



EMUL51TM PC Windows

User Guide

This page should be blank.

EMUL51™ PC Windows

User Guide

Copyright ©

ICE Technology

Tel: 650.375.0409 - 800.686.6428

Fax: 650.375.8666

E-Mail: sales@icetech.com

URL: <http://www.icetech.com>

All rights reserved worldwide.

Edition 1:

Development Team

Documentation

Nils Johansson

Randy Devol

Peter Zou

Steve Ehret

Warranty Information

The EMUL51™-PC Windows Emulator board, Trace board, Pods, Emulator Cable, and LanICE hardware are sold with a one-year warranty starting from the date of purchase. Defective components under warranty will either be repaired or replaced at Nohau's discretion.

Pods that use a bond-out processor are also warranted for one year from the date of purchase except for the processor. The bond-out processor will be replaced once if Nohau determines that the failure in the bond-out processor was not due to user's actions. This replacement limit does not apply to the rest of the pod.

Each optional adapter, cable, and extender is sold with a 90-day warranty, except that it may be subject to repair charges if damage was caused by the user's actions.

The EMUL51™-PC Windows Emulation software is sold with no warranty, but upgrades will be distributed to all customers up to one year from the date of purchase.

ICE Technology makes no other warranties, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. In no event will ICE Technology be liable for consequential damages. Third-party software sold by ICE Technology carries the manufacturer's warranty.

Table of Contents

Introducing EMUL51™-PC Windows	1
How to use this manual	1
If you are new to emulators of any kind,	
If you have used emulators with other microprocessors,	
If you are familiar with emulators, MS Windows, and the chip,	
Manual Conventions	2
Quick Installation Instructions	5
System Requirements	5
Quick Setup Instructions	5
Installing the Emulator	
Installing the Pod	
Installing the Software	
Installing the Trace Board (If used)	
Quick Start Instructions	8
To load and execute a program:	
To set a breakpoint either:	
To make a breakpoint inactive either:	
To delete a breakpoint either:	
To use the in-line assembler to change the program loaded:	
To change a RAM value:	
Chapter 1: Emulator Board	11
Emulator Detailed Installation Instructions	11
Materials and Supplies	
System Requirements	
Setting the Jumpers	
Standard Emulator Board Headers	13
32k Emulator	
One-Memory E32 Board	
Four-Memory E32 Board	

128k Configurations	
Setting the Emulator Address	18
Addressing Examples	
Mapping Emulation Memory	20
Files Provided	21
Memory Bank Switching	22
The IAR Banked Memory Model	22
IAR ICC8051 Compiler Options For bankswitching:	
IAR XLINK Linker Options For bank-switching:	
Example 1:	
Example 2:	
Example 3:	
Bank switching with the Keil/Franklin Banked Linker	26
Keil / Franklin BL51 Linker Options	
Example 1:	
Example 2:	
Example 3:	
Example 4:	
Hardware Bank-Switching Examples	28
Configuring EMUL51 for Bank switching	
Example 1:	
Example 2:	
Example 3:	
Tracing and bank switching	32
Bank switching and Breakpoints	
EMUL51-PC/E128-BSW Bank-switch Emulator	35
EMUL51-PC/E256-BSW Bank switch Emulator	37
EMUL51-PC/EA256-256k	43
Memory Configurations	
Description of Headers:	
Bank switch / Memory Modes:	
(BM) Headers	
Banksize = 32k	
Bank size = 64k	
Banksize = 48k	

Reprogramming an EPROM for another type:	
EMUL51-PC/EA768-768k	57
Memory Configurations	
Description of Headers:	
Bank switch / memory modes:	
(BM) jumpers	
Bank size = 32k	
Bank size = 64k	
Reprogramming an EPROM for another type:	
Chapter 2: Software User Interface	75
Detailed Installation Instructions	75
Configuring the Software	76
Projects	
Setting the Paths ..	
Mapping memory	
Emulator Hardware Configuration	
Miscellaneous Configuration	
Window Colors	
Trace Config Menu	
Performance Analysis	
Tool Bar	90
Dialog Boxes	91
Child Windows	91
Register Windows	
SpecialRegs Window	
Data Windows	
Program Windows	
Source Windows	
Trace Window	
Other Windows	
Help Line	
Menus	100
File Menu	
View/Edit Menu	
Run Menu	

