Data Memory (DBYTE and RBIT). The user can access the entire 128 bytes of internal data memory with the DBYTE keyword. The RBIT keyword is used to access the 16 bytes of bit-addressable memory on a bit-by-bit level and bit-addressable portions of the 8051 registers as shown in Figure 7-1 and Table 7-5.

Special Function Register Memory (RBYTE). The MCS-51 microcontroller family contains a block of memory that consists of the input/output registers and various internal control registers. The emulator software assigns system symbolic names to the registers in this memory block. The user can access the pre-defined registers by entering the RBYTE keyword followed by an address or system symbol. Table 7-1 lists the contents, system symbol, and a brief description of the register memory locations. Refer to the MCS-51 Macro Assembler User's Guide, manual order number 9800937, for additional information.

External Data Memory (PBYTE)

The MCS-51 microcontroller family isolates program instructions from program data by providing separate 64K byte blocks of program memory and data memory. The PBYTE keyword accesses the 64K bytes of external data memory.

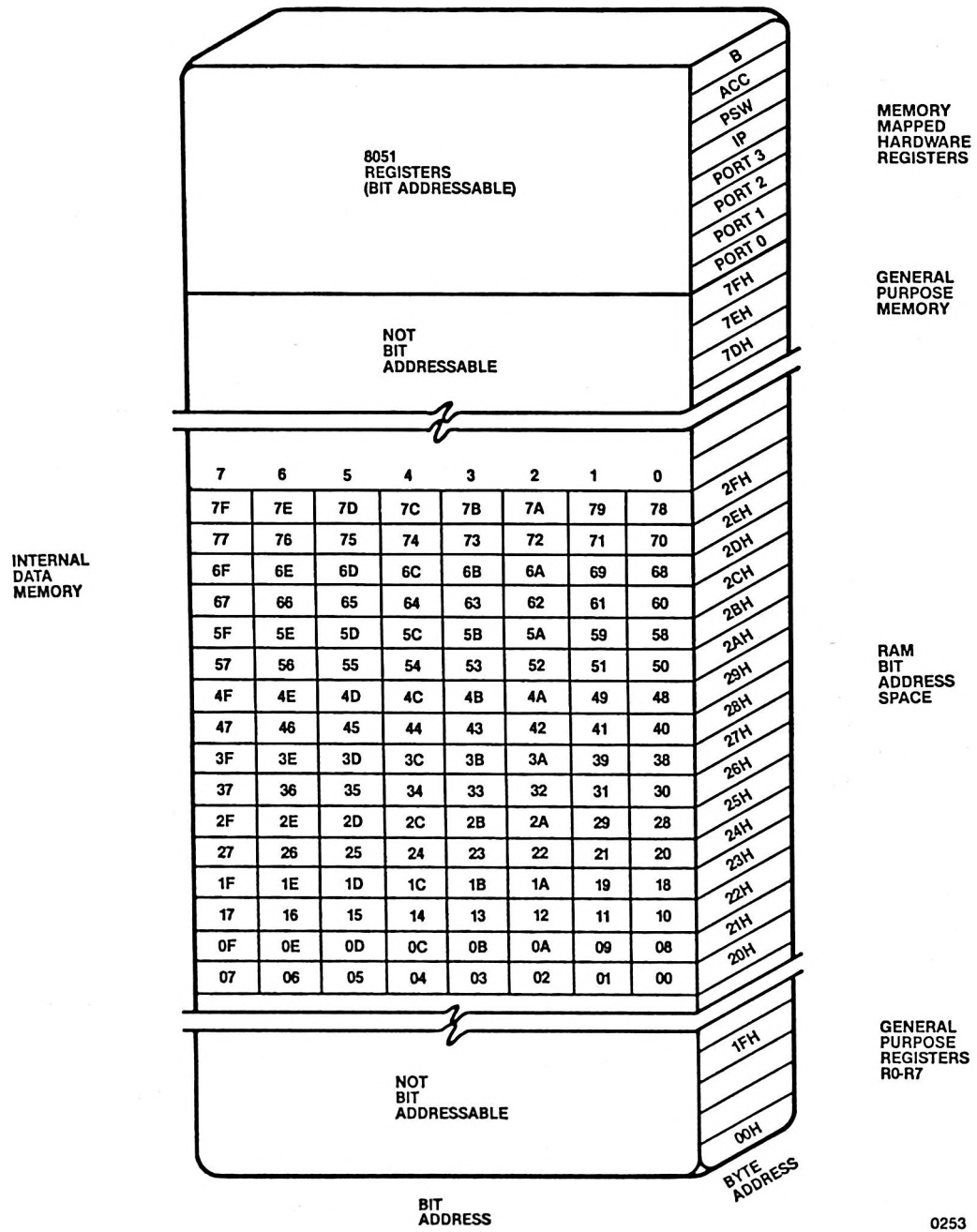


Figure 7-1 Internal Data Memory Organization

Table 7-5 Bit-Addressable Register Memory Contents

Hex Address	System Symbol	Meaning
80H-87H	.P0	Port 0
88H	.IT0	Timer Control
89H	.IE0	Timer 0 Interrupt Edge Flag
8AH	.IT1	Timer 1 Interrupt Type Control Bit
8BH	.IE1	Timer 1 Interrupt Edge Flag
8CH	.TRO	Timer 0 Run Control Bit
8DH	.TFO	Timer 0 Overflow Flag
8EH	.TR1	Timer 1 Run Control Bit
8FH	.TF1	Timer 1 Overflow Flag
90H-97H	.P1	Port 1
98H	.RI	Receive Interrupt Flag
99H	.TI	Transmit Interrupt Flag
9AH	.RB8	Receive Bit 8
9BH	.TB8	Transmit Bit 8
9CH	.REN	Receiver Enable
9DH	.SM2	Serial Mode Control Bit 2
9EH	.SM1	Serial Mode Control Bit 1
9FH	.SM0	Serial Mode Control Bit 0
A0H-A7H	.P2	Port 2
A8H	.EX0	Enable External Interrupt 0
A9H	.ET0	Enable Timer 0 Interrupt
AAH	.EX1	Enable External Interrupt 1
ABH	.ET1	Enable Timer 1 Interrupt
ACH	.ES	Enable Serial Port Interrupt
B0H	.RXD	Serial Port Receive Pin
B1H	.TXD	Serial Port Transmit Pin
B2H	.INT0	Interrupt 0 Input Pin
B3H	.INT1	Interrupt 1 Input Pin
B4H	.T0	Timer/Counter 0 External Flag
B5H	.T1	Timer/Counter 1 External Flag
B6H	.WR	Write Data (For External Memory)
B7H	.RD	Read Data (For External Memory)
B8H	.PX0	Priority of External Interrupt 0
B9H	.PT0	Priority of Timer 0 Interrupt
BAH	.PX1	Priority of External Interrupt 1
BBH	.PT1	Priority of Timer 1 Interrupt
BCH	.PS	Priority of Serial Interrupt
D0H	.P	Parity Flag
D2H	.OV	Overflow Flag
D3H	.RS0	Register Bank Select Bit 0
D4H	.RS1	Register Bank Select Bit 1
D5H	.F0	Flag 0
D6H	.AC	Auxiliary Carry Flag
D7H	.CY	Carry Flag
E0H-E7H	.ACC	Accumulator
F0H-F7H	.B	Multiplication Register

Relational Operators

Relational operators consist of keywords that indicate the desired relational comparison to be performed. Table 7-6 lists the relational operators.

Table 7-6 Relational Operators

Operator	Relation
=	Is equal to
>	Is greater than
<	Is less than
>=	Is greater than or equal to
<=	Is less than or equal to
<>	Is not equal to

Logical Operators

Logical operators consist of keywords that specify a logical combining of numbers. Table 7-7 lists the logical operators.

Table 7-7 Logical Operators

Operator	Operation
NOT	1's complement
AND	Bitwise AND
OR	Bitwise OR
XOR	Bitwise exclusive OR



Introduction

This chapter presents the information necessary to understand the EMV-51 commands. It describes how commands are entered and edited, the form of EMV-51 commands, command format notations, and gives definitions of terms used in the command format descriptions. The information in this chapter is designed to be used by the more experienced user of the EMV-51 emulator. The following topics are covered:

- Entering Commands
- Form of EMV-51 Commands
- Command Description Formats
- Special Command Format Terms
- Alphabetical Listing of Commands

Entering Commands

EMV-51 operations are controlled by command lines. The emulator software receives command lines from one of four sources:

1. Commands entered from the keyboard.
2. Previously defined command files containing valid EMV-51 commands. After loading the EMV-51 software, the command file is loaded and executed using the INCLUDE command (described in Chapter 6).
3. Macros, defined as a sequence of related commands, stored in a disk file. After loading the EMV-51 software, the macro definition is loaded and executed using the INCLUDE command (macros and the INCLUDE command are described in Chapter 6).
4. Previously defined files loaded and executed using the SUBMIT command. Refer to the ISIS-PDS User's Guide for additional information on the SUBMIT command.

Entering the Command Line

An EMV-51 command line is a sequence of one or more command words separated by spaces and ending with a carriage return. Normally, the command line is limited to 122 characters. In most cases, the command words can be abbreviated to one, two, or three characters.

Specifying the Workfile

Then the emulator software invocation command (EMV51) is entered, the ISIS-PDS operating system automatically sets a workfile on the drive containing the EMV-51 software. This workfile is used for temporary storage of macro

definitions. While entering the invocation command, the user can specify an optional drive number for the workfile. The following command demonstrates specifying another drive as the location of the workfile:

```
:F1:EMV51 WORKFILE (:Fn:)
```

In this example, the emulator software diskette is inserted in drive :F1: and the workfile is specified as being on drive :Fn:, where n is replaced by 0, 1, 2, 3, or 4.

Command Line Execution

With six exceptions, the carriage return causes the command to be interpreted and, if no syntax errors are detected, executed immediately. The exceptions are macro commands, continuation lines, the REPEAT command, the COUNT command, the WRITE command, and the IF command. Macro, REPEAT, COUNT, WRITE, and IF commands are described in Chapter 6; continuation lines are described in the following section.

Continuation Lines

If a command line exceeds 122 characters, the line must be separated into multiple lines, known as continuation lines. Continuation lines are formed by entering an ampersand character (&) and a carriage return prior to the 122nd character.

When entering the ampersand and carriage return, the emulator software responds by displaying a double asterisk prompt (**). The double prompt indicates the EMV-51 is accepting a command line that exceeds the 122 character maximum length. The command line is not executed until termination of a continuation line with just a carriage return. At this point, the emulator returns to displaying a single asterisk prompt character.

NOTE

If the continuation character (&) appears in a comment line, it is interpreted as a part of the comment.

Entry Editing and Display Control

The emulator software recognizes control characters that permit editing of command line entries and controlling of the display generated by command execution. These control characters are described in the following sections.

Command Line Editing

The control characters list below are used to correct errors in command line entry. The control characters consist of one or two keyboard characters. If a character is preceded by the word CTRL, press the CTRL key and simultaneously press the specified character. Otherwise, the indicated key is entered.



Aborts any operation that is executing or aborts any command being entered.



Signifies the end of the command line.



Pressing the RUBOUT key deletes the last character typed. One character is deleted each time the key is pressed.



Gets the next keyboard entry and places the character in the command line buffer without interpreting it.



Echo command line being entered.



Deletes the current line only, not any previous lines. If several continuation lines have been entered, the CTRL X deletes only the current portion.



Deletes an entire command line, including all continuation lines.

Controlling the Display

The emulator software displays information on the screen by displaying a line, scrolling the displayed line upward one line, and displaying another line. Control of the output to the display is accomplished with the control characters listed below. Refer to the iPDS User's Guide for a complete explanation of control character functions.



Stops the output of new information to the screen.



Continues the output of new information to the screen.



Slows down or speeds up screen display. Pressing FUNCT S once slows the display down, while pressing it again speeds up the display.

Form of EMV-51 Commands

The command format requirements, listed later in this chapter, use special command format notations terms. This section lists the format notations and defines the command notation terms.

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 formats.

Command format notation consists of symbols adopted to help describe EMV-51 commands. These symbols are not part of the command but are used to precisely describe the format of the command.

The special characters used in these formats have no significance to the emulator software 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:

```
DEFINE .<label> = <expression> <cr>
```

<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.

[<option>] Brackets enclose optional material that may or may not be included on the command line. For example, [=<radix>] is an optional item that may be appended to the SUFFIX command.

... 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 }
```

INCLUDE Indicates a choice must be made to enter either Y or N. The following example indicates that either A or B or neither can be selected. The enclosed choices are printed in a vertical column.

```
[ { A }  
  { B } ]
```

{ <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.

the underline denotes a valid abbreviation of the command word.

Expression

An expression `<expr>` is composed of numerals, keywords, and symbolic references. The expression is evaluated to a 16-bit number that replaces the original expression.

A numeral consists of one or more digits and, optionally a one character suffix that specifies the desired number base. The value can be preceded by a unary sign that indicates whether the value is positive or negative. Table 8-1 lists the number bases and numerical digits recognized by EMV-51 software.

Table 8-1 Available Number Bases and Digits

Number Base	Valid Digits	Explicit Radix	Example
Binary (base 2)	0,1	Y	11110011Y
Octal (base 8)	0-7	Q	363Q
Decimal (base 10)	0-9	T	243T
Hexadecimal (base 16)	0-9, A-F	H	F3H

Example:

123T Decimal numeric expression
 12H*23H Hexadecimal numerals connected by an arithmetic operator
 to form an expression.
 .ABLE AND .BAK Symbolic references connected by a logical operator to form
 an expression.

String

The term string refers to a sequence of one or more alphanumeric characters enclosed in apostrophes ('), as shown below:

'HELLO' The apostrophes (') enclose the string HELLO.

If a string contains an apostrophe, use two apostrophes to mark the apostrophe. When the string is displayed on the screen, the emulator software displays the double apostrophe as a single apostrophe. The following example contains an embedded apostrophe.

'they"re' Double apostrophe marks an embedded apostrophe.

When a string of characters is stored in memory, each character occupies one byte or memory location. In the example given above, the string HELLO would occupy 5 memory locations. If the string has more than one character, the emulator uses consecutive memory locations to store each character.

Source

The term `<source>` in command lines refers to an assembler mnemonic input to the command. The `<source pn>` is the assembler mnemonic input file.

Destination

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

Command

The term `<command>` in command lines refers to any valid EMV-51 command. The `<command list>` is a sequence of EMV-51 commands.

Condition

A `<condition>` can have the same kinds of entries as `<expr>`. However, in a `<condition>` only the least significant bit (LSB) of the result is tested.

Alphabetical Listing of Commands

The following pages contain an alphabetical list of the EMV-51 commands and their command formats.

ACCUMULATOR

Display or change the contents of the accumulator

Command Format

ACC [= `<expr>`]

where

`<expr>` specifies the 8-bit numeric value that replaces the current contents of the accumulator. The value can be specified as a symbolic name or an expression.

Comments

The accumulator (ACC) is a general purpose register used in data transfers and arithmetic operations. The ACC command displays or changes the contents of the register. When `<expr>` is used, it is evaluated to a 16-bit number and truncated to the lower 8-bits.

Examples

ACC

displays the current contents of the accumulator. The numeric value is displayed according to the default base.

ACC=10H

changes the contents of the accumulator to the value 10H.

ASM

Assembles one line of instruction mnemonics into program memory

Command Format

```
ASM [ {ORG <expr> } ]
```

where

ORG <expr> changes the contents of the assembly location counter to the 16-bit value of <expr>. The expression consists of numeric values and symbolic references.

<source> specifies a valid 8051/8751/8031 instruction mnemonic.

Comments

The ASM command displays the current value of the assembly location counter, changes the current contents of the assembly location counter to the indicated value, or assembles one line of 8051 microcontroller instructions into program memory.

The assembly location counter is a pointer that indicates where the next available location is in program memory. It is automatically incremented for each instruction stored in memory and the new value is displayed.

When a new instruction is assembled into program memory, it is stored at the location indicated by the contents of the assembly location counter. Refer to the Microcontrollers User's Guide, order number 210359, for additional information on the 8051 microcontroller instruction set.

Examples

```
ASM
```

displays the current address value in the assembly location counter.

