# ICD-378
## FOR
# 8051
# USER'S MANUAL

# ICD-378
# FOR
# 8051
# USER'S MANUAL

Copyright © 1986, U.S. ZAX CORPORATION. All Rights Reserved.
Part No. ZTP-111-XX, Rev. A. Printed: April 1986

## Limitation on Warranties and Liability

ZAX Corporation warrants this equipment to be free from defects in materials and workmanship for a period of 1 (one) year from the original shipment date from ZAX. This warranty is limited to the repair and replacement of parts and the necessary labor and services required to repair this equipment.

During the 1-year warranty period, ZAX will repair or replace, at its option, any defective equipment or parts at no additional charge, provided that the equipment is returned, shipping prepaid, to ZAX. The purchaser is responsible for insuring any equipment returned, and assumes the risk of loss during shipment.

Except as specified below, the ZAX Warranty covers all defects in material and workmanship. The following are not covered: Damaged as a result of accident, misuse, abuse, or as a result of installation, operation, modification, or service on the equipment; damage resulting from failure to follow instruction contained in the User's Manual; damage resulting from the performance of repairs by someone not authorized by ZAX; any ZAX equipment on which the serial number has been defaced, modified, or removed.

## Limitation of Implied Warranties

ALL IMPLIED WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO THE LENGTH OF THIS WARRANTY.

## Exclusion of Certain Damages

IN NO EVENT WILL ZAX BE LIABLE TO THE PURCHASER OR ANY USER FOR ANY DAMAGES, INCLUDING ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, EXPENSES, LOST PROFITS, LOST SAVINGS, OR OTHER DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THIS EQUIPMENT. THIS EXCEPTION INCLUDES DAMAGES THAT RESULT FROM ANY DEFECT IN THE SOFTWARE OR MANUAL, EVEN IF THEY HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

## Disclaimer

Although every effort has been made to make this User's Manual technically accurate, ZAX assumes no responsibility for any errors, omissions, inconsistencies, or misprints within this document.

## Copyright

This manual and the software described in it are copyrighted with all rights reserved. No part of this manual or the programs may be copied, in whole or in part, without written consent from ZAX, except in the normal use of software or to make a backup copy for use with the same system. This exception does not allow copies to be made for other persons.

TABLE OF CONTENTS


1. INTRODUCTION ..... 1-1
   1.1. What is an In Circuit Emulator?

2. GETTING STARTED ..... 2-1
   2.1 Hardware Installation ..... 2-1
      2.1.1 RS-232C Interface Board Installation ..... 2-1
      2.1.2 RS-232C Cable Installation ..... 2-3
      2.1.3 Emulator Module Power Supply Cable Assembly ..... 2-3
      2.1.4 Powering Up the Emulator Module ..... 2-4
   2.2 Software Installation ..... 2-5
   2.3 Warm Start Feature ..... 2-6
   2.4 Getting Oriented ..... 2-6

3. USER INTERFACE OVERVIEW ..... 3-1
   3.1 Menu Organization ..... 3-1
   3.2 Using Menus ..... 3-2
   3.3 User Options, Errors and Warnings ..... 3-3
   3.4 Directory Facility ..... 3-3

4. MAIN MENU ..... 4-1
   4.1 Main Menu Overview ..... 4-1
   4.2 Main Menu Screen ..... 4-1
   4.3 Load Command ..... 4-2
   4.4 Upload Command ..... 4-4
   4.5 Download Command ..... 4-5
   4.6 Config Command ..... 4-6
   4.7 Restore Command ..... 4-7
   4.8 Store Command ..... 4-8
   4.9 Map Command ..... 4-9
   4.10 % (Macro) Command ..... 4-11
   4.11 Interrogate Command ..... 4-11
   4.12 Help Command ..... 4-12
   4.13 Exit Command ..... 4-12

5. CONFIGURATION MENU ..... 5-1
   5.1 Configuration Menu Overview ..... 5-1
   5.2 Configuration Menu Screen ..... 5-1
   5.3 Execute Command ..... 5-1
   5.4 Change Command ..... 5-2
      5.4.1 Mode 1 - ROMless part / 16 bit ext addr bus ..... 5-4
      5.4.2 Mode 2 - ROM part / NO ext address bus ..... 5-4
      5.4.3 Mode 3 - ROM part / 8 but ext address bus ..... 5-5
      5.4.4 Mode 4 - ROM part / 16 bit ext address bus ..... 5-7

6. INTERROGATE MENU ..... 6-1
   6.1 Interrogate Menu Overview ..... 6-1
   6.2 Interrogate Menu Screen ..... 6-2
   6.3 Go Command ..... 6-5
   6.4 S-Step Command ..... 6-9
   6.5 Reset Command ..... 6-11
   6.6 Fantom (Phantom) Command ..... 6-13
   6.7 Brk-cnt (Break-Count) Command ..... 6-18
   6.8 Break/Trace-Trigger Points ..... 6-19

6.9 Increment Pass-Count Points ..... 6-21
6.10 Loop-cnt (Loop-Count) Command ..... 6-24
6.11 Trace-trig (Trace-Trigger) Command ..... 6-25
6.12 Help Command ..... 6-26
6.13 Quit Command ..... 6-26
6.14 Modify-Regs Command ..... 6-26
6.15 Data-Mem Command ..... 6-31
6.16 Xdata-Mem Command ..... 6-31
6.17 Code-Mem Command ..... 6-31
6.18 View-trace Command ..... 6-32
6.19 START Trace-Trigger ..... 6-33
6.20 CENTER Trace-Trigger ..... 6-35
6.21 END Trace-Trigger ..... 6-37
6.22 Experiment Command ..... 6-39
6.23 I/O Command ..... 6-39
6.24 Pass-cnt (Pass-Count) Command ..... 6-39

7. EXAMINE / MODIFY PROGRAM CODE MEMORY ..... 7-1
7.1 Examine / Modify Program Code Memory Overview ..... 7-1
7.2 Examine / Modify Program Code Memory Screen ..... 7-1
7.3 Dissassemble Command ..... 7-2
7.4 Assemble Command ..... 7-6
7.5 Table Command ..... 7-12
7.6 Help Command ..... 7-12
7.7 Quit Command ..... 7-12

8. EXAMINE/MODIFY INTERNAL DATA MEMORY ..... 8-1
8.1 Examine/Modify Memory Data Overview ..... 8-1
8.2 Examine/Modify Memory Data Screen ..... 8-1
8.3 Dump Command ..... 8-3
8.4 Enter Command ..... 8-5
8.5 Fill Command ..... 8-8
8.6 Move Command ..... 8-13
8.7 Search Command ..... 8-16
8.8 Compare Command ..... 8-20
8.9 RAM-Bits Command ..... 8-23
8.10 Help Command ..... 8-25
8.11 Quit Command ..... 8-25

9. THE EXPERIMENT ..... 9-1
9.1 What is an Experiment? ..... 9-1
9.2 Specifying Breakpoints ..... 9-2
9.2.1 PC Address Breakpoints ..... 9-3
9.2.2 PC Address Range Breakpoints ..... 9-3
9.2.3 Opcode Value Breakpoints ..... 9-4
9.2.4 Opcode Class Breakpoints ..... 9-5
9.2.5 Direct Byte Address Breakpoints ..... 9-5
9.2.6 Direct Byte Address Range Breakpoints ..... 9-6
9.2.7 Direct Bit Address Breakpoints ..... 9-7
9.2.8 Direct Bit address Range Breakpoints ..... 9-8
9.2.9 Immediate Operand Value Breakpoints ..... 9-9
9.3 Complex Conditional Statements ..... 9-9
9.4 Constructing An Experiment ..... 9-10
9.5 Experiment Language Syntax Summary ..... 9-11

10. EMAMINE/MODIFY EXPERIMENT MENU ..... 10-1
10.1 Examine/Modify Experiment Menu Overview ..... 10-1
10.2 Examine/Modify Experiment Menu Screen ..... 10-1
10.3 Edit Command ..... 10-1
10.4 Compile Command ..... 10-1
10.5 Load Command ..... 10-6
10.6 Store Command ..... 10-7
10.7 Delete Command ..... 10-8
10.8 Opcode Command ..... 10-8
10.9 Help Command ..... 10-8
10.10 Quit Command ..... 10-8

11. EXPERIMENT EDITOR ..... 11-1
11.1 Experiment Editor overview ..... 11-1
11.2 Experiment Editor Screen ..... 11-1
11.3 Using the Experiment Editor ..... 11-2
11.4 Line Entry Mode ..... 11-2
11.5 Edit Mode ..... 11-3
11.5.1 Edit-Replace Mode ..... 11-5
11.5.2 Edit-Insert Mode ..... 11-5
11.6 Edit Command Mode ..... 11-5
11.6.1 Edit Command ..... 11-6
11.6.2 Save Command ..... 11-6
11.6.3 Quit Command ..... 11-7

12. OPCODE CLASS MENU ..... 12-1
12.1 Opcode Class Menu Overview ..... 12-1
12.2 What is an Opcode Class? ..... 12-1
12.3 Opcide Class Menu Screen ..... 12-2
12.4 Load Command ..... 12-2
12.5 Edit Command ..... 12-4
12.5.1 File Prompt Mode ..... 12-4
12.5.2 Class Selection Mode ..... 12-5
12.5.2.1 Edit-Class Command ..... 12-7
12.5.2.2 Delete-Class Command ..... 12-7
12.5.2.3 Create-Class Command ..... 12-8
12.5.2.4 Rename-Class Command ..... 12-9
12.5.2.5 Quit Command ..... 12-10
12.5.3 Class Edit Mode ..... 12-11
12.6 Help Command ..... 12-18
12.7 Quit Command ..... 12-18

13. MACRO MENU ..... 13-1
13.1 Macro Menu Overview ..... 13-1
13.2 Macro Menu Screen ..... 13-1
13.3 Execute Command ..... 13-1
13.4 Learn Command ..... 13-3
13.5 Help Command ..... 13-5
13.6 Quit Command ..... 13-5

14. A TUTORIAL EXAMPLE ..... 14-1
14.1 Introduction ..... 14-1
14.2 Getting Started ..... 14-1
14.3 The Program ..... 14-2
14.4 a Sample Session ..... 14-4

15. SYSTEM REQUIREMENTS ..... 15-1
15.1 Hardware Requirements ..... 15-1
15.2 Software Requirements ..... 15-1

A. TROUBLE SHOOTING ..... A-1
A.1 Cannot Establish Communication ..... A-1
A.1.1 Red LED is NOT Glowing ..... A-2
A.1.2 Green LED is NOT Glowing ..... A-3
A.1.3 Baud Rate Selection ..... A-4
A.1.4 RS-232C Hardware Check ..... A-4
A.2 Excessive Number of Communication Errors ..... A-4

B. ERROR MESSAGE SUMMARY ..... B-1
B.1 PLINK86 Overlay Loader Errors ..... B-1
B.2 Errors Messages ..... B-1
B.3 DOS Errors Messages ..... B-9

C. SIGNAL SPECIFICATIONS AND DIFFERENCES ..... C-1
C.1 Signal Specifications ..... C-1
C.2 Probe Cable Characteristics ..... C-4
C.3 Signal Differences ..... C-4

D. OTHER DIFFERENCES ..... D-1
D.1 Timer0 and Timer1 Values ..... D-1
D.2 Serial Port ..... D-1
D.3 Port Registers ..... D-1

E. CHARACTER SETS AND RESERVED SYMBOLS ..... E-1
E.1 Single Line Code Assembler Character Set ..... E-1
E.2 Experiment Compiler Character Set ..... E-1
E.3 Fill and Search Pattern Character Set ..... E-1
E.4 Experiment Compiler Reserved Keywords ..... E-1

F. PREDEFINED BYTE AND BIT ADDRESSES ..... F-1
F.1 8031 Predefined Addresses ..... F-1
F.1.1 Predefined Byte Addresses ..... F-1
F.1.2 Predefined Bit Addresses ..... F-1
F.2 8032 Predefined Addresses ..... F-2
F.2.1 Predefined Byte Addresses ..... F-2
F.2.2 Predefined Bit Addresses ..... F-3

G. PREDEFINED OPCODE CLASSES ..... G-1
G.1 OPCODE Class PGMFLOW ..... G-1
G.2 Opcode Class STACK ..... G-2

# CHAPTER 1
# INTRODUCTION

The ICD-378 for 8051 emulator is an in-circuit emulator which is designed for use in developing and debugging circuits based on an Intel single chip microcontroller. The ICD-series emulators currently support the following Intel single chip microcontrollers:

ICD-378 for the 8051
8052
8031
8032

## 1.1 What is an In-Circuit Emulator?

An In-circuit Emulator is a tool which enhances the productivity of system design engineers. It is used by engineers who are designing a system which incorporates a microcontroller. It provides for the engineers, the ability to interactively control and examine the state of the system at any chosen time. This is essential for speeding up the debugging process.

As the name implies, this capability is provided by removing the microcontroller from the system and replacing it with the emulator's probe. Thus the emulator's probe is In the Circuit in place of the microcontroller. The emulator probe is in turn connected to the host computer. It is through the computer (and through the probe) that the system can be completely controlled.

You may ask 'why is this important?'. It is not enough for the emulator to simply behave as if it were the target processor (although this is a requirement also). The emulator must also provide read/write access to all signals and all data to which the microcontroller itself has access. This includes information which resides inside the microcontroller itself. Without this access, the engineer may NOT be able to completely control and debug the system.

The many uses of the emulator can be easily visualized after we examine a typical system design cycle.

DESIGN CYCLE

| Phase: | Tools Used: |
|---|---|
| conception | RTL simulator |
| architecture | logic simulator |
| software/hardware prototype | emulator/simulator<br>CAD/CAM |
| integration | emulator |
| manufacturing testing | emulator/testor |
| field testing | emulator/testor |

The first use of an emulator in the design cycle is in the software development phase. In this phase, the software which is going to run on the microcontroller is being developed. The emulator provides an ideal environment in which to debug the software. It executes the program exactly as the target would (in real time) and it provides all of the interactive debugging capabilities. By utilizing the emulator to develop the software, it can be completely debugged (except for the hardware interface) before it is integrated with the system hardware.

The second use of the emulator in the design cycle is in the integration of the target software and the system hardware. This constitutes the major use of the emulator. Even when the hardware and software have each been individually debugged, new problems can surface when they are joined together. The emulator is used in this case to find and debug these interface problems.

After a prototype has been completely debugged, the emulator can then be used to test the specs of the system. Worst case parametric tests can be developed and tested on the prototype. This provides the designer with valuable information about the limitiation of the system. It also provides test programs which can be used in the manufacturing process.

The third use of an emulator is in the manufacturing phase of the product. The same test routines which were used to develop and debug the prototype (or even more comprehensive test routines) can be used to test the finished products after manufacturing. Any non-functioning units can be easily debugged using the emulator's full range of debugging capabilities.

The fourth use of an emulator is in the field service phase of the product. The ICD can run on any IBM PC or PC compatible host computer (including the PC compatible portables). If the field location already has a host computer, the field service team need only carry with them, the emulator module itself (which can easily fit in a briefcase) and some floppy disks. If a host computer isn't available, a portable host can also be easily brought along. This eliminates the need for carrying around bulky and heavy test equipment, while still providing the power of an emulator debugging environment.

CHAPTER 2
GETTING STARTED

## 2.1 Hardware Installation

The following is a step by step procedure for the installation of the ICD hardware.

### 2.1.1 RS-232C Interface Board Installation

The following signals are used in a standard RS-232C interface:

TxD - transmit data
RxD - receive data
RTS - request to send
CTS - clear to send
DSR - data set ready
DCD - data carrier detect
DTR - data terminal ready

A standard RS-232C interface is configured as follows:

| HOST | | | PERIPHERAL | |
|---|---|---|---|---|
| Signal | Pin | Cable | Pin | Signal |
| Ground | 1 | ---------- | 1 | Ground |
| TxD | 2 | ----------> | 2 | RxD |
| RxD | 3 | <---------- | 3 | TxD |
| RTS | 4 | ----------> | 4 | RTS |
| CTS | 5 | <---------- | 5 | CTS |
| DSR | 6 | <---------- | 6 | DSR |
| Ground | 7 | ---------- | 7 | Ground |
| DCD | 8 | <---------- | 8 | DCD |
| DTR | 20 | ----------> | 20 | DTR |

Figure 1.

Notice that the cable connects each of the pins on the host side to its counterpart on the peripheral side. It is important that this 1:1 correspondence is maintained.

The ICD utilizes a 3 wire version of this configuration. It is configured as follows:

| HOST | | | ICD | |
|---|---|---|---|---|
| Signal | Pin | Cable | Pin | Signal |
| Ground | 1 | | | |
| TxD | 2 | ----------> | 2 | RxD |
| RxD | 3 | <---------- | 3 | TxD |
| RTS | 4 | | | |
| CTS | 5 | | | |
| DSR | 6 | | | |
| Ground | 7 | ---------- | 7 | Ground |
| DCD | 8 | | | |
| DTR | 20 | | | |

Figure 2.

Pins 1, 4, 5, 6, 8 and 20 of a standard RS-232C interface are not used. The common ground is provided through pin 7. The RTS, CTS, DRS, DTR and DCD are not required in the ICD communication protocol. The configuration will actually be as follows:

| HOST | | | ICD | |
|---|---|---|---|---|
| Signal | Pin | Cable | Pin | Signal |
| TxD | 2 | ----------> | 2 | RxD |
| RxD | 3 | <---------- | 3 | TxD |
| Ground | 7 | ---------- | 7 | Ground |

Figure 3.

The ICD RS-232C cable is a 3 wire cable which is shown in Figure 3. A standard RS-232C cable (shown in Figure 1.) can also be used to connect the host with the emulator module provided the emulator end of the cable has a male connector.

If you have purchased the RS-232C interface board from ZAX, the instructions for installing the board in the host computer are packaged with the board.

The RS-232C interface board which is used with the ICD MUST be configured as COM1.

### 2.1.2 RS-232C Cable Installation

Attach the RS-232C cable to both the RS-232C connector on the interface card (in the card slot at the back of the host) and to the connector on the ICD module. Be sure that both ends are seated firmly and securely.

### 2.1.3 ICD Module Power Supply Cable Assembly

If you have purchased the ICD Power Supply, you can skip this section because your power supply cable comes preassembled.

If you are using your own power supply to power the ICD module, please ensure that it meets the following specifications:

+ 5VDC +- 5%
2.0A
50mV ripple peak to peak

+ 12VDC +- 3V
150mA

- 12VDC +- 3V
150mA

A 5 wire connector is supplied with the ICD which you can use to supply power to the ICD module. The wires themselves are not provided. The connector accepts wire gage #18 A.W.G. Figure 4 shows the order in which the power lines should be supplied.

Power Supply cable key:

Pin #1 = Ground

Pin #2 = Ground

Pin #3 = +12VDC

Pin #4 = -12VDC

Pin #5 = +5VDC

### 2.1.4. Powering Up the Emulator Module

The ICD module can be powered up by first inserting the power cable into the power connector on the side of the module. The ICD is shipped with the emulator probe inserted in the DIP socket on the Simulator board. Be sure that the probe is still inserted in that socket and that it is in the correct orientation. This can be determined by matching pin 1 on the emulator probe with pin 1 on the DIP connector.

Next turn on the power supply. If you look into the air vents on the side of the emulator module toward the end of the RS-232 connector, you can see two LEDs. If the red LED is on, the power is supplied properly to the emulator module. If the green LED is on, the oscillator in the emulator is running and the emulator is ready to establish communication with the host.

If the LEDs were not on as described above, see Appendix A for the trouble shooting guide.

Turn off the power supply until you are ready to begin an emulation session.

## 2.2 Software Installation

The ICD user interface software is supplied on 1 5-1/4" double sided double density floppy disk. It is formatted under PC DOS with 9 sectors per track. The first thing you should do is to make a backup copy of the software. You can copy the program a file at a time or with the DOS 'diskcopy' command. The files which are included with the ICD system are:

```
@.exe
$model
$hlpfile
$config
demo.dbg
demo.hex
demo.asm
demo.exp
opclass.opc
```

Be sure that your working copy of the ICD software is not write protected. This enables the ICD system software to update the $config file when you change the system configuration of your ICD environment.

When running the ICD user interface software, ALWAYS be sure that the ICD floppy disk is in the DEFAULT disk drive. The program will not run properly if it is not in the DEFAULT drive.

During the course of debugging a product, you will most likely create many different test files. The following is a listing of the different types of files which are used with the ICD.

| File Types | Formats |
|---|---|
| assembly language source code | text |
| assembled hex files | Intel hex object format |
| assembled symbolic debug code | ICD object format |
| experiment files | text |
| macro command files | ICD command format |
| opcode class files | ICD opcode class format |
| system status files | ICD system status format |
| macro command files | ICD macro command format |

It is suggested that you adopt a naming strategy which will avoid confusing these files with each other. ICD file formats are NOT interchangable with each other. If you try to use a file in a context other than that for which it was created, you will get a FILE FORMAT error.

If you want to use the opcode class file 'opclass.opc' in your experiments, it is suggested that you copy it onto the disk with your other working files. This is because opcode class files are opened in the read/write mode (so you can delete opcode classes, edit opcode classes and create new opcode classes). If you used the opcode class file on the system disk (which should be write protected), you would encounter an error when you tried to update the file.

## 2.3 Warm Start Features

The ICD system provides a warm start feature. A warm start is achieved through the following sequence:

1) Enter the ICD environment.
2) While in the ICD environment, execute the system configuration command.
3) Leave the ICD environment.
4) At some later time, re-enter the ICD environment.

The reentering of the ICD environment is called a warm start. Communication with the emulator will automatically be reestablished and the map settings (see chapter 4.9) for code and external data memories will be reset to their prior values. The register values will also be initialized to the actual values read from the emulator. This assumes of course that the emulator module's power has been maintained in between ICD debugging sessions.

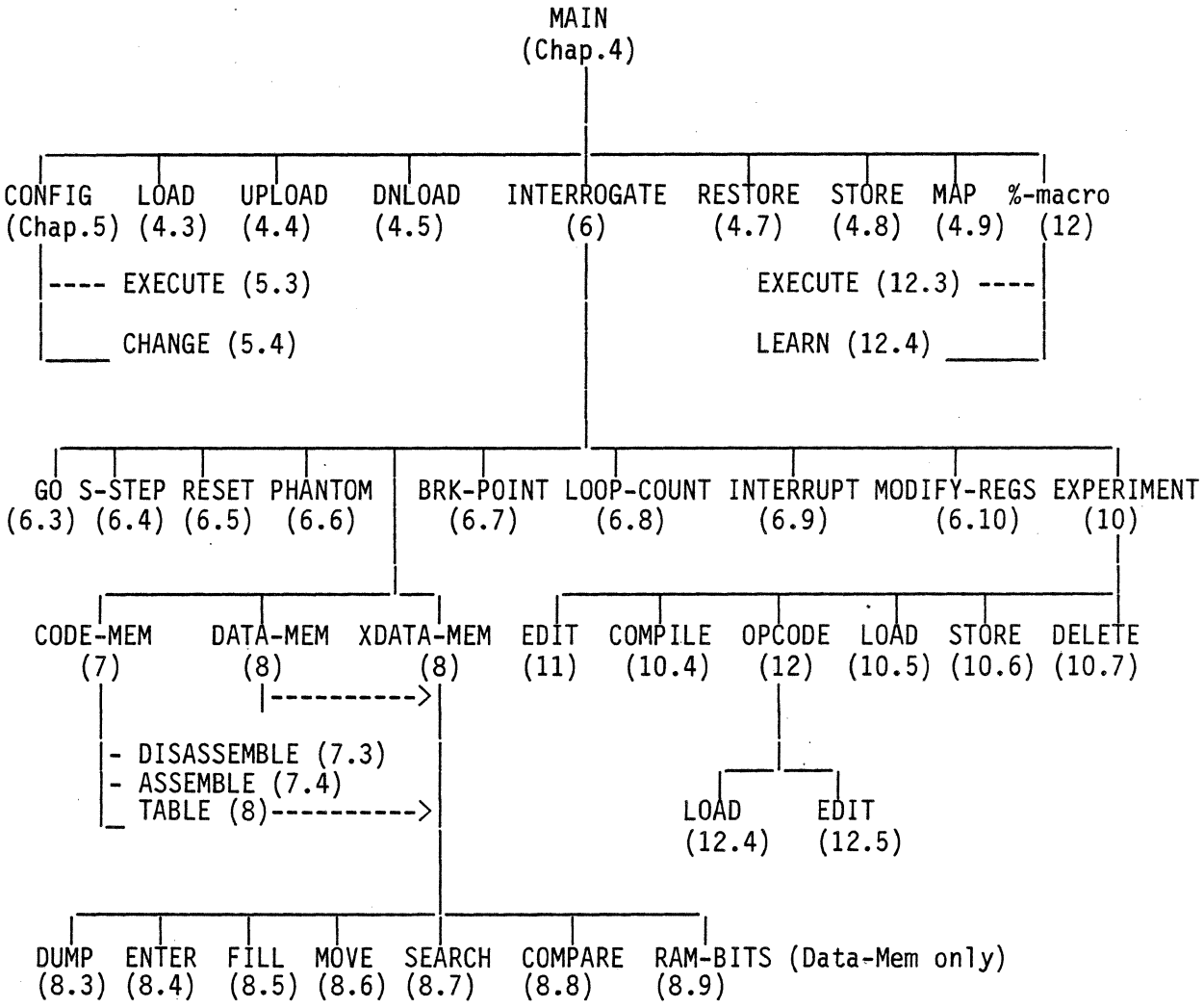
## 2.4. Getting Oriented

The ICD User's Manual provides a detailed description of the use and capabilities of the ICD system. It is strongly recommended that you read through the manual to familiarize yourself with the system. After having read the manual (not as an alternative to reading it) it is suggested that you work through the tutorial example which is presented in Chapter 14.

CHAPTER 3
USER INTERFACE OVERVIEW

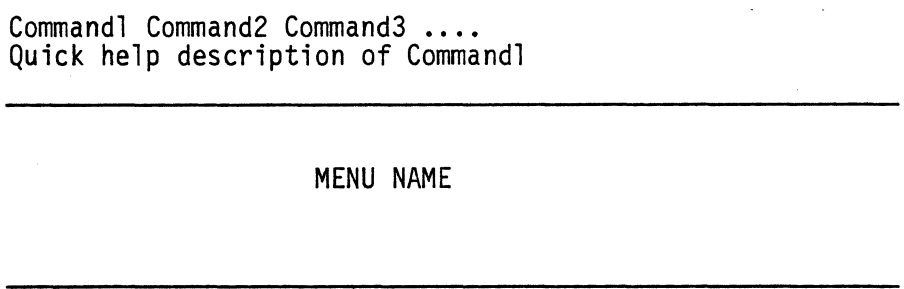
## 3.1 Menu Organization

The menu organization of the ICD user interface software is shown below. Under each menu item can be found the chapter or sub-chapter in which that menu item is described.



## 3.2 Using Menus

A menu screen is structured as follows:




The first line of the screen will contain a list of the command options available for that menu. The second line will contain a one line description of the highlighted command (see below). The middle of the screen will contain the men's name. The line at the bottom of the screen will contain any errors, warnings or messages encountered during the execution of a command.

Menu commands may be selected for execution by either of the following two methods. The first method is to move the highlight to the desired menu command and then hit the RETURN key. Upon first entering a menu, the first command on the left is always highlighted. The highlight may be moved through the use of the cursor control keys on the numeric keypad at the right of the keyboard. The cursor control keys cursor right (-->) and cursor down (↓) will move the highlight one command to the right. The cursor control keys cursor left (<--) and cursor up (↑) will move the highlight one command to the left. In either case wrap-around occurs when the end of the command list is encountered.

The second method of executing menu item is to type the first letter of the desired item in either upper or lower case. Only the first character of the item name is required since no two items in any given menu begin with the same letter.

Any other character will be ignored with the exception of CTRL-C. Typing CTRL-C at any time will abort the program with the ensuing loss of any information which was not previously stored. The preferred method of exiting the program is by executing the Exit command in the Main Menu.

## 3.3 User Options, Errors and Warnings

The system software was designed to be user friendly and fault tolerant. All menus are operated as described above. All tasks other than menus will prompt you for the required information. Any options open to you will be presented at the bottom of the screen. Thus you will always know what your choice of actions is.

When you are prompted for addresses or data by the system, it will always inform you of the required radix. When responding to these prompts, only the numeric value in the required radix should be provided. A radix indicator is not necessary since the system expects the information to be in the specified radix.

Any errors or warnings will also be displayed at the bottom of the screen as they occur. These messages are in English text and are explicit enough to determine the nature of the error. A fuller description of all error and warning messages can be found in Appendix B.

Whenever a command is executed which will take some time to perform, the WORKING sign will appear at the bottom of the screen. It will be flashing. As long as the display is flashing, the system is busy and will not accept commands.

## 3.4 Directory Facility

The ICD provides a directory facility through which you can get a listing of the entries in any directory. The facility can be invoked any time the system prompts you for a file name. Responding to the prompt which appears as follows:

```
Enter file name > ?
Enter disk or directory name > _
```

---

MENU NAME

---

The response can be a disk drive specification, a directory pathname, or a [RETURN] which selects the current default directory.

A disk drive is specified by its drive designator. The valid drive designators are A through P. when specifying a drive, you do not need to specify the ':' suffix. Any one letter response is assumed to be a drive specification.

A directory pathname specification can consist of a drive specification and/or a path of directory names to the desired directory. Directory names are separated by backslashes (\).

Examples of valid responses are:

| | |
|---|---|
| [RETURN] | default directory |
| a | top directory on A: disk |
| a: | top directory on A: disk |
| \dir1\subdir1 | subdirectory on on default disk |
| c:\dir1\subdir1\subdir2 | subdirectory on C: disk |

The files in the directory will be in alphabetical order. The directory display will appear as follows:

```
Enter file name > ?
Enter disk or directory name > _
```

---

Directory for <disk or directory pathname>

```
file1.ext  file2.ext  file3.ext  file4.ext  file5.ext
file6.ext  file7.ext  file8.ext  file9.ext  file10.ext
                          .
                          .
                          .
```

---

The number of files in a directory may exceed the display area of one screen. In this case, the screen fills up from top to bottom until the display area is full. As more names are listed, the display area is scrolled upward one line and a new line with five names appears on the bottom of the display area.

You can halt the scrolling action at any time by hitting the [SPACE BAR] key. This puts the system in the single step display mode. After entering the single step display mode, one new line of five names is displayed each time you hit the [SPACE BAR] key. Normal display scrolling can be resumed by hitting ANY key other than the [SPACE BAR] key. The single step display mode is automatically terminated when the directory listing is completed. Once the directory listing has begun, it can be aborted at any time by hitting the [ESC] key.

If the specified directory contains no files, the following message is displayed:

Directory is empty

Error messages which may be encountered when specifying the disk drive or directory name include:

Illegal drive specification - the specified drive designator is not between A and P.

Directory not found - a directory was specified which did not exist or couldn't be reached.

CHAPTER 4
MAIN MENU

## 4.1 Main Menu Overview

The Main Menu is used to initialize system parameters prior to running an emulation experiment, to call the Interrogate Menu and to terminate a session. In this menu you can:

1) load program code memory from disk files
2) upload program code memory from your target system board
3) download user board external data memory from disk files
4) call the system Configuration Menu
5) restore a previously saved system status
6) store the system status in a disk file
7) setup the mapping for the code and external data memories
8) create or execute a macro command file
9) call the Interrogate Menu
10) terminate a session.

## 4.2. Main Menu Screen

The Main Menu screen appears as follows:

Load Upload Dnload Config Restore Store Map % Interrogate
Help Quit (Quick help line for highlighted command)

---

MAIN MENU

---

Upon entering, the Load command will be highlighted.

## 4.3 Load Command

The Load command is used to load the program code memory with object code from a disk file. You will be prompted to supply the name of the disk file. The Load screen appears as follows:

```
Enter file name > _
```

---

LOAD CODE MEMORY FROM A FILE

---

The name should be a complete file name specification including a drive specification. Hitting the '?' key in response to the prompt will call the directory facility (see Chapter 3) through which you can get a listing of the entries in any specified directory. Hitting the [RETURN] key in response to the prompt will abort execution of the command. If the file cannot be opened or cannot be found, you will be notified of the error.

Three different object file formats are accepted: standard Intel hex file format, Intel obsolute object module format and ZLINK obsolute object file format. Standard Intel hex files can be created by assembling your program code with most of the currently available 8051 cross assemblers. Intel object module files can be created by linking/locating modules with Intel's RL51 program. These source modules can be either asembled Intel ASM51 object modules or compiled PLM object modules. ZLINK absolute object files are created by the ZLINK 8051 family cross assembler.

The difference between the file formats is that the Intel hex format contains only the object code. The Intel object module format and ZLINK format contain, along with object code, the symbols used in the assembly language file (of PLM file) and can thus be used for symbolic debugging. In addition, ONLY the ZLINK file contains information which allows the use of the more sophisticated breakpoint triggering conditions. These include: opcode values, opcode classes, direct byte addresses, direct byte address ranges, direct bit addresses, direct bit address ranges and immediate operand values. (See chapter 9 for details).

You need not specify which format your object file contains. The system will make that determination. If while reading in the file, an error is encountered, an error message will be displayed. Because these object files are created by computer programs, please ensure that our object file is in an acceptable format.

