EB-51 Emulation Board

User's Manual

Rev. 04/19 - V3.00

CONTENTS

CHAPTER 4. NEW CEIBO WINDOWS DEBUGGER

CHAPTER 5. TRADITIONAL CEIBO WINDOWS DEBUGGER

CHAPTER 6. WORKING WITH THE HARDWARE

CHAPTER 7. REAL TIME EMULATION

CHAPTER 8. UTILITIES, SOFTWARE SUPPORT AND SYNTAX

CHAPTER 9. ON-LINE ASSEMBLER

CHAPTER 10. SYSTEM ERRORS AND TROUBLESHOOTING

CHAPTER 11. ABOUT THE DOS DEBUGGER

PREFACE

EB-51 Emulation Board

PREFACE

EB-51 Emulation Board

The EB-51 *Emulation Board* for PHILIPS 80C51 Microcontrollers and derivatives is very effective in meeting the diverse demands of emulation operations.

P.1. Features

- Emulates most 8051 derivatives
- 64K code and 64K data memory
- Memory with mapping capabilities
- Frequency range up to 40MHz
- DOS and Windows software
- Source-Level Debugger for C, PLM and Assembler
- Performance analyzer
- Serially linked to IBM PC at 115 KBaud
- 3.3V and 5V emulation support

P.2. The Software

The EB-51 is supplied with three software packages:

- CEB51D DOS Debugger
- Windows Debugger: New and Traditional Ceibo Debugger

All the debuggers have similar functions, although the Windows debugger is wholly based on a Windows platform (e.g., multi-tasking and multi-window display).

Refer to Chapter 11 of this manual for a complete description of the DOS debugger functions and instructions for operation. It is recommended to read this chapter in its entirety if you are planning on using this debugger.

If you will be using the Windows debugger, please refer to Chapters 3, 4 and 5.

The three working modes:

- Real time
- Simulator
- In-circuit simulator

are introduced in Chapter 1.

P.3. Emulation Restrictions and Troubleshooting

You must read carefully Chapter 1, especially the ***Emulation Support and Emulation Restrictions*** paragraphs before using the hardware system. These will explain how to prepare your project to take full advantage of the system.

Chapter 10 gives some ***troubleshooting*** recommendations to follow in case that you are experiencing problems with your hardware or software.

P.4. About the Manual

1. Description of the System

Describes the system and its capabilities, as well as detailing its specifications and applications.

2. Installation

Provides step-by-step instructions on software and hardware installation procedures. It is recommended that this chapter be read in its entirety prior to connecting the system.

3. About the Traditional Ceibo Windows Debugger

Includes the basic information you will need to begin using the Ceibo Windows Debugger. It also gives a session example for a quick introduction to the Windows Debugger capabilities.

4. New Ceibo Windows Debugger

Details the use of the New Windows Debugger menus and their related commands. This is a general description of the debugger which is used for different Ceibo products. Refer to Chapter 5 to learn about specific EB-51 commands.

5. Traditional Ceibo Windows Debugger

Details the use of the Windows Debugger menus and their related commands.

6. Working with the Hardware

Provides a session example of the In-Circuit Simulation Mode.

7. Real Time Emulation

Gives all the information regarding the system in real-time Emulation Mode.

8. Utilities, Software Support and Syntax

Explains the use of utilities provided to adapt different assemblers and compilers to Ceibo's Debuggers.

9. On-Line Assembler

Describes the syntax used by the On-line Assembler command. It also gives a complete list of examples of the instruction set.

10. System Errors and Troubleshooting

Lists the errors detected while operating EB-51 and gives common questions and answers for troubleshooting.

11. About the DOS Debugger

Imparts the information you will need to begin using the CEB51D program from DOS.

CHAPTER 1

Description of the System

CHAPTER 1

Description of the System

1.1. Introduction

This chapter describes the capabilities and applications of the system. The technical specifications and the emulation restrictions are detailed in the following paragraphs.

1.2. General Description

Ceibo EB-51 is a development tool that supports Philips 8051 microcontrollers at any frequency allowed by the devices. It is serially linked to a PC or compatible systems and can emulate the microcontrollers using either the built-in clock oscillator or any other clock source connected to the microcontroller. The clock oscillator generates 16MHz and the rated frequency divided by 1, 2 and 4. The rated frequency is supplied by the installed 24MHz crystal or any clock source up to 40MHz.

Emulation is carried out by loading the system with the user software and an embedded monitor program. EB-51 locates the monitor in the upper 1K of the code memory space.

Three working modes are available: real-time, simulator and in-circuit simulator.

In the ***real-time mode*** the user software is executed transparently and without interfering with the microcontroller speed. Breakpoints can be added to stop program execution at a specific address.

The ***simulation mode*** is used to debug the software without any hardware.

Therefore, EB-51 may be disconnected while using the simulation mode.

In the ***in-circuit simulation mode*** an additional microprocessor is used to take control of the microcontroller lines and to simulate its operation but not in real-time. This operating mode allows access to all the microcontroller functions (I/O, timers, etc.) and interacts with the hardware according to the user software execution or directly by means of emulator commands sent from the host computer. The Trace, Complex Breakpoints, Performance Analyzer and many other useful functions are enabled in the in-circuit simulation mode.

The combination of all the available working modes allows an easy way to debug hardware and software functions.

The software includes C, PLM and Assembler Source Level Debugger, On-line Assembler and Disassembler, Software Trace, Conditional Breakpoints and many other features. The system is supplied with DOS and Windows debugger software, RS-232 cable and a power supply.

1.3. Applications

The main applications of EB-51 Emulation Board are:

- Emulation of microcontrollers
- Demonstration of microcontroller capabilities
- Development of microprocessor based systems
- Hardware and software debugging purposes
- Training in the field of microprocessors

1.4. Specifications

System Memory

EB-51 provides 64K of user code memory and 64K of user data memory. This RAM memory permits downloading and modifying of user's programs and variables.

Code Memory

Code memory is mapped as belonging to the EB-51 Emulation Board. The system includes 64 KBytes of RAM to be used as code memory. However, this RAM memory can be used partially and up to 63K.

Data Memory

Data memory can be mapped as belonging to the emulator or to the target circuitry. The system includes 64 KBytes of RAM to be used as data memory.

Breakpoints

Breakpoints allow real-time program execution until an opcode is executed at a specified address.

User Software

The CEIBO DOS Debugger may be installed to run under DOS or Windows 3.X or later. The program is based on pull-down menus. The CEIBO Windows Debugger runs only under Windows 3.X or later.

Symbolic Debugger

EB-51 allows symbolic debugging of assembler or high-level languages. The symbolic debugger uses symbols contained in the absolute file generated by the most commonly used Assemblers and High Level Language Compilers.

Source Level Debugger

The EB-51 software includes a source level debugger for Assembler and High-Level Languages (PLM, C and others) with the capability of executing lines of the program while displaying the state of any variable.

Software Trace

Program execution can be recorded in a 64K buffer. Conditional breakpoints may be defined to stop program execution. The user can define events and variables to be added to the software trace. The software trace is not a real-time function and is performed by slowing down the emulation speed.

Supported Microcontrollers

The supported microcontrollers are most of the Philips 80C51 microcontrollers in both ROMless and ROMed versions. EB-51 also has different daughter boards to emulate other microcontrollers. All standard 40-pin DIP and 44-pin PLCC 8051 derivatives can be emulated with the standard system.

Microcontroller Selection

EB-51 uses standard Philips microcontrollers for hardware and software emulation. The selection of a different microcontroller is made by replacing the microcontroller on the board. EB-51 runs at a frequency of the crystal or from the clock source supplied by the user hardware. The minimum and maximum frequencies are determined by the emulated chip characteristics, while the emulator maximum frequency is 40MHz.

Frequency

The system includes a crystal oscillator able to supply clock frequencies of 24MHz, 12MHz and 6MHz. Additionally, the user may select any other frequency by connecting an external clock source through the application hardware. The crystal oscillator itself is mounted on a socket and may be replaced by another oscillator with different frequency value. Frequency selection is done by means of jumpers.

Host Characteristics

PC or compatible systems with 1 MByte of RAM, one RS-232C interface card for the PC, DOS or Windows 3.X or later.

Input Power

5V, 1.5A power supply supplied.

Mechanical Dimensions

10cm x 10cm.

Items Supplied as Standard

Development tool with 128 KByte Memory. One Philips 8051 microcontroller or derivative, 40-pin Emulation Header, DOS and Windows software including source level debugger, on-line assembler and disassembler, User's Manual, RS-232 Cable and Power Supply.

Options

Daughter boards and emulation headers for the different microcontrollers. Adapter for 44-pin PLCC devices.

1.5. Hardware Description

The first step required to work with the EB-51 board is to be able to identify its different parts and to understand how the electronics function. This will help you to take full advantage of all the available capabilities of the EB-51. On the upper left side of the EB-51 you will see a phone jack. It is used to serially link the EB-51 to your computer and to utilize the software emulation mode instead of just the simulator. The emulation mode is used to interact with the hardware while the simulator is independent of any hardware connection. The RS-232 cable connections are given separately.

The next part you should identify is the DC POWER jack located on the bottom left section of the board. This connector is used to apply a regulated DC voltage to the board. You should use the power supply included with the system. If you are using another power supply, the plug must be with the plus on the tip.

A GND testpoint is located on the bottom right corner of the board.

EB-51 emulation frequency is defined by the 24MHz crystal oscillator installed on a socket. The 24MHz is the maximum frequency you can apply to the microcontroller while emulating the on-chip ROM. In ROMless mode the maximum frequency may be up to 40MHz, depending on the microcontroller installed in the board. A lower frequency is achieved by dividing the 24MHz according to a jumper selection.

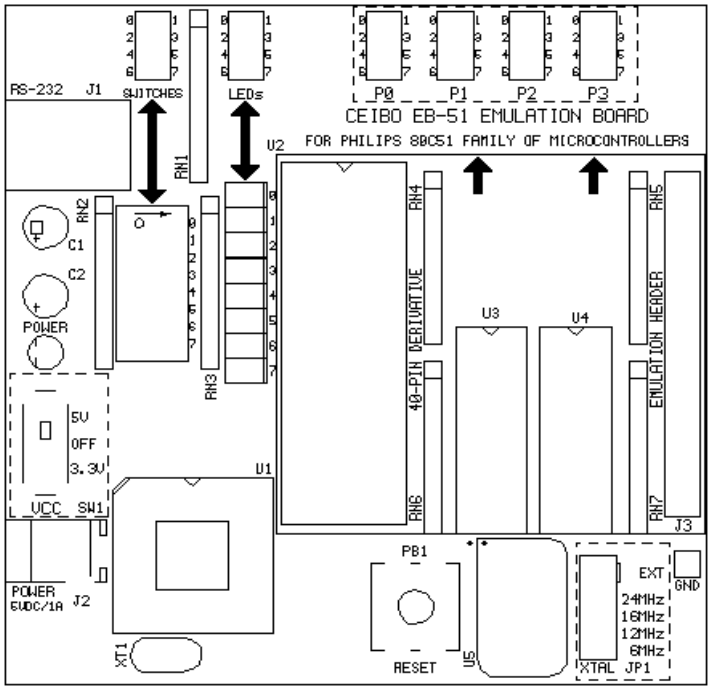


FIGURE 1.1: ***Circuit Layout***

JP1 on EB-51 emulator board can select one of the following frequency options:

1. 6 MHz (divide by 4).
2. 12 MHz (divide by 2).
3. 24 MHz.
4. External clock can be selected by placing the jumper cap in the EXT position.
5. The 16MHz frequency is supplied by the second fixed crystal of the board.

Use one jumper cap to select one of the available internal frequencies. For example, the following figure shows the setting for 12MHz:

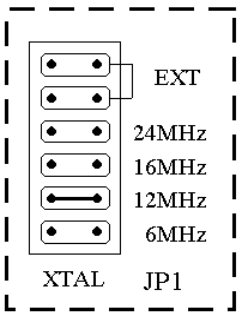


FIGURE 1.2: ***12MHz Setup***

You can connect any frequency you desire to the microcontroller from an external clock source. This can easily be done by setting the JP1 jumper on the bottom right corner of the board to EXT and applying any clock to the corresponding clock input pin of the emulation header.

In case you select the EXT option, use two jumper caps to set the selection. This is shown in the following figure:

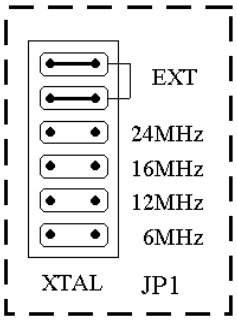


FIGURE 1.3: ***EXT Setup***

Different frequency options may be achieved by replacing the 24MHz crystal oscillator (U5) by another oscillator. That is possible because EB-51 emulation is not frequency dependent and any value accepted by the microcontroller is also suitable for the emulation.

The normal setup of JP1 is 12MHz or a higher frequency. If you are using the 16MHz or 24MHz option, please check that the microprocessor on the board (U2) is able to run at such a frequency.

On the left side you will see a POWER LED indicating whether or not the power is applied to the system. The LED blinks at a 0.1s rate while detecting an error or at 0.6s ON/ 0.2s OFF while executing a program in real-time.

J3 is the emulation header. You may connect the supplied flat cable with the 40-pin socket attached to J3. The connector pin-out is given in the following figure and it is a mirror image of the microprocessor pin-out.

On the top of the board you will find six 8-pin connectors. They may be used to easily interface EB-51 to an application board or to carry out test programs using the on-board resources. The SWITCHES connector provides logic states to input ports. The LEDs connector permits to connect any logic signal and to observe its logic state.

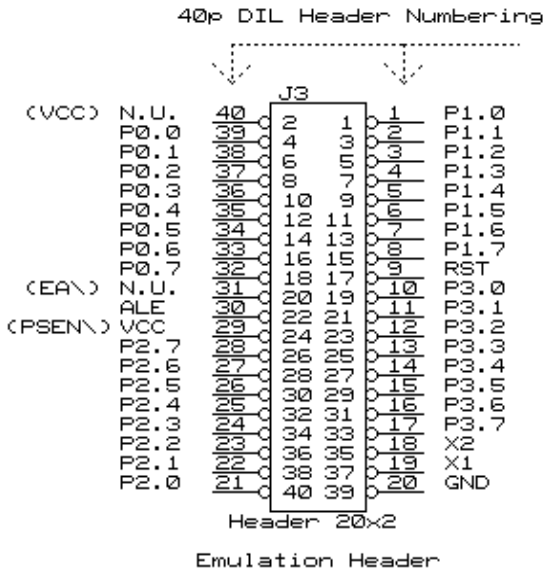


FIGURE 1.4: ***Emulation Header***

Port 0 to 3 are accessed through the remaining 8-pin connectors, additionally to the emulation header. Port 0 and 2 are taken from the reconstructed signals. That means, these ports are being recreated and are not the direct bus signals from the microcontroller.

EB-51 is supplied with an 8-wire ribbon cable to connect the ports to LEDs and Switches, thus allowing to easily test your program.

1.6. Emulation Support

The list of emulation support for standard EB-51 systems is as follows:

Microcontroller on EB-51	Supported Microcontrollers
Philips 80C32 or 87C52	80C31/2, 80C51/2, 87C51/2
Philips 87C524 or 87C528	8xC524, 8xC528
Philips 87C51FB	8xC51FA/FB/FC
Philips 87C550	8xC550
Philips 87C654	8xC652, 8xC654

TABLE 1.1: ***Supported Microcontrollers***

Lock bits 1 and 2 **must** be programmed in the microcontroller on the probe (even if it is 80C32). Programming these security bits is the required operation needed by the Philips microcontrollers to activate the "ROM emulation mode". Using different microcontrollers not listed above or not programming the security bits may cause the system to operate only in ROMless mode.

3.3V and 5V operation may be achieved by using a 3.3V set of chips that are able to operate at both voltages. This kit includes the four components to replace U1 to U4.

The following table gives the 3.3V and 5V supported devices:

Microcontroller on EB-51	Supported Microcontrollers
Philips 87L51FB	8xL51FA/FB/FC

TABLE 1.2: ***3.3V and 5V Supported Microcontrollers***

By using different daughter boards, you can emulate additional 80C51 derivatives. The addition of the 80C552 daughter board allows emulation of the microcontrollers listed in the following table:

Microcontroller on EB-51	Supported Microcontrollers
Philips 87C552	80C552, 83C552, 87C552, 80C562, 83C562, 87C562

TABLE 1.3: ***80C552 Supported Microcontrollers***

As the list of supported devices and available daughter boards is continuously evolving, call Ceibo to receive the latest update.

1.7. Emulation Restrictions

The following restrictions are valid for EB-51:

1. EB-51 Monitor Program shares 1 KByte of the 64K memory code space. Therefore, user programs can be up to 63 KBytes.

2. Code memory cannot be mapped external and always belongs to the emulator system.

3. The program also uses 3 Bytes of the internal stack memory.

Port 3.6 and 3.7 (RD and WR) lines may be used for MOVX operations and not as quasi-bidirectional I/O ports.

4. MOVX @Ri instructions affect Port 2 state. During the execution of these instructions Port 2 becomes an output port.

5. Port 2 outputs are recreated as open-drain with 10 KOhm pull-up resistors. That implementation may affect the rise-time when switching from 0 to 1.

6. The stack pointer may not be defined below address 7.

7. Emulation is possible at 5V only. 3.3V support requires microprocessors and two PLDs working at such voltage (U1 to U4).

8. Memory requirements of the Ceibo DOS Debugger are around 400 KBytes of free conventional memory for minimum operation. You will not be allowed to enter the DOS Debugger at all with less free memory. However, for complete operation of all nested windows in full zoom, you may require more free

memory. If the extra optional memory is not available, any operation which may require it will fail, with an appropriate error message. Try to free as much conventional memory as possible before running the DOS Debugger.

9. The Ceibo DOS Debugger Symbol information limitation is as follows:

- Local symbols limitation allows up to 2000 Local symbols records.
- Global (Public) symbols allowed are up to 2500 Global symbols records.
- Code Line symbols allowed are up to 500 Line symbols records.
- The total allowed number of Modules and Procedures (Functions) is 500.
- The total allowed number of Module Names is 500.
- The total allowed Number of Lines in any single module is 4000.

Each record may contain more than one symbol depending on the Compiler/Assembler output. If any of the above limitations are exceeded, a warning message will be displayed while loading, and the rest of the symbol information of that type will be ignored.

Debugging however will still be possible without access to the ignored symbols.

If the number of Modules and Procedures exceeds the above limitation, then the Source Level Debugger will not function on those Modules and Procedures and you will be able to debug them using the CPU window. Whenever encountering the above limitations, try to reduce unnecessary symbol information by applying the NODEBUG directive in modules which are of no interest to the current debugging cycle.

11. The Simulator can simulate timers only in Timer Mode and while the Gate control is 0.

CHAPTER 2

Installation

CHAPTER 2

Installation

2.1. Introduction

This chapter describes hardware and software installation procedures and details the connection steps required before starting up.

2.2. RS-232C Interface

EB-51 is serially linked to a host computer through an RS-232C interface. A standard phone cable is used to connect the EB-51 to COM1, 2, 3 or 4 in the host computer. This cable has a 9-pin female connector to fit into the COM1 (or other) connector, and a phone plug to connect to the EB-51 serial interface jack.

If your computer has a 25-pin connector for the RS-232 interface, use an additional 9-pin to 25-pin adapter.

The cable characteristics are given in Table 2.1.

TABLE 2.1: ***RS-232 Cable***

Signal Name	Phone Jack (EB-51)	9-Pin Female Connector (PC Side)
Receive Data	pin 1	pin 2
Transmit Data	pin 2	pin 3
Signal Ground	pin 4	pin 5

2.3. Connecting the EB-51 Components

No special tools are required to install the EB-51.

For proper installation the host must be properly configured and the power should be OFF.

Carry out the following steps to connect the system components:

a) Connect the serial cable into the serial jack on EB-51.

b) Connect the other end of the serial cable to host PC serial channel COM1 (COM2, COM3 or COM4).

c) Plug the power supply into the power connector.

d) Plug the other end of the cable into a power outlet.

2.4. Installing the Software

Follow the steps described below to install your EB-51 software:

1. Turn on the host computer.
2. Insert your EB-51 disk into Drive A or B.
3. If you are installing the Windows Debugger:
4. Run Windows.
5. From the Program Manager select the drive with the Windows Debugger disk and run SETUP.

If you are installing the CEB51D DOS Debugger:

Type INSTALL and follow the indications given in the installation procedure. The software will allow you to install the debugger in any directory and to run it under DOS or Windows.

Complete Install and Setup utilities are supplied with the CEIBO Debuggers. All the Debugger files are compressed and expanded during installation. The Windows Setup creates the Ceibo Windows Debugger icon.

Follow the instructions on the screen. If instructed, the DOS Debugger can also be setup for automatic Windows installation. This will automatically create and update groups and icons the next time you run Windows.

When running DOS 5.0 or a later version, it is recommended to use the smartdrive utility supplied with DOS.

This will increase the Debugger's performance dramatically, especially when working with large files having a lot of symbol information.

Refer to your DOS manual or Help utility for more information on smartdrive installation and usage. Any other similar utility is also satisfactory.

The Ceibo DOS Debugger fully supports Windows operations when running as a Windowed or as a Full Screen DOS application (Windows 3.0 and up).

The Debugger's CEB51D.pif file is factory configured to Windowed Display Usage.

You can change the configuration to Full Screen with the PIF editor. Please consult your Windows user's manual for more information on the PIF editor and pif files.

2.5. Starting Up

Turn the EB-51 power supply on. Set the power switch to the 5V position.

If you are installing the Windows Debugger, double click the Debugger icon. Otherwise, for the DOS debugger type:

C> CEB51D

and press the <ENTER> key.

The system will display a copyright screen after successfully invoking the software. Now the system is ready for operation.

If the software responds with a message indicating that the system is not connected, check the following:

1. Power supply - the LED must be on.
2. RS-232 - check the connections to your computer.
3. Communication port - verify that the connected serial port corresponds to the setup of the system.

2.6. RS-232 Interrupt

EB-51 DOS software does not use any interrupt request from the serial COM port.

For multitasking operation under Windows or other environments, use the /IRQ switch while invoking the DOS software. This will force the communications to use:

1. IRQ4 for COM1 or COM3
2. IRQ3 for COM2 or COM4

Make sure your RS-232 adapter configuration supports this setting.

Any resident connected to the same IRQ will be disabled until the CEB51D Debugger operation is terminated.

The syntax for this option is:

CEB51D /IRQ

More information about command lines is given in Chapter 11.

Normal invocation of the CEIBO CEB51D Debugger will set the communication to Polling Mode. In most cases this mode of operation will work without problems, both in Low and High baud rates.

When running the CEIBO CEB51D Debugger under Windows, or any other multitasking environment, set the communication to Interrupt Mode.

This is done by applying the /IRQ switch on the CEIBO Debugger invocation line command.

The CEIBO DOS Debugger assumes default IRQs when enabled: IRQ4 for COM1 and COM3, IRQ3 for COM2 and COM4. If none of these IRQs is available on your system (both are used concurrently with the Debugger), do not invoke the CEIBO CEB51D Debugger with the /IRQ switch.

One of the symptoms of time consuming residents running in the background,

such as smartdrive on a very slow hard disk, is occasionally having an unexplained Communication Error message. If you encounter such problems try setting the Interrupt Mode as explained above.

2.7. DOS Mouse Operation

The CEIBO CEB51D DOS Debugger automatically activates and supports mouse operation when a mouse driver is found in memory upon invocation of the Debugger.

Be sure to load the mouse driver into memory if you wish to operate the CEIBO DOS Debugger with the mouse.

When setting the Debugger in Interrupt communication Mode with the /IRQ switch, be sure that the mouse driver is not configured to be on the same IRQ, otherwise both the mouse and the Debugger will not function correctly.

2.8. Using The Mouse with CEB51D DOS Debugger Under Windows

Windows 3.1 or later fully supports mouse operation in a Windowed DOS application under Windows. However, this is restricted to several mouse drivers that support this mode of operation.

The Logitech mouse driver supplied with Windows 3.1 or later as well as the Microsoft Mouse driver support this operation. Most other mouse vendors are compatible and can operate with one of the above drivers.

Please consult your mouse vendor for the latest driver update supporting Windowed DOS application under Windows.

Logitech Mouse driver installation

1. Install the Logitech mouse under Windows by using the Windows Setup utility found in your Main window. You should then have the following files in your Windows directory: lvmd.386, lmouse.com.

2. Change the system.ini file found under your Windows directory to the following:

3. In [386Enh] section, check that the mouse driver is: mouse=lvmd.386

4. In [NonWindowsApp] section, add: MouseInDosBox=1

5. You must run the lmouse.com mouse driver supplied with Windows in your DOS environment (normally in your autoexec.bat file) before invoking Windows for correct operation.

Microsoft Mouse Driver Installation

1. Install the Microsoft Mouse under Windows by using the Windows Setup utility found in your Main window. You should then have the following files in your Windows directory : mscvmd.386, mouse.com.

2. Change the system.ini file found under your Windows directory to the following:

   In [386Enh] section, check that the mouse driver is: mouse=mscvmd.386

   In [NonWindowsApp] section, add: MouseInDosBox=1

3. Run the mouse.com mouse driver supplied with Windows in your DOS environment (normally in your autoexec.bat file) before invoking Windows.

More information on mouse operation can be found in your Windows user's manual or Readme utility.

2.9. Installing a Daughter Board

Follow the steps described below to install a Daughter Board:

1. Ensure that the power is OFF to EB-51.

2. To connect a daughter board to EB-51, remove the 40-pin microcontroller (U2) and the emulation ribbon cable from J3. Align and press the connectors together until they are plugged into the EB-51 main board.

2.10. Changing the Header

EB-51 has different headers to emulate the different packages and pin-outs of the microcontrollers.

Follow the steps described below to replace the header:

1. Ensure that the power is OFF to EB-51.

2. To connect a different header to EB-51, align and press the connectors together until the ribbon cable is plugged in.

2.11. Changing the Microcontroller

EB-51 may be used with different microcontrollers that are pin to pin compatible with the socket or the adapter of your system.

Follow the steps described below to replace the header:

a.) Ensure that the power is OFF to EB-51.

b.) Remove the microcontroller from the socket. Please note that only the microcontroller on socket U2 - 40-pin DIP or that on the special adapter may be changed and ***not*** the microcontroller placed on socket ***U1 - 87C51FB - 44-pin PLCC.***

CHAPTER 3

About the Windows Debugger

CHAPTER 3

About the Windows Debugger

3.1. Introduction

This chapter includes the basic information you will need to begin using the Ceibo Windows Debugger program and to understand the meaning of the different menus.

3.2. Debug Capabilities

The Windows Debugger is used to load a program, execute it in real-time, simulate the software environment as well as many other functions.

You may be familiar with other debuggers' nomenclature. Nevertheless, the following review is useful to understand some terms used by The Ceibo Windows Debugger.

Tracing

A program may be executed one line at a time. You can trace a program using high-level language lines or assembly instructions.

Stepping

This is like tracing but program execution steps over CALL instructions without leaving the current procedure.

Viewing

Ceibo Windows Debugger opens special windows showing your program state from various perspectives: variables and their values, breakpoints, a source file, CPU registers, memory, peripheral registers, etc.

Inspecting

The debugger can delve deeper into your program and show you the variable contents.

Changing

The current value of a variable can be replaced with your specified value.

Watching

Program variables can be isolated and their values kept track of while the program runs.

3.3. Global Menus

A Global Menu is the list of commands easily accessible from a bar which runs along the top of the window.

A pull-down menu is available for each item on the menu bar and allows the following:

- Execute a command.
- Open a pop-up menu. Pop-up menus appear when a menu item is chosen followed by a menu icon (▶).
- Open a dialog box. Dialog boxes appear when a menu item is chosen and they are indicated as (**...**).
- Check an option to select it.

Global menus are accessed by pressing F10 and using the arrow keys, pressing Alt and typing the first letter of the menu name or clicking the option.

Some of the menu commands have hot key shortcuts that are available from any part of the Debugger.

3.4. Local Menus

The Windows Debugger is context-sensitive and uses Local Menus specifying different windows. Local menus are tailored to the particular window you are in. It is important not to confuse them with global menus.

To prompt a local menu press Alt-F10 or click the right button of your mouse.

Menu placement and contents depends on which window or pane you are in and where your cursor is.

Contents may vary from one local menu to another. Many local commands appear in almost all local menus. The results of these similarly-named commands may differ, depending on the context.

Every command on a local menu has a hot key shortcut consisting of Ctrl plus the underscored letter in the command.

Because of this setup, a hot key, like Ctrl-C might mean one thing in one context but something quite different in another. The core commands are still consistent across local menus. For example, the Go To command and the Search command always do the same thing, even when they are invoked from different windows.

3.5. Input Boxes

Many of the Windows Debugger command options are available in input boxes. An input box prompts you to type in a string. All your entries are recorded in a history buffer, so you can pick up any entry just by selecting it with the arrows.

3.6. Windows

The Windows Debugger displays all information and data in both global and local menus, dialog boxes (where options are set and information entered) and windows.

There are many window types, depending on the kind of information it holds.

Windows may be opened and closed using menu commands (or hot key shortcuts for those commands).

After a window has been opened, you can move, resize, close, and otherwise manage them with commands from the Window and System menus.

3.7. Using the Menus

There are four ways to open the menus:

1. Press F10, use left or right arrow to get to the desired menu, press Enter.
2. Press F10, then the first letter of the menu name (F,V,R,B,D,O,W,H).
3. Press Alt plus the first letter of any menu bar command. For example, wherever you are in the system, Alt-F takes you to the File menu.
4. Click in the menu bar command with the mouse.

To navigate within the global system:

1. Use left and right arrows to move from one pull-down menu to another.
2. Use up and down arrow to scroll through the commands in a specific menu.
3. Highlight a menu command and press Enter to move to a lower-level (pop-up) menu or dialog box.

To get out of a menu or the menu system:

1. Press Esc to exit a lower-level menu and return to the previous menu.
2. Click the active window with the mouse to leave the menu system and return to the active window.
3. Press F10 to return to the current active window.

Some menu commands have a shortcut "hot key" that you press to execute them. The hot key appears in the menu to the right of these commands.

3.8. Toolbar

FIGURE 3.1: ***Tololbar***

The buttons on the ***Toolbar*** are the commands you need to operate the most useful functions:

 get help information

 open a file dialog box

 open the list of Modules dialog box

 select the CPU window

 select the Watches window

3.9. Status Line

FIGURE 3.2: ***Status Line***

The ***status line*** on the bottom of the main application window displays messages related to the cursor position in the Module window, chip type, operating mode (simulation, emulation or in-circuit simulation) and current status (program running, ready, error). It also provides on-line help information on selected menus.