```
ASM ORG 0
```

sets the assembly location counter to program memory location 0.

```
A NOP
```

assembles the NOP instruction into memory at the location pointed to by the assembly location counter.

B

Display or change the contents of
the multiplication register

Command Format

B [= <expr>]

where

<expr> specifies the 8-bit numeric value that replaces the current contents of the B register. The value can be specified as a symbolic name or an expression.

Comments

The multiplication register (B) is used as a multiplication register during arithmetic operations and a general purpose register for all remaining operations. The B command displays or changes the contents of the register. When <expr> is used, it is evaluated to a 16-bit number and truncated to the lower 8-bits.

Examples

B

displays the current contents of the B register. The numeric value is displayed according to the default base.

B=10H

changes the contents of the B register to the value 10H.

BASE

Displays or changes the base for display of numeric data

Command Format

$$\underline{\text{BASE}} \left[= \left\{ \begin{array}{c} \text{Y} \\ \text{H} \\ \text{T} \\ \text{Q} \end{array} \right\} \right]$$

where

Y specifies binary as the base for display of numeric data.

H specifies hexadecimal as the base for display of numeric data.

T specifies decimal as the base for display of numeric data.

Q specifies octal as the base for display of numeric data.

Comments

The BASE command displays or changes the base for the display of numeric values. The initial base is hexadecimal (H). All numeric values are displayed according to the contents of base.

Examples

BASE

displays the base for numeric values displayed on the screen.

BAS=T

changes the base to decimal.

BC

Clears all breakpoints

Command Format

BC

Comments

The BC command clears all breakpoints set by the user. Breakpoints are not recognized during emulation after the BC command is executed until set again.

Examples

BC

removes all breakpoints set by the user.

BRB

Enable or Disables
the branch breakpoint

Command Format

$$\text{BRB} \left[= \begin{Bmatrix} \text{ON} \\ \text{OFF} \end{Bmatrix} \right]$$
Comments

The BRB command enables or disables the branch breakpoint. Prior to the start of GO emulation and if BRB is enabled, the emulator places a breakpoint at the location of the next instruction that causes a branch. The BRB breakpoint disables all other breakpoints while it is active. During STEP emulation, the BRB breakpoint is ignored. The default state of BRB is OFF.

The emulator does not distinguish between conditional branches that fail (fall through) and those that pass (branch). The break occurs on the fetch of the branch instruction, not on its outcome.

Examples

BRB=ON

enables the branch breakpoint.

BRB=OFF

disables the branch breakpoint.

BREAK

Displays a table containing all
break commands and their values

Command Format

BREAK

Comments

The BREAK command displays a table containing all breakpoint commands and their current states. The display remains on the screen until a non-break command is entered, such as the GO command. When a break command is entered with a new state or content, the table is updated to reflect the new contents.

Examples

BREAK

displays a table consisting of all the break commands and their current state or content. The table remains on the screen until a non-break command is entered.

BRR

Specifies a range breakpoint

Command Format

$$\text{BRR} \left[= \left\{ \begin{array}{l} \langle \text{expr1} \rangle \text{ TO } \langle \text{expr2} \rangle \\ \text{OFF} \end{array} \right\} \right]$$

where

<expr1> specifies the beginning address within the range of addresses. It can be specified as an actual address, symbolic name, or an expression.

<expr2> specifies the ending address within the range of addresses. It can be specified as an actual address, symbolic name, or an expression.

OFF disables the Range Breakpoint.

Comments

The BRR command specifies a range breakpoint between the limits of **<expr1>** and **<expr2>**, where **<expr1>** is less than or equal to **<expr2>**. When breakpoints are enabled and program execution branches to an address within the specified range during an emulation, the EMV-51 hardware stops the current emulation and returns control to the user. The default state of BRR is OFF.

Examples

```
BRR=100H TO 150H
```

sets a range breakpoint between addresses 100H and 150H.

```
BRR=.ABLE+4 TO .SETUP-2
```

sets a range breakpoint between the addresses indicated by the expressions **.ABLE+4** and **.SETUP-2**.

```
BRR=OFF
```

disables the range breakpoint.

```
BRR
```

displays the current range setting.

BR<n>

Specifies up to four address breakpoints

Command Format

$$BR<n> \left[= \left\{ \begin{array}{l} <expr> \\ OFF \end{array} \right\} \right]$$

where

- <n> specifies a numeric value between 0 and 3 that indicates the desired address breakpoint.
- <expr> consists of numeric values, symbol names, and mathematical operators. The expression is evaluated to a 16-bit result.
- OFF disables the specified address breakpoint.

Comments

The BR<n> command specifies from 1 to 4 address breakpoint: BR0, BR1, BR2, and BR3. When the 8051 fetches an instruction at an address specified by an address breakpoint during an emulation, the emulation is stopped and control is returned to the user. The default state of the address breakpoints is OFF.

Examples

BR0=100H

sets an address breakpoint for address 100H.

BR2=.ABLE+5

sets an address breakpoint for address specified by the expression .ABLE+5.

BR3=OFF

disables address breakpoint number 3.

BR0

displays the current breakpoint 0 setting.

BV

Specifies a value breakpoint

Command Format

$$BV \left[= \left\{ \begin{array}{l} \langle \text{register} \rangle \langle \text{expr} \rangle \\ \langle \text{OFF} \rangle \end{array} \right\} \right]$$

where

<register> specifies one of the following registers of the 8051 microcontroller: R0-R7 or ACC.

<expr> consists of numeric values, symbol names, and mathematical operators. The expression is evaluated to 8 or 16 bits.

OFF disables the value breakpoint.

Comments

The BV command specifies a register and specific value as the value breakpoint. It is used in the STEP emulation mode. The **<register>** is one of the 8051 microcontroller registers, while the value is an 8 or 16-bit number depending on the register selected.

When the 8051 executes an instruction that references the specified register when the register contains the indicated value, program emulation is stopped and control is returned to the user. The default state of the BV breakpoint is OFF.

Examples

BV=ACC 87H

specifies the accumulator when it contains the value 87H as the value breakpoint. When the accumulator contains 87H during emulation, the emulator software stops program execution and returns control to the user.

BV=OFF

disables the value breakpoint.

CBYTE

Display or modify contents of program memory

Command Format

$$\text{CBYTE } \langle \text{expr1} \rangle \left[\left\{ \begin{array}{l} \text{TO } \langle \text{expr2} \rangle \\ \text{LEN } \langle \text{expr4} \rangle \end{array} \right\} \right] \left[= \left\{ \langle \text{expr3} \rangle \right\} \left[\langle \text{expr} \rangle \right] \left[\langle \text{string} \rangle \right] \dots \right]$$

where

- <expr1>** specifies a beginning address within program memory.
- TO <expr2>** specifies an ending address within program memory.
- LENGTH <expr4>** specifies a range of locations within program memory to be displayed or operated on. The <expr4> indicates the number of locations to be displayed.
- <expr3>** specifies the 8-bit value stored at the address specified in <expr1>. If a value greater than 0FFH is specified, the software uses the least significant byte as the result. The expression can consist of numerals, symbol names, and arithmetic operators.
- <string>** specifies an alphanumeric string of characters that are stored in memory beginning at <expr1>. Each character in the string is stored in successive memory locations.

Comments

The CBYTE command displays or modifies the contents of program memory. The command can display or modify the contents of EMV-51 program memory. The user can read the contents of external program memory from 4K to 64K, but can not modify the contents of that memory.

If the TO or LENGTH parameters are entered, the command displays the memory contents at the indicated addresses. If the equal parameter (=) is entered, the command modifies the contents of memory to the value specified by the user. The optional TO, LENGTH, and equal sign can be combined within the command line to address multiple memory locations.

Examples

CBYTE 0 TO 3FH

displays the contents of program memory locations 0 to 3FH.

CBYTE 0 LEN 9 = 1,2,CBYTE 56H

changes the contents of program memory locations 0 through 9 to the sequence 1, 2, and the contents of program memory location 56H. The sequence (1, 2, CBYTE 56H) is repeated until all ten memory locations are changed.

CDUMP

Display content of program memory as both hexadecimal and ASCII values

Command Format

```
CDUMP <expr1> TO <expr2>
```

where

<expr1> specifies the beginning address of a range of addresses to be displayed.

TO <expr2> specifies the ending address of a range of addresses to be displayed.

Comments

The CDUMP command displays the contents of program memory as hexadecimal values and ASCII characters. The hexadecimal values are displayed on the left side of the display, while the ASCII characters are displayed on the right side of the display.

Examples

```
CDUMP 80H TO 83H
```

displays the contents of program memory locations 80 to 83H as hexadecimal values and ASCII characters.

```
CDUMP .ABLE TO .FIN
```

displays the contents of program memory as hexadecimal values and ASCII characters. The addresses are specified by the contents of the symbolic names .ABLE and .FIN.

COUNT

Begins a program loop

Command Format

```

COUNT <expr> <cr>
[ UNTIL <condition> <cr>
  WHILE <condition> <cr> ]...
<command> <cr>
END

```

where

- <expr> specifies the number of times the loop is repeated.
- WHILE indicates that the loop terminates when the least significant bit of the condition tested is false.
- UNTIL indicates that the loop terminates when the least significant bit of the condition tested is true.
- <condition> indicates the test condition that satisfies the WHILE/UNTIL clauses. Only the least significant bit (LSB) of the <condition> is tested.
- <command> is any EMV-51 command entered by the user that executes until the test condition is met.

Comments

The COUNT...END command sequence forms a command loop that executes until a specified count is reached or a specified condition is encountered. If the optional test condition is met during the loop execution, the programming loop terminates before the final count is reached. The optional test value is entered with the WHILE or UNTIL clause.

Examples

```

COUNT 5
  WHILE DBYTE 0 < .TOTAL
  STEP
  DBYTE 0 = (DBYTE 0) + 1
END

```

the loop executes 5 times or until the contents of DBYTE 0 equals or exceeds the value assigned to the symbolic name .TOTAL.

```

COUNT = .CNT
  STEP
  UNTIL DBYTE 0 = .TOTAL
  DBYTE 0 = (DBYTE 0) + 1
END

```

the loop executes 10T times or until the contents of DBYTE 0 equals the value assigned to the symbolic name .TOTAL.

DASM

Displays program memory
as instruction mnemonics

Command Format

DASM <expr1> TO <expr2>

where

<expr1> specifies the beginning address of the range of addresses to be disassembled. It consists of numeric values, symbolic references, and mathematical operators.

TO <expr2> specifies the end address of a range of addresses to be disassembled. It consists of numeric values, symbolic references, and mathematical operators.

Comments

The DASM command disassembles the contents of program memory and displays the contents as mnemonic instructions. If symbolic display is enabled (via the ENABLE command) and a match is found, the command displays opcode addresses as symbolic names.

An instruction is displayed if its first byte is within the range of indicated addresses. The address contained in <expr1> must point to the first byte of an instruction. Otherwise, the command produces invalid results.

Examples

DASM 100H TO 150H

disassembles the contents of program memory locations 100H through 150H.

DAS .ABLE TO .BAKER

disassembles the contents of program memory beginning with the location indicated by the contents of the symbolic name .ABLE and ending with the location indicated by the contents of the symbolic name .BAKER.

DBYTE

Display or modify contents
of internal data memory

Command Format

DBYTE <expr1> [{ TO <expr2> }] [= { <expr3> } [, <expr>] [, <string>] ..]

where

- <expr1> specifies a beginning address within internal data memory.
- TO <expr2> specifies an ending address within internal data memory. The address must be between 0 and 7FH.
- LENGTH <expr4> specifies a range of locations within internal data memory to be displayed. The <expr4> indicates the number of locations to be displayed. It ranges between 1 and 80H.
- <expr3> specifies the 8-bit value stored at the address specified in <expr1>. The value consists of numerals, symbol names, and arithmetic operators.
- <string> specifies an alphanumeric string of characters that are stored in memory beginning at <expr1>. Each character in the string is stored in successive memory locations.

Comments

The DBYTE command displays or modifies the contents of internal data memory, depending on how the command is entered.

If the TO or LENGTH parameters are entered, the command displays the memory contents at the indicated address. If the equal parameter (=) is entered, the command modifies the contents of memory to the value specified by the user. The optional TO, LENGTH, and equal sign can be combined within the command line to assign values to multiple memory locations.

Examples

DBYTE 0 TO 3FH

displays the contents of data memory locations 0 to 3FH.

DBYTE 0 LEN 9 = 56H

changes the contents of data memory locations 0 through 8 to 56H.

DBYTE 7 = 'THIS IS A STRING'

changes the contents of data memory beginning at address 07H and continuing for the length of the string.

DDUMP

Display content of external data memory
as both hexadecimal and ASCII values

Command Format

DDUMP <expr1> TO <expr2>

where

<expr1> specifies the beginning address of a range of addresses to be displayed.

TO <expr2> specifies the ending address of a range of addresses to be displayed.

Comments

The DDUMP command displays the contents of external data memory as hexadecimal values and ASCII characters. The hexadecimal values are displayed on the left side of the display, while the ASCII characters are displayed on the right side of the display.

Examples

DDUMP 8H TO 4AH

displays the contents of data memory locations 8H to 4AH as hexadecimal and ASCII values.

DDUMP .ABLE+4 TO .SEC+10H

displays the contents of data memory as hexadecimal values and ASCII characters.

DEFINE

Defines symbolic names
or macro definitions

Command Format

DEFINE {:<string>
<string> = <expr> }

where

:<string> specifies the beginning of a macro definition where <string> is an alphanumeric string of 2 to 32 characters, including the colon, that is the name of the new macro.

- .<string> specifies an alphanumeric string of 2 to 32 characters, including the period, that becomes a user symbolic name.
- <expr> specifies the 16-bit value to be assigned to the symbol name. It consists of numeric values, symbolic references, and mathematical operators.

Comments

The DEFINE command performs two tasks:

1. Evaluates the expression entered with it and assigns the 16-bit result to the symbolic name. Once assigned, the user can enter the symbolic name rather than the numeric value or expression. The symbolic name is added to the end of the EMV-51 user symbol table.
2. Begins a new macro definition where :<string> is the name of the new macro. All commands following the DEFINE command are placed in the new definition until the EM command, which terminates the definition.

Once the macro definition is created, the user can save it in a disk file with the PUT command (described later in this chapter).

Examples

```
DEFINE .CHALK=.ABLE+FFH
```

assigns the result of the expression .ABLE+FFH to the new symbolic name .CHALK.