If you wish to utilize the symbolic debugging capabilities of the system, you may use the ZLINK 8051 family cross assembler with the debug switch to create your object code file. (See the ZLINK 8051 Family Assembler User's Manual for details). The symbolic debugging capability is automatically enabled when you load the ZLINK absolute object code file or an Intel obsolute object code file into the system.

A warning message which may be encountered when executing this command is:

Program memory overflow: the specified file contains code at an address beyond the address limits of the emulator's code memory.

Error messages which may be encountered when executing this command include:

Must establish communication first - the code cannot be downloaded to the emulator before communication with the emulator has been established.

File not found - the specified file could not be found on the specified drive, the default drive, or the A: drive.

File is not proper Intel hex format- the file was being processed as an Intel hex formatted file but some formatting error was encountered while records were being read.

File is not proper absolute object format- the file was being processed as a ZLINK absolute object formatted file but some formatting error was encountered while records were being read.

## 4.4 Upload Command

The Upload command is used to load the program code memory with object code from your target system board. It is assumed that you have memory in the external code memory space and that this memory contains your program object code. You will be prompted to supply the starting address of your code that you wish to upload. (The normal response is 0). The Upload screen appears as follows:

```
UPLOAD PROGRAM CODE MEMORY FROM TARGET BOARD

Enter starting address (in hex) > _
```

Hitting the [RETURN] key in response to the prompt will abort execution of the command. Execution of this command copies program code, starting at the address specified, into the emulator program code memory. The amount of code copied is determined by the size of the program code memory option you have purchased.

PLEASE NOTE that code uploaded from any starting location other than 0 can ONLY be examined in the raw memory mode (see Chapter 10). It cannot be assembled, disassembled or used to run an experiment.

Error messages which may be encountered when executing this command include:

Number is too large- the starting address specified exceeds 64K

Illegal number specification- the address contains an illegal hexadecimal digit.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic addresses are therefore not allowed.

Must establish communication first- an attempt was made to issue a command to the emulator module before communication was established.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS 232 board could not perform a transmission. Check your RS232 board.

## 4.5 Download Command

The Download command is used to download data from a disk file to the user board's external data memory. This command can be used when the external data memory is to be initialized or when the user board's external memory is configured as a Von Neumann type memory. (I.E. both code and data reside in the same memory). If the latter is true, the Download command can be used to write the code into the memory.

You will be prompted to supply the name of the disk file. The Download screen appears as follows:

```
Enter file name >
Are you downloading code memory? (Y/N)
```

---

DOWNLOAD MEMORY FROM A FILE

---

The name should be a complete file name specification including a drive specification. Hitting the '?' key in response to the prompt will call the directory facility (see Chapter 3) through which you can get a listing of the entries in any specified directory. Hitting the [RETURN] key in response to the prompt will abort execution of the command. If the file cannot be opened or cannot be found, you will be notified of the error.

Three different object file formats are accepted: standard Intel hex file format, Intel absolute object module format and ZLINK absolute object file format. You need not specify which format your file contains. The system will make that determination. If while reading the file, an error is encountered, an error message will be displayed.

After entering a valid file name, you will be asked whether or not you are downloading code memory. This is to determine if the external memory on your user board is a Von Neumann type memory. Only a Y or N response will be accepted as a response to this prompt.

A negative response (N) indicates that the data will be used as external data memory only. The emulator's code memory will not be affected by this command. A positive response (Y) indicates that the data is the code for a Von Neumann type memory. In this case, the code will also be loaded into the emulator's code memory and any symbology (if a debug file is loaded) will be loaded into the system's symbol table.

A Warning message which may be encountered when downloading code with this command is:

Program memory overflow- the specified file contains code at an address beyond the address limits of the emulator's code memory.

This message informs you that the emulator's code memory is not big enough to contain the entire code. In any event, ALL code is downloaded to the user board memory, even when this message is encountered.

Error messages which may be encountered when executing this command include:

Must establish communication first- the code cannot be downloaded to the emulator before communication with the emulator has been established.

File not found- the specified file could not be found on the specified drive, the default drive, or the A: drive.

File is not proper Intel hex format- the file was being processed as an Intel hex formatted file but some formatting error was encountered while records were being read.

File is not proper absolute object format- the file was being processed as a ZLINK absolute object formatted file but some formatting error was encountered while records were being read.

## 4.6 Config Command

The Config command calls up the system Configuration menu which allows you to set up the system's configuration as specified by the system configuration file. It also allows you to change the configuration specification in the configuration file. The system configuration specification

includes the baud rate used for communication between the emulator and the host compputer. For the ICD-51 and ICD-52 emulators, it also includes the mode of operation and the configuration of the external data bus (if any).

(See Chapter 5 for a complete description of the Configuration Menu.)

## 4.7 Restore Command

The Restore command is used to restore the system status to some previously saved state. You will be prompted to supply the name of the disk file which contains the saved status. The Restore screen appears as follows:

```
Enter file name > _

-------------------------------------------------

        RESTORE SYSTEM STATUS FROM A DISK FILE

-------------------------------------------------
```

The name should be a complete file name specification including a drive specification. Hitting the '?' key in response to the prompt will call the directory facility (see Chapter 3) through which you can get a listing of the entries in any specified directory. Hitting the [RETURN] key in response to the prompt will abort execution of the command. If the file cannot be opened or cannot be found, you will be notified of the error. The status file has a file format which is recognizable to the system. If the specified file's format is not that of a status file, an error message will be displayed.

Error messages which may be encountered when executing this command include:

File not found- the specified file could not be found on the specified drive, the default drive, or the A: drive.

Illegal System Status File Format- the file specified was not the proper format for a system status file.

Must establish communication first- an attempt was made to issue a command to the emulator module before communication was established.

Communication error - reset commm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 4.8 Store Command

The Store command is used to save the current status of the system in a disk file. You will be prompted to supply the name of the disk file in which you want the status saved, whether or not you want the contents of the emulator's program code memory saved and whether or not you want the contents of the external data memory saved. The Store screen appears as follows:

```
Enter File name > _

-------------------------------------------------

     STORE SYSTEM STATUS IN A DISK FILE

     Save program code memory [Y/N]? > _
     Save external data memory [Y/N] > _
     Enter memory size in KBytes > _

-------------------------------------------------
```

The file name should be a complete file name specification including a drive specification. Hitting the '?' key response to the prompt will call the directory facility (see Chapter 3) through which you can get a listing of the entries in any specified directory. Hitting the (RETURN) key in response to the file name prompt will abort execution of the command. If the file cannot be opened, you will be notified of the error. Parameters which are always stored include:

- Special function registers
- Internal data memory
- The current mappings of the code and external data memories
- The current emulation experiment
- The current PC value
- The name of the OPcode class file being used (if any)
- Any opcode classes which are currently loaded

If you choose to save the program code memory, the parameters which are stored are:

The contents of the emulator's program code memory
All user defined symbols (if symbolic debug is enabled)

If you choose to save the external data memory, you will be prompted for the size of the external data memory in your system. This number should reflect the size of the emulator's external data memory PLUS any external data memory which resides on your target system board. You can therefore save the contents of the entire external data memory space up to 64 KBytes.

Error messages which may be encountered when executing this command include:

Cannot open file- a file could not be opened in the write mode on the specified disk.

Illegal integer value- the number used to specify the size of the external data memory contained an illegal digit.

Number is too large- the number specified for the size of the external data memory in KBytes is greater than 64.

Must establish communication first- communication was never established with the emulator module or was not reestablished after a communications error occurred.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 4.9 Map Command

The Map command is used to select where the program code memory and/or the external data memory will be accessed during an emulation. Each of these memories can be mapped to either the Emulator or to the User Board. The mappings indicates where the emulator module expects to access the memory in question. When the Emulator is selected as the mapping, all memory accesses will be made to the memory supplied as part of the emulator. When the User Board is

selected, all memory accesses will be made to the memory on the user's board regardless of whatever memory is supplied in the emulator. The Map screen appears as follows:

```
Code External-Data Help Quit
(Quick help description for lighlighted command)

                MEMORY MAP MENU

MEMORY                      MAPPING

Program Code Memory         (mapping)
External Data Memory        (mapping)
```

The mapping displayed for each of the memories indicates the current mapping selected. Both memories are mapped into the Emulator upon a cold system startup. The code and External-Data commands are toggle switches which will switch the mappings alternately between the Emulator and the User Board upon each execution of the command.

Please Note that when these memories are mapped to the Emulator, any accesses to addresses beyond the range of the memory options which you have purchased will automatically wrap around onto your user board.

Error messages which may be encountered when executing this command include:

Must establish communication first- communication was never established with the emulator module or was not reestablished after a communications error occurred.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check you RS232 board.

The Help command is used to display the detailed description of the function of each of the commands in the Map Menu.

Error messages which may be encountered when executing the Help command include:

Help file not found- the file "HLPFILE" could not be found on either the default or A: drives.

The Quit command is used to return to the Main Menu.

## 4.10 % (Macro) Command

The % (Macro) command calls up the Macro Menu which allows you to create or execute a macro command file.

(see Chapter 12 for a complete description of the Macro Menu.)

Error messages which may be encountered when executing this command include:

A macro function is currently invoked- macro commands may not be nested. It is therefore illegal to enter the macro menu while in the macro-learn mode.

Must establish communication first- communication must be established before the macro facility can be invoked.

## 4.11 Interrogate Command

The Interrogate command calls up the Interrogate Menu which allows you to run an emulation experiment and inspect and modify the system status. It allows:

- Running an emulation experiment
- Single stepping the target
- Reseting the target
- Setting a phantom breakpoint then running an emulation
- Setting simple breakpoints
- Setting the repetition counter
- Enabling and disabling interrupts
- Examination and modification of registers
- Examination and modification of internal data memory
- Examination and modification of external data memory
- Examination and modification of code memory
- Examination and modification of emulation experiment

(See Chapter 6 for a complete description of the Interrogate Menu.)

Error messages which may be encountered when executing this command include:

Must Establish Communication First- the Interrogate menu cannot be invoked before communcation with the emulator has been established.

## 4.12 Help Command

The Help command is used to display a detailed description of the function of each of the commands in the Main Menu.

Error messages which may be encountered when executing this command include:

Help file not found- the file "$HLPFILE" could not be found on either the default or A: drives.

## 4.13 Exit Command

The exit command is used to terminate a working session. Upon exiting the system, control of the host is returned to the operating system.

When this command is executed, the following prompt will appear at the bottom of the screen:

Are you sure you want to exit [Y/N]?

A 'Y' response exits the system while a 'N' response aborts the exit command. This insures that you can never accidentally exit the system and lose your system status.

CHAPTER 5
CONFIGURATION MENU

## 5.1 Configuration Menu Overview

The Configuration Menu is used to configure the system as specified by the system configuration file. A system configuration file is included as part of the ZLINK system software. The system configuration file is used by the system software to determine how to configure the ICD. Once you specify a configuration (via the Change command), the new system copnfiguration specification becomes the default until it is changed again. You must execute the configuration file when you first enter the ICD environment or whenever you want to change the system's configuration. This is accomplished via the Execute command. the system configuration specification includes the baud rate used for communication specification includes the baud rate used for communication between the ICD emulation module and the host computer. For the ICD-51 and ICD-52 emulators, it also includes the mode of operation and the configuration of the external data bus (if any).

## 5.2 Configuration Menu Screen

The Configuration Menu screen appears as follows:

```
Execute Change Help Quit
(Quick help line for highlighted command)
-------------------------------------------------


                    CONFIGURATION MENU

_____________________________________________________
```

Upon entering, the Execute command will be highlighted.

## 5.3 Execute Command

The Execute command is used to set up the system's configuration according to the specification in the sytem configuration file. This includes establishing a communications link-up between the host and the ICD

module. This only needs to be performed once at the beginning of the session. If no system configuration file exists, the following default specifications are used:

| ICD-31 / ICD-32 | ICD-51 / ICD-52 |
|---|---|
| 9600 baud | 9600 baud<br>single chip mode with NO external bus. |

(See below for a detailed description of the available modes).

Successful execution of this command will establish a communications link-up between the host and the ICD module, set the mode of operation, and return you to the Main Menu.

Error messages which may be encountered when executing this command include:

Must reset emulator to change baud rate- an attempt was made to reconfigure the system's baud rate when communication had already been established at a different baud rate. In this case, reset the emulator module and try again.

RS232 transmission problem - check board - the RS232 board could not perform a transmission. Check your RS232 board.

Emulator not ready - the emulator module did not respond to the host's request to establish communication. Check that power is applied to the emulator module, that it has been reset and that the emulator probe has been supplied with a crystal oscillator.

Communication NOT established - try again - the emulator module responded to the host's request to establish communication but it's response was incorrect. Reset the emulator module and try again.

## 5.4 Change Command

The Change command is used to examine and/or change the specification of the system configuration in the system configuration file. The system configuration specification includes the baud rate used for communication between the ICD module and the host computer. For the ICD-51 and ICD-52 emulators, it also includes the mode of operation and the configuration of the external data bus (if any).

PLEASE NOTE that only the system configuration file is affected by the Change command. The system itself is not reconfigurated to the new configuration specification until the Execute command is executed.

Error messages which may be encountered when executing this command include:

Cannot open file - the sytem configuration file could not be opened in the update mode on the default drive.

The system configuration screen appears as follows for the ICD-31 and ICD-32 emulators:

CHANGE CONFIGURATION

---

SELECT BAUD RATE

| BUAD RATE | TARGET CRYSTAL FREQUENCY |
|---|---|
| 9600 | 8MHz and up |
| 4800 | 4MHz and up |
| 2400 | 2MHz and up |
| 1200 | 1MHz and up |
| 600 | all |
| 300 | all |
| 150 | all |
| 110 | all |

---

Move cursor to desired baud rate then hit [RET] to save it.

Along with each baud rate, the minimum required crystal frequency of the target is displayed. The baud rate which is currently selected as the default in the system configuration file will be highlighted. (If no system configuration file exists, 9600 will be selected as the default).

Along with each operation mode, a brief description of that mode is displayed. The mode which is currently selected as the default in the system configuration file will be highlighted. (If no system configuration file exists, Mode 2 will be selected as the default.) If Mode 3 is selected, the current default for the Port 2 address mask will also be displayed. If Mode 4 is selected, the current rollover

boundary for code memory accesses will be displayed (in the ICD emulator only). (See below for a detailed description of the available modes).

The Change configuration screen allows you to change the default system configuration simply by moving the highlights via the cursor up (↑) or cursor down (↓) keys on the numeric keypad at the right of the keyboard. You must first select which specification (buad rate or mode of operation) you want to change. This is determined by the highlight on the selection titles. Either the SELECT BAUD RATE or the SELECT MODE OF OPERATION will be highlighted to indicate which specification is currently enabled for modification. The selected specification can be changed via the cursor left (←) or cursor right (→) keys on the numeric keypad at the right of the keyboard.

When the correct specification title is selected, you can change that specification by moving the highlight up or down.

PLEASE NOTE that the microcontroller part number in the Mode Description will actually reflect the part number for the ICD you have purchased.

The available modes of operation are described below:

### 5.4.1 Mode 1 - 8031 or 8032 Operation

In this mode of operation, the ICD is configured to operate as a ROMless version of the microcontroller. All code accesses are made to external code memory. Port 0 is used as the multiplexed low order address/data bus. Port 2 is used as the high order address bus.

### 5.4.2 Mode 2 - 8051 or 8052 with NO external address bus

In this mode of operation, the ICD is configured to operate as a ROM version of the microcontroller. All code memory accesses are made to internal code memory. Port 0 and port 2 are used as I/O ports. In this mode, external code memory and external data memory accesses are not permitted.

### 5.4.3 Mode 3 - 8051 or 8052 with an 8 bit external address bus

In this mode of operation, the ICD is configured to operate as a ROM version of the microcontroller. All code memory accesses are made to internal code memory. Port 0 is used as the multiplexed address/data bus. Port 2 is used as an I/O port.

In this mode of operation, the ICD is configured to operate as a ROM version of the microcontroller. All code memory accesses are made to internal code memory. Port 0 is used as the multiplexed address/data bus. Port 2 is used as an I/O port.

The ICD also provides another capability when configured in this mode. If the external data memory is mapped to the emulator (see chapter 4.9), the Port 2 pins can be individually configured as I/O pins or as address lines for the emulator's external data memory. When Mode 3 is selected, the Change configuration screen appears as follows:

```
                    CHANGE CONFIGURATION

-------------------------------------------------------

    SELECT BAUD RATE              SELECT MODE OF OPERATION

 BAUD RATE  TARGET XTAL FREQ.    MODE   MODE DESCRIPTION
 ---------  ---------------     -----  ----------------
   9600       8MHZ and up       Mode 1 8031 Operation
   4800       4MHz and up       Mode 2 8051 W/0 ext addr bus
   2400       2MHz and up       Mode 3 8051 W/8 bit ext addr bus
   1200       1MHz and up       Mode 4 8051 W/16 bit ext addr bus
    600           all
    300           all
    150           all
    110           all

                                 Port 2 address mask = xx

                                 Enter new addr mask > _
                                       (in hex)

_____________________________________________________________
Move cursor to set up configuration hit [RET] to save it.
```

The Port 2 address mask is provided with this mode in order to specify which Port 2 pins are to be used as address lines and which are to be used as I/O pins.

The individual bits of the address mask register have a 1:1 relationship with the bits of Port 2 as shown below:

| | MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|---|
| Mask | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
| Port 2 | P2.7 | P2.6 | P2.5 | P2.4 | P2.3 | P2.2 | P2.1 | P2.0 |

A 0 in the address mask register configures the corresponding Port 2 bit as an I/O line. A 1 in the address mask configures it as an address line.

For example: suppose you are running a program which requires 4K of external data memory. In addition you would like to use 4 bits of Port 2 as I/O lines. This can be accomplished by setting the Port 2 address mask to 0fh. The 0 in the upper nibble configures the four MSBs of Port 2 to be I/O lines. The F in the lower nibble configures the four LSBs of Port 2 to be upper four bits of the 12 bit address bus.

PLEASE NOTE that care must be taken when specifying which bits of Port 2 are to be used as address lines. In an ICD with an 8k external data memory option, bits P2.0 through P2.4 may be used as address lines. In an ICD with a 16K external data memory option, bits P2.0 through P2.5 may be used as address lines.

When Mode 3 is selected, you will be prompted for a new Port 2 address mask. Entering a new value will update the Port 2 address mask to the new value. When a new value has been accepted, you will again be prompted for a new mask value. If the updated value is correct, you may change the specification selection to SELECT BAUD RATE, or you may enter [RETURN]. This will save the new configuration specification in the system configuration file.

Error messages which may be encountered when entering a new Port 2 address mask include:

Number is too large- the number specified was greater 64K.

Illegal number specification- a non-hexadeciamal character was found in the mask specification

Illegal Port 2 address mask- the mask specification was too large to fit in a byte wide register or the mask specification began with non-numeric character.

### 5.4.4 Mode 4 - 8051 or 8052 with a 16 bit External Address Bus

In this mode of operation, the ICD is configured to operate as a ROM version of the microcontroller. Any code memory accesses for addresses 0 up to 4K (ICD-51) or 8K (ICD-52) are made to internal code memory. Any code memory accesses above those values are made to external code memory. Port 0 is used as the multiplexed low order address/data bus. Port 2 is used as the high order address bus.

The ICD-52 emulator also provides another capability when configured in this mode. The rollover boundary can be changed from 8K to 4K to allow emulation of an 8051. When Mode 4 is selected in a ICD-52 emulator, the Change configuration screen appears as follows:

CHANGE CONFIGURATION

```
SELECT BAUD RATE                SELECT MODE OF OPERATION

BAUD RATE  TARGET  XTAL  FREQ.   MODE     MODE DESCRIPTION
---------  ------------------    -----    ----------------
  9600      8MHz and up          Mode 1 8032 Operation
  4800      4MHz and up          Mode 2 8052 w/ NO ext address bus
  2400      2MHz and up          Mode 3 8052 w/ 8 bit ext addr bus
  1200      1MHz and up          Mode 4 8052 w/ 16 bit ext addr bus
   600         all
   300         all
   150         all
   110         all

                                 Rollover boundary = 8K
                                 Enter rollover boundary in K > _
                                          (4 or 8)

    Move cursor to set up configuration hit [RET] to save it
```

You will be prompted for a new rollover boundary. Entering a new value will update the rollover boundary display to the new value. Only 4 or 8 (the K is implied) will be accepted. When a new value has been accepted, you will again be prompted for a new rollover boundary. If the updated boundary is correct, you may change the specification selection to SELECT BAUD RATE, or you may enter [RETURN]. This will save the new configuration specification in the system configuration file.

Error messages which may be encountered when entering a new rollover boundary include:

Number is too large- the number specified was greater 64K.

Illegal number specification- a non-hexadecimal character was found in the boundary specification.

Illegal rollover boundary- the rollover boundary was not 4K or 8K or the rollover boundary specification began with non-numeric character.

# CHAPTER 6
# INTERROGATE MENU

NOTE: This Chapter 6 applies to ICD units that have the optional ICD-CHEST feature installed.

## 6.1 Interrogate Menu Overview

The Interrogate Menu is used to run emulation experiments and to examine the status of the system. This menu allows:

Running an emulation experiment
Single stepping the target
Resetting the target
Setting a phantom breakpoint then running an emulation
Setting simple breakpoint or trace
Setting the repetition counter
Setting the trace trigger type
Calling the help menu
Returning to the main menu
Examination and modification of registers
Examination and modification of internal data memory
Examination and modification of code memory
Viewing the trace buffer
Examination and modification of the emulation experiment
Selecting the I/O port for trace
Setting the increment pass-count number

## 6.2 Interrogate Menu Screen

The Interrogate Menu screen appears as follows for the ICD-8031, ICD-8031A, and ICD-8051 emulators:

```
Go S-Step Reset Fantom Brk-trace Loop-cnt Trace-Trig Help Quit
Modify-Regs Data Xdata Code View-Trace Experiment I/O Pass-cnt
(Quick help line for highlighted command)

Repetition Counter: (#)                                  Trace Trigger: (Point)
Pass Count:  (#)              INTERROGATE MENU           Port Selected: Port (#)
                                                                      GPR
xx ACC      xx B      xx DPH      xx DPL      xx IE       Bank (#)
xx IP       xx PO     xx P1       xx P2       xx P3       --------
xx PCON     xx PSW    xx SBUF     xx SCON     xx SP       xx R0
xx TCON     xx TH0    xx TL0      xx TH1      xx TL1      xx R1
xx TMOD                                                  xx R2
                                                         xx R3
                                                         xx R4
                                                         xx R5
                                                         xx R6

PC Address = xxxx  DPTR = xxxx  Break Address = xxxx

     Next Instruction: (Disassembled instruction)
                       Target Address= (address hex);
```

For the ICD-8032 and ICD-8052 emulators, it appears as follows:

```
Go S-Step Reset Fantom Brk-trace Loop-cnt Trace-Trig Help Quit
Modify-Regs Data Xdata Code View-Trace Experiment I/O Pass-cnt
(Quick help line for highlighted command)

Repetition Counter: (#)                              Trace Trigger: (Point)
Pass Count:  (#)          INTERROGATE MENU           Port Selected: Port (#)
                                                                   GPR
xx ACC      xx B      xx DPH      xx DPL      xx IE    Bank (#)
xx IP       xx PO     xx P1       xx P2       xx P3    --------
xx PCON     xx PSW    xx RCAP2H   xx RCAP2L   xx SBUF  xx R0
xx SCON     xx SP     xx T2CON    xx TCON     xx TH0   xx R1
xx TL0      xx TH1    xx TL1      xx TH2      xx TL2   xx R2
XX TMOD                                               xx R3
                                                      xx R4
                                                      xx R5
                                                      xx R6
                                                      xx R7

PC Address = xxxx  DPTR = xxxx  Break Address = xxxx

     Next Instruction: (Disassembled instruction)
                       Target Address= (address hex);
```

For the ICD-8051 emulator, it appears as follows:

```
Go S-Step Reset Fantom Brk-trace Loop-cnt Trace-Trig Help Quit
Modify-Regs Data Xdata Code View-Trace Experiment I/O Pass-cnt
(Quick help line for highlighted command)

Repetition Counter: (#)                Trace Trigger: (Point)
Pass Count:  (#)    INTERROGATE MENU   Port Selected: Port (#)

                                                                    GPR
xx ACC    xx ADCON   xx ADDAT   xx B     xx CCEN   xx CCH1   Bank (#)
xx CC11   xx CCH2    xx CCL2    xx CCH3  xx CCL3   xx CRCH   -------
xx CRCL   xx DAPR    xx DPH     xx DPL   xx IEN0   xx IEN1    xx R0
xx IP0    xx IP1     xx IRCON   xx PO    xx P1     xx P2      xx R1
xx P3     xx P4      xx P5      xx PCON  xx PSW    xx SBUF    xx R2
xx SCON   xx SP      xx T2CON   xx TCON  xx TH0    xx TL0     xx R3
xx TH1    xx TL1     xx TH2     xx TL2   xx TMOD               xx R4
                                                             xx R5
                                                             xx R6
   PC Address = xxxx    DPTR = xxxx    Break Address = xxxx    xx R7

      Next Instruction: (Disassembled instruction)
                        Target Address = (address hex);
```

Upon entering, the Go command will be highlighted.

The Interrogate menu is different from other menus in that it has two lines of command options available rather than just one.

The Loop Count is the number of times the Go or Single Step commands will execute. (See the Loop-Count command below for details.)

The special functions registers (SFRs) are displayed in the middle of the screen in alphabetic order. The value contained in each register is displayed followed by the register name.

The two byte registers are displayed below the SFRs. These include the current PC address, the data pointer (DPTR) and the break address. The PC Address is the address of the next instruction to be executed. The DPTR is the concatenation of the DPH and DPL registers.

The Next Instruction is the diassembled assembly language mnemonic of the next instruction to be executed when emulation continues. The disassembled instruction will be displayed with all symbolic replacements made if you are using the symbolic debugging capabilities.

The lines directly below each of the disassembled instructions will display any information referred to by the instruction which is not already displayed on the screen. All registers and memories referred to will have their contents displayed. Examples are presented below:

```
Next Instruction: INC    90H        (direct address)
                         90H-24H

Next Instruction: CPL    91H        (bit address)
                         91H=1;

Next Instruction: DEC    @R0        (indirect address)
                         data mem[3]=45;
```

Note: The value of R0 (03H) is displayed on the screen under the GPRs.

```
Next Instruction: MOVX   A,@DPTR    (ext. data address)
                         xdata mem[245]=33H;
```

Note: The value of DPTR (0245H) is displayed on the screen with the two byte registers.

```
Next Instruction: MOVC   A,@A+PC   (code mem address)
                         code mem[123]=FDH;
     Note: The values of A and DPTR are already displayed
           on the screen.

Next Instruction: LJMP   LABEL     (code targets)
                         target address=1234H;

Next Instruction: CLR    C         (carry flag)
                         cy=0;

Next Instruction: JZ     LABEL     (zero flag)
                         zr-1; target address=55H;
```

The target address will appear when the next instruction is a conditional jump instruction and the condition is true or has been met. The target address is equal to where the jump instruction is going to in a memory.

## 6.3 Go Command

The Go command is used to run an emulation experiment. It can be used to start the target emulation. It can also restart the target emulation after a breakpoint or trace-trigger has been reached. Emulation begins at the program code memory location indicated by the program counter (PC).

Prior to running an experiment, the following conditions must be met:

Communication must be established with the ICD emulator.
If an experiment exists within the environment, it must be compiled.
The PC must contain a legal value.

If any of these conditions are not met, an error message will be displayed and the command will be aborted.

Once emulation has begun, the bottom portion of the screen

is cleared, the GO command is displayed and the WORKING sign appears at the bottom of the screen. It appears as follows:

```
Go S-Step Reset Fantom Brk-trace Loop-cnt Trace-Trig Help Quit
Modify-Regs Data Xdata Code View-Trace Experiment I/O Pass-cnt
(Quick help line for highlighted command)
-----------------------------------------------------------------
Repetition Counter: (#)                        Trace Trigger: (Point)
Pass Count:  (#)          INTERROGATE MENU     Port Selected: Port (#)

                                                                 GPR
xx ACC     xx B      xx DPH     xx DPL     xx IE               Bank (#)
xx IP      xx PO     xx P1      xx P2      xx P3               -------
xx PCON    xx PSW    xx SBUF    xx SCON    xx SP                xx R0
xx TCON    xx THO    xx TLO     xx TH1     xx TL1               xx R1
xx TMOD                                                       xx R2
                                                              xx R3
PC ADDRESS = xxxx    DPTR = xxxx    Break Address = xxxx      xx R7

                              GO
-----------------------------------------------------------------
                           WORKING
```

When a breakpoint or in particular a trace-trigger is reached by the ICD emulator, control will be returned to the host and the Interrogate Menu will be repainted. The condition and time frame by which this control is returned to the host is based on the trace-trigger type chosen.

There are three (3) trace-trigger types available. (See the Trace-trigger command in this chapter.) Depending on which is choosen will determine the point in time when an actual break occurs.

If the START trace-trigger type is choosen it will result in the break occuring when the address is equal to the trace-trigger address plus 4,096 ALE cycles.

If the CENTER trace-trigger type is choosen it will result in the break occurring when the address is equal to the trace-trigger address plus 2,048 ALE cycles.

If the END trace-trigger type is choosen it will result in the break occurring when the address is equal to the trace-trigger address and it will break emulation BEFORE the instruction at the address is executed.

You may wish at some point to force the emulator to break emulation. This may happen if the emulation takes longer than expected to reach a breakpoint and you suspect that emulator is executing code other than what you anticipated; if you realize that you made a mistake and the experiment will emulate forever; or if you simply want to break a very long emulation. In any case, a break condition may be forced on the emulator by pressing the [ESC] key while the emulation is running. When this is done, the emulator breaks and a message is displayed. The message indicates that the break was caused by a host interrupt.

The repetition counter is used with the Go command to execute N breaks before stopping (where N is the repetition count). This is especially useful where a break resides in a loop and you wish to execute N cycles of the loop before stopping. When the repetition counter is used, the count value in the display will decrement each time the break is encountered until its value is 0. At that point the emulation will stop and the Interrrogate Menu will be repainted.

The [ESC] key can be used to abort emulation before the repetition counter has decremented to 0. As described above, the host interrupt message will be displayed.

Two modes of operation are possible when using the repetition counter. They are the Non-Update mode and the Update mode. In the Non-update mode, the register values are updated only AFTER the repetition counter has decremented to 0. In the Update mode, the register display is updated after each breakpoint has been encountered. The update mode slows down the execution of the repetition loop because the screen must be repainted after each breakpoint has been encountered. The Update mode therefore takes about twice as long to execute as the Non-update mode.

CTRL-U is used as a toggle switch to alternate between the Non-update and the Update modes of operation. The system is initialized to the Non-Update mode. The CTRL-U switch can be executed from any menu and remains in effect until it is toggled again or the emulation session is terminated.

The Go command can restart an experiment from the last break. The system automatically performs a single step (see Single Step command below) to get past the break instruction and then executes until a break or trace-trigger is encountered. The single step is performed automatically so that you don't manually have to single step past the break instruction before restarting an experiment.

Error messages which may be encountered when executing this command include:

Must establish communication first- communication was never established with the emulator module or was not reestablished after a communications error occurred.

Cannot execute offset uploaded code- the code in the emulator's program code memory was uploaded with an offset value. Code uploaded in this manner cannot be executed.

Must compile experiment first- an experiment was loaded but the experiment compiler was not executed prior to running the experiment. This is an error because it is the experiment compiler which sets the breakpoints in the emulator.

Must reload code memory first- the experiment was deleted but the breakpoints set by that experiment are still resident and active in the emulator.

Must reinitialize PC value first- the PC value was not reinitialized after halting an emulation via a host interrupt.

Cannot restart - instruction jumps on self- the next instruction is a jump which has itself as the target address.

Cannot restart- instruction calls itself- the next instruction is a call which calls itself.

Code jumps out of range- an attempt was made to restart an experiment at an instruction which may cause the program counter to jump outside the range of the emulator's code memory.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Comm command in the Main Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

BREAK CAUSED BY HOST INTERRUPT - an abnormal break condition was caused by a host generated interrupt. If you have not pressed the [ESC] key to cause this break condition then it was caused by noise on the RS232 link.

## 6.4 S-STEP COMMAND

The Single Step command is used to execute one instruction from the program code and then stop. The instruction executed is the one which the program counter (PC) points to in code memory.

Prior to running and experiment, the following conditions must be met:

> Communication must be established with the ICD emulator. The PC must contain a legal value.

If either of these conditions are not met, an error message will be displayed and the command will be aborted. It is not necessary to have compiled an experiment to single step through program code because a break is automatically set to stop after each instruction is executed.

Once emulation has begun, the center of the display screen is cleared, the S-STEP command is displayed and the WORKING sign appears at the bottom of the screen. It appears as follows:

```
Go S-Step Reset Fantom Brk-trace Loop-cnt Trace-Trig Help Quit
Modify-Regs Data Xdata Code View-Trace Experiment I/O Pass-cnt
(Quick help line for highlighted command)
-------------------------------------------------------------------
Repetition Counter: (#)                Trace Trigger: (Point)
Pass Count:  (#)   INTERROGATE MENU    PORT SELECTED: PORT (#)

                                                        GPR
xx ACC    xx B     xx DPH    xx DPL    xx IE          Bank (#)
xx IP     xx PO    xx P1     xx P2     xx P3          ________
xx PCON   xx PSW   xx SBUF   xx SCON   xx SP          xx R0
xx TCON   xx TH0   xx TL0    xx TH1    xx TL1         xx R1
xx TMOD                                             xx R2
                                                    xx R3
                                                    xx R4
                                                    xx R5
                                                    xx R6
PC Address = xxxx  DPTR = xxxx  Break Address = xxxx xx R7

                          S-STEP

-------------------------------------------------------------------
                          WORKING
```

When a breakpoint or trace-trigger instruction is reached by the ICD emulator, control will be returned to the host and the Interrogate menu will be repainted.