Breakpoints Menu
 Config Menu
 Program Menu
 Source Menu
 Data Menu
 Register Menu
 Trace Menu
 Stack Menu
 Watch Menu
 Window Menu
 Help Menu

Chapter 3: Trace Board	113
Introduction	
Detailed Installation Instructions	113
Power Requirements	
I/O Address	113
Addressing Examples	
Installing the Standard Trace Board	116
Factory Settings	
Other Miscellaneous Standard Trace Jumpers	
4K Trace	
16K Trace	
Standard Trace Board Configuration	119
Triggering and Filtering	
Record Filtering	
Advanced Trace Board	127
Introduction	
Detailed Installation Instructions	127
Power Requirements	
I/O Address	127
Addressing Examples	
Installing the Advanced Trace Board	130
Factory Settings	
Other Miscellaneous Advanced Trace Jumpers	130

Advanced Trace Board Configuration	131
Advanced Trace Board "Program Fields"	132
Trig	
Delay	
Loop Count	
Break Emulator	
Record	
Filter Delay	
Time Stamp Prescaler	
Time Stamp Overflow	
State Flags S0 - S5	137
Boolean expressions	
Cycle Count Enable S5=	
POD Signal "ANB" S4=	
Loop counter condition S3=	
S2=, S1=, S0=	
Conditions A through H	
Condition Fields	
Using the Sx Flags	142
Using SET and CLEAR with the Sx functions:	
Introduction to Tracing	145
Triggers and Hardware breakpoints	
Trace Speed Bar	
Trace Window	147
Instruction Size	
Trace Menu	148
Toggle trace speed bar!	
Go to beginning of trace buffer	
Go to end of trace buffer	
Find Frame number ..	
Search Address ..	
Search Next Address	
Search Previous Address	
Find Trig point	
Save trace as text ..	
Save trace image to file!	

Show trace image from file	
Show misc data	
Show ports	
Show P3.6 and P3.7	
Show all frames	
Show timestamp	
Absolute cycle	
Relative cycle	
Absolute time	
Relative time	
Synchronize program window	
Index	157

Introducing EMUL51™-PC Windows

The EMUL51™-PC Windows is a personal computer-based, emulator for the the 8051 8-bit family of microcontrollers. The EMUL51™-PC Windows consists of emulator "plug-in" board, a five-foot-long (1.5 m) twisted-pair ribbon cable, various pod boards and an optional trace board.

The EMUL51™-PC Windows software is a *Microsoft Windows 3.x/NT* application.¹ It follows the *MS Windows Multiple Document Interface Standard*. That means that it has the same look and feel as applications produced by Microsoft and others for *MS Windows*.

The EMUL51™-PC Windows user interface is consistent with most other *MS Windows* applications and includes dynamically changing menus, moveable and scrollable "child" windows, function key shortcuts for menu items, and context-sensitive help. Anyone familiar with *MS Windows* applications will be able to use EMUL51™-PC Windows with little or no other assistance.

The EMUL51™-PC Windows hardware is modular. The software user interface implements an effective high-level debugger. It has support for local variables, C typedefs, and C structures. The Trace board options add tracing, triggering and filtering of executed instructions, as well as data transfers.

How to use this manual

This manual was written with different kinds of users in mind. All users should have *MS Windows* installed and have learned the skills taught in the Basic Skills chapter of the *Microsoft Windows User's Guide*. Many of the EMUL51™-PC Windows features are designed

1. There are two DOS user interfaces available: EMUL51 and Chip View, a Borland keypress compatible user interface. There are also interfaces for EMUL51-PC available from Franklin/Keil, and PLC.

around the features of the supported chips. Being familiar with the chip is a prerequisite to understanding how to use the emulator productively.

Note: *This manual does not include a section describing the many pods available for EMUL51™-PC Windows. For information about those pods, their configuration headers, and their special features, please refer to the manual describing the DOS user interface. If you did not receive one along with this manual, please call Nohau customer support.*

If you are new to emulators of any kind,

read the manual completely, including the reference chapters. You may skip the sections that describe pods and accessories you do not have.

If you have used emulators with other microprocessors,

but are not familiar with the chip being emulated, you are strongly encouraged to review the features of the chip you have, then thoroughly read the section in the EMUL51™-PC Windows manual that describes your pod on the applicable pod board before running the emulator.