```
DEF :NULL
```

begins a macro definition labeled :NULL. All commands following the DEFINE command are entered into the definition until the EM command, which terminates the definition. The macro name is added to a user macro list maintained by the emulator software.

DIR

Displays the names of all macros

Command Format

```
DIR
```

Comments

The DIR command displays the names of all macros in the user macro list. This includes macros in RAM memory and in the temporary macro workfiles. The macro text is not displayed.

DISABLE

Disables the display of symbolic names or macro text

Command Format

`DISABLE {SYMBOLIC }
{EXPANSION}`

Comments

The DISABLE command performs two functions:

1. Disables the display of symbolic names for the screen display commands: DTRACE, ASM, and the entry or display of memory locations. When a numeric value is encountered in one of the above commands, the value itself is displayed according to the current BASE.
2. Disables the expansion of macro definitions prior to execution of the macro. The default condition is the disable condition, where macro definitions are not expanded.

Examples

`DISABLE SYMBOLIC`

disables the use of symbolic names for numeric values.

`DISABLE EXPANSION`

disables the expansion of macro definitions prior to execution of the macro.

DPTR

Display or change the contents of the data pointer register

Command Format

`DPTR [= <expr>]`

where

<expr> specifies the 16-bit address value to be assigned to the data pointer register. It can consist of numeric values, symbol names, and mathematical operators.

Comments

The data pointer register (DPTR) serves as the base register in indirect jumps, table look-up instructions, and external data transfers. The DPTR command displays or changes the contents of the data pointer register. When <expr> is used, it is evaluated to a 16-bit value.

Examples

DPTR

displays the current contents of the data pointer register. The numeric value is displayed according to the default base.

DPTR=100H

changes the contents of the data pointer register to the value 100H.

DTRACE

Displays a table containing all trace display commands

Command Format

DTRACE

Comments

The DTRACE command displays a table containing the trace display commands and their current condition, whether enabled or disabled. The table remains on the screen until a non-trace display command is entered, such as the GO command. When a trace display command is entered with a new state or content, the table is updated to reflect the new contents.

Examples

DTRACE

displays a table consisting of the trace display commands and their current state or content. The table remains on the screen until a non-trace command is entered.

ENABLE

Enables the display of symbolic names or macro text

Command Format

ENABLE {SYMBOLIC
EXPANSION }

Comments

The ENABLE command performs two functions:

1. The display of symbolic names for the screen display commands: DTRACE, ASM, and the entry or display of memory locations. When a numeric value corresponds with a value assigned to a symbolic name, the symbolic name is displayed rather than the numeric value. Otherwise, numeric values are displayed as a symbolic name plus an offset.
2. Enables the expansion of macro definitions prior to the execution of the macro definition. The default state is macro expansion disabled.

Examples

ENABLE SYMBOLIC

enables the use of symbolic names for numeric values.

ENABLE EXPANSION

enables the expansion of macro definitions prior to the execution of the macro.

EVALUATE

Evaluates an expression entered with the command

Command Format

EVALUATE <expr>

where

<expr> specifies an arithmetic expression or symbolic name that consists of numeric values, symbolic references, and mathematical operators. The expression is evaluated and a 16-bit result is generated.

Comments

The <expr> is evaluated and a 16-bit numeric value is produced. The value is displayed in binary, octal, decimal, hexadecimal, and the ASCII equivalent. In addition, the value is displayed as an offset from a symbol name that is less than the value.

Examples

```
EVALUATE 30H+23*(.ABLE+4)
```

evaluates the <expr> 30H+23H*(.ABLE+4) and displays the 16-bit result in four bases, ASCII, and an offset from the nearest symbol name.

```
EVALUATE .ABLE
```

displays the value assigned to the symbolic name .ABLE and displays the result in four bases, ASCII, and an offset from the nearest symbol name.

EXIT

Terminates the debugging session
and returns control to ISIS-PDS

Command Format

EXIT

Comments

The EXIT command terminates the debugging session and returns control ISIS-PDS operating system. The user's program and macros should be saved if desired prior to entering the EXIT command.

Example

```
EXIT
```

terminates the current debugging session and returns control to the operating system.

FUNCTION

Displays or assigns function keys

Command Format





FUNCTION [= <n> :<string>]

where

- <n> a numeric value between 0 and 9 inclusive.
- <string> a pre-defined macro name, where the name consists of 2 to 9 alphanumeric characters including the colon.

Comments

The FUNCTION command assigns macro names to the specified function keys. Macros requiring parameters are not assigned to function keys. Function keys 1 through 4 are pre-defined as listed below:

- | | | |
|---|----------|--|
|  | 1 | Executes the MEMORY command and produces the table containing the display/modify commands. |
|  | 2 | Executes the BREAK command and produces the table containing the breakpoint commands. |
|  | 3 | Executes the DTRACE command and produces the table containing the trace display commands. |
|  | 4 | Executes the REGISTER command and produces the table containing the register control commands. |

The function keys 1 through 4 can be re-defined, but special steps must be taken to retrieve the original definitions. The two methods that can be used to retrieve the original definitions are listed below:

1. EXIT the emulation session and reload the EMV-51 software by entering the EMV51 command. This procedure retrieves the original function key definitions.
2. Define a macro that duplicates the original function and assign the macro to the desired function key.

The user can incorporate function keys in command files as described below:

1. Enter a numeric value within the range of 0 through 9. This value corresponds to the desired function key number.
2. Enter a carriage return.

When the command file is loaded with the INCLUDE command, the numeric value between the carriage returns executes as if the user had depressed the FUNCT key and one of the numeric keys 0-9 on the keyboard.

Examples

FUNCTION=0 :DEMO

assigns the macro :DEMO to function key 0.

FUN=5 :MULT

assigns the macro :MULT to function key 5.

FUN

displays all function key assignments.

NOTE

All predefined and user defined functions can be accessed by pressing the number associated with the function without pressing the FUNCT key.

GO

Begins full speed emulation
of the user's program

Command Format

GO [[FROM] <expr>]

where

<expr> specifies an address where emulation is to begin. Consists of numeric values, symbolic references, and mathematical operators. The expression is evaluated to a 16-bit result.

Comments

The GO command begins program emulation from the address contained in the program counter or from the address entered with the command. The GO command differs from the STEP command as described below.

1. Breakpoints are checked in hardware with the GO command, where they are checked in software with the STEP command.
2. Trace display controls are not checked during execution of the GO command.

Examples

GO

begins emulation at the address contained in the program counter and proceeds indefinitely or until a breakpoint is encountered.

G FROM 100H

begins emulation at address 100H and proceeds indefinitely or until a breakpoint is encountered.

HELP

Displays general or specific information about EMV-51 commands

Command Format

HELP [item][,...]

where

item specifies an EMV-51 command or keyword.

Comments

The HELP command provides information about EMV-51 commands and keywords. The user can enter multiple items with the command.

Examples

HELP

displays a list of the HELP items. The user can obtain assistance with any item in the list.

HELP LOAD,LIST

displays information about the LOAD and LIST commands.

IF...THEN...ELSE

Conditionally executes a series of commands

Command Format

```
IF <cond1> [THEN] <cr>
  [<command> <cr>...]
  [ORIF <cond2> <cr>
  [<command> <cr>] ...] ...
  [ELSE <cr>
  [<command> <cr>] ...]
```

ENDIF

where

- <cond1> specifies a conditional test whose 16-bit result is true or false. Consists of numeric values, symbolic references, and mathematical operators.
- <cond2> specifies a conditional test whose result is true or false. Consists of numeric values, symbolic references, and mathematical operators.
- <command> specifies one or a list of EMV-51 commands to be executed if the test condition is true.

Comments

The IF..THEN..ELSE conditional command sequence performs conditional execution of command sequences. The <cond1> and <cond2> specify a conditional tests whose 16-bit results are true (low-order bit is a one) or false (low-order bit is a zero). If the result is true, the command(s) following the conditional test are executed. Otherwise, the ELSE or ENDIF commands are executed.

During command execution, the emulator software performs the following tests on the IF..THEN..ELSE sequence:

1. If <cond1> is true, execute the <command> following it and branch to ENDIF.
2. If <cond1> is false, test <cond2>. If true execute the command list following <cond2> and branch to ENDIF. If multiple ORIF's are present, test each <cond2> until one is true or all test false.
3. If <cond1> and <cond2> are both false, execute the <command> following the ELSE and branch to ENDIF.

The <command> specifies that one or more of EMV-51 commands are entered by the user. The series of commands are executed and then control branches to the ENDIF command.

Examples

```
IF DBYTE 100=1
  STEP FROM 100
ELSE
  CBYTE 0 TO 200
ENDIF
```

this example performs the conditional test DBYTE 100=1. If the test is true, the command STEP FROM 100 is executed and control passes to the ENDIF command. Otherwise the command CBYTE 0 TO 200 is executed.

```
If .G=0 THEN
  STEP FROM 100H
  ORIF .G=1 THEN
    .ABLE=.ABLE+1
    STEP FROM 200H
  ORIF .G=.ABSCAN THEN
    CBYTE 0 TO 300H
    .G=.G+1
ELSE
  GO FROM 50H
ENDIF
```

In this example, the IF and ORIF commands perform conditional command execution by testing for a specified condition. Depending on the test result, the emulator software performs from 1 to 3 conditional tests. If all the tests fail, the command following the ELSE command is executed. If one of the tests passes, the commands following the test are executed, and control passes to the ENDIF command.

```
IF .T<3 THEN
  .ABLE=10*.T
  .T=.ABLE/10
  IF .T>1 THEN
    .BAKER=.ABLE*.T/4H
    .T=.BAKER/.ABLE
    STEP FROM .T COUNT=.ABLE
    IF .T=1 THEN
      STEP FROM .T COUNT=.BAKER
    END
  END
ELSE
  STEP FROM .T BR
ENDIF
```

In this example, the value of the symbolic label .T determines the execution sequence of a series of nested IF commands. If the initial value of .T is less than 3, the testing begins. If the value of .T is greater than 1, the second IF command is executed. If the value of .T equals 1, the third IF command is executed.

INCLUDE

Loads a macro definition
or command file

Command Format

INCLUDE <source pn>

where

<source pn> specifies the file name of the macro or command file.

Comments

The <source pn> is any valid ISIS-PDS device name followed by a filename as described in the iPDS User's Guide. The emulator software executes any valid EMV-51 commands as they are loaded from the <source pn>.

Macro definitions are only loaded with the INCLUDE command. Since the first command in the macro definition is the DEFINE command, the emulator software recreates the macro and lists the definition on the screen. The DISABLE EXPANSION command has no effect when loading macros from a file.

Command files are also loaded with the INCLUDE command. A command file is created prior to loading the EMV-51 software and consists of valid EMV-51 commands. Refer to the iPDS User's Guide for additional information on command files.

The INCLUDE command can be nested within any other command, including a macro definition.

Examples

INCLUDE :F1:MAC.SAV

loads the contents of filename MAC.SAV and executes the contents as they are entered. In this case the file consists of macro definitions, but the file could be a command file.

INCLUDE :CI:

loads the contents of filename :CI:. Since this file is the keyboard, the user can enter EMV-51 commands as if the commands originated from a command file. A control-Z must be entered to close the file.

INTERRUPT

Displays a table containing
the interrupt indicators

Command Format

INTERRUPT

Comments

The INTERRUPT command displays a table containing the Interrupt Enable (IE) and Interrupt Priority (IP) registers, including the status of Interrupts In Progress for priority 0 (IIP0) and priority 1 (IIP1). For IP, "0" and "1" indicate the priority levels with level 1 priority being higher than level 0. A "1" indicates an interrupt is enabled for IE, while a "1" indicates an interrupt in progress for both IIP0 and IIP1.

Examples

INTERRUPT

displays the interrupt table as described in the comments section.

LIST

Sends a copy of the debugging session to a disk file

Command Format

LIST <destination pn>

where

<destination pn> specifies the output file that receives a copy of the debugging session.

Comments

The <destination pn> is any valid ISIS-PDS device name followed by a filename as described in the iPDS User's Guide. The emulator software sends a copy of all input and output from the debugging session to the file.

The LIST command destroys any data that was initially in the file specified with the command. The command is terminated by entering the following command:

LIST :CO:

Examples

LIST :F2:TEST.V3

saves a copy of all input and output from the debugging session to the file TEST.V3 until the LIST :CO: command is entered.

LIST :LP:

saves a copy of all input and output from the debugging session to the line printer until the LIST :CO: command is entered.

LIST :CO:

terminates the LIST command and closes the <source pn>.

LOAD

Loads the user's object code and/or symbol table

Command Format

LOAD <source pn> [NOCODE] [NOSYMBOL]

where

- <source pn> specifies the input file that contains the object code and symbol table of the user's program.
- NOCODE specifies the user's symbol table is loaded from the input file, but not the object code.
- NOSYMBOL specifies the user's object code is loaded from the input file, but not the symbol table.

Comments

Depending on the options selected with the command, the LOAD command loads the user's object code program and symbol table from the input file. Unless the user specifies otherwise, the command defaults to loading object code and symbol table. The command operates on EMV program memory.

The LOAD command will load non-relocateable (absolute) code in the Hex or Object format. Thus, absolute assembler output or RL51 output can be loaded.

If the operation is not successful, an error message is displayed on the console.

Examples

```
LOAD :F1:EMVDEM.OBJ
```

loads the user's object code program and symbol table from the file :F1:EMVDEM.OBJ.

```
LOAD :F2:EMVDEM.OBJ NOCODE
```

loads the user's symbol table from the file :F2:EMVDEM.OBJ.

```
LOAD :F4:EMVDEM.OBJ NOSYMBOL
```

loads the user's object code program without the symbol table from the file :F4:EMVDEM.OBJ.

NOTE

The LOAD command does not check for duplicate symbols when loading the symbol table. If the user loads the same program twice, all symbol names will be duplicated in the user symbol table.

MACRO

Displays the text definition
of one or more macros

Command Format

MACRO [**;** <string1> [**:** <string2> ,...]]

where

<string> a predefined macro name the consists of 1 to 32 alphanumeric characters including the colon.

Comments

If **MACRO** command is entered by itself, the emulator software displays the definitions of all user defined macros. Otherwise, the command displays the definitions of all macros entered with the command.

Examples

MACRO

displays the definitions of all user defined macros.

MACRO :SUM, :DIV

displays the definitions of the macros **:SUM** and **:DIV**.

MEMORY

Displays format requirements
for display/modify commands

Command Format

MEMORY

Comments

The MEMORY command displays a table consisting of the display/modify commands (CBYTE, DBYTE, RBYTE, PBYTE, RBIT, CDUMP, and DDUMP) and their format requirements. The table is displayed on the screen until the user enters any command. When a command is entered, the emulator software displays the results of the command, but does not preserve the MEMORY table.

Examples

MEMORY

displays a table containing the display/modify commands.

M

displays a table containing the display/modify commands.

P

Displays the last two
instructions emulated

Command Format

P

Comments

The P command displays the previous two instructions emulated with either the GO or STEP commands. The command displays "NO DATA" if no instructions have been emulated or Warning D0 has occurred.

Examples

P

displays the last two 8051 instructions emulated with either the STEP or GO commands.

PBYTE

Display or modify contents
of external data memory

Command Format

PBYTE <expr1> [{ TO <expr2> }] [= { <expr3> } [, <expr>] [, <string>] ...]

where

- <expr1> specifies a beginning address within external data memory.
- TO <expr2> specifies an ending address within external data memory. The address must be between 0 and 64K.
- LENGTH <expr4> specifies a range of locations within external data memory to be displayed. The <expr4> indicates the number of locations to be displayed. It ranges between 0 and 64K.
- <expr3> specifies the 8-bit value that will be stored at the address specified in <expr1>. The expression consists of numerals, symbol names, and arithmetic operators.
- <string> specifies an alphanumeric string of characters that are stored in memory beginning at <expr1>. Each character in the string is stored in successive memory locations.

Comments

The PBYTE command displays or modifies the contents of external data memory, depending on how the command is entered.

If the TO or LENGTH parameters are entered, the command displays the memory contents at the indicated address. If the equal parameter (=) is entered, the command modifies the contents of memory to the value specified by the user. The optional TO, LENGTH, and equal sign can be combined within the command line to assign values to multiple memory locations.

Examples

PBYTE 0 TO 3FH

displays the contents of external data memory locations 0 to 3FH.

PBYTE 0 LEN 9 = 56H

changes the contents of external data memory locations 0 through 8 to the value 56H.

PBYTE 0 LEN 1K = 1,2,3,4

stores a repeating in external data memory beginning at location 0 and continuing for the next 1024 locations.

PC

Display or change the contents
of the program counter

Command Format

PC [= <expr>]

where

<expr> specifies a 16-bit memory address that replaces the current contents of the program counter. It consists of numeric values, symbolic references, and mathematical operators.

Comments

The program counter (PC) indicates the next instruction to be executed. The PC command displays or changes the contents of the register. When <expr> is used, it is evaluated to a 16-bit value.

Examples

PC

displays the current contents of the program counter. The numeric value is displayed according to the default base.

PC=100H

changes the contents of the program counter to the value 100H.

PSW

Display or change the contents of program status word register (PSW)

Command Format

PSW [= <expr>]

where

<expr> specifies the numeric value that replaces the current contents of the PSW. Consists of numeric values, symbolic references, and mathematical operators. The expression is evaluated to a 16-bit result and truncated to the lower 8 bits.

Comments

The program status word register (PSW) contains 8 flags that indicate the status of microcontroller. These flags are listed below corresponding to the bit position with the PSW.

- bit 7 Carry flag (CY)
- bit 6 Auxiliary carry flag (AC)
- bit 5 User flag 0 (FO)
- bit 4 Register bank select 1 (RBS1)
- bit 3 Register bank select 0 (RBS0)
- bit 2 Overflow flag (OV)
- bit 1 Reserved flag not accessible to the user (UTL)
- bit 0 Parity flag (P)

The PSW command displays or changes the contents of the PSW. When the contents are being changed, the user can enter the value as an 8-bit binary value, a byte value, or an expression.

Examples

PSW

displays the current contents of the 8051 PSW. The numeric value is displayed as a binary value.

PSW=1

sets the parity flag and clears the remaining flags.

PUT

Saves the definition of user macros in a file

Command Format

```
PUT <destination pn> {:<string> [,<string>,...] }
                    {MACRO}
```

where

- <destination pn> specifies the output file that receives a copy of the macro definition.
- <string> a pre-defined macro name, where the name consists of 2 to 32 alphanumeric characters including the colon.
- MACRO specifies all macros currently residing in RAM or workfiles.

Comments

The PUT command saves macro definitions on the <destination pn>. The user can specify from one to all macros currently residing in RAM or macro workfiles. The <destination pn> can be any valid ISIS-PDS device attached to the iPDS.

If the specified macro name does not exist, it is created by the PUT command.

Examples

```
PUT :F2:MAC.SAV :DIV,:SUM
```

saves the macros :DIV and :SUM on the file :F2:MAC.SAV.