The repetition counter is used with the S-STEP command to execute N instructions before stopping (where N is the repetition count). When the repetition counter is used, the count value in the display will decrement each time an instruction is executed until its value is 0. At that point the emulation will stop and the Interrogate Menu will be repainted. The [ESC] key can be used to abort the S-STEP command before the repetition counter has decremented to 0.

Error messages which may be encountered when executing this command include:

Must establish communication first- communciation was never established with the emulator module or was not reestablished after a communications error occurred.

Cannot execute offset uploaded code- the code in the emulator's program code memory was uploaded with an offset value. Code uploaded in this manner cannot be executed.

Must reinitialize PC value first- the PC value was not reinitialized after halting an emulation via a host interrupt.

Cannot restart - instruction calls itself- the next instruction is a call which calls itself.

Code jumps out of range- an attempt was made to execute an instruction which may cause the program counter to jump outside the range of the emulator's code memory.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Comm command in the Main Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 6.5 Reset Command

The Reset command is used to emulate the target starting from a reset condition. Prior to running an experiment, the following conditions must be met:

> Communication must be established with the ICD emulator. If an experiment exists in the environment, it must be compiled.

If either of these conditions are not met, an error message will be displayed and the command will be aborted.

When this command is executed, you will be queried whether or not the reset comes from your target system board. The screen appears as follows:

```
Go S-Step Reset Fantom Brk-trace Loop-cnt Trace-Trig Help Quit
Modify-Regs Data Xdata Code View-Trace Experiment I/O Pass-cnt
(Quick help line for highlighted command)
-----------------------------------------------------------------
Repetition Counter: (#)                          Trace Trigger: (Point)
Pass Count: (#)          INTERROGATE MENU        Port Selected: Port (#)

                                                        GPR
xx ACC    xx B     xx DPH    xx DPL     xx IE          Bank (#)
xx IP     xx PO    xx P1     xx P2      xx P3          -------
xx PCON   xx PSW   xx SBUF   xx SCON    xx SP           xx R0
xx TCON   xx TH0   XX TL0    xx TH1     xx TL1          xx R1
xx TMOD                                                xx R2
                                                       xx R3
                                                       xx R4
                                                       xx R5
                                                       xx R6
PC Address = xxxx  DPTR = xxxx  Break Address = xxxx   xx R7

Will RESET come from target board [Y/N]? > _
```

---

An affirmative response to this question ('Y' or 'y') indicates that the reset will come from your target system board. This means that the emulator will remain in an 'idle' condition until a reset is received from your system. At that time the emulator performs just as the target would under reset conditions and emulation continues from there. A negative response ('N' or 'n') indicates that you wish for the emulator itself to supply the reset. This would be used primarily in the case where the emulator is being used in the stand alone mode as full speed simulator.

Pressing the [RETURN] key in response to the question will abort execution of the command.

If you have selected the reset to come from your target system board, the following message will be displayed on the screen:

Hit RESET on target board to begin emulation

This is your prompt that it is time to execute the reset on your target system board. Doing so will begin emulation.

If the reset is supplied by the emulator, the RESET command will be displayed on the screen.

For either type of reset, once control has been transferred from the host to the emulator, the WORKING sign will appear and it will continue to flash until a breakpoint or trace-trigger is encountered. It appears as follows:

```
Go S-Step Reset Fantom Brk-trace Loop-cnt Trace-Trig Help Quit
Modify-Regs Data Xdata Code View-Trace Experiment I/O Pass-cnt
(Quick help line for highlighted command)
-------------------------------------------------------------
Repetition Counter:  (#)                    Trace Trigger: (Point)
Pass Count: (#)          INTERROGATED MENU  Port Selected: Port (#)

                                                            GPR
xx ACC    xx B     xx DPH    xx DPL    xx IE               Bank (#)
xx IP     xx PO    xx P1     xx P2     xx P3               -------
xx PCON   xx PSW   xx SBUF   xx SCON   xx SP               xx R0
xx TCON   xx TH0   xx TL0    xx TH1    xx TL1              xx R1
xx TMOD                                                  xx R2
                                                         xx R3
                                                         xx R4
                                                         xx R5
                                                         xx R6
PC Address = xxxx  DPTR = xxxx  Break Address = xxxx     xx R7

                  (Reset type indicator)
-------------------------------------------------------------
                          WORKING
```

When a breakpoint or trace-trigger instruction is reached by the ICD emulator, control will be returned to the host and the Interrogate Menu will be repainted.

The repetition counter is not used with the Reset command.

Error messages which may be encountered when executing this command include:

Must establish communication first- communication was never established with the emulator module or was not reestablished after a communications error occurred.

Cannot execute offset uploaded code- the code in the emulator's program code memory was uploaded with an offset value. Code uploaded in this manner cannot be executed.

Must compile experiment first- an experiment was loaded but the experiment compiler was not executed prior to running the experiment. This is an error because it is the experiment compiler which sets the breakpoints in the emulator.

Must reloade code memory first- the experiment was deleted but the breakpoints set by that experiment are still resident and active in the emulator.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Comm command in the Main Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

BREAK CAUSED BY HOST INTERRUPT- an abnormal break condition was caused by a host generated interrupt. If you have not pressed the [ESC] key to cause this break condition then it was caused by noise on the RS232 link.

## 6.6 Fantom (Phanton) Command

The Phantom command is used to set a phantom breakpoint or trace-trigger and then run an emulation experiment. The phantom break is so named because it remains in effect for only one emulation cycle then it disappears.

Prior to running an experiment, the following conditions must be met:

Communication must be established with the ICD emulator.
If an experiment exists in the environment, it must be compiled.
The PC must contain a legal value.

If any of these conditions are not met, an error message will be displayed and the command will be aborted.

When this command is executed, you will be queried for the address where you want the break to be set. The screen appears as follows:

```
Go S-Step Reset Fantom Brk-trace Loop-cnt Trace-Trig Help Quit
Modify-Regs Data Xdata Code View-Trace Experiment I/O Pass-cnt
(Quick help line for highlighted command)
-----------------------------------------------------------------
Repetition Counter: (#)                  Trace Trigger: (Point)
Pass Count: (#)        INTERROGATE MENU  Port Selected: Port (#)

                                                          GPR
xx ACC    xx B    xx DPH    xx DPL    xx IE              Bank (#)
xx IP     xx P0   xx P1     xx P2     xx P3              ------
xx PCON   xx PSW  xx SBUF   xx SCON   xx SP                xx R0
xx TCON   xx TH0  xx TL0    xx TH1    xx TL1               xx R1
xx TMOD                                                  xx R2
                                                         xx R3
                                                         xx R4
                                                         xx R5
PC Address = xxxx   DPTR = xxxx   Break Address = xxxx   xx R7

Enter Phantom breakpoint address (in hex) or label > _

-----------------------------------------------------------------
```

Pressing the [RETURN] key in response to the prompt will abort execution of the command. The breakpoint address can be supplied as a hexadecimal numeric address or as a program code label (if symbolic debug is enabled).

The Phantom breakpoint capability is used to cause an emulation break at some breakpoint which is NOT a breakpoint in the current experiment. The scope of the Phantom breakpoint is for one emulation cycle only. After a break occurs, (even if it was not the phantom breakpoint which caused the break), the Phantom breakpoint is removed.

An example of the use of the Phantom breakpoint follows: Suppose you have a loop in your program in which no breakpoints have been set by the current experiment.

```
You are here >      instruction     < current breakpoint
              loop: instruction
                    instruction
                        .
                        .
                    JMP loop
             after: instruction
                    instruction
                        .
                        .
                    instruction     < next breakpoint
```

After an emulation, the PC is at the instruction before the start of the loop. You now want to emulate the target until you are at the first PC location after the loop (i.e. at PC location "after:").

One method of achieving this is to continually single step the target until it gets to the desired PC location. If the loop is repeated many times before it terminates, this could take a long time.

An alternative method is to set the Repetition counter with the number of instructions which will be executed before the desired PC location is achieved and then execute the Single Step command. For an intricate program, it may be difficult to determine how many instructions this entails.

The preferred method is to set a Phantom breakpoint at PC location "after:". This achieves the desired results with the least effort.

PLEASE NOTE that the breakpoints which are set by the current experiment are still in effect. If you set a Phantom breakpoint but an experiment breakpoint is reached before the Phantom breakpoint, the Phantom breakpoint has no effect. It is as if you executed the Go command. Remember also that the Phantom breakpoint is removed after the emulation cycle so it cannot be used twice.

After a valid breakpoint address has been supplied, the PHANTOM command will be displayed on the screen. Once control has been trasferred from the host to the emulator, the WORKING sign will appear and it will continue to flash until a breakpoint is encountered. It appears as follows:

```
Go S-Step Reset Fantom Brk-trace Loop-cnt Trace-Trig Help Quit
Modify-Regs Data Xdata Code View-Trace Experiment I/O Pass-cnt
(Quick help line for highlighted command)
-----------------------------------------------------------------
Repetition Counter:  (#)                      Trace Trigger: (Point)
Pass Count:  (#)          INTERROGATE MENU    Port Selected: Port (#)

                                                              GPR
xx ACC     xx B     xx DPH     xx DPL     xx IE             Bank (#)
xx IP      xx PO    xx P1      xx P2      xx P3             _______
xx PCON    xx PSW   xx SBUF    xx SCON    xx SP             xx R0
xx TCON    xx THO   xx TLO     xx TH1     xx TL1            xx R1
xx TMOD                                                    xx R2
                                                           xx R3
                                                           xx R4
                                                           xx R5
                                                           xx R6
PC Address = xxxx    DPTR = xxxx    Break Address = xxxx   xx R7

Enter Phantom breakpoint address (in hex) or label > _

                            PHANTOM
-----------------------------------------------------------------
                            WORKING
```

When a breakpoint instruction is reached by the ICD emulator, control will be returned to the host and the Interrogate Menu will be repainted.

> The repetition counter is not used with the Phantom command.
>
> The Phantom command can be used to restart an experiment from the last breakpoint.

Error messages which may be encountered when executing this command include:

Must establish communication first- communication was never established with the emulator module or was not reestablished after a communications error occurred.

Cannot execute offset uploaded code- the code in the emulator's program code memory was uploaded with an offset value. Code uploaded in this manner cannot be executed.

Must compile experiment first- an experiment was loaded but the experiment compiler was not executed prior to running the experiment. This is an error because it is the experiment compiler which sets the breakpoints in the emulator.

Must reload code memory first- the experiment was deleted but the breakpoints set by that experiment are still resident and active in the emulator.

Must reinitialize PC value first- the PC value was not reinitialized after halting an emulation via a host interrupt.

Number is too large- the address specified was greater than 64K.

Illegal number specification- a non-hexidecimal character was found in the address specification.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic addresses are therefore not allowed.

Improper address segment- the specified symbol was not an address symbol for program code memory.

Undefined symbol- the symbol specified doesn't exist.

Address is out of range- the specified address was outside the valid address range for the ICD emulator's program code memory.

Cannot restart - instruction jumps on self- the next instruction is a jump which has itself as the target address.

Cannot restart - instruction calls itself- the next instruction is a call which calls itself.

Code jumps out of range- an attempt was made to restart an experiment at an instruction which may cause the program counter to jump outside the range of the emulator's code memory.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Comm command in the Main Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

BREAK CAUSED BY HOST INTERRUPT- an abnormal break condition was caused by a host generated interrupt. If you have not pressed the [ESC] key to cause this break condition then it was caused by noise on the RS232 link.

## 6.7 Brk-cnt (Break-Count) Command

The Break-Count command is used to set and reset eight simple break or trace-triggers and eight simple increment pass count addresses. They are called simple break / trace-triggers and increment pass-count addresses because they can be used without having to compile an experiment. These simple break / trace-triggers and increment pass-count conditions are, however, restricted to PC addresses.

When a simple break / trace-trigger address is encountered the system will start tracing and depending on the trace-trigger type choosen, will break emulation then (END trigger choosen) or after 2,048 ALE cycles (CENTER trigger choosen) or after 4,096 ALE cycles (START trigger choosen).

When a increment pass-count address is encountered the system will increment the pass-count internally. When the internal pass-count equals the value the user assigned to it, the system will start tracing and depending on the trace-trigger type choosen, will break emulation then (END trigger choosen) or after 2,048 ALE cycles (CENTER trigger choosen) or after 4,096 ALE cycles (START trigger choosen).

When the Break-Count command is executed the system will initially enter into the Break / Trace-trigger points.

## 6.8 Break/Trace-Trigger Points

The Break / Trace-trigger points screen appears as follows:

```
                                        Trace-Trigger Type= (Pointer)

Break address = Trace-Trigger address
-------------------------------------------------------------------
          SET SIMPLE BREAK / TRACE-TRIGGER POINTS

               BREAK /
          TRACE-TRIGGER   ADDRESS  (HEX)
          -------------   --------
                1          0033
                2          0001
                3
                4          00F0
                5
                6
                7          0120
                8

             Enter Address > _
          (in Hex or label)

-------------------------------------------------------------------
[↓]-down [↑]-up [DEL]-DELETE [RET]-Exit [ESC]-Toggle Inc/Break
```

The top of the screen is used to display the trace-trigger type pointer choosen, either START, CENTER or END and the break address is equal to statement. If the START trace-trigger type was choosen then the break address is equal to the trace-trigger address plus 4,096 ALE cycles.

If the CENTER trace-trigger type was choosen then the break address is equal to the trace-trigger address plus 2,048 ALE cylces.

If the END trace-trigger type was choosen then the break address is equal to the trace-trigger address.

The center of the screen is used to display the status of the eight simple breaks. If any of the breaks are set, the break address is displayed as a hexadecimal number. Any break which are not set, are displayed as a 'blank' field on the screen.

The highlight is used to identify which break is to be operated on. Only one break can be operated on at a time. When the Break-Count command is first invoked, the first break address is highlighted.

A different break can be selected for modification by moving the highlight to the desired break. The highlight is moved through the use of the cursor control keys on the numeric keypad at the right of the keyboard. The cursor movement control keys operate as follows:

(↑) - up
(↓) - down

The upward movement of the highlight is limited by the first break address. The downward movement of the highlight is restricted by the last break address.

Break addresses are modified by moving the highlight to the desired break and then entering a new address. When you begin to enter a new address, it will appear in the address field following the prompt. The new address will not be entered into the selected break until the [RETURN] key is pressed. This allows you to correct any errors in the new address before it is entered into the break address field. Use the backspace key to erase characters from the newly entered address.

New break addresses can be entered as hexadecimal numbers or as labels which have been defined as code memory address symbols.

When the [RETURN] key is entered, the new address is examined to determine if it is a valid address. If it is not a valid address, an error message will be displayed and the old value of the break will remain unchanged. After the new address has been processed; the break setting will be updated with the new address and the highlight will be moved down to the next break address field.

Pressing the DELETE key alone will delete the break address at the selected break.

Pressing the [RETURN] key alone will exit the break / trace trigger screen and return to the interrogate menu.

Pressing the ESCAPE key will toggle the system between the break / trace-trigger points and the simple increment pass-count points.

## 6.9 Increment Pass-Count Points

The Increment Pass-count points screen appears as follows:

```
------------------------------------------------------------
              SET SIMPLE INCREMENT COUNT POINTS

          INCREMENT COUNT   ADDRESS  (HEX)
          ---------------   -------------
                  1          0033
                  2          0001
                  3
                  4          00F0
                  5
                  6
                  7          0120
                  8

                    Enter Address > _
                  (in Hex or label)

------------------------------------------------------------
 [↓]-down [↑]-up [DEL]-DELETE [RET]-Exit [ESC]-Toggle Inc/Break
```

The center of the screen is used to display the status of the eight simple increment pass-counts. If any of the increment passcounts are set, the increment pass-count address is displayed as a hexadecimal number. Any increment pass-counts which are not set, are displayed as a 'blank' field on the screen.

The highlight is used to identify which increment pass-count is to be operated on. Only one increment pass-count can be operated on at a time. When the Break-count command is first invoked and the ESCAPE key pressed, the first increment passcount address is highlighted.

A different increment pass-count can be selected for modification by moving the highlight to the desired increment pass-count. The highlight is moved through the use of the cursor control keys on the numeric keypad at the right of the keyboard. The cursor movement control keys operate as follows:

(↑) - up
(↓) - down

The upward movement of the highlight is limited by the first increment pass-count address. The downward movement of the highlight is restricted by the last increment pass-count address.

Increment pass-count addresses are modified by moving the highlight to the desired increment pass-count and then entering a new address. When you begin to enter a new address, it will appear in the address field following the prompt. The new address will not be entered into the selected increment pass- count until the [RETURN] key is pressed. This allows you to correct any errors in the new address before it is entered into the increment pass-count address field. Use the backspace key to erase characters from the newly entered address.

New increment pass-count addresses can be entered as hexadecimal numbers or as labels which have been defined as code memory address symbols.

When the [RETURN] key is entered, the new address is examined to determine if it is a valid address. If it is not a valid address, an error message will be displayed and the old value of the break will remain unchanged. After the new address has been processed; the increment pass-count setting will be updated with the new address and the highlight will be moved down to the next increment pass-count address field.

Pressing the DELETE key alone will delete the increment pass-count address at the selected increment pass-count.

Pressing the [RETURN] key alone will exit the increment pass-count screen and return to the interrogate menu.

Pressing the ESCAPE key will toggle the system between the break / trace-trigger points and the simple increment pass-count points.

PLEASE NOTE that when symbolic addresses are entered as the address of a breakpoint, the symbol is replaced by the represented hexadecimal address.

Error messages which may be encountered when executing this command include:

Must establish communication first - communication was never established with the emulator module or was not reestablished after a communicant error occurred.
Error messages which may be encountered when specifying a new breakpoint address include:

Number is too large- the address specified was greater than 64K.

Illegal number specification- a non-hexadecimal character was found in the address specification.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic addresses are therefore not allowed.

Improper address segment- the specified symbol was not an address symbol for code memory.

Undefined symbol- the symbol specified doesn't exist.

Address is out of range- an address was specified which is outside the addressable range of the program code memory.

Error messages which may be encountered when exiting from the Break Point screen include:

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Comm command in the Main Menu.

RS232 transmission problem - check board- the RS232 board could not perform a tranmission. Check your RS232 board.

## 6.10 Loop-cnt (Loop-Count) Command

The Loop-count command is used to set the value of the repetition counter. The repetion counter is used to determine how many times the Go or S-Step commands are executed before emulation stops. You will be prompted to supply the count value. The Loop-count screen appears as follows:

```
Go S-Step Reset Fantom Brk-trace Loop-cnt Trace-Trig Help Quit
Modify-Regs Data Xdata Code View-Trace Experiment I/O Pass-cnt
(Quick help line for highlighted command)
-------------------------------------------------------------------
Repetition Counter: (#)                        Trace Trigger: (Point)
Pass Count: (#)          INTERROGATE MENU      Port Selected: Port (#)

                                                              GPR
xx ACC   xx B    xx DPH   xx DPL    xx IE                    Bank
xx IP    xx PO   xx P1    xx P2     xx P3                    ------
xx PCON  xx PSW  xx SBUF  xx SCON   xx SP                    xx R0
xx TCON  xx TH0  xx TL0   xx TH1    xx TL1                   xx R1
xx TMOD                                                     xx R2
                                                            xx R3
                                                            XX R4
                                                            XX R5
                                                            xx R6
PC Address = xxxx  DPTR = xxxx  Break Address = xxxx        xx R7

Enter count (in decimal) > _
-------------------------------------------------------------------
```

The count should be decimal number between 1 and 32,767. Pressing the [RETURN] key in response to the count prompt will abort exection of the command and retain the current value of the counter.

Error messages which may be encountered when executing this command include:

Illegal integer value- the number used to specify the count contained an illegal decimal digit.

Number is too large- a count value greater 32,767 was requested.

## 6.11 Trace-trig (Trace-Trigger) Command

The trace-trigger command is used to select which of the three trace-trigger types the user would like to implement and thus a view of the trace buffer and its contents. Pressing the trace-trigger command will toggle the trace trigger portion of the interrogate menu screen from START to CENTER to END.

If the START trace-trigger type is choosen it will result in a break occurring when the address is equal to the trace-trigger address plus 4,096 ALE cycles. This will allow the user to view the next 4,096 memory locations and their contents. The system will NOT execute those instructions and therefore any program path alterations (e.g. JUMPS) will not be performed.

If the CENTER trace-trigger type is choosen it will result in a break occurring when the address is equal to the trace-trigger address plus 2,047 ALE cycles. This will allow the user to view the last 2,047 memory locations and their contents. This will enable the user to determine the instruction path the program has just taken and therefore determine if the correct instructions and conditions are being met. In addition, it will allow the user to view the next 2,048 memory locations and their contents. The sytem will NOT execute those instructions and their contents. The system will NOT execute those instructions and therefore any program path alterations (e.g. JUMPS) will not be performed.

If the END trace-trigger type is choosen it will result in a break occurring when the address is equal to the trace-trigger address and it will break emulation BEFORE the instruction at the address is executed. This will allow the user to view the last 4,096 memory locations and their contents. This will enable the user to determine the instruction path the program has just taken and therefore determine if the correct instructions and conditions are being met.

## 6.12 Help Command

The Help command is used to display a detailed description of the function of each of the commands in the Interrogate Menu.

Error messages which may be encountered when executing this command include:

Help file not found- the file "&HLPFILE" could not be found on either the default or A: drives.

## 6.13 Quit Command

The Quit command is used to return to the main menu.

## 6.14 Modify-Regs Command

The Modify-Regs command is used to examine and/or modify the contents of the target's special function registers (SFRs) and general purpose registers (GPRs). When this command is executed, the register screen will be displayed.

```
                    EXAMINE / MODIFY REGISTERS

 Register name or hex address > _                    Bit Display OFF
 ------------------------------------------------------------------

                                                                 GPR
 xx ACC    xx B    xx DPH   xx DPL    xx IE                     Bank (#)
 xx IP     xx P0   xx P1    xx P2     xx P3                     ------
 xx PCON   xx PSW  xx SBUF  xx SCON   xx SP                     xx R0
 xx TCON   xx TH0  xx TL0   xx TH1    xx TL1                    xx R1
 xx TMOD                                                      xx R2
                                                              xx R3
                                                              xx R4
                                                              xx R5
                                                              xx R6
 PC Address = xxxx   DPTR = xxxx   Break Address = xxxx       xx R7

 Value (in hex) > xx

 New Value (in hex) > _
 ------------------------------------------------------------------
```

All SFRs and GPRs are displayed in the middle of the screen. The value contained in each register is displayed followed by the register name. Two 2-byte registers are displayed: the PC and the DPTR. (The DPTR is a read only register. It is actually the concatenation of the DPH and DPL register. It's value can be changed only by changing the values in the DPH or DPL registers.)

When emulation of an experiment is halted by manual intervention (i.e. pressing the [ESC] key - see section 5.3), the PC value will show its value as xxxx. The PC value remains indeterminate until a new value is assigned to it.

At the top of the display, you are prompted for the name or address of the register you wish to change. Pressing the [RETURN] key in response to the prompt will abort the register screen and return you to the Interrogate Menu. Pressing the [ESCAPE] key in response to the prompt will toggle the Bit Display status between ON and OFF (see Bit Display below). When the Bit Display status is OFF, you are in the byte register mode.

In the byte register mode, register specification can be supplied as hexadecimal addresses or the name of the register. This includes the general purpose registers in the selected bank (i.e. R0 through R7). When a valid register specification has been supplied, its current value is displayed as a hexadecimal value. Pressing the [RETURN] key in response to the prompt retains the current value and returns you to the register specification prompt. Entering a new value will change the contents of the specified register to the new value.

When a new value has been accepted, you are again prompted for a register specification.

PLEASE NOTE that the values for the ports which are displayed represent the actual values at the port pins and not the value in the port registers. Care must be taken when changing the values of the ports. If any changes are made to the port values, all input pins which are to remain inputs must have their corresponding bits set to 1. This restriction comes about because the ICD emulator must write to the port register in order to change the value of any port output pins.

PLEASE NOTE that although the Port0 register is provided for completeness in the 8031 and 8032 emulator, changing it's contents is not meaningful when using these emulators. The actual Port0 register is destroyed when Port0 is used as the external address/data bus.

If the Bit Display status is ON, you are in the bit register mode. This mode is used to modify individual bits within bit addressable registers. If the name or address of a non-bit addressable register is supplied while in the bit register mode, the register is displayed as described in the byte register mode.

Bit addressable register specifications may be supplied as a hexadecimal address or the name of the register. In addition, specifying the name of any bit in a bit addressable register will also serve to specify that register. A bit addressable register display appears as follows:

```
                    EXAMINE / MODIFY REGISTERS

Register name or hex address > scon                  Bit Display ON
-------------------------------------------------------------------

                                                          GPR
xx ACC    xx B    xx DPH    xx DPL    xx IE             Bank (#)
xx IP     xx P0   xx P1     xx P2     xx P3             ------
xx PCON   xx PSW  xx SBUF   xx SCON   xx SP               xx R0
xx TCON   xx TH0  xx TL0    xx TH1    xx TL1             xx R1
xx TMOD                                                 xx R2
                                                        xx R3
                                                        xx R4
                                                        xx R5
                                                        xx R6
PC Address = xxxx   DPTR = xxxx   Break Address = xxxx  xx R7

       SM0    SM1    SM2    REN    TB8    RB8    TI     RI
Value > 0      0      1      0      1      0      1      0

New Bit Value > _
-------------------------------------------------------------------
```

When a valid bit addressable register specification has been supplied, its current value is displayed as individual bits. If the bits of the register have symbolic names, these names will be displayed along with the bit values.

You are then prompted for a new bit value. This bit value is actually a specification of the bit you wish to modify, and whether you wish to set or reset the bit. Pressing the [RETURN] key in response to the prompt retains the current value and returns you to the register specification prompt.

Bits can be specified as a bit name or as a bit number. Valid bit numbers are 0 through 7 and apply to the bit display as the rightmost bit being bit 0 and the leftmost bit being bit 7. (E.G. for the SCON register in the figure above, RI is bit 0 and SM0 is bit 7).

Specifying a bit will set its value to 1. Specifying a bit with a '/' before it will reset its value to 0. (E.G. RI will set the RI bit while /SM2 will reset the SM2 bit).

When a new bit value has beeen accepted, you are again prompted for a new bit value.

Error messages which may be encountered when executing the Registers command include:

Must establish communication first- communication with the ICD emulator module must be established before the register can be examined.

Error messages which may be encountered when specifying a register include:

Number is too large- the number specified was greater than 64K.

Illegal number specification- a non-hexadecimal character was found in the address specification.

Address is out of range- the address specified was greater than 255.

Illegal symbol type- the name specified was not that of a special function register.

Undefined symbol- the name specified doesn't exist.

Illegal register address- the hexadeciamal address provided does not correspond to a special function register.

Error messages which may be encountered when specifying a PC value include:

Number is too large- the number specified was greater than 64K.

Illegal number specification- a non-hexadecimal character was found in the value specification.

Illegal symbol type- the symbol specified was not a numeric symbol.

Undefined symbol- the symbol specified doesn't exist.

Error messages which may be encountered when specifying a value in the byte register mode include:

Illegal number specification- a non-hexadecimal character was found in the number specification.

Number is too large- the value specified was greater than 255.

Illegal symbol type- the symbol specified was not a numeric symbol.

Undefined symbol- the symbol specified doesn't exist.

Error messages which may be encountered when specifying a value in the bit addressable register mode include:

Number is too large- the number specified was greater than 64K.

Illegal number specification- a non-hexadecimal character was found in the number specification.

Illegal bit designator- the value specified was greater than 7.

or

the symbol specified was not a bit symbol.

Undefined symbol- the symbol specified doesn't exist.

Illegal bit for specified register- the symbolic bit specified is not a bit in the specified bit addressable register.

Error messages which may be encountered when reading or writing GPRs include:

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Comm command in the Main Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 6.15 Data-Mem Command

The Data-mem command calls up the Examine/Modify Internal Data Menu which allows you to examine and/or modify the contents of the target's internal data memory. It allows:

- Dumping a block of the memory's contents
- Scanning and modifying the memory a byte at a time
- Filling a block of the memory with data
- Moving a block of the memory's contents from one location to another.
- Searching the memory for a data pattern
- Verify/Compare one block of memory data with another
- Examine and modifying the directly addressable bits which are mapped to the internal-data memory space.

(See Chapter 8 for a complete description of the Examine/Modify Internal-Data Menu.)

## 6.16 Xdata-Mem Command

The Xdata-Mem command calls up the Examine/Modify External-Data Menu which allows you to examine and/or modify the contents of the target's external data memory. It allows:

- Dumping a block of the memory's contents
- Scanning and modifying the memory a byte at a time
- Filling a block of the memory with data
- Moving a block of the memory's contents from one location to another.
- Searching the memory for a data pattern
- Verify/Compare one block of memory data with another.
- Mapping the external-data memory to the user system, the emulator or to both in varying 16-byte block addresses.

(See Chapter 8 for a complete description of the Examine/Modify External-Data Menu.)

## 6.17 Code-Mem Command

The Code-Mem command calls up the Examine/Modify Program Code Memory Menu which allows you to examine and modify the ICD's program code memory, wish to examine. It allows:

- Disassembly of the program code
- Single line assembly of the program code

Examination and modification of raw program code memory data.
Mapping the external-data memory to the user sytem, the emulator or to both in varying 16-byte block addresses.

(See Chapter 7 for a complete description of the Examine/Modify Program Code Memory Menu.)

## 6.18 View-trace Command

The View-trace command is used to examine the contents of the 4,096 frames of the trace buffer. The trace can be examined in two possible modes; the Code mode and the Raw mode. Pressing the view-trace command calls the trace buffer in from the emulator and displays it on the screen, as such there is a time elaspe before the screen is updated with the trace buffer. This time elaspe is based on the communication transfer rate (baud rate).

Before entering into a discussion about the trace buffer display it will be necessary to define some terms and contents of the display screen.

Code Mode- The mode of trace display when the content of the trace buffer is fully disassembled including user supplied labels.

Raw Mode- the mode of trace display when the content of the trace buffer is the binary content of the data/address bus for each bus cycle in the Hex format.

Rel Address- Relative Address in the 4k trace buffer. The value can range from 0 (zero) to 4,096. The + (plus) and - (minus) sign before the number indicate either forward, +, into the trace buffer and as such what is the memory content of what the next N locations in memory. Or backward, -, into the trace buffer and as such the audit trail of what instructions the microcontroller has just executed. In the Code Mode the relative address will increment or decrement its count according to the start address of an instruction. In the Raw Mode the relative address will increment or decrement its count by one to correspond to the data content.

Abs Address- Absolute Address of the microcontroller instruction, address or data in the program flow.

Label- the label for that particular instruction, address or data that was assigned to it in the program development.

Mnemonic Instruction- the mnemonic instruction breakdown of that particular absolute address in the program flow.

Port- The Hex display of the activity on the port selected by the I/O command in the interrogate menu.

Data- the Hex display of the data content of the absolute address in the program flow.

Asteriks- A single asterik (*) indicates when an instruction was executed (Raw Mode only).
- A double asterik (**) indicates when an interrupt occurred (Both Modes).

Where the trace buffer begins to display the data is based on the trace-trigger type choosen. The trace buffer will always enter trace display in the Code Mode.

## 6.19 START Trace-Trigger

If the START trace-trigger type was choosen it will result in the break occurring when the address is equal to the trace-trigger address plus 4,096 ALE cycles. The screen will display relative address 0 and forward (+) addresses. The user may move forward in the trace buffer up to 4,096 relative addresses. The display will always start halfway down.

If the break was set to occur on address 30H the Code Mode would appear as follows:

```
                          Trace Menu
 Trigger Type: Start                          Trace Mode: Code
-------------------------------------------------------------

  Rel      Abs
 Address  Address     Label      Mnemonic Instruction     Port

  0000     0030       start:      MOV DPTR,#0              FF
 +0003     0033       outerloop:  CLR outbit               FF
 +0005     0035                   MOV tempcount, #10       FF
 +0008     0038       innerloop:  CALL wastetime           FF
 +000B     003B                   CPL  outbit              FF
-------------------------------------------------------------
[↑]- scroll up  [↓]- scroll down  [ESC]- change mode [RET]- Exit
```

The highlight appears to the left of the relative address 0 (zero) and over the port Hex activity for relative address 0.

Pressing the [RETURN] key will return the user to the interrogate menu.

Pressing the scroll down [⬇] key moves the user a single relative address at a time down into the trace buffer. Pressing the scroll up [⬆] key moves the user a single relative address at a time into the trace buffer.

A faster method of moving in the trace buffer is by entering the relative address desired to be viewed in the highlight area. This is done by pressing a + (plus) or - (minus) sign and an address and then pressing [RETURN]. The trace buffer will then advance or retreat to the address choosen and display the contents. If the address choosen is greater than the valid contents of the trace buffer the buffer will terminate the address at the last valid data location and display that data.