3.10. Debbugging Session - Preparing the Software

1. Do not apply power to the EB-51. It is not necessary for the first stage which mostly explains the software capabilities.

2. Invoke the Windows Debugger by double clicking the Ceibo icon.

FIGURE 3.3: ***Debbuger Icon***

3.11. Accessing the Global Menu

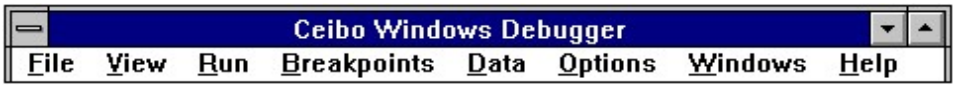


FIGURE 3.4: ***Global Menus***

1. The Global menu commands may be activated by simultaneously pressing the ALT and the command first letter keys.

2. After invoking the program, press Alt-V to open the View command. Select CPU and observe the new window added to the screen. The CPU window becomes the active window.

3. Open the Port Windows from the Target Command in the View Menu.

4. Press CTRL-F6 several times to select a different active window.

3.12. Accessing the Local Menu

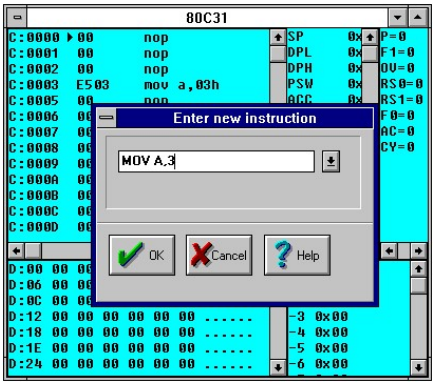


FIGURE 3.5: ***CPU Assembly Command***

1. The Local menu command may be accessed by pressing Alt-F10 or clicking the right button.

2. A direct access to a local menu command is possible by holding down the Ctrl key and pressing the letter that identifies the command.

3. Select the CPU window and check its Local Menu. The default is Assemble, meaning you can enter directly any assembly instruction.

4. Move the cursor to any line and type MOV A,3 directly. Observe how the code has been changed.

3.13. Selecting the Simulation Mode

Select the Simulation Mode from the Options Menu and the Architecture Command.

The bar on the top of the screen is the global menu.

The bottom bar displays the software status.

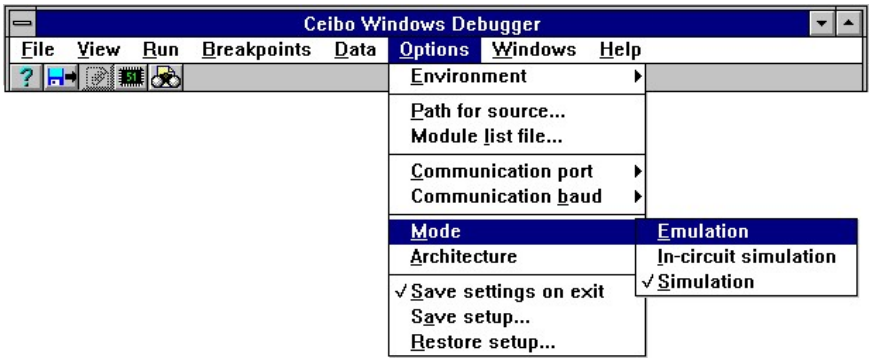


FIGURE 3.6: ***Simulation Mode***

3.14. Adding Watches

Open the Watches Window by selecting the View Menu and the Watch command.

FIGURE 3.7: ***Adding Watches***

Press the <INS> key or click the right button while the cursor points to somewhere inside the Watches Window. You may also add a watch by clicking the Watches button on the toolbar.

Type P1 and press the Enter key. Then, Port 1 will be added to the Watches Window. Note the your entry is case sensitive and P1 is not the same as p1.

3.15. Changing Watches

1. If you want to change the Port value, invoke the Local menu again and select the Change command. A selection may be done either by moving the arrows until the Change command is highlighted or by pressing the C key.

   Type 55 and press the Enter key. Observe that the Watches Window has an updated value for Port 1.

2. Click the OK button.

3. You can also change the watches by positioning the cursor on the variable and then pressing Ctrl-C.

4. The Language Command in the Options and Environment Menus determines the base and syntax of your entries. For example, if you choose C Language, 55 is a decimal value and 0x55 is a hexadecimal number. In case the Assembler is selected, you should type 55h to enter the same hexadecimal value. The syntax of your inputs is explained in the next chapter.

The Integer Display Format Command in the Options and Environment Menus defines the base of the display in the Watches Window. You can select hexadecimal or decimal display of your variables.

FIGURE 3.8: ***Changing Watches***

3.16. Watching Memory Spaces

1. You can add to the Watches Window any memory space as a succession of values. That may be achieved by specifying the type, initial address and length.

2. Add to the Watches Window the first 10 bytes of the on-chip RAM by typing D:0,10.

3. Add to the Watches 10 bytes of the code memory. Your entry may be C:1000,10.

4. Display 3 bits of the internal RAM. Type B:0,3.

5. Display 5 bytes of the external memory space accessed by MOVX instruction. Type X:100,5.

6. Add to the Watches Window any SFR by entering the absolute address. For example, type P:0x90,2 to display Port 1 and the following SFR (address 90H and 91H). 0x90 is 90H if you selected C language in the Language Command of the Options Menu. If your selection is Assembler, just type P:90H,2.

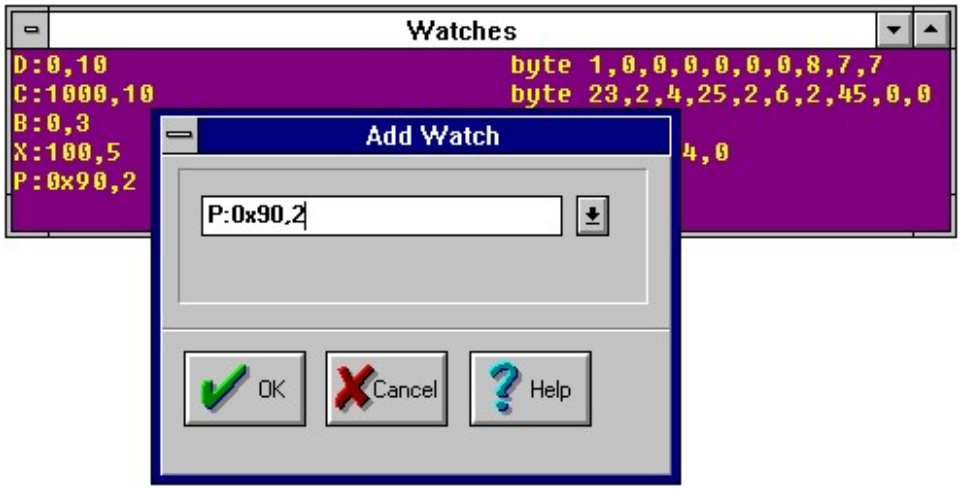


FIGURE 3.9: ***Adding Strings to the Watches***

3.17. Displaying a Memory Space

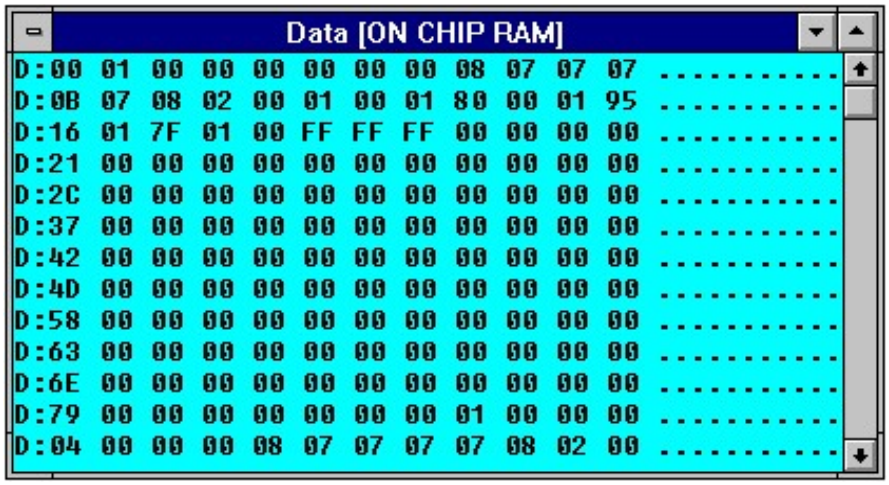


FIGURE 3.10: ***Data Window***

1. Open the Data Window by selecting the Memory Space Command in the View Menu.

2. Change the contents of this memory. Click the right button and select the Change Command.

3. Enter successive new values separated by spaces or commas: 11,22,33,44.

4. Check the changes in the Data Window.

3.18. Changing the Windows

1. Press Alt-W to select the Window menu, that permits changing the windows Try all the options given in this menu.

2. A window may be resized by moving the borders with the left button.

3. The arrows surrounding the window borders can be moved to position the desired information on the screen.

3.19. Loading a File

1. Press Alt-F to activate the File menu.

2. Set the cursor to highlight the Load option and press the Enter key.

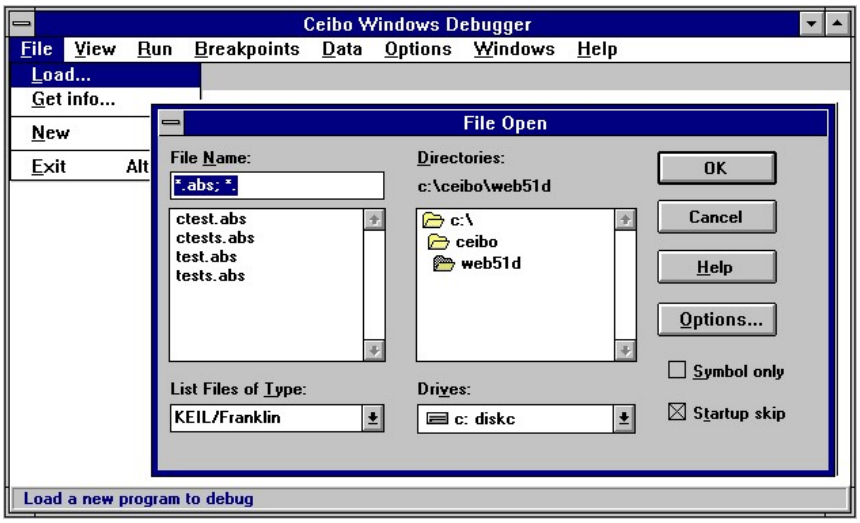


FIGURE 3.11: ***Loading a File***

3. Select the CTESTS.ABS sample file and Keil/Franklin to load code and symbols.

Note that it is very important to define the type of list files to get the proper symbol information. TEST.ABS and TESTS.ABS are Intel ASM/PLM51 files. CTEST.ABS and CTESTS.ABS are Keil/Franklin files.

You may also load a Hex file without symbol information, but in that case the symbolic debugger will not function.

4. After successfully loading the CTESTS.ABS file, the Module and Watches windows will be opened. If you try with a different file with incomplete debug information, the CPU window will be opened instead of the Module and Watches windows.

FIGURE 3.12: ***Default Screen***

3. 20. Capturing Watches

You can capture the name of a variable and include it in the Watches Window.

1. Open the Module Window with your source code.
2. Position the cursor on the variable name in any part of your source code.
3. Press Ctrl-W or use the right button to include the variable in the Watches Window.

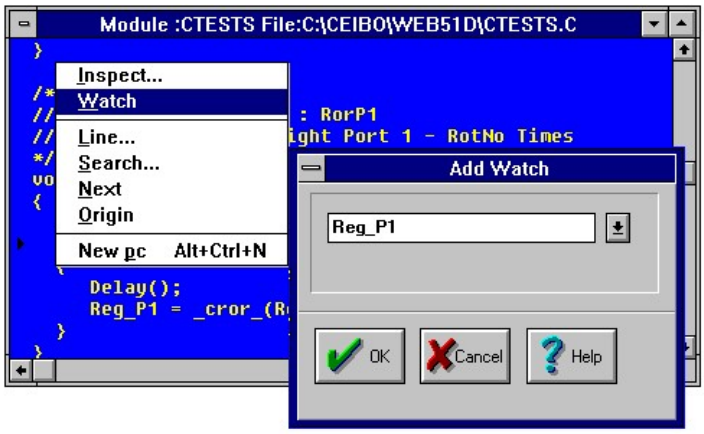


FIGURE 3.13: ***Capturing Watches***

3.21. Debugging the Program

1. Position the cursor on the Reg_P1 variable and click the left button. Click the right button and select the Watch command or press Ctrl-W to add the variable to the Watches Window.

2. Move and resize the three windows according to your convenience.

3. Open the CPU window.

4. Select the RUN Menu and execute a Program Reset. That can be done directly by pressing Ctrl-F2.

5. Execute a few assembly steps. These are available from the Run menu and the command name is Instruction Trace. Press the Alt-F7 keys several times.

6. Execute the program. Press the F9 key.

7. Halt the program execution by pressing Ctrl-Break.

8. Display the executed instructions from the View Menu, using the Trace Buffer command.

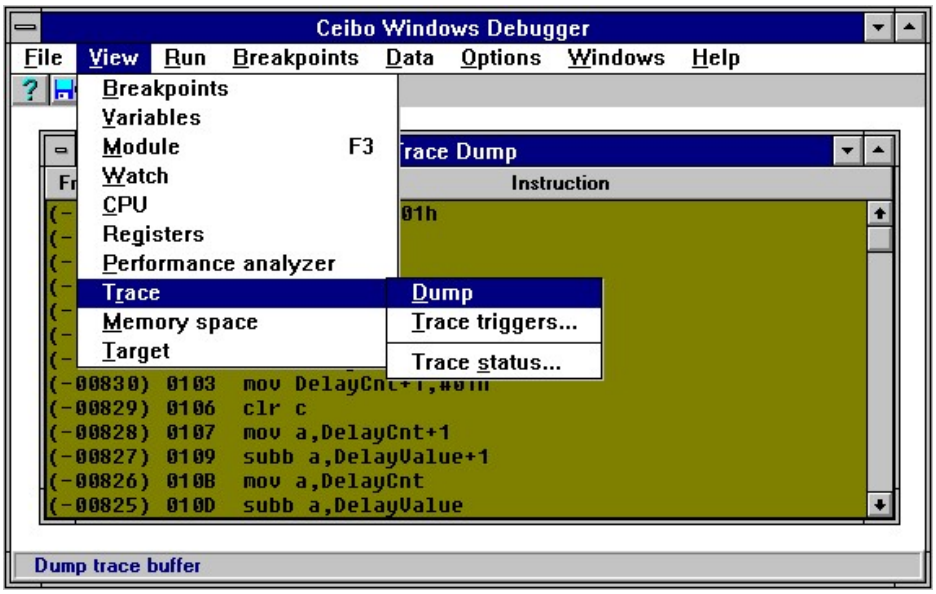


FIGURE 3.14: ***View Menu***

1. Set a Breakpoint by moving the cursor in the Module window and pressing the F2 key. Execute the program by pressing the F9 key.

2. Execute a high-level language step by pressing the F8 key. Repeat that operation several times.

3. Continue the high-level-language stepping by pressing F7. Note that the Modules are changed in accordance with the actual program counter value.

CHAPTER 4

New Ceibo Windows Debugger

CHAPTER 4

New Ceibo Windows Debugger

4.1. Introduction

CEIBO DEBUGGER is the latest software available from Ceibo and can be used with most of Ceibo emulators. You will find a detailed description of the menus and their related commands in the following paragraphs. ***As the debugger is general for all the 8051 derivatives, some specific functions may not apply to all the derivatives.***

4.2. Global and Local Menus

- A Global Menu is the list of commands easily accessible from a bar which runs along the top of the window. A pull-down menu is available for each item on the menu bar and allows executing a command, opening a pop-up menu and checking an option to select it. Global menus are accessed by using the mouse to click the option, pressing F10 and arrow keys or pressing Alt and typing the underlined letter of the menu name.

- Ceibo Debugger is context-sensitive and uses Local Menus specifying different windows. Local menus are tailored to the particular window you are in. To prompt a local menu press Alt-F10 or click the right button of your mouse. Menu placement and contents depends on which window or pane you are in and where your cursor is.

4.3. Toolbars

The buttons on the ***Toolbar*** are the commands you need to operate the most useful functions:

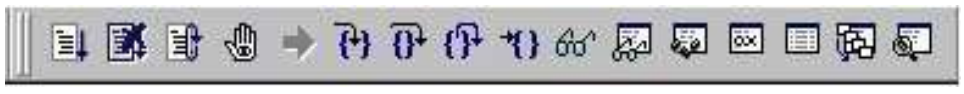


FIGURE 4.1: ***Debug bar***

Debug bar buttons from left to right: Go, Break, Restart, Toggle Breakpoint, Origin, Trace Into, Step Over, Step Out, Go to Cursor, Output, Watch, Variables, Modules and Registers, Memory, Stack and CPU.

FIGURE 4.2: ***Windows bar***

Windows bar buttons from left to right: New Windows, Split, Cascade Windows, Tile Horizontal and Tile Vertical.

FIGURE 4.3: ***Edit bar***

Edit bar buttons from left to right: Cut, Copy and Paste.

FIGURE 4.4: ***Main Bar***

Main bar buttons from left to right: Open File, Find File, About and Help

4.4. Status Line

The ***status line*** on the bottom of the main application window displays messages related to the cursor position in the Module window, chip type, operating mode (simulation or emulation) and current status (program running, ready, error). It also provides on-line help information on selected menus.

Important: if SIM appears on that bar, the system is in SIMULATION MODE and your code will not run in real time. Select EMULATION MODE in the Options Menu and connect your Ceibo Emulator to your computer.

FIGURE 4.5: ***Status Bar***

4.5. File Menu

The File menu options deal with operations external to the Ceibo Debugger, such as loading programs for debugging, and leaving the application. The available commands are: Load, Close, Unload, Print, Print Preview and Exit. This menu also provides a list of the last opened files, so you may load a file just by clicking on the desired file name.

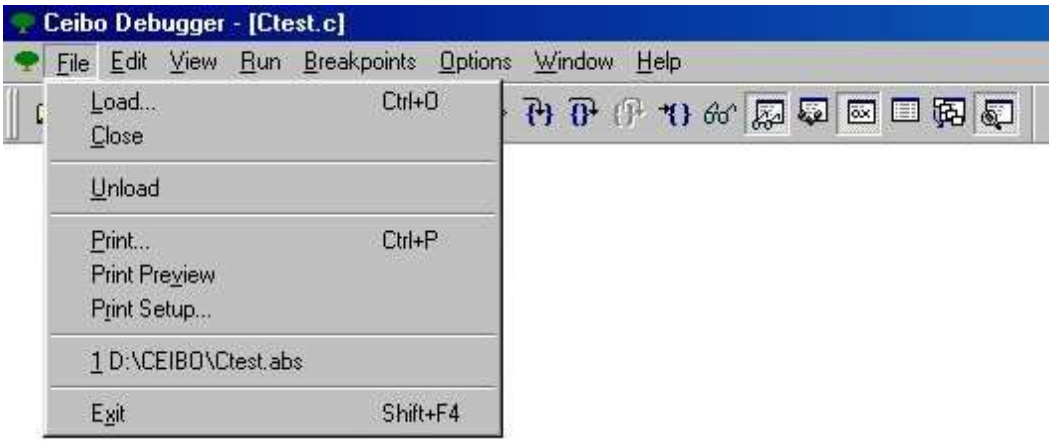


FIGURE 4.6: ***File Menu***

Load

The Load command loads a program for debugging from a disk. You may select the directory and the file name to be loaded, as well as the format of the file to achieve complete debug information compatibility.

The supported file formats are: Intel Hex, OMF51, Keil, IAR-UBROF, Tasking-IEEE695 and many others.

Utility programs and software updates may be released in the future to support new compilers with different formats.

From the File Menu you can specify the Symbol Only option, thus loading only the symbol information in the file for debugging ROM applications. Code is not loaded in this case.

The Startup Skip option is used to run the program automatically until the first line of your main program is reached.

If selected, the program will run from 0000h to the Main label while debugging C. ***Uncheck the Skip Startup Code box if you want to start debugging from address 0000h.***

Select the software vendor before loading a file. Use the File of Type list to make your selection.

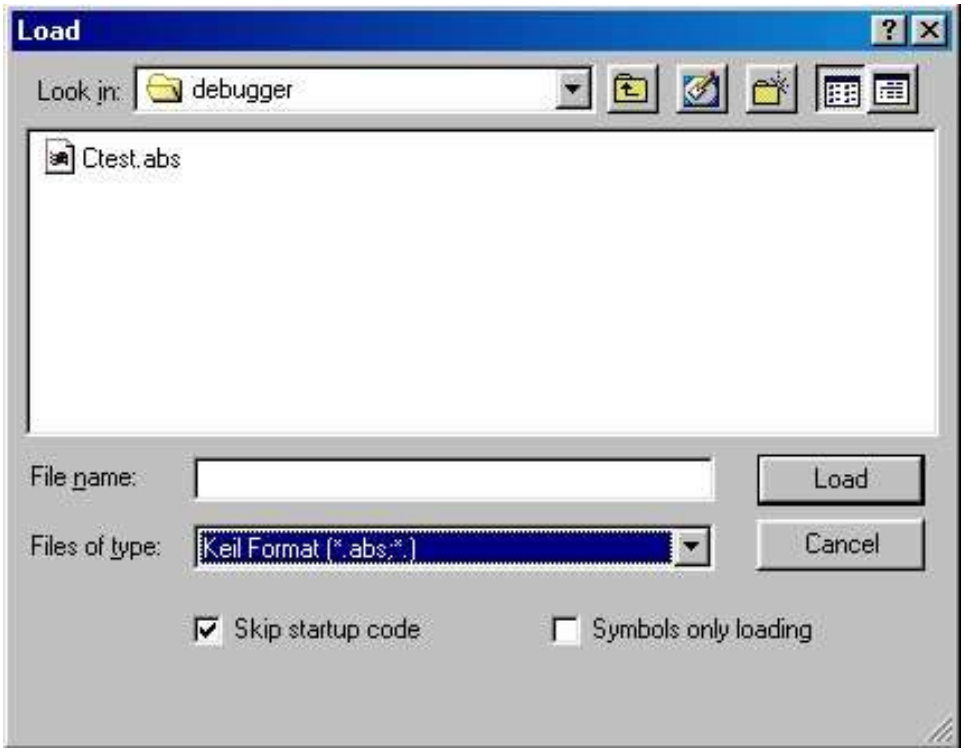


FIGURE 4.7: ***File Open Dialog***

FIGURE 4.8: ***Files of Type***

Close

The Close command closes the loaded file.

Unload

This command deletes all the symbol information loaded by the Load Command, thus clearing symbols and code.

Print

This command prints the loaded file. You may preview the file and also define the print setup.

Exit

The Exit command terminates the debugging session. The hot key Alt-F4 can also be used to leave the debugger. The Windows Alt-Tab key sequence may also be used to leave temporarily the session without closing it.

4.6. Edit Menu

The Edit menu commands are standard for editing, finding and replacing text y the loaded file. The available commands are Cut, Copy, Paste, Find, Find in Files and Replace.

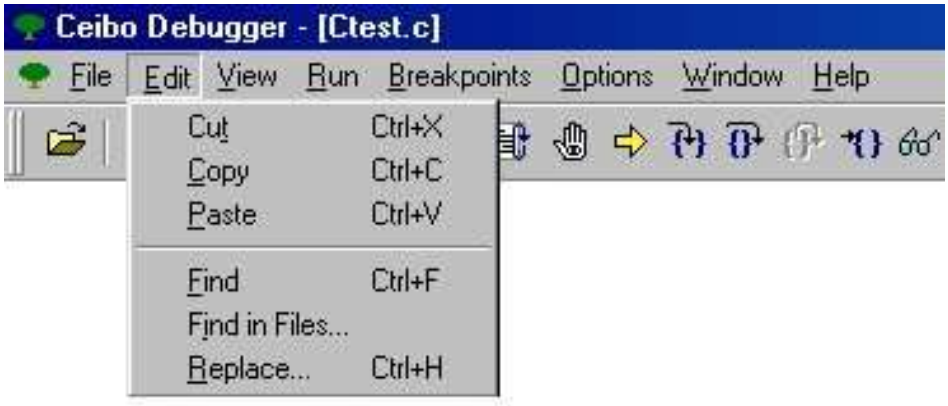


FIGURE 4.9: ***Edit Menu***

From the Find Dialog you may specify what to find and the direction in the text, as well as other text attributes.

FIGURE 4.10: ***Find Dialog***

You may also specify in which files to find a string from the Find in Files Dialog.

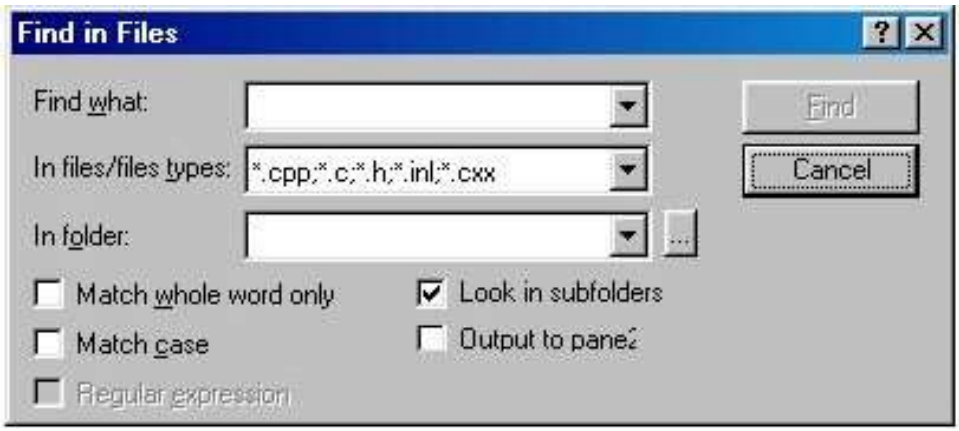


FIGURE 4.11: ***Find in Files Dialog***

4.7. View Menu

The View menu commands open windows that display different aspects of the program being debugged.

You may open as many windows as you need. The following figure shows an example.

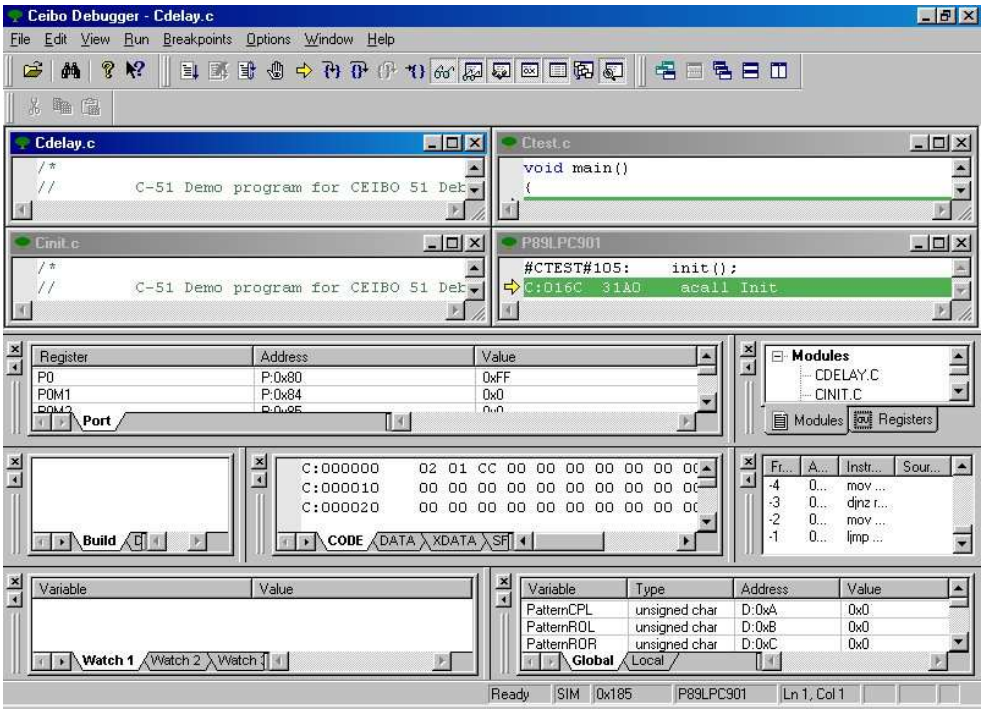


FIGURE 4.12: ***Opening Multiple Windows***

The available commands are: Debug Space, Status Bar, Output, Toolbars, Breakpoints, Memory Windows, Debug Windows, Target and Trace.

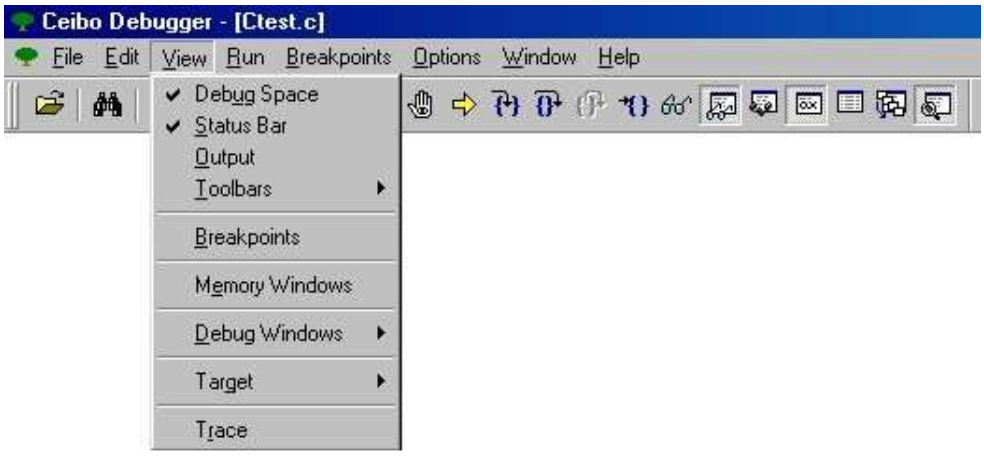


FIGURE 4.13: ***View Menu***

Debug Space

From this window you may select the module of the loaded project to set breakpoints, watch variables, source level debugging and more. By using the local menu you may find a file or open all the modules. You may also select one module from the list just by clicking on its name.