```
PUT :F2:MAC.SAV MACRO
```

saves all macro definitions currently in RAM on the file :F2:MAC.SAV.

```
PUT :LP: MACRO
```

sends a copy of all macro definitions currently in RAM or in the macro workfiles to the line printer.

RBIT

Display or modify contents
of bit-addressable memory

Command Format

$$\text{RBYTE } \langle \text{expr1} \rangle \left[\left\{ \begin{array}{l} \text{TO } \langle \text{expr2} \rangle \\ \text{LEN } \langle \text{expr4} \rangle \end{array} \right\} \right] \left[= \left\{ \langle \text{expr3} \rangle \right\} \left[, \langle \text{expr} \rangle \right] \dots \right]$$

where

- <expr1>** specifies a beginning address in bit-addressable data memory that is between 0H and 0FFH.
- TO <expr2>** specifies an ending address in bit-addressable data memory. The address must be between 0H and 0FFH.
- LENGTH <expr4>** specifies a count of locations in bit-addressable data memory to be displayed. The <expr4> indicates the number of locations to be displayed. It ranges between 0 and FFH.
- <expr3>** specifies the 1-bit value that is stored at the specified address. The expression consists of numerals, symbol names, and arithmetic operators.
- <string>** specifies an alphanumeric string of characters. Each character's least significant bit is stored in successive memory locations.

Comments

The RBIT command displays or modifies the contents of bit-addressable data memory, depending on how the command is entered.

If the TO or LENGTH parameters are entered, the command displays the memory contents at the indicated address. If the equal parameter (=) is entered, the command modifies the contents of memory to the value specified by the user. The optional TO, LENGTH, and equal sign can be combined within the command line to assign values to multiple memory locations.

Examples

RBIT 20H to 27H

displays the contents of bit-addressable memory locations 20 to 27H.

RBIT D0H to D7H = 1

changes the contents of the program status word register (PSW) to the value FFH.

RBS

Display or change the contents
of register bank select flags

Command Format

RBS [= <expr>]

where

<expr> specifies the numeric value that replaces the current contents of register bank select flags in the program status register. The 2-bit value can be specified as a physical address, symbolic name, or an expression with a value between 0 and 3, inclusive.

Comments

The 8051 microcontroller contains four banks of 8 general purpose registers numbered R0 through R7. The RBS command specifies one of the four banks of general purpose registers that is the active register bank. When <expr> is used, it is evaluated to a 2-bit value between the limits of 0 to 3, inclusive.

Examples

RBS

displays the current contents of the RBS flags in the program status register. The numeric value is displayed according to the current base.

RBS=1

changes the contents of the RBS flags in the program status register to the value 01, indicating general purpose register bank 1.

RBYTE

Display or modify contents of
special function register memory

Command Format

RBYTE <expr1> [{ TO <expr2> }] [= { <expr3> } [, <expr>] [, <string>] ...]

where

<expr1> specifies a beginning address within register memory.

TO <expr2> specifies an ending address within register memory. The address must be between 80H and FFH.

- LENGTH <expr4>** specifies a range of locations within register memory to be displayed. The <expr4> indicates the number of locations to be displayed. It ranges between 0 and 7FH.
- <expr3>** specifies the 8-bit value that will be stored at the specified address. The value consists of numerals, symbol names, and arithmetic operators.
- <string>** specifies an alphanumeric string of characters that are stored in memory beginning at <expr1>. Each character in the string is stored in successive memory locations.

Comments

The RBYTE command displays or modifies the contents of register memory, depending on how the command is entered.

If the TO or LENGTH parameters are entered, the command displays the memory contents at the indicated address. If the equal parameter (=) is entered, the command modifies the contents of memory to the value specified by the user. The optional TO, LENGTH, and equal sign can be combined within the command line to assign values to multiple memory locations.

Examples

```
RBYTE 80H TO 83H
```

displays the contents of register memory locations 80 to 83H.

```
RBYTE 84H LEN 4 = 2
```

changes the contents of register memory locations 84H through 87H to the value 2.

REGISTER

Displays the 8051 registers
and their contents

Command Format

```
REGISTER
```

Comments

The REGISTER command displays a table consisting of the 8051 registers and their contents. The display remains on the screen until a non-register command is entered. When a register command is entered, the table is updated to reflect the new contents entered with the command.

Although the following command is not displayed with the REGISTER command, it can be entered while the table of registers is displayed:

```
B
```

Examples

```
REG
```

displays a table containing the 8051 microcontroller registers and their contents.

R<n>

Display or change the contents of the general purpose registers

Command Format

R<n> [= <expr>]

where

- <n> signifies a digit with the limits of 0 to 7 inclusive, that specifies one of the general purpose registers R0 through R7 within the register bank selected via RBS.
- <expr> specifies the numeric value that replaces the current contents of the general purpose register. The 8-bit value can be specified as a physical address, symbolic name, or an expression.

Comments

The 8051 microcontroller contains four banks of 8 general purpose registers numbered R0 through R7. Depending on the particular bank selected via the RBS command, the R<n> command displays or modifies the contents of register. When <expr> is used, it is evaluated to a 16-bit value and truncated to 8 bits.

Examples

R0

displays the current contents of general purpose register R0. The numeric value is displayed according to the default base.

R2=10H

changes the contents of general purpose register R2 to the value 10H.

REMOVE

Removes user-symbolic names
or macro definitions

Command Format

$$\text{REMOVE} \left\{ \begin{array}{l} \text{.<string> [,<string>, ...]} \\ \text{:<string> [,<string>, ...]} \\ \text{SYMBOLS} \\ \text{MACRO} \end{array} \right\}$$

where

- <string>** name of user-defined symbol(s) in the symbol table that are 2 to 32 characters long including the period.
- <string>** name of user-defined macro(s) in RAM or the workfiles that are 2 to 32 characters long including the colon.
- SYMBOLS** specifies that all user symbolic names are removed from the user symbol table.
- MACRO** specifies that all macro definitions are removed from RAM or the workfiles.

Comments

The REMOVE command deletes from one to all user symbolic names from the symbol table, or one to all macro definitions from RAM or the workfiles.

Examples

```
REMOVE .ABLE,.BAKER
```

removes the user symbolic names .ABLE and .BAKER.

```
REMOVE SYMBOLS
```

removes all user assigned symbolic names.

```
REMOVE :DIV
```

removes the macro definition :DIV.

```
REMOVE MACRO
```

removes all user macro definitions from memory.

REPEAT

Begins command loop

Command Format

```

REPEAT <cr>
  [ WHILE <condition> <cr>
    UNTIL <condition> <cr>
    <command> <cr> ] ...
END

```

where

WHILE indicates the <command list> executes as long as the <condition> entered with the WHILE command is true.

UNTIL indicates the <command list> executes as long as the <condition> entered with the UNTIL command is false.

<condition> indicates the test condition that satisfies the WHILE/UNTIL commands. The test condition consist of an arithmetic, logical, or relational test, such as:

```
DBYTE 0 < .TOTAL
```

<command> is a sequence of EMV-51 commands entered by the user that executes until the test condition is met.

Comments

The REPEAT..END command sequence sets up a command loop that executes until optional specific test conditions are met. The ESCAPE key can be used to exit from REPEAT loops

The REPEAT..END command sequence with the WHILE clause, executes a series of EMV-51 commands as long a test condition is true. The REPEAT..END command sequence with the UNTIL clause executes a series of EMV-51 commands as long as a test condition is false. Both command loops permit the user's sequence of commands to control the number of times the command loop executes. The REPEAT..WHILE and REPEAT..UNTIL command sequence can appear anywhere in the loop.

Examples

```

REPEAT
  WHILE DBYTE 0 < .TOTAL
  ASM MOV.P0,#(DBY 0)
  DBY 0=(DBY 0+1)
END

```

the commands execute while the contents of DBYTE 0 equals or exceeds the value assigned to the symbolic name .TOTAL.

```
REPEAT
  UNTIL DBYTE 0 = .TOTAL
  ASM MOV.P0,#(DBY 0)
  DBY 0=(DBY 0+1)
END
```

the commands execute until the contents of DBYTE 0 equals the value assigned to the symbolic name .TOTAL.

RESET

Resets emulator to its default settings

Command Format

RESET

Comments

The RESET command simulates a hardware RESET. Refer to the MCS-51 User's Manual for additional information on the hardware reset.

The command sets the DTRACE and BREAK commands to their default states. The RESET command does not affect the SUFFIX and BASE commands, or the function key assignments.

Examples

RESET

re-initializes the EMV-51 hardware and sets the DTRACE and BREAK commands to their default states.

SAVE

Saves the user's program in a file

Command Format

```
SAVE <destination pn> [ { <expr1> TO <expr2> } ] [NOCODE] [NOSYMBOL]
```

where

<destination pn>	specifies the output file that receives a copy of the user's program.
<expr1> TO <expr2>	indicates a specific range of the user's program that is saved in the output file.
NOCODE	specifies that the user's program code is not saved.
NOSYMBOL	specifies that the user's symbol table is not saved.

Comments

Depending on the options selected with the command, the SAVE command puts a copy of the user's program and symbol table in the output file. The command options default to saving the program code and symbol table.

If a program range is not specified, the emulator software saves the entire 4K of EMV-51 program memory as the user's program.

If the operation is not successful, an error message is displayed on the console.

Examples

```
SAVE :F1:EMVDEM.SAV
```

saves a copy of the user's program and symbol table in filename EMVDEM.SAV on drive :F1:

```
SAVE :F3:EMVDEM.S3 100H TO 1FFH
```

saves the range of addresses 100H to 1FFH of the user's program and the symbol table in filename EMVDEM.S3 on drive :F3:

```
SAVE :F2:EMVDEM.S2 NOCODE
```

saves the user's program symbol table in filename EMVDEM.S2 on drive :F2:

```
SAVE :F4:EMVDEM.S4 NOSYMBOL
```

saves the user's program without the symbol table in filename EMVDEM.S4 on drive :F4:

SP

Display or change the contents
of the stack pointer

Command Format

SP [= <expr>]

where

<expr> specifies the memory location that replaces the current contents of the stack pointer. The location can be specified as a physical address, symbolic name, or an expression that is evaluated to a 8-bit result.

Comments

The stack pointer (SP) indicates the top of the stack. The SP command displays or changes the contents of the register. When <expr> is used, it is evaluated to a 8-bit value.

Examples

SP

displays the current contents of the stack pointer register. The numeric value is displayed according to the default base.

SP=70H

changes the contents of the stack pointer register to the value 70H.

STEP

Begins step emulation
of user's program

Command Format

```
STEP [[FROM] <expr>] [ {COUNT = <expr1> }  
BR }
```

where

[[FROM] <expr>] specifies an optional starting address within the user's program. The <expr> consists of numeric values, symbolic names, and mathematical operators that are evaluated to a 16-bit result.

COUNT = <expr1> specifies an optional number of instructions to be emulated.

<expr1> a numeric value between 0 and FFFFH inclusive.

BR enables software breakpoint checking.

Comments

The STEP command emulates the user's program in a slow-down mode. During STEP emulation, the EMV-51 software emulates one instruction and then checks for the following items:

1. Trace display commands enabled. If trace display commands are set, the emulator software displays from one to four lines of trace information about the instruction emulated.
2. If BR is specified when the user enters the STEP command, the emulator software performs software breakpoint checking. If a breakpoint is encountered, the software stops the emulation and returns control to the user.

NOTE

If a trace display command (TRn or TV) has turned off the software trace display, it will remain off permanently until another trace display command which is set to turn on the software trace display is executed, or until EMV-51 software is reset.

Emulation begins with the instruction pointed to by the program counter or the address specified in the STEP command (the [FROM] <expr> option).

The COUNT = <n> option permits a range of instructions to be executed. If the COUNT = <n> option and the BR option are both specified, the BR option takes precedence over the COUNT option.

Examples

STEP

emulates one instruction of the user's program from the address contained in the program counter. The emulator software checks for trace display commands after each emulated instruction.

STEP FROM 100 BR

begins emulation of the user's program from address 100H and enables software breakpoint checking. The emulator software checks for trace display commands after emulating each instruction and then checks for breakpoints.

STEP FROM 100H COUNT=10

emulates 10 instructions of the user's program, checking for trace display commands after each emulated instruction. The emulator software does not check for breakpoints.

SUFFIX

Displays or changes the default base for input numeric data

Command Format

$$\text{SUFFIX} \left[= \left\{ \begin{array}{c} Y \\ H \\ T \\ Q \end{array} \right\} \right]$$

where

- Y specifies binary as the default suffix for input numeric data.
- H specifies hexadecimal as the default suffix for input numeric data.
- T specifies decimal as the default suffix for input numeric data.
- Q specifies octal as the default suffix for input numeric data.

Comments

The SUFFIX command displays or changes the default suffix for numeric values entered by the user. The default suffix is overridden when the user enters an explicit suffix with a numeric value. The initial default suffix is hexadecimal (H).

Examples

SUFFIX

displays the default suffix for numeric values entered by the user.

SUF=T

changes the default suffix to decimal. Unless the user enters an explicit suffix with a numeric value, all numeric values that are input are evaluated as base ten.

SYMBOLS

Displays all user-assigned symbolic names and values

Command Format

SYMBOLS

Comments

The SYMBOLS command displays all user-assigned symbolic names and their values. The command does not display system symbolic names.

Examples

SYMBOLS

displays all user-assigned symbolic names and their values.

SYM

displays all user-assigned symbolic names and their values.

TMO

Display or change the contents of timer/counter register 0

Command Format

TMO [= <expr>]

where

<expr> specifies the 16-bit numeric value that replaces the current contents of timer/counter register 0.

Comments

The timer/counter register 0 (TMO) is in operations requiring timing operations or event counters. The TMO command displays or changes the contents of the register. When <expr> is used, it is evaluated to a 16-bit value.

Examples

TMO

displays the current contents of timer/counter register 0. The numeric value is displayed according to the default base.

TMO=100H

changes the contents of timer/counter register 0 to the value 100H.

TM1

Display or change the contents
of timer/counter register 1

Command Format

TM1 [= <expr>]

where

<expr> specifies the 16-bit numeric value that replaces the current contents of timer/counter register 1.

Comments

The timer/counter register 1 (TM1) is in operations requiring timing operations or event counters. The TM1 command displays or changes the contents of the register. When <expr> is used, it is evaluated to a 16-bit value.

Examples

TM1

displays the current contents of timer/counter register 1. The numeric value is displayed according to the default base.

TM1 = 100H

changes the contents of timer/counter register 1 to the value 100H.

WRITE

Evaluates an expression and displays the results on the screen

Command Format