Pressing the [ESC] key will change the mode of display of the trace buffer from Code mode to Raw mode or vice versa. The Raw Mode display of the same program flow appears as follows:

Trace Menu

Trigger Type: Start  Trace Mode: Raw

| Rel Address | Abs Address | Data | Port |
|---|---|---|---|
| 0000 | 0030 | 90* | FF |
| +0001 | 0031 | 00 | FF |
| +0002 | 0032 | 00 | FF |
| +0003 | 0033 | C2* | FF |
| +0004 | 0034 | 90 | FF |

[⬆]- scroll up [⬇]- scroll down [ESC]- change mode [RET]- Exit

The scroll up, scroll down, change mode and exit functions perform the same as in the Code mode.

## 6.20 CENTER Trace-Trigger

If the CENTER trace-trigger type is choosen it will result in the break occurring when the address is equal to the trace-trigger address plus 2,048 ALE cylces. The screen will display relative address 0 and forward (+) and backward (-) addresses. The user moves forward up to 2,048 relative addresses or backwards up to 2,047 relative addresses. The display will always start with relative address 0 (zero) in the middle.

If the break was set to occur on address 38H the Code Mode would appear as follows:

Trace Menu

Trigger Type: Center  Trace Mode: Code

| Rel Address | Abs Address | Label | Mnemonic | Instruction | Port |
|---|---|---|---|---|---|
| -0008 | 0030 | start: | MOV | DPTR,#0 | FF |
| -0005 | 0033 | outerloop: | CLR | outbit | FF |
| -0003 | 0035 | | MOV | tempcount,#10 | FF |
| 0000 | 0038 | innerloop: | CALL | wastetime | FF |
| +0003 | 003B | | CPL | outbit | FF |
| +0005 | 003D | | CLR | A | FF |
| +0006 | 003E | | JNB | outbit, skipover | FF |

[↑]- scroll up [↓]- scroll down [ESC]- change mode [RET]- Exit

The highlight appears to the left of the relative address o (zero) and over the port Hex activity for relative address 0.

Pressing the [RETURN] key will return the user to the interrogate menu.

Pressing the scroll down [↓] key moves the user a single relative address at a time down into the trace buffer. Pressing the scroll up [↑] key moves the user a single relative address at a time into the trace buffer.

A faster method of moving in the trace buffer is by entering the relative address desired to be viewed in the highlight area. This is done by pressing a + (plus) or -

(minus) sign and an address and then pressing [RETURN]. The trace buffer will then advance or retreat to the address choosen and display the contents. If the address choosen is greater than the valid contents of the trace buffer the buffer will terminate the address at the last valid data location and display that data.

Pressing the [ESC] key will change the mode of display of the trace buffer from Code mode to Raw mode or vice versa. The Raw Mode display fo the same program flow appears as follows:

Trace Menu

Trigger Type: Center  Trace Mode: Raw

| Rel Address | Abs Address | Data | Port |
|---|---|---|---|
| -0003 | 0035 | 75* | FF |
| -0002 | 0036 | 64 | FF |
| -0001 | 0037 | 0A | FF |
| 0000 | 0038 | 12* | FF |
| +0001 | 0039 | 00 | FF |
| +0002 | 003A | 50 | FF |
| +0003 | 003B | B2* | FF |
| +0004 | 003C | 90 | FF |

[↑]- Scroll up [↓[- Scroll down [ESC]- change mode [RET]- Exit

The scroll up, scroll down, change mode and exit functions perform the same as in the Code mode.

## 6.21 END Trace-Trigger

If the END trace-trigger type is choosen it will result in the break occurring when the address is equal to the trace-trigger address and it will break emulation BEFORE the instruction at the address is executed. The screen will display relative address 0 and backward (-) addresses. The user may move backwards up to 4,096 relative addresses. The display will always start with relative address 0 (zero) at the bottom.

The END trace-trigger type will also be entered if the user presses the [ESC] key during instruction execution and therefore, causes a host interrupt condition to occur.

If the break was set to occur on address 41H Code Mode would appear as follows:

Trace Menu

Trigger Type: End  Trace Mode: Code

| Rel Address | Abs Address | Label | Mnemonic Instruction | Port |
|---|---|---|---|---|
| -0017 | 0030 | start: | MOV DPTR,#0 | FF |
| -0014 | 0033 | outerloop: | CLR outbit | FF |
| -0012 | 0035 | | MOV tempcount,#10 | FF |
| -0009 | 0038 | innerloop: | CALL wastetime | FF |
| -0006 | 003B | | CPL outbit | FF |
| -0004 | 003D | | CLR A | FF |
| -0003 | 003E | | JNB outbit,skipover | FF |

[↑]- scroll up [↓]- scroll down [ESC]- change mode [RET]- Exit

The highlight appears to the left of the relative address 0 (zero) and over the port hex activity for relative address 0.

Pressing the [RETURN] key will return the user to the interrogate menu.

Pressing the scroll down [↓] key moves the user a single relative address at a time down into the trace buffer. Pressing the scroll up [↑] key moves the user a single relative address at a time into the trace buffer.

A faster method of moving in the trace buffer is by entering the relative address desired to be viewed in the highlight area. This is done by pressing a + (plus) or - (minus) sign and an address and then pressing [RETURN]. The trace buffer will then advance or retreat to the address choosen and display the contents. If the address choosen is greater than the valid contents of the trace buffer the buffer will terminate the address at the last valid data location and display that data.

Pressing the [ESC] key will change the mode of display of the trace buffer from Code mode to Raw mode or vice versa. The Raw Mode display of the same program flow appears as follows:

Trace Menu

Trigger Type: Center  Trace Mode: Raw

| Rel Address | Abs Address | Data | Port |
|---|---|---|---|
| -0007 | 0039 | 00 | FF |
| -0006 | 003A | 50 | FF |
| -0005 | 003B | B2* | FF |
| -0004 | 003C | 90 | FF |
| -0003 | 003D | E4* | FF |
| -0002 | 003E | 30* | FF |
| -0001 | 003F | 90 | FF |
| 0000 | 0040 | 01 | FF |

[↑]- scroll up [↓]- scroll down [ESC]- change mode [RET]- Exit

The scroll up, scroll down, change mode and exit functions preform the same as in the Code mode.

## 6.22 Experiment Command

The Experiment command calls up the Examine Experiment Menu which allows you to examine and/or modify an emulation experiment. In this menu you can:

Edit an experiment
Compile an experiment to set break points
Load an experiment from a disk file
Store an experiment in a disk file
Reset the current experiment
Delete the current experiment
Call the Opcode Class Menu

(See Chapter 10 for a complete description of the Examine Experiment Menu.)

## 6.23 I/O Command

The I/O command allows the user to select which I/O port activity in the emulator is to be used in the trace buffer. The activity on the selected port will be traced during execution of an experiment and can be viewed using the view-trace command. Pressing the I/O command sequencially toggles the port selected number on the interrogate menu screen.

## 6.24 Pass-cnt (Pass-Count) Command

The Pass-count command is used to set the pass counter. The pass counter is used to trigger the tracing capability, which in turn triggers a break condition. The pass-count is initialized to zero (0) and the user must insure that some condition in either the experiment or the program code is incrementing the pass-counter before the pass-count can be set to some other number.

The Brk-cnt command (see break-count section) can be used to set a simple pass-count or the users experiment must contain a pass-count increment statement. Both conditions will enable the pass-count to increment. If no condition exists the pass-counter cannot be set to any value and an error message will appear.

If the conditions are met the pass-count screen will appear as follows:

```
Go S-Step Reset Fantom Brk-trace Loop-cnt Trace-Trig Help Quit
Modify-Regs Data Xdata Code View-Trace Experiment I/O Pass-cnt
(Quick help line for highlighted command)
-----------------------------------------------------------------

Repetition Counter: (#)                      Trace Trigger: (Point)
Pass Count:  (#)          INTERROGATE MENU   Port Selected: Port (#)

                                                              GPR
 xx ACC    xx B    xx DPH    xx DPL    xx IE                Bank (#)
 xx IP     xx P0   xx P1     xx P2     xx P3                ------
 xx PCON   xx PSW  xx SBUF   xx SCON   xx SP                 xx R0
 xx TCON   xx TH0  xx TL0    xx TH1    xx TL1                xx R1
 xx TMOD                                                   xx R2
                                                           xx R3
                                                           xx R4
                                                           xx R5
                                                           xx R6
 PC Address = xxxx   DPTR = xxxx   Break Address = xxxx    xx R7

 Enter count (in decimal) > _
 -----------------------------------------------------------------
```

The count should be a decimal number between 1 and 65,534. Entering a number and the pressing the [RETURN] will update the pass count number on the interrogate menu screen. Pressing the [RETURN] key in response to the count prompt will abort execution of the command and retain the current value of the counter.

Error messages which could be encountered when executing this command include:

Illegal integer value- the number used to specify the count contained illegal decimal digit.

Number is too large- a count value greater 32,767 was requested.

Count not incremented- Pass counter was attempted to be set to some value other than zero (0) and the user has not provided for an increment to the pass-counter.

CHAPTER 7
EXAMINE / MODIFY PROGRAM CODE MEMORY

## 7.1 Examine / Modify Program Code Memory Overview

The Examine / Modify Program Code Memory Menu is used to examine and/or modify the contents of the ICD's program code memory. It allows:

1) disassembly of the program code,
2) single line assembly of the program code,
3) examination and modification of raw code memory data.

## 7.2 Examine / Modify Program Code Memory Screen

The Examine / Modify Program Code Memory screen appears as follows:

```
Disassemble Assemble Table Help Quit
(Quick help line for highlighted command)
------------------------------------------------------

EXAMINE / MODIFY PROGRAM CODE MEMORY

------------------------------------------------------
```

Upon entering, the Disassemble command will be highlighted.

## 7.3 Disassemble Command

The Disassemble command is used to display the contents of the ICD's program code memory as 8051 assembly language mnemonic instructions. You will be prompted to supply the address of the instruction in program code memory where you desire the disassembly to begin. You will also be prompted to supply the number of instructions to disassemble. The disassembly screen will appear differently depending on whether or not you are using the symbolic debugging capability. The Disassembly screen appears as follows for the non-symbolic mode:

```
Enter starting address (in hex) or label > 0 DISASSEMBLER
Enter number of instructions (in decimal) > 8
```

| Address | code | Mnemonic Instuction |
|---|---|---|
| 0000 | 03 | RR A |
| 0001 | F3 | MOVX @R1,A |
| 0002 | 020055 | LJMP 0055H |
| 0005 | 74CD | MOV A,#CDH |
| 0007 | 120055 | LCALL 0055H |
| 000A | 753620 | MOV 36H,#20H |
| 000D | E3 | MOVX A,@R1 |
| 000E | 7603 | MOV @R0,#03H |

The Disassembly screen appears as follows for the symbolic mode:

```
Enter starting address (in hex) or label > 0   DISASSEMBLER
Enter number of instructions (in decimal) > 8
```

| Addr | Code | Label | Mnemonic Instruction |
|---|---|---|---|
| 0000 | 03 | START: | RR A |
| 0001 | F3 | | MOVS @R1,A |
| 0002 | 020055 | | LJMP ENDPFPROG |
| 0005 | 74CD | | MOV A,#CDH |
| 0007 | 120055 | LABEL1: | LCALL ENDOFPROG |
| 000A | 753620 | | MOV NOTE,#20H |
| 000D | E3 | | MOVX A,@R1 |
| 000E | 7603 | | MOV @R0,#03H |

The first prompt in the disassembly screen is for the starting address. Hitting the [RETURN] key in response to this prompt will abort the Disassembly command. The starting address can be supplied as a hexadecimal numberic address or as an instruction label (if symbolic debug is enabled). If the address is supplied as a number, it is important to

ensure that the address corresponds to an instruction boundary. Specifying an address which is in the middle of an instruction will cause erroneous code disassembly.

The second prompt in the disassembly screen is for the number of instructions to disassemble. Hitting the [RETURN] key in response to this prompt will abort the Disassembly command. The number of instructions must be supplied as a decimal number.

Three checks are made prior to displaying the disassembled code. The first check ensures that the ICD's program code memory has been loaded. If code has not been loaded into the memory then the code memory mapping must be mapped to the user board. If neither of these conditions are true, an error message is displayed and the Disassembly command is aborted. In this case, return to the Main Menu to load your program code memory or to change the mapping of the code memory.

The second check is only used when the code memory is mapped to the emulator's code memory. It ensures that the starting address you have specified is within the address range of the emulator code memory. If the address is outside the range of the emulator code memory, an error message is displayed and the disassembly command is aborted. (e.g. the starting address is at 16K and the emulator only contains 8K code memory).

The third check is only used when the code memory is mapped to the emulator's code memory. It ensures that code has been loaded at the starting address you have specified. If code has been loaded but does not exist at the starting address you have specified, an error message is displayed and the disassembly command is aborted. (e.g. your program consumes 2K of memory from 0 to 3FFh and you have asked to disassemble code starting at address 500h.)

If you are using the symbolic debugging capability of the system, symbols will replace numbers in the mnemonic instruction whenever applicable. To be specific, all program code addresses, direct byte addresses and direct bit addresses which have symbolic representations will be displayed as symbols rather than numbers.

The number of instructions specified for disassembly can exceed the display area of one screen. In this case the screen fills up from top to bottom until the display is full. As more instructions are disassembled, the display area is scrolled upward one line and the newest instruction appears on the bottom line of the display area.

You can halt the scrolling action at any time by hitting the [SPACE BAR] key. This puts the system in the single step display mode. After entering the single step display mode, one new disassembled instruction is displayed each time you hit the [SPACE BAR] key. Normal display scrolling can be resumed by hitting ANY key other than the [SPACE BAR] key. The single step display mode is automatically terminated when the Disassembly execution is completed.

Once the disassembly has begun, it can be aborted at any time by hitting the [ESC] key.

If the number of instructions specified takes us beyond the end of the valid program code, the disassembly terminates and the message

*** end of valid code ***

is displayed beneath the disassembly of the last valid instruction. This end of valid code can be determined ONLY when code is mapped to the emulator's code memory.

Upon completion of the disassembly, the top of the screen returns to the Examine / Modify Program Code memory command list while the disassembly in the center of the screen remains.

Error messages which may be encountered when specifying the starting address include:

Number is too large- the address specified was greater than 64K.

Illegal number specification- a non-hexadecimal character was found in the address specification.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic addresses are therefore not allowed.

Improper address segment- the specified symbol was not a code address symbol.

Undefined symbol- the symbol specified doesn't exist.

Error messages which may be encountered when specifying the number of instructions to disassemble include:

Illegal integer value- the number specified contained non-decimal characters.

Error messages which may be encountered during the code disassembly include:

Code memory is not loaded; the ICD's program code memory has not been loaded with program code and the code memory is mapped to the emulator. Return to the Main Menu to load the code.

Address is outside loaded range- the specified starting address is not within the address range of the emulator code memory.

No code at specified memory location- the specified starting address does not contain valid code.

Sync - possible table disassembly- this error only applies when in the symbolic debug mode. Advantage is taken of the fact that the instruction labels must lie on instruction boundaries. If the disassembly passes a known label without displaying it, then an address synchronization problem exists. Either the starting address was not at an instruction boundry or the disassembly process passed through a non-instruction portion of code memory (i.e. a data table that lies in code memory space).

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 7.4 Assemble Command

The Assemble command is used to overwrite an 8051 instuction in the program code memory with a new instruction, to add a label at a specified PC address or both. You will be prompted to supply the address at which you want to start overwritting code. The Assembly screen will appear as follows:

```
Enter starting address (in hex) or label > _     ASSEMBLER
```

---

---

If the code memory is mapped to the user board, the warning message

WARNING: CODE MEMORY IS MAPPED TO USER BOARD

will be displayed. If this message appears, the Assembly command should be continued ONLY if the memory on your user board is configured as a Von Neumann type memory. (i.e. code and data reside in the same memory).

The prompt is for the address at which you want to start overwriting code or adding a label. Hitting the [RETURN] key in response to this prompt will abort the Assembly command. The address can be supplied as a hexadecimal numeric address or as an instruction label (if symbolic debug is enabled). If the address is supplied as a number, it is important to ensure that the address corresponds to an instruction boundary. Specifying an address which is in the middle of an instruction will cause erroneous code assembly.

Once a starting address has been supplied, you will be prompted to supply the instruction. In addition, the address

of the next instruction to be assembled is displayed. The assembly screen will appear as follows:

```
Next instruction address: (address)        ASSEMBLER
Enter next instruction > _
-------------------------------------------------------

-------------------------------------------------------
```

Hitting the [RETURN] key in response to this prompt will return you to the first prompt which allows you to specify a new starting address. The new instruction must be supplied in the form of an 8051 assembly language mnemonic instruction. Any errors in the mnemonic instruction will be reported.

The mnemonic instruction can contain a symbolic representations. This includes the ability to define a new label at the address specified as the Next Instruction Address. A symbol can be added as a label to the current instruction by entering a label in response to the prompt with no instruction following it.

```
Enter next instruction:  (address)        ASSEMBLER
Enter next instuction > _
```

---

---

Hitting the [RETURN] key in response to this prompt will return you to the first prompt which allows you to specify a new starting address. The new instruction must be supplied in the form of an 8051 assembly language mnemonic instuction. Any errors in the mnemonic instruction will be reported.

The mnemonic instruction can contain symbolic representations. This includes the ability to define a new label at the address specified as the Next Instruction Address. A symbol can be added as a label to the current instruction by entering a label in response to the prompt with no instruction following it.

```
Enter next instruction > Label:
```

Labels can even be added when symbolic debug is not enabled. When this occurs however, the symbolic debug capability becomes enabled.

If the assembled mnemonic instruction does not contain the same number of bytes as the original instruction at the specified location, you will be warned by the message:

New instruction length <> original length - Replace (Y/N)?

If this situation occurs, you have the option to either continue with the instruction replacement (a 'Y' or 'y' response) or to abort the replacement and retain the original instuction (a 'N' or 'n' response).

Please note that it is highly recommended that you inspect our program code carefully before running an experiment if you get this warning message. It is informing you that your code is not contiguous.

Single line assembly is permitted even when the code memory is mapped to the user board. In this case, it is assumed that the memory on the user board is used as a Von Neumann type memory. (i.e. code and data reside in the same memory). This allows you to change the code which resides in the RAM on your board. If however, it is determined that the code memory on the user board is READ ONLY, an error message will be displayed and the Assembly command will be aborted. This error message is:

User board code memory is READ ONLY - could not modify code memory on the user board.

Upon completion of the single line assembly, the Next Instruction Address is incremented to the next location in code memory following the instruction which was just assembled. This allows you to enter a number of continous instuctions in memory without having to specify the address for each instruction.

Error messages which may be encountered when executing the Assemble command include:

Cannot assemble offset uploaded code- code was uploaded from the target system at some starting address other than 0. This code can ONLY be examined using the Table command (see below).

Error messages which may be encountered when specifying the address include:

Number is too large- the address specified was greater than 64K.

Illegal number specification- a non-hexadecimal character was found in the address specification.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic addresses are therefore not allowed.

Improper address segment- the specified symbol was not a code address symbol.

Undefined symbol- the symbol specified doesn't exist.

Error messages which may be encountered when specifying the new assembly language mnemonic instruction include:

Illegal character- a character was encountered which is not part of the legal character set. (See Appendix E for the single line assembler's character set).

Undefined symbol- use was made of a symbol which hasn't been defined.

Duplicate symbol- a label was used in the mnemonic instruction which has a differrent code address value than the address specified in the first prompt.

Illegal opcode after label- the symbol after a label wasn't an opcode.

Illegal assembly line- the assembly line doesn't begin with a label or instruction mnemonic.

Illegal or missing expression- a number, symbol, or arithmetic expression was expected but was either missing or could not be evaluated properly.

Illegal or missing expression operator- an arithmetic operator was expected but was either missing or was not a legal operator. (See Appendix E for the single line assembler's legal operators.)

Unbalanced parenthesis- in evaluating an expression, the parenthesis in the expression were found not to balance.

Illegal or missing expression value- in evaluating an expression, an expected number or symbol was either missing or illegal.

Illegal literal expression- a null ASCII literal string ('') was found.

Expression stack overflow- the expression stack has a depth of 32 values. The expression being evaluated exceeds this depth.

Division by zero- the expression being evaluated includes an attempt to divide by zero.

Illegal bit designator- an illegal bit designator address was specified. A bit designator contains a byte address, followed by a PERIOD, followed by the bit index into the byte address (e.g. ACC.7). This error can be caused by two errors. First, the specification of the byte address part of the bit designator was not a legal bit addressable address. Second, the bit index into the byte address exceeds 7.

Target address exceeds relative address range- a relative jump was specified with the target exceeding 127 bytes forward or 128 bytes backward.

Illegal operand- the operand specified is not a legal operand for the instruction.

Illegal indirect register- the indirect addressing mode designator (@) was followed by something other than R0 and R1. This error can also occur in the MOVC A,@A+DPTR, MOVX A,@DPTR,MOVX @DPTR,A and the JMP @A+DPTR instructions if the operands after the indirect addressing mode designator are not specified properly.

Missing operand delimiter- a COMMA operand delimiter was missing from the operand fields of the instruction.

Expecting an EOL- the assembly language mnemonic instruction supplied contains too many operands.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 7.5 Table Command

The Table command calls up the Examine / Modify Program-Code Menu which allows you to examine and/or modify the contents of the ICD's program code memory as raw data. This mode is useful when working with tables which reside in program code memory. This menu allows:

1) dumping a block of the code memory's contents,
2) scanning and modifying the code memory a byte at a time,
3) filling a block of the code memory,
4) moving a block of the code memory's contents from one location to another,
5) searching the code memory for a data pattern,
6) comparing one block of code memory data with another.

(See Chapter 8 for a complete description of the Examine/Modify Program-Code Menu.)

## 7.6 Help Command

The HELP command is used to display a detailed description of the function of each of the commands in the Examine/Modify Program Code Memory Menu.

Error messages which may be encountered when executing this command include:

Help file not found- the file "HLPFILE" could not be found on either the default or A: drives.

## 7.7 Quit Command

The quit command is used to return to the previous menu.

CHAPTER 8
EXAMINE/MODIFY INTERNAL DATA MEMORY

## 8.1 Examine/Modify Memory Data Overview

The Examine/Modify Memory Data Menu is used to examine and/or modify the contents of the ICD internal data memory, the target system's external data memory and the ICD's program code memory. It allows:

1) dumping a block of the memory's contents,
2) scanning and modifying the memory a byte at a time,
3) filling a block of the memory with data,
4) moving a block of the memory's contents from one location to another,
4) searching the memory for a data pattern,
5) comparing one block of memory data with another,
6) examining and modifying the directly addressable bits which are mapped to the internal data memory space (Internal Data Memory ONLY).

## 8.2 Examine/Modify Memory Data Screen

The Examine/Modify Memory Data screen appears as follows for the External Data Memory and the Program Code Table Memories:

Dump Enter Fill Move Search Compare Help Quit
(Quick help line for highlighted command)

---

EXAMINE / MODIFY (Memory Name) MEMORY

---

The Examine/Modify Internal Data Memory screen appears as follows:

Dump Enter Fill Move Search Compare RAM-Bits Help Quit
(Quick help line for highlighted command)

---

EXAMINE/MODIFY INTERNAL DATA MEMORY

---

Upon entering, the Dump command will be highlighted.

The Examine/Modify Memory Data Menu can be used with internal data memory, external data memory and program code memory. The commands function similarly for all three memory spaces. The memory space being operated on is always displayed on the screen. This will avoid any confusion concerning which memory space is being manipulated.

All of the commands in this menu perform address range checks before performing their functions. These checks ensure that the addresses specified are valid for the selected memory.

For internal data memory, the address range is the entire internal data memory.

| Component | Address Range |
|---|---|
| 8031 | 0 - 7Fh |
| 8032 | 0 - 0FFh |
| 8344 | 0 - 0C0h |

For external data memory the address range is the entire external data memory (i.e. addresses 0 - FFFFh).

For program code memory, the range is dependent on how the Program Code Memory is mapped (see chapter 4.9). If mapped to the ICD, the range is determined by the starting address of the program code memory and the size of the ICD program code memory option you have purchased.

When mapped to the emulator, the starting address of the program code memory is determined by the method of loading the memory. If the memory was loaded via the Load command (see Main Menu), then the starting address

is automatically set to 0. If however, the program code memory is loaded via the Uploaded command (see Main Menu), then the starting address is whatever address was specified in the Upload command.

The range of addressability in the program code memory thus begins at the above described starting address, and continues up to the size of the program code memory option.

For example: Suppose you have uploaded code from your system starting at address 400h (1K) and you have purchased the 8K program code memory option. The addressable range of the program code memory in this case is 400h - 23FFh (1 - 9K).

If mapped externally, the range is the entire program code memory (i.e. addresses 1 - FFFFh).

## 8.3 Dump Command

The Dump command is used to display a block of memory data. You will be prompted to supply the starting address of the block and the number of bytes you want to display. The dump screen appears as follows:

```
Enter starting address (in hex) > 0          DUMP
Enter number of bytes (in decimal) > 24      (Memory Name)

-------------------------------------------------------

  Address          Memory Data              ASCII
  -------    -----------------------        --------
   0000      41 42 43 44 45 46 47 48        ABCDEFGH
   0008      C0 C1 C2 30 C4 C5 C6 35        ...1...5
   0010      61 62 63 64 65 66 67 68        abcdefgh

-------------------------------------------------------
```

The first prompt in the Dump screen is for the starting address. Hitting the [RETURN] key in response to this prompt will abort the Dump command. The starting address can be supplied as a hexadecimal numeric address or as a symbolic address (if symbolic debug is enabled).

The second prompt in the Dump screen is for the number of bytes to dump. Hitting the [RETURN] key in response to this prompt will abort the Dump command. The number of bytes must be supplied as a decimal number.

The number of bytes specified for the dump can exceed the display area of one screen. In this case the screen fills up from the top to bottom until the display area is full. As more bytes are dumped, the display area is scrolled upward one line and a new line of 8 bytes appears on the bottom line of the display area.

You can halt the scrolling action at any time by hitting the [SPACE BAR] key. This puts the system in the single step display mode. After entering the single step display mode, one new group of 8 bytes is displayed each time you hit the [SPACE BAR] key. Normal display scrolling can be resumed by hitting ANY key other than the [SPACE BAR] key. The single step display mode is automatically terminated when execution of the Dump command is completed.

Once the Dump command has begun execution, it can be aborted at any time by hitting the [ESC] key.

If the number of bytes specified goes beyond the end of the addressable range of the memory, all valid data bytes are dumped and then the dump terminates.

Upon completion of the Dump command, you are prompted for another starting address for another dump.

Error messages which may be encountered when specifying the starting address include:

Number is too large- the address specified was greater than 64K.

Illegal number specification- a non-hexadecimal character was found in the address specification.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic addresses are therefore not allowed.

Improper address segment- the specified symbol was not an address symbol for the selected memory space.

Undefined symbol- the symbol specified doesn't exist.

Address is out of range- a program code memory address was specified which is below the program code memory's starting address.

SFR not in internal memory- the special function register specified does not reside in the internal data memory space.

Error messages which may be encountered when specifying the number of bytes to dump include:

Illegal integer value- the number specified contained non-decimal characters.

Error messages which may be encountered during the data dump include:

Address is out of range- the address specified is outside the valid address range of the selected memory.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 8.4 Enter Command

The Enter command is used to examine and/or modify memory one byte at a time. It allows you to step through memory in either incrementing or decrementing address locations. You will be prompted to supply the starting address to examine. The Enter screen appears as follows:

```
                                                    ENTER
Enter starting address (in hex) > 0                 (Memory Name)
```

| Address | Data | New Data |
|---|---|---|
| 0000 | 41 | - |

[↓] - Forward  [↑] - Backward  [ESC] - Exit

If the memory space being operated on is program code memory and the code memory is mapped to the user board, the warning message:

WARNING: CODE MEMORY IS MAPPED TO USER BOARD

will be displayed. If this message appears, the contents of

the memory should be changed ONLY if the memory on your user board is configured as a Von Neumann type memory. (i.e. code and data reside in the same memory).

The first prompt in the Enter screen is for the starting address. Hitting the [RETURN] key in response to this prompt will abort the Enter command. The starting address can be supplied as a hexadecimal numeric address or as a symbolic address (if symbolic debug is enabled).

When a valid starting address has been specified, the address and current value of the memory at that address are displayed. You are then prompted to enter new data. This is called the entry command mode.

Hitting the [↓] cursor control key on the numeric keypad at the right of the keyboard key in response to this prompt will retain the current value, increment the address to the next location and display the address and value of the new location.

Hitting the [↑] cursor control key on the numeric keypad at the right of the keyboard in response to this prompt will retain the current value, decrement the address to the previous location and display the address and value of the new location.
Hitting the [ESCAPE] key will terminate the entry command mode and you will again be prompted for a starting address. This allows you to examine another area of memory.

In addition, while in the Entry command mode, you may supply a new value in response to the prompt. In this case the new data value must be supplied as a hexadecimal number. When a valid data value has been supplied, the address and value will be displayed again to ensure that the value was updated correctly.

Changing data in the program code memory is permitted even when the code memory is mapped to the user board. In this case, it is assumed that the memory on the user board is used as a Von Neumann type memory. (i.e. code and data reside in the same memory). This allows you to change the code which resides in the RAM on your board. If however, it is determined that the code memory on the user board is READ

ONLY, an error message will be displayed and the following error message will be displayed:

User board code memory is READ ONLY

The Enter screen fills up from top to bottom until the display area is full. As more locations are examined, the display area is scrolled upward one line and a address and value appear on the bottom line of the display area.

If incrementing or decrementing the location takes us outside the valid address range of the memory, the address will remain unchanged and the last valid location will be displayed over again.

Error messages which may be encountered when specifying the starting address include:

Number is too large- the address specified was greater than 64k.

Illegal number specification- a non-hexadecimal character was found in the address specification.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic addresses are therefore not allowed.

Improper address segment- the specified symbol was not an address symbol for the selected memory space.

Undefined symbol- the symbol specified doesn't exist.

Address is out of range- a program code memory address was specified which is below the program code memory's starting address.

SFR not in internal memory- the special function register specified does not reside in the internal data memeory space.

Error messages which may be encountered when entering a new data value include:

Too many characters- the new data value contained more than two hexadecimal degits. It can not therefore represent a byte value.

Illegal number specification- the new data value contained digits which were non-hexadecimal characters.

User board code memory is READ ONLY- could not modify the code memory on the user board.

Communication error - reset comm link- a non-recoverable error occured. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 8.5 Fill Command

The Fill command is used to fill a block of memory with a specified data pattern. You will be prompted to supply the starting address of the block and the number of bytes in memory you want filled with data. The Fill screen appears as follows:

```
Enter starting address (in hex) >          FILL
Enter number of bytes (in decimal) > _     (Memory Name)
```

---

```
                Enter Fill Pattern
> _
```

---

If the memory space being operated on is program code memory and the code memroy is mapped to the user board, the warning message

WARNING: CODE MEMORY IS MAPPED TO USER BOARD

Will be displayed. If this message appears, the contents of the memory should be changed ONLY if the memory on your user board is configured as a Von Neumann type memory. (i.e. code and data reside in the same memory).

The first prompt in the Fill screen is for the starting address. Hitting the [RETURN] key in response to this prompt will abort the Fill command. The starting address can be supplied as a hexadecimal numeric address or as a symbolic address (if symbolic debug is enabled).

The second prompt in the Fill screen is for the number of bytes of memory to be filled with data. Hitting the [RETURN] key in response to his prompt will abort the Fill command. The number of bytes must be supplied as a decimal number.

The third prompt in the Fill screen is for the Fill pattern. Hitting the [RETURN] key in response to this prompt will abort the Fill command. The fill pattern is the pattern of data you want to put into memory. The fill pattern data can be specified as numbers, symbolic numbers, character strings, or any combination of the above. A pattern can contain up to 32 bytes of data.

Numbers may be specified as numeric values in any radix. Valid radix specifiers include: h - hexadecimal, d - decimal, o or q - octal and b - binary. No default radix is provided. Specifying a number without a radix specifier will result in an error message.

Symbolic numbers are symbols which have been defined in the currently loaded program to represent numbers. In order to use symbolic number specifications, you must assemble your program with the 8051 Family Cross Assembler with the debug switch on. (See the 8051 Cross Assembler User's Manual for details.)

Character strings are simply strings of characters delineated by an apostrophe ('). In order to include the apostrophe itself in the character string, the double apostrophe ('') is used. Character strings are entered in memory as the ASCII representation of the characters in the string.

Below is an example of a fill patern specification:

```
                Enter Fill Pattern

> 23h 64d symnum 'String'
```

Assuming that symnum is defined to represent the value 10h, the fill pattern specified is:

```
23  40  10  53  74  72  69  6E  67  (all numbers are in hex)
```

Any error s encountered while processing the fill pattern will result in an error message being displayed and an arrow pointing to the offending entry in the fill pattern. Below is an example of a fill patern specification with an error:

```
                    Enter Fill Pattern


> 2rh 64d symnum 'String'

                    ^
ERROR> Illegal number specification - Hit [ESC] to return
```

If the fill pattern specified contains more bytes than was requested by the number of bytes prompt (the second prompt) then the warning message:

```
        WARNING> Fill pattern has been truncated
```

will be displayed. This indicates that the entire fill pattern could not be used to fill the block of memory with data.