If you are familiar with emulators, MS Windows, and the chip,

read the Hardware and Software overviews, then begin using EMUL51™-PC Windows, referring to on-line help as needed. After a few days of use, skimming the Reference chapters may highlight useful features.

Manual Conventions

Type the words in double quotes exactly as shown, but without the quotation marks, except for the <Enter>, <Ctrl>, <Tab>, and <Alt> keys. Use the <Alt> and <Ctrl> keys like shift keys.

Hold them down while you press the key that follows them in the text. For example, if the text instructs you to type <Alt>F, press and hold down the <Alt> key, then press the F key. Window names and labels that appear on the screen are printed using the **screen** font to set them apart from the rest of the text.

Notes and hints are printed in italics, and

Warning: Warnings are boxed to set them apart from the rest of the manual text. Pay careful attention to them.

Quick Installation Instructions

System Requirements

The EMUL51™-PC Windows requires a personal computer with at least one free ISA (or EISA) -bus slot. The Trace board, if present, will require one additional full-length, 16- or 8- bit slot near the emulator board. The PC must also have at least 2 megabytes of RAM (8 megabytes for Windows '95), a CPU that is either 80386, 80486, or Pentium-compatible, a hard disk with at least one megabyte of unused space, and *Microsoft Windows 3.1*, *3.11* or *Windows '95*, *Windows NT* or *OS/2 2.1* (or higher) installed. A mouse is not required, but is strongly recommended.

Quick Setup Instructions

The hardware and software are designed to be easily installed and quickly running on most personal computer systems. Users can normally begin using their emulator after following these initial steps. However, if you are new to personal computers, if you are unsure about what to do after reading the quick installation instructions, or if your emulator does not work after you follow these instructions, follow the more detailed steps for installing and configuring each board and the software as outlined in their respective chapters.

Warning: Always turn on the PC before applying power to the target. Always turn off the target power before turning off the PC power.

Installing the Emulator

Installing the emulator board is much like installing most other AT-style boards:

1. Turn off the power.
2. Remove the PC cover.
3. Remove the slot cover (if present) for an available 8-bit slot.
4. Insert the emulator board into the slot and use a screw to secure the emulator.
5. Unless you will be installing a trace board, you may now close the cover, and attach the ribbon cable to the emulator and the pod. For instructions on installing a trace board, see “Installing the Trace Board” on page 10.

Installing the Pod

With the PC power off, line up the ribbon-cable connector key with the keyed slot on the emulator board, and insert. There is no lock, but friction will secure the cable adequately. On the other end of the cable, open the jaws on the pod connector, line up the key with the keyed slot on the pod board, and insert the ribbon cable connector into the slot firmly, pressing until the jaws on the pod close (or nearly close). Remove any antistatic foam from the pins on the bottom of the pod. Before attaching the pod to your target, it is a good idea to power up the PC, install the software, and follow the procedures described below in “Quick Start Instructions.”

Installing the Software

To install this software under *MS Windows 3.x*, run SETUP.EXE by typing "**WIN A:SETUP**" at the DOS prompt or, from within *MS Windows*, by selecting the **RUN** item in the **Program Manager File** menu and typing "**A:SETUP**" as the file to run. If using *Windows '95*, open the **My Computer** facility, select the floppy drive containing the installation diskette, then double-click on the **Setup** icon.

After SETUP.EXE is started, a dialog box will ask for a directory for the EMUL51™-PC Windows software. Either accept the suggested directory or type a different one. SETUP will create the directories as needed, decompress and copy the files from the floppy to the hard disk directory specified and change the paths in the ".ini" file. When installed, there will be a **Nohau** program group containing the EMUL51-PC WindowsEMUL51 icon. Double-clicking on this icon will start the EMUL51™-PC Windows application.

Installing the Trace Board (If used)

1. Turn off the power.
2. Remove the PC cover.
3. Remove the slot cover (if present) for an available 8-bit slot.
4. Insert the trace board into the an 8 or 16 bit slot near the emulator board. Once you are sure the trace board is fully inserted, use a screw to secure it.
5. Connect the two ribbon cables attached to the Trace Board to the emulator board, making sure that the pins are fully inserted into the connectors, there are no exposed pins, no connector is offset relative to the pins either vertically or horizontally, there are no

twists in either cable, and the cables do not cross. The most common error is to insert only one row of pins into the connector. This could damage either of the boards.