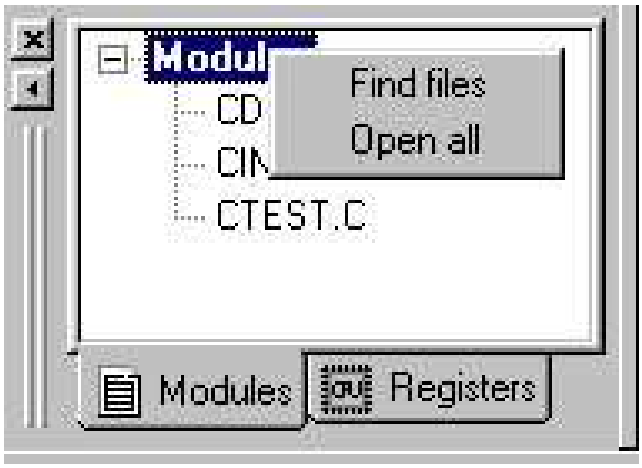


FIGURE 4.14: ***Debug Space - Modules***

From the Module window you can use the local menu add a variable to the Watch window, define the program counter (New pc) or go to the certain line. The Origin command refreshes the screen to the current position of the program counter. Set directly a breakpoint by moving the cursor to the desired line and pressing F2. Furthermore, you can drag a variable to the Watch window.

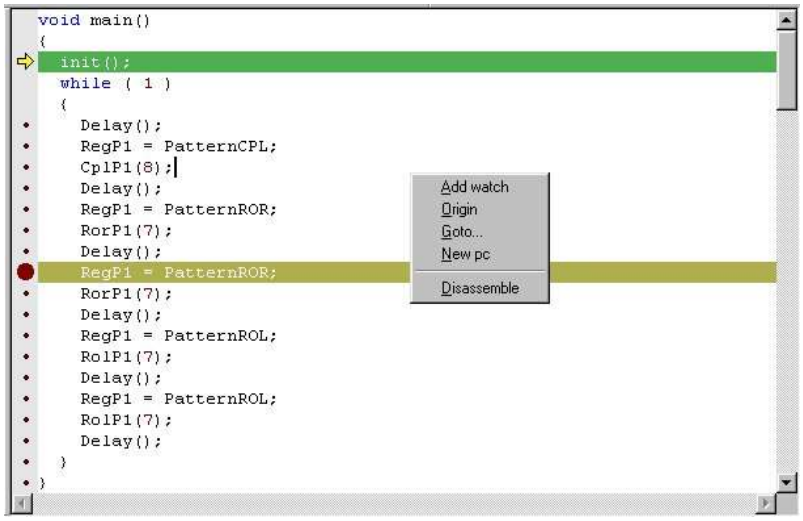


FIGURE 4.15: ***Debug Space - Modules***

The Registers page shows the CPU main registers and the local menu can be used to modify them.

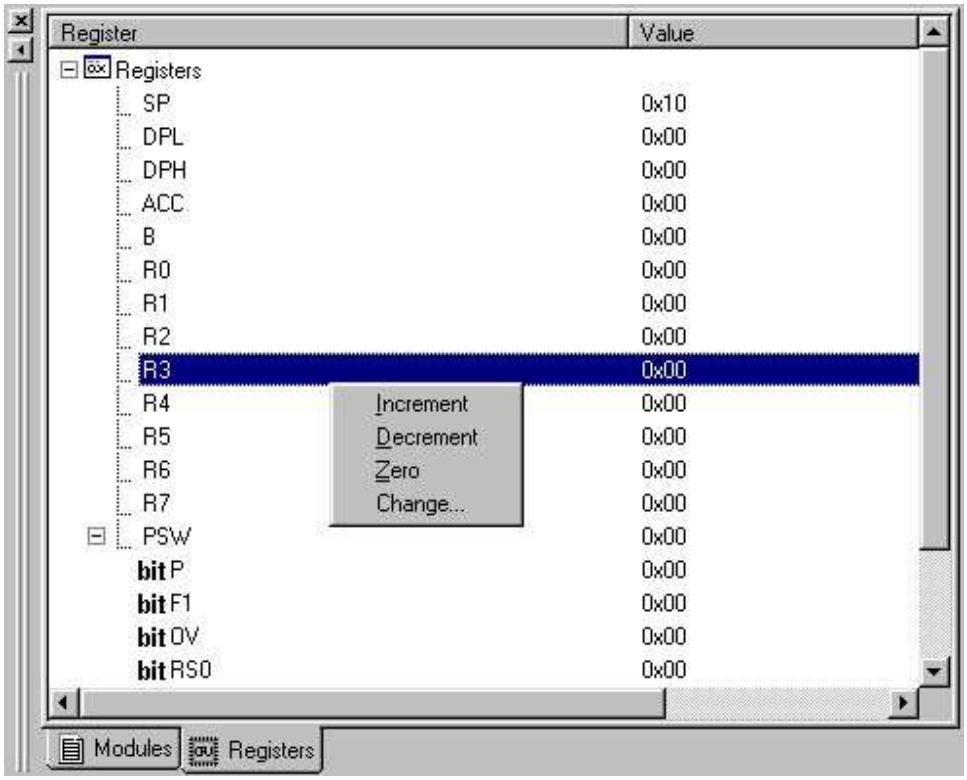


FIGURE 4.16: ***Debug Space - Registers***

Status Bar

This command is used to turn on or off the status bar. It is used to arrange windows according to your preferences.

Output

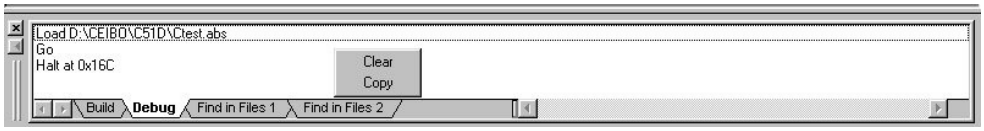


FIGURE 4.17: ***View Output***

Use this window to log and display debug activities. This is useful to review the sequence of the debug session.

Toolbars

This command is used to turn on or off the main, debug, edit and windows bar. It is used to arrange windows according to your preferences.

Breakpoints

The Breakpoints menu commands let breakpoints be displayed, set and cleared. The different options may be accessed through the Local menu of this window and they are used to add, delete, enable or disable breakpoints.

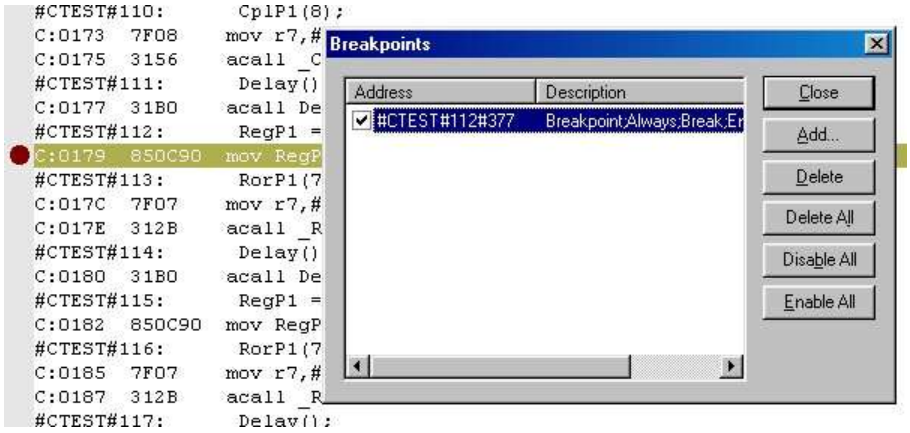


FIGURE 4.18: ***Breakpoints Local Menu***

Memory Windows

The Memory windows show the specified areas of memory. Data can be viewed as raw hex bytes with their corresponding ASCII representation. You may open a page showing internal data memory (Data), code memory (Code) and external data accessed by MOVX instructions (Xdata). Click on a value to modify it.

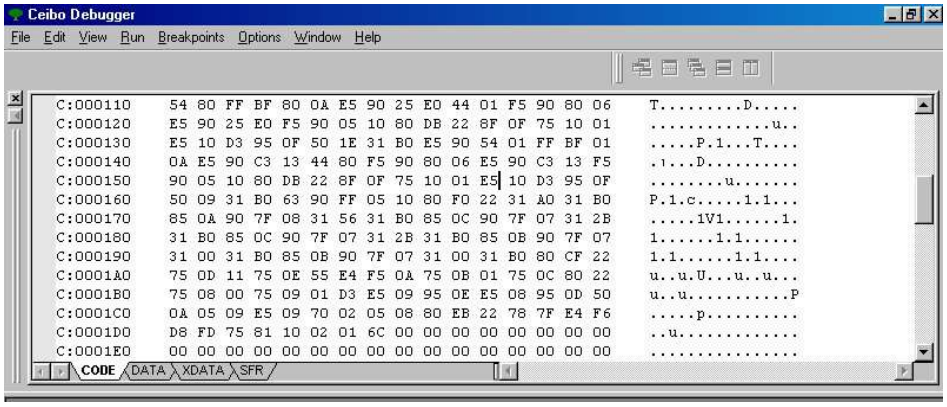


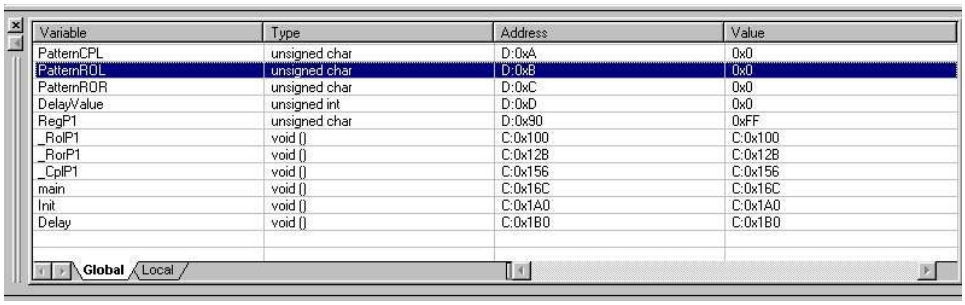
FIGURE 4.19: ***Memory Windows***

Debug Windows

The Debug windows show information related the debug session: variables in your code, watch windows with your defined variables to be displayed, Stack values and CPU window with the assembly code.

Variables

The Variables command opens a Variables window displaying a list of the program global (or public) and local symbols, and their locations.



Variable	Type	Address	Value
PatternCPL	unsigned char	D:0xA	0x0
PatternROL	unsigned char	D:0xB	0x0
PatternROR	unsigned char	D:0xC	0x0
DelayValue	unsigned int	D:0xD	0x0
RegP1	unsigned char	D:0x90	0xFF
_RolP1	void ()	C:0x100	C:0x100
_RorP1	void ()	C:0x12B	C:0x12B
_CplP1	void ()	C:0x156	C:0x156
main	void ()	C:0x16C	C:0x16C
Init	void ()	C:0x1A0	C:0x1A0
Delay	void ()	C:0x1B0	C:0x1B0

Global Local

FIGURE 4.20: ***Variables Window***

Click on a variable to change the value.

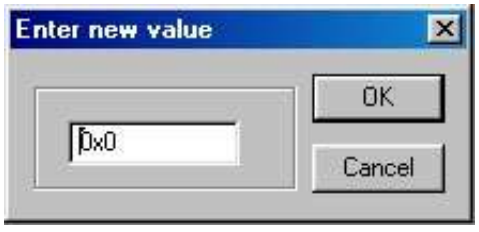


FIGURE 4.21: ***Changing Variable Values***

Watches

The Watches command shows the value of specified variables. You may drag a variable from the module window or type in the name of the variable.

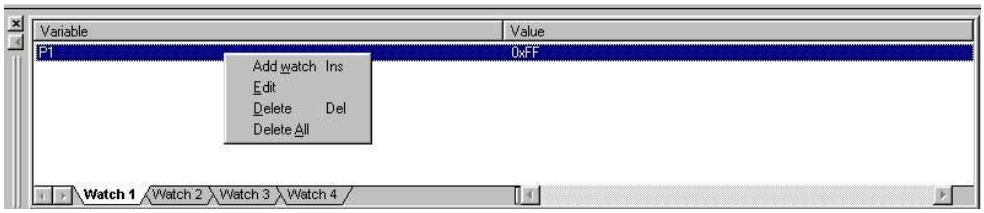


FIGURE 4.22: ***Variable Local Menu***

These Local menu is used to add a watch, edit and delete.

You may enter any predefined symbol like P1, P1.0, R3, etc. or make an absolute reference using the first letter to define the variable type followed by a colon and the address. Furthermore, absolute references may be expressed with a length.

The syntax is:

absolute_reference:address,length

Absolute references are D for the on-chip RAM, X for XRAM, P for ports and any SFR (special function registers), C for code memory and B for bit memory. Some examples are:

D:100,5

B:0X80,5

X:0X1000,8

P:0X431

Addresses are entered using the syntax of the selected language in the Options menu, Environment, Interface. Length is always decimal.

Your entries use the syntax of the selected language in the Options menu. For example, if you select C and you want to enter 9Fh, just type 0x9F. The 9FH entry is recognized if the selected language is ASM. In case that you are using Pascal, enter $9F.

A string may be changed by entering values separated by commas or blank spaces.

Stack

The Stack window shows the stack bytes with the associated addresses relative to the stack pointer. For example, the address -2 means stack pointer contents minus 2.

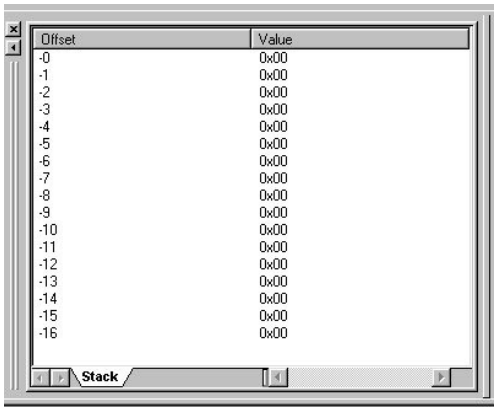


FIGURE 4.23: ***Stack***

CPU

The CPU command opens a CPU window displaying the disassembled instructions of your program. An instruction may be displayed with symbol information, and mixed with source code lines. You may also patch-up code using the built-in assembler.

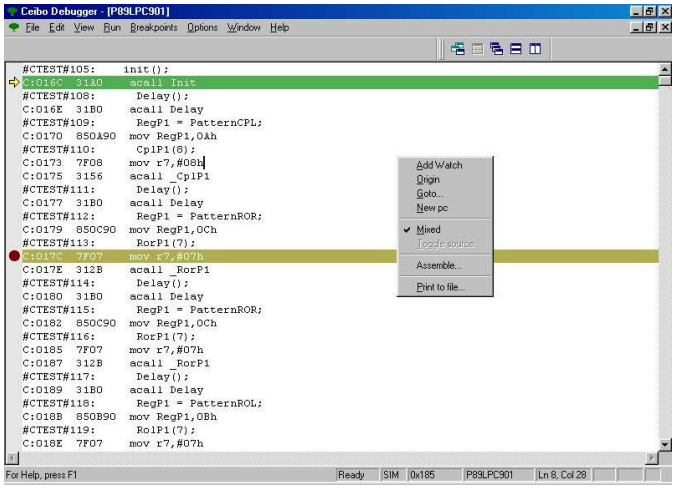


FIGURE 4.24: ***CPU***

The Local menu enables the following commands: ***Add Watch***, ***Origin*** that refreshes the screen to the current position of the program counter, ***Go to*** a specified address, ***New PC*** setting the program counter to the current position of the cursor, ***Mixed*** to display lines with source information, ***Toggle Source*** to toggle between disassembled code and code embedded with source line numbers, ***Assemble*** to on-line modify your code and finally, ***Print to File*** saving any portion of your code in ASCII format to a disk file.

Target

This command opens different windows that are related to the target microcontroller emulated or simulated by the debugger.

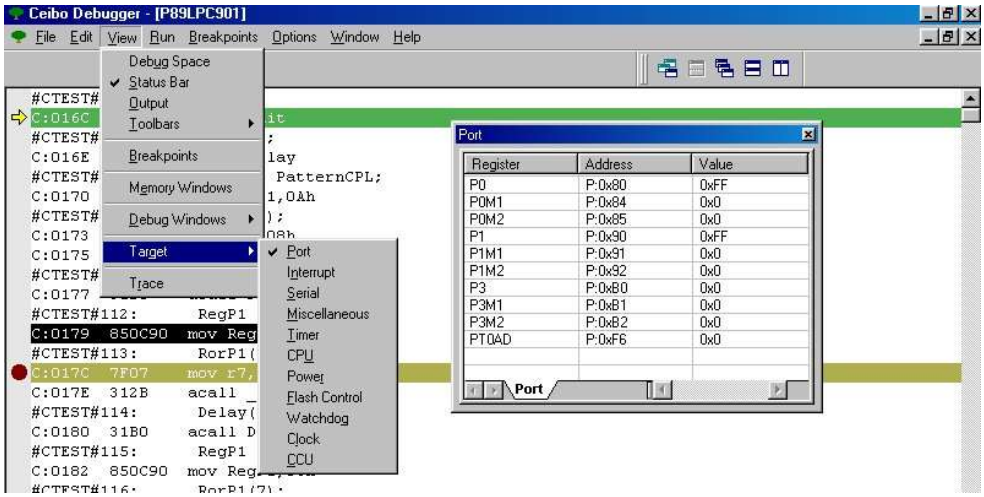


FIGURE 4.25: ***Target Menu***

The available windows that may be opened depend on the microcontroller and they may be Port, Interrupt, Serial and many others.

From any of there windows you may click on a variable to change its value.

Trace

The Trace menu allows the current Trace window to be opened, as well as viewing the trace status. Trace functions are enabled in both emulation and simulation modes.

The Trace window allows the current trace buffer to be viewed, display different formats for the trace selected, filter data from the trace display and search data patterns in the buffer. *Go to s*ets the cursor to the specified frame number.

*Origin d*isplays the window starting from the first recorded frame. *Inspect* shows additional information about the variables recorded in the trace buffer.

Display Mode selects source code, disassembled instruction or mixed source and disassembled code.

Time Stamps is used to display the absolute cycles (accumulated number of cycles), absolute time (accumulated time according to the XTAL selection in the Options menu) and relative cycles (number of cycles of each frame).

Clear Trace deletes all the accumulated data. *Print to File* saves the trace buffer in a disk file.

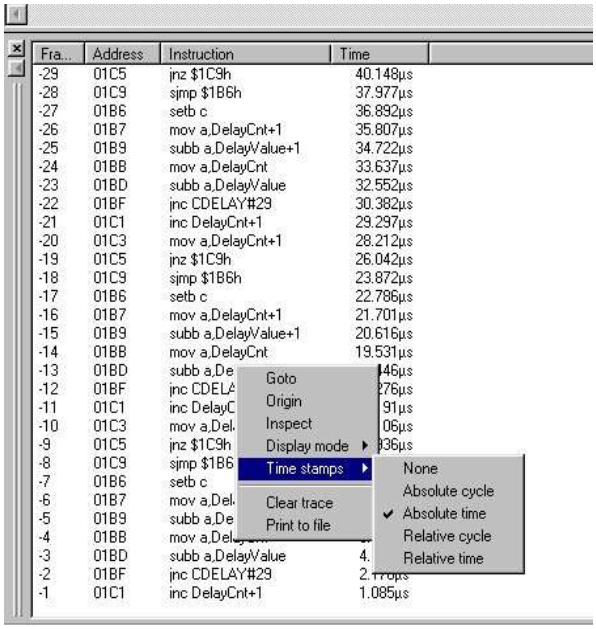


FIGURE 4.26: *Trace Menu*

4.8. Run Menu

The Run menu commands execute the program being debugged. The following options are available: Run, Execute Forever, Go to Cursor, Trace Into, Execute to, Step Over, Animate, Instruction Trace, Continuous Run, Halt and Program Reset.

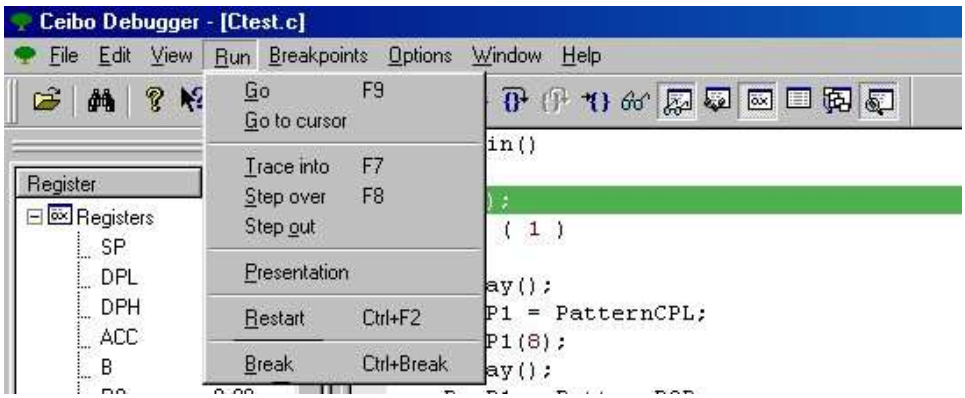


FIGURE 4.27: ***Run Menu***

Go

The Go command executes the program continuously until either the program is halted with the Halt key, or a breakpoint is reached. The F9 key is the hot key that executes this command.

Go to Cursor

The Go to Cursor command executes the program until the instruction or source line pointed by the cursor is reached. The current window must be a CPU window or a Module window in order to determine which location to execute.

Trace Into

The Trace Into command executes a high-level language source line or a single machine instruction. If your code module does not include debug information, the Trace into command executes a single machine instruction that may be observed in the CPU window. In case you have a code module with debug information, this command executes a complete line step. The F7 key is the hot key that executes this command.

Step Over

The Step over command executes a high-level language source line or a single machine instruction. If your code module does not include debug information, the Step over command executes a single machine instruction that may be observed in the CPU window. The only exception occurs when your code has a CALL instruction; then the program execution continues until it returns to the line following the CALL instruction. If the program does not return to the next line it will keep running. The F8 key is the hot key that executes this command.

Step Out

The Step out command executes a high-level language source lines until the end of a function.

Presentation

This command steps automatically though the program.

Restart

The Restart command issues a hardware reset to the emulated Microcontroller, causing all registers and on chip peripherals to return to their reset state. If you loaded a program with the Startup Skip option, the program will execute the startup code as well. Ctrl-F2 is the hot key for this command.

Break

The Break command stops the currently running program. Ctrl-Break is the hot key for this command.

4.9. Breakpoints Menu

The Breakpoints menu commands let breakpoints be set and cleared.

The following commands are available: Toggle and Delete All.

FIGURE 4.28: ***Breakpoints Menu***

Toggle

The Toggle command sets or clears a breakpoint at whatever address the cursor is pointing to in the CPU window or in the Module window. The program will stop each time it reaches a line where a breakpoint has been set, or a global or hardware breakpoint occurs. The F2 key is the hot key that executes this command.

At..

The At command executes the program and stops the program at a specific location. The address can be entered in any of the valid address formats.

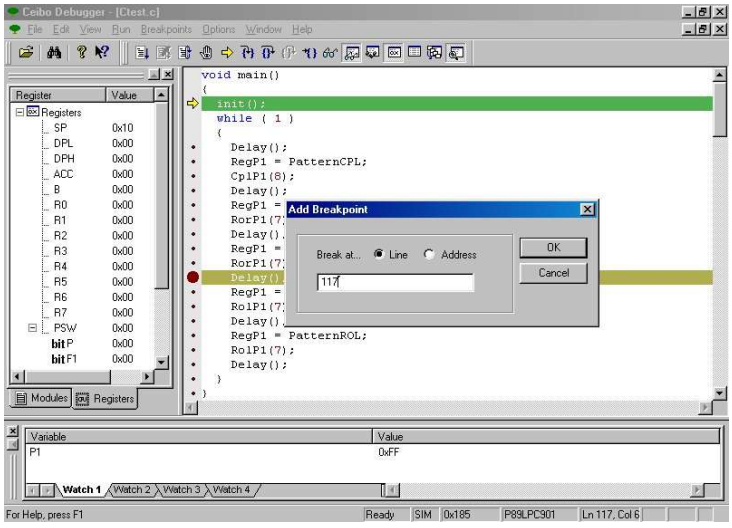


FIGURE 4.29: ***Breakpoint At..***

Delete All

The Delete All command deletes all the breakpoints from the program. This includes software, hardware, or expression true global breakpoints. This command is used when debugging is to be continued without stopping the program at any previously set breakpoint location.

4.10. Options Menu

The Options menu allows adjustment of some options that have a global effect on the conduct of the Windows Debugger, and the remote emulator system.

The following are the available options: Environment, Module List File, Mode, Communication, Chip, Architecture, Debug Controls, Save Options and Restore Default.

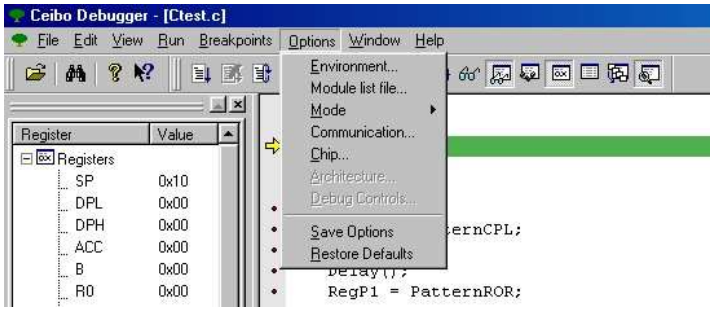


FIGURE 4.30: ***Options Menu***

Environment

The Environment option allows you to control the general environment parameters of Windows Debugger. The following options are available: Path, Format and Interface.

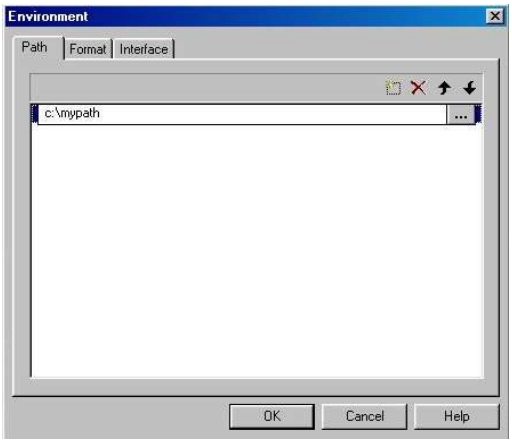


FIGURE 4.31: ***Options Menu - Environment - Path***

The Path for Source List option defines the directory trees in which the debugger will search for the source list files of your program.

The Format option is used to set your preferred font and color for each window.

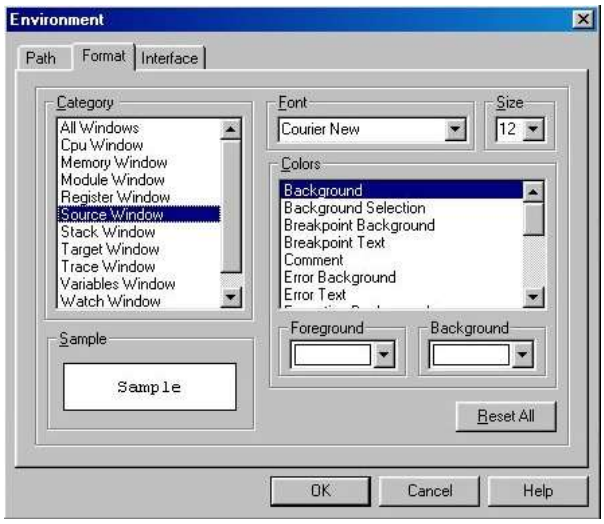


FIGURE 4.32: ***Options Menu - Environment Format***

The Interface option selects the source language for your value entries and several display options.

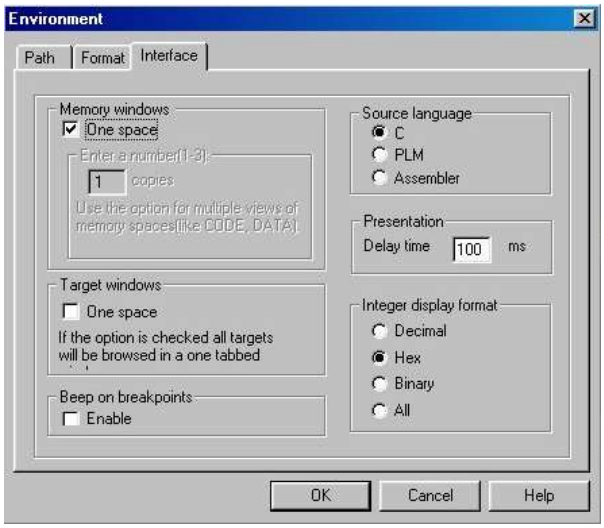


FIGURE 4.33: ***Options Menu - Environment Interface***

The Language command determines the base and syntax of your entries. For example, if you choose C Language, 55 is a decimal value and 0x55 is a hexadecimal number. In case the Assembler is selected, you should type 55h to enter the same hexadecimal value.

The Integer Display Format command defines the base of the display in the Watches Window. You can select hexadecimal, decimal or both bases for displaying your variables.

Module List File

The Module List File option is used to set the list file name associated with each module of the program being debugged.

The Module command will only allow access to modules with valid list files found in your disk.

Use the File Find button to redefine the list file names and paths. First click on a module name to select it. Then, use the File Find button to open the Module List Files Dialog Box.

Whenever loading an ASM program for the first time, the default setting will be the module_name.LST. It is therefore recommended to keep the module name equal to the list file name, as it will prevent you from having to configure this setting. Otherwise you will need to assign the list file name for each module after loading a new program to debug for the first time.

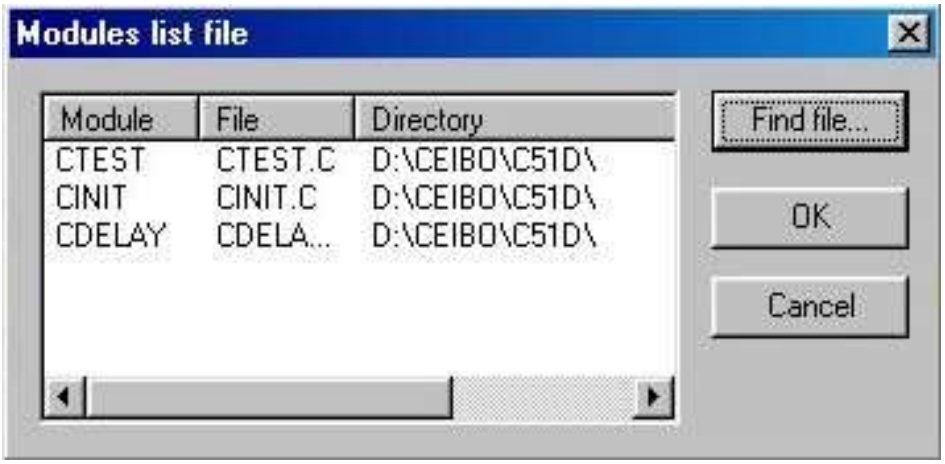


FIGURE 4.34*: Module List Dialog Box*

Mode

The Mode option allows you to select the operation mode of Windows Debugger. The following options are available: Emulation and Simulation.