If the number of bytes specified exceeds the number of bytes supplied in the fill pattern specification, then the fill pattern will be used over again repeatedly until the requested number of bytes have been filled with data. No warning message will be given in this case.

If the number of bytes specified takes the address beyond the end of the addressable range of the memory, an error message is displayed and the Fill command is aborted without changing data in the memory. You will again be prompted for the starting address so that you can start over.

Changing data in the program code memory is permitted even when the code memory is mapped to the user board. In this case, it is assumed that the memory on the user board is used as a Von Neumann type memory. (i.e. code and data reside in the same memory). This allows you to change the code which resides in the RAM on your board. If however, it is determined that the code memory on the user board is READ ONLY, the following error message will be displayed:

```
        User board  code memory is READ ONLY
```

Upon completion of the Fill command, you are prompted for another starting address for another fill.

Error messages which may be encountered when specifying the starting address include:

Number is too large- the address specified was greater than 64K.

Illegal number specification- a non-hexadecimal character was found in the address specification.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic addresses are therefore not allowed.

Improper address segment- the specified symbol was not an address symbol for the selected memory space.

Undefined symbol- the symbol specified doesn't exist.

Address is out of range- a program code memory address was specified which is below the program code memory's starting address.

SFR not in internal memory- the special function register specified does not reside in the internal data memory space.

Error messages which may be encountered when specifying the number of bytes to fill include:

Illegal integer value- the number specified contained non-decimal characters.

Error messages which may be encountered when specifying the fill pattern include:

Too many bytes in pattern- the fill pattern contained more than 32 bytes of data.

Illegal entry found- an entry was found in the fill pattern which could not be identified as a number, a symbol, or a character string.

Illegal number specification- a non-hexadecimal digit was encountered in the specification of a number.

Illegal digit for specified radix- an illegal digit for the specified radix of a number was encountered.

No default radix provided- a number was specified without a radix specifier.

Number is too large- a number specified either by a numberic value or a symbolic number is greater than OFFh. The value cannot be represented as a data byte.

Undefined symbol- the specified symbol doesn't exist.

Illegal symbol type- the symbol specified was not a numeric symbol.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic numbers are therefore not allowed.

Illegal end of string- the end of the fill pattern was encountered before a terminating character string delimiter was found.

Illegal string character- an apostrophe was found in a character string which was followed neither by another apostrophe nor a space. Such use of an apostrophe within a character string is illegal.

Error messages which may be encountered during the fill process include:

Address is out of range- the address specified is outside the valid address range of the selected memory.

Byte count causes address out of range- the sum of the specified starting address and the specified number of bytes causes an illegal memory address to be generated.

User board code memory is READ ONLY- could not modify the code memory on the user board.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 8.6 Move Command

The Move command is used to move a block of memory from one location in memory to another. You will be prompted to supply the address of the source block, the destination address and the number of bytes you want moved. The Move screen appears as follows:

```
                                        MOVE
                                        (Memory Name)

-------------------------------------------------------------

            Enter Source address (in hex)

            Enter destination address (in hex) > _

            Enter number of bytes (in decimal) > _

-------------------------------------------------------------
```

If the memory space being operated on is program code memory and the code memory is mapped to the user board, the warning message

```
MESSAGE: CODE MEMORY IS MAPPED TO USER BOARD
```

will be displayed. If this message appears, the contents of the memory should be changed only if the memory on your user board is configured as a Von Neuman type memory. (i.e. code and data reside in the same memory).

The first prompt in the Move screen is for the source address. Hitting the [RETURN] key in response to this prompt will abort the Move command. The source address can be supplied as a hexadecimal numeric address or as a symbolic address (if symbolic debug is enabled).

The second prompt in the Move screen is for the destination address. Hitting the [RETURN] key in response to this prompt will abort the Move command. The source address can be supplied as a hexadecimal numeric address or as a symbolic address (if symbolic debug is enabled).

The third prompt in the Move screen is for the number of bytes to move. Hitting the [RETURN] key in response to this prompt will abort the Move command. The number of bytes must be supplied as a decimal number.

If the number of bytes specified to move takes either the source address or destination address beyond the end of the addressable range of the memory, an error message is displayed and the Move command is aborted wihout moving any data in the memory. You will again be prompted for the source address so that you can start over.

Changing data in the program code memroy is permitted even when the code memory is mapped to the user board. In this case, it is assumed that the memory on the user board is used as a Von Neumann type memory. (i.e. code and data reside in the same memory). This allows you to change the code which resides in the RAM on your board. If however, it is determined that the code memory on the user board is READ ONLY, the following error message will be displayed:

User board code memory is READ ONLY

Upon completion of the Move command, you are prompted for another source address for another move.

Error messages which may be encountered when specifying the source and destination addresses include:

Number is too large- the address specified was greater than 64K.

Illegal number specification- a non-hexadecimal character was found in the address specification.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic addresses are therefore not allowed.

Improper address segment- the specified symbol was not an address symbol for the selected memory space.

Undefined symbol- the symbol specified doesn't exist.

Address is out of range- a program code memory address was specified which is below the program code memory's starting address.

SFR not in internal memory- the special function register specified does not reside in the internal data memory space.

Error messages which may be encountered when specifying the number of bytes to move include:

Illegal integer value- the number specified contained non-decimal characters.

Error messages which may be encountered during the move process include:

Source address is out of range- the address specified is outside the valid address range of the selected memory.

Destination address is out of range- the address specified is outside the valid address range of the selected memory.

Byte count causes address out of range- the sum of the specified source or destination address and the specified number of bytes causes an illegal memory address to be generated.

User board code memory is READ ONLY- could not modify the code memory on the user board.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 8.7 Search Command

The Search command is used to search the memory for a specified data pattern. You will be prompted to supply the starting address for the search and the number of bytes in memory you want to search. The Search screen appears as follows:

```
Enter starting address (in hex) >          SEARCH
Enter number of bytes (in decimal)> _      (Memory Name)
```

---

```
                Enter Search Pattern

> _
```

---

The first prompt in the Search screen is for the starting address. Hitting the [RETURN] key in response to this prompt will abort the Search command. The starting address can be supplied as a hexadecimal numeric address or as a symbolic address (if symbolic debug is enabled).

The second prompt in the Search screen is for the number of bytes to search. Hitting the [RETURN] key in response to this prompt will abort the Search command. The number of bytes must be supplied as a decimal number.

The third prompt in the Search screen is for the search pattern. Hitting the [RETURN] key in response to this prompt will abort the Search command. The search pattern is the pattern of data you want to find in memory. The search pattern is specified in exactly the same manner as the fill pattern for the Fill command (see Fill command in this menu).

If the search pattern specified contains more bytes than was requested by the number of bytes prompt ( the second prompt) then an error message will be displayed and the Search command is aborted without searching the memory. You will again be prompted for the starting address so that you can start over.

If the number of bytes specified to search takes the address beyond the end of the addressable range of the memory, an error message is displayed and the Search command is aborted without searching the memory. You will again be prompted for the starting addres so that you can start over.

After all prompts have been successfully entered, the search results are displayed. The search results screen appears as follows:

```
Enter starting address (in hex) >          SEARCH
Enter number of bytes (in decimal) > _     (Memory Name)
```

(Search pattern)

| String found at location(s) |
|---|
| 007E |
| 0081 |
| 0084 |

The search pattern is copied to the first line of the display area. The addresses in memory where the matched pattern have been located are displayed one per line. The address corresponds to the location in memory of the first byte of the pattern. If no matches can be found, the message

*** String Not Found ***

is displayed to inform you of the results of the search.

The number of matching locations can exceed the display area of one screen. In this case the screen fills up from top to bottom until the display area is full. As more matches are found, the display area is scrolled upward one line and a new location appears on the bottom line of the display area.

You can halt the scrolling action at any time by hitting the [SPACE BAR] key. This puts the system in the single step display mode. After entering the single step display mode, one new match location is displayed each time you hit the [SPACE BAR] key. Normal display scrolling can be resumed by hitting ANY key other than the [SPACE BAR] key. The single step display mode is automatically terminated when execution of the Search command is completed.

Once the Search command has begun execution, it can be aborted at any time by hitting the [ESC] key.

Upon completion of the Search command, you are prompted for another starting address for another search.

Error messages which may be encountered when specifying the starting address include:

Number is too large- the address specified was greater than 64K.

Illegal number specification - a non-hexadecimal character was found in the address specification.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic addresses are therefore not allowed.

Improper address segment- the specified symbol was not an address symbol for the selected memory space.

Undefined symbol- the symbol specified doesn't exist.

Address is out of range- a program code memory address was specified which is below the program code memory's starting address.

SFR not in internal memory- the special function register specified does not reside in the internal data memory space.

Error messages which may be encountered when specifying the number of bytes to search include:

Illegal integer value- the number specified contained non-decimal characters.

Error messages which may be encountered when specifying the search pattern include:

Search pattern is larger than search area- there are more bytes in the search pattern than in the specified search area.

Too many bytes in pattern- the search pattern contained more than 32 bytes of data.

Illegal entry found- an entry was found in the search pattern which could not be identified as a number, a symbol, or a character string.

Illegal number specification- a non-hexadecimal digit was encountered in the specification of a number.

Illegal digit for specified radix- an illegal digit for the specified radix of a number was encountered.

No default radix provided- a number was specified without a radix specifier.

Number is too large- a number specified either by a numberic value or a symbolic number is greater than OFFh. The value cannot be represented as a data byte.

Undefined symbol- the specified symbol doesn't exist.

Illegal symbol type- the symbol specified was not a numeric symbol.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic numbers are therefore not allowed.

Illegal end of string- the end of the search pattern was encountered before a terminating character string delimiter was found.

Illegal string character- an apostrophe was found in a character string which was followed neither by apostrophe nor a space. Such use of an apostrophe within a character string is illegal.

Error messages which may be encountered during the search process include:

Address is out of range- the address specified is outside the valid address range of the selected memory.

Byte count causes address out of range- the sum of the specified starting address and the specified number of bytes causes an illegal memory address to be generated.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 8.8 Compare Command

The Compare command is used to compare the data in one block of memory with the data in another block of memory. You will be prompted to supply the starting address of the first block, the starting address of the second block, and the number of bytes in each block you want to compare. The Comparison is performed byte by byte, starting at the beginning of each block and continuing until the specified number of bytes has been compared. The Compare screen appears as follows:

```
Enter block 1 starting address (in hex) > 0     COMPARE
Enter block 2 starting address (in hex) > 10    (Memory Name)

-------------------------------------------------------------

Enter number of bytes (in decimal) > 3

-------------------------------------------------------------
```

The first prompt in the Compare screen is for the starting address of block 1. Hitting the [RETURN] key in response to this prompt will abort the Compare command. The starting address can be supplied as a hexadecimal numeric address or as a symbolic address (if symbolic debug is enabled).
The second prompt in the Compare screen is for the starting address of block 2. Hitting the [RETURN] key in response to this prompt will abort the compare command. The starting address can be supplied as a hexadecimal numeric address or as a symbolic address (if symbolic debug is enabled).

The third prompt in the Compare screen is for the number of bytes to compare. Hitting the [RETURN] key in response to this prompt will abort the Compare command. The number of bytes must be supplied as a decimal number.

After all prompts have been successfully entered, the

comparison results are displayed. The comparison results screen appears as follows:

```
Enter block 1 starting address (in hex) > 0      COMPARE
Enter block 2 starting address (in hex) > 10     (Memory Name)

                        MISMATCHED BYTES

      Block 1  Address Data Data Address  Block 2
               ------- ---- ---- -------
                0000    43   63   0010
                0001    44   64   0011
                0002    45   65   0012
```

All data which does not match in the byte by byte comparison between the two blocks are displayed. The addresses where the mismatched data was found are also displayed. If all data matched, the message

*** All Bytes Match ***

is displayed to inform you of the results of the comparison

The number of mismatched data bytes can exceed the display area of one screen. In this case the screen fills up from top to bottom until the display area is full. As more mismatches are found, the display area is scrolled upward one line and the new mismatched data bytes appear on the bottom line of the display area.

You can halt the scrolling action at any time by hitting the [SPACE BAR] key. This puts the system in the single step display mode. After entering the single step display mode, one new mismatch location is displayed each time you hit the [SPACE BAR] key. Normal display acrolling can be resumed by hitting ANY key other than the [SPACE BAR] key. The single step display mode is automatically terminated when execution of the Compare command is completed.

Once the Compare command has begun execution, it can be aborted at any time by hitting the [ESC] key.

Upon completion of the Compare command, you are prompted for another starting address for another comparison.

Error messages which may be encountered when specifying the starting addresses include:

Number is too large- the address specified was greater than 64K.

Illegal number specification- the symbolic debugging capabilty is not enabled. Symbolic addresses are therefore not allowed.

Improper address segment- the specified symbol was not an address symbol for the selected memory space.

Undefined symbol- the symbol specified doesn't exist.

Address is out of range- a program code memory address was specified which is below the program code memory's starting address.

SFR not in internal memory- the special function register specified does not reside in the internal data memory space.

Error messages which may be encountered when specifying the number of bytes to compare include:

Illegal integer value- the number specified contained non-decimal characters.

Error messages which may be encountered during the comparison process i

Block 1 address is out of range- the address specified is outside the valid address range of the selected memory.

Block 2 address is out of range- the sum of either of the specified starting addresses and the specified number of bytes cuases an illegal memory address to be generated.

Communication error- reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 8.9 RAM-Bits Command (Internal Data Memory Only)

The RAM-Bits command is used to examine and modify the 128 directly addressable bits whose addresses are a subset of the Internal Data Memory's address space. The RAM-Bits appears as follows:

RAM BIT MEMORY

```
                          Address Low Nibble

                    0 1 2 3 4 5 6 7 8 9 A B C D E F
                   \
                0 | x x x x x x x x x x x x x x x x
Address         1 | x x x x x x x x x x x x x x x x
                2 | x x x x x x x x x x x x x x x x
High            3 | x x x x x x x x x x x x x x x x
                4 | x x x x x x x x x x x x x x x x
Nibble          5 | x x x x x x x x x x x x x x x x
                6 | x x x x x x x x x x x x x x x x
                7 | x x x x x x x x x x x x x x x x

Bit name or hex address > _

Value = x

New Value >
```

All RAM bits are displayed in the middle of the screen. The address of any bit can be determined by reading the high order nibble of the address on the left and the low order nibble of the address at the top.

At the bottom of the display, you are prompted for the name or address of the bit you wish to change. Hitting the [RETURN] key in response to the prompt will abort the RAM-Bit screen and return you to the Examine / Modify Internal Data Memory menu.

Bit specifications can be supplied as hexadecimal addresses or the name (symbolic representation) of a bit. When a valid bit specification has been supplied, its current value is displayed. You are then prompted for a new bit value. Hitting the [RETURN] key in response to this prompt retains the current value and returns you to the bit specification

prompt. Entering a new value will change the contents of the specified bit to the new value. Only 0 and 1 are accepted as the new value for a bit.

When a new value has been accepted, you are again prompted for a bit specification.

Error messages which may be encountered when specifying a bit to examine include:

Address is out of range- a bit address was specified which is greater than 7Fh.

Symbolic debug not enabled- the symbolic debugging capability is not enabled. Symbolic bit addresses are therefore not allowed.

Illegal symbol type- the name specified was not that of a bit.

Undefined symbol- the name specified doesn't exist.

Error messages which may be encountered when specifying a new bit value include:

Illegal binary value- the new value supplied was not a 0 or 1.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a transmission. Check your RS232 board.

## 8.10 Help Command

The Help command is used to display a detailed description of the function of each of the commands in the Examine / Modify Memory Data Menu

Error messages which may be encountered when executing this command include:

Help file not found- the file "$HLPFILE" could not be found on either the default or A: drives.

## 8.11 Quit Command

The quit command is used to return to the previous menu.

CHAPTER 9
THE EXPERIMENT

## 9.1 What is an experiment?

Before explaining the concept of an experiment, it is important to first understand the function and use of an in-circuit emulator (ICE). Novice readers are advised to review the ICE concept presented in Chapter 1 before reading further in this chapter.

Below are some definitions which will aid in understanding the concept of an experiment:

Experiment- the specification of where and when in the execution of the program the processor is to stop.

Program Counter- (PC) the pointer to the location in program code memory which will be accessed on the next memory fetch.

Breakpoint- a specific PC location where the processor is to stop executing its program.

Opcode- the first byte (the opcode byte) of an instruction

Direct Byte Address- the address of a special function register or internal data memory location which is accessed via an instruction which contains the address.

Direct Bit Address- the address of a bit which is accessed via an instruction which contains the bit address as one of the operands.

Immediate Value- an operand of an instruction whose value is taken directly from the instruction stream.

The main advantage of using an ICE to debug your design is its ability to stop the execution of the target processor in mid-execution of your program. This allows you to examine the state of your system at any point you specify. This ability can only be utilized to its fullest potential, however, when the mechanism by which you specify breakpoints is flexible and easy to use.

The specification of where breakpoints are to occur is called an experiment. The ICD software allows you to describe an experiment in a high level language (the Experiment Language). An experiment is simply the Experiment Language test which describes where the breakpoints are to occur.

An experiment can be created outside the @ environment by using any available text editor to create an experiment text file. This file can then be read into the ICD system (see Chaper 10 for more details). For your convenience, a full screen experiment editor has been provided as part of the ICD software which allows you to create and modify experiments without leaving the ICD environment. (See Chapter 11 for a full description of the experiment editor.)

## 9.2 Specifying Breakpoints

The Experiment Language uses the if-then conditional statement as its basic construct. Experiment statements will be of the form:

```
if (condition) then break.
```

The condition represents a breakpoint specification. Breakpoints can be specified by any of the following methods:

- A PC address
- A PC address range
- An opcode value
- An opcode class
- A direct byte address
- A direct byte address range
- A direct bit address
- A direct bit address range
- An immediate operand value

Each of these methods for specifying breakpoints will be discussed in detail below. The ICD will break emulation BEFORE the instruction at the breakpoint address is executed.

### 9.2.1 PC Address Breakpoints

A breakpoint can be set at a particular PC address through the use of the following experiment statement:

```
if pc = xx then break
```

This statement will cause an emulation break to occur if the next instruction to be executed resides at location xx in program code memory. The address can be specifed as a numeric address in any radix or as a symbolic code memory address (if symbolic debug is enabled). The address must reside in the ICD's program code memory.

PLEASE NOTE that setting a breakpoint at a PC address which does not contain the opcode of an instruction (i.e. in the middle of an instruction) will cause erroneous and unpredictable operation.

PLEASE NOTE that if you are using the PC address breakpoints to set individual breakpoints irregardless of any other conditions, a simpler method of achieving the same result is to use the simple breakpoint capability via the Brk-Point command in the Interrogate Menu (see chapter 6.7 for details).

### 9.2.2 PC Address Range Breakpoints

A series of breakpoints can be set within a range of PC address through the use of the following experiment statement:

```
if pc comparator xx then break
```

where comparator can be any of the following relational operators: < (less than), > (greater than), <= (less than or equal to) or >= (greater than or equal to). This statement will cause an emulation break to occur if the next instruction to be executed resides within the specified range of locations in program code memory. The address can be speciifed as a numeric address in any radix or as a symbolic code memory address (if symbolic debug is enabled). The address range must reside in the ICD's program code memory.

Bounded ranges may be created by ANDing together two range conditions with the & (logical AND) operator. For example, the experiment statement

```
if pc >= 10h & pc <= 40h then break
```

will cause an emulation break to occur if the next instruction to be executed resides within locations 10h and 40h inclusive in program code memory. (See Section 9.3 for a full description of a complex conditional statement.)

PLEASE NOTE that care must be taken to ensure that the beginning address of a PC address range must fall on an instruction opcode. Beginning a breakpoint range at a PC address which does not contain the operand of an instruction (i.e. in the middle of an instruction) will cause erroneous and unpredictable operation.

### 9.2.3 Opcode Value Breakpoints

A breakpoint can be set for all occurrences of a particular instruction through the use of the following experiment statement:

```
if opcode = xx then break
```

This statement will cause an emulation break to occur every time the instruction with the opcode value xx is the next instruction to be executed. The opcode can be specified as a numeric value in the radix. Only instances of the opcode which reside in the ICD's program code memory will cause emulation breaks to occur.

PLEASE NOTE that if your program code memory contains tables, the ICD's program code memory MUST be loaded with the ICD object code file (see ICD 8051 Cross Assembler User's Manual for details) in order to use the opcode class breakpoint capability. If your code contains tables, using the opcode class breakpoint with any other cross assembler may cause erroneous and unpredictable operation.

WARNING: the opcode class breakpoint capability WILL NOT operate properly on code loaded from Intel absolute object module file.

### 9.2.4 Opcode Class Breakpoints

A breakpoint can be set for all occurrences of a class of instructions through the use of the following experiment statement:

```
if opcode class (name) then break
```

This statement will cause an emulation break to occur every time an instruction defined in the opcode class (name) is the next instruction to be executed. The opcode classes can be defined to encompass as many or as few instructions as desired. Examples of opcode classes could include: any instruction which affects the stack, any instruction which moves data to the accumulator, or any instruction which changes the flow of the program. (See Chapter 12 for a full description of the opcode calss capability.) Only instances of instuctions in the opcode class which reside in the ICD emulator's program code memory will cause emulation breaks to occur.

PLEASE NOTE that if your program code memory contains tables, the ICD emulator's program code memory MUST be loaded with a ZLINK object code file (see ZLINK 8051 Cross Assembler User's Manual for details) in order to use the opcode class breakpoint capability. If your code contains tables, using the opcode class breakpoint with any other cross assembler may cause erroneous and unpredictable operation.

WARNING: The opcode class breakpoint capability WILL NOT operate properly on code loaded from Intel absolute object module file.

### 9.2.5 Direct Byte Address Breakpoints

A breakpoint can be set for all occurrences of instructions which access (either read or write) a directly addressable register or internal memory location. This is accomplished through the use of the following experiment statement:

```
if daddr = xx then break
```

where 'daddr' stands for 'Direct Address'. This statement will cause an emulation break to occur every time an instruction which accesses the specified direct address is the next instruction to be executed. The address may be specified as a numeric value in any radix or as a symbolic direct address (if symbolic debug is enabled). Only instances of the direct address instuctions which reside in the ICD's code memory will cause emulation breaks to occur.

It is also possible to limit breakpoints to only reads from or only writes to a direct address. This is accomplished by creating a complex conditional (see the section on complex conditionals below) with a direct address specification and an opcode class specification (see Chapter 12 for full details on opcode classes). The direct address specification determines the address and the opcode class specification determines whether read or write access will set a breakpoint. The experiment statement would appear as follows:

If daddr - xx & opcode class (class name) then break

This will limit breakpoints to only read or only write accesses to the specified register except for the move direct address to direct address instruction (MOV daddr,daddr). This instruction accesses two direct addresses either of which can set a breakpoint for either read or write accesses.

PLEASE NOTE that if your program code memory contains tables, the ICD's program code memory MUST be loaded with a ICD object code file (see ICD 8051 Cross Assembler User's Manual for details) in order to use the direct address breakpoint capability. If your code contains tables, using the direct address breakpoint with any other cross assembler may cause erroneous and unpredictable operation.

WARNING: The direct address breakpoint capability WILL NOT operate properly on code loaded from Intel absolute object module file.

### 9.2.6 Direct Byte Address Range Breakpoints

A breakpoint can be set for all occurrances of instructions which access (either read or write) a range of directly addressable register or internal memory location. This is accomplished through the use of the following experiment statement:

if daddr comparator xx then break

where 'daddr' stands for 'Direct Address' and the comparator can be any of the following relational operators: < (less than), > (greater than), <= (less than or equal to). This statement will cause an emulation break to occur every time an instruction which accesses a direct address (within the specified range) is the next instruction to be executed. The

address may be specified as a numeric value in any radix or as a symbolic direct address (if symbolic debug is enabled). Only instances of the direct address instructions which reside in the ICD's program code memory will cause emulation breaks to occur.

Bounded ranges may be created by ANDing together two range conditions with the & (logical AND) operator as described in the PC range breakpoint section.

PLEASE NOTE that if your program code memory contains tables, the ICD's program code memory MUST be loaded with an ICD object code file (see ICD 8051 Cross Assembler User's Manual for details) in order to use the direct address range breakpoint capability. If your code contains tables, using the direct address range breakpoint with any other cross assembler may cause erroneous and unpredictable operation.

WARNING: The direct address range breakpoint capability WILL NOT operate properly on code loaded from Intel absolute object module file.

### 9.2.7 Direct Bit Address Breakpoints

A breakpoint can be set for all occurances of instructions which access (either read or write) a directly addressable bit. This is accomplished through the use of the following experiment statement:

```
if baddr = xx then break
```

where 'baddr' stands for 'Bit Address'. This statement will cause an emulation break to occur every time an instruction which accesses the specified bit address is the next instruction to be executed. The address may be specified as a numeric value in any radix or as a symbolic bit address (if symbolic debug is enabled). Only instances of the bit address instructions which reside in the ICD's program code memory will cause emulation breaks to occur.

PLEASE NOTE that if your program code memory contains tables, the ICD's program code memory MUST be loaded with an ICD object code file (see ICD 8051 Cross Assembler User's Manual for details) in order to use the bit address breakpoint capability. If your code contains tables, using the bit address breakpoint with any other cross assembler may cause erroneous and unpredictable operation.

WARNING: The bit address breakpoint capability WILL NOT operate properly on code loaded from Intel absolute object module file.

### 9.2.8 Direct Bit Address Range Breakpoints

A breakpoint can be set for all occurrances of instructions which access (either read or write) a range of directly addressable bits. This is accomplished through the use of the following experiment statement:

```
if baddr comparator xx then break
```

where 'baddr' stands for 'Bit Address' and the comparator can be any of the following relational operators: < (less than), > (greater than, <= (less than or equal to) or >= (greater than or equal to). This statement will cause an emulation break to occur every time an instruction which accesses bit address (within the specified range) is the next instruction to be executed. The address may be specified as a numeric value in any radix or as a symbolic bit address (if symbolic debug is enabled). Only instances of the bit address instructions which reside in the ICD's program code memory will cause emulation breaks to occur.

Bounded ranges may be created by ANDing together two range conditions with the & (logical AND) operator as described in the PC range breakpoint section.

PLEASE NOTE that if your program code memory contains tables, the ICD's program code memory MUST be loaded with an ICD object code file (see ICD 8051 Cross Assembler User's Manual for details) in order to use the bit address range breakpoint capability. If your code contains tables, using the bit address range breakpoint with any other cross assembler may cause erroneous and unpredictable operation.

WARNING: The bit address range breakpoint capability WILL NOT operate properly on code loaded from Intel absolute object module file.

### 9.2.9 Immediate Operand Value Breakpoints

A breakpoint can be set for all occurrences of a particular immediate operand value through the use of the following experiment statement:

```
if immed = xx then break
```

This statement will cause an emulation break to occur every time an instruction with the immediate operand value xx is the next instruction to executed. The immediate value can be specified as a numeric value in any radix. Only instances of these instructions which reside in the ICD's program code memory will cause emulation breaks to occur.

PLEASE NOTE that if your program code memory contains tables, the ICD's program code memory MUST be loaded with an ICD object code file (see ICD 8051 Cross Assembler User's Manual for details) in order to use the immediate operand value breakpoint capability. If your code contains tables, using the immediate operand breakpoint with any other cross assembler may cause erroneous and unpredictable operation.

WARNING: The immediate operand breakpoint capability WILL NOT operate properly on code loaded from Intel absolute object module file.

## 9.3 Complex Conditional Statements

In the above discussion of address ranges, mention was made of the ability to link together simple conditional statements in order to form more complex ones. This section will explore complex conditionals in more detail.

As discussed above, the & operator (logical AND) is used to join together two conditionals (simple or complex) both of which must be met before an emulation break will occur. For example:

```
if pc < 40h & opcode = 20h then break
```

will cause an emulation break only if an instruction whose opcode value is 20h is executed at any PC location between 0 and 3Fh.

As might be expected, the ! operator (logical OR) is also provided. This operator is used to join together two conditionals (simple or complex) either of which must be met before an emulation break will occur.

The ! and & operators are of equal precedence. Conditional expressions are evaluated from left to right.

Very complex conditionals can be created with the use of the ! and & operators. You may, therefore, use parenthesis to make a conditional expression more readable. The parenthesis are ONLY for readability and do not override the natural precedence of the expression.

## 9.4 Constructing An Experiment

The specification of an experiment can contain as many experiment statements as desired. Each statement specifies its' own set of break conditions. The net effect of specifying an experiment with more than one experiment statement is to logically OR together the break conditions specified by each of the statements. In the following experiment for example:

```
if (condition 1) then break
if (condition 2) then break
if (condition 3) then break
```

any one of the three conditions could cause an emulation break. This is the same as specifying:

```
if (condition 1) ! (condition 2) ! (condition 3) then break
```

Both experiments have the same results. Readability considerations and personal preference will determine how you construct an experiment.

Experiment statements are parsed in free form. Extra spaces and new lines are ignored by the experiment compiler (see chapter 10.4). Experiment statements, therefore, need not be contianed on one line of the experiment file. Statements may take any visual form as long as they are syntactically correct. The only constraint is that the entire experiment be limited to 32 lines of 76 characters. The ICD experiment editor enforces these limitations automatically. If another editor is used to create the experiment, care must be taken to keep the experiment within the required limits.

## 9.5 Experiment Language Syntax Summary

This section presents a summary of the experiment language's syntax.

The experiment language

```
statement :: = If condition THEN result | EOF

condition :: = simp cond | comp cond

simp cond :: = address | opcode | immediate

comp cond :: = simp cond & simp cond | simp cond & comp cond |
               simp cond ! simp cond | simp cond ! comp cond

address   :: = PC compare pcaddr | DADDR compare addr |
               BADDR compare addr

pcaddr    :: = number | symbolic-address |address expression

addr      :: = number | symbolic-address

compare   :: = '=' | '>' | '>=' | '>='

opcode    :: = OPCODE = number | OPCODE CLASS symbolic-class

immediate :: = IMMED = number

result    :: = BREAK
```

CHAPTER 10
EXAMINE/MODIFY EXPERIMENT MENU

## 10.1 Examine/Modify Experiment Menu Overview

The Examine/Modify Experiment Menu is used to examine and/or modify an experiment specification. In this menu you can:

1) edit an experiment,
2) compile an experiment to set the breakpoints,
3) load an experiment from a disk file,
4) store an experiment in a disk file,
5) delete the current experiment,
6) call the Opcode class Menu.

## 10.2 Examine/Modify Experiment Menu Screen

The Examine/Modify Experiment Menu screen appears as follows:

Edit Compile Load Store Delete Opcode Help Quit
(Quick help line for highlighted command)

---

EXAMINE/MODIFY EXPERIMENT

---

Upon entering, the Edit command will be highlighted.

## 10.3 Edit Command

The Edit command calls up the full ICD screen Experiment Editor. This editor allows you to create experiments or edit existing experiments within the ICD environment.

(See Chapter 11 for a complete description of the Experiment Editor.)

## 10.4 Compile Command

The Compile command is used to compile an experiment and to down-load the breakpoints (which are the output of the compilation) to the ICD. The compiler conducts the experiment in two passes. On the first pass, the experiment is checked for syntax errors. On the second pass, the breakpoints are generated and sent to the ICD.

Prior to running the compiler, the following conditions must be met:

1) The program code memory must be loaded with the program to execute.
2) An experiment must exist,
3) If an opcode class is used in the experiment, the opcode class file must be loaded into the system and the disk containing the opcode class file must be in the dirve form which it was originally loaded (the compiler will read the opcode class file).
4) Communication must be established with the ICD.

When the compiler is executed, the compiler screen is displayed. It appears as follows:

EXPERIMENT COMPILER

---

(Experiment Text)

---

The middle of the screen is cleared. As the experiment is parsed during the first pass, the experiment text will begin appearing on the screen. If a syntax error is found during this pass, the offending item will be highlighted, an error message will be displayed and the parsing will stop. The error message will specifically explain the error encountered. You will be prompted to hit the [ESCAPE] key to continue parsing. When parsing continues, the errors thus far encountered remain highlighted.

In the course of parsing an experiment with serious errors, the compiler may become so lost that it must resynchronize itself. In this case it may skip a portion of a statement without checking the syntax of that portion. When this happens, the entire portion will become highlighted and will remain highlighted. This serves to inform that a resynchronization took place.

If any errors were detected on the first pass, the number of errors is reported by the following message:

RESULT> x errors detected

When errors are detected on the first pass, the second pass is aborted so that the errors can be corrected with the Experiment Editor.

If no errors are detected on the first pass, the screen is again cleared and the second pass begins. The experiment test will again appear on the screen as the experiment is parsed.

The second pass generates the breakpoints. After parsing the experiment for the second time, the WORKING sign will appear. This indicates that the breakpoints are being down-loaded to the ICD. If no errors are encountered on the second pass, the message:

RESULT> 0 errors detected

will be displayed. This indicates that the experiment was compiled correctly and that the breakpoints have been properly loaded into the emulator.

Error messages which may be encountered while compiling an experiment include:

Must load experiment first- an attempt has been made to compile an experiment before it has been loaded or created.

Must load program code memory first- an attempt has been made to compile an experiment before the program code has been loaded into the ICD's code memory (see Chapter 4 for details about loading the program code memory).

'if' expected- an experiment statement did not begin with 'if'.

Illegal end of experiment- the end of the experiment file was encountered in the middle of parsing an experiment statement.

'then' expected- the conditional portion of an experiment statement was not terminated by a 'then'.