6. Once the ribbon cables are attached, close the PC cover, and install the pod.

Quick Start Instructions

This section describes how to quickly start using EMUL51™-PC Windows to debug an existing program or target board once the EMUL51-PC Windows hardware is installed and the user interface software is running.

To load and execute a program:

1. Select **Load code ..** from the **File** menu and identify the “absolute” file to load by using the dialog box.
2. Select **Reset Chip!** from the **Run** menu.
3. Click on the **GO** button in the tool bar.

To set a breakpoint either:

1. Click twice on the desired instruction in any **Program** window, or
2. Double-click (not the same) on the desired instruction in any **Source** window.

To make a breakpoint inactive either:

1. Click on the desired breakpoint in any **Program** or **Source** window, then press F2, or
2. Select **Setup ..** from the **Breakpoints** menu, click on the breakpoint, click on the **Toggle** button, or

3. Highlight (click once on) the breakpoint and select **Toggle Breakpoint** from the **Program** or **Source** menu. or
4. Highlight (click once on) the breakpoint and select **Toggle** from the **Breakpoints** menu.

To delete a breakpoint either:

1. Select **Setup ..** from the **Breakpoints** menu, click on the breakpoint, click on the **Delete** button, or
2. Select **Delete All** from the **Breakpoints** menu.

To use the in-line assembler to change the program loaded:

1. Scroll a **Program** window until it shows the address to be changed.
2. Highlight the instruction to be changed with the cursor or arrow keys.
3. Type the desired mnemonic (this will open an **Enter new instruction:** dialog box) and hit <Enter>.

To change a RAM value:

1. Scroll a **Data** window until it shows the address to be changed or hit <Ctrl>A and type in the desired address.
2. Highlight the instruction to be changed with the cursor or arrow keys.
3. Type the desired value (this will open an **Enter data** dialog box) and hit <Enter>.

Chapter 1: Emulator Board

Emulator Detailed Installation Instructions

Materials and Supplies

Check that the following items are present:

- a. One pod board
- b. One emulator board
- c. One ribbon cable
- d. Floppy disk(s)
- e. One trace board (optional)
- f. Other optional items

System Requirements

If your PC can run Windows 3.1 (or a later version) or OS/2 version 2.1 or later, then it has the basic resources necessary for running EMUL51™-PC Windows. For example, for Windows 3.1, a minimum memory size of 2 Megabytes is required and 4 or more megabytes are recommended. The same is true for EMUL51™-PC Windows.

The emulator board requires about 1.7A from the PC's 5V power supply, and the trace board typically requires 1.3A. Before proceeding, check that the PC's 5V power supply is sufficient to deliver the necessary current. If it can't, a larger power supply will have to be installed. Your computer dealer can give you information on where to purchase one.

Setting the Jumpers

Several sets of jumper pins are provided on the boards for configuring the on-board functions. The connection method is to slide a jumper block over two adjacent pins.

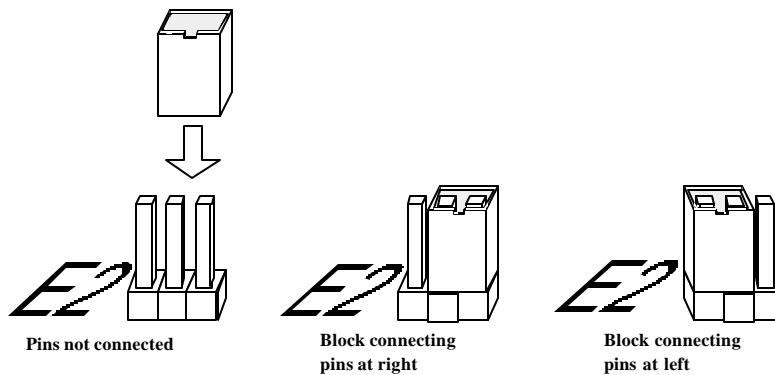


Figure 1: Typical Set of Three Jumper Pins

Jumpers for the emulator boards are described in this section of the manual. For information on the trace board jumpers, please refer to the Trace chapter. For information about pod board headers and jumpers, please refer to the Pods chapter in the EMUL51 DOS manual.