```
WRITE {<string> } [,<string> ]...
      {<expr> }
```

where

'<string>' a sequence of alphanumeric characters.

<expr> specifies an arithmetic expression or numeric value. If an expression is used, it is evaluated and replaced by a 16-bit result.

Comments

The WRITE command is used primarily inside of macro definitions to display messages on the screen. The ,<string> consists of a message that is displayed when the WRITE command is executed.

Example

```
WRITE 'BOARD'S BAD'
```

displays the message BOARD'S BAD when the RETURN key is pressed. The double apostrophe is displayed as an apostrophe.

(macro execution)

Executes macro definitions

Command Format

```
:<string> [<parameter1 >, <parameter2>, ...]
```

where

:<string> a pre-defined macro name, where the name consists of 2 to 32 alphanumeric characters including the colon.

<parameter> specifies any parameter values required within the macro definition.

Comments

The user executes macro definitions by preceding the macro name with the colon (:).

If macro expansion is enabled with the ENABLE EXPANSION command, the macro definition is listed prior to execution of the macro.

If the macro definition requires parameters, the parameters are entered with the macro name. The parameters are entered as:

<parameter 1 >, <parameter 2 >, ...

and sequentially replace the parameter place holder within the definition. Refer to Chapter 6 for additional details on using parameters within macros.

Examples

:SUM

executes the macro :SUM

:DEMO 123,12,.ABLE

executes the macro :DEMO after substituting the parameters 123, 12, and .ABLE into the macro definition.

(symbol handling)

Displays both system and user symbol names or modifies user symbol names

Command Format

.<string> [= <expr>]

where

.<string> a pre-defined symbolic name consisting of 2 to 32 alphanumeric characters including the period.

<expr> specifies the value to be assigned to the symbolic name. It consists of numeric values, symbol names, and arithmetic operators and is evaluated to a 16-bit result.

Comments

The user generates the value assigned to system symbolic names or user symbolic names by preceding the name with a period. The emulator software responds by substituting the value assigned to the name for the name itself.

If the symbolic name is a user symbolic name, the user changes the value assigned to the name with the = <expr> command modifier.

Examples

.ABLE

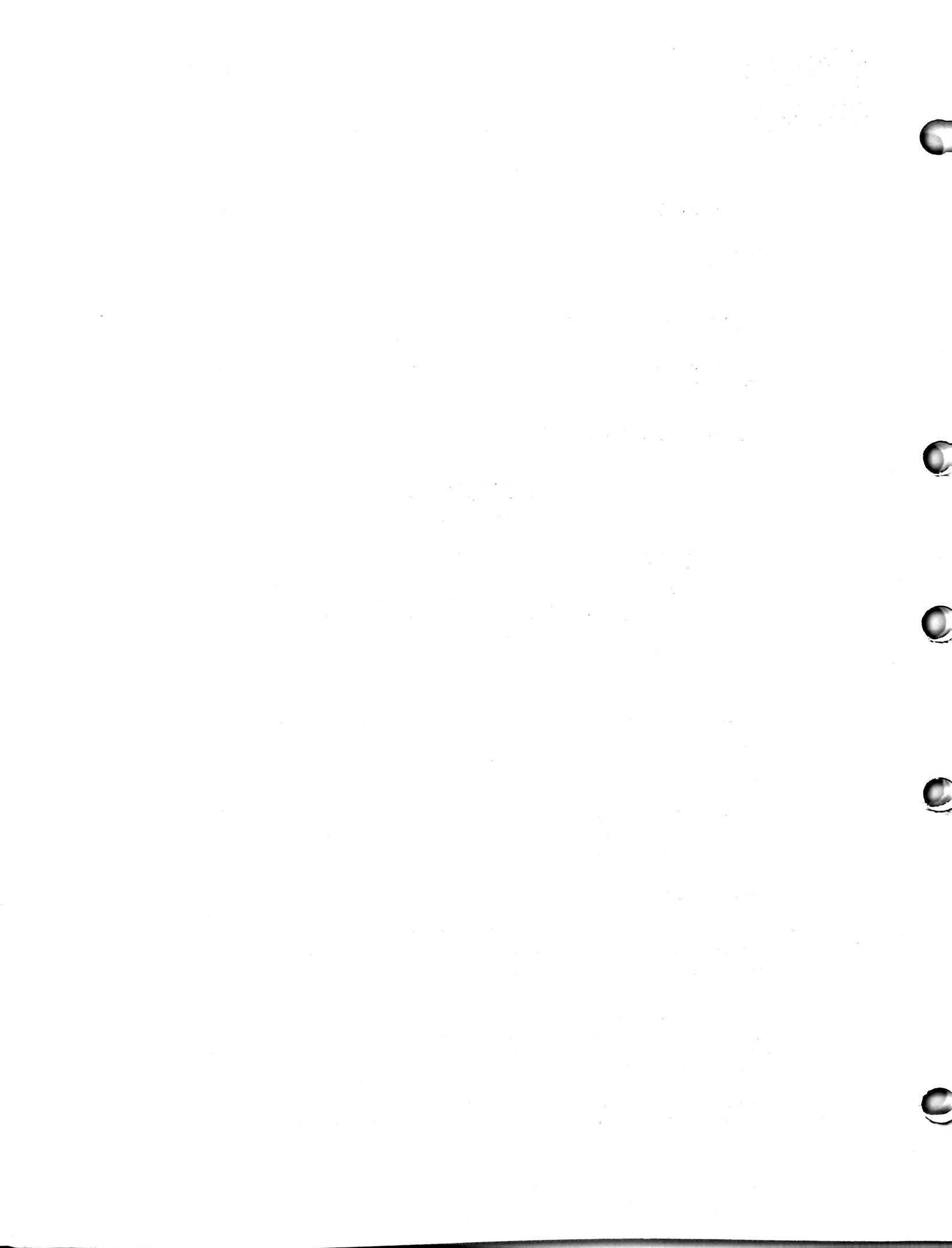
displays the value assigned to the symbolic name .ABLE

.CY

displays the current contents of the system carry flag

.ABLE=123H

changes the contents of user symbolic name .ABLE to the value 123H.





Introduction

This appendix provides the installation instructions and the hardware specifications for the EMV-51 Emulation Vehicle.

Installing the EMV-51

This section provides installation considerations and installation procedures for the EMV-51.

Installation Considerations

CAUTION

The emulation processor on the user plug is an MOS integrated circuit. MOS devices are easily damaged by static electricity, even when the system is powered down. Avoid touching the pins or exposed portions of the chip. Do not allow objects to come in contact with the chip. Handle the user plug by the edges. If downward pressure is required when installing in the user socket, push down only on the metal cover.

To minimize the system's sensitivity to static electricity, i.e., electrostatic discharge (ESD):

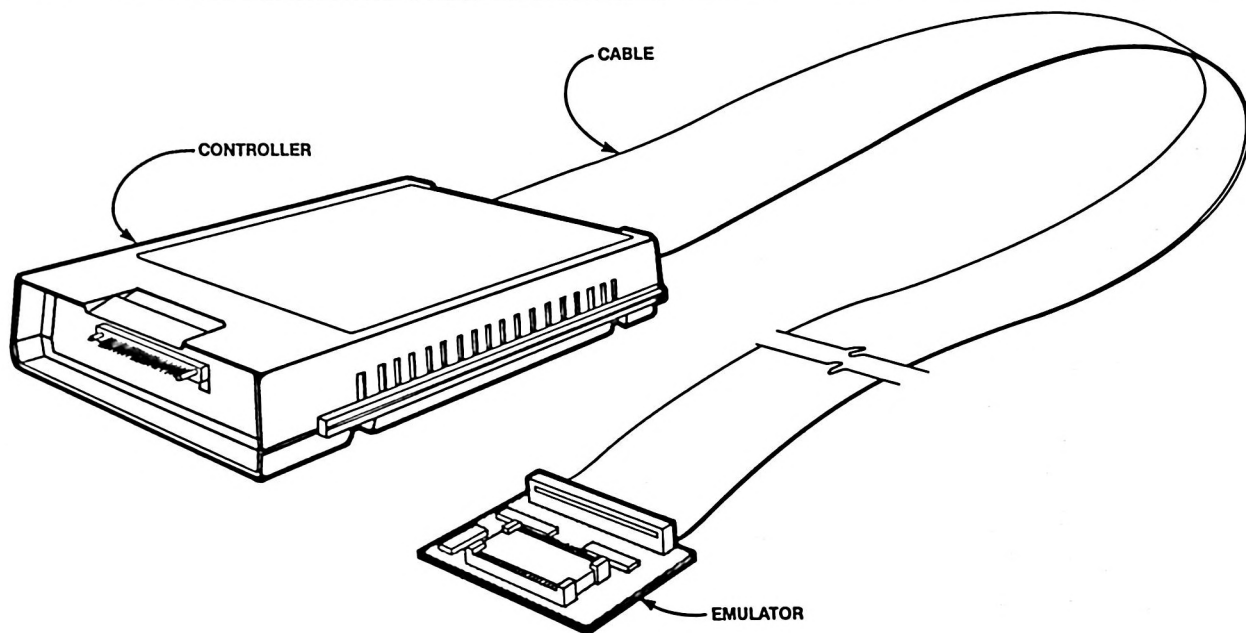
- a. Maintain a relatively high humidity (> 60%).
- b. Use antistatic mats in the work area.

Installation Procedures

Prior to installing EMV-51 into the iPDS, the iPDS EMV PROM ADAPTER must be installed in the iPDS. Refer to the section labeled PLUG-IN MODULE ADAPTER in Appendix A in the *iPDS User's Guide* for the installation instructions.

The following step-by-step procedure is for the initial installation of the EMV-51. The selection of optional jumper configurations is covered in later sections. The emulator hardware setup is described first, followed by the emulator software setup.

Figure A-1 shows the EMV-51 emulator hardware. The emulator hardware consists of the controller, an 80-inch cable, and the emulator module.

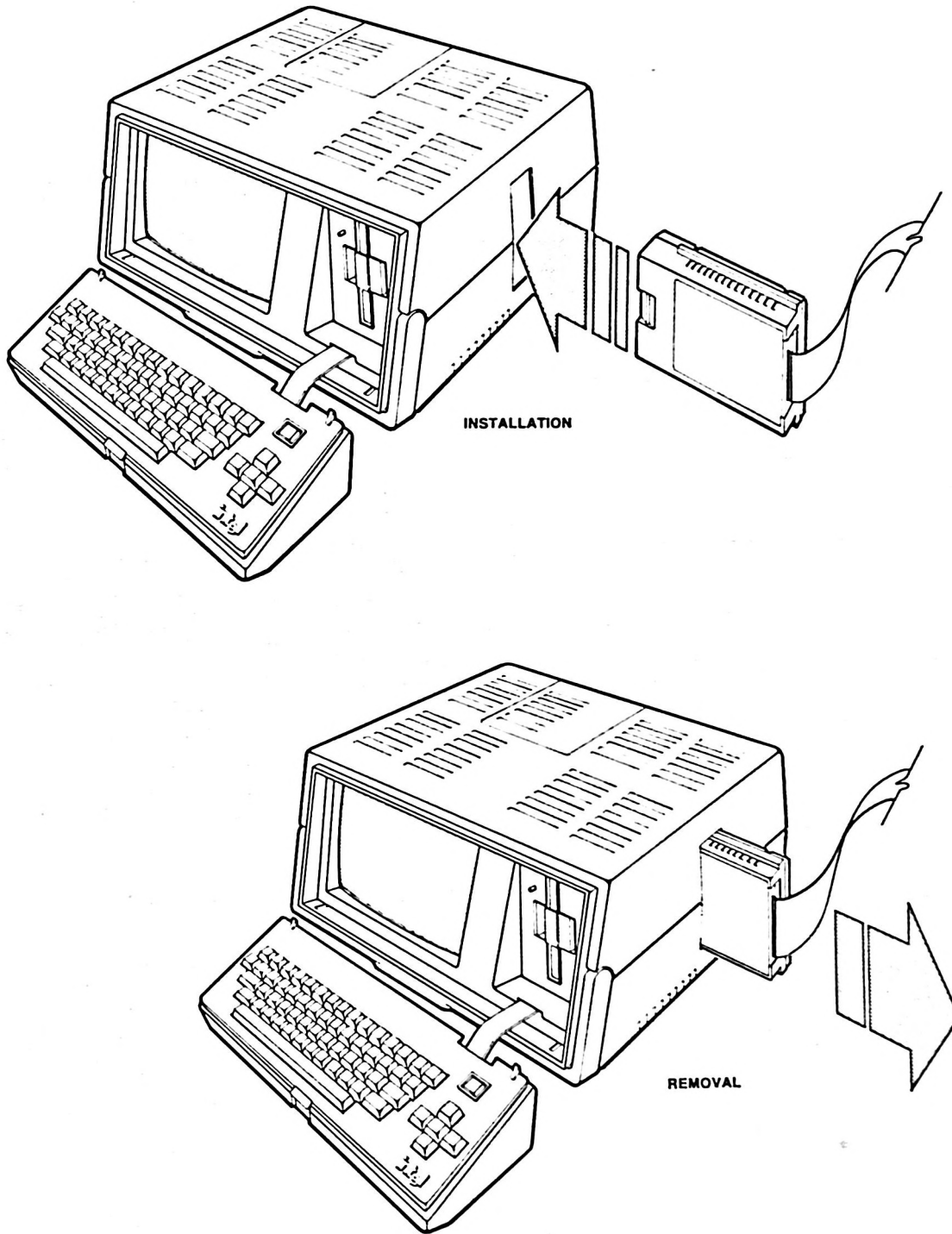


0250

Figure A-1 EMV-51 Emulator Hardware Components

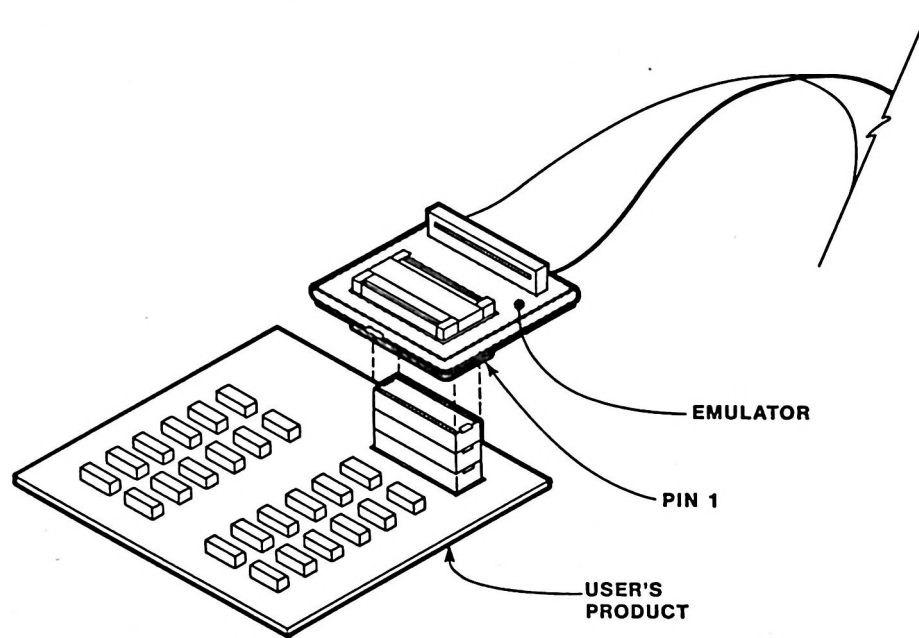
Once the emulator hardware is removed from the shipping carton, proceed as follows:

1. Installing the Controller:
 - a. The Controller is inserted and removed from the right side of the Intel Personal Development System. Figure A-2 shows insertion and removal of the Controller.
 - b. The Controller module has plastic ribs along the top and bottom of the module, permitting insertion in one direction only. Insert the module in the iPDS until it seats firmly against the connector.
2. Installing the Emulator Module
 - a. The emulator plugs into the socket normally occupied by the 8051 microcontroller in the user's product. The EMV-51 can operate without connecting the emulator to the user's product. Figure A-3 shows the emulator connected to the user's prototype in place of the prototype's microcontroller.
 - b. If the prototype hardware is to be used, remove the socket protector from the emulator and attach the emulator to the prototype via the 40-pin socket. If necessary, insert 40-pin sockets between the emulator and the prototype to clear other components. Otherwise leave the socket protector attached to the emulator.
3. Installing the EMV-51 Software
 - a. As shipped, the EMV-51 software resides on a diskette. If the user has a single disk drive iPDS, the emulator software must be copied onto the system software diskette using the ISIS-PDS COPY command. Refer to the *iPDS User's Guide* for information on using the COPY command.



0251

Figure A-2 Installation in the Personal Development System



0252

Figure A-3 Installation of Emulator Module in Prototype

- b. Insert the system diskette into drive 0. If the user has a single-drive iPDS, the system diskette must contain the emulator software. If the user has a multiple-drive iPDS, insert the EMV-51 diskette in disk drive :F1:, :F2:, or :F3:.
- c. If the iPDS has not been initialized, press RESET to initialize the iPDS system software and generate the ISIS-PDS sign-on message.

When the emulator software is loaded and executed, it selects the disk drive that the software is loaded from as the default drive for the macro workfile. The workfile is a temporary storage area on disk that holds macro definitions. The emulator software always assigns the name MAC.TMA to the workfile.

As shown below, the user can specify another disk drive as the workfile drive when the emulator software is loaded.

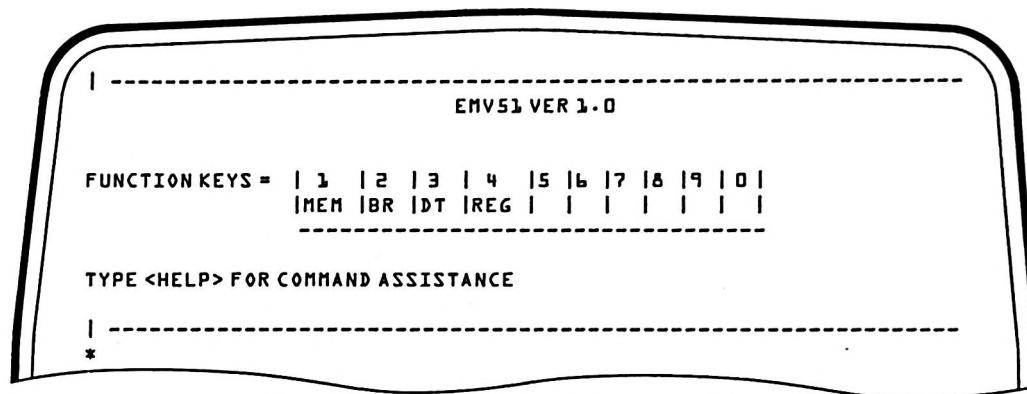
- d. If the user has a single-drive iPDS and the emulator software resides on the system diskette, the following command is entered to load the emulator software:


EMV51 Drive :F0: contains the workfile

If the user has a multiple-drive iPDS and inserted the emulator software diskette in a drive :F2:, enter the following command:

:F2:EMV51 WORKFILE (:F1:) Specifies that drive :F1: contains the workfile, while the software is loaded from drive :F2:.

After entering the file name EMV51, the ISIS-PDS operating system loads the emulator software and EMV51 responds with a sign-on message. The following screen display demonstrates loading the emulator software and the resulting sign-on message.



Key-in Sequence	Comments
EMV51 	Enters EMV51 invocation. The ISIS-PDS operating system loads and executes the EMV-51 software. The software displays the sign-on message and asterisk prompt shown in this display.

Jumper Configurations

The EMV-51 Emulator board contains five jumpers that select the supply voltage source, the clock source, the reset source, and internal or external program memory. These jumpers are listed in table A-1.

Table A-1. EMV-51 Emulator Jumper Descriptions

Jumper Configuration	Description
E1-E2	EA/pin strapped for standalone mode.
E1-E2	EA/ pin strapped for standalone mode.
E2-E3	EA/ state selected by user's prototype
E4-E5	Emulator powered by EMV-51 controller
E5-E6	Emulator powered by user's prototype
E7-E8	Reset provided by EMV-51
E8-E9	Reset provided by user's prototype
E11-E12	Crystal provided by EMV-51
E14-E15	Crystal provided by user's prototype
E11-E10	Crystal provided by user's prototype
E13-E14	Crystal provided by user's prototype

The following paragraphs describe the various configurations for the four jumpers. Figure A-4 shows the jumper locations on the EMV-51 emulator board.

Selecting the Program Memory

Use jumper configuration E1-E2 for standalone operation. Use jumper E2-E3 when the user's prototype selects the state of the EA/ pin. To use EMV-51 supplied code memory to debug code in a prototype with EA/ strapped low, use jumper E1-E2.

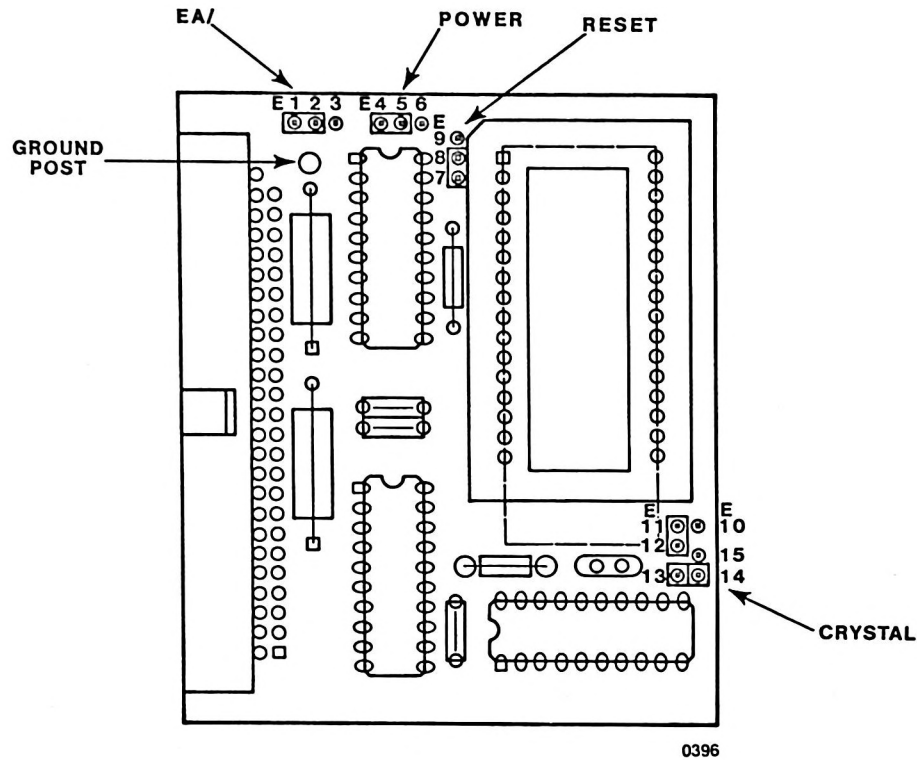


Figure A-4 EMV-51 Emulator Jumper Locations

Selecting Supply Voltage

The jumper is moved to the E5-E6 position if supply voltage is provided by the user's prototype. If the EMV-51 controller is providing the supply voltage, the jumper is moved to the E4-E5 position.

Selecting Oscillator Input

The jumpers are moved to the E10-E11, E13-E14 positions if the clock is supplied by the prototype. When the jumpers are moved to the E11-E12, E14-E15 position, the clock is supplied by the emulator's on-board crystal.

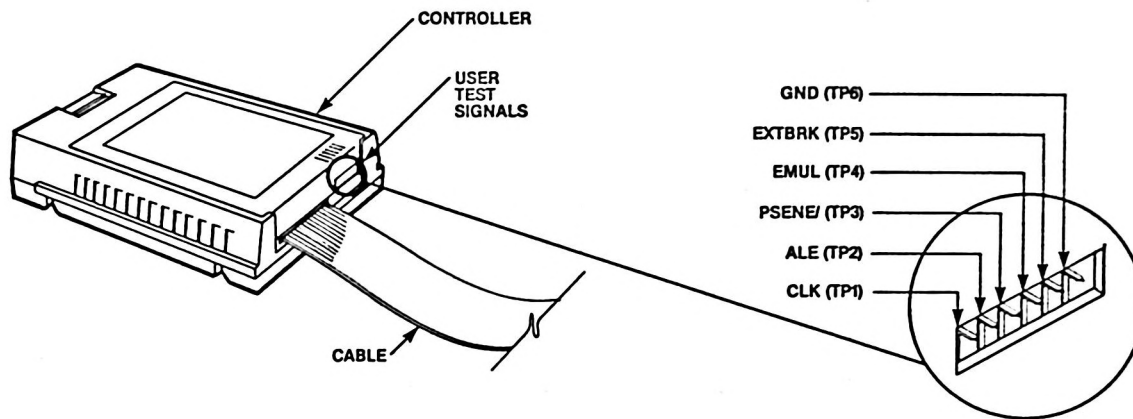
Selecting Reset Input

The jumper is moved to the E7-E8 position if the reset capacitor on the emulator is to be connected to the 8051.

If a user's prototype is to supply the reset signal to the 8051, the jumper is moved to the E8-E9 position.

User Test Signals

The Emulator module has 6 terminal posts that supply signals to the user's test equipment. These signals indicate the internal state of various operations in the 8051 microcontroller. Table A-2 lists the signals supplied to the user and gives a brief description of each signal, while Figure A-5 shows the location of the terminal posts.