Illegal conditional- a simple conditional was encountered which was not part of the legal conditional set.

Unbalanced parenthesis- there are an unequal number of opening and closing parenthesis used in the conditional protion of a statement.

Number expected- something other than a numeric value was encountered where a number was expected.

Code address expected- a symbol other than a code address symbol was used in the specification of a code address value.

Comparator expected- something other than a comparator was encountered where a comparator was expected.

Number or address symbol expected- something other than a numeric address or a symbolic address was encountered where an address was expected.

PC value is out of range- the pc address specified in a PC type conditional was outside the addressability range of the ICD's program code memory. (i.e. a breakpoint could not be set at the requested address.)

Address value is too large- an address was specified for a direct byte or direct bit conditional which exceeds OFFh.

'=' or 'class' expected- something other than an '=' or 'class' was used in an opcode type conditional.

Number is too large- the opcode value specified exceeds OFFh.

Undefined opcode class- the symbol used to specify an opcode class was not a valid opcode class symbol.

Could not open opcode class file xxxxx- the opcode class file could not be opened. This is most likely caused by the removal of the disk between the time the opocde class file was loaded and the time the experiment was compiled.

Class xxxxx not found in file xxxxx- the specified opcode class could not be found in the expected opcode class file. This is most likely caused by editing a 2nd opcode class file between the time the 1st opcode class file was loaded and the experiment was compiled.

Illegal result statement- a result other than break was encountered.

Illegal character- a character was encountered which is not part of the legal character set. (See Appendix E for the experiment compiler's character sit.)

or

a non-hexadecimal number was found in the specification of a numeric value.

Undefined symbol- use was made of a symbol which hasn't been defined.

No default radix provided- a number value was specified without a radix specifier.

Illegal digit for specified radix- an illegal digit was encountered for the specified radix of a number.

Number is too large- the number specifed exceeds 64K.

Illegal or missing expression- a number, symbol or arithmetic expression was expected but was missing or could not be evaluated properly.

Code address expected- a symbol other than a code address symbol was used in the specification of a code address value.

Direct byte address expected- a symbol other than a direct byte address symbol was used in the specification of a direct byte address value.

Direct bit address expected- a symbol other than a direct bit address symbol was used in the specification of a direct bit address value.

Operator stack overflow- the operator stack has a depth of 32 values. The expression being evaluated exceeds this value.

Illegal operator- the arithmetic operator specified is not a legal operator.

Illegal bit designator- a illegal bit designator address was specified. A bit designator contains a byte address, followed by a PERIOD, followed by the bit index into the

byte address (e.g. ACC.7). This error can be caused by two errors. First, the specification fo the byte address part of the bit designator was not a legal bit addressable address. Second, the bit index into the byte address exceeds 7.

Communication error - reset comm link- a non-recoverable error occurred. Communication must be reestablished via the Execute command in the Configuration Menu.

RS232 transmission problem - check board- the RS232 board could not perform a trnasmission. Check your RS232 board.

## 10.5 Load Command

The Load command is used to laod a predefined experiment from a disk file. You will be prompted to supply the name of the disk file. The Load screen appears as follows:

```
Enter file name > _

-------------------------------------------------------

                LOAD EXPERIMENT FROM A FILE

-------------------------------------------------------
```

The name should be complete file name specification including a drive specification. Hitting the '?' key in response to the prompt will call the directory facility (see Chapter 3) through which you can get a listing of the entries in any specified directory. Hitting the [RETURN] key in response to the prompt will abort execution of the command. If the file cannot be opened or cannot be found, you will be notified of the error.

The experiment files can be created either by a foreign editor or by executing the Store command in this menu (see below). Any errors encountered while reading an experiment file will be reported. If errors are encountered while reading file, the Load command is aborted.

If an experiment currently exists within the system, you will be prompted whether or not you want to overwrite the

current experiment. (Only one experiment can exist within the system at a time.) The prompt will appear as follows:

Overwrite current experiment (Y/N)? _

A positive response to this prompt ('Y' or 'y') will read the new experiment into the system thereby destroying the current experiment. A negative response ('N' or 'n') will abort the Load command thereby giving you the opportunity to save the current experiment before loading a new one.

Error messages which may be encountered when executing this command include:

File not found- the specified file could not be found on the specified drive, the default drive, or the A: drive.

Illegal character- a non-printable character was encuntered while reading the file.

Line greater than 76 characters found- a line was found in the file which is longer than the legal length.

File has more than 32 lines- the file contains more than the legal number of lines.

## 10.6 Store Command

The Store command is used to store the current experiment in a disk file. You will be prompted to supply the name of the disk file. The Store screen appears as follows:

Enter file name > _

---

STORE EXPERIMENT IN A FILE

---

The name should be a complete file name specification including a drive specification. Hitting the '?' key in response to the prompt will call the directory facility (see Chapter #) through which you can get a listing of the entries in any specified directory. Hitting the [RETURN] key in response to the prompt will abort execution of the command. If the file cannot be opened, you will be notified of the error.

Error messages which may be encountered when executing this command include:

Cannot open file- the specified file could not be opened on the specified drive.

## 10.7 Delete Command

The Delete command is used to delete the current experiment from the system. You will be prompted to ensure that you actually wanted to delete the experiment from the system. (If the experiment is deleted from the system before it is saved in a file via the Store command, it is permanently lost). The Delete screen appears as follows:

```
DELETE CURRENT EXPERIMENT

Are you sure you want to delete the experiment [Y/N]? > _
```

A 'N' response aborts the command without affecting the experiment. A 'Y' response deletes the experiment and resets all breakpoints which were set by the experiment.

## 10.8 Opcode Command

The Opcode command calls up the Opcode Class Menu which allows you to load, create, examine and modify Opcode Class files. (See Chapter 12 for a complete description of the Opcode Class Menu.)

## 10.9 Help Command

The Help command is used to display a detailed description fo the function of each of the commands in the Examine / Modify Experiment Menu.

Error messages which may be encountered when executing this command include:

Help file not found- the file "$HLPFILE" could not be found on either the default of A: drives.

## 10.10 Quit Command

The Quit command is used to return to the previous menu.

CHAPTER 11
EXPERIMENT EDITOR

## 11.1 Experiment Editor Overview

The Experiment Editor is used to create, examine and/or modigy an experiment. The Experiment Editor has three modes of operation. These three modes are: the Line Entry mode, the Edit mode and the Edit Command mode. Each of these is discussed in detail below.

## 11.2 Experiment Editor Screen

The Experiment Editor screen appears as follows:

```
                    EXPERIMENT EDITOR

 Experiment status: (Status)          (Insert status)
_____________________________________________________
1  (line 1 of text)
2  (line 2 of text)
.
.
.
_____________________________________________________
                    (Command options)
```

The experiment status display is always present to inform you of the current status of the experiment being edited. The status can be one of the following:

| | |
|---|---|
| NULL | - no experiment exists (default when editor is invoked and there is no current experiment) |
| UNMODIFIED | - the experiment has not been changed (default when editor is first invoked and there is a current experiment). |
| MODIFIED | - the experiment has been modified during the edit session. |

The Insert status display informs you whether or not you are in the Edit-Insert mode (see Edit mode for details). The display is blank when in the Edit-Replace mode. The display says INSERT when in the Edit-Insert mode.

Line numbers for the lines in the experiment are displayed on the left side of the Edit display.

The command options which are valid for the current mode are displayed at the bottom of the screen.

## 11.3 Using the Experiment Editor

The Experiment Editor is designed for use as a text editor for experiment statements. The legal set of keys which may be used with the editor consists of:

Any of the printable character keys
The [RETURN] key
The [ESCAPE] key
The four cursor control keys on the numberic keypad at the right of the keypad
( )
The [INSERT] key
The [DELETE] key

Use of any other keys (especially CTRL keys and other cursor control keys) will cause strange and unpredictable results.

## 11.4 Line Entry Mode

The Line Entry mode is usde to create a new experiment. It is invoked automatically when the editor is called and no experiment exists. The Edit screen appears as follows in the Line Entry mode:

EXPERIMENT EDITOR

Experiment status: NULL

---

1 _

---

Hit [RET] to terminate line entry mode

When the Line Entry mode is entered, the line number 1 is displayed and the cursor is positioned at the beginning of the line. The command option is displayed at the bottom of the screen. As printable characters are entered, they will appear as text on the line. Line Entry mode is terminated when a [RETURN] key is hit at the beginning of a new line.

The maximum number of characters permitted on a line is 76. Exceeding that amount will cause another new line to be displayed and any excess characters from the previous line to be entered on the new line. After completing entry of a line, simply hit [RETURN] to advance to the next line.

Line Entry mode will continue until the [RETURN] key is hit at the beginning of a new line or until 32 lines have been entered. If 32 lines have been entered, the Line Entry mode is automatically terminated.

When the Line Entry mode is terminated, the Edit mode is automatically invoked.

## 11.5 Edit Mode

The Edit mode provides the full screen editing capability. It is used to examine and modify an existing experiment. Edit mode is invoked automatically when the editor is called and an experiment exists or when the Line Entry mode is terminated. The Edit screen appears as follows in the Edit mode:

```
                    EXPERIMENT EDITOR
Experiment status: (Status)           (Insert Status)
_____________________________________________________
1 (line 1 of text)
2 (line 2 of text)
.
.
.
_____________________________________________________
              Hit [ESC] to exit edit mode
```

Movement of the cursor on the screen is provided through the use of the cursor control keys on the numeric keypad at the right of the keyboard. The cursor movement control keys operate as follows:

▲ - up
▼ - down
◄ - left
► - right

Wrap around is provided if the cursor is moved past the beginning or end of a line. If at the beginning of a line, a cursor left movement will move the cursor to the end of

the previous line. If at the end of a line, a cursor right movement will move the cursor to the beginning of the next line.

The [DELETE] key is used to delete the character at the cursor position. All subsequent characters will be shifted left one character position. Hitting the [DELETE] key when the curosr is at the end of a line has no effect.

A line can be deleted by hitting the [DELETE] key when the cursor is positioned at an empty line (i.e. a newly created line with no characters or a line which has had all of its characters deleted). This is the only mechanism provided for deleting lines in the Experiment Editor. When a line is deleted, all subsequent lines are shifted upward one line.

The full screen editor operates in two modes: the Edit-Replace mode (default) and the Edit-Insert mode. Both of these modes are discussed below.

The [INSERT] key is a toggle switch which toggles the Edit mode between the Edit-Replace mode and the Edit-Insert mode. When the Edit-Insert mode is selected, the INSERT sign appears at the top right of the Editor display screen.

### 11.5.1 Edit-Replace Mode

The Edit-Replace mode is the default mode for the full screen editor. It is used to overwrite the character at the cursor position. When a character is entered in this mode, it replaces whatever character was originally present. The cursor is then advanced to the next character position. The net effect is to overwrite the old text with new text.

The effect of entering a [RETURN] key in the Edit-Replace mode is to write an end-of-line mark at the cursor position. The previous character at the cursor position and all subsequent characters in the line are deleted. Hitting the [RETURN] key when characters in the line are deleted. Hitting the [RETURN] key when the cursor is at the end of a line has no effect. Hitting the [RETURN] key when the cursor is at the beginning of a line has the effect of deleting all characters from the line (i.e. creating an empty line). This empty line can be deleted if desired by hitting the [DELETE] key.

### 11.5.2 Edit-Insert Mode

The Edit-Insert mode is entered by hitting the [INSERT] key. This mode is active when the insert sign is displayed at the top right of the Edit screen. It is used to insert a new character in front of the character at the cursor position. When a character is entered in this mode, it is placed at the cursor position. The original character at the cursor position and all subsequent characters in the line are shifted one character position to the right. The cursor is then advanced to the next character position.

If the line is full (i.e. contains 76 characters) when an insert is attempted, an error message will be displayed indicating that the line is full.

The effect of entering a [RETURN] key in the Edit-Insert mode is to insert an end-of-line mark at the cursor position. A new line is created following the current line. The original character at the cursor position and all subsequent characters in the original line are moved to the beginning of the newly created line. All lines which follow the original line are shifted downward one line.
If the edit buffer is full (i.e. the experiment contains 32 lines) when a line insertion is attempted, an error message will be displayed indicating that the edit buffer is full.

Error messages which may be encountered while editing in the Edit-Insert mode include:

Line is full- an attempt was mde to insert a character in a line which already contains 76 characters.

Experiment buffer is full- an attempt was made to insert a line in an experiment which already contains 32 lines.

## 11.6 Edit Command Mode

The Edit Command mode is entered by hitting the [ESCAPE] key while in the Edit mode. The Edit Command mode is used to exit from the edit session, abort an edit session or

save an edited experiment without leaving the edit environment. The Edit screen appears as follows in the Edit command mode:

```
                    EXPERIMENT EDITOR
Experiment status: (status           (Insert status)
_____________________________________________________
1 (line 1 of text)
2 (line 2 of text)
.
.
.
_____________________________________________________
       E - edit     S - save     Q - quit
```

The commands available in the Edit Command mode are Edit, Save and Quit. Each is discussed in detail below.

### 11.6.1 Edit Command

The Edit command is selected by hitting 'e' or 'E' while in the Edit Command mode. Execution of this command returns control to the Edit mode (described above).

### 11.6.2 Save Command

The Save command is selected by hitting 's' or 'S' while in the Edit Command mode. Execution of this command saves the current state of the experiment in the system's experiment buffer. This has the effect of overwriting the previous buffer. This has the effect of overwriting the previous (uneditied) verison of the experiment status display at the upper left of the Edit screen is changed to UNMODIFIED. This indicates that the experiment in the edit buffer has not been changed during the current edit session.

### 11.6.3 Quit Command

The Quit command is selected by hitting 'q' or 'Q' while in the Edit Command mode. Execution of this command terminates the edit session. If the experiment has not been modified during the edit session, control will be returned to the Examine / Modify Experiment Menu. If, however, the experiment has been changed during the edit session, you will be prompted whether or not you want to save the editied version of the experiment. The prompt will appear as follows:

Save edited experiment? (Y/N)

A positive reponse ('y' or 'Y') will save the experiment before exiting. A negative response ('n' or 'N') will terminate the edit session without saving the edited experiment (the old pre-edited version will remain intact).

CHAPTER 12
OPCODE CLASS MENU

## 12.1 Opcode Class Menu Overview

The Opcode Class Menu is used to create, examine and/or modify Opcode Class files. In this menu you can:

1) load an Opcode Class file from a disk file,
2) edit an Opcode class disk file.

## 12.2 What is an Opcode Class?

An opcode class is a collection of 8051/8031 instructions which, taken collectively, comprise a set. You define opcode classes for the needs of your particular experiment. Some examples of opcode classes are the predefined opcode classes provided in the file "OPCLASS.OPC" on the @ system disk.

These classes include:

STACK - all instructions which affect the state of the stack (PUSH, POP, ACALL, LCALL, RET, RETI)

PGMFLOW - all instructions which change the normal flow of a program (AJMP, SHMP, LJMP, ACALL, LCALL, RET, RETI, JBC, JB, JNB, JC, JNC, JZ, JNZ, CJNE, DJNZ, JMP @A+DPTR)

The method by which opcode classs are defined is described later in this chapter.

The use of opcode classes is best illustrated by an example. Suppose in debugging your program, you find that the program counter is getting corrupted. An obvious debugging strategy is to break emulation on every instruction which can change the contents of the program counter. You can therefore define the experiment:

```
if opcode = 01h then break   ;AJMP
if opcode = 02h then break   ;LJMP
if opcode = 10h then break   ;JBC
if opcode = 11h then break   ;ACALL
.
.
.
```

A much easier method of setting the same breakpoints is by

using opcode classes. The same breakpoints are set by the experiment.

if opcode class PGMFLOW then break.

As will be discussed below, the Opcode Class Editor provides shorthand methods of specifying instruction attributes so that all instructions with similar attributes can easily be joined together to form an opcode class.

An Opcode Class file is a file containing up to 72 opcode class definitions. All classes in this file need not pertain to any one experiment. It may contain classes which are used for other experiments or even other projects.

The purpose of collecting opcode class definitions together in a file is to encourage the sharing and reuse of opcode classes and to reduce disk file clutter.

## 12.3 Opcode Class Menu Screen

The Opcode Class Menu screen appears as follows:

Load Edit Help Quit
(Quick help line for highlighted command)

---

OPCODE CLASS MENU

---

Upon entering, the Load command will be highlighted.

## 12.4 Load Command

The Load command is used to load the names of opcode classes defined in a specified Opcode Calss file into the system. You will be prompted to supply the name of the disk file.

The prompt will differ slightly depending on whether or not the system is 'remembering' an Opcode Class file which was specified previously.

If no file name is 'remembered', no default exists and the prompt screen appears as shown below. In this case, the filename should be complete file name specification including a drive specification. Hitting the '?' key in response to the prompt will call a listing of the entries in any specified directory. Hitting the [RETURN] key in

If the file cannot be opened or cannot be found, you will be notified of the error. The Load screen will appear as follows:

```
Enter file name > _
```

---

LOAD OPCODE CLASS FILE

---

When a file name is 'remembered', the default is presented as part of the prompt as shown below. In this case, you can hit the [RETURN] key to select the default file or you can enter a complete file name specification including a drive specification. If the file cannot be opened or cannot be found, you will be notified of the error. The Load screen will appear as follows:

```
Enter file name [(default file name)] > _
```

---

LOAD OPCODE CLASS FILE

---

The names of the defined opcode classes must be loaded into the system because they are required by the first pass of the experiment compiler. They are used to determine whether or not an opcode class which is used in an experiment has been defined.

The second pass of the experiment compiler then opens the Opcode Class file and reads the class definitions when it is determining where to set the breakpoints.

PLEASE NOTE it is extremely important that when you are running an experiment compilation, the Opcode Class file's disk MUST be present in the same disk drive from which it was originally loaded. This is because the system 'remembers' where the Opcode Class file was read from and returns there to read the opcode class definitions.

Error messages which may be encountered when executing this command include:

Could not open file - the specified file could not be found on the specified drive, the default drive, or the A: drive.

Illegal opcode class file format - the file specified is not the proper format for an Opcode Class file.

Duplicate symbol - xxxxx- the symbol for an opcode class name has been previously defined.

## 12.5 Edit Command

The Edit command calls the Opcode Class Editor. This editor is used to edit the contents of an Opcode Class file. It allows you to: create a new opcode class, edit an e opcode class or delete an existing opcode class. The Opcode class Editor operates in three distinct modes: the File Prompt mode, the Class Selection mode and the Class Edit mode. Each of these will be discussed in detail below.

### 12.5.1 File Prompt Mode

The File Prompt mode is used to supply the name of the Opcode Class file which is to be manipulated. You will be prompted to supply the name of the disk file.

The prompt will differ slightly depending on whether or not the system is 'remembering' an Opcode class file which was specified previously.

If no file name is 'remembered', no default exists and the file Prompt screen appears as shown below. In this case, the file name should be a complete file name specification including a drive specification. Hitting the '?' key in response to the prompt will call the directory facility (see Chapter 3) through which you can get a listing of the entries in any specified directory. Hitting the [RETURN] key in response to the prompt will abort execution of the command. If the file cannot be opened or cannot be found, you will be notified of the error. The File Prompt screen will appear as follows:

```
Enter file name > _
```

---

OPCODE CLASS EDITOR

---

When a file name is 'remembered', the default is presented as part of the prompt as shown below. In this case, you can hit the [RETURN] key select the default file or you can

specification. If the file cannot be opened or cannot be found, you will be notified of the error. The File prompt screen will appear as follows:

```
Enter file name [(default file name)] > _
```

```
                    OPCODE CLASS EDITOR
```

Error messages which may be encountered when executing this command include:

Could not open file - the specified file could not be found on the specified drive, the default drive, or the A: drive.

Illegal opcode class file format - the file specified is not the proper format for an Opcode Class file.

Opcode class table is full - more than 72 opcode classes are defined in the specified file.

Once all of the opcode class names have been read into the Opcode Class Editor, the Class Selection mode is automatically invoked.

### 12.5.2 Class Selection Mode

The Class Selection mode performs two functions. It is first used to select which opcode class from the specified Opcode Class file will be manipulated. Its second function is to select the operation which will be performed on the selected opcode class. The operations available are: Edit the selected opcode class, delete the selected opcode class, create a new opcode class, or rename the selected opcode class. The class Selection screen appears as follows:

```
                    OPCODE CLASS EDITOR
Opcode Class:  (Class Name)        File: (File name)

     (name1)     (name2)     (name3)     (name4)
     (name5)     (name6) ...

E - edit  D - delete  C - Create  R - Rename  Q - quit
```

The Opcode Class status display in the upper left corner of the Opcode Class Editor screen indicates which opcode class will be manipulated. Upon entry into the Class Selection screen, the opcode class name in the (name1) position is always selected.

The file status display in the upper right corner of the Opcode Class Editor screen indicates which Opcode class file is currently being edited.

The File status display in the upper right corner of the opcode Class Editor screen indicates which Opcode Class file is currently being edited.

The center of the display contains the names of all of the opcode classes defined in the current Opcode Class file. These are displayed in alphabetical order.

The currently selected opcode class is always presented in the Opcode class status display. In addition, the occurrance of the selected class name in the center display will also be highlighted. A different Opcode Class can be selected for manipulated by moving the highlight to the desired opcode class name. The highlight is moved through the use of the cursor control keys on the numeric keypad at the right of the keyboard. The cursor movement control keys operate as follows:

(▲) - up
(▼) - down
(◄) - left
(►) - right

Wrap around is provided if the highlight is moved past the beginning or end of a line. If the first class name on a line is selected, a cursor left movement will move the highlight to the last class name onthe previous line. If the last class name on the line is selected, a cursor right movement will move the highlight to the first class name on the next line.

The command option part of the diplay at the bottom of the screen indicates what commands are available. The legal command set includes the Edit-class command, the Delete-Class command, the Create-Class command, the Rename-Class, and the Quit command. Each of these will be discussed below.

#### 12.5.2.1 Edit-class Command

The Edit-Class command is used to invoke the Class Edit mode. This mode allows you to examine and/or modify the definition of the selected opcode class. The selected opcode class is the one displayed in the Opcode Class status display when the Edit-Class command if executed. See section 12.5.3 for a detailed description of the Class Edit mode.

Error messages which may be encountered when executing this command include:

Could not open temporary workfile - because the file is being editedd, a backup copy is required. In the attempt to open a backup file on the same disk as the Opcode Class file, an error was encountered.

#### 12.5.2.2 Delete-Class Command

The Delete-Class command is used to delete the currently selected opcode class from the Opcode class file. When this command is executed, a prompt will appear at the bottom of the screen. The prompt will display the currently selected opcode class and will ask you if you still want to delete it. The prompt will appear as follows:

Delete opcode class (Class Name) (Y/N)?

A positive response ('y' or 'Y') will delete the selected opcode class from the Opcode Class file and the screen will be repainted with the selected class deleted. A negative response ('n' or 'N') will abort the Delete-Class command.

Error messages which may be encountered when executing this command include:

Could not open temporary workfile - because the file is being edited, a backup copy is required. In the attempt to open a backup file on the same disk as the opcode Class file, an error was encountered.

### 12.5.2.3 Create-Class Command

The Create-Class command is used to add a new opcode class to the Opcode Class file. When this command is executed, you will be prompted for the name of the new opcode class. The prompt will appear as follows:

```
                    OPCODE CLASS EDITOR

Opcode Class: (Class Name)        File: (File Name)

_____________________________________________________

                Enter opcode class name > _

_____________________________________________________
                    Hit [RET] to exit
```

Opcode class names are limited to 16 characters. Any name supplied which is longer than 16 characters will be truncated. Any leading spaces in the specification of the name will be ignored.

If the class name provided is anme which is already defined in the Opcode Class file, then an error message is displayed to inform you fo the error. If the class name provided is a name which is already defined in the symbol table and is a symbol for something other than an opcode class name, then an error message is displayed to inform you of the error.

After a valid opcode class name has been supplied, the Class Edit mode is automatically invoked so that you can define the new opcode class.

Error messages which may be encountered when executing this command include:

Duplicate opcode class name - the ocode class name specified has already been defined.

Duplicate symbol - the opcode class name specified is a symbol which is already used.

Could not open temporary workfile - because the file is being edited, a backup copy is required. In the attempt to open a backup file on the same disk as the Opcode Class file, an error was encountered.

### 12.5.2.4 Rename-Class Command

The Rename-Class command is used to rename the selected opcode class. You may want to rename an opcode class if you are using symbolic debug and an opcode class name is a symbol which is also defined (differently) in your assembly language program. When this command is executed, you will be prompted to be certian that you want to rename the selected opcode class. The prompt will appear as follows:

```
                    OPCODE CLASS EDITOR
Opcode Class: (Class Name)      File: (File name)

-------------------------------------------------------------------------

      Rename opcode class (class name) (Y/N)? > _

-------------------------------------------------------------------------
E - edit  D - delete  C - Create  R - Rename  Q - quit
```

A negative response to the prompt ('N' or 'n') will abort the rename command and return control to the Class Selection Mode. A positive response ('Y' or 'y') will cause the Rename screen to appear. No other responses are accepted.

The rename screen is displayed when you have indicated that you want to rename the selected opcode class. The Rename screen appears as follows:

```
                    OPCODE CLASS EDITOR
 Opcode Class: (Class Name)      File: (File name)

-------------------------------------------------------------------------

                Enter new class name > _

-------------------------------------------------------------------------
                        Hit [RET] to exit
```

Opcode class names are limited to 16 characters. Any name supplied which is longer than 16 characters will be truncated. Any leading spaces in the specification of the name will be ignored.

If the class name provided is a name which already defined in the Opcode Class file, then an error message is displayed to inform you of the error. If the class name provided is a name which is already defined in the symbol table and is a symbol for something other than an opcode

class name, then an error message is displayed to inform you of the error.

After a valid opcode class name has been supplied, control is returned to the Class Selection mode.

Error messages which may be encountered when executing this command include:

Duplicate opcode class name - the opcode class name specified has already been defined.

Duplicate symbol - the opcode class name is a symbol which is already used.

Could not open temporary workfile - because the file is being edited, a backup copy is required. In the attempt to open a backup file on the same disk as the Opcode class file, an error was encountered.

#### 12.5.2.5 Quit Command

The Quit command is used to exit from the Opcode class Editor and return control to the Opcode Class Menu.

### 12.5.3 Class Edit Mode

The Class Edit mode is used to examine and/or modify the instructions which define an opcode class. Before explaining the method of defining opcode classes, however, a few definitions are required.

An 8051/8031 assembly language instruction can be broken down into components which together make up the instruction. The three basic components of an instruction are:

Instruction mnemonic: The basic operation of the instuction i.e. JMP,MOV, ADD, ETC. The instruction mnemonic is always the first symbol in a mnemonic instruction (i.e. opcode).

Operand 1: The first operand of the instruction (i.e. the destination).

Operand 2: The second operand of the instruction (i.e. the source).

Not all instuctions will contain both operands. Some will consist of 0 or 1 operand. All, however, must have an instruction mnemonic component. The set of valid operands consist of:

| | |
|---|---|
| # | - immediate data |
| DADDR | - direct on-chip memory or special function register address |
| BADDR | - direct on-chip bit address |
| /BADDR | - complemented contents of bit address |
| CADDR | - code address can be: full 16-bit address, 11-bit page address or 8-bit relative offset. |
| A | - accumulator |
| C | - carry flag |
| DPTR | - data pointer |
| PC | - program counter |
| AB | - register pair |
| Rn | - general purpose register (n = 0 through 7) |
| @Rn | - indirect register address (n = 0 of 1) |

The last two operand types (the register operands) can be broken down into sub-components. These consist of the

register addressing mechanism R or @R, and the register number. any 8051/8031 instruction can therefore be completely specified through the 5 instruction components: the instruction mnemonic, the first operand (if any), the register number of the first operand (if any), the second operand (if any), and the register number of the second operand (if any).

PLEASE NOTE that in the following discussion and on the opcode class editor display itself, the Instruction Menmonic field is labeled simply Instruction for the sake of brevity.

Classes of instuctions can be specified by NOT specifying a particular component of the instuction. As an example, suppose you wish to specify a class of instruction which includes all MOV instructions with the accumulator as the destination. This is specified by:

| Instruction | Operand1 | Register | Operand2 | Register |
|---|---|---|---|---|
| MOV | A | | | |

Any instuction which moves data to the implicitly addressed accumulator from anywhere else is specified. Notice that operand2 is not specified. An instruction component which is not specified is taken as a 'wild card' (i.e. it matches anything). Therefore any instruction of the form MOV A,xxxxx will be specified.

This includes:

| | | |
|---|---|---|
| MOV A,# | MOV A,daddr | MOV A,@R0 |
| MOV A,@R1 | MOV A,R0 | MOV A,R1 |
| MOV A,R2 | MOV A,R3 | MOV A,R4 |
| MOV A,R5 | MOV A,R6 | MOV A,R7 |

The class can be further restricted to all instruction which move data to the accumulator from general purpose registers. This is specified by:

| Instruction | Operand1 | Register | Operand2 | Register |
|---|---|---|---|---|
| MOV | A | | R | |

The R in operand 2 restricts the source of the data to a general purpose register. because the register number of operand 2 is not specified, any valid register number is

specified. The class of instuctions thus specified includes:

```
MOV A,R0     MOV A,R1     MOV A,R2
MOV A,R3     MOV A,R4     MOV A,R5
MOV A,R6     MOV A,R7
```

The class can be further restricted to a single instruction by specifying which general purpose register is to be the source. This is specified by :

| Instruction | Operand1 | Register | Operand2 | Register |
|---|---|---|---|---|
| MOV | A | | R | 4 |

The only instruction thus specified is MOV A,R4

PLEASE NOTE that the register fields are used ONLY when an operand specification field is R or @R. Use of this field in any other case is illegal. When the operand is R, the valid set of entries in the register field is the numbers 0 through 7. When the operand is @R, the valid set of entries in the register field is 0 or 1.

PLEASE NOTE that an instruction specification with all of its component fields left empty is ignored. It DOES NOT specify every possible instruction.

PLEASE NOTE that one class of 8051/8031 instructions actually contains three operands. These are the CJNE instructions. These, however, can still be uniquely specified through the use of the Instruction Mnemonic, Operand1 and Operand2 fields.

When the Class Edit mode is invoked, the following screen appears:

```
                    OPCODE CLASS EDITOR
 Opcode Class:(Name)  Status:(Status)  File:(File name)
 ------------------------------------------------------------

  Instruction  Operand1 Register  Operand2 Register
  -----------  --------  --------  --------  --------
        ( instruction specification 1 )
        ( instruction specification 2 )
        ( instruction specification 3 )
                        .
                        .
                        .
 ------------------------------------------------------------
                Hit [ESC] to exit edit mode
```

The Opcode class status display in the upper left corner of the Edit Command indicates which opcode class is being manipulated.

The Edit status display at the top center indicates the current status of the opcode class being edited. The status can be one of the following:

UNMODIFIED - the opcode class definition has not changed (default when editor is first invoked).

MODIFIED - the opcode class has been modified during the edit session.

The File status display in the upper right corner of the Opcode class Editor screen indicates which Opcode class file is currently being edited.

The center of the display contains specifications of the instructions which define the opcode class. Each specification is on a a separate line. The Opcode class Editor can accommodate up to 50 lines of instruction specifications for each opcode class.

The highlight is used to identify which field of which instruction specification is to be editied. Only one field can be manipulated at a time. When the editor is first invoked, the instruction field of the first instruction specification is highlighted.

A different field can be selected for editing by editing by moving the highlight to the desired field. The highlight is moved thorugh the use of the cursor control keys on the numeric keypad at the right of the keyboard. The cursor movement control keys operate as follows:

(▲) - up
(▼) - down
(◄) - left
(►) - right

The right and left movements of the highlight are limited to the fields of the current instruction specification. No wrap around is provided. The upward movement of the highlight is limited by the first instruction specification. The downward movement of the highlight is restricted to one line below the last instruction specification. This allows you to add a new specification at the bottom.

Field entries are modified by moving the highlight to the desired field and then entering the new data. When you begin to enter the new data, the New field data display will appear at the upper left of the center screen. The data entered will appear there as well (converted to upper case). It will appear as follows:

```
                    OPCODE CLASS EDITOR
 Opcode Class:(Name)  Status:(Status)  File:(File name)
 ------------------------------------------------------
 New field data > (new data)

   Instruction  Operand1 Register  Operand2 Register
   -----------  -------- --------  -------- --------
           ( instruction specification 1 )
           ( instruction specification 2 )
           ( instruction specification 3 )
                        .
                        .
                        .
```

---

The new data will not be entered into the selected field until the [RETURN] key is hit. This allows you to correct any errors in the new data before it is entered into the field. The backspace key allows you to erase characters from the New field data display.

The number of characters permitted in the New field data display depends on which field is being modified. Below is listed the number of characters permittted for each of the fields.

```
Instruction Mnemonic field - 5 characters
Operand field              - 7 characters
Register field             - 1 character
```

Within these limitations, you can completely specify the instuctions. If more characters are entered than the field can hold, the bell rings and the character is ignored.

PLEASE NOTE that no check is made when you enter data into a field that the data is a legitimate entry for that field. That function is performed when you try to exit the Class Edit mode (see below).

The field data replacement can be aborted by simply moving the highlight to another field BEFORE the [RETURN] key has been hit. This aborts the field data replacement, erases the New field data display and moves the highlight to the new field.