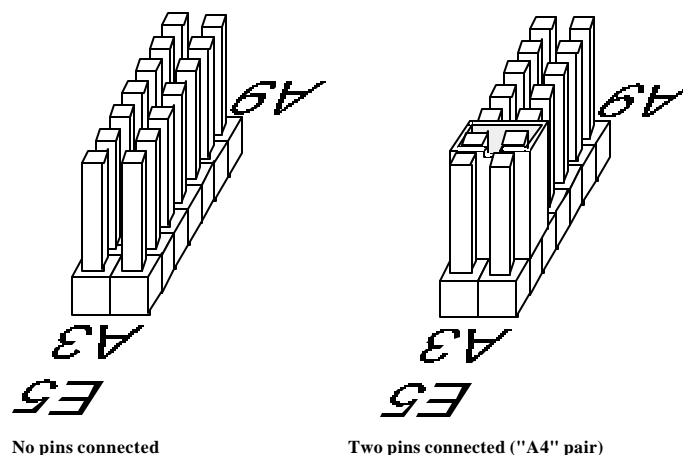


Figure 2: Typical Set of Paired Jumper Pins

Jumpers have been preset at the factory prior to shipment. On the boards you receive it is important to compare the jumper configurations against the configurations described in this manual. If a jumper is installed other than shown, refer to the “Jumper Descriptions” information before changing its position. Jumpers may be installed for operating characteristics required at the time of order.

Standard Emulator Board Headers

On the emulator board the locations of jumper pins are designated on the board artwork as E1, E2, E3, E4 and E5. (See specified rectangles in Figure 3.)

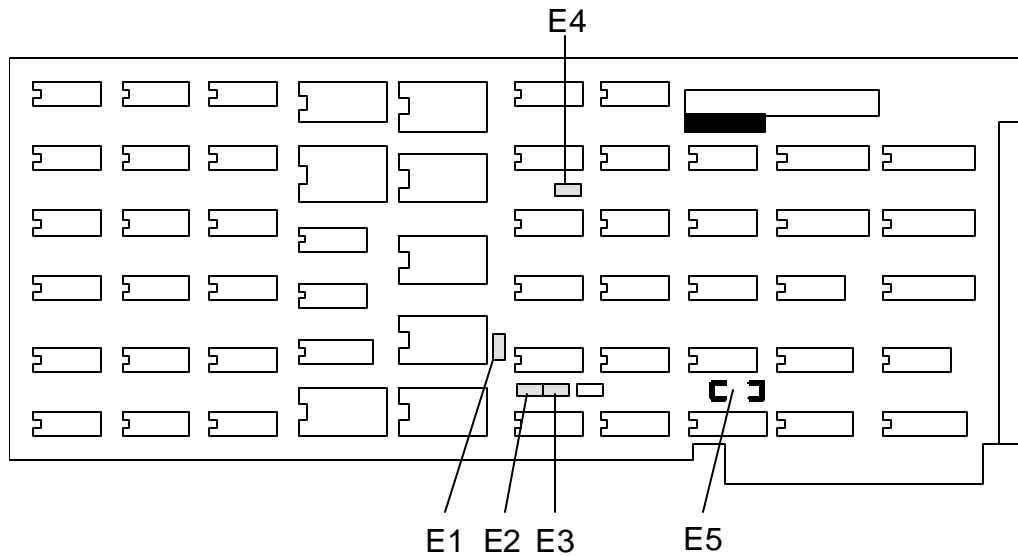


Figure 3: Emulator Header Positions

32k Emulator

When shipped from the factory, the emulator board is normally configured for 32k RAM and with I/O Port Address set for 110H. Figure 4 and Figure 5 show details of the jumper connections for this configuration.

The 32k emulator board can be manufactured in either of two ways, depending on memory availability at the time of manufacture. The variations are: one large memory chip installed in the bottom socket, or four small memory chips installed in the four sockets.