0414

Figure A-5 EMV-51 Controller's Terminal Post Location

The CLK signal indicates the first fetch of the first byte of an instruction from program memory (either the prototype's program memory or the EMV-51's 4K RAM).

The falling edge of the ALE signal indicates that the 8051 microcontroller's address bus contains a valid address. The signal is high at the start of every bus cycle.

The PSENE/ signal strobes data between the EMV-51 controller and the EMV-51 Emulator module. When the signal is low, data passes from the Controller to the Emulator module.

The EMUL signal is high when the EMV-51 is emulating the user's program. The signal is high for one instruction prior to the actual start of program execution and remains high until the instruction that follows a breakpoint is fetched.

The rising edge of the EXTBRK signal, supplied by the user, stops the emulation currently in progress. The signal is terminated in a 220/330 ohm resistor network.

If the user connects a logic analyzer to EMV-51, the EMUL signal is normally connected as a triggering input to the logic analyzer. The logic analyzer feeds a signal to the EXTBRK input to halt EMV-51 during an emulation.

Table A-2. User Test Signals

Test Signal	Description
CLK	Instruction Fetch Clock
ALE	Address latch Enable
PSENE/	Program Store Enable
EMUL	Emulation
EXTBRK	External Break Input
GND	Signal Ground

Port 0 in Stand-Alone Mode

To be able to read and write the 8051 port 0 in a stand-alone mode, the port must be pulled up by a resistor network, one resistor for each line on port 0. Figure A-6 shows schematically the necessary connections.

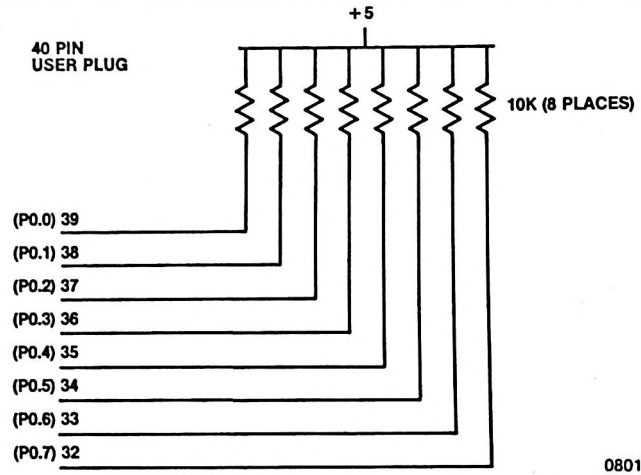


Figure A-6 Port 0 Pullup Configuration



APPENDIX B ERROR MESSAGES

This appendix contains a list of the error and warning messages produced by the emulator software.

ERR 22:COPROCESSOR HAS FILE

An attempt was made to open or write to a file currently being written by the other processor.

ERR 24:TOO MANY BREAKS

Too many breakpoints are set for this command to operate. Delete some of the breakpoints and re-enter the command.

ERR 25:RELOCATABLE FILE

The command attempted to load a relocatable object file. Use the Assembler or Linker to convert the file to absolute format, then repeat the command.

ERR 26:UNRESOLVED EXTERNALS

The command attempted to load a file with unresolved symbols. Use the Assembler or Linker to resolve the external symbol or name, then repeat the command.

ERR 52:UNWRITEABLE MEMORY

The last command attempted to write to external memory. Check command to ensure writing hasn't been attempted above FFH.

ERR 80:SYNTAX ERROR

The command is ignored. Check command syntax and re-enter the command.

ERR 81:INVALID TOKEN

A token in the previous line is not properly formed, or is not appropriate for the command context; for example, the entry CBYTE 300 when SUFFIX =Y (binary).

ERR 83:INAPPROPRIATE NUMBER

Too many values were used for a specified range.

ERR 85:ITEM ALREADY EXISTS

The previous DEFINE command referred either to a symbol already in the user's or system symbol table, or to a macro already in the macro definition table.

ERR 86:ITEM DOES NOT EXIST

The symbol or macro reference in the previous command is not defined.

1. For a symbol, check version of code loaded, or define the symbol in the current session.
2. For a macro, define or **INCLUDE** the macro definition in the current session.

ERR 88:MACRO PARAMETER ERROR

A macro call contained more than ten actual parameters, or a parameter contained too many characters. Check the definition of the macro.

ERR 89:MISSING CR-LF IN FILE

The current **INCLUDE** file does not end with a carriage return/line feed. Exit and use the editor to correct the file.

ERR 8F:NON-NUL STRING NEEDED

A null string (apostrophes with no enclosed characters) was used where at least one character is required, such as **DBYTE 1 = ''**.

ERR 90:MEMORY OVERFLOW

The emulator workspace exceeded the amount allocated to it. The workspace contains the user symbol table and space for expansion of macros prior to their execution. The command that produced the overflow is aborted, but the memory already written remains as written. To reclaim workspace, remove some user symbols.

ERR 92:COMMAND TOO LONG

The command exceeds the capacity of the emulator's command buffer. Possibly caused by too many operators. Break the command or expression into several smaller units.

ERR 95:INVALID OBJECT FILE

The object file referenced in a **LOAD** command is not written in the proper format. Perhaps it is a text file rather than an object file.

1. Select another file for loading.
2. **EXIT** and verify file type.

ERR 99:EXCESSIVE ITERATED DATA

The number of data items to be repeated in memory exceeds the buffer size for iterated data (128 bytes).

Example: **CBYTE 30 TO 1000H = RBYTE 0 TO 256T**

ERR 9D:LINE TOO LONG

The input line exceeds 120 characters.

1. Use continuation lines (ampersand at end of line to identify intermediate carriage return) to divide the command line into several input lines.
2. Break command into two or more shorter command lines.

ERR A4:MACRO FILE FULL

The temporary file MAC.TMA has used all the available space on the diskette specified at invocation as the WORKFILE diskette. Save and remove macro definitions to make room for more, using the PUT and REMOVE MACRO commands.

ERR B3:OFFSET IS TOO LARGE

This error occurs when the an expression specified within an assembly mnemonic instruction results in a relative offset that is larger than 8 bits.

ERR B9:NO HELP AVAILABLE

There is no help message for the item requested.

ERR BC:SYSTEM SYMBOL ERROR

Illegal operation on system type symbols, such as attempting to change the value of a system symbol or using a multiple reference involving system symbols.

ERR E7:ILLEGAL FILENAME

The command specified a filename that does not conform to ISIS-PDS format specifications. Check the ISIS-PDS User's Guide for proper filename format.

ERR E8:ILLEGAL DEVICE

The command contained a reference to an ISIS-PDS device, but the reference is illegal or unrecognizable. Check ISIS-PDS User's Guide for proper device names.

ERR F0:NO SUCH FILE

Command specified file that does not exist on the designated diskette.

1. Verify the drive number and filename.
2. Check to see that the correct diskette is inserted in the drive.

ERR F1:WRITE-PROTECTED FILE

The command attempted to open a write-protected file for write access.

1. Select another file.
2. EXIT and remove write protection from the target file (refer to the ISIS-PDS manual for details).

ERR F3:CHECKSUM ERROR

A checksum error in an object file was encountered during loading.

1. Select another file.
2. EXIT and create a correct object file.

ERR F6:DISKETTE FILE REQUIRED

The command specified a device other than a diskette file, but the operation requires a diskette file.

ERR F9:ILLEGAL ACCESS

Command attempted to open a read-only device for write access (e.g., :CI: as a LIST device), or attempted to open a write-only device for read access (e.g., :LP: in LOAD command). Check command syntax for list of appropriate devices.

ERR FA:NO FILE NAME

The command references a diskette device, but omitted the filename; for example: LOAD :F1:

ERR FF:NULL FILE EXTENSION

The command referenced a file terminated by a period, but the implied extension is missing. Omit the period or include the extension (refer to the *iPDS User's Guide* for details on filenames and extensions).

WARNING D0

The instruction at the breakpoint was fetched but not executed because an interrupt occurred at or before the instruction. The breakpoint instruction is the first to execute after the interrupt service routine.

WARNING D1

A breakpoint occurs on the first instruction of an interrupt service routine. The break instruction executed normally, but the previous instruction did not execute. The interrupted instruction is the first instruction to execute when emulation resumes. Check for a breakpoint set at the address of the first instruction of the interrupt service routine.

WARN D2

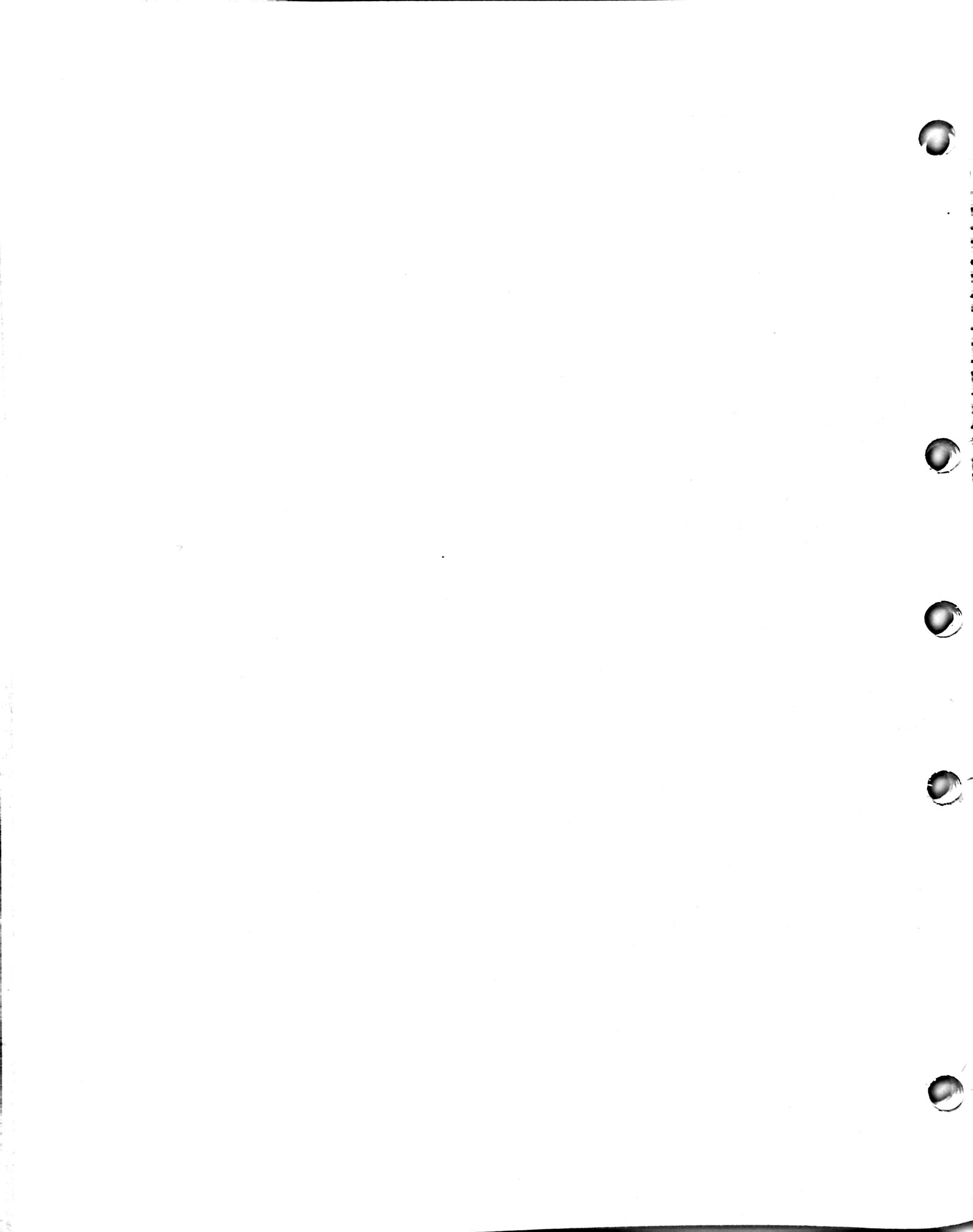
This warning is displayed when beginning GO or STEP emulation and a timer is enabled with an initial value of 0 or 1. This condition generates a spurious timer overflow and forces an interrupt if interrupts are enabled.

WARN CB:TRUNCATED TO 8 BITS

This error occurs when an opcode, value, or an expression in a mnemonic instruction to be assembled is greater than 8 bits. The value is truncated to the low 8 bits, the warning is issued, and processing continues.

WARN CD:TRUNCATED TO 11 BITS

This error occurs when an expression value in the ACALL or AJMP instruction in assembly is larger than 11 bits. The value is truncated to 11 bits, the warning error message is issued, and processing continues.





Confidence Test

This section describes the operation of the confidence test for the EMV-51. It is not necessary to run this test prior to using the system. However, it is recommended that the test be run when the emulator is initially installed.

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 further assumes that the EMV-51 is installed. The test runs as a utility under the ISIS-PDS operating system and provides a set of subcommands that help verify the emulator.

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 E51CON command.
3. Enter any of the nine confidence test commands to perform the 15 different tests.

These steps are described in detail in the following sections. Notational conventions used throughout for command descriptions are introduced and explained in Chapter 8.

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 E51CON command.

INIT E51CON Command

The INIT E51CON command produces the following display:

```
*INIT E51CON
EMV-51 CONFIDENCE TESTS, Vx.y
USER RETURN
```

After the INIT E51CON 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 15 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 15 confidence tests are described under the TEST command.

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 15 confidence tests by number (00H through 0EH). 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 15 confidence tests by number (00H through 0EH). 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 BREAK REGISTER MEMORY TEST      **** IGNORED ****
00004H BREAK/COMMAND MULTIPLEXER TEST
00005H RUNNING/INTD STATUS TEST
00006H BREAK SYNCHRONIZATION TEST
```

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 15 confidence tests by number (00H through 0EH). Enter the hexadecimal value of the test number.

The IGNORE command allows the user to declare the specified test or tests to be ignored and not run.

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 <destination>
```

where

<destination> specifies a valid ISIS-PDS device to receive any output from the tests. See Chapter 8 for an explanation of destination.

The LIST command causes a copy of all subsequent output from the tests to be sent to <destination>. The output includes prompts, user entered input, and error messages. The output is also displayed on the display screen. If the <destination> is :CO:, there is no effect, since the console receives all the output anyway.

To copy all output on 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 15 confidence tests by number (00H through 0EH). 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.

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:

```
<test number> [, <test number>] ..  
SUMMARY <test number> TO <test number> [EO]
```

where

<test number> specifies one of the 15 confidence tests by number (00H through 0EH). 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
- 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 E51CON 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 EXECUTE 00002H TIMES,
                          00000H FAILURES
00004H BREAK/COMMAND    TEST EXECUTE 00002H TIMES,
                          00001H FAILURES
00005H RUNNING/INTD     TEST EXECUTE 00002H TIMES,
                          00000H FAILURES
```

Note that all test numbers and error counts are given in hexadecimal.

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 BREAK/COMMAND TEST 00002H TIMES, 00001H FAILURES
```

TEST Command

The format of the TEST command is:

```
TEST <test number> [, <test number>]... ON ERROR
ON NOERROR
<test number> TO <test number> COUNT <nnnn>
FOREVER
```

where

<test number> specifies one of the 15 confidence tests by number (00H through 0EH). Enter the hexadecimal value of the test number.

- <nnnn>** specifies the number of times the specified test or tests are to be run. The value can be given in hexadecimal (H), decimal (T), octal (Q), or binary (Y) by attaching the appropriate suffix to the value. The default base is hexadecimal. The maximum value of **<nnnn>** is 65,535 in decimal.
- ON ERROR** specifies that the tests will execute repeatedly if one or more errors are detected.
- ON NOERROR** specifies that the tests will execute repeatedly if no errors are detected.
- COUNT <nnnn>** specifies that the tests will execute in numerical order for **<nnnn>** times. If **<nnnn> = 0** or if **COUNT** is specified with no **<nnnn>**, no tests will execute.
- FOREVER** specifies that the tests will execute in numerical order and repeat until the user presses the ESC key.

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, enter the command:

TEST

To run tests 0 through 0EH, enter the command:

TEST 0 TO 0E

Note that the value 0E is a hexadecimal value.

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 if an error is detected, enter the command:

TEST 0 TO 8 ON ERROR

To run test 7 and repeat if no 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	Function
0	Program Memory Test
1	Command Memory Test
2	Break Register Memory Test
3	Break/Command Multiplexer Test
4	RUNNING/INTD Status Test
5	Break Synchronization Test
6	Break Detection Inputs
7	Address Line Test
8	Command Sequence End Detection
9	Interrogation "INTR1" Test
A	Program Memory Data Path
B	"MOVC" Logic Test
C	Interrupt at Break
D	Instruction Set Test
E	Miscellaneous Processor Tests

Almost all of these 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.

Each of these tests is described in the following sections.

Test 0 - Program Memory Test

Test 0 verifies the address uniqueness and data integrity of the program storage memory.

Test 1 - Command Memory Test

Test 1 verifies the address uniqueness and data integrity of the command sequence memory.

Test 2 - Break Register Memory Test

Test 2 verifies the address uniqueness and data integrity of the break register memory.

Test 3 - Break/Command Multiplexer Test

Test 3 verifies the operation of the address line multiplexer into the break and command memory.

Test 4 - RUNNING/INTD Status Test

Test 4 verifies that the "RUNNING" and the "INTD" status bits can be set and cleared with control procedures. It verifies that the proper idle state data is sent to the break address register.

Test 5 - Break Synchronization Test

Test 5 verifies that the break point detection circuit and the stop emulation circuit stop the emulation. This test fills the break register with data to break at any location executed.

Test 6 - Break Detection Inputs

Test 6 verifies that each of the four pairs of break register data in the break detector sets the address compare line properly.

Test 7 - Address Line Test

This test verifies the address line data in the break address register.

Test 8 - Command Sequence End Detector

Test 8 verifies that asserting address line 8 under command sequence control stops the sequence and returns the unit to the IDLE mode.

Test 9 - Interrogation "INTR1" Test

This test verifies that after writing to the general purpose emulator register in a command sequence the correct data is brought out of the processor when the "INTER" line is asserted in the control register.

Test A - Program Memory Data Path

This test single steps a two byte instruction (AA,55 - MOV R2,55) to verify the program data path to the processor.

Test B - "MOVC" Logic Test

Test B verifies the command sequence to fetch a byte of code data from program memory location with the address line number 11 asserted and not asserted.

Test C - Interrupt at Break

Test C verifies the operation of the simultaneous interrupt along with a "breakpoint" logic by emulating a routine that forces a timer interrupt to occur with an emulation break.

Test D - Instruction Set Test

This test loads an 8051 instruction set test into program memory and executes it.

Test E - Miscellaneous Processor Tests

Test E loads a routine that verifies the operation of the microprocessors internal RAM, timers, interrupts, and serial port.

Confidence Test Error Messages

CONTROLLER FAILED TO INITIALIZE
STATUS EXPECTED: 05H, ACTUAL: xxH

After writing to the control register 20 times, the controller failed to reset to the initial idle state.

PROGRAM
COMMAND MEMORY FAILED
BREAK REGISTER

DATA EXPECTED: eeH, ACTUAL: aaH, LOCATION: xxxxH

While testing one of the memory sets, a data or addressing error occurred at location (xxxxH). The expected data (eeH) did not match the actual data (aaH).

BREAK/COMMAND MEMORY MULTIPLEXER FAILED

SELECTOR STUCK LOW
SELECTOR STUCK HIGH
LOW NIBBLE FAILED TO SELECT
HIGH NIBBLE FAILED TO SELECT

Either the multiplexer select line does not change states or one of the multiplexers is failing to change states.

RUNNING FAILED TO SET
INTD CLEAR

Control output sequences failed to cause one of the status inputs to change properly.

IDLE BREAK ADDRESS FAILED

EXP: 0202, ACT: xxxxH

After putting the processor into an idle state, the break address did not contain the proper address.

BREAK SYNCHRONIZATION FAILED

BREAK ADDR-EXP: 0000H, ACT: xxxxH

NEXT ADDR-EXP: 0001H, ACT: yyyyH

After setting the break register to break on any instruction and executing a "NOP" instruction at location 0, the break address register failed to contain the right data.

BREAK DETECTOR FAILED WITH 11H AS INPUTS

22H

44H

88H

One of the four pairs of inputs into the break detector failed to trigger a break sequence.

ADDRESS LINE FAILURE

EXP: AAAAH, ACT: xxxxH

5555H

After executing a single step sequence to location AAAAH or 5555H, the break address failed to contain the proper data.

END OF COMMAND SEQUENCE FAILED

In a command sequence, a jump to a trigger address failed to set the "CMDEN" bit in the status register.

"INTR1" DATA FAILED
EXPECTED: AAH, ACT: xxH

After loading the general purpose EMV register in the processor with a command sequence and trying to read it back by asserting the "INTER" signal in the control register, the data failed to match.

PROGRAM MEMORY DATA PATH FAILED
R2 DATA - EXP: 55H, ACT: xxH

After loading location 55H with a 55H and R2 with 00 and executing a sequence (AAH, 55H - MOV R2, 55H), register R2 failed to contain a value of 55H.

"MOVC" SEQUENCE FAILED AT 0000H
0800H

The execution of a "MOVC" failed to fetch the proper data.

INTERRUPT DURING BREAK FAILED

After executing a program sequence that causes an interrupt during break, the status bit failed to indicate so.

INSTRUCTION SET FAILURE IN GROUP #xx
The processor failed the instruction set test.

TIMER x FAILED
INTERNAL RAM FAILURE
SERIAL PORT OUTPUT FAILED

These are errors that occurred while testing the internal functions of the 8051 processor.



- ACCUMULATOR command 8-7
 - Tutorial 4-3
- Address breakpoints 5-1, 5-3, 8-13
 - Tutorial 1-7, 5-3, 5-11, 6-12
- Advanced commands 2-1, 6-1 thru 6-19
- Architecture overview 1-1, 1-3
- Arithmetic operators 7-2
- ASM command 4-7 thru 4-8, 8-22, 8-24
- Assembly location counter 4-7

- Base 3-9, 8-6, 8-9, 8-10, 8-49
- BASE command 3-10, 8-10
- BC command 5-2, 8-10
- Branch breakpoints 5-2, 8-11
- BRB command 5-1, 8-11
 - Tutorial 5-10
- BREAK command 1-7, 5-3, 8-11
 - Tutorial 1-7, 5-3, 5-11
- Breakpoint restrictions 5-2
- B command 8-9
- B register 8-9
- BRR command 5-2, 8-12
- Br<n> command 5-1, 8-13
- BV command 5-2, 8-14

- CBYTE command 4-4, 4-5, 7-2 thru 7-3, 8-15
 - Tutorial 4-7, 6-11
- CBYTE restrictions 7-3
- CDUMP command 8-16
- Chapter preview 1-10
- CLEAR command C-2
- Clearing breakpoints 5-2, 8-10
- Clock A-5, A-6
- Code memory 7-2 thru 7-3
- Command categories 2-1
- Command files 6-5 thru 6-6
- Command line 2-2 thru 2-4, 8-1, 8-2
- Command line editing 2-3, 8-2 thru 8-3
- Command line execution 2-2, 8-2
- Comment lines 2-2
- Compound commands 6-7 thru 6-12
- Conditional IF commands 6-7 thru 6-9, 8-28 thru 8-29
- Confidence test C-1
- Confidence test commands C-2 thru C-6
- Confidence test error messages C-9
- Content operators 7-3 thru 7-4
- Continuation lines 2-2 thru 2-3, 8-2
- Control characters 2-3, 8-3
- Controller 1-1, 1-2, A-2, A-6, A-7
- Controlling the display 2-3, 8-3
- COUNT command 6-7, 6-12, 8-17
 - Tutorial 6-15
- CTRL P 2-3, 8-3
- CTRL Q 2-3, 8-3
 - Tutorial 1-9
- CTRL R 2-3, 8-3

- CTRL S 2-3, 8-3
 - Tutorial 1-9
- CTRL X 2-3, 8-3
- CTRL Z 2-3, 8-3

- DASM command 1-8 thru 1-9, 4-8, 8-18
 - Tutorial 1-10
- Data pointer register (DPTR) 4-1, 8-22 thru 8-23
- DBYTE command 4-4 thru 4-5, 8-19, 8-44
 - Tutorial 6-2, 6-8, 6-9, 6-18
- DBYTE content operator 7-3
- DDUMP command 4-5, 8-20
- DEFINE command 3-5, 6-1, 6-2, 6-8, 8-20 thru 8-21
 - Tutorial 3-7, 3-8, 6-2, 6-9, 6-11, 6-14, 6-18, 6-18
- DESCRIBE command C-2 thru C-3
- Designing the product 1-5
- Designing with the 8051 1-3
- Destination 8-6
- DIR command 6-3, 8-21
 - Tutorial 6-3
- DISABLE command 3-8, 6-6, 8-22
 - Tutorial 3-11, 6-6
- DISABLE EXPANSION 6-4, 8-22
 - Tutorial 6-6
- Display/modify commands 2-1, 4-1 thru 4-8
- Displaying register contents 4-1
- DTRACE command 5-6 thru 5-8, 8-23
 - Tutorial 5-8

- EA/ jumper 4-5, A-5
- Editing 2-3, 8-2 thru 8-3
- EM command 6-2
 - Tutorial 6-2, 6-8, 6-9, 6-11, 6-14, 6-16, 6-18
- Emulation
 - Before emulation 1-4 thru 1-5
 - Definition 1-4
 - Procedure 1-6 thru 1-10
 - Purpose 1-3
 - With emulation 1-5
- Emulation commands 2-1
- Emulation session 1-6 thru 1-10, 3-2 thru 3-4
- Emulator 1-1, 1-2, 1-6, A-2
- EMUL signal A-7
- EMV-51 components
 - Hardware 1-1 thru 1-2, A-2
 - Software 1-3, A-2, A-4 thru A-5
- EMV-51 demonstration program
 - Listing 1-11
 - Tutorials 3-4, 3-7, 3-8, 3-9, 3-11, 4-2, 4-3, 4-4, 4-6, 4-7, 4-9, 5-3 thru 5-4, 5-7 thru 5-8, 5-10 thru 5-12, 6-2, 6-3, 6-4, 6-6, 6-7, 6-8 thru 6-9, 6-10, 6-11 thru 6-12, 6-14 thru 6-19
- EMV51 invocation
 - Tutorial 1-7, 4-2
- ENABLE command 3-8, 8-24
 - Tutorial 3-11

- Enable expansion 6-4
 - Tutorial 6-6
- ERROR command C-3
- ESC 2-3, 8-2
- EVALUATE command 3-8, 8-24
 - Tutorial 3-8
- EXIT command 3-10, 8-25, C-3
 - Tutorial 3-11
- EXTBRK signal A-7
- EXTI0 4-4
- EXTI1 4-4
- Expression 8-5, 8-6, 8-7, 8-8, 8-9
- External data memory 7-3

- FUNCTION command 6-12 thru 6-13, 8-26 thru 8-27
 - Tutorial 6-15
- FUNCT S 2-3
- FUNCT 2 5-3
- FUNCT 3 5-7
- Function keys 6-12 thru 6-13

- GO command 5-9, 5-10, 8-14, 8-27
 - Tutorial 1-10, 5-10, 6-11

- Hardware 1-2
- HELP command 8-28
 - Tutorial 3-1, 3-2, 3-4
- Help information 3-1

- IF..THEN..ELSE 6-7 thru 6-8, 8-28
 - Tutorial 6-8 thru 6-10, 6-16 thru 6-19
- IGNORE command C-3 thru C-4
- INCLUDE command 6-5 thru 6-6, 8-30
 - Tutorial 6-6
- INI51CON command C-1
- Installation procedures
 - Controller A-2
 - Emulator module A-2
 - EMV-51 software A-2, A-4 thru A-5
- INTERRUPT command 8-30 thru 8-13
 - Tutorial 4-4
- Interrupt enable 4-3
- Interrupt priority 4-3
- Interrupts in progress 4-3
- Item 8-4

- Jumper configurations A-5
- LIST command 3-3, 3-4, 8-31, C-4
 - Tutorial 3-4
- List file 3-3, 3-4
- LOAD command 1-7, 3-2, 3-3, 8-32
 - Tutorial 1-7, 3-4, 4-2
- Loading programs 1-7, 3-2 thru 3-4
- Loading multiple programs 3-3
- Logical operators 7-2, 7-6

- Macros 2-1, 2-2, 6-1 thru 6-19
- MACRO command 6-3, 8-33
 - Tutorial 6-3
- Macro definitions 6-1, 6-2
- Macro execution 6-3 thru 6-4, 8-52
 - Tutorial 6-4, 6-6, 6-10, 6-12, 6-17, 6-19
- Macro information 6-2 thru 6-3
- Manipulating symbols 3-5
- MEMORY command 4-6, 8-34
 - Tutorial 4-7
- Memory contents 4-4, 4-5
- Messages 2-1 thru 2-2
- Modify commands 2-1
- Multiplication register 8-9

- Notational conventions 8-3 thru 8-4

- ON ERROR C-6
- ON NOERROR C-6
- Operators 7-2, 7-6

- Parameters 6-1 thru 6-2, 6-3
- PC 4-1, 8-36
- P command 5-10, 8-34
 - Tutorial 1-10, 5-10
- PBYTE command 8-35
- PCONF command C-1
- Port 0 A-7, A-8
- Prompts 2-1, 2-2
- PSENE signal A-7
- PSW 8-37
- PUT command 6-5, 6-6, 8-38
 - Tutorial 6-6

- Range breakpoints 5-2
- RBIT command 8-39
- RBS command 8-40
- RBYTE command 8-40
- RBYTE content operator 7-3
- RECOGNIZE command C-4
- REGISTER command 4-2, 4-3, 8-41
 - Tutorial 4-3, 6-16, 6-18
- Register contents
 - Displaying 4-1
 - Displaying INTERRUPT 4-3
 - Modifying 4-1
- Register keyword names 4-1
- Register value breakpoints 5-2
- Relational operators 7-2, 7-6
- REMOVE command 3-6, 6-5, 8-43
 - Tutorial 3-7, 6-6, 6-16, 6-18
- REPEAT command 6-7, 6-11 thru 6-12, 8-44
 - Tutorial 6-8, 6-11
- RESET command 3-10, 8-45
- Reset input A-6
- RETURN 2-3, 8-2
- RUBOUT 2-3, 8-3

- SAVE command 3-10, 8-46
 - Tutorial 1-10, 3-10
- Saving programs 1-10, 3-10, 3-11
- Setting breakpoints 5-1
- Sint 4-4

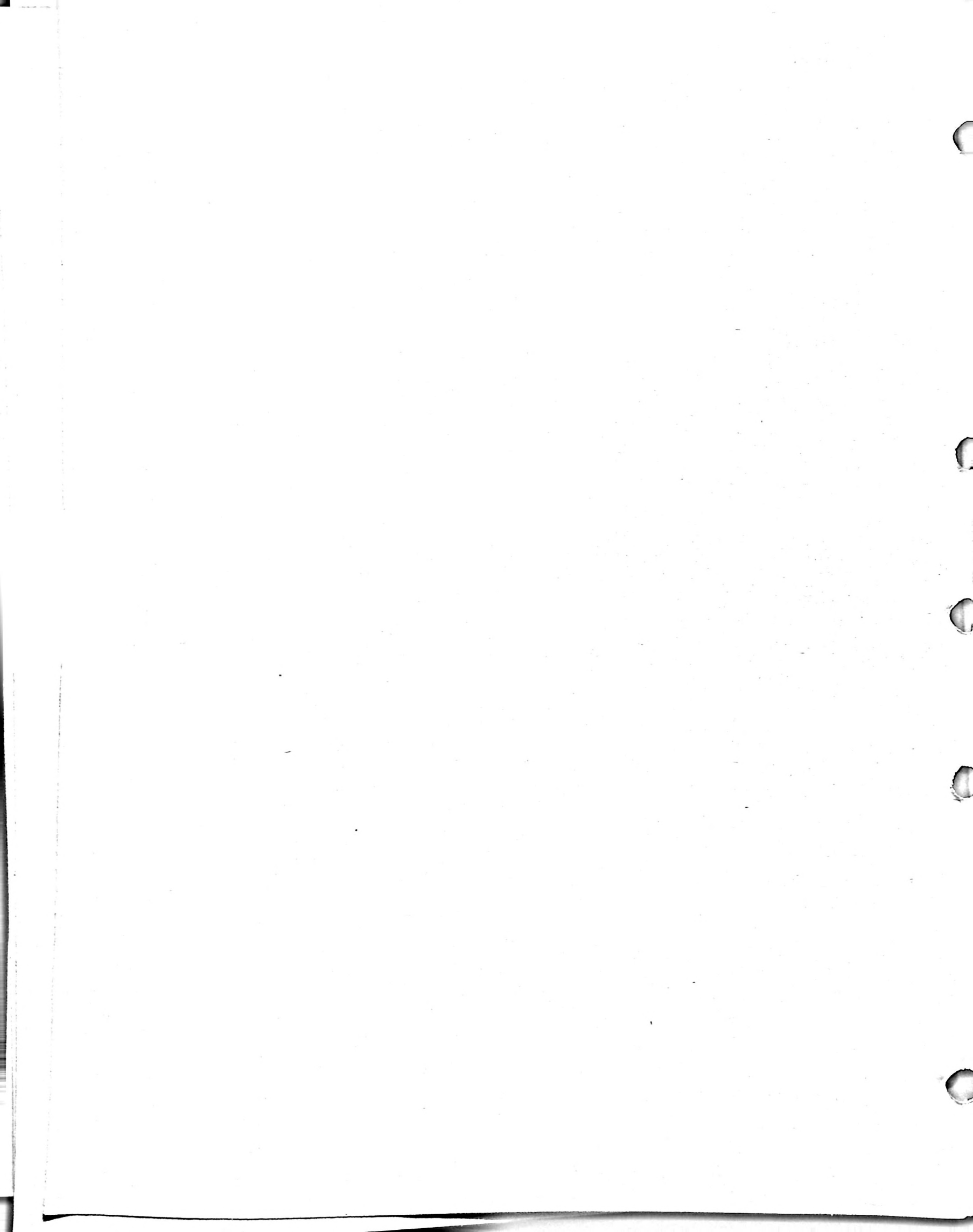
Software 1-2, 1-7, 1-9, A-2, A-4 thru A-5
Source 8-6
SP command 4-1, 8-47
STEP command 5-8, 5-9, 8-48
 Tutorial 5-12, 6-9, 6-16, 6-18
String 8-5, 8-6
Suffix 3-9
SUFFIX command 3-9, 8-4, 8-49
 Tutorial 3-9
SUMMARY command C-4 thru C-5
Supply voltage A-6
SYMBOLS command 3-6, 8-50
 Tutorial 3-7
Symbol handling 8-53
Symbolic reference
 Assigning symbol names 3-5
 Displaying symbol names 3-6
 Manipulating symbols 3-5
 Removing symbol names 3-6
 Tutorial 3-7

TBn command 5-5
TD command 5-5
 Tutorial 5-11
TEST 0 C-7
Test 1 C-7
Test 2 C-7
Test 3 C-7
Test 4 C-7
Test 5 C-7 thru C-8

Test 6 C-8
Test 7 C-8
Test 8 C-8
Test 9 C-8
Test A C-8
Test B C-8
Test C C-8
Test D C-8
Test E C-8
TM0 8-50
TM1 8-51
Timer0 4-4
Timer1 4-4
TR command 5-5
 Tutorial 5-11
Trace 5-1, 5-4 thru 5-8, 5-11
TRn command 5-6, 5-7
 Tutorial 5-7
TS command 5-5
TV command 5-6

UPPERCASE 8-4
User publications 1-3
User test signals A-6 thru A-7
Utility commands 2-1, 3-1 thru 3-11

Workfiles 8-1 thru 8-2
WRITE command 6-2, 8-2, 8-52
 Tutorial 6-7, 6-8, 6-14





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 _____

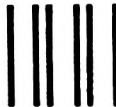
ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____
(COUNTRY)

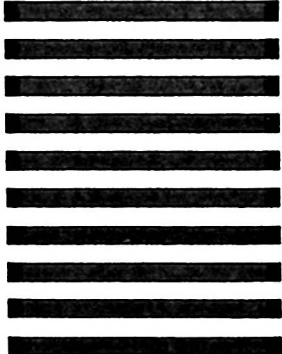
Please check here if you require a written reply.

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.