The data in a field may be deleted by simply hitting the [DELETE] key while that field is highlighted. More than one field can be deleted through the use of the [RETURN] key. When the [RETURN] key is hit (and you are not in the New field data entry mode) the data at the highlighted field and all subsequent fields of the current instruction specification are deleted. All fields of an instruction specification can be deleted by hitting the [RETURN] key when the instruction field is highlighted.

A line can be deleted from the opcode class definition only when all of its fields have been deleted. In this case, hitting the [DELETE] key will delete the entire line. It does not matter which field was highlighted when you hit the [DELETE] key. All subsequent lines will be moved upward one line.

A new line can only be inserted in the opcode class definition when the Insert mode has been selected. the Insert mode is entered by hitting the [INSERT] key. The [INSERT] key is a toggle switch which alternately turns the Insert mode on and off. The Insert mode is active when the INSERT sign is displayed at the top right of the center screen. When the [RETURN] key is hit while in the Insert mode, a new line is inserted before the current instruction

specification (the one whose field is highlighted). The current instruction specification and all subsequent lines are moved downward one line. A NULL instruction specification replaces the original line.

The [ESCAPE] key is used to exit from the Class Edit mode. When the [ESCAPE] key is hit, the Class Edit command options are displayed at the bottom of the screen. The display will appear as follows:

```
                    OPCODE CLASS EDITOR
Opcode Class:(Name)   Status:(Status)   File:(file name)

 Instruction  Operand1 Register  Operand2 Register
 -----------  -------- --------  -------- --------
         ( instruction specification 1 )
         ( instruction specification 2 )
         ( instruction specification 3 )
                       .
                       .
                       .

                 E - edit       Q - quit
```

The options available are to return to the Class Edit mode (the 'e' or 'E' response) or to return to the Class Selection mode (the 'q' or 'Q; repsonse). Executing the Edit command will reinvoke the Class Edit mode. Executing the Quit command will cause one of two things to occur. If the opcode class definition has not been modified, control will return immediately to the Class Selection mode. If however, the opcode class definition has been modified, you will be prompted as to whether or not you want to save the edited opcode class. The prompt will appear as follows:

```
Save edited opcode class? (Y/N)
```

A negative response ('n' or 'N') will cause the editied version of the opcode class definition to be discarded. The original (pre-edit) version of the opcode class defintion will remain intact. Control will be returned to the Class Selection mode.

A positive response ('y' or 'Y') indicates that you want to save the edited version of the opcode class definition. In this case, the validity of the instruction specifications must be checked before we can continue.

The instruction specification verification causes the middle of the screen is cleared. The instruction specifications are then examined one at a time. As each is examined, it is displayed on the screen. If an instruction specification is found which does not correspond to any of the 8051/8031 instruction, an error message will be displayed and the verification halts. You will be prompted to hit the [ESCAPE] key to continue the verification.

When all instuction specifications have been checked, one of two actions can occur. If no errors were encountered, control is returned to the Class Selection mode. If however, an error was encountered, the Class Edit command options (E - edit Q - quit) are again displayed at the bottom of the screen. This allows you the opportunity to re-enter the Class Edit mode in order to correct the errors.

Error messages which may be encountered while in the class Edit mode include:

Could not match instruction - the instruction specification does not correspond to any 8051/8031 instructions.

## 12.6 Help Command

The Help command is used to display a detailed description of the function of each of the commands in the Opcode Class menu.

Error messages which may be encountered when executing this command include:

Help file not found - the file "HLPFILE" could not be found on either the default or a: disks.

## 12.7 Quit Command

The Quit command is used to return to the previous menu.

CHAPTER 13
MACRO MENU

## 13.1 Macro Menu Overview

The Macro menu is used to create macro command files and execute macro command files. A macro command file is a file which contains groupings of ICD commands which, when executed together, perform a macro function. These macro functions are typically repetitious tasks which are done over and over again in one or many debugging sessions. The macro command facility allows you to define the macro command file ONCE and they execute it anytime later in the same or even another debugging session. In this menu you can:

1) execute from a macro command file,
2) define a macro command file.

## 13.2 Macro Menu Screen

The Macro Menu screen appears as follows:

```
Execute  Learn  Help  Quit
(Quick help line for highlighted command)
```

---

MACRO MENU

---

Upon entering, the Execute command will be highlighted.

## 13.3 Execute Command

The Execute command is used to begin execution from a macro command file. You will be prompted to supply the name of the macro command disk file. In addition, you will be prompted for any parameters to pass to the macro command file. The Execute screen appears as follows:

```
Enter file name >_
Enter parameters >_
```

---

EXECUTE FROM A MACRO DISK FILE

---

The name should be a complete file name specification including a drive specification. Hitting the '?' key in response to the prompt will call the directory facility (see Chapter 3) through which you can get a listing of the entries in any specified directory. Hitting the [RETURN] key in response to the prompt will abort execution of the command. If a file cannot be opened or cannot be found, you will be notified of the error. If the specified file's format is not that of a macro command file, an error message will be displayed.

Once the command file has been properly opened, you will be prompted for the parameters to pass to the macro command. All parameters should be listed in the order they are used in the macro command. The parameters must be separated by either COMMAs (,) of SPACEs. Entering a [RETURN] in response to this prompt will cause the macro command to be executed with NO parameters passed to it.

For example, suppose you have defined a macro command file called 'MYMACRO.MAC'. Within this macro, you have defined three parameters. The order in which they are called is:

1) a code file
2) an experiment file
3) an opcode class file

For the current execution of this macro, you wish these parameters to be:

```
1) (default):test.hex   (the code file)
2) /test.exp            (the experiment file)
3) a:opcode.opc         (the opcode class file)
```

These parameters could then be specified by:

```
Enter file name > mymacro.mac
Enter parameters > test.hex,\test.exp,a:opclass.opc
```

or

```
Enter file name > mymacro.mac
Enter parameters > test.hex\test.exp a:opclass.opc
```

The parameter parser will accept up to 8 parameters. Eight is the maximum number of parameters allowed in a macro command. An attempt to enter more than 8 parameters will result in a error message being displayed.

Once the macro command file and the parameters (if any) have been specified, the system returns to the Main menu and execution of the macro command begins.

If the macro command file contains more parameter specifications than were passed in th parameter list i.e. we will run out of parameters) then execution of the macro is aborted at the first unmatched macro file parameter. At this point the system reverts to the normal mode of taking manuallly entered system commands.

Error messages which may be encountered when executing this command include:

Must establish communication first - a macro command cannot be executed before communication with the emulator has been established.

Cannot open file - the specified file could not be found on the specified drive, the default drive, or the A: drive.

File is not proper macro file format - the macor command file specified was not the proper format for a macro command file.

Too many macro parameters - more than 8 macro parameters were specified.

## 13.4 Learn Command

The Learn command is used to create a macro command file. You will be prompted to supply the name for the new file. The Learn screen appears as follows:

Enter file name > _

---

CREATE A MACRO DISK FILE

---

The name should be a complete file name specification including a drive specification. Hitting the '?' key in response to the prompt will call the directory facility

(see Chapter 3) through which you can get a listing of the entries in any specified directory. Hitting the [RETURN] key in response to the prompt will abort execution of the command. If the file cannot be opened or cannot be found, you will be notified of the error. Any existing file of the same name will be overwritten.

The Learn mode is a means of creating complex macro command files by teaching the sytem (in a step by step manner) what you want it to do. Once the Learn mode is invoked, you are returned to the Main menu. From there, every keystroke entered into the system is also entered into the macro command file. You can go from menu to menu entering commands as you like. They will all be collected in the macro file.

The Learn mode can be terminated by entering the macro sign [%] in response to any command which is executed via a single keystroke; or by entering the macro sign [%] followed by [RETURN] in response to any command which accepts a string.

REMEMBER: Although you are creating a macro command file, you are also exercising the emulator in the normal manner. All conditions which are required for normal operation of the emulator are therefore also required for creating a macro command file (i.e. establish communications, load program memory, etc.).

Parameters may be entered into the macro command file by prefacing any string which you want to be a parameter with the macro sign [%]. This has the effect of entering a parameter substitution instead of the actual string into the macro command file. I.e. suppose you want ot laod code into the sytem but you want the code file name to be a parameter. When the system prompts for the file name you respond:

```
Enter file name > %demo.dbg.
```

In addition to putting the parameter substitution in the macro command ile, it also has the effect of loading the code file 'demo.dbg' into the program code memory. This allows to continue creating the macro command file via the learn mode in the normal manner.

A macro command file may contain up to 8 parameters. An attempt to define more than 8 parameters will result in an error message being displayed and the Learn mode being

terminated. The macro command file will be closed normally and will be valid up to but not including the offending parameter.

Error messages which may be encountered when executing this commadn include:

Must establish communication first - a macro command cannot be created before communication with the emulator has been established.

Cannot open file - the specified file could not be opened.

Too many macro parameters - more than 8 macro parameters were specified.

## 13.5 Help Command

The Help command is used to display a detailed description of the function of each of the commands in the Macro Menu.

Error messages which may be encountered when executing this command include:

Help file not found - the file "$HLPFILE" could not be found on either the default or A: drives.

## 13.6 Quit Command

The quit command is used to return to the previous menu.

CHAPTER 14
A TUTORIAL EXAMPLE

## 14.1 Introduction

This chapter presents a sample debugging session using the ICD. Throughout the session you will be issuing commands to the system and it will be responding.

When you see this:

You Enter >

it is the signal for you to type something into the system. If you find that at some point in the tutorial you are not in the menu you should be, refer to the menu organization at the beginning of Chapter 3 to find out how to get back to the proper menu.

## 14.2 Getting Started

This sample session will be run in the full speed simulator mode. This means that a target system board is not required. All that is required is that the ICD probe be inserted into the simulator board which is provided as part of the ICD system. This is the mode which would normally be used when developing and debugging software for the microcontroller. The ICD, in effect, acts like a full speed microcontroller simulator.

The first thing you must do is to boot up your host's operating system. Next, the ICD system software disk MUST be placed in the default disk drive.

You enter > zlink [RETURN]

This calls the ICD system's user interface program. At first the Zlink logo will be displayed, then the Main Menu screen will appear.

Now we must establish communication with the ICD. First of all, be sure that the RS232 cable is connected at both the host and emulator module ends. Next plug in the power supply to the emulator module (this serves to reset the emulator).

You Enter > E

This executes the system configuration file which establishes communication with the emulator. The default baud rate in the configuration file supplied with the Zlink software is 9600 baud. The Simulator board supplies a crystal frequency which will support the maximum baud rate.

If the system has been configured properly, the system should have returned to the Main Menu. If communication has not been established, see the troubleshooting section in Appendix A.

## 14.3 The Program

Before going further, let's take a look at the program we are going to be working with. It is important that you understand how this program works before we go on with the tutorial.

The purpose of this program is to output a pulse train on Port1-Pin0. The pulse train is: a series of 5 positive going pulses of equal duration; a skipped pulse; then a continual repetition of the cycle. The pulse train will appear as follows:

The following program generates the pulse train:

```
                    1    $debug
                    2    $mod51
                    3
0090                4    outbit          BIT     90h
                    5
0064                6                    deg at 64h
0064                7    tempcount:      DS      1
                    8
---                 9                    cseg
0000 020030        10                    LJMP    start
                   11
0030               12                    cseg at 30h
0030 900000        13    start:          MOV     DPTR,#0
0033 C290          14    outerloop:      CLR     outbit
0035 75640A        15                    MOV     tempcount,#10
0038 120050        16    innerloop       CALL    wastetime
003B B290          17                    CPL     outbit
003D E4            18                    CLR     A
003E 309001        19                    JNB     outbit,skipover
0041 F4            20                    CPL     A
0042 D564F3        21    skipover:       DJNZ    tempcount,innerloop
0045 A3            22                    INC     DPTR
0046 120050        23                    CALL    wastetime
0049 120050        24                    CALL    wastetime
004C 80E5          25                    JMP     outerloop
004E 80E0          26    endofprogram:   JMP     start
                   27    ;
                   28
0050 78FF          29    wastetime       MOV     R0,#0FFh
0052 D8FE          30                    DJNZ    R0,$
0054 22            31                    RET
                   32                    END
```

There are three loops in this program. The first is a subroutine called "wastetime." This loop executes five hundred and eleven 8051 machine cycles before exiting.

The second loop is in the main program and begins at the label "innerloop." It is responsible for generating the 5 pulses. It accomplishes this by calling the "wastetime" routine to polarity of the output pin. Notice that the accumulator is used to reflect the state of the output pin.

The third loop is also in the main program and begins at the label "outerloop." It is responsible for calling "innerloop" to generate the 5 pulses and then calling "wastetime" twice to the generate the blank pulse between sets of 5 pulses. Notice that the data pointer (DPTR) is used to count the number of sets of 5 pulses which have been transmitted.

When you feel comfortable with the program, go on to the next section.

## 14.4 A Sample Session

First we must load the program into the program code memory.

You Enter > L to begin the load command.

When the system prompts you for the file name,

You Enter > demo.dbg [RET] to read in the desired file.

After the file has been read into the system, (the WORKING sign has disappeared, we go on. The next thing we want to do is to define the experiment. We therefore must call up the Examine Experiment menu.

You Enter > I to call the Interrogate menu.

You Enter > E to call the Examine/Modify Experiment menu.

We are now ready to define the experiment. The first experiment we will preform is a predefined experiment which will be loaded into the system.

You Enter > L to begin the Load command.

The system will prompt you for the file name of the experiment file.

You Enter > demo.exp [RET] to load the desired experiment.

After the file has been read into the system, we will examine the experiment.

You Enter > E to enter the Experiment Editor.

The following experiment will appear:

```
if pc = innerloop then break
```

This experiment sets a breakpoint at PC location 38h ('innerloop'). Every time the inner loop is entered, the emulation will break.

Now we will exit from the editor.

You Enter > [ESC] to exit the Edit mode.

You Enter > Q to exit from the editor.

Next we want to compile the experiment. Compiling the experiment does two things. First, it makes sure that the experiment definition is legal. Second, it downloads the breakpoints to the emulator module.

You Enter > C to compile the experiment

When the compilation is completed, the results will be displayed at the bottom of the screen.

You Enter > [RET] to continue.

We are now ready to execute the experiment. We must first return to the Interrogate Menu.

You Enter > Q to return to the Interrogate menu.

Let's take a look at the program we have loaded.

You Enter > C to call the Examine Program Code Memory menu.

We have a choice of examining the memory in raw data mode (Table) or as disassembled mnemonic instructions. We will choose the latter.

You Enter > D to call the code disassembler.

The system will prompt you for the starting address of the disassembly.

You Enter > start [RET]  to start at PC location 30h.

The system will next prompt you for the number of instructions you want to disassemble.

You Enter > 14 [RET]  to disassemble the main program

Notice that all of the symbolic references have been preserved. This occurs automatically when you assemble your code with the @ 8051 Cross assembler with the debug switch.

To disassemble the 'wastetime' subroutine,

You Enter > D  to call the code disassembler

You Enter > wastetime [RET] to start at PC location 50h.

The system will next prompt you for the number of instructions you want to disassemble.

You Enter > 3 [RET]  to disassemble the subroutine.

Now we will return to the Interrogate menu.

You Enter > Q  to return to the Interrogate menu.

Now we can begin the emulation. Notice that the PC is initialized to 0. Following a cold system startup, the registers are automatically initialized to their RESET values. It is, therefore, NOT necessary to begin the experiment by executing a Reset command.

You Enter > G  to execute until a breakpoint.

Let's see that has occurred. The PC value is now at 38h. This is where we set the breakpoint (at 'innerloop'). The next instruction to be executed will be:

LCALL wastetime

Notice that the accumulator and data pointer values have not yet been modified. We would next like to single step through the program and watch the accumulator value get updated to reflect the state of the output pin.

You Enter > S  to execute a single instuction.

The program made the call to 'wastetime'. Notice that the PC value is now at 50h. The instruction to be executed will be

MOV R0,#0FFh

which is the first instruction of the 'wastetime' routine.

You enter > S  to execute a single instruction.

The PC value is now 52h. The next instruction to be executed will be

DJNZ R0,0052h

You Enter > S  to execute a single instruction.

Notice that the R0 register in the GPR bank has been decremented by 1. Let's try that again.

You Enter > S  to execute a single instruction

And again.

You Enter > S  to execute a single instruction.

You will remain in this tight loop until the value in R0 is decremented to 0. Instead of continually hitting the Single Step command lets change the value in R0.

You Enter > M  to call the examine register menu.

The system will prompt you for the register you wish to examine.

You Enter > R0 [RET]  to examine the R0 register.

The system then displays the value contained in R0. The value is FCh just as we could expect after 3 decrement cycles. The system next prompts you for the new register value.

You Enter > 3 [RET]  to change the value of R0.

The system will again prompt you for another register to examine.

You Enter > [RET] to return to the Interrogate menu.

Now we want to continue our experiment. We should only have to execute single step 3 times before we decrement R0 to 0 and exit the loop.

You Enter > S to execute a single instruction

You Enter > S to execute a single instruction.

You Enter > S to execute a single instruction.

The PC value is now 54h. The next instruction to be executed will be

RET

don't forget that a breakpoint is still set at location 38h. If we begin execution again, emulation will break at the beginning of the next pass through the inner loop.

You Enter > G to execute until a breakpoint

Notice that we are again at PC location 38h. Also notice that the value of the accumulator is now FFh. This means that the output pin is currently outputting a 1. We did, however, miss the instruction which updated the accumulator which is what we were trying to watch in the first place.

If we execute a single step, as we did last time, we will again enter the 'wastetime' routine. What we sould really like to do is to skip over the 'wastetime' routine and stop at location 3Bh. This can be accomplished by setting phantom breakpoint at that location. A phantom breakpoint exists only until the next break in emulation. After that time, it no longer exists.

You Enter > P to set a phantom breakpoint

The system will prompt you for the address of the breakpoint.

You Enter > 3B [RET] to set the address

Once the address has been supplied, the execution begins. The PC value is now at the breakpoint we set, namely 3Bh. Now we can single step until the accumulator gets updated.

You Enter > S  to execute a single instuction.

The next instuction is

CLR A

You Enter > S  to execute a single isntruction.

The accumulator has now been cleared. The next instruction will check the state of the output bit. If the output bit is a 0 (as it should be on this pass through 'innerloop'), the program should jump to the PC location 42h ('skipover') because the accumulator already reflects the state of the output pin.

You Enter > S  to execute a single instruction.

Notice that we did indeed make the jump and that the accumulator does reflect the state of the output pin. One more single step and we will be back at the beginning of the inner loop again.

You Enter > S  to execute a single instruction.

We have just completed 2 passes through the inner loop. Notice that the value of the data pointer (DPTR) is still 0. It will not change until we have completed 10 passes through the inner loop. We should, therefore, be able to watch the value of DPTR change if we complete 8 more passes through the inner loop. Remembering that breakpoint is still set at the beginning of the inner loop, executing a Go command 8 times should change the data pointer.

1) You Enter > G  to execute until a breakpoint.

2) You Enter > G  to execute until a breakpoint.

3) You Enter > G  to execute until a breakpoint.

4) You Enter > G  to execute until a breakpoint.

The data pointer should now contain the value 1. There is an easier method of repeating the Go command. It is with the use of the repetition counter. Let's try to increment the data pointer again, this time using the repetition counter. In order to increment the data pointer, we must execute 10 passes through the inner loop.

You Enter > L  to call the repetition counter.

the system will prompt you for the number of repetitions you desire.

You Enter > 10 [RET]  to set the repetition counter.

Now we can execute the Go command. it will automatically repeat 10 times.

You Enter > G  to execute until 10 breakpoints

Notice that the data pointer has been incremented to 2.

Enough of this. Let's go back to the experiment definition and change it.

You Enter > E  to call the Examine Experiment menu.

You Enter > E  to call Experiment Editor.

The following experiment will appear:

```
if pc = innerloop then break
```

We want to change the breakpoint from 'innerloop' to 'endofprogram'. This is accomplished by moving the cursor to the right until it is on top of the first character (the i) of 'innerloop'. This will require 8 move cursor right (-) commands.

1) You Enter > [-> ]  to move the cursor to the right.

2) You Enter > [-> ]  to move the cursor to the right.

3) You Enter > [-> ]  to move the cursor to the right.

4) You Enter > [-> ]  to move the cursor to the right.

5) You Enter > [-> ]  to move the cursor to the right.

6) You Enter > [-> ]  to move the cursor to the right.

7) You Enter > [-> ]  to move the cursor to the right.

8) You Enter > [-> ]  to move the cursor to the right.

Next we will replace the breakpoint specification.

You Enter > endofprog  to begin replacing the text.

You Enter > [INS]  to enter the insert mode.

You Enter > ram  to complete the text replacement.

Now we will exit from the editor.

You Enter > [ESC]  to exit from the edit mode.

You Enter < Q  to exit from the editor.

You will be questioned as to whether or not you want to save the edited experiment.

You Enter > Y  to save the edited experiment.

Now we want to compile the new experiment.

You Enter > C  to compile the experiment.

When the results are displayed

You Enter > [RET]  to continue.

Now we are ready to execute the new experiment. We must first return to the Interrogate menu.

You Enter > Q  to return to the Interrogate menu.

Notice the PC value. It is still at 38h. We want to begin the new experiment. Starting at location 0. The Reset command will accomplish this. The Reset command resets the target processor then executes the experiment until a breakpoint is encountered.

You Enter > R  to execute the Reset command.

The system will ask you if the reset pulse will be coming from your system board. If you are debugging your system board, the reset will probably come from your board. Because we are executing in the simulator mode, the emulator must supply its own reset pulse.

You Enter > N — to make the emulator reset itself.

Now the emulator is running, and running, and running ... What happened? The breakpoint was set at locaion 4Eh. Why didn't it stop? Notice that the instruction before the breakpoint instruction is an unconditional jump back to 'outerloop'. Therefore the program never reaches the breakpoint instruction and will loop forever. A runaway emulation can be stopped by executing a host interrupt.

You Enter > [ESC] — to stop the emulator.

The system will respond with an error condition stating that the emulation was stopped by host interrupt. This is an error condition becuase the PC value is indeterminate after a host interrupt has occured. Notice that the value of the PC is xxxx. The Break address however can be used to determine the approximate value of the PC when the forced break occurred. We cannot continue with this experiment until the PC value is initialized.

You Enter > M — to call the Modify Register menu.

You Enter > PCxxxET] — to select the PC register.

You Enter > 0 [RET] — to reset the PC value to 0.

Notice the PC value now has the value 0. We can now continue with the experiment.

You Enter > [RET] — to return to the Interrogate.

As we have just demonstrated, the experiment will run forever if we execute a Go command. Let's set a phantom breakpoint at location 33h ('outerloop') and see if execution stops.

You Enter > P — to set a phantom breakpoint.

The system will prompt you for the address of the breakpoint.

You Enter > 33 [RET] — to set the address.

Once the address has been supplied, the execution begins. The PC value is now at the phantom breakpoint address 33h. To demonstrate that phantom breakpoints are valid for only one execution cycle, we will execute a Go command.

| | |
|---|---|
| You Enter > G | to execute until a breakpoint |

As expected, the emulation runs forever because the breakpoint at location 33h no longer exists.

This concludes the sample session. To exit from the system:

| | |
|---|---|
| You Enter > [ESC] | to stop the emulator. |
| You Enter > Q | to return to the Main menu. |
| You Enter > E | to exit the system. |
| You Enter > Y | to indicate that you really want to exit. |

# CHAPTER 15
# SYSTEM REQUIREMENTS

This chapter lists the minimum system requirements which must be met in order to use a ICD.

## 15.1 Hardware Requirements

- IBM PC or a compatible PC

- 1 5 1/4 inch double sided/double density floppy disk drive (2 are recommended)

- 256K bytes of memory

- RS232C interface board

- RS232 cable with a male connector at the emulator end.

- Emulator power supply

  2.0A + 5VDC +-5%

  150mA +- 9-15VDC

## 15.2 Software Requirements

- PC DOS version 2.0 or later

APPENDIX A
TROUBLESHOOTING GUIDE

This section is provided to help you work through problems which prevent the ICD from operating properly. Please refer to this section FIRST when any problems arise. If, however, after using this guide, the emulator is still not operating properly, call ZAX between 8AM and 5PM Pacific Standard time. Our phone number is: (714) 474-1170 and 800 421-0982.

If you need to report a problem, please have the following information available:

- The ICD model
- The version number of the Zlink system software
- The emulator options you have purchased
- The manufacturer and model of the host you are using
- The manufacturer and model of the RS-232C interface card you are using (also it is important to know how the card is configured if options are available)
- The name and version of the operating system you are using

If you do call, try to have your system available and near the phone. Often, some simple tests can determine the cause of the problem.

PLEASE NOTE that we cannot return your calls if we have the wrong phone number or your phone doesn't answer. If we are supposed to call you back and you haven't heard from us in 2 days, please call us again.

## A.1 Cannot Establish Communication

One very common problem is the failure to establish communication between the host computer and the emulator module. There can be many causes for this and we will cover each one separately.

In order for the emulator to communicate, it must have both power applied to it and a frequency source for its on-board oscillator. (Remember that the emulator uses the frequency source from your system board to generate its clock.)

You can easily check that these two fundamental requirements are met. By looking into the vents on the side of the emulator module nearest the DB25 connector, you can see two LEDs. The red LED will be glowing when power is applied to

the emulator module. The green LED will be glowing when power is applied AND the on-board oscillator is running.

### A.1.1 Red LED is NOT Glowing

If the red LED is not glowing, the emulator module is NOT getting its power properly. Check the following:

- Is the emulator's power supply plugged in?
- Is the emulator's power supply turned on?

If both of these are true then disconnect the power supply from the emulator module. Check to be sure that the proper voltages are coming from the proper pins in the power supply connector. The correct voltage for each of the pins is shown below:

| | |
|---|---|
| Pin 1 | ground |
| Pin 2 | ground |
| Pin 3 | +12VDC |
| Pin 4 | -12VDC |
| Pin 5 | +5VDC |

If the voltages are correct and are supplied through the proper pins, try to reseat the connector with the emulator header. If the red LED still doesn't come on, call ZAX.

If the voltages are not correct and you have supplied your own emulator power supply, check your power supply and wiring to the connector.

### A.1.2 Green LED is NOT Glowing

If the green LED is not glowing, be sure that the red LED is glowing. If it is not, see the previous section.

The green LED indicates that the emulator's on-board oscillator is running. In order to run, the oscillator requires a frequency source.

If you are debugging your system board, ensure first of all that the emulator probe is oriented properly in the DIP socket on your board. Once you have checked this, ensure that you have a crystal connected to the XTAL1 and XTAL2 pins of the emulator probe or that you are supplying a constant frequency source to the XTAL2 pin. If the frequency source on your board seems to be in order (i.e. Voh > 2.5V, Vol < 0.8V and XTAL1 = Gnd), then remove the emulator probe from our board and insert it in the DIP socket on the Simulator board (being careful to orient it properly). If the green LED is now glowing, then the problem is in the oscillator circuitry on your board.

If you are using the ICD in the full speed simulator mode (i.e. the ICD probe is inserted in the DIP socket on the Simulator board then ensure the following:

- The emulator probe is oriented correctly in the DIP socket. Correct orientation can be determined by alining pin 1 on the probe with pin 1 on the socket.

- The Simulator board has not been physically damaged.

If these have been checked and the green LED is not glowing then call ZAX.

### A.1.3 Baud Rate Selection

If both the green and red LEDs are glowing but communication cannot be established, you may have selected an illegal Baud Rate for the frequency you are supplying to the emulator. Remember that the emulator derives its operating frequency from the frequency source you supply. Below is a table which shows the minimum frequency source required for each Baud Rate:

| Baud Rate | Minimum Frequency Source |
|---|---|
| 9600 | 8MHz |
| 4800 | 4MHz |
| 2400 | 2MHz |
| 1200 | 1MHz |
| 600 | any |
| 300 | any |
| 150 | any |
| 110 | any |

Selecting a Baud Rate for a supplied frequency which is below the required minimum can cause a communication failure.

### A.1.4 RS-232C Hardware Check

If both the green and red LEDs are glowing and the Baud Rate is correct for the supplied frequency but communication has not been established, do the following:

- Check that the RS-232C interface card has been configured correctly for the number of wires in your RS-232C cable.

- Check the pins of the RS-232C cable for continuity. (Which pins you check depends on how many wires are in your RS-232C cable. See Chapter 2 for details.

If both the above items have been checked, ensure that the cable is seated properly at both the host and emulator ends. If you still cannot establish communication, call ZAX.

## A.2 Excessive Number of Communication Errors

This section is used ONLY after communication has initially been established. When you have established communication with the emulator, communication errors can come from a number of sources. These will each be discussed below.

One major cause of an excessive number of communication errors that a Baud Rate has been selected which cannot be supported by the operating frequency of the emulator. See the Baud Rate Selection section (above) to ensure that you have selected a legal Baud Rate.

If the Baud Rate has been checked and is legal, do the following:

- Ensure that the supplied frequency source is stable.
- Ensure that the RS-232C cable is seated properly at both the host and emulator ends.
- Ensure that the power supply connector is seated properly in the emulator module.

If none of the above alleviates the problem then read on.

The ICD communication protocol was designed to provide transparent recovery from small random bursts of noise. It CANNOT however provide recovery from large or frequent bursts of noise.

If you are operating in a noisy environment you may want to lower the communication Baud Rate. This will provide some extra noise margin. Another alternative is to use shielded RS-232C cable.

If you are not operating in a noisy environment, have checked all of the above and are still getting an excessive number of communication errors, call ZAX.

APPENDIX B
ERROR MESSAGE SUMMARY

This Appendix provides a complete listing of all error messages which can be encountered while in an ICD debugging session. There are actually 3 classes of errors which can be encountered. Each will be discussed individually.

## B.1 PLINK86 Overlay Loader Errors

There is a class of errors which can occur when the ICD system disk is removed from the default drive during a debugging session or when the disk is not running in the DEFAULT drive. In either case you will get PLINK86 Overlay Loader errors. They will appear as follows:

```
PLINK86 Overlay Loader - (error description)
Enter file name prefix (X: or path name/) or '.' to
quit=>
```

To recover from this class of error, reinsert the Zlink disk in the default drive. Next type the drive designator of the default drive. (E.g. if the default is drive A: you would enter a:).

## B.2. ICD Errors Messages

The second class of errors are the user interface error messages which the ICD system provides. These are all nonfatal errors and can be exited by entering [ESC]. The error messages are listed alphabetically.

'=' expected - something other than an '=' was used in an immediate type conditional.

'=' or 'class' expected - something other than an '=' or 'class' was used in an opcode type conditional.

Address is out of range - the specified address was outside the valid address range in the given context.

Address value is too large - an address was specified for a direct byte or direct bit conditional which exceeds OFFh.

Block 1 address is out of range - the address specified is outside the valid address range of the selected memory.

Block 2 address is out of range - the address specified is outside the valid address range of the selected memory.

BREAK CAUSED BY HOST INTERRUPT - an abnormal break condition was caused by a host generated interrupt. If you have not hit the [ESC] key to cause this break condition, then it was caused by noise on the RS232 link.

Byte count causes address out of range - the sum of the specified address and the specified number of bytes causes an illegal memory address to be generated.

Cannot assemble offset uploaded code - code was uploaded from the target system at some starting address other than 0. This code can only be examined using the table command.

Cannot execute offset uploaded code - the code in the emulator's program code memory was uploaded with an offset value. Code uploaded in this manner cannot be executed.

Cannot open file - a file could not be opened in the write mode on the specified disk.

Cannot restart - instruction calls itself- an experiment cannot be restarted from a jump instruction which calls itself.

Cannot restart - instuction jumps on self- an experiment cannot be restarted from a jump instruction which specifies itself as the jump target.

Class xxxxx not found in file xxxxx- the specified opcode class could not be found in the expected opcode class file. This is most likely caused by editing a 2nd opcode class file between the time the 1st opcode class file was loaded and the experiment was compiled.

Code address expected- a symbol other than a code address symbol was used in the specification of a code address value.

Code jumps out of range- an attempt was made to restart an experiment at an instruction which may cause the PC value to jump outside the range of the emulator's code memory.

Code memory is not loaded- the ICD's program code memory has not been loaded with program code. Return to the Main menu to load code.

Communication error - reset comm link- a non-recoverable error occured. Communication must be reestablished via the Execute command in the Configuration menu.

Communication NOT established - try again- the emulator module responded to the host's request to establish communication but it's response was incorrect. Reset the emulator module and try again.

Comparator expected - something other than a comparator was encountered where a comparator was expected.

Could not match instruction - the instruction specification does not correspond to any 8051/8031 instructions.
Could not open opcode class file xxxxx - the opcode class file could not be opened. This is most likely caused by the removal of the disk between the time the opcode class file was loaded and the time the experiment was compiled.

Could not open temporary work file - because the file is being edited, a backup copy is required. In the attempt to open a backup file on the same disk as the Opcode Class file, an error was encountered.

Destination address is out of range - the address specified is outside the valid address range of the selected memory.

Direct bit address expected - a symbol other than a direct bit address symbol was used in the specification of a direct bit address value.

Direct byte address expected - a symbol other than a direct byte address symbol was used in the specification of a direct byte address value.

Directory not found - a directory was specified which did not exist or couldn't be reached.

Division by zero - the expression being evaluated includes an attempt to divide by zero.

Duplicate opcode class name - the opcode class name specified has already been defined.

Duplicate symbol - a label was specified which has previously been defined.