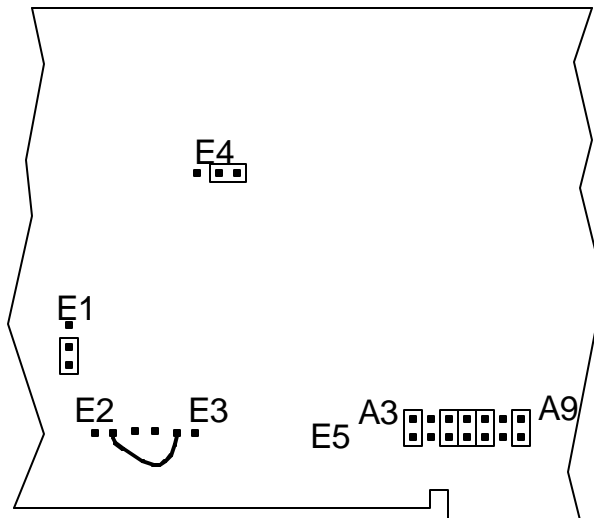


Figure 4: Emulator Board with ONE CHIP 32k installed, Address 110H.
Code and XDATA (if mapped to emulator) always overlaid.

One-Memory E32 Board

Figure 4 shows the jumper configuration if there is a single memory in the bottom socket. Note the wire wrap connection between E2 and E3. The memory is a 32k by 8 static RAM, of the type 62256, 43256, 55256, or similar number.

Four-Memory E32 Board

Figure 5 shows the jumper configuration if there are four small memories in the sockets. The memories are 8k by 8 of the type 6264 or similar. (If there are four large memories in the sockets, the board is not a 32k emulator board.)

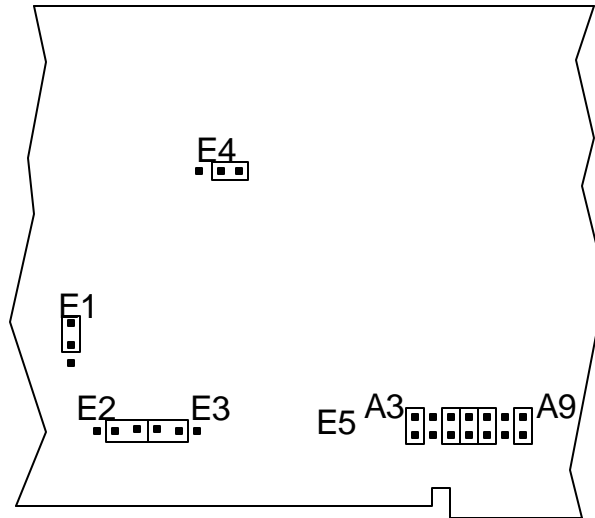


Figure 5: Emulator Board with FOUR CHIP 32k RAM, Address 110H.
Code and XDATA (if mapped to emulator) always overlaid.

128k Configurations

If four 32k by 8 RAM chips are inserted in sockets U25, U26, U27 and U28, then the emulator board can be configured either as shown in Figure 6 or as in Figure 7. The differences between these configurations depend on whether the 64k code and 64k XDATA are in separate areas or are overlaid. (Note: For bankswitch configurations, see “Memory Bank Switching” on page22.)

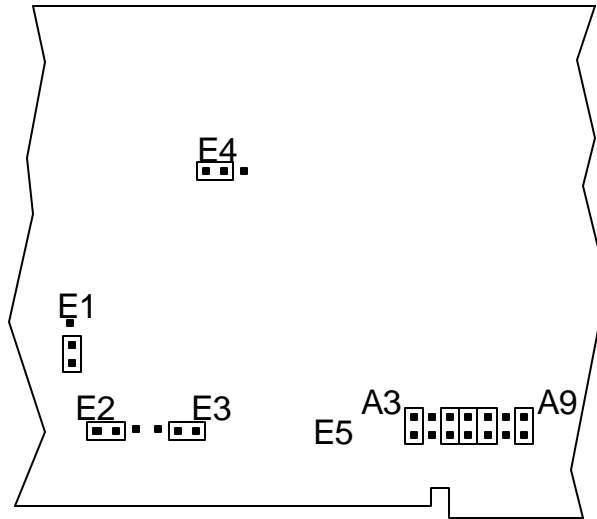


Figure 6: Emulator Board with 128kb installed, 64k Code and 64k XDATA in SEPARATE areas, Address 110H.