Emulation: The Emulation Mode option sets the Debugger to operate in full emulation mode. In this mode your program will be executed in real time on the remote emulator system. This mode requires the use of a Ceibo Emulator connected to your PC. Communication error will result if the emulator is not found on the selected COM port.

Simulation: The Simulation Mode option sets the Windows Debugger to operate in full simulation mode. In this mode, your program instruction execution will be simulated by the debugger built-in simulator. This mode can be operated without connecting any remote emulator system, thus allowing software debugging to be done while the emulator is used for hardware debugging, or other projects. This

is ***not a real-time mode***; only basic functions are supported and not all SFRs belonging to particular derivatives. ***Set the system to emulation mode***, while using the debugger with a Ceibo Emulator.

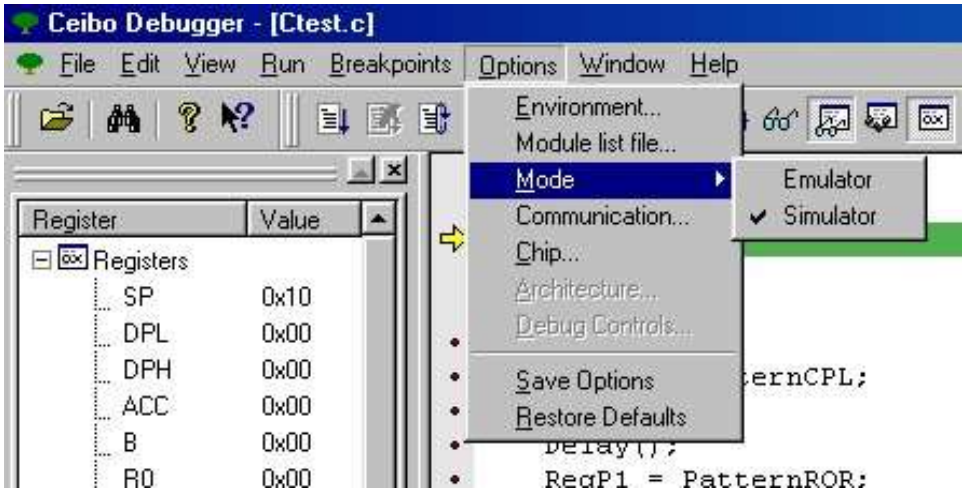


FIGURE 4.35: ***Options Menu - Mode***

Communication

The Communication option allows selection of the host PC COM port number to be used for communication with the remote emulator system.

Make sure that there are no resident programs hooked up to the selected COM port, when being used for remote emulation interface.

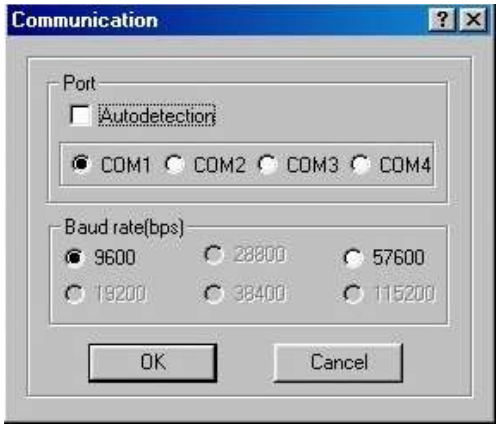


FIGURE 4.36: ***Options Menu - Communication***

Chip

The Chip option is used to tell the debugger which microcontroller is being used by the emulator or simulator. It defines SFRs and other parameters accordingly.

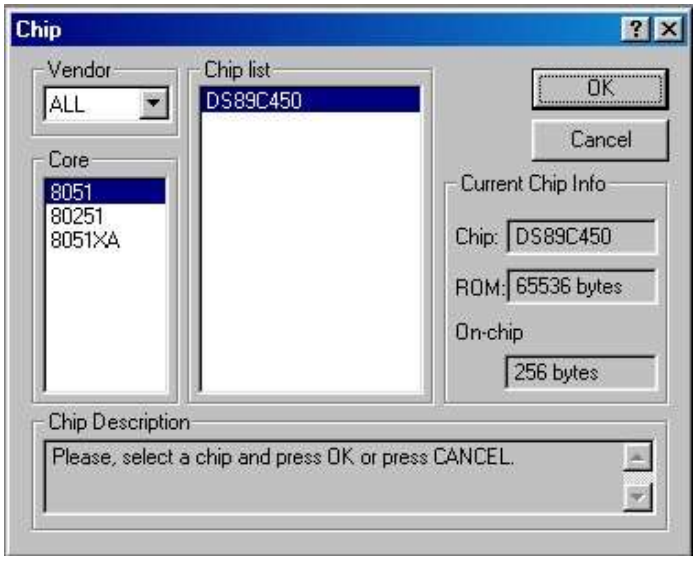


FIGURE 4.37: ***Options Menu - Chip***

Architecture

This command defines specific emulator options in case which are required by the hardware configuration, as the same debugger supports many microcontroller derivatives with different hardware emulators. The available options are Data Map, Xtal and Halt Mechanism.

Data Map specifies the memory accessed by MOVX instructions. It can belong to the on-chip memory or to the target board.

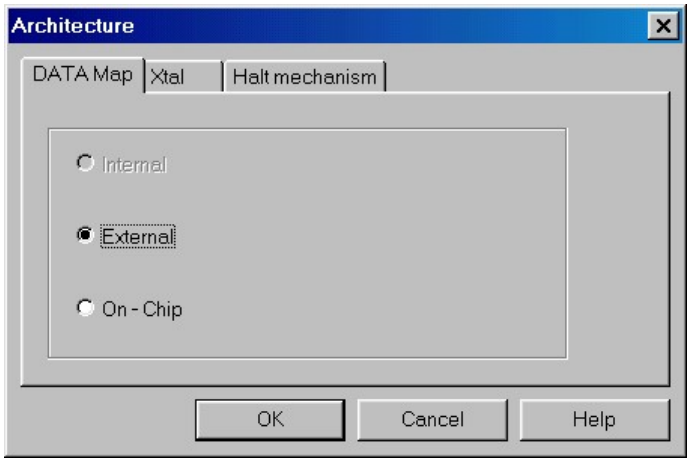


FIGURE 4.38: ***Architecture - Data Map***

Xtal sets the clock frequency applied to the microcontroller. A programmable clock generator is implemented in the emulator to support this feature.

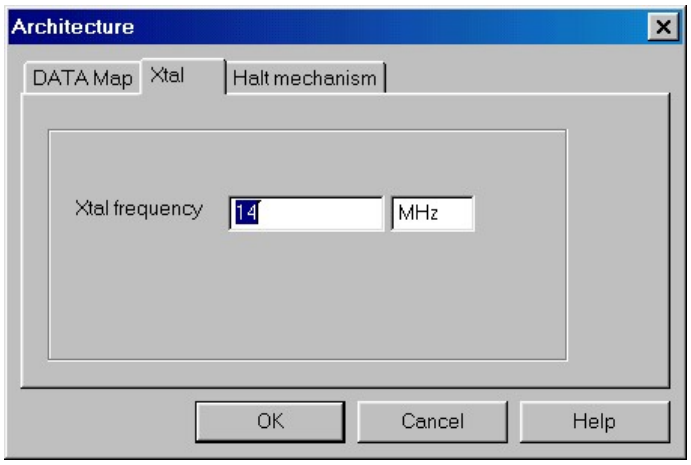


FIGURE 4.39: ***Architecture - XTal***

Halt Mechanism sets the options to stop the emulation while it cannot be done by a breakpoint. More details about this important setup are given in the Emulator User Manual.

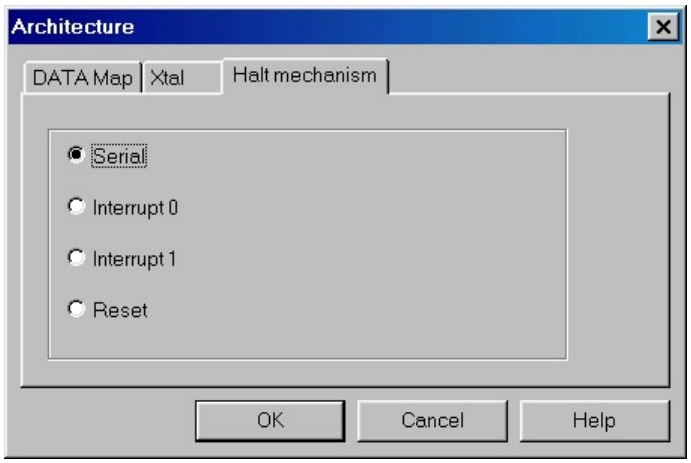


FIGURE 4.40*: Architecture - Halt Mechanism*

Debug Controls

Debug Controls dialog is used to define the following:

***Reload Setting*:** you may define whether the last loaded code will be automatically reloaded or not. Automatic reloading is possible while powering up the system or when a program reset is executed.

***Origin Enable*:** this option specifies where the cursor will point to while stopping the emulation. If the box is checked, that enables the origin and the cursor will point to the actual program counter value in the CPU and Module windows. If it is not checked, those windows will remain in the same state as they have been while running the program.

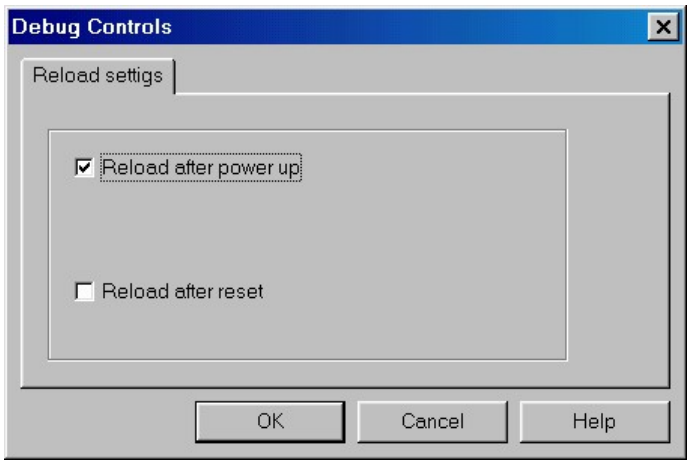


FIGURE 4.41*: Debug Controls*

Save Options

Select this command if you want to save the setup while leaving the debugger. This setup will be restored while invoking again the debugger. It saves window layouts at any time and with any filename.

Restore Defaults

The Restore Defaults command allows you to load a configuration file from disk. The configuration file should have been previously saved by using the Save Options command.

4.11. Windows Menu

The Window menu allows various operations on the currently open windows with the following commands: New, Tile, Cascade and Arrange Icons.

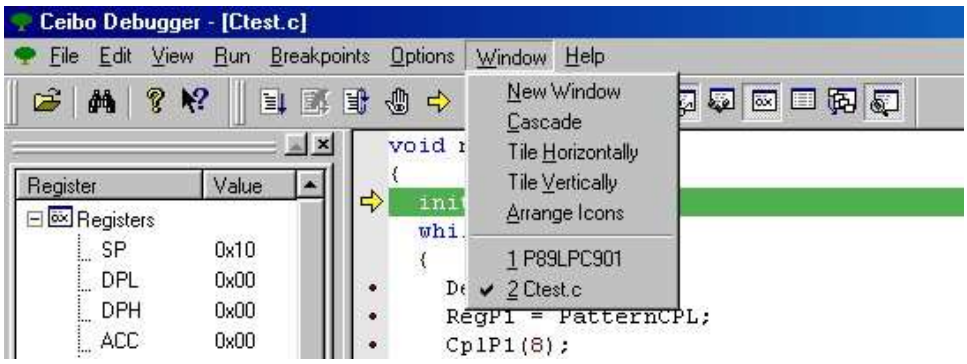


FIGURE 4.42: *Windows Menu*

4.12. Help Menu

The Help menu commands open a help window for whichever subject will be selected from the menu. This menu offers the following options: Help Topics and About.

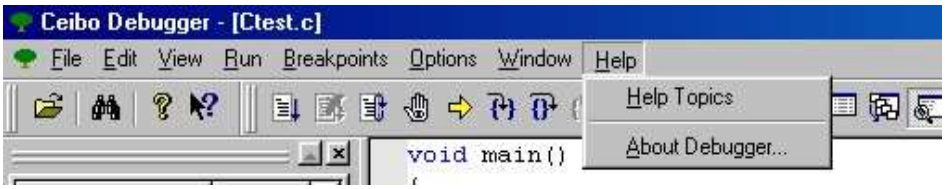


FIGURE 4.43: ***Help Menu***

Not all the help contexts are listed, only the useful subjects and starting points. Shift-F1 is the hot key that executes this command.

The About command displays the debugger software version.

CHAPTER 5

Traditional Ceibo Windows Debugger

CHAPTER 5

Traditional Ceibo Windows Debugger

5.1. Introduction

The previous chapter gave you a general approach to the Windows Debugger menus and commands. You will find a detailed description of the menus and their related commands in the following paragraphs.

5.2. File Menu

The File menu options deal with operations external to the Windows Debugger, such as loading programs for debugging, and leaving the application.

The available commands are: Load, Get Info, New and Exit.

This menu also provides a list of the last opened files, so you may load a file just by clicking on the desired file name.

Load

The Load command loads a program for debugging from a disk.

You may select the directory and the file name to be loaded, as well as the format of the file to achieve complete debug information compatibility. The supported file formats are: Intel Hex, Intel ASM51 and PLM51 OMF51, Keil/Franklin extended OMF, IAR/Archimedes UBROF, BSO/Tasking OMF51, Metalink ASM51, MCC Software and general OMF51. Call Ceibo for additional format support.

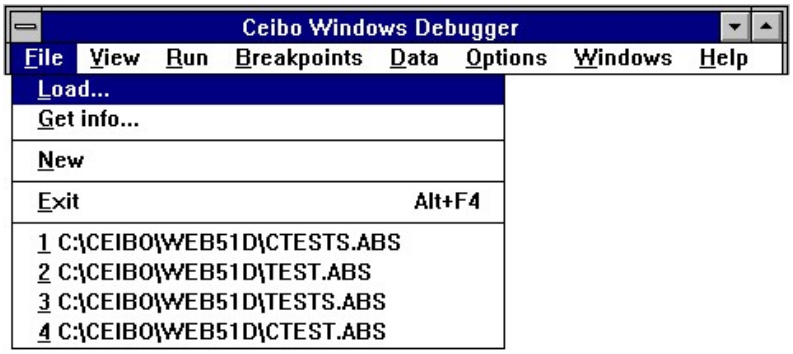


FIGURE 5.1: ***File Menu***

From the File Menu you can specify the Symbol Only option, thus loading only the symbol information in the file for debugging ROM applications.

The Startup Skip option is used to run the program automatically until the first line of your main program is reached.

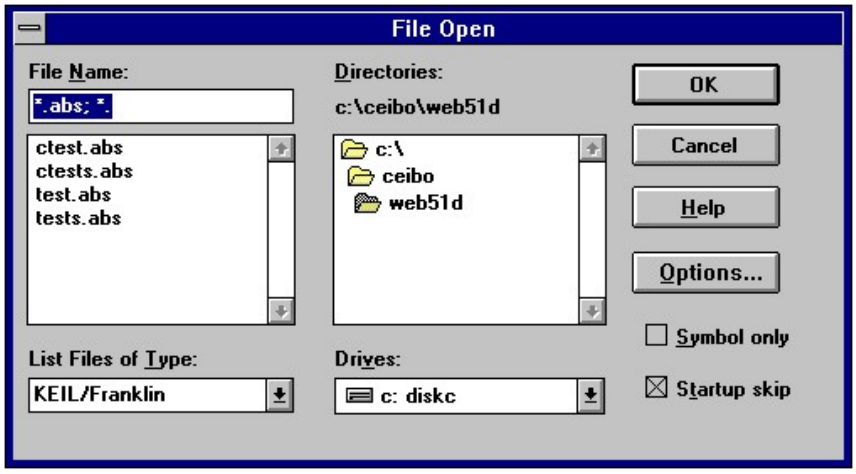


FIGURE 5.2: ***File Open Dialog***

The Options button accesses the Set Options Dialog box, which lists the list of files that may be loaded automatically for special hardware support. This subject is explained in Chapter 7.

Get Info

The Get Info command displays a window showing the current state of your computer resources software and system versions.

New

This command deletes all the symbol information loaded by the Load Command, thus clearing symbols and code.

Exit

The Exit command terminates the debugging session. The hot key Alt-F4 can also be used to leave the debugger. The Windows Alt-Tab key sequence may also be used to leave temporarily the session without closing it.

5.3. View Menu

The View menu commands open windows that display different aspects of the program being debugged. The available commands are: Breakpoints, Variables, Module, Watch, CPU, Registers, Performance Analyzer, Trace, Memory Space and Target.

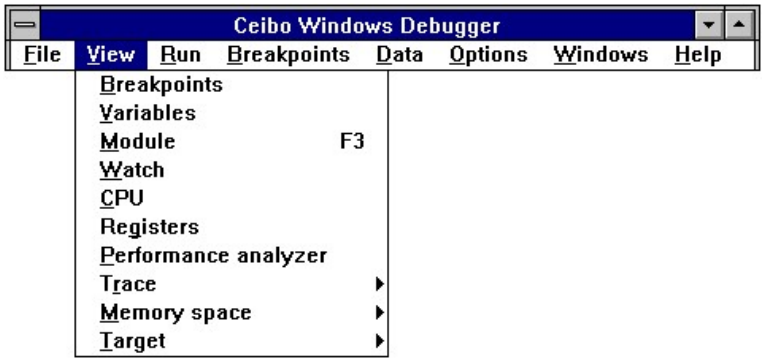


FIGURE 5.3: ***View Menu***

Breakpoints

The Breakpoints menu commands let breakpoints be displayed, set and cleared. The different options may be accessed through the Local menu of this window; type Alt-F10 or click the right button.

The different breakpoint options available from the Local menu are:

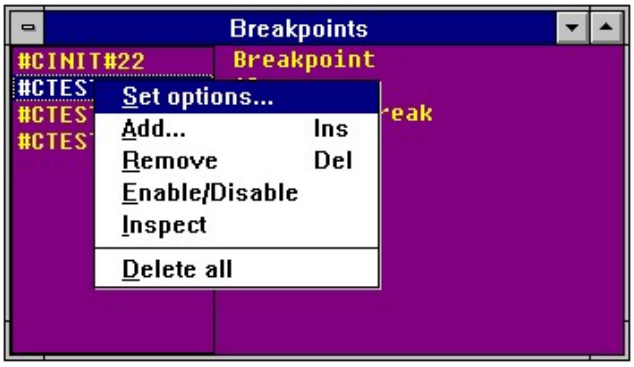


FIGURE 5.4: ***Breakpoints Local Menu***

Set Options: This command is used to redefine the type of bus cycle, address match condition, passcount and address of the selected breakpoint. Click first one of the breakpoints on the left side of the window and then select the Set Options command. Use this command to define the cycle type, address match condition, address range and passcount.

Cycle: Use this option to specify the bus cycle or execution state of a breakpoint.

Address match: Selects the address qualification mode.

Address: Address ranges may be defined by selecting range and length and entering the corresponding addresses separated by a comma or a blank space. The format for an address range is: address, length.

Passcount: From the Set Options Dialog Box you may define the passcount value, thus selecting the number of occurrences before the program actually stops.

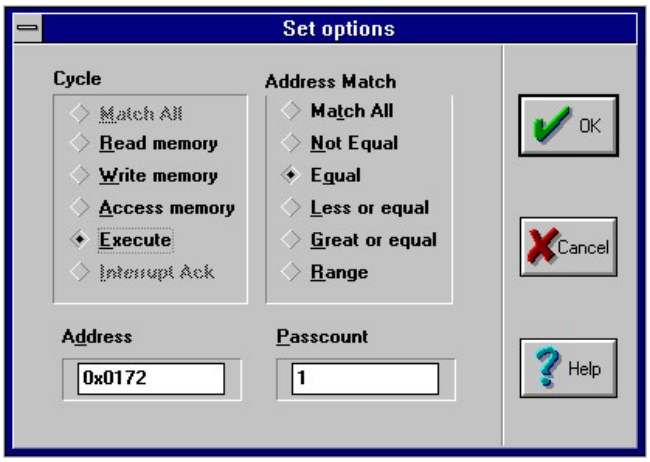


FIGURE 5.5: ***Set Options***

Add: This command is used to set a new breakpoint. The operation is similar to the above Set Options, with the difference that you do not need to select an existing breakpoint from the window.

Remove: Use this command to cancel a selected breakpoint.

Enable/Disable: Sets the status of the selected breakpoint without removing it or affecting its definition.

Inspect: Displays the information regarding the code location, such as module name and CPU address.

Delete All: Removes all the defined breakpoints.

Variables

The Variables command opens a Variables window displaying a list of the program global (or public) and local symbols, and their locations. The window is split into two sections. The upper area shows the global symbols while the lower one displays the local symbols.

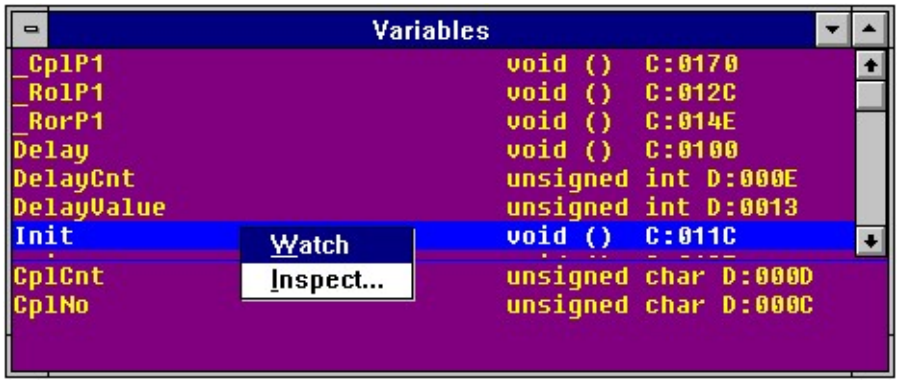


FIGURE 5.6: ***Variables Window***

The Local menu of the Variables window has two useful commands:

Watch**:** Adds a variable to the Watches window. Click first the desired variable to highlight it and then you may include it to the Watches Window by using the Local Menu.

Inspect**:** Displays all the available information about the highlighted variable.

Module

The Module command opens a Module window showing the list file of the selected module.

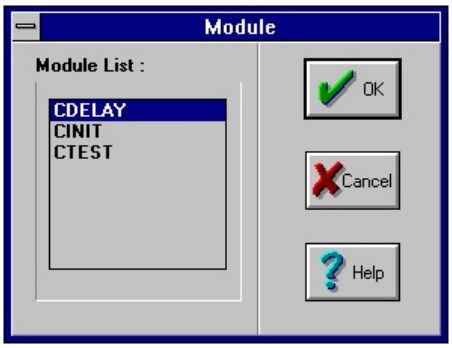


FIGURE 5.7: ***Module List***

A module to be viewed can be picked from a list of the current program available modules. Only modules which have a corresponding list file will appear in this pick list.

This command will be disabled if no debug information has been loaded.

F3 or the button in the Speed Bar are the hot keys for this command.

The Module window title indicates the module name currently being viewed.

List and source files may be in different directories and the software should know about it. The Options menu gives the possibility to define the path for the files and filenames.

From the Module window you may open the Local menu with the following options:

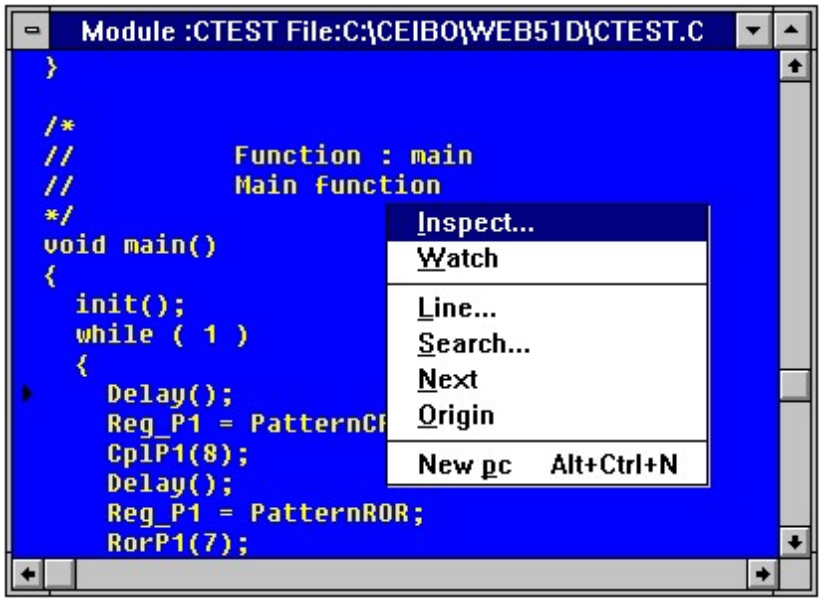


FIGURE 5.8: ***Module Local Menu***

Inspect: This command provides all the available information on the selected variable.

Watch: Use this command to add the variable pointed by the cursor to the Watches window.

Line: Specifies the line number of the source file to be displayed in the Module window.

Search: The Search command may be used to locate any string in the Module window.

Next: This command searches the next location of the string specified by the Search command.

Origin: The Origin command refreshes the screen to the current position of the program counter.

New PC: This command sets the program counter to the current position of the cursor. Stack operations and variable values may be affected by this redefinition of the program counter.

Watches

The Watches command opens a Watches window showing the value of variables specified from the Data menu. Different Local menus may also be used to enter new variables. These Local menus are available in the Watches, Module and Variables windows.

FIGURE 5.9: ***Watches Local Menu***

The Local menu of the Watches window has the following options:

Watch: Use this command to add a new variable to the window. You may enter any predefined symbol like P1, P1.0, R3, ACC, etc. or make an absolute reference using the first letter to define the variable type followed by a colon and the address. Furthermore, absolute references may be expressed with a length.

The syntax is:

absolute_reference:address,length

Absolute references are D for the on-chip RAM, X for external RAM accessed by MOVX instructions, P for ports and any SFR (special function registers), C for code memory and B for bit memory (below 80H is for RAM bits and above that value represents SFR bits). Some examples are:

D:100,5

X:MYSYMBOL,2

B:0X80,5

B:P1.0,8

P:0XFE

Addresses are entered using the syntax of the selected language in the Options menu. Length is always decimal.

Edit: This command is used to modify the selected watch name.

Remove: The Remove command deletes the selected variable from the Watches window.

FIGURE 5.10: ***Watches Window***

Delete All: This command clears the Watches window.

Inspect: The Inspect command may be selected to get the address information of the selected variable.

Change: The Change command permits the modification of variable contents. Your entries use the syntax of the selected language in the Options menu. For example, if you select C and you want to enter 9Fh, just type 0x9F. The 9FH entry is recognized if the selected language is ASM or PLM. In case that you are using Pascal, enter $9F. A string may be changed by entering values separated by commas or blank spaces.

CPU

The CPU command opens a CPU window displaying the disassembled instructions of your program, the Stack, the internal Registers and any memory space according to the Local menu selection.

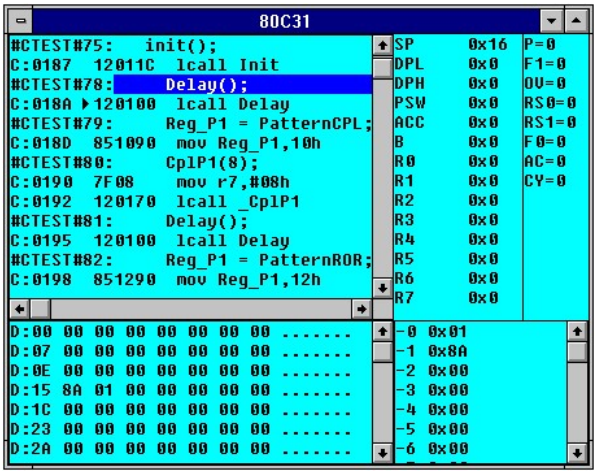


FIGURE 5.11: ***CPU Window***

An instruction may be displayed with symbol information, and mixed with source code lines.

You may also patch-up code using the built-in assembler.

The CPU window is divided into five sections: disassembled code, registers, status bits, stack and memory. Each of them has its own Local menu.

From the disassemble section the Local menu enables the following commands:

Go to: This command is used to set the cursor to any desired address.

Origin: The Origin command refreshes the screen to the current position of the program counter.

Toggle Source: This command toggles between pure disassembled code and code embedded with source line numbers.

Assemble: The Assemble command is used to on-line modify your code. The syntax of this command is fully explained in the On-Line Assembler chapter.

New PC: This command sets the program counter to the current position of the cursor. Stack operations and variable values may be affected by this redefinition of the program counter.

Print to File: Use this command to save any portion of your code in ASCII format to a disk file.

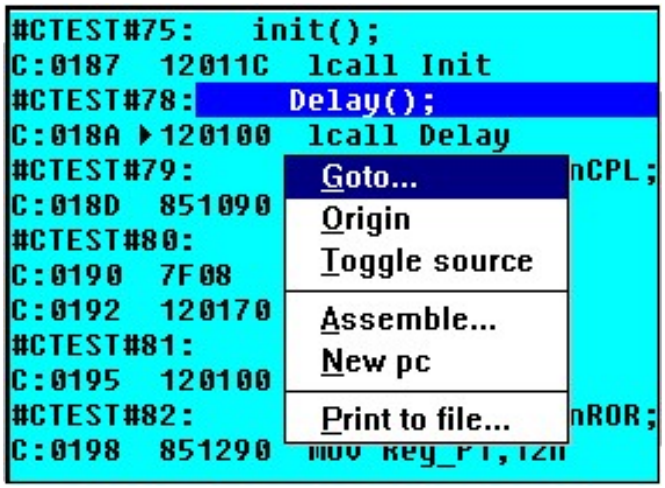


FIGURE 5.12*: Disassembly Local Menu*

The registers and status bits sections in the CPU window are similar to the Register window. The Local menu is explained in the description of the Registers window.