Emulator not ready - the emulator module did not respond to the host's request to establish communication. Check that power is applied to the emulator module, that it has been reset, and that the emulator probe has been supplied with a frequency source.

Expecting an EOL - the assembly language mnemonic instruction supplied contains too many operands.

Experiment buffer is full an attempt was made to insert a line in an experiment which already contains 32 lines.

Expression stack overflow - the expression stack has a depth of 32 values. The expression being evaluated exceeds this depth.

File has more than 32 lines - the experiment file contains more than the legal number of lines.

File is not proper absolute object format - a file was being processed as a Zlink absolute object formatted file but some formatting error was encountered while records were being read.

File is not proper Intel hex format - a file was being processed as an Intel hex formatted file but some formatting error was encountered while records were being read.

File is not proper macro file format - the macro command file specified was not the proper format for a macro command file.

File not found - the specified file could not be found on the specified drive, the default drive, or the A: Drive.

Help file not found - the file "$HLPFILE" could not be found on either the default drive or the A: drive.

'if' expected - an experiment statement did not begin with 'if'.

Illegal assembly line - The line doesn't begin with a label or instruction mnemonic.

Illegal binary value - the new value supplied was not a 0 or 1.

Illegal bit designator - the value specified was greater than 7 or the symbol specified was not a bit type symbol (Modify-Regs).

Illegal bit designator - an illegal bit designator address was specified. A bit designator contains a byte address,

followed by a PERIOD, followed by the bit index into the byte. First, the specification of the byte address part of the bit designator was not a legal bit addressable address. Second, the bit index into the byte address exceeds 7. (Code assembler).

Illegal bit for specified register - the symbolic bit specified is not a bit in the specified bit addressable register.

Illegal character - a character was encountered which is not part of the legal character set. (See Appendix D).

Illegal conditional - a simple conditional was encountered which was not part of the legal conditional set.

Illegal digit for specified radix - an illegal digit was encountered for the specified radix of a number.

Illegal drive specification - the specified drive designator is not between A and P.
Illegal end of experiment - the end of the experiment file was encountered in the middle of parsing an experiment statement.

Illegal end of string - the end of the pattern was encountered before a terminating character string delimiter was found.

Illegal entry found - an entry was found in the pattern which could not be identified as a number, a symbol, or a character string.

Illegal indirect register - the indirect addressing mode designator (@) was followed by something other than R0 or R1. This error can also occur in the MOVC A,@A+DPTR, MOVX A,@DPTR, MOVX @DPTR,A and the JMP @A+DPTR instructions if the operands after the indirect addressing mode designator are not specified properly.

Illegal integer value - the number specified contains an illegal decimal digit.

Illegal literal expression - a null ASCII literal string ('') was found.

Illegal number specification - the number contains an non-hexadecimal digit.

Illegal opcode after label - the symbol after a label wasn't an opcode.

Illegal opcode class file format - the file specified is not the proper format for an Opcode Class file.

Illegal operand - the operand specified is not a legal operand for the instruction.

Illegal operator - the arithmetic operator specified is not a legal operator.

Illegal or missing expression - a number, symbol, or arithmetic expression was expected but was either missing or could not be evaluated properly.

Illegal or missing expression operator - and arithmetic operator was expected but was either missing or was not a legal operator. (See Appendix D for the single line assembler's legal operators.)

Illegal or missing expression value - in evaluating an expression, an expected number or symbol was either missing or illegal.

Illegal Port 2 address mask - the mask specified was too large to fit in a byte wide register or the mask specification began with non-numeric character.

Illegal register address - the address specified does not correspond to a special function register.

Illegal register specification - the register name began with a non-alphanumeric character.

Illegal result statement - a result other than break was encountered.

Illegal rollover boundary - the rollover boundary was not 4K or 8K or the rollover boundary specification began with non-numeric character.

Illegal string character - an apostrophe was found in a character string which was followed neither by another apostrophe nor a space. Such use of an apostrophe within a character string is illegal.

Illegal symbol type - the symbol specified was of the correct type in the given context.

Illegal system status file format - the file specified was not the proper format for a system status file.

Improper address segment - the specified symbol was not an address for the selected memory space.

Line greater than 76 characters found - a line was found in an experiment file which was longer than the legal length.

Line is full - an attempt was made to insert a character in a line which already contains 76 characters.

Missing operand delimiter - a COMMA operand delimiter was missing from the operand fields of the instruction.

Must compile experiment first - an experiment was loaded but the experiment compiler was not executed prior to running the experiment. This is an error because it is the experiment compiler which sets the breakpoints in the emulator.

Must establish communication first - an attempt was made to issue a command to the emulator module before communication was established.

Must load experiment first - an attempt has been made to compile an experiment before it has been loaded or created.

Must load program code memory first - the program code memory was not loaded prior to executing a single step instruction.

Must reinitialize PC value first - the PC value was not reintialized after halting an emulation with host interrupt.

Must reset emulator to change baud rate - an attempt was made to reconfigure the system's baud rate when communication had already been established at a different baud rate. In this case, reset the emulator module and try again.

No code at the specified memory location - the specified starting address does not contain valid code.

Must reload code memory first - the experiment was deleted but the breakpoints set by that experiment are still resident and active in the emulator.

No default radix provided - a number was specified without a radix specifier.

Number expected - something other than a numeric value was enocuntered where a number was expected.

Number is too large - the number specified exceeds 64K.

Number or address symbol expected - some thing other than a numeric address or a symbolic address was encountered where an address was expected.

Opcode class table is full - more than 72 opcode classes are defined in the specified file.

Operator stack overflow - the operator stack has a depth of 32 values. The expression being evaluated exceeds this value.

PC value is out of range - the PC address specified file contains code at an address beyond the address limits of the emulator's code memory.

RS232 transmission problem - check board - the RS232 board could not perform a transmission. Check your RS232 board.

Search pattern is larger than search area - there are more bytes in the search pattern than in the specified search area.

SFR not in internal memory - a special function register was specified which does not reside within the internal data memory.

Source address is out of range - the address specified is outside the valid address range of the selected memory.

Symbolic debug not enabled - the symbolic debugging capability is not enabled. Symbolic addresses are therefore not allowed.

Sync - possible table disassembly - this error only applies when symbolic debug is enabled. Advantage is taken of the effect that the instruction labels must lie on instruction boundaries. If the disassembly passes a known instruction label without displaying it, then an address synchronization problem exists. Either the starting address was not at an instruction boundary or the disassembly process passed through a non-instruction portion of the code memory (i.e. a data table that lies in code memory space).

Target address exceeds relative address range - a relative jump was specified with the target address exceeding 127 bytes forward or 128 bytes backward.

'then' expected - the conditional portion of an experiment statement was not terminated by a 'then'.

Too many bytes in pattern - the pattern contained more than 32 bytes of data.

Too many characters - the new data value contained more than two hexadecimal digits. It can not therefore represent a byte value.

Too many macro parameters - more than 8 macro parameters were specified.

Unbalanced parenthesis - in evaluating an expression, the parenthesis in the expression were found not to balance. (Code Assembler).

Unbalanced parenthesis - there are an unequal number of opening and closing parenthesis used in the conditional portion of a statement. (Experiment compiler).

Undefined opcode class - the symbol used to specify an opcode class was not a valid opcode class symbol.

Undefined symbol - the symbol specified doesn't exist.

## B.3 DOS Errors Messages

The third class of errors which may be encountered are DOS operating system error messages. Refer to your DOS manual for details of the specific error messages.

APPENDIX C
SIGNAL SPECIFICATIONS AND DIFFERENCES

## C.1 Signal Specifications

Table 1 lists the signal specifications for the ICD 8031, 8032 and 8344 emulators. In addition, the specifications for Port 0 and Port 2 apply to the ICD 8051, 8052 and 8044 emulators when those ports are configured as address lines.

Table 2 lists the signal specifications for the ICD 8051, 8052 and 8044 emulators. The Port 0 and Port 2 specifications only apply when those ports are configured as I/O lines.

NOTES:

1. When not in the emulation mode, the PSEN RD (Port 3.7), WR (Port 3.6), and Port 0 pins are normally in the state listed in the tables. When an access is made to program code memory or external data memory which resides on your system board, these signals become active. They will exhibit their normal characteristics for the duration of the memory access before returning to the state listed in the tables.

2. Port 3.6 and Port 3.7 signals do not contain an active pullup when used as be-directional port pins. In that mode, they have a resistive pullup with 10K ohm impedence.

3. N.C. means not connected.

4. The Port 0 and Port 2 pins change values one state later than specified in the Intel Microcontroller Handbook when used as I/O pins.

TABLE 1

| Pin No. | Signal Name | Buffer Type | Output Drive | | Input Load | | Delay | Non-emulation Status |
|---|---|---|---|---|---|---|---|---|
| | | | High (mA) | Low (mA) | High (mA) | Low (mA) | nSec. Typ. | |
| 1 | Port1.0 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 2 | Port1.1 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 3 | Port1.2 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 4 | Port1.3 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 5 | Port1.4 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 6 | Port1.5 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 7 | Port1.6 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 8 | Port1.7 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 9 | Reset | LS14 | - | - | .02 | .4 | 17 | inactive |
| 10 | Port3.0 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 11 | Port3.1 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 12 | Port3.2 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 13 | Port3.3 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 14 | Port3.4 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 15 | Port3.5 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 16 | Port3.6 | LS05 | O.C. | 24.0 | .02 | .4 | 32 | normal |
| 17 | Port3.7 | LS05 | O.C. | 24.0 | .02 | .4 | 32 | normal |
| 18 | XTAL2 | none | - | - | - | 3.2 | 2 | normal |
| 19 | XTAL1 | none | - | - | - | - | - | normal |
| 20 | VSS | none | - | - | - | - | - | n.c. |
| 21 | Port2.0 | LS244 | 15.0 | 24.0 | - | - | 14 | normal |
| 22 | Port2.1 | LS244 | 15.0 | 24.0 | - | - | 14 | normal |
| 23 | Port2.2 | LS244 | 15.0 | 24.0 | - | - | 14 | normal |
| 24 | Port2.3 | LS244 | 15.0 | 24.0 | - | - | 14 | normal |
| 25 | Port2.4 | LS244 | 15.0 | 24.0 | - | - | 14 | normal |
| 26 | Port2.5 | LS244 | 15.0 | 24.0 | - | - | 14 | normal |
| 27 | Port2.6 | LS244 | 15.0 | 24.0 | - | - | 14 | normal |
| 28 | Port2.7 | LS244 | 15.0 | 24.0 | - | - | 14 | normal |
| 29 | PSEN | LS04 | .4 | 8.0 | - | - | 17 | high |
| 30 | ALE | LS04 | .4 | 8.0 | - | - | 17 | normal |
| 31 | EA | none | - | - | - | - | - | Vss |
| 32 | Port0.7 | LS245 | 15.0 | 24.0 | .02 | .4 | 10 | 3-state |
| 33 | Port0.6 | LS245 | 15.0 | 24.0 | .02 | .4 | 10 | 3-state |
| 34 | Port0.5 | LS245 | 15.0 | 24.0 | .02 | .4 | 10 | 3-state |
| 35 | Port0.4 | LS245 | 15.0 | 24.0 | .02 | .4 | 10 | 3-state |
| 36 | Port0.3 | LS245 | 15.0 | 24.0 | .02 | .4 | 10 | 3-state |
| 37 | Port0.2 | LS245 | 15.0 | 24.0 | .02 | .4 | 10 | 3-state |
| 38 | Port0.1 | LS245 | 15.0 | 24.0 | .02 | .4 | 10 | 3-state |
| 39 | Port0.0 | LS245 | 15.0 | 24.0 | .02 | .4 | 10 | 3-state |
| 40 | VCC | none | - | - | - | - | - | n.c. |

TABLE 2

| Pin No. | Signal Name | Buffer Type | Output Drive | | Input Load | | Delay | Non-emulation Status |
|---|---|---|---|---|---|---|---|---|
| | | | High (mA) | Low (mA) | High (mA) | Low (mA) | nSec. Typ. | |
| 1 | Port1.0 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 2 | Port1.1 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 3 | Port1.2 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 4 | Port1.3 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 5 | Port1.4 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 6 | Port1.5 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 7 | Port1.6 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 8 | Port1.7 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 9 | Reset | LS14 | - | - | .02 | .4 | 17 | inactive |
| 10 | Port3.0 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 11 | Port3.1 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 12 | Port3.2 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 13 | Port3.3 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 14 | Port3.4 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 15 | Port3.5 | none | .8 | 1.6 | .08 | .8 | 2 | normal |
| 16 | Port3.6 | LS05 | O.C. | 24.0 | .02 | .4 | 32 | normal |
| 17 | Port3.7 | LS05 | O.C. | 24.0 | .02 | .4 | 32 | normal |
| 18 | XTAL2 | none | - | - | - | 3.2 | 2 | normal |
| 19 | XTAL1 | none | - | - | - | - | - | normal |
| 20 | VSS | none | - | - | - | - | - | Vss |
| 21 | Port2.0 | LS05 | O.C. | 24.0 | .1 | .5 | note 4 | normal |
| 22 | Port2.1 | LS05 | O.C. | 24.0 | .1 | .5 | note 4 | normal |
| 23 | Port2.2 | LS05 | O.C. | 24.0 | .1 | .5 | note 4 | normal |
| 24 | Port2.3 | LS05 | O.C. | 24.0 | .1 | .5 | note 4 | normal |
| 25 | Port2.4 | LS05 | O.C. | 24.0 | .1 | .5 | note 4 | normal |
| 26 | Port2.5 | LS05 | O.C. | 24.0 | .1 | .5 | note 4 | normal |
| 27 | Port2.6 | LS05 | O.C. | 24.0 | .1 | .5 | note 4 | normal |
| 28 | Port2.7 | LS05 | O.C. | 24.0 | .1 | .5 | note 4 | normal |
| 29 | PSEN | LS04 | .4 | 8.0 | - | - | 17 | high |
| 30 | ALE | LS04 | .4 | 8.0 | - | - | 17 | normal |
| 31 | EA | none | - | - | - | - | - | Vcc |
| 32 | Port0.7 | LS05 | O.C. | 24.0 | .1 | .1 | note 4 | normal |
| 33 | Port0.6 | LS05 | O.C. | 24.0 | .1 | .1 | note 4 | normal |
| 34 | Port0.5 | LS05 | O.C. | 24.0 | .1 | .1 | note 4 | normal |
| 35 | Port0.4 | LS05 | O.C. | 24.0 | .1 | .1 | note 4 | normal |
| 36 | Port0.3 | LS05 | O.C. | 24.0 | .1 | .1 | note 4 | normal |
| 37 | Port0.2 | LS05 | O.C. | 24.0 | .1 | .1 | note 4 | normal |
| 38 | Port0.1 | LS05 | O.C. | 24.0 | .1 | .1 | note 4 | normal |
| 39 | Port0.0 | LS05 | O.C. | 24.0 | .1 | .1 | note 4 | normal |
| 40 | VCC | none | - | - | - | - | - | n.c. |

## C.2 Probe Cable Characteristics

The probe cable adds the following electrical properties to the pins as seen at the emulator probe:

+ 20pF / pin
+ 30uH / pin
+ 2nS / pin (the extra delay introduced by the cable has already been factored into the table above.)

## C.3 Signal Differences

### Signal Voltage Levels

PSEN, ALE, RD, WR, Port0, and Port2 all appear as TTL level signals rather than MOS level signals.

The standard voltage levels for the XTAL2 input when an external frequency source is supplied is:

Voh > 2.5V  Vol < .8V  (XTAL1 = Vss)

### Reset Pulses

The Reset from the system board should supply only one reset pulse. Any further reset pulses after the first has been accepted will be ignored for the rest of the emulation cycle.

### Power

The ICD does not draw its power from the system board. The electrical power requirements on your board will differ slightly after you replace the emulator probe with the actual target component.

APPENDIX D
OTHER DIFFERENCES

## D.1 Timer0 and Timer1 Values

The ICD automatically turns off all timer/counters when emulation stops. If a timer/counter is running when a breakpoint is encountered, it is stopped 8 machine cycles after the breakpoint.

If a timer/counter is turned on before running an experiment for the first time (by setting the appropriate bits in the TCON register via the Modify Registers command), it will actually be turned on 4 machine cycles before the first instruction of the experiment is executed.

If a timer/counter is turned on for an experiment which must be restarted (i.e. it has already executed until a breakpoint), it will be turned on 16 machine cycles before full speed emulation begins. Remember that when an experiment is restarted, a single step is automatically executed to move one instruction past the breakpoint instruction (See the Go command in chapter 6 for more details).

## D.2 Serial Port

The serial port is not stopped when a breakpoint is encountered. After a breakpoint, the interrupts are disabled. A serial port transmit or receive interrupt which occurs after the breakpoint has been encountered will be ignored.

## D.3 Port Registers

The values for the ports which are displayed the Modify Register menu represent the actual values at the port pins and not the value in the port registers. If you wish to change the value of one of the output pins, care must be taken to ensure that a 1 is written to any input pins in the same port (otherwise they will become output pins).

APPENDIX E
CHARACTER SETS AND RESERVED SYMBOLS

## E.1 Single Line Code Assembler Character Set

The legal character set for the single line code assembler is made up of the following characters (listed in ascending ASCII order):

```
# $ ' ( ) * + , - . / 0-9 : ? @ A-Z _ a-z
```

## E.2 Experiment Compiler Character Set

The legal character set for the experiment compiler is made up of the following characters (listed in ascending ASCII order):

```
! & ( ) * + - . / 0-9 < = > ? A-Z _ a-z
```

## E.3 Fill and Search Pattern Character Set

The legal character set for the fill and search pattern specification is made up of the following characters (listed in ascending ASCII order):

```
' 0-9 ? A-Z _ a-z
```

## E.4 Experiment Compiler Reserved Keywords

The following is an alphabetical listing of the reserved experiment compiler keywords:

```
AND, BADDR, BREAK, CLASS, DADDR, HIGH, IF, IMMED,
LOW, MOD, NOT, OPCODE, OR, PC, SHL, SHR, THEN, XOR
```

Some of these keywords are operators which can be used in an expression which specifies an address or a number. These operators are:

| | |
|---|---|
| AND | Bitwise logical AND |
| HIGH | High order 8 bits |
| LOW | Low order 8 bits |
| MOD | Modulus |
| NOT | Bitwise logical negation (1's complement) |
| OR | Bitwise logical inclusive OR |
| SHL | Shift left |
| SHR | Shift right |
| XOR | Bitwise logical exclusive OR |

Below is a listing of the precedence of all of the expression operators in descending order. Operations with higher precedence are performed first.

| Operator | Precedence |
|---|---|
| (,) | Highest |
| HIGH,LOW | |
| *,/,MOD,SHR,SHL | |
| +,- | |
| NOT | |
| AND | |
| OR,XOR | Lowest |

The following are examples of all the available operations and their result.

```
HIGH(OAADDh)      will return a result of OAAh
LOW(OAADDh)       will return a result of ODDh
7*4               will return a result of 28
7/4               will return a result of 1
                       (integer division)
7 MOD 4           will return a result of 3
1000b SHR 2       will return a result of 0010b
1010b SHL 2       will return a result of 101000b
10+5              will return a result of 15
+72               will return a result of 72
25-17             will return a result of 8
-1                will return a result of 1111111111111111b
NOT 1             will return a result of 1111111111111110b
1101b AND 0101b   will return a result of 0101b
1101b OR 0101b    will return a result of 1101b
1101b XOR 0101b   will return a result of 1000b
```

APPENDIX F
PREDEFINED BYTE AND BIT ADDRESSES

## F.1 8031 and 8051 Predefined Addresses

### F.1.1 Predefined Byte Addresses

```
P0      DATA    080H    ;PORT 0
SP1     DATA    081H    ;STACK POINTER
DPL     DATA    082H    ;DATA POINTER - LOW BYTE
DPH     DATA    083H    ;DATA POINTER - HIGH BYTE
PCON    DATA    087H    ;POWER CONTROL
TCON    DATA    088H    ;TIMER CONTROL
TMOD    DATA    089H    ;TIMER MODE
TLO     DATA    08AH    ;TIMER 0 - LOW BYTE
TL1     DATA    08BH    ;TIMER 1 - LOW BYTE
THO     DATA    08CH    ;TIMER 0 - HIGH BYTE
TH1     DATA    08DH    ;TIMER 1 - HIGH BYTE
P1      DATA    090H    ;PORT 1
SCON    DATA    098H    ;SERIAL PORT CONTROL
SBUF    DATA    099H    ;SERIAL PORT BUFFER
P2      DATA    0A0H    ;PORT 2
IE      DATA    0A8H    ;INTERRUPT ENABLE
P3      DATA    0B0H    ;PORT 3
IP      DATA    0B8H    ;INTERRUPT PRIORITY
PSW     DATA    0D0H    ;PROGRAM STATUS WORD
ACC     DATA    0E0H    ;ACCUMULATOR
B       DATA    0F0H    ;MULTIPLICATION REGISTER
```

### F.1.2 Predefined Bit Addresses

```
IT0     BIT     088H    ;TCON.0 - EXT. INTERRUPT 0 TYPE
IE0     BIT     089H    ;TCON.1 - EXT. INTERRUPT 0 EDGE FLAG
IT1     BIT     08AH    ;TCON.2 - EXT. INTERRUPT 1 TYPE
IE1     BIT     08BH    ;TCON.3 - EXT. INTERRUPT 1 EDGE FLAG
TR0     BIT     08CH    ;TCON.4 - TIMER 0 ON/OFF CONTROL
TF0     BIT     08DH    ;TCON.5 - TIMER 0 OVERFLOW FLAG
TR1     BIT     08EH    ;TCON.6 - TIMER 1 ON/OFF CONTROL
TF1     BIT     08FH    ;TCON.7 - TIMER 1 OVERFLOW FLAG
RI      BIT     098H    ;SCON.0 - RECEIVE INTERRUPT FLAG
TI      BIT     099H    ;SCON.1 - TRANSMIT INTERRUPT FLAG
RB8     BIT     09AH    ;SCON.2 - RECEIVE BIT 8
TB8     BIT     09BH    ;SCON.3 - TRANSMIT BIT 8
REN     BIT     09CH    ;SCON.4 - RECEIVE ENABLE
SM2     BIT     09DH    ;SCON.5 - SERIAL MODE CONTROL BIT 2
SM1     BIT     09EH    ;SCON.6 - SERIAL MODE CONTROL BIT 1
SM0     BIT     09FH    ;SCON.7 - SERILA MODE CONTROL BIT 0
EX0     BIT     0A8H    ;IE.0   - EXTERNAL INTERRUPT 0 ENABLE
ET0     BIT     0A9H    ;IE.1   - TIMER 0 INTERRUPT ENABLE
```

```
EX1     BIT     0AAH    ;IE.2   - EXTERNAL INTERRUPT 1 ENABLE
ET1     BIT     0ABH    ;IE.3   - TIMER 1 INTERRUPT ENABLE
ES      BIT     0ACH    ;IE.4   - SERIAL PORT INTERUPT ENABLE
EA      BIT     0AFH    ;IE.7   - GLOBAL INTERRUPT ENABLE
RXD     BIT     0B0H    ;P3.0   - SERIAL PORT RECEIVE INPUT
TXD     BIT     0B1H    ;P3.1   - SERIAL PORT TRANSMIT OUTPUT
INT0    BIT     0B2H    ;P3.2   - EXTERNAL INTERRUPT 0 INPUT
INT1    BIT     0B3H    ;P3.3   - EXTERNAL INTERRUPT 1 INPUT
T0      BIT     0B4H    ;P3.4   - TIMER 0 COUNT INPUT
T1      BIT     0B5H    ;P3.5   - TIMER 1 COUNT INPUT
WR      BIT     0B6H    ;P3.6   - WRITE CONTROL FOR EXT. MEMORY
RD      BIT     0B7H    ;P3.7   - READ CONTROL FOR EXT. MEMORY
PX0     BIT     0B8H    ;IP.0   - EXTERNAL INTERRUPT 0 PRIORITY
PT0     BIT     0B9H    ;IP.1   - TIMER 0 PRIORITY
PX1     BIT     0BAH    ;IP.2   - EXTERNAL INTERRUPT 1 PRIORITY
PT1     BIT     0BBH    ;IP,3   - TIMER 1 PRIORITY
PS      BIT     0BCH    ;IP.4   - SERIAL PORT PRIORITY
P       BIT     0D0H    ;PSW.0  - ACCUMULATOR PARITY FLAG
OV      BIT     0D2H    ;PSW.2  - OVERFLOW FLAG
RS0     BIT     0D3H    ;PWS.3  - REGISTER BANK SELECT 0
RS1     BIT     0D4H    ;PSW.4  - REGISTER BANK SELECT 1
F0      BIT     0D5H    ;PSW.5  - FLAG 0
AC      BIT     0D6H    ;PSW.6  - AUXILIARY CARRY FLAG
CY      BIT     0D7H    ;PSW.7  - CARRY FLAG
```

## F.2 8032 AND 8052 Predefined Addresses

### F.2.1 Predefined Byte Addresses

```
P0      DATA    080H    ;PORT 0
SP      DATA    081H    ;STACK POINTER
DPL     DATA    082H    ;DATA POINTER - LOW BYTE
DPH     DATA    083H    ;DATA POINTER - HIGH BYTE
PCON    DATA    087H    ;POWER CONTROL
TCON    DATA    088H    ;TIMER CONTROL
TMOD    DATA    089H    ;TIMER MODE
TL0     DATA    08AH    ;TIMER 0 - LOW BYTE
TL1     DATA    08BH    ;TIMER 1 - LOW BYTE
TH0     DATA    08CH    ;TIMER 0 - HIGH BYTE
TH1     DATA    08DH    ;TIMER 1 - HIGH BYTE
P1      DATA    090H    ;PORT 1
SCON    DATA    098H    ;SERIAL PORT CONTROL
SBUF    DATA    099H    ;SERIAL PORT BUFFER
P2      DATA    0A0H    ;PORT 2
IE      DATA    0A8H    ;INTERRUPT ENABLE
P3      DATA    0B0H    ;PORT 3
IP      DATA    0B8H    ;INTERRUPT PRIORITY
T2CON   DATA    0C8H    ;TIMER 2 CONTROL
```

```
RCAP2L   DATA   0CAH    ;TIMER 2 CAPTURE REGISTER - LOW BYTE
RCAP2H   DATA   0CBH    ;TIMER 2 CAPTURE REGISTER - HIGH BYTE
TL2      DATA   0CCH    ;TIMER 2 - LOW BYTE
TH2      DATA   0CDH    ;TIMER 2 - HIGH BYTE
PSW      DATA   0D0H    ;PROGRAM STATUS WORD
ACC      DATA   0E0H    ;ACCUMULATOR
B        DATA   0F0H    ;MULTIPLICATION REGISTER
```

F.2.2 Predefined Bit Addresses

```
IT0      BIT    088H    ;TCON.0 - EXT. INTERRUPT 0 TYPE
IE0      BIT    089H    ;TCON.1 - EXT. INTERRUPT 0 EDGE FLAG
IT1      BIT    08AH    ;TCON.2 - EXT. INTERRUPT 1 TYPE
IE1      BIT    08BH    ;TCON.3 - EXT. INTERRUPT 1 EDGE FLAG
TR0      BIT    08CH    ;TCON.4 - TIMER 0 ON/OFF CONTROL
TF0      BIT    08DH    ;TCON.5 - TIMER 0 OVERFLOW FLAG
TR1      BIT    08EH    ;TCON.6 - TIMER 1 ON/OFF CONTROL
TF1      BIT    08FH    ;TCON.7 - TIMER 1 OVERFLOW FLAG
R1       BIT    098H    ;SCON.0 - RECEIVE INTERRUPT FLAG
TI       BIT    099H    ;SCON.1 - TRANSMIT INTERRUPT FLAG
RB8      BIT    09AH    ;SCON.2 - RECEIVE BIT 8
TB8      BIT    09BH    ;SCON.3 - TRANSMIT BIT 8
REN      BIT    09CH    ;SCON.4 - RECEIVE ENABLE
SM2      BIT    09DH    ;SCON.5 - SERIAL MODE CONTROL BIT 2
SM1      BIT    09EH    ;SCON.6 - SERIAL MODE CONTROL BIT 1
SM0      BIT    09FH    ;SCON.7 - SERIAL MODE CONTROL BIT 0
EX0      BIT    0A8H    ;IE.0   - EXTERNAL INTERRUPT 0 ENABLE
ET0      BIT    0A9H    ;IE.1   - TIMER 0 INTERRUPT ENABLE
EX1      BIT    0AAH    ;IE.2   - EXTERNAL INTERRUPT 1 ENABLE
ET1      BIT    0ABH    ;IE.3   - TIMER 1 INTERRUPT ENABLE
ES       BIT    0ACH    ;IE.4   - SERIAL PORT INTERRUPT ENABLE
EA       BIT    0AFH    ;IE.7   - GLOBAL INTERRUPT ENABLE
RXD      BIT    0B0H    ;P3.0   - SERIAL PORT RECEIVE INPUT
TXD      BIT    0B1H    ;P3.1   - SERIAL PORT TRANSMIT OUTPUT
INT0     BIT    0B2H    ;P3.2   - EXTERNAL INTERRUPT 0 INPUT
INT1     BIT    0B3H    ;P3.3   - EXTERNAL INTERRUPT 1 INPUT
T0       BIT    0B4H    ;P3.4   - TIMER 0 COUNT INPUT
T1       BIT    0B5H    ;P3.5   - TIMER 1 COUNT INPUT
WR       BIT    0B6H    ;P3.6   - WRITE CONTROL FOR EXT. MEMORY
RD       BIT    0B7H    ;P3.7   - READ CONTROL FOR EXT. MEMORY
PX0      BIT    0B8H    ;IP.0   - EXTERNAL INTERRUPT 0 PRIORITY
PT0      BIT    0B9H    ;IP.1   - TIMER 0 PRIORITY
PX1      BIT    0BAH    ;IP.2   - EXTERNAL INTERRUPT 1 PRIORITY
PT1      BIT    0BBH    ;IP.3   - TIMER 1 PRIORITY
PS       BIT    0BCH    ;IP.4   - SERIAL PORT PRIORITY
CAP2     BIT    0C0H    ;T2CON.0- CAPTURE OR RELAOD SELECT
CNT2     BIT    0C1H    ;T2CON.1- TIMER OR COUNTER SELECT
TR2      BIT    0CAH    ;T2CON.2- TIMER 2 ON/OFF CONTROL
```

```
EXEN2   BIT     0CBH    ;T2CON.3- TIMER 2 EXTERNAL ENABLE FLAG
TCLK    BIT     0CCH    ;T2CON.4- TRANSMIT CLOCK SELECT
RCLK    BIT     0CDH    ;T2CON.5- RECEIVE CLOCK SELCTT
EXF2    BIT     0CEH    ;T2CON.6- EXTERNAL TRANSITION FLAG
TF2     BIT     0CFH    ;T2CON.7- TIMER 2 OVERFLOW FLAG
P       BIT     0DD0H   ;PSW.0  - ACCUMULATOR PARITY FLAG
OV      BIT     0D2H    ;PSW.2  - OVERFLOW FLAG
RS0     BIT     0D3H    ;PWS.3  - REGISTER BANK SELECT 0
RS1     BIT     0D4H    ;PSW.4  - REGISTER BANK SELECT 1
F0      BIT     0D5H    ;PSW.5  - FLAG 0
AC      BIT     0D6H    ;PSW.6  - AUXILIARY CARRY FLAG
CY      BIT     0D7H    ;PSW.7  - CARRY FLAG
```

APPENDIX G
PREDEFINED OPCODE CLASSES

The predefined opcode classes described here are contained in the file "OPCLASS.OPC" on the Zlink system disk.

## G.1 Opcode Class PGMFLOW

The opcode class PGMFLOW is used to set breakpoints on all instructions which can change the normal sequential flow of the program. These instructions include:

AJMP, SJMP, LJMP, ACALL, LCALL, RET, RETI, JBC, JB, JNB, JC, JNC, JZ, JNZ, CJNE, DJNZ, AND JMP @A+DPTR.

The following is the opcode class specification which defines this class:

| | Instruction | Operand1 Register | Operand2 Register |
|---|---|---|---|
| 1) | | CADDR | |
| 2) | | | CADDR |
| 3) | CJNE | | |
| 4) | RET | | |
| 5) | RETI | | |
| 6) | JMP | | |

Instruction specification 1 specifies the following instructions which have a code address as the first operand:

All AJMPs, LJMP, all ACALLs, LCALL, JC, JNC, JZ, JNZ, and SJMP.

Instruction specification 2 specifies the following instructions which have a code address as the second operand:

JBC, JB, JNB, and all DJNZs.

Instruction specification 3 specifies all of the CJNE instructions.

Instruction specification 4 specifies the RET instuction.

Instruction specification 5 specifies the RET1 instruction.

Instruction specification 6 specifies the JMP @A+DPTR instruction.

## G.2 Opcode Class STACK

The opcode class STACK is used to set breakpoints on all instructions which can change the target's internal stack. These instructions include:

PUSH, POP, ACALL, LCALL, RET, and RETI.

The following is the opcode class specification which defines this class:

| Instruction | Operand1 Register | Operand2 Register |
|---|---|---|

1) PUSH
2) POP
3) ACALL
4) LCALL
5) RET
6) RET1

Each instruction specification specifies one of the required instructions.

ZAX

Zax Corporation 2572 White Road, Irvine, California 92714
(714) 474-1170 • 800-421-0982 • TLX 183829