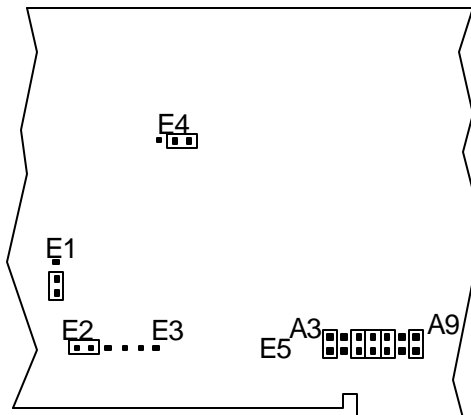


Figure 7: Emulator Board with 128k installed, 64k Code and 64k XDATA in OVERLAID areas, Address 110H, 32K MUST BE SELECTED under **Config Emulator hardware..**

Setting the Emulator Address

The emulator address jumpers have been factory preset to 110 (HEX) and the trace board jumpers have been preset to 100 (HEX). These address settings are set for a typical system. The table below shows how a typical PC uses I/O address locations. If your system is presently using locations 110 and 100 (HEX), you will need to find alternate addresses, set the jumpers on each board accordingly, and set the software configuration as well.

Hex Location	Used by
000 - 0FF	Operating system
1F0 - 1F8	Fixed Disk
200 - 207	Game Adaptor
210 - 213	Expansion Unit
278 - 27F	Parallel Printer Port 2
2F8 - 2FF	Secondary Asynchronous Printer Adaptor
300 - 31F	Prototype Card
320 - 323	Fixed Disk Controller
360 - 36F	Reserved
378 - 37A	Printer Adaptor
380 - 38F	Alternate Binary Synchronous Communications Adapter, SDLC Adaptor
3A0 - 3AF	Primary Binary Synchronous Communications Adaptor
3B0 - 3BF	Monochrome Display and Printer Adaptor
3C0 - 3CF	Reserved
3D0 - 3DF	Color/Graphics Monitor Adaptor
3F0 - 3F7	Floppy Disk Controller

3F8 - 3FF	Primary Asynchronous Printer Adaptor
-----------	--------------------------------------

To change address jumpers, free address space must be found between 000 and 3FF (HEX) for the emulator and trace I/O addresses. The emulator board requires 8 consecutive addresses and the trace board requires 16 consecutive addresses. If there is a change to the addresses and/or memory, the software will not find the hardware and will automatically display the configuration window where you must put the new address. The Emulator Hardware Config dialog box is described in detail in the Software chapter.

Addressing Examples

The tables below give examples of addressing on the emulator (Figure 8) and trace boards (Figure 9). Table entries are arranged in the same order that the jumpers are physically located on the boards.

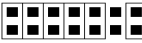






3 4 5 6 7 8 9	Hex Adr.
	100 - 107
	110 - 117
	120 - 127
	140 - 147
	180 - 187
.....	
	210 - 217
.....	
	310 - 317

Figure 8: Emulator Addressing Example








9 8 7 6 5 4	Hex Adr.
	100 - 10F
	110 - 11F
	120 - 12F
	140 - 14F
	180 - 18F
	200 - 20F
	300 - 30F

Figure 9: Trace Addressing Example

Mapping Emulation Memory

Code memory and external data memory can be mapped in 4K blocks to either the emulator or the target (your board). By default the code memory is mapped to the emulator, and the external data memory is mapped to the target. For details on how to change the mapping, please refer to “Mapping memory” on page 79 in the Software chapter of the manual.

If you have a 32k emulator and you map both code and external data to the emulator, code and external data will be overlaid. In other words, for this to work you must have your code and your data occupying different address ranges.

If you have a 128k emulator and you map both code and external data to the emulator, code and external data will reside in physically different areas, so they will NOT be overlaid. If, however, you want code and external data to be overlaid, (all 64k), you must change

the jumpers as shown in Figure 7 on Page 17 of the manual AND
YOU MUST SELECT 32K UNDER **CONFIG EMULATOR
HARDWARE..**

Files Provided

The following files are provided with EMUL51™-PC Windows:

Command/File	Function
EMUL51.EXE	Emulator program.
EMUL51.HLP	On-line Help File
EMUL51.INI	Initialization information
STR.SYM	Directory containing .STR and .SYM files.
8031.STR (and others)	Typical 8051 object files are automatically loaded into the EMUL51 at invocation.
8031.SYM (and others)	Symbol file for 8031 . Symbol files for other processors are also available.
BWCC.DLL 51.DLL	Dynamically loaded libraries that support Windows and the 8051 processor family
EMUL51.CFG	Data file that describes all the supported processors
EXAMPLES	Directory containing sample 8051 programs
EPROMS	Directory containing Nohau EPROM files. See README.1ST

Note that it is not necessary to have *.STR and the *.SYM in the
current directory or even the same directory as the emulator
executable. For more information about where these files can be
placed, please “Setting the Paths ..” on page78 in the Software
chapter.

Memory Bank Switching

Many users today are breaking through the 64k barrier of the 8051 family chips.

If a bank-switching emulator is used without bank-switching there will be 64k of overlaid memory. (CODE and XDATA). In this case the “bank-switch cables” should be left grounded.

Currently Nohau has several types of emulators that we feel will meet most users' needs. If none of the configurations will satisfy your requirements, please contact us.

This section covers the following topics:

1. Software examples with IAR and Keil / Franklin compilers
2. Hardware examples
3. EMUL51™-PC Windows bank-switch commands and examples on setup
4. Description of the Nohau EMUL51-PC/E128-BSW bank-switch emulator
5. Description of the Nohau EMUL51-PC/E256-BSW bank-switch emulator
6. Hints concerning memory bank-switching the Nohau way

The IAR Banked Memory Model

The banked memory model is normally supported by the IAR compiler and does not require any additional software. However, you must use version 4.00 or later of the compiler / linker package. The compiler supports up to 256 banks of code. The IAR compiler supports common area and banks but has (unlike the Keil / Franklin) the restriction that the common area must be at least 16k

byte. Like the Keil / Franklin, the IAR compiler works very well with the Nohau bankswitch emulators. Writing code for switching banks with the IAR compiler does not differ from normal use. However, you should keep the module size as small as possible, or at least minimize the module size and maximize the bank size. This will enable the linker to fit all banks together without losing too much code space.

IAR ICC8051 Compiler Options For bankswitching:

-mb
<p>This parameter determines that banked memory model should be used. The library file used with the banked memory model is called CL8051B.LIB</p> <p>The default port assignment for bank-switching is port P1. If any other assignment is required the assembler file L18.S03 needs to be modified. After the modification, assemble and put the module in the CL8051B.R03 library using the XLIB librarian replace-module command (see the XLIB section in the IAR manual for further reference).</p>

IAR XLINK Linker Options For bank-switching:

-Z(segment type)segment=address
<p>The -Z option is the standard option for attaching addresses to segments. When using bank-switching, this switch is used for the common areas and the -b switch, described below, is used for the switched bank areas.</p>
-b(segment type)segment=bank+start address, bank length, bank increment+offset
<p>The bank determines what the first bank number should be. The start address determines the start address of the banked area. The bank length sets the size of each bank in the banked area.</p> <p>The bank increment consists of two 16 bit values. The first two bytes determine the number of bank increments to be performed when the previous bank is filled. The last parameter offset is an offset from the local address to be able to create asymmetrical bank arrangements. This value is normally set to zero.</p>

Example 1:

```
-b(CODE)CODE, MOD1, MOD2, DISPLAY = 00006000, 2000, 00010000
```

The first two bytes tell the linker that the first bank number is zero and the next two bytes determine that the bank area starts at 6000H. The next parameter sets the bank size to 2000H (8kbyte). The last parameter determines that each new bank number should be incremented by one and that no local offset should be used. Then the linker automatically fills the different 8k-byte banks with code. In this example, no common (root) area is defined. AS a result, this example is not suitable for the Nohau bank-switch emulators.

A more useful example may be as follows:

Example 2:

```
-Z(CODE)MOD_INT, KEY_POLL=0
```

```
-b(CODE)MAIN, KEYBOARD, DISPLAY, PWM_UNIT = 00008000, 8000, 00010000
```

This configuration defines the first bank number to be zero, every bank starts at 8000H, each bank is 32k of size, and the bank number increment is one with no local address offset. The standard -Z option defines the common (root) area. In this case the interrupt and a keyboard poll routine are put in the root area.

This configuration is supported by all of Nohau's different bank-switch emulators and is also quite easy