The stack section of the CPU window shows the stack bytes with the associated addresses relative to the stack pointer. For example, the address -2 means stack pointer contents minus 2. The Local menu of this section has the following commands:

Go to: This command is used to set the cursor to any stack position.

Origin: The Origin command refreshes the screen to the current position of the program counter.

Change: You can use this command to modify the stack contents.

FIGURE 5.13: *Stack Local Menu*

The memory section of the CPU window has the same Local menu as the Memory Space windows in the View menu, with the addition of the capability to change the type of memory displayed in the CPU window: external data, on-chip RAM

or code. The explanation of the Local menu commands is given in the Memory Space windows description.

Registers

The Registers command opens a Registers window where the flags and current CPU registers state appear.

The Registers are displayed according to the selected microcontroller in the Options menu, Architecture and Chip commands.

The Local menu of this window has the following commands:

Toggle: This command is available for the register bits and clears or sets the selected bit state.

Increment: Adds one to the current value of the selected register byte.

Decrement: Subtracts one from the current value of the selected register byte.

Zero: Clears the selected register byte.

Read: The Read command forces the reading of the specified register. This is a useful command in case that the register is affected by the reading operation.

Change: Use this command to modify the selected register.

Update: Reads all the registers to update the window.

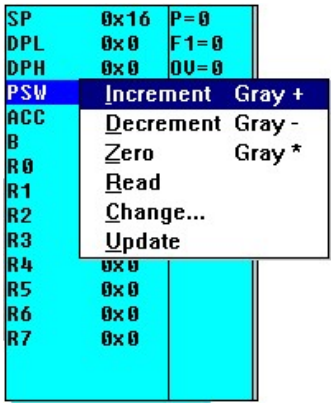


FIGURE 5.14: ***Registers Window***

Performance Analyzer

This command processes the information recorded in the trace buffer and provides a graphics representation of the executed modules and the percentage of time spent in each of them.

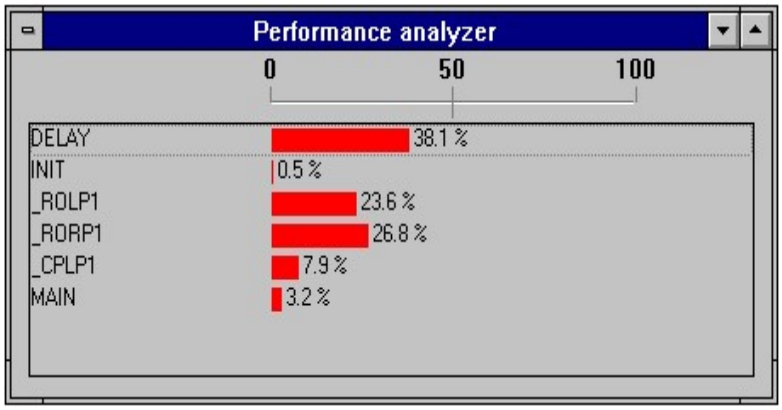


FIGURE 5.15: ***Performance Analyzer***

The local menu of this window may be used to get more useful information about your software performance.

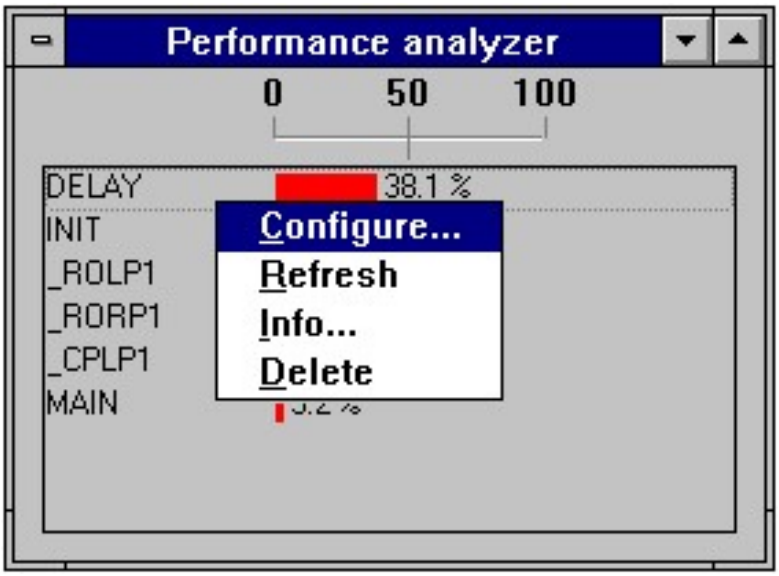


FIGURE 5.16: ***Performance Analyzer - Local Menu***

Configure: A dialog box allows the selection of the functions belonging to the module that you may want to include in the performance analysis. Just select the desired module and then the respective functions. Furthermore, if you desire to define your own address range, click the [*user defined...*] line and the Add button. Then, you will be able to select a customized address range to analyze.

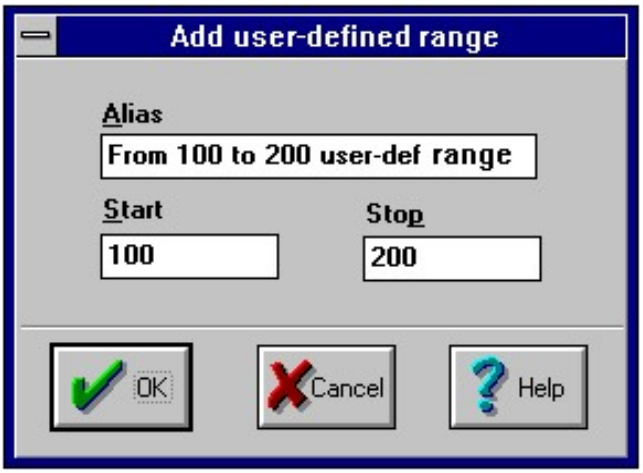


FIGURE 5.17: ***User Defined Dialog Box***

Refresh: This command reads again the trace buffer and recalculates the performance of the modules.

Info: Displays information about the module selected by the cursor in the window.

Trace

The Trace menu allows the current Trace window to be opened, as well as viewing the trace status. The Trace functions are enabled in simulation modes only.

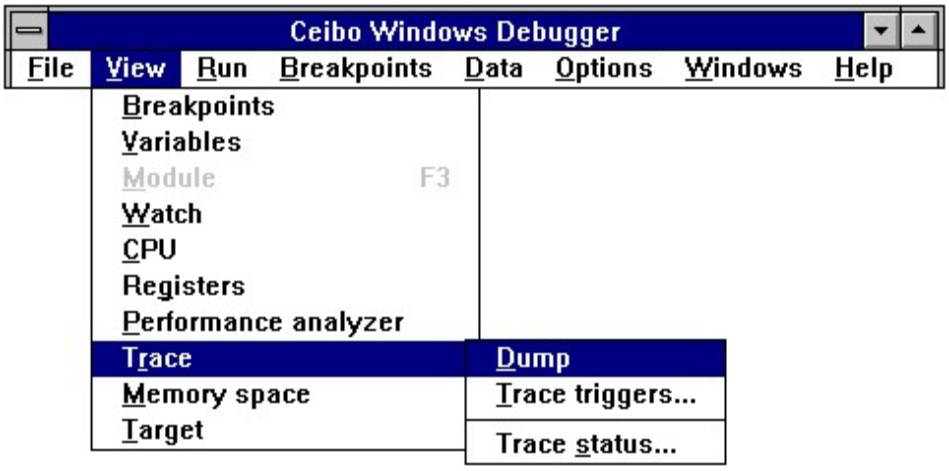


FIGURE 5.18: ***Trace Menu***

The following options are available: Trace Dump, Trace Triggers and Trace Status.

Trace Dump: The Trace Buffer Dump command opens a Trace window allowing the current trace buffer to be viewed, different display formats for the trace selected, data from the trace display filtered and data patterns in the buffer searched (see Figure 5.19).

The Local menu of the Trace Dump windows provides many useful functions to setup the operation of the trace and manipulate the accumulated information.

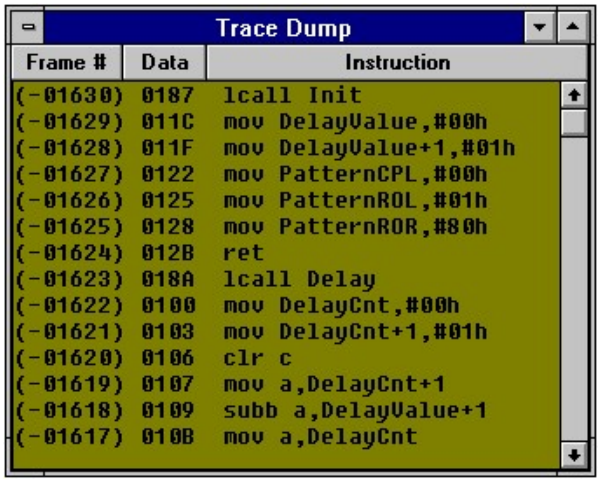


FIGURE 5.19: ***Trace Dump***

These functions are:

Go to: Sets the cursor to the specified frame number.

Origin: Displays the window starting from the first recorded frame.

Display Mode: A selection of source code, disassembled instruction or mixed source and disassembled code is available.

Time Stamps: You can display the absolute cycles (accumulated number of cycles), absolute time (accumulated time according to the XTAL selection in the Options menu) and relative cycles (number of cycles of each frame).

Filters: Defines the trace filters of the displayed data. You may specify which instructions or sequences are of your interest.

Trace Triggers: This command is the same as available from the Trace Menu and it is explained separately.

Inspect: You may display additional information about the variables recorded in the trace buffer.

Clear Trace: Deletes all the accumulated data.

Trace Status: This command is the same as available from the Trace Menu and it is explained separately.

Read all Trace: Gets all the recorded data from the trace buffer for performance analysis, absolute time stamp calculations, etc.

Print to File: Saves the trace buffer in a disk file.

Trace Triggers: This command sets the trace control to the selected option to start and stop the recording.

The different buttons and functions are:

Run begins: When execution is started, the trace buffer will capture data from the start program execution until the trace buffer is filled. This mode automatically selects the *Stop when Full* option.

Halt ends: Trace buffer capture is enabled and data is continuously collected until the break condition occurs. In this mode, the trace buffer will be filled with bus cycles immediately preceding the break condition.

From event: Trace capture begins when the Start event trigger condition is satisfied and continues until the program stops due to a breakpoint or Halt command. In this mode, the trigger event will be found in the beginning of the trace buffer contents. After clicking this button you must select the Start event button to complete the trigger definition.

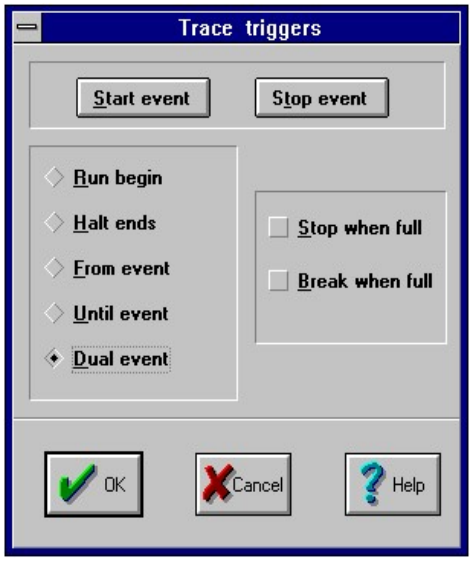


FIGURE 5.20: ***Trace Triggers***

Until event: Trace capture is enabled and the trace buffer filled until the Stop event trigger condition is satisfied. In this mode, the trigger event will be found in the end of the trace buffer. After clicking this button you must select the Stop event button to complete the trigger definition.

Dual event: This option combines both From and Until events, therefore you have to complete the selection by defining Start and Stop events.

Start event: Select the Start event button to define the trigger conditions for trace modes using a start event. This button will be dimmed if the selected trace mode does not require a start event.

Stop event: Select the Stop event button to define the trigger conditions for trace modes using a stop event. This button will be dimmed if the selected trace mode does not require a stop event.

After selecting the Start or Stop event buttons, the Set Trace Trigger dialog box will be displayed.

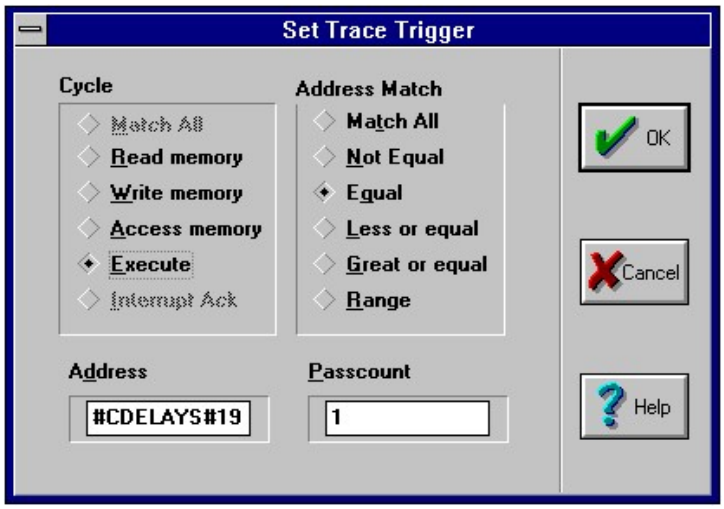


FIGURE 5.21: ***Set Trace Trigger Dialog Box***

A trace trigger is the means for selecting the criteria for capturing an execution trace in the target system trace buffer. As we will see, setting a trace event and a hardware breakpoint are in fact the same operation. The only difference between the two operations is that action for a tracepoint is to turn the trace buffer on or off while breakpoints are used to implement the various breakpoint actions.

The Trace Trigger definition is used to define the condition used to start trace data collection. When this option is selected, a dialog box containing field for selecting the bus cycle type, address and data bus contents is displayed and you are prompted to fill in the conditions used to trigger the trace collection hardware in the target system.

When a trace trigger is defined, each field of the dialog box is combined to specify the event. Trace triggers operate on the recognition of user-defined events, using combinations of the following controls.

Cycle: Use this option to specify the bus cycle or execution state to be used in defining the trace trigger condition.

Address Match: Selects the address qualification mode. If any mode other than Match All is selected, the Address input box must be filled in with the address or address range to be recognized by the trace trigger. Addresses can be entered as symbols, modules and line numbers or as physical addresses.

Passcount: Selects the number of times the event will be recognized before the trace trigger occurs.

When all the condition have been specified, select OK to confirm the trace trigger setup.

Trace Status: The Trace Status command displays a window showing the current state of the trace. This includes trace state (recording/halted), trace overflow indication and number of frames currently in the trace buffer.

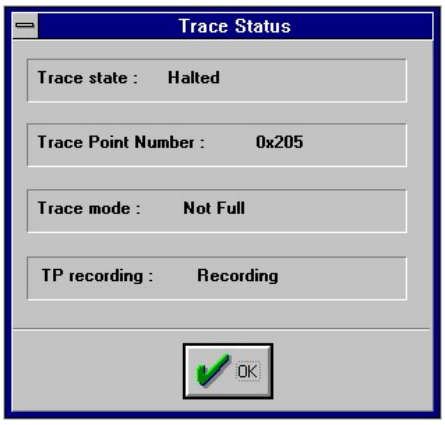


FIGURE 5.22: ***Trace Status***

Memory Space

The Memory Space command opens up to three windows where the specified areas of memory are displayed. The data can be viewed as raw hex bytes with their corresponding ASCII representation.

From this command you may open a window with internal data memory (Data), external data memory (XData) or the code memory (Code).

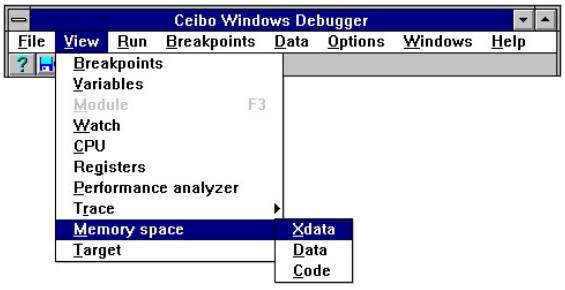


FIGURE 5.23: ***Memory Local Menu***

Any of the three memory spaces has a Local Menu with the following commands.

Go to: This command is used to set the cursor to any desired address.

Search: Use this command to find a data value in the window.

Next: Locates the cursor in the next finding of the data value specified by the Search command.

Change: This command modifies the contents of the selected data memory. These data bytes may be written sequentially with blank spaces or commas between them, if you need to specify a string, i.e. 11, 22, 33, 44, 55, where the base is specified according to the selected language in the Options menu.

Block: The Block command is very useful to manipulate the data. You can clear all the data, set it to any value, move portions of the memory space, read a file and put its contents into the memory and write any portion of the memory to a disk file.

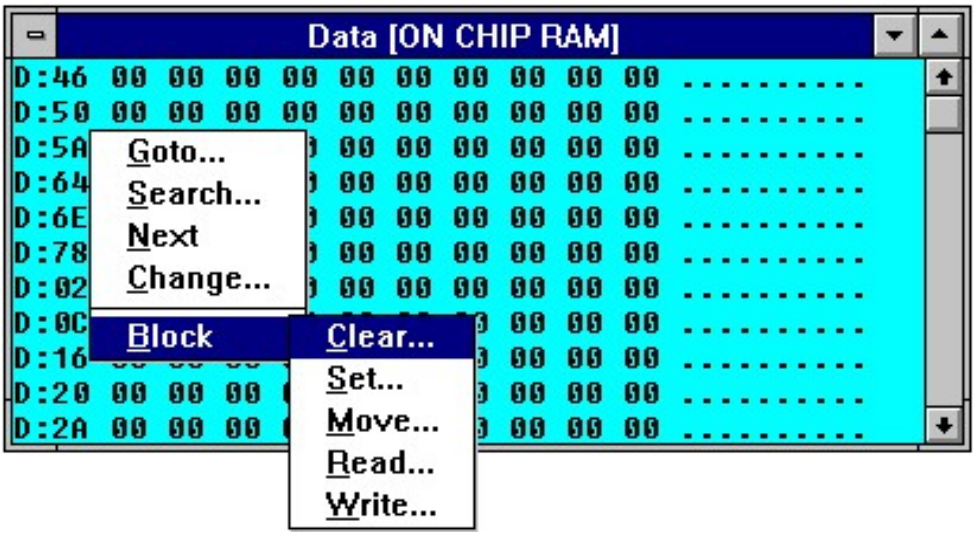


FIGURE 5.24: ***Block Menu***

Target

This command opens different windows that are related to the target microcontroller emulated or simulated by the debugger. The available windows that may be opened are Port, Interrupt, Serial, Miscellaneous, Timer, Power, PWM, Watchdog and others depending on the selected chip type.. From any of there windows you may use the Local menu to carry out the same functions described in the Registers window: Increment, Decrement, Zero, Read, Change and Update.

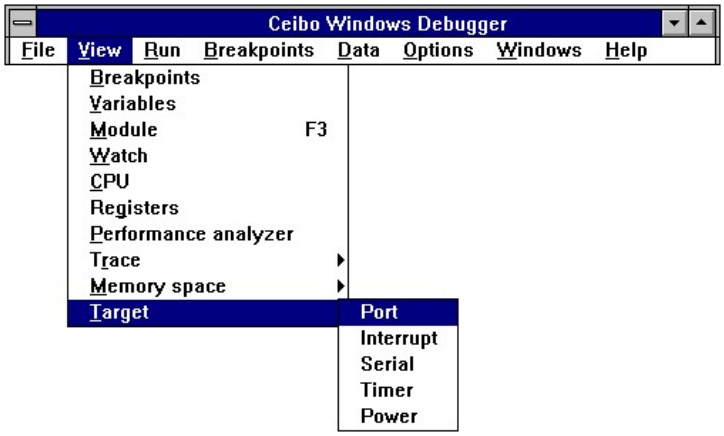


FIGURE 5.25: ***Target Menu***

5.4. Run Menu

The Run menu commands execute the program being debugged. The following options are available: Run, Execute Forever, Go to Cursor, Trace Into, Execute to, Step Over, Animate, Instruction Trace, Continuous Run, Halt and Program Reset.

Run

The Run command executes the program continuously until either the program is halted with the Halt key, or a breakpoint is reached.

The F9 key is the hot key that executes this command.

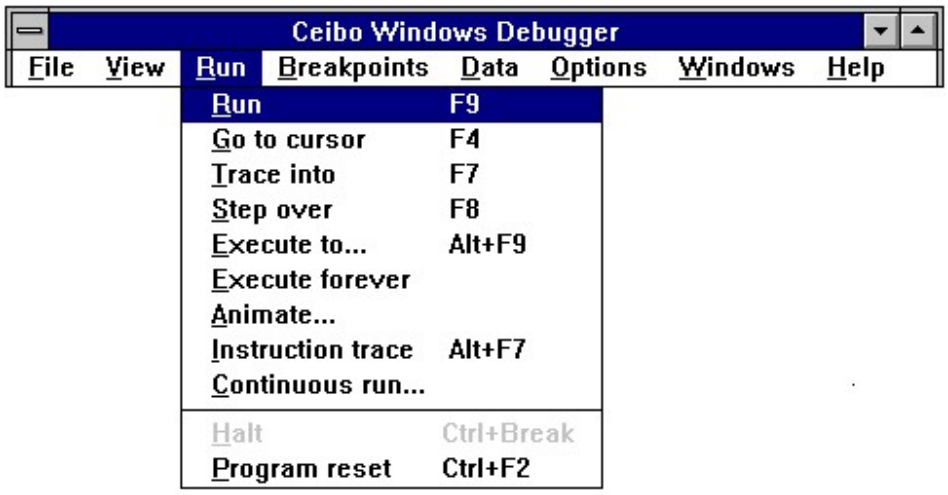


Figure 5.26: ***Run Menu***

Execute Forever

The Execute Forever command executes the program being debugged continuously until halted with the Halt key.

Any breakpoints set are temporarily deleted, until halting program execution.

This is useful for temporarily running the program without interruptions, and without having to delete all the previously set breakpoints.

Go to Cursor

The Go to Cursor command executes the program until the instruction or source line pointed by the cursor is reached.

The current window must be a CPU window or a Module window in order to determine which location to execute.

The F4 key is the hot key that executes this command.

Trace Into

The Trace Into command executes a high-level language source line or a single machine instruction.

If your code module does not include debug information, the Trace into command executes a single machine instruction that may be observed in the CPU window.

In case you have a code module with debug information, this command executes a complete line step.

The F7 key is the hot key that executes this command.

Execute To

The Execute To command executes the program and stops the program at a specific location. The address can be entered in any of the valid address formats.

Alt-F9 is the hot key that executes this command.

Step Over

The Step over command executes a high-level language source line or a single machine instruction.

If your code module does not include debug information, the Step over command executes a single machine instruction that may be observed in the CPU window. The only exception occurs when your code has a CALL instruction; then the program execution continues until it returns to the line following the CALL instruction. If the program does not return to the next line it will keep running.

The F8 key is the hot key that executes this command.

Animate

The Animate command is a self-repeating Trace into command.

The source lines or instructions are continuously executed until any key is pressed. CEIBO Debugger shows changes reflecting the current program state between each step; this allows you to watch the program control flow.

The Animate Speed dialog box prompts you for the rate at which animated steps will be executed.

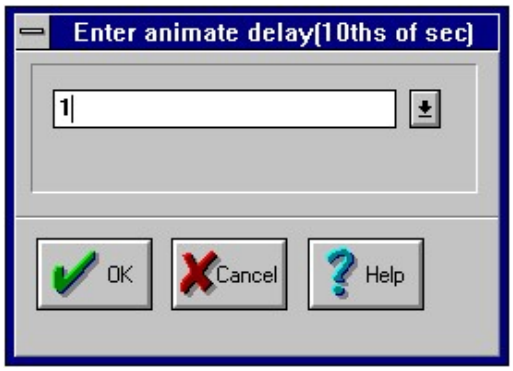


FIGURE 5.27: ***Animate Speed***

Instruction Trace

The Instruction Trace command executes a single machine instruction.

Use this command for the following: trace into a function in a module that was not compiled with debug information or watch execution of instructions which belong to a source line.

Alt-F7 is the hot key for this command.

Continuous Run

This command executes the program and breaks automatically to refresh all the windows according to the halt intervals defined in the dialog box.

Halt

The Halt command stops the currently running program.

This command will be disabled if there is no program currently running. Ctrl-Break is the hot key for this command.

Program Reset

The Program Reset command issues a hardware reset to the emulated Microcontroller, causing all registers and on chip peripherals to return to their reset state. If you loaded a program with the Startup Skip option, the program will execute the startup code as well.

Ctrl-F2 is the hot key for this command.

5.5. Breakpoints Menu

The Breakpoints menu commands let breakpoints be set and cleared.

The following commands are available: Toggle, Breakpoint at, Change Memory Global, Expression True Global and Delete All.

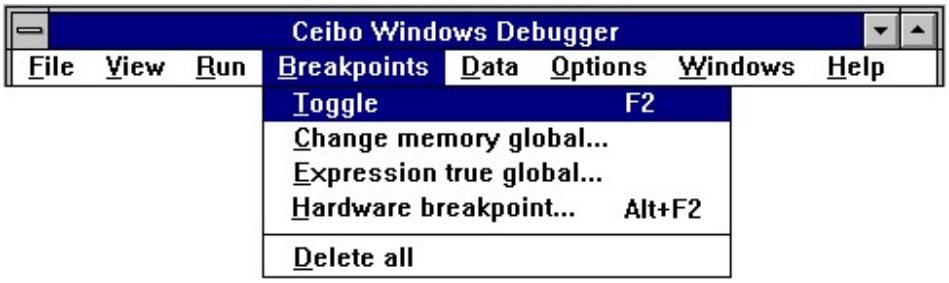


FIGURE 5.28: ***Breakpoints Menu***

Toggle

The Toggle command sets or clears a breakpoint at whatever address the cursor is pointing to in the CPU window or in the Module window.

The program will stop each time it reaches a line where a breakpoint has been set, or a global or hardware breakpoint occurs.

The F2 key is the hot key that executes this command.

Expression True Global

The Expression True Global command allows setting a breakpoint that will occur when the value of the specified memory space location matches the specified data value. This command is available in simulation modes only.

Change Memory Global

This command may be used to set a breakpoint on access to any specified memory space, regardless of the data contents. The Change Memory Global command is available in simulation modes only.

Hardware Breakpoint

This command opens a dialog box to set a breakpoint according to the cycle, address and number of occurrences. The Add Breakpoint Dialog box has been explained in the Set Options Dialog box of the View menu and Breakpoints Local menu.

Delete All

The Delete All command deletes all the breakpoints from the program. This include software, hardware, or expression true global breakpoints.

This command is used when debugging is to be continued without stopping the program at any previously set breakpoint location.

5.6. Data Menu

The Data menu commands permit the examination of variables and symbols in the program.

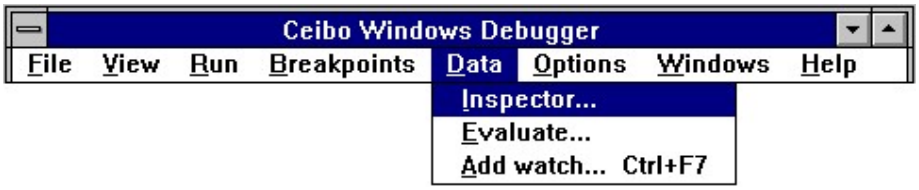


FIGURE 5.29: ***Data Menu***

The following commands are available: Inspector, Evaluate and Add Watch.

Inspector

The Inspect command prompts you for a symbol name to be inspected, if the specified symbol is found in the current program debug information, the relevant information on this symbol is displayed, this includes type, memory space reference and location.

You cannot change the inspected variable value from this command. Use the Watch Change command from the Watches window for modifying variable values.

Evaluate

The Evaluate command opens a dialog box which prompts you for an expression to evaluate. Then evaluates it according to the selected language in the Options menu.

Add watch

The Add Watch command prompts you for a variable name, or a memory space reference, and places it on the watch list displayed in the Watches window.

5.7. Options Menu

The Options menu allows adjustment of some options that have a global effect on the conduct of the Windows Debugger, and the remote emulator system.

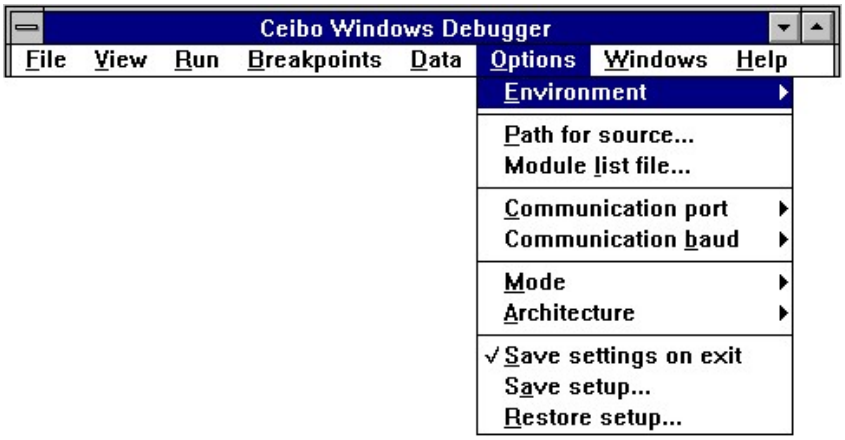


FIGURE 5.30: ***Options Menu***

The following are the available options: Environment, Path for Source, Module List File, Communication Port, Communication Baud, Mode, Architecture, Save Settings on Exit, Save Setup and Restore Setup.

Environment

The Environment option allows you to control the general environment parameters of Windows Debugger.

The following options are available: Language, Integer Display Format and Beep on Breakpoint

Language: The Language command determines the base and syntax of your entries. For example, if you choose C Language, 55 is a decimal value and 0x55 is a hexadecimal number. In case the Assembler is selected, you should type 55h to enter the same hexadecimal value.

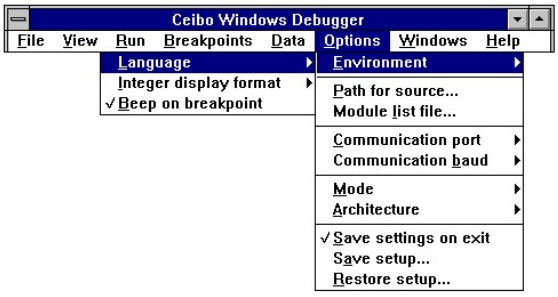


FIGURE 5.31: ***Environment Options***

Integer Display Format: The Integer Display Format command defines the base of the display in the Watches Window. You can select hexadecimal, decimal or both bases for displaying your variables.

Beep on Breakpoint: This option toggles On and Off the beep sound generation whenever a breakpoint is reached.

Path for Source

The Path for Source List option defines the directory trees in which the debugger will search for the source list files of your program.

The source list files are first searched for in the directories specified by this command, followed by the current directory and finally in the directory from which the program was loaded.

The syntax for this command is: *directory1;directory2;...,* thus allowing definition of several paths.

Module List File

The Module List File option is used to set the list file name associated with each module of the program being debugged.

The Module command will only allow access to modules with valid list files found in your disk.

Use the File Find button to redefine the list file names and paths. First click on a module name to select it. Then, use the File Find button to open the Module List Files Dialog Box.

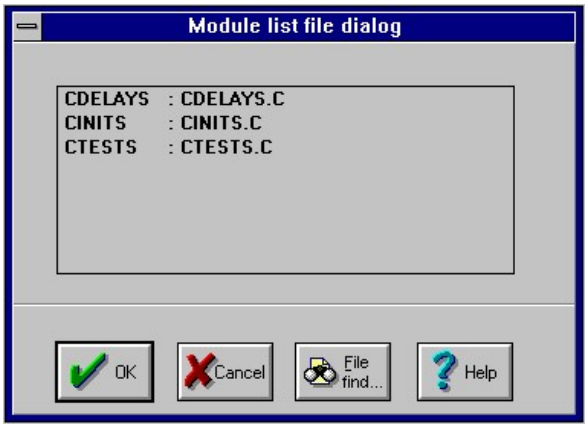


FIGURE 5.32: ***Module List Dialog Box***

Whenever loading a PLM or ASM program for the first time, the default setting will be the module_name.LST. It is therefore recommended to keep the module

name equal to the list file name, as it will prevent you from having to configure this setting. Otherwise you will need to assign the list file name for each module after loading a new program to debug for the first time.

Communication Port

The Communication port option allows selection of the host PC COM port number to be used for communication with the remote emulator system.

Make sure that there are no resident programs hooked up to the selected COM port, when being used for remote emulation interface.

Communication Baud

The Communication Baud option allows selection of the RS-232 interface baud rate to be used for communication with the remote emulator system.

You can toggle between High and Low options. Low baud rate is set to 9600 baud, and High baud rate is set to the maximum baud rate possible at time of selection, up to the maximum of 115K baud.

The baud rate synchronization is done in the software. There is no need to change any setting on the remote emulator system, whenever changing the baud rate.

Mode

The Mode option allows you to select the operation mode of Windows Debugger.

The following options are available: Emulation, In-Circuit Simulation and Simulation.

Emulation: The Emulation Mode option sets the Debugger to operate in full emulation mode. In this mode your program will be executed in real time on the remote emulator system.

This mode requires the use of Ceibo's real time emulator systems. Communication error will result if the emulator is not found on the selected COM port.

In-Circuit Simulation: The In-Circuit Simulation Mode option sets the Windows Debugger to operate in a special simulation mode.

In this mode your program instructions will be simulated by the debugger built-in simulator, but any SFR references will be transferred to the remote emulator system, thus causing ports and other SFRs to change while simulating.

This mode requires the use of EB-51 connected to your PC. Communication error will result if the emulator is not found on the selected COM port.

Simulation: The Simulation Mode option sets the Windows Debugger to operate in full simulation mode. In this mode your program instruction execution will be simulated by the debugger built-in simulator. This mode can be operated without connecting any remote emulator system, thus allowing software debugging to be done while the emulator is used for hardware debugging, or other projects.

Architecture

The Architecture option allows you to control and configure the remote emulator system options.

The available options are: Chip, Map Code, Map Data, Xtal, Halt Mechanism, Interrupt Shared and Reset.

Chip: This command selects the emulated microcontroller type.

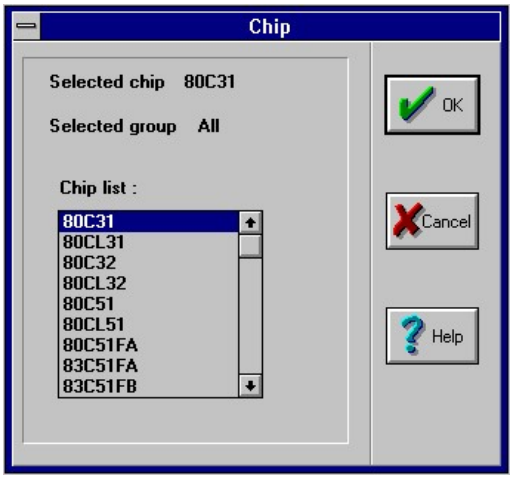


FIGURE 5.33: ***Chip Dialog Box***

Map Code: This command does not apply for EB-51, where code is always

mapped as belonging to the system and not to a target board.

Map Data: The Map Data command allows you to define the data memory as belonging to the emulator or to your target hardware. Data memory is accessed by MOVX instructions.

Xtal: Use this command to tell the software which crystal frequency you are using. With this value the system will be able to calculate time events as displayed in the Trace window.

Halt Mechanism: The Halt Mechanism option allows you to select the method used for halting real time emulation. Refer to Chapter 7 for further explanation.

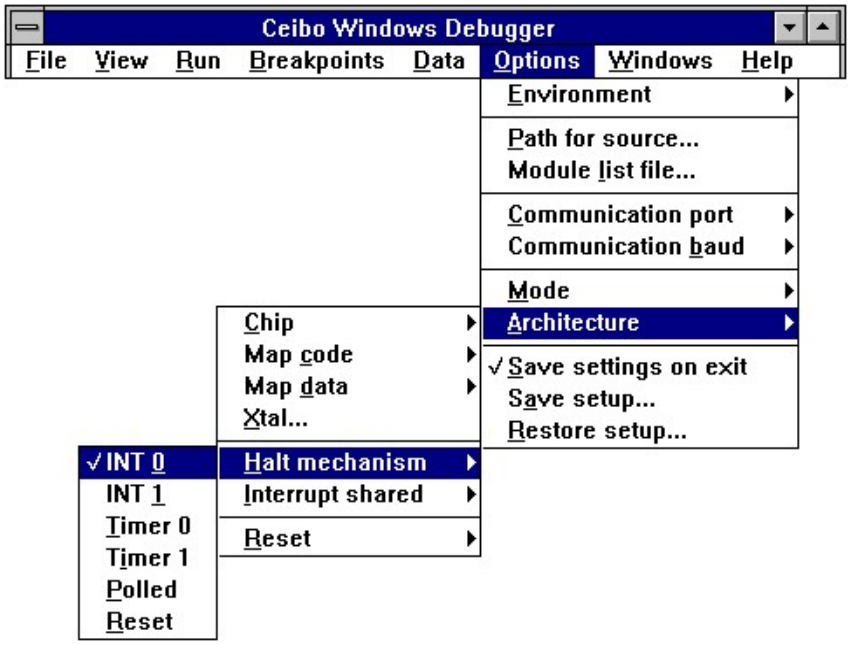


FIGURE 5.34: ***Halt Mechanism***

The available methods include: Interrupts, Reset and Polling.

Interrupt Int0: Selecting Int 0 as the Halt mechanism will configure the EB-51, so that interrupts on port INT0 of the target emulated Microcontroller will halt the program running in real time, whenever the Halt command is issued. This interrupt can also be shared with the user interrupt routine. Use the Interrupt shared option to set this.

Interrupt Int1: Selecting Int 1 as the Halt mechanism will configure the EB-51, so that interrupts on port INT1 of the target emulated Microcontroller will halt the program running in real time, whenever the Halt command is issued. This interrupt can also be shared with the user interrupt routine. Use the Interrupt shared option to set this.

Interrupt Timer 0: Selecting Timer 0 as the Halt mechanism will configure the EB-51, so that interrupts on Timer 0 overflow will halt the program running in real time, whenever the Halt command is issued. This interrupt can also be shared with the user interrupt routine. Use the Interrupt shared option to set this.

Interrupt Timer 1: Selecting Timer 1 as the Halt mechanism will configure the EB-51, so that interrupts on Timer 1 overflow will halt the program running in real time, whenever the Halt command is issued. This interrupt can also be shared with the user interrupt routine. Use the Interrupt shared option to set this.

Polled: Selecting Polled as the Halt mechanism will configure the EB-51, so that user polling of the supplied monitor routine will halt the program running in real time, whenever the Halt command is issued. It is your responsibility to add the monitor call instruction in any main or time consuming loop of your program, in which you want the Halt command to be operative. This option is intended for applications who require the use of all the emulated Microcontroller resources and interrupts.

Reset: Selecting Reset as the Halt mechanism will configure the EB-51, so that hardware reset will halt the program running in real time, whenever the Halt command is issued. This will evidently cause a Halt command to always stop the program at location 0000h. Information on the whereabouts of your program before issuing the Halt command will not be available. This option is intended for applications who requires the use of all the emulated Microcontroller resources and interrupts.

Interrupt Shared: Select this option when you want to stop the emulation by using an interrupt that is also being used in your application code.

Reset: The Reset command enables, or disables the reset input from the target hardware while running. When enabled and the program is running, the external target reset signal is connected to the emulated Microcontroller.

When disabled, this signal is disconnected from the emulated Microcontroller.

This is useful in systems with external watchdog circuitry, whenever it is desired to disable the watchdog operation.

Save Settings on Exit

Select this command if you want to save the setup while leaving the debugger. This setup will be restored while invoking again the debugger.

Save Setup

The Save Setup command allows you to save the options set in the options and window layouts at any time and with any filename.

Restore Setup

The Restore Setup command allows you to load a configuration file from disk.

The configuration file should have been previously saved by using the Save Setup command.

5.8. Windows Menu

The Window menu commands allow various operations on the currently open windows with the following commands: Tile, Cascade, Close all and Restore Standard.

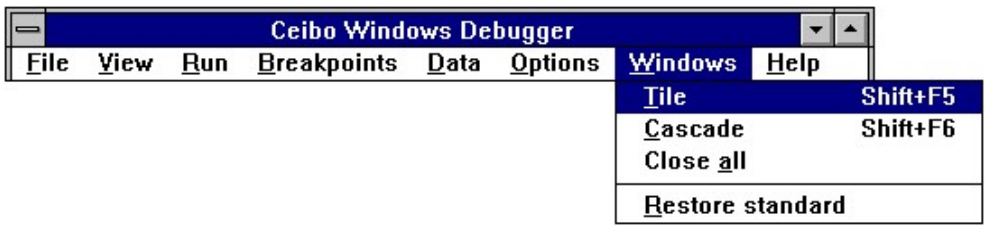


FIGURE 5.35: ***Windows Menu***

5.9. Help Menu

The Help menu commands open a help window for whichever subject will be selected from the menu. This menu offers the following options: Index, Topic Search and About.

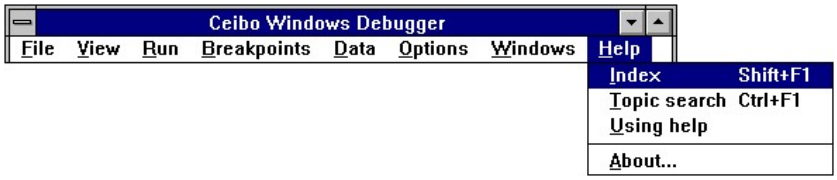


FIGURE 5.36: *Help Menu*

Index

The Index command displays a list of help topics. Not all the help contexts are listed, only the useful subjects and starting points.

Shift-F1 is the hot key that executes this command.

Topic Search

The Topic Search command find specific information about the debugger features and commands.

Alt-F1 is the hot key that executes this command.

About

The About command displays the debugger software version.

CHAPTER 6

Working with the Hardware

CHAPTER 6

Working with the Hardware

6.1. Introduction

Follow the steps of the session example for a quick introduction to the In-Circuit Simulation Mode.

The In-Circuit Simulation Mode option executes your program instructions with the simulator, but affects the Special Function Registers (SFR) of the microcontroller installed in the EB-51. Thus, ports and other SFRs will be altered while executing your program.

Although this working mode is not in real time, it easily checks your hardware like in the emulation mode and enables the Trace and some other useful functions.

Therefore, until your program properly interacts with your application hardware, use the In-Circuit Simulation Mode to debug and integrate the hardware and software.

6.2. Starting-Up

1. Invoke the debugger.

2. Select the Options Menu, choose the right Communication Port and set the Mode to In-Circuit Simulation.

3. Apply a Reset by pressing Ctrl-F2. Now you are ready to operate the system in the In-Circuit Simulation Mode.

6.3. Port Testing

The following steps describe how to test a port connected to a target hardware. The simple program you are going to try is a counter on Port 1 acting as output.

1. Select the View Menu and open the Watches Window.

2. Add P1 (Port 1) to the watches.

3. Select the View Menu and open the CPU Window.

4. Use the on-line assembler to modify the CPU and change the code as follows:

Address 0000h ..... INC P1

Address 0002h ..... AJMP $0h

The first instruction increments Port 1. The second instruction is a jump to the absolute address 0h. The $ symbol indicates that the numeric value is an absolute address and not the offset.

5. Press Ctrl-Break to halt the program execution. Observe that the Watches Window has been updated with the actual Port 1 value.

6. Select the View Menu and choose the Trace command to display the history of the executed instructions.

6.4. High-Level Language Debugging

1. Press Alt-F to activate the File menu.

2. Set the cursor to highlight the Load option and press the Enter key.

3. Select the CTESTS.ABS sample file and Keil/Franklin Type. This file includes code and symbols.

4. The file generates a sequence that is an output to the microcontroller port lines.

5. After successfully loading the CTESTS.ABS file, the Module window will be opened with the updated code.

6. Open the Watches window.

7. Type P1 to add Port 1 to the Watches window.

8. Select the View Menu and the Modules command to choose the CTESTS Module.

9. Select the Run Menu and execute a Program Reset. This can be done directly by pressing Ctrl-F2.

10. Execute the program by pressing the F9 key. Connect the ribbon cable between Port 1 testpoints and the LEDs connector and observe the port outputs as they are changing while running the program.

11. Halt the program execution by pressing Ctrl-Break.

12. Practice the capabilities of the In-Circuit Simulation Mode while testing the port outputs.

13. Press Alt-X to leave the Debugger.

CHAPTER 7

Real Time Emulation

CHAPTER 7

Real Time Emulation

7.1. Introduction

The Emulation Mode option sets the Debugger to operate in real time using the EB-51 emulator.

As the system uses some of the microcontroller resources in Emulation Mode, you must learn how to use it and how to prepare your code in order to avoid operational problems.

Table 7.1 provides a list of used resources and a quick reference about the actions that you have to carry out while working in real-time with EB-51.

Please read carefully this chapter before using EB-51 in Emulation Mode.

7.2. Emulation Memory

1. ***Address FC00h-FFFFh:*** Real time emulation is carried out by downloading your code into the RAM memory of EB-51. The upper 1 KByte of the 64 KByte code memory is automatically loaded with a Monitor program to control the emulation.

2. ***Address 0000h-0002h:*** Before executing your program in real-time, some areas of your code are changed automatically. The Reset vector content is modified by an LJMP Monitor instruction, so your own code must begin with an LCALL or LJMP instructions. The same usage of the Reset vector does not allow to write JMP or CALL instructions with address 0001h or 0002h as destination.

3. ***Address 0000h-FBFFh:*** Code memory cannot be mapped external and always belongs to the emulator system.

µC Resource	Function	Restriction/How to use it	Para.
Address 0000h-0002h	Reset Vector	- Initialize the Reset vector with LJMP or LCALL instructions.	7.2
Address FC00h-FFFFh	Monitor	- 1K reserved by the system.	7.2
Address 0000h-FBFFh	Breakpoint	- Replaces 3-5 bytes of code memory.	7.4
RD/WR - P3.6/P3.7	Reserved I/O	- Use them as memory RD/WR only. - Do not clear register bits P3.6 and P3.7 directly or indirectly by writing to P3.	7.5
Port 0,2	I/O Port and A/D Bus	- Select a ROMed chip for I/O use or a ROMless chip for A/D bus use.	7.6
MOVX @Ri	Paged MOVX	- Replace it by MOVX @DPTR instructions.	7.6
EA, IE.7	Enable Interrupt	- Do not disable if interrupt Halt Mechanism is selected.	7.8
Timer 0 not shared	Halt Mechanism	- Do not alter register bits ET0, PT0, TR0 directly or indirectly by writing to IE, IP, TCON, TMOD. - Write NOPs to addresses 000Bh to 000Dh.	7.12
Timer 0 shared	Halt Mechanism	- Ctrl-Break stops emulation with Timer 0 interrupt. - Initialize the Interrupt vector with LJMP or LCALL instructions.	7.13
Timer 1 not shared	Halt Mechanism	- Do not alter register bits ET1, PT1, TR1 directly or indirectly by writing to IE, IP, TCON, TMOD. - Write NOPs to addresses 001Bh to 001Dh.	7.14
Timer 1 shared	Halt Mechanism	- Ctrl-Break stops emulation with Timer 1 interrupt. - Initialize the Interrupt vector with LJMP or LCALL instructions.	7.15
Int 0 not shared	Halt Mechanism	- Do not alter register bits EX0, PX0, IT0, P3.2 directly or indirectly by writing to IE, IP, CON, P3. - Write NOPs to addresses 0003h to 0005h. - Do not connect or write to P3.2 (INT0).	7.16
Int 0 shared	Halt Mechanism	- Ctrl-Break stops emulation with INT0 interrupt. - Initialize the Interrupt vector with LJMP or LCALL instructions.	7.17
Int 1 not shared	Halt Mechanism	- Do not alter register bits EX1, PX1, IT1, P3.3 directly or indirectly by writing to IE, IP, TCON, P3 - Write NOPs to addresses 0013h to 0015h. - Do not connect or write to P3.3 (INT1).	7.18
Int 1 shared	Halt Mechanism	- Ctrl-Break stops emulation with INT1 interrupt. - Initialize the Interrupt vector with LJMP or LCALL instructions.	7.19
Stack	Stack Pointer	- Do not define the stack address below 07h. - The system uses 3 bytes of the stack.	7.3
Voltage	3.3V/5V	- Set the Vcc switch to 5V.	7.7

TABLE 7.1: ***How to Use the Hardware***

7.3. Stack Pointer

1. ***System Stack:*** The emulator uses 3 bytes of the stack while halting real-time emulation.

2. ***Stack Pointer:*** This pointer must not be set to an address below 07h.

7.4. Breakpoints

Breakpoints in real time emulation are implemented by replacing the user code at the breakpoint location with an LCALL Monitor instruction.

Since an LCALL instruction is used it requires three address locations to be changed. Therefore, setting a breakpoint on a one or two byte instruction may also alter the next one or two consecutive instructions.

The following restrictions apply to breakpoints:

1. ***Breakpoint address:*** The address ***must*** belong to an opcode, otherwise undetermined operation will result.

2. ***Consecutive instructions:*** Setting a breakpoint to a one or two byte instruction will cause alteration of the next one or two consecutive instructions as well. This will be indicated on the screen with a bright red line marking. The original breakpoint instruction will be marked with a red line. If one of these instructions are executed before the originally marked instruction, a NOP will be executed instead of the original opcode. These instructions ***must not*** be executed before the breakpoint instruction because an undetermined operation may result.

3. ***Expression True Global Breakpoints:*** These Breakpoints will force execution to be done in In-Circuit Simulation Mode.

4. ***Reset and Interrupt Vectors:*** Breakpoints cannot be set on the Reset vector (address 0000h to 0002h) and on the three first addresses of an interrupt vector used by the Halt Mechanism.

7.5. RD and WR Lines

1. ***Port 3.6 and 3.7:*** These RD and WR lines may be used for MOVX operations and not as quasi-bidirectional I/O ports. Do not use any instruction that clears register bits P3.6 and P3.7.

2. ***Port 3:*** If your code has instructions that write to this port, you have to mask bits P3.6 and P3.7 to "1".

For example, instead of executing **MOV P3,A**, you should write:

```
ORL A,#C0h
MOV P3,A
```

7.6. Ports 0 and 2

These two ports may act as I/O lines or address/data buses. If the selected chip by the software is a ROMless device like 80C31, both ports behave as buses. Otherwise, if the selected chip is a ROMed device like 80C51 or 87C51FB, ports P0 and P2 may be used as I/O lines.

1. ***ROMless Operation:*** Port 0 can only be used as AD0-7 address/data bus lines. Port 2 can only be used as A8-15 address bus lines.

2. ***ROMed Operation:*** Port 0 can only be used as I/O port. Port 2 can only be used as I/O port.

3. ***MOVX @Ri:*** These instructions affect Port 2 state and are not supported by EB-51. Replace them by MOVX @DPTR instructions.

4. ***External Data:*** This data cannot be accessed while selecting a ROMed device like 87C51FB. Select a ROMless device like 80C31 while using MOVX instructions.

7.7. Voltage

Emulation is possible at 5V only. 3.3V support requires microcontrollers and two PLDs working at such voltage (U1 to U4). The 3.3V set of chips are marked as "L" devices and not "C". If you are not sure that you have a set of chips to emulate both 3.3V and 5V microcontrollers, set the Vcc switch to 5V.

7.8. Halting the Emulation

The EB-51 Halt Mechanism offers 10 different modes to Halt the user program while executing it in real-time, so that the state of the program can be examined and/or altered.

These modes define the microcontroller resources assigned for that purpose, which are used only when you press Ctrl-Break or select the Halt command from the Run menu.

The Halt Mechanism mode is selected from the Options and Architecture menus, Halt Mechanism and Interrupt Shared commands. The selected option does not affect the behavior of breakpoints.

Your selection determines which resources are going to be used while running in real-time. The three main categories of halt mechanisms are:

Reset: One way is halting real-time emulation by hardware reset, however this will halt the program at address 0000h.

Polled: If you do not want to assign any resources for halting the emulation, a polling option is available. The polling is implemented by adding an LCALL Monitor (address 0FC08h) instruction to the main loop of your program.

Interrupt: You can choose an unused interrupt to halt the emulation, like INT0, INT1, TIMER 0 and TIMER 1 interrupts. The interrupts may also be shared with your own

interrupt routine if required. The software disk includes some files that you can link to your application while selecting the interrupt.

If you try to stop your program with the Halt command (Ctrl-Break), and the debugger continues running, this means that the selected Halt Mechanism condition did not occur. The debugger will only halt the emulation the next time that the selected Halt Mechanism condition occurs. Executing the Reset command (Ctrl-F2) after executing the Halt command and before the debugger is halted will force the program to stop at address 0000h.

7.9. Halt Mechanism - RESET

This is the most easy to use Halt Mechanism. It requires ***no*** resources of the microcontroller or of your application. However, it will cause the microcontroller to halt in the Reset state; this mean at location 0000h and with all SFR's initialized to their respective Reset value. There will be no indication on the program counter prior to halting. Internal Ram data contents will be preserved, and will reflect the values of your program as were immediately prior to halting. If you are having difficulties adapting the other Halt Mechanism modes to your application, choose this option as it will always work.

7.10. Halt Mechanism - Polled

This mode requires no microcontroller resources, and may halt inside any loop of the user application program. However, it requires that the user will insert an LCALL POLL_MONITOR (***LCALL 0FC08h***) instruction in the program where halting will be desired. This will only allow halting in loops where the above LCALL has been inserted.

Use the POLLMON.ASM/OBJ files contained in the software disk to compile and link with your program. These files contain the Public declaration of POLL_MONITOR label location.

The monitor is invoked each time it is called by the user LCALL POLL_MONITOR, and checks if halting is needed. If not, it returns to the user program, otherwise it remains in the monitor and informs the debugger that halting has been achieved successfully.

After the Halt command is issued from the debugger (Ctrl-Break), actual halting will only occur on the next LCALL POLL_MONITOR instruction execution If the emulation does not stop, you may press Ctrl-F2 to stop it at address 0000h.

7.11. Using Interrupts to Halt the Emulation

When setting the Halt mechanism to any one of the interrupt options, the first rule is not disabling the interrupt. EA bit (IE.7) must be set as well as the enable interrupt bit of the selected interrupt.

The second consideration is to handle the interrupt vector addresses. If you reserve the interrupt for the emulator, the first 3 addresses of the interrupt vector must be filled

with NOPs, so you will be sure that your compiler did not use these addresses for your application code. The debugger will give an interrupt warning message whenever code other than 00h is found on the reserved interrupt vector locations.

When using shared interrupts, the system replaces the interrupt vector by an LCALL instruction. Therefore, your code must never jump directly to the two address locations following the first address of the interrupt vector.

7.12. Halt Mechanism -Timer 0 not Shared

This mode can only be used when the Timer 0 and the Timer 0 interrupt are ***not used*** by your application program. It requires that the 3 locations of the Timer 0 interrupt vector (000Bh to 0000Dh) will not be used by your program, and will contain no code other than NOP (00h).

The monitor after reset will automatically run Timer 0, enable its interrupt in high priority and enable the global interrupt. It is your responsibility not to stop Timer 0, not to disable the global interrupt and the Timer 0 interrupt bits, otherwise the Halt command will not function.

You may use the supplied HALTTIM0.ASM/OBJ assembly and object files to compile and link with your code in order to reserve the interrupt vector locations, so that the linker/locator will not attempt to locate your application code there.

The Timer 0 interrupt will constantly run in the user's application background, causing a slight degradation of real-time operation.

The monitor is invoked each time the Timer 0 interrupt occurs, and checks if halting is needed, if not it returns to the location where the interrupt occurred, otherwise it remains in the monitor and informs the debugger that halting has been achieved successfully.

After the Halt command is issued from the debugger, actual halting will only occur on the next Timer 0 interrupt. This may take some time (depending on Timer 0 mode and frequency) and may take forever if your application had stopped the Timer or disabled the interrupt. If the emulation does not stop, you may press Ctrl-F2 to stop it at address 0000h.

7.13. Halt Mechanism -Timer 0 Shared

This mode may only be used when the Timer 0 and the Timer 0 interrupt are ***used*** by your application program. It is your responsibility to handle the Timer 0 control bits and interrupt control bits. Furthermore, you ***must*** have a valid interrupt routine (or a jump/call to one) beginning at the Timer 0 interrupt vector (000Bh) and occupying at least 3 locations (000Bh to 000Dh). These locations ***must not*** contain any relative or absolute jump instructions (i.e. SJMP, AJMP, etc.). Only sequential and LJMP/LCALL instructions are allowed.

This Halt Mechanism will only add a few instructions to the Timer 0 interrupt routine, thus having hardly any affect on real-time operation (as opposed to the not Shared mode).

The monitor is invoked each time the Timer 0 interrupt occurs, and checks if halting is needed. If the interrupt has been generated by the debugger Halt command (Ctrl-Break), the Monitor program takes control and informs the debugger that halting has been achieved successfully. Otherwise, control passes to your Timer 0 interrupt routine.

After the Halt command is issued from the debugger, actual halting will only occur on the next Timer 0 interrupt. This may take some time (depending on Timer 0 mode and frequency) and may take forever if your application had stopped the Timer or disabled the interrupt. If the emulation does not stop, you may press Ctrl-F2 to stop at address 0000h.

7.14. Halt Mechanism - Timer 1 not Shared

This mode can only be used when the Timer 1 and the Timer 1 interrupt are ***not used*** by your application program. It requires that the 3 locations of the Timer 1 interrupt vector (001Bh to 0001Dh) will not be used by your program, and will contain no code other than NOP (00h).

The monitor after reset will automatically run Timer 1, enable its interrupt in high priority and enable the global interrupt. It is your responsibility not to stop Timer 0, not to disable the global interrupt and the Timer 1 interrupt bits, otherwise the Halt command will not function.

You may use the supplied HALTTIM1.ASM/OBJ assembly and object files to compile and link with your code in order to reserve the interrupt vector locations, so that the linker/locator will not attempt to locate your application code there.

The Timer 1 interrupt will constantly run in the user's application background, causing a slight degradation of real-time operation.

The monitor is invoked each time the Timer 1 interrupt occurs, and checks if halting is needed, if not it returns to the location where the interrupt occurred, otherwise it remains in the monitor and informs the debugger that halting has been achieved successfully.

After the Halt command is issued from the debugger, actual halting will only occur on the next Timer 1 interrupt. This may take some time (depending on Timer 1 mode and frequency) and may take forever if your application had stopped the Timer or disabled the interrupt. If the emulation does not stop, you may press Ctrl-F2 to stop it at address 0000h.

7.15. Halt Mechanism - Timer 1 Shared

This mode may only be used when the Timer 1 and the Timer 1 interrupt are ***used*** by your application program. It is your responsibility to handle the Timer 1 control bits and interrupt control bits. Furthermore, you ***must*** have a valid interrupt routine (or a jump/call to one) beginning at the Timer 0 interrupt vector (001Bh) and occupying at least 3 locations (001Bh to 001Dh).

These locations ***must not*** contain any relative or absolute jump instructions (i.e. SJMP, AJMP, etc.). Only sequential and LJMP/LCALL instructions are allowed.

This Halt Mechanism will only add a few instructions to the Timer 1 interrupt routine, thus having hardly any affect on real-time operation (as opposed to the not Shared mode). The monitor is invoked each time the Timer 1 interrupt occurs, and checks if halting is needed. If the interrupt has been generated by the debugger Halt command (Ctrl-Break), the Monitor program takes control and informs the debugger that halting has been achieved successfully. Otherwise, control passes to your Timer 1 interrupt routine.

After the Halt command is issued from the debugger, actual halting will only occur on the next Timer 1 interrupt. This may take some time (depending on Timer 1 mode and frequency) and may take forever if your application had stopped the Timer or disabled the interrupt. If the emulation does not stop, you may press Ctrl-F2 to stop at address 0000h.

7.16. Halt Mechanism - INT 0 not Shared

This mode can only be used when the INT0 pin and the EXT INT0 interrupt are ***not used*** by your application program. It requires that the 3 locations of the INT0 interrupt vector (0003h to 0005h) will not be used by your program, and will contain no code other than NOP (00h).

The monitor after reset will automatically set INT0 to be edge sensitive, enable EXT INT0 interrupt in high priority and enable the global interrupt. It is your responsibility not to disable the global interrupt and the INT0 interrupt bits, otherwise the Halt command will not function. The EXT INT0 interrupt will constantly run in your application background, but if the pin is unused, no interrupt will be generated, thus not affecting real-time operation.

You may use the supplied HALTINT0.ASM/OBJ assembly and object files to compile and link with your code in order to reserve the interrupt vector locations, so that the linker/locator will not attempt to locate your application code there.

The EXT INT0 interrupt will constantly be active although no interrupt will be generated until you press Ctrl-Break.

After the Halt command is issued from the debugger, the system will generate a short low going pulse on the INT0 pin of the target microcontroller. If the emulation does not stop, you may press Ctrl-F2 to stop at address 0000h.

7.17. Halt Mechanism - INT 0 Shared

This mode may only be used when the EXT INT0 interrupt is ***used*** by your application program. It is your responsibility to handle the EXT INT0 control bits and interrupt control bits. Furthermore, you ***must*** have a valid interrupt routine (or a jump/call to one) beginning at the INT0 interrupt vector (0003h) and occupying at least 3 locations (0003h to 0005h). These locations ***must not*** contain any relative or absolute jump instructions (i.e. SJMP, AJMP etc.). Only sequential and LJMP/LCALL instructions are allowed.

This Halt Mechanism will only add a few instruction to your INT0 interrupt routine, thus having hardly any affect on real-time operation.

The monitor is invoked each time the INT0 interrupt occurs, and checks if halting is needed. If the interrupt has been generated by the debugger Halt command (Ctrl-Break), the Monitor program takes control and informs the debugger that halting has been achieved successfully. Otherwise, control passes to your INT0 interrupt routine. After the Halt command is issued from the debugger, the system will generate a short low going pulse on the INT0 pin of the target microcontroller. If the emulation does not stop, you may press Ctrl-F2 to stop it at address 0000h.

7.18. Halt Mechanism - INT 1 not Shared

This mode can only be used when the INT1 pin and the EXT INT1 interrupt are ***not used*** by your application program. It requires that the 3 locations of the INT0 interrupt vector (0013h to 0015h) will not be used by your program, and will contain no code other than NOP (00h).

The monitor after reset will automatically set INT1 to be edge sensitive, enable EXT INT1 interrupt in high priority and enable the global interrupt. It is your responsibility not to disable the global interrupt and the INT1 interrupt bits, otherwise the Halt command will not function.

The EXT INT1 interrupt will constantly run in your application background, but if the pin is unused, no interrupt will be generated, thus not affecting real-time operation.

You may use the supplied HALTINT1.ASM/OBJ assembly and object files to compile and link with your code in order to reserve the interrupt vector locations, so that the linker/locator will not attempt to locate your application code there.

The EXT INT1 interrupt will constantly be active although no interrupt will be generated until you press Ctrl-Break.

After the Halt command is issued from the debugger, the system will generate a short low going pulse on the INT1 pin of the target microcontroller. If the emulation does not stop, you may press Ctrl-F2 to stop at address 0000h.

7.19. Halt Mechanism - INT 1 Shared

This mode may only be used when the EXT INT1 interrupt is ***used*** by your application program. It is your responsibility to handle the EXT INT1 control bits and interrupt control bits. Furthermore, you ***must*** have a valid interrupt routine (or a jump/call to one) beginning at the INT1 interrupt vector (0013h) and occupying at least 3 locations (0013h to 0015h). These locations ***must not*** contain any relative or page absolute instructions (i.e. SJMP, AJMP etc.). Only sequential and LJMP/LCALL instructions are allowed.

This Halt Mechanism will only add a few instruction to your INT1 interrupt routine, thus having hardly any affect on real-time operation.

The monitor is invoked each time the INT1 interrupt occurs, and checks if halting is needed. If the interrupt has been generated by the debugger Halt command (Ctrl-Break), the Monitor program takes control and informs the debugger that halting has been achieved successfully. Otherwise, control passes to your INT1 interrupt routine. After the Halt command is issued from the debugger, the system will generate a short low going pulse on the INT1 pin of the target microcontroller. If the emulation does not stop, you may press Ctrl-F2 to stop it at address 0000h.

7.20. Customizing the System

EB-51 behavior while halting real-time emulation and reassuming it may be customized. That means, you can add your own instructions to control the state of the microcontroller while halting and also while restarting your code. This function is useful if you need to set a port state or any other memory or SFR location to a desired value when the program stops running.

By using this special feature you can add your own code to stop and start peripherals belonging to the microcontroller and not supported by the Monitor program.

You may add up to 16 bytes containing specific entrance and exit code: 8 bytes on entrance and 8 bytes on exit. This code will be executed by the Monitor immediately following Halt or Breakpoint operation (Entrance) and just before Running back your program (Exit). This code will be executed approximately 50-60 cycles after Halting and before Running.

This code is restricted in the resources it may use. Any misuse of the microcontroller resources will cause undetermined behavior, and may corrupt your program which relies on the value of the registers saved while stopping the emulation.

Do not alter ACC, B, DPTR, PSW and the stack pointer.

The Carry bit may be used freely.

You may reserve some on-chip RAM locations that your application does not need to manipulate and save temporary variables. The example program given in Table 7.1 assumes location 7Fh to be unused.

You may use LCALL instructions to call external routines if more than 8 bytes are needed, but this must be to a known absolute address, and it will require 2 more stack locations.

You have to prepare an Intel Hex file with the following contents:

1. Locations 00h-07h should contain the user monitor entrance code.
2. Locations 10h-17h should contain the user monitor exit code.
3. Locations 08h-0Fh may only contain NOPs (00h).
4. Locations 18h-1Fh may only contain NOPs (00h).
5. No code may be located above location 1Fh.
6. The file must have the extension .UMH.

You may use the on-line assembler to generate this file and then save it on the disk. Then, while loading your file, you can click the Options button in the File Dialog box and enable the loading of your customized application too. Following is an example of a file that handles the PCA timer available in the 8xC552 microcontrollers. This code will save TM2CON SFR to address 7Fh of the internal RAM, and will restore the TM2CON value from this location.

```
ORG   00h
MOV   7Fh,TM2CON          ; Save TM2CON value
ANL   TM2CON,#11111100b   ; Clear T2MS0, T2MS1 (Halt)
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
END

ORG   10h
MOV   TM2CON,7Fh          ; Set all of TM2CON to its previous state
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
END
```

TABLE 7.2: ***PCA Control Program***

CHAPTER 8

Utilities, Software Support and Syntax

CHAPTER 8

Utilities, Software Support and Syntax

8.1 Introduction

Two utilities are included in the software for assemblers and C compilers. FIXASM and FIXC are the utilities provided to support the listing files which are not fully compatible with Intel format. This chapter explains the use of these programs. The support for different software vendors and the Windows Symbolic Debugger syntax are covered in this chapter as well.

8.2 FIXASM Utility

This utility modifies the Assembler listing files to be accepted by the DOS Debugger. The Windows Debugger does not require this utility.

Output file overwrites the input file, although you may select a different output filename.

Run FIXASM without parameters for command line syntax and options.

FIXASM acts as follows: it scans the input file and replaces the beginning of every valid line containing the assembler directives ' RSEG ' and ' CSEG ' with the pattern '----', as it is defined by Intel format.

- Intel ASM51 Assembler does not need to be fixed up, because it generates the listing file in the above mentioned format.
- IAR/Archimedes Assembler does not need to be fixed up, because it generates line information.
- Keil/Franklin A51 Assembler needs to be fixed up because it lacks the above pattern in the ' RSEG ' directive line.
- MCC MA51 Assembler needs to be fixed up because it lacks the above pattern in the ' CSEG ' directive line.

8.3. FIXC Utility

This utility modifies the C listing files to be accepted by the DOS Debugger. The Windows Debugger does not require this utility.

The modification is intended to support two C Compilers: MCC and IAR/Archimedes.

Output file is by default input file name with .LST extension although it may be redirected to another output file.

Run FIXC without parameters for command line syntax and options.

FIXC with /M option is needed while using MCC MCC51 C Compiler package. This option adds line numbers to your C source file (i.e. filename.c). Output filename is by default the input filename with .LST extension. Output file must differ from input file. The generated output file is a line by line copy of the input file with a line counter added to the beginning of each source line. A total of 6 characters are added to the beginning of each line with the format: 'xxxx '.

FIXC with /A option is needed while using IAR/Archimedes ICC8051 C Compiler package. With this option FIXC expects the input file to be the listing output file of the compiler.

Output filename is by default input filename of your C source with .LST extension. The generated output file is a line by line copy of the input file with the line counter format changed to the following format 'xxxx '.

This modification is not mandatory. The debugger will fully function without changing the IAR listing files, although significant speed performance improvement will be achieved with large files after running the utility.

- Keil/Franklin C51 Compiler does not need any modification.
- MCC MCC51 Compiler needs to be fixed up because it does not generate a valid list file with the correct line numbering format. Invoke the FIXC program with the /M option and with the source filename that was entered into the compiler.
- IAR/Archimedes ICC8051 Compiler fix-up is recommended for improved speed performance with large list files.

8.4. Debugging Assembler Files

Ceibo's Debuggers can be used with files generated by Intel, Signetics, Philips, BSO/Tasking and other Assemblers that generate Intel OMF files and compatible listings.

The first thing you have to do is to prepare the assembler files. Use the DEBUG option while assembling your program to include symbol information in the object file. The LIST option is also mandatory.

The Debug Option of your Assembler generates symbol information such as Line Numbers, Public and Local references, Labels and others. Ceibo's software recognizes those symbols.

The List Option creates an ASCII file that contains the source written by the user and references to Line Numbers. Ceibo's software allows invoking the List of your program and debugging it according to assembler language lines, while showing all the variables you want to watch.

The normal way to prepare your program for debugging is as follows:

1. Use an Editor to write your source code.

2. Assemble you sources with an Assembler program using the Debug and List options.

3. Link and locate your object files with the Intel RL51 program or a similar linker and locator that generates an Intel compatible object file.

   If you are using an Assembler with linking and relocating capabilities (Intel, Keil/Franklin, BSO/Tasking , etc.) you must write all your code under a CSEG or RSEG directive.

   If the files are written to be linked with other programs, you may use the RSEG directive. Some assembler programs do not generate the required debugging information to support Relocatable Code Segments. Therefore, it is necessary to use the following syntax with RSEG:

   myseg SEGMENT CODE

   RSEG myseg

   mylabel:

   The only difference is the Label added to the third line, that is required to recognize the address of the relocatable code segment.

8.5. Using IAR Systems Compiler Package

Like any other compiler, working with IAR Systems compiler requires three major steps:

1. Writing a program in any given editor.

2. Compiling with ICC8051 compiler.

3. When no errors are detected, linking the programs to a desired memory location.

Compiling

The compiler must produce the following files :

1. File with extension *.R03 - object file

2. File with extension *.LST - listing file

Required compiler options:

-*r* : include debug information

-*L* : produce list file

Linking

While linking object file with IAR Xlink the *.XCL file must be created. The required linker options are:

-faomf8051:	Produce file in OMF-51 format for Ceibo DOS Debugger only.
-fdebug:	Produce file in UBROF format for Ceibo Windows Debugger only.

When linking file with the IAR System environment Link, Options must be set to the following:

debug option 1:	None
format 2:	*aomf8051* for Ceibo DOS Debugger only or *debug* for Ceibo Windows Debugger

Preparing for Downloading

Before downloading your absolute file into the DOS Debugger only, you have to fix all your list files with the FIXC utility program.

The Windows Debugger does not require the FIXC utility.

The command line for running FIXC is :

FIXC <your module file name>.LST /A

Loading the file with the DOS Debugger

Load the file resulting from the Linker. The default extension of the file is *.A03*.

Loading the file with the Windows Debugger

Specify the vendor and load the file resulting from the Linker. The default extension of the file is *.DBG*.

8.6. Using Keil Software

This software is supported with the OMF format.

Use the following command lines with *noamake* in both C51 and L51:

c51 hello.c *debug objectextend noamake*

l51 hello.obj, my_libraries *noamake*

The *noamake* switch is necessary only for the Ceibo DOS Debugger.

While using Windows Debugger, select the vendor Keil/Franklin in the Load Dialog Box.

8.7. Using BSO/Tasking Software

To prepare a file for debugging use the following options:

Compiler Switches

-g: include debug information

i.e. *cc51 -Ms -g* sieve.c

ASM Switches

debug: include debug information

i.e. *asm51* sieve *debug*

LINK Switches

The Linker does not need any special options.

Generate a file in OMF-51 Format

i.e. *omf51* sieve.out sieve.abs

8.8. Using MCC Software

To prepare a file for debugging use the following options:

Compiler Switches

mcc51 hello.c */l1/t/w0* ;do not use spaces between options

ASM Switches

ma51 hello.src *debug nolist noprint format(momf)*

LINK Switches

ml51 hello.obj+*mcm51.lib format(iomf)*

Generating valid Debug Information

mcsu hello

Generating Valid Listing File for DOS Debugger with FIXC

FIXC hello.c hello.lst */m*

8.9. Windows Debugger Syntax

Predefined Names

EB-51 accepts the following symbols that you may invoke directly while adding a watch, setting a breakpoint or assembling an instruction. The complete list of predefined Ports and Register names may be found in the Microcontroller Data Book.

Ports: P0, P1, P3, etc.

Port Bits: P0.0 to P0.2, P1.0 to P1.7, etc.

Registers: ACC, B, etc.

Register Bits: ACC.0 to ACC.7, B.0 to B.7, etc.

Absolute References

You can make an absolute reference using the first letter to define the variable type followed by a colon and the address. Furthermore, absolute references may be expressed with a length. The syntax is:

absolute_reference:address,length

Absolute references are D for the on-chip RAM, X for external RAM accessed by MOVX instructions, P for ports and any SFR (special function registers), C for code memory and B for bit memory (below 80H is for RAM bits and above that value represents SFR bits). Some examples are:

D:100,5

X:MYSYMBOL,2

B:0x80,5

B:P1.0,8

P:0xFE

Addresses are entered using the syntax of the selected language in the Options menu. Length is always decimal.

Direct Access to Local Variables and Executable Lines

The general syntax for line numbers is:

#module name#line number

Local variables of modules are invoked as:

#module_name#variable

A local variable of a procedure has the following syntax:

#module_name#procedure_name#variable

Examples:

#test#12	means line number 12 belonging to module TEST.
#tcdelayt#delay#DelayCnt	means variable DelayCnt belonging to procedure delay and module tcdelayc.

CHAPTER 9

On-Line Assembler

CHAPTER 9

On-Line Assembler

9.1. Introduction

EB-51 software includes two useful functions: On-line Assembler and Disassembler. Both functions permit the user to translate instructions from and into mnemonics.

The On-line Assembler command assembles a single instruction mnemonic into the code memory.

After writing an instruction mnemonic, press the <ENTER> key If an invalid instruction is entered, a syntax error message is displayed.

The Disassembler function allows the user to disassemble memory values, into instruction mnemonics.

This function assumes that the starting address points to the first byte of an instruction and takes second and third bytes if they are necessary to be used as operands.

Stepping down through the CPU window assumes the next instruction is an opcode.

Stepping up in the CPU window may result with erroneous mnemonic display.

9.2. Assembler Syntax

- The comma is used to separate operands.
- A dollar symbol ($) is used to specify absolute jump addresses. If it is omitted, the specified address is relative.

The following are examples of valid expressions:

JBC 12H,$123H

CPL P1.1

AJMP 0H

The Disassembler function allows the user to disassemble memory values, into instruction mnemonics.

This function assumes that the starting address points to the first byte of an instruction and takes second and third bytes if they are necessary to be used as operands.

Stepping down through the CPU window assumes the next instruction is an opcode.

Stepping up in the CPU window may result with erroneous mnemonic display.

9.3. Instruction Set

The following pages detail the syntax of the mnemonics and operands used by the On-line Assembler and Disassembler functions.

HEX CODE	NUMBER OF BYTES	SYNTAX	EXAMPLE
00	1	NOP	NOP
01	2	AJMP page-addr	AJMP 0010H
02	3	LJMP code-addr	LJMP 00F3H
03	1	RR A	RR A
04	1	INC A	INC A
05	2	INC byte-addr	INC 1FH
06	1	INC @R0	INC @R0
07	1	INC @R1	INC @R1
08	1	INC R0	INC R0
09	1	INC R1	INC R1
0A	1	INC R2	INC R2
0B	1	INC R3	INC R3
0C	1	INC R4	INC R4
0D	1	INC R5	INC R5
0E	1	INC R6	INC R6
0F	1	INC R7	INC R7
10	3	JBC bit-addr, rel-offset	JBC 02H, A0H
11	2	ACALL page-addr	ACALL 0000H
12	3	LCALL code-addr	LCALL 0110H
13	1	RRC A	RRC A
14	1	DEC A	DEC A
15	2	DEC byte-addr	DEC FAH
16	1	DEC @R0	DEC @R0
17	1	DEC @R1	DEC @R1
18	1	DEC R0	DEC R0
19	1	DEC R1	DEC R1
1A	1	DEC R2	DEC R2
1B	1	DEC R3	DEC R3
1C	1	DEC R4	DEC R4
1D	1	DEC R5	DEC R5
1E	1	DEC R6	DEC R6
1F	1	DEC R7	DEC R7

bit	addr	:	00H	-	FFH	-	8051 bit address
byte	addr	:	00H	-	FFH	-	On-chip 8-bit RAM address
code	addr	:	0000H	-	FFFFH	-	Absolute code address
page	addr	:	0000H	-	07FFH	-	11-bit code address
rel	offset	:	00H	-	FFH	-	8-bit 2's complement addr
data	byte	:	00H	-	FFH	-	Immediate data byte
data	word	:	0000H	-	FFFFH	-	Immediate data word

HEX CODE	NUMBER OF BYTES	SYNTAX	EXAMPLE
20	3	JB bit-addr, rel-offset	JB 01H, F5H
21	2	AJMP page-addr	AJMP 010FH
22	1	RET	RET
23	1	RL A	RL A
24	2	ADD A, #data-byte	ADD A, #22H
25	2	ADD A, byte-addr	ADD A, 32H
26	1	ADD A, @R0	ADD A, @R0
27	1	ADD A, @R1	ADD A, @R1
28	1	ADD A, R0	ADD A, R0
29	1	ADD A, R1	ADD A, R1
2A	1	ADD A, R2	ADD A, R2
2B	1	ADD A, R3	ADD A, R3
2C	1	ADD A, R4	ADD A, R4
2D	1	ADD A, R5	ADD A, R5
2E	1	ADD A, R6	ADD A, R6
2F	1	ADD A, R7	ADD A, R7
30	3	JNB bit-addr, rel-offset	JNB 0CH, 57H
31	2	ACALL page-addr	ACALL 0143H
32	1	RETI	RETI
33	1	RLC A	RLC A
34	2	ADDC A, #data-byte	ADDC A, #87H
35	2	ADDC A, byte-addr	ADDC A, ACH
36	1	ADDC A, @R0	ADDC A, @R0
37	1	ADDC A, @R1	ADDC A, @R1
38	1	ADDC A, R0	ADDC A, R0
39	1	ADDC A, R1	ADDC A, R1
3A	1	ADDC A, R2	ADDC A, R2
3B	1	ADDC A, R3	ADDC A, R3
3C	1	ADDC A, R4	ADDC A, R4
3D	1	ADDC A, R5	ADDC A, R5
3E	1	ADDC A, R6	ADDC A, R6
3F	1	ADDC A, R7	ADDC A, R7

bit	addr	:	00H	-	FFH	-	8051 bit address
byte	addr	:	00H	-	FFH	-	On-chip 8-bit RAM address
code	addr	:	0000H	-	FFFFH	-	Absolute code address
page	addr	:	0000H	-	07FFH	-	11-bit code address
rel	offset	:	00H	-	FFH	-	8-bit 2's complement addr
data	byte	:	00H	-	FFH	-	Immediate data byte
data	word	:	0000H	-	FFFFH	-	Immediate data word

HEX CODE	NUMBER OF BYTES	SYNTAX	EXAMPLE
40	2	JC rel-offset	JC BBH
41	2	AJMP page-addr	AJMP 0260H
42	2	ORL byte-addr, A	ORL 32H, A
43	3	ORL byte-addr, #data-byte	ORL 23H, #C5H
44	2	ORL A, #data-byte	ORL A, #67H
45	2	ORL A, byte-addr	ORL A, 43H
46	1	ORL A, @R0	ORL A, @R0
47	1	ORL A, @R1	ORL A, @R1
48	1	ORL A, R0	ORL A, R0
49	1	ORL A, R1	ORL A, R1
4A	1	ORL A, R2	ORL A, R2
4B	1	ORL A, R3	ORL A, R3
4C	1	ORL A, R4	ORL A, R4
4D	1	ORL A, R5	ORL A, R5
4E	1	ORL A, R6	ORL A, R6
4F	1	ORL A, R7	ORL A, R7
50	2	JNC rel-offset	JNC BAH
51	2	ACALL page-addr	ACALL 0220H
52	2	ANL byte-addr, A	ANL 04H, A
53	3	ANL byte-addr, #data-byte	ANL 23H, #C7H
54	2	ANL A, #data-byte	ANL A, #AEH
55	2	ANL A, byte-addr	ANL A, 03H
56	1	ANL A, @R0	ANL A, @R0
57	1	ANL A, @R1	ANL A, @R1
58	1	ANL A, R0	ANL A, R0
59	1	ANL A, R1	ANL A, R1
5A	1	ANL A, R2	ANL A, R2
5B	1	ANL A, R3	ANL A, R3
5C	1	ANL A, R4	ANL A, R4
5D	1	ANL A, R5	ANL A, R5
5E	1	ANL A, R6	ANL A, R6
5F	1	ANL A, R7	ANL A, R7

bit	addr	:	00H	-	FFH	-	8051 bit address
byte	addr	:	00H	-	FFH	-	On-chip 8-bit RAM address
code	addr	:	0000H	-	FFFFH	-	Absolute code address
page	addr	:	0000H	-	07FFH	-	11-bit code address
rel	offset	:	00H	-	FFH	-	8-bit 2's complement addr
data	byte	:	00H	-	FFH	-	Immediate data byte
data	word	:	0000H	-	FFFFH	-	Immediate data word

HEX CODE	NUMBER OF BYTES	SYNTAX	EXAMPLE
60	2	JZ rel-offset	JZ 04H
61	2	AJMP page-addr	AJMP 03F0H
62	2	XRL byte-addr, A	XRL DEH, A
63	3	XRL byte-addr, #data-byte	XRL E9, #77H
64	2	XRL A, #data-byte	XRL A, #55H
65	2	XRL A, byte-addr	XRL A, 01H
66	1	XRL A, @R0	XRL A, @R0
67	1	XRL A, @R1	XRL A, @R1
68	1	XRL A, R0	XRL A, R0
69	1	XRL A, R1	XRL A, R1
6A	1	XRL A, R2	XRL A, R2
6B	1	XRL A, R3	XRL A, R3
6C	1	XRL A, R4	XRL A, R4
6D	1	XRL A, R5	XRL A, R5
6E	1	XRL A, R6	XRL A, R6
6F	1	XRL A, R7	XRL A, R7
70	2	JNZ rel-offset	JNZ 56H
71	2	ACALL page-addr	ACALL 0323H
72	2	ORL C, bit-addr	ORL C, 7FH
73	1	JMP @A+DPTR	JMP @A+DPTR
74	2	MOV A, #data-byte	MOV A, #56H
75	3	MOV byte-addr, #data-byte	MOV 34H, #45H
76	2	MOV @R0, #data-byte	MOV @R0, #89H
77	2	MOV @R1, #data-byte	MOV @R1, #23H
78	2	MOV R0, #data-byte	MOV R0, #25H
79	2	MOV R1, #data-byte	MOV R1, #12H
7A	2	MOV R2, #data-byte	MOV R2, #FDH
7B	2	MOV R3, #data-byte	MOV R3, #15H
7C	2	MOV R4, #data-byte	MOV R4, #13H
7D	2	MOV R5, #data-byte	MOV R5, #E9H
7E	2	MOV R6, #data-byte	MOV R6, #A7H
7F	2	MOV R7, #data-byte	MOV R7, #14H

bit	addr	:	00H	-	FFH	-	8051 bit address
byte	addr	:	00H	-	FFH	-	On-chip 8-bit RAM address
code	addr	:	0000H	-	FFFFH	-	Absolute code address
page	addr	:	0000H	-	07FFH	-	11-bit code address
rel	offset	:	00H	-	FFH	-	8-bit 2's complement addr
data	byte	:	00H	-	FFH	-	Immediate data byte
data	word	:	0000H	-	FFFFH	-	Immediate data word

HEX CODE	NUMBER OF BYTES	SYNTAX	EXAMPLE
80	2	SJMP rel-offset	SJMP 0013H
81	2	AJMP page-addr	AJMP 0402H
82	2	ANL C, bit-addr	ANL C, 09H
83	1	MOVC A, @A+PC	MOVC A, @A+PC
84	1	DIV AB	DIV AB
85	3	MOV byte-addr, byte-addr	MOV 01H, 34H
86	2	MOV byte-addr, @R0	MOV 06H, @R0
87	2	MOV byte-addr, @R1	MOV 21H, @R1
88	2	MOV byte-addr, R0	MOV 15H, R0
89	2	MOV byte-addr, R1	MOV 05H, R1
8A	2	MOV byte-addr, R2	MOV 24H, R2
8B	2	MOV byte-addr, R3	MOV 09H, R3
8C	2	MOV byte-addr, R4	MOV 01H, R4
8D	2	MOV byte-addr, R5	MOV 20H, R5
8E	2	MOV byte-addr, R6	MOV 34H, R6
8F	2	MOV byte-addr, R7	MOV 67H, R7
90	3	MOV DPTR, #data-word	MOV DPTR, #AAAAH
91	2	ACALL page-addr	ACALL 0445H
92	1	MOV bit-addr, C	MOV 05H, C
93	2	MOVC A, @A+DPTR	MOVC A, @A+DPTR
94	2	SUBB A, #data-byte	SUBB A, #33H
95	1	SUBB A, byte-addr	SUBB A, 32H
96	1	SUBB A, @R0	SUBB A, @R0
97	1	SUBB A, @R1	SUBB A, @R1
98	1	SUBB A, R0	SUBB A, R0
99	1	SUBB A, R1	SUBB A, R1
9A	1	SUBB A, R2	SUBB A, R2
9B	1	SUBB A, R3	SUBB A, R3
9C	1	SUBB A, R4	SUBB A, R4
9D	1	SUBB A, R5	SUBB A, R5
9E	1	SUBB A, R6	SUBB A, R6
9F	1	SUBB A, R7	SUBB A, R7

bit	addr	:	00H	-	FFH	-	8051 bit address
byte	addr	:	00H	-	FFH	-	On-chip 8-bit RAM address
code	addr	:	0000H	-	FFFFH	-	Absolute code address
page	addr	:	0000H	-	07FFH	-	11-bit code address
rel	offset	:	00H	-	FFH	-	8-bit 2's complement addr
data	byte	:	00H	-	FFH	-	Immediate data byte
data	word	:	0000H	-	FFFFH	-	Immediate data word

HEX CODE	NUMBER OF BYTES	SYNTAX	EXAMPLE
A0	2	ORL C, /bit-addr	ORL C, /04H
A1	2	AJMP page-addr	AJMP 05FEH
A2	2	MOV C, bit-addr	MOV C, 7FH
A3	1	INC DPTR	INC DPTR
A4	1	MUL AB	MUL AB
A5	1	Reserved	Reserved
A6	2	MOV @R0, byte-addr	MOV @R0, 45H
A7	2	MOV @R1, byte-addr	MOV @R1, 33H
A8	2	MOV R0, byte-addr	MOV R0, FFH
A9	2	MOV R1, byte-addr	MOV R1, 00H
AA	2	MOV R2, byte-addr	MOV R2, E5H
AB	2	MOV R3, byte-addr	MOV R3, 34H
AC	2	MOV R4, byte-addr	MOV R4, 12H
AD	2	MOV R5, byte-addr	MOV R5, 67H
AE	2	MOV R6, byte-addr	MOV R6, 89H
AF	2	MOV R7, byte-addr	MOV R7, 32H
B0	2	ANL C, /bit-addr	ANL C, /23H
B1	2	ACALL page-addr	ACALL 0503H
B2	2	CPL bit-addr	CPL 02H
B3	1	CPL C	CPL C
B4	3	CJNE A, #data-byte, rel-offset	CJNE A, #32H, 23H
B5	3	CJNE A, byte-addr, rel-offset	CJNE A, 12H, 56H
B6	3	CJNE @R0, #data-byte, rel-offset	CJNE @R0, #66H, 23H
B7	3	CJNE @R1, #data-byte, rel-offset	CJNE @R1, #34H, 12H
B8	3	CJNE R0, #data-byte, rel-offset	CJNE R0, #23H, 56H
B9	3	CJNE R1, #data-byte, rel-offset	CJNE R1, #00H, 10H
BA	3	CJNE R2, #data-byte, rel-offset	CJNE R2, #56H, 23H
BB	3	CJNE R3, #data-byte, rel-offset	CJNE R3, #76H, 56H
BC	3	CJNE R4, #data-byte, rel-offset	CJNE R4, #23H, 45H
BD	3	CJNE R5, #data-byte, rel-offset	CJNE R5, #80H, ACH
BE	3	CJNE R6, #data-byte, rel-offset	CJNE R6, #BAH, CAH
BF	3	CJNE R7, #data-byte, rel-offset	CJNE R7, #09H, 14H

bit	addr	:	00H	-	FFH	-	8051 bit address
byte	addr	:	00H	-	FFH	-	On-chip 8-bit RAM address
code	addr	:	0000H	-	FFFFH	-	Absolute code address
page	addr	:	0000H	-	07FFH	-	11-bit code address
rel	offset	:	00H	-	FFH	-	8-bit 2's complement addr
data	byte	:	00H	-	FFH	-	Immediate data byte
data	word	:	0000H	-	FFFFH	-	Immediate data word

HEX CODE	NUMBER OF BYTES	SYNTAX	EXAMPLE
C0	2	PUSH byte-addr	PUSH 23H
C1	2	AJMP page-addr	AJMP 0604H
C2	2	CLR bit-addr	CLR 04H
C3	1	CLR C	CLR C
C4	1	SWAP A	SWAP A
C5	2	XCH A, byte-addr	XCH A, 12H
C6	1	XCH A, @R0	XCH A, @R0
C7	1	XCH A, @R1	XCH A, @R1
C8	1	XCH A, R0	XCH A, R0
C9	1	XCH A, R1	XCH A, R1
CA	1	XCH A, R2	XCH A, R2
CB	1	XCH A, R3	XCH A, R3
CC	1	XCH A, R4	XCH A, R4
CD	1	XCH A, R5	XCH A, R5
CE	1	XCH A, R6	XCH A, R6
CF	1	XCH A, R7	XCH A, R7
D0	2	POP byte-addr	POP 32H
D1	2	ACALL page-addr	ACALL 0634H
D2	2	SETB bit-addr	SETB 03H
D3	1	SETB C	SETB C
D4	1	DA A	DA A
D5	3	DJNZ byte-addr, rel-offset	DJNZ 23H, 34H
D6	1	XCHD A, @R0	XCHD A, @R0
D7	1	XCHD A, @R1	XCHD A, @R1
D8	2	DJNZ R0, rel-offset	DJNZ R0, 02H
D9	2	DJNZ R1, rel-offset	DJNZ R1, 23H
DA	2	DJNZ R2, rel-offset	DJNZ R2, 43H
DB	2	DJNZ R3, rel-offset	DJNZ R3, 87H
DC	2	DJNZ R4, rel-offset	DJNZ R4, ADH
DD	2	DJNZ R5, rel-offset	DJNZ R5, EAH
DE	2	DJNZ R6, rel-offset	DJNZ R6, 34H
DF	2	DJNZ R7, rel-offset	DJNZ R7, F9H

bit	addr	:	00H	-	FFH	-	8051 bit address
byte	addr	:	00H	-	FFH	-	On-chip 8-bit RAM address
code	addr	:	0000H	-	FFFFH	-	Absolute code address
page	addr	:	0000H	-	07FFH	-	11-bit code address
rel	offset	:	00H	-	FFH	-	8-bit 2's complement addr
data	byte	:	00H	-	FFH	-	Immediate data byte
data	word	:	0000H	-	FFFFH	-	Immediate data word

HEX CODE	NUMBER OF BYTES	SYNTAX	EXAMPLE
E0	1	MOVX A, @DPTR	MOVX A, @DPTR
E1	2	AJMP page-addr	AJMP 0734H
E2	1	MOVX A, @R0	MOVX A, @R0
E3	1	MOVX A, @R1	MOVX A, @R1
E4	1	CLR A	CLR A
E5	2	MOV A, byte-addr	MOV A, 12H
E6	1	MOV A, @R0	MOV A, @R0
E7	1	MOV A, @R1	MOV A, @R1
E8	1	MOV A, R0	MOV A, R0
E9	1	MOV A, R1	MOV A, R1
EA	1	MOV A, R2	MOV A, R2
EB	1	MOV A, R3	MOV A, R3
EC	1	MOV A, R4	MOV A, R4
ED	1	MOV A, R5	MOV A, R5
EE	1	MOV A, R6	MOV A, R6
EF	1	MOV A, R7	MOV A, R7
F0	1	MOVX @DPTR, A	MOVX @DPTR, A
F1	2	ACALL page-addr	ACALL 0703H
F2	1	MOVX @R0, A	MOVX @R0, A
F3	1	MOVX @R1, A	MOVX @R1, A
F4	1	CPL A	CPL A
F5	2	MOV byte-addr, A	MOV 34H, A
F6	1	MOV @R0, A	MOV @R0, A
F7	1	MOV @R1, A	MOV @R1, A
F8	1	MOV R0, A	MOV R0, A
F9	1	MOV R1, A	MOV R1, A
FA	1	MOV R2, A	MOV R2, A
FB	1	MOV R3, A	MOV R3, A
FC	1	MOV R4, A	MOV R4, A
FD	1	MOV R5, A	MOV R5, A
FE	1	MOV R6, A	MOV R6, A
FF	1	MOV R7, A	MOV R7, A

bit	addr	:	00H	-	FFH	-	8051 bit address
byte	addr	:	00H	-	FFH	-	On-chip 8-bit RAM address
code	addr	:	0000H	-	FFFFH	-	Absolute code address
page	addr	:	0000H	-	07FFH	-	11-bit code address
rel	offset	:	00H	-	FFH	-	8-bit 2's complement addr
data	byte	:	00H	-	FFH	-	Immediate data byte
data	word	:	0000H	-	FFFFH	-	Immediate data word

CHAPTER 10

System Errors and Troubleshooting

CHAPTER 10

System Errors and Troubleshooting

10.1. Introduction

This chapter describes the errors detected while operating EB-51 and answers the most common questions related to hardware and software. Please read carefully the error message and follow the indications that appear on the screen to solve the problem.

10.2. Error Description

Error #2 - Reset

This error indicates that the emulated microcontroller could not be properly reset.

Error #3 - Run

The Run Error means that the system is not able to start executing your program.

Error #4 - Halt

This error appears when the system cannot stop the program execution.

Error #5 - Update

The system failed to update register contents.

Error #6 - Hardware

An internal hardware failure occurred.

Error #7 - Hardware

An internal hardware failure occurred.

Error #8 - XTAL

The crystal oscillator of the emulated microcontroller does not operate.

Error #9 - Power Down

The emulated microcontroller is in the Power Down Mode.

Error #10 - Idle

The emulated microcontroller is in the Idle Mode.

Error #11 - Power Down or Idle

The emulated microcontroller is in the Idle or Power Down Mode.

Error #20 - Communications

The PC cannot communicate with EB-51.

Error #21 - Communications

An invalid command has been transmitted to EB-51.

Error #22 - Communications

An invalid command has been transmitted to EB-51.

Error #32 - Communications

EB-51 does not respond to the PC command.

Error #33 - Communications

There are missing bytes in the transmission.

Error #34 - Communications

This is a time-out error.

Error #35 - Communications

An unexpected system status has been received.

Error #36 - Communications

The system identifier is wrong.

Errors #37-#255 - Communications

The PC does not receive any response from EB-51.

Error #256 - Load Error

Wrong File Format.

Error #257 - Load Error

Your program uses banked memory.

Error #300 - User's Entry Error

Address is not from XDATA memory.

Error #301 - User's Entry Error

Address is not from CODE memory.

Error #302 - User's Entry Error

Undefined cycle.

Error #303 - User's Entry Error

Unrecognized input

Error #304 - User's Entry Error

Unrecognized end of line.

Error #305 - User's Entry Error

Trace point does not exist.

Error #306 - User's Entry Error

Path is not correct.

Error #307 - User's Entry Error

Out of range error.

Error #308 - User's Entry Error

The watch value cannot be changed.

Error #309 - User's Entry Error

The trigger event must be set.

Error #310 - User's Entry Error

Invalid passcount entered; Passcount must be 1 to 32767.

Error #311 - User's Entry Error

Invalid memory space identifier.

Error #312 - User's Entry Error

Symbol not found.

Error #313 - User's Entry Error

Search expression not found.

Error #314 - User's Entry Error

Space cannot be dynamically updated.

Error #315 - User's Entry Error

The option cannot be provided while Running.

Error #316 to #450 - User's Entry Error

Error detected while entering a value or symbol.

Error #800 & #801 - System Error

This is an internal error; call CEIBO for technical assistance.

Error #802

Demo version can load only 1K code.

Error #803

Out of memory space.

Error #804

File operation failed.

Error #805 - to #900 - System Error

This is an internal error; call CEIBO for technical assistance.

10.3. Common Questions and Answers

ROMless Operation

1. How does the system work in ROMless mode?

ROMless operation means that the chip selected is 80C31, 80Cxx, etc., and ports 0 and 2 are the bus.

P3.6 and P3.7 can only be used as RD/WR lines.

Port 0 can only be used as AD0-7 address/data bus lines.

Port 2 can only be used as A8-15 address lines.

ROMed Operation

2. How does the system work in ROMed mode?

ROMed operation means that the chip selected is 83Cxx, 87Cxx, 80C51, etc., and ports 0 and 2 are I/O ports.

P3.6 and P3.7 cannot be used.

Port 0 can only be used as I/O port.

Port 2 can only be used as I/O port.

Any MOVX instruction is not supported and must not be used.

Ports

3. I cannot access (read or write) Port 0 and 2.

Select chip type ROMed (like 87C51) and not ROMless (like 80C31).

4. By writing MOV port3,#0 the system does not work.

Write MOV port3,#c0h, because you cannot write to port 3, bits 6 and 7. The other bits of Port 3 do not have limitations.

5. I do not know how to use port 3.6 and 3.7.

These ports can never be used as I/O ports on the EB-51. They can be used only as RD and WR lines.

6. I do not know how to use Port 2.

MOVX @Ri instruction is not supported in regards to Port 2 operation. That means, you cannot use Port 0 as a bus and Port 2 as a Port; work with both as ports or as buses.

Timers

7. How many timer ticks are lost while stopping the emulation and reassuming it?

Up to 50 timer ticks, depending on the timer mode and frequency.

Microcontroller

8. I replaced the microcontroller and the system does not work.

EB-51 may accept many different derivatives. EB-51 uses standard microcontrollers working in emulation mode. This special mode used by the emulator in ROMed mode, outputs the internal buses and it is not documented in the Philips Microcontroller Data Book. Programming the security bits is the operation required by the chip to accept entering in the emulation mode.

If you cannot program the security bits or you are not using a Philips microcontroller, select in the software menu a ROMless type (i.e. 80C32, 80C51FA, etc.) and not ROMed (i.e. 87C51FB, 87C...., etc.).

9. Which security bits should I program, if I replace the microcontroller ?

Use Philips microcontrollers and program lock bit 1 and 2 only. Do not program lock bit 3.

10. How can I program security bits ?

Use Ceibo MP-51 or any other programmer with similar capabilities.

Daughter Board

11. I am trying to use Ceibo 552 daughter board for EB-51 and the system does not work.

Check that the daughter board is properly connected. Use the power switch only in 5V and not 3.3V. Set the frequency to 12MHz and not 24MHz or higher.

System Problems

12. EB-51 works at 5V but not at 3.3V.

You do not have the chip set for 3.3V operation (the set is U1 to U4 and they are "L" devices and not "C").

13. Yesterday I worked with the system and it was OK. Today the ports are not showing any activity.

The software has been invoked without a system connected to the power supply; then the mode has changed from Emulation to Simulation and therefore you are working only with the simulator.

14. I cannot establish a communication between the system and my PC, although the serial port of my computer works with another system as well as the serial cable.

1. A cable from another system may be the problem. Use only the black cable supplied with the system.

2. Set the Power Switch to 5V. Maybe you do not have the 3.3V chip set.

3. Your PC does not support the high baud rate. Try to set the baud rate to low in the Options Menu.

15. The system shows always Error #6 - crystal problems.

Set the crystal jumper to Internal.

Software

16. Why cannot Port 0 and 2 be accessed in the watches window, etc. ?

Select chip type ROMed (like 87C51) and not ROMless (like 80C31).

17. Why some options are grayed out?

Not available yet in the current software version.

18. I am using BSO/Tasking assembler and when I try to enter a symbol in the Watches window, I get Error #311 and the values of the variable are represented as question marks (?????).

This assembler and many others are case sensitive, so enter the variable manually and always in upper case.

19. I cannot open the Module Window.

You did not load a file with DEBUG information. Check again how the file has been generated and if you selected the appropriate vendor in the Load command.

20. I press Ctrl-Break and the program does not halt.

If you try to halt your program with the Halt command (Ctrl-Break), and the Debugger continues to Run, it means that the selected Halt Mechanism Interrupt did not occur. The Debugger will only Halt the next time the selected Halt Mechanism Interrupt occurs. Executing the Reset command (Ctrl-F2) after executing the Halt command and before the Debugger is Halted will force the user program to Halt by forcing the Reset Halt Mechanism.

CHAPTER 11

About the DOS Debugger

CHAPTER 11

About the DOS Debugger

11.1. Introduction

CEB51D is a DOS menu-driven program supplied with EB-51. This chapter includes the basic information you will need to begin using the CEB51D program and to understand the meaning of the different menus and its syntax.

The Debug Capabilities and menus are similar to those explained for the Windows Debugger. This program may also be installed to run under Windows.

11.2. Preparing the Software

1. Do not apply power to the EB-51. It is not necessary for the first stage which mostly explains the software capabilities.

2. Install your EB-51 software in any directory of your computer. Please check that you have 0.5 Mbyte spare disk space after copying the files and /450KByte of free memory.

3. Invoke the EB-51 by typing CEB51D.

11.3. Selecting the Simulation Mode

1. After invoking the program, select the Simulation Mode from the Options Menu and the Architecture Command.

2. The bar on the top of the screen is the global menu and that is indicated by the three dashes on the left upper corner.

3. The upper right corner displays the software status.

4. Hold the CTRL key for a while to display the Control Menu. Hold the ALT key for a while to view the Alternate Menu option.

5. The white double-line frame surrounding the Watches Window indicates that the window is active. That means all the available commands for that window are

accessible by three ways: using the Local Menu, applying a direct entry or utilizing a key combination.

11.4. Adding the Watches

The Local Menu is opened by holding down the ALT key and pressing F10.

A list of commands will then appear on the screen from which you can make your selection.

Lets try to add a watch. Highlight the Watch command and press the Enter key. A window with the message:

"Enter New Watch"

will appear on the screen. Type P1 and press the Enter key. Then, Port 1 will be added to the Watches Window.

11.5. Changing Watches

1. If you want to change the Port value, invoke the Local menu again and select the Change command.

2. A selection may be done either by moving the arrows until the Change command is highlighted or by pressing the C key.

3. Type 55 and press the Enter key. Observe that the Watches Window has an updated value for Port 1.

4. Press the ESC key.

11.6. Hot Keys

1. Every window assumes that a hot key is active if you press a key that is not a command.

2. You can directly add a new watch just by typing the desired variable. That is possible because the Watches window default is the Watch command for the Local menu.

3. Type ACC to add the Accumulator to the Watches window. Press the ESC key.

4. The hot keys or key combinations that permit an immediate access to the commands of the Local menu are displayed by holding down the Control key. Press Ctrl-W and type BYTE 10 to display the contents of the Internal RAM of the microcontroller belonging to address 10H. Press the ESC key.

11.7. Accessing the Global Menu

1. The Global menu commands may be activated by using the function keys or by simultaneously pressing the ALT and the command first letter keys. Press F5 several times to change the Watches window size.

2. Press Alt-V to open the View command. Select CPU and observe the new window added to the screen. The CPU window becomes active as indicated by the white double-line frame.

3. Check the Local, Control and Alternate options of the CPU window. The default is Assemble.

4. Move the cursor to any line and type NOP. Observe how the code has been changed.

5. Press the F6 key to select the next active window.

11.8. Changing the Windows

1. Press Alt-W to select the Window menu, that permits changing the position and size of the active window. Try all the options for both the CPU and Watches windows. The Enter key validates the change while the ESC key cancels it.

2. A window may be resized by pressing the Ctrl-F5 keys and then pressing the Shift key together with the arrows.

3. Similarly, the Ctrl-F5 and then the arrows move the window position.

4. Press Alt-O to access the Options menu and select the Save Options command to save the screen configuration you created in a disk file.

11.9. Loading a File

1. Press Alt-F to activate the File menu.

2. Set the cursor to highlight the Load option and press the Enter key.

3. Press the Enter key again if you do not know which file you want to load, otherwise type in the file name. Select the TESTS.ABS Sample file. This file includes code and symbols.

4. You may also load a Hex file without symbol information, but in that case the symbolic debugger will not function.

5. After successfully loading the TESTS.ABS file, the CPU window will be opened with the updated code. The Watch window will be erased. You can navigate through the disassembled code in the CPU window by using the arrow keys, PgUp and PgDn.

11.10. Debugging the Program

1. Press F6 and select the Watches window.

2. Type P1 to add Port 1 to the Watches window.

3. Select the View Menu and the Modules Command to choose the TESTS Module.

4. Move and resize the three windows according to your convenience. Please note that originally the Modules window is hidden by the CPU Window, but is still present behind the source code, and reappears if you press the F6 key to select it.

5. Select the RUN Menu and execute a Program Reset. That can be done directly by pressing Ctrl-F2.

6. Pass the control to the CPU window and execute a few assembly steps. These are available from the Alternate menu and the command name is Instruction Trace. Press the Alt-F7 keys several times.

7. Execute the program. Press the F9 key.

8. Halt the program execution by pressing Ctrl-Break.

9. Display the executed instructions from the View Menu, using the Trace Buffer command.

10. Set a Breakpoint at line #33 by moving the cursor in the TESTS window to that line and pressing the F2 key. Execute the program by pressing the F9 key.

11. Execute a high-level language step by pressing the F8 key. Repeat that operation several times.

12. Continue the high-level-language stepping by pressing F7. Note that the Modules are changed in accordance with the actual program counter value.

13. Press Alt-X to leave the Debugger.

11.11. System Menu

The System menu mainly includes commands affecting the entire desktop screen. It can be accessed by pressing the Alt-F10 keys.

File pick dialogs allow typing the item searched and moving the highlight bar to the first item matching the already typed in letters. This allows item matching with minimum time and key strokes.

The available commands are: Repaint Desktop, Restore Standard and About.

Repaint Desktop

The Repaint Desktop command simply displays again the entire desktop screen, updating all currently active windows.

Restore Standard

The Restore Standard command restores the window layout to its original state. This is useful in case too many windows begin to accumulate on the desktop screen.

About

The About command displays the copyright screen of the CEB51D Debugger. This is useful for version identification of the software.

11.12. File Menu

The File menu options deal with operations external to CEB51D Debugger, such as loading programs for debugging, saving code memory, and returning to DOS.

File pick dialogs allow typing the item searched and moving the highlight bar to the first item matching the already typed in letters. This allows item matching with minimum time and key strokes.

The available commands are: Load, Save, Get Info, Symbols Load, Program and Quit.

Load

The Load command loads a program for debugging from a disk. User is prompted for the name of the program to be loaded, a wildcard specification can be entered for selecting the file from the list of matching files.

No file name entry will be interpreted as *.* wild card entry.

The supported file formats are: Intel hex format, Intel OMF51 object format and Franklin/Keil extended OMF object format.

Save

The Save command stores the code memory contents onto a disk file in Intel hex format. The user is prompted for the code memory address range to be saved.

Get Info

The Get Info command displays a window showing the current state of the program being debugged.

This includes the program name, where and why your program was stopped, memory used by DOS, emulator system version information and the current date and time.

Symbols Load

The Symbols Load command allows the loading of a new symbol table to replace the current one.

Quit

The Quit command terminates CEB51D debugging session. The hot key Alt-X can also be used to return to DOS from any point. Any open files are closed and occupied memory is released.

DOS Shell

The DOS shell command allows a temporary exit from the Debugger to carry out any DOS operation. You may exit to DOS shell at any time, even while running the program. When exiting to DOS shell the Debugger memory will be swapped to a disk file, consuming in all about 10 KByte memory space and 10 KByte disk space. This is extremely useful for re-compiling your program when a Bug is found. Upon returning to the Debugger from DOS shell after compiling the program debugged, you will be prompted of the program change and the Debugger will automatically re-load the newly compiled program.

By using this command you can restore files and parameters

1. Exit to DOS Shell

2. Recompile your project

3. Type Exit to return to the debugger

The debugger will detect automatically that the project has been recompiled, according to the change in the absolute object file, and will ask whether to reload the new recompiled project. If yes, the new object file will be loaded updating all the environment to the new project. All breakpoints and watches will be remembered and restored, although breakpoints are restored according to absolute addresses and not line numbers. Then a Reset command will be executed with start up skip. Execute again the Reset command and you will be ready to continue the debug session.

11.13 View Menu

The View menu commands open windows that display different aspects of the program being debugged.

The available commands are: Watches, Variables, Module, CPU, Dump, Registers, Trace Buffer and File.

Watches

The Watches command opens a Watches window showing the value of variables specified using either the Data add watch main menu command or the Watch local menu command in a Module window.

Variables

The Variables command opens a Variables window displaying a list of the program global (or public) and local symbols, and their values.

The different variable types are as explained for the Windows Debugger.

Module

The Module command opens a Module window showing the list file of the selected module.

A module to be viewed can be "picked" from a list of the current program available modules, only modules which have a corresponding list file will appear in this pick list. Module pick dialogs allow typing the item searched and moving the highlight bar to the first item matching the already typed in letters. This allows item matching with minimum time and key strokes.

Use the Module's List File of the Option command for setting up the list file name associated with each module of the program you wish to debug. Use the Path for Source List of the Option command to set a path list in which the list files are to be found.

This command will be disabled if no debug information has been loaded. F3 is the hot key for this command.

If a Module window is already shown on the screen, the selected module will be displayed on that window, and it will become the active window.

The Module window title indicates the module name currently being viewed.

Make sure that the list files will be in accordance with the program being debugged, otherwise the cursor might not always show up on the correct source code line.

The Goto command is included in the local menu of the Module window. This allows direct display jump to any code memory location, either an absolute address, source line number or a procedure/function name may be specified.

When a list file cannot be opened for the specified address, the CPU window will be automatically opened at the specified address location.

CPU

The CPU command opens a CPU window displaying the disassembled instructions of your program.

An instruction may be displayed with symbol information, and mixed with source code lines.

You may also patch-up code using the built-in assembler.

If a CPU window is already displayed on the screen, it will become the active window upon invoking this command.

The CPU window indicates the last item of code that was selected in the View CPU command.

It allows a closer look to be taken at the machine code which corresponds to one of the source-level views.

Dump

The Dump command opens a Dump window where a specified area of memory is displayed. The data can be viewed as raw hex bytes with their corresponding ASCII representation. Read and Write commands are included in the local menu of every Dump window. This allows saving onto a disk file or loading from a disk file (in Intel Hex format) any portion of the dumped memory space (including Bit and RBit space).

The different memory spaces are: Code, Data, Byte, Bit, RByte, RBit and Registers.

Code: The Dump Code command opens a Dump window where the code memory space is displayed. The code is viewed as raw hex bytes with their corresponding ASCII representation. You may change or fill any portion of the internally mapped code space, with any given values.

Byte: The Dump Byte command opens a Dump window where the Microcontroller on-chip ram byte memory space is displayed. The data is viewed as raw hex bytes with their corresponding ASCII representation. You may change or fill any portion of the on-chip ram byte space, with any given values.

Data: The Dump Data command opens a Dump window where the external data space is displayed. This data is accessed by the MOVX instructions.

Bit: The Dump Bit command opens a Dump window where the Microcontroller on-chip ram bit memory space is displayed. This is a 16 byte area which starts at address location: 20H and the 128 bits in this area can be directly addressed. The data is viewed as "1" or "0" bit information. You may change or fill any portion of the on-chip ram bit space, with any given values.

RByte: The Dump RByte command opens a Dump window where the Microcontroller SFR byte memory space is displayed. The data is viewed as raw hex bytes with their corresponding ASCII representation. You may change or fill any portion of the SFR byte space, with any given values.

Rbit: The Dump Rbit command opens a Dump window where the Microcontroller SFR bit memory space is displayed. The data is viewed as "1" or "0" bit information. You may change or fill any portion of the SFR bit space, with any given values.

Registers

The Registers command opens a Registers window where the flags and current CPU registers state appear.

Trace Buffer

The Trace buffer menu allows the current Trace window to be opened, as well as viewing the trace status. The Trace functions are enabled in simulation modes only.

The following options are available: Trace Dump and Trace Status. The local menu (ALT F10) selects the trace display mode.

File

The File command opens a File window in which any specified ASCII file contents may be displayed. You will be prompted for the file name to be viewed, a wildcard specification can be entered for selecting the file from the list of matching files. No file name entry will be interpreted as *.* wildcard entry.

Performance Analyzer

The Performance analyzer window allows the statistics information to be viewed on the program execution. The statistics are calculated from the current Trace buffer contents. This is displayed as the number of execution cycles per procedure and per module, together with the percentage from the total execution time stored in the Trace buffer.

11.14. Run Menu

The Run menu commands execute the program being debugged. The following options are available: Run, Execute Forever, Go to Cursor, Trace Into, Execute to, Step Over, Execute Forever, Animate, Instruction Trace and Program Reset.

Run

The Run command executes the program continuously until either the program is halted with the Halt key, or a breakpoint is reached. The functions are similar to those explained for the Windows Debugger.

The F9 key is the hot key that executes this command.

11.15. Breakpoints Menu

The Breakpoints menu commands let breakpoints be set and cleared.

The available commands are similar to those explained for the Windows Debugger: Toggle, Expression True Global, Hardware Breakpoint and Delete All.

11.16. Data Menu

The Data menu commands permit the examination of variables and symbols in the program. The functions are similar to those explained for the Windows Debugger.

The following commands are available: Inspect, Evaluate and Add Watch.

11.17. Options Menu

The Options menu allows adjustment of some options that have a global effect on the conduct of the CEIBO Debugger, and the remote emulator system.

The following are the available options: Environment, Path for source List, Module's List File, Communication Port, Communication Baud, Architecture, Mode, Halt Mechanism, Interrupt Shared, Save Options and Restore. These are similar to those explained for the Windows Debugger.

11.18. Window Menu

The Window menu commands allow various operations on the currently open windows with the following commands: Zoom, Size/move, Next and Close.

Use the View menu commands to open new windows. The F6 key makes the next window to be the active window. The F5 key zooms the current active window in/out.

Zoom

The Zoom command makes the currently active window occupy the entire screen area. This command functions as a toggle, so that an already zoomed window will return to its original size and screen position upon issuing the command.

The F5 key zooms the current window.

Next

The Next window command changes the windows order on the screen.

Invoking this command will cause the next window in the sequence of all open windows to be made the currently active window. The active window can always be identified as the window with the double border frame.

The F6 key moves to the next window.

Size/Move

The Size/Move command allows the position and size of the current window to be changed.

Once the command has been issued, use the cursor keys to move the window and Shift plus the cursor keys to change its size.

Ctrl-F5 is the hot key for this command.

Close

The Close command removes the current active window. To create new windows use the View command.

The Alt-F3 key is the hot key that executes this command.

11.19. Help Menu

The Help menu commands open a help window for whichever subject will be selected from the menu. This menu offers the following options: Index, Previous topic and Help on Help.

Index

The Index command displays a list of help topics. Not all the help contexts are listed, only the useful subjects and starting points.

Shift-F1 is the hot key that executes this command.

Previous Topic

The Previous Topic command displays the last help screen viewed.

Alt-F1 is the hot key that executes this command.

Help on Help

The Help on help command displays a help screen describing how to use the CEB51D Debugger help system.

11.20. Command Line Options

You may get the command line options by invoking the debugger as follows:

CEB51D>CEB51D /help
or:
CEB51D>CEB51D /?

The command line syntax is:

CEB51D [filename] [options]

where the text is not case sensitive.

By specifying a filename, the debugger will automatically load your program.

The /Scan option allows automatic detection of the host PC COM port to which the remote emulator system is connected. Use this switch whenever changing the Com port

connection. CEB51D will automatically invoke Scan mode even when not instructed so, whenever running for the first time from a new project directory.

The /IRQ option enables the use of interrupt for serial communications.

The /Mono option allows use of the debugger with monochrome screens.

The /L option will force the program counter to zero in the CPU window after loading a program. If you omit this option, the program counter will point to the main procedure. If the program does not run correctly the Debugger may be left running after such an operation (the first line of Main was never reached). You can then halt the program using the Halt menu command (Ctrl-Brk). Afterwards the Debugger will not attempt automatic start-up skip again, until a new program is loaded.

11.21. Predefined Names

EB-51 accepts the following symbols that you may invoke directly while adding a watch, setting a breakpoint or assembling an instruction. The complete list of predefined Ports and Register names may be found in the Microcontroller Data Book.

Ports

P0, P1, P3, etc.

Port Bits

P0.0 to P0.2, P1.0 to P1.7, etc.

Registers

ACC, B, etc.

Register Bits

ACC.0 to ACC.7, B.0 to B.7, etc.

11.22. Absolute References

You may invoke a parameter without any symbolic reference just by specifying its type.

Examples:

BYTE 10H

BIT 5H

CODE 0H

RBYTE 90H

RBIT 90H

11.23. Preparing Your Files

The Symbolic Debugger requires your files to be prepared with debugging information. Therefore, files must be compiled with the Debug and List options.

The Debug Option of your compiler generates symbol information such as Line Numbers, Global (or Public) and Local references, Labels and others. The EB-51 software recognizes these symbols.

The List Option creates an ASCII file that contains the source written by the user and references to Line Numbers.

The normal way to prepare your program for debugging is as follows:

1. Use an Editor to write your source code.

2. Compile the sources with an Assembler or high-level language compiler, using the Debug and List options.

3. Link and locate your object files with the Intel RL51 program or a similar one that generates an Intel compatible format.

The EB-51 Disk includes a sample program to clarify the above:

1. TEST.LST is the listing of a source program written in PLM that has been generated by Intel PLM-51.

2. TINIT.LST is another listing file generated by the same software.

3. TEST.ABS is the output file that linked three modules and was created by Intel RL51 Relocator and Linker Program.

11.24. Loading a Program

Use the LOAD command to load your code and symbol information. First try the supplied TEST.ABS program, that was generated by Intel RL51 program.

Use the View Menu to display the Modules, Globals, Locals and Lines of the modules.

11.25. Symbol Syntax

Symbols are valid arguments instead of numeric values or absolute addresses and are specified as follows:

Predefined Symbols

Predefined symbols are register and port names, that are recognized even if you did not load a program into the system.

Examples:

P1

P2.0

IE

ACC

TCON.3

Publics and Global Symbols

Public or Global symbols are those declared in your program as public variables and assembled or compiled using the Debug option.

The dot <.> is used to inform the software that a symbol follows and not a numeric value.

A dot followed by a name defined in your program is used to invoke a public or local symbol, although in the Watches and CPU windows the dot may be omitted.

Examples:

.DELAY_CNT

.INIT

Modules

A Module is the name assigned to a portion of your program.

A module may be invoked by the module name preceded by a colon (:).

Examples:

:TEST

:TINIT

Procedures

A Procedure is a subroutine of your program and may be public or local depending on what was defined for in the software.

Local procedures must be preceded by a colon (:), the module name and the procedure name separated by a dot.

Global or public procedures must be preceded by a dot and the procedure name.

Examples:

.INIT

:TEST.ROL_P1

Module Local Symbols

Module local symbols are those declared in your program as local variables belonging to a module.

Local symbols must be preceded by a colon (:), the module name and a dot.

Examples:

:TEST.REG_P1

:TEST.CPL_P1

Procedure Local Symbols

Procedure local symbols are those declared in your program as local variables belonging to a module.

These symbols must be preceded by a colon (:) the module name, a dot, the procedure name and an additional dot.

Examples:

:TEST.ROR_P1.ROT_NO

:TEST.ROR_P1.ROT_CNT

Line Numbers

Line numbers are treated as local symbols and are preceded by a colon (:) and the module name, followed by the numeric symbol #, and the specific line number.

Examples:

:TEST#12

:TEST#33

Length

The Length parameter allows invoking contiguous parameters starting from one specified symbol.

Examples:

RBIT 90H LEN 8

:TEST.REG_P1 LEN 4

Index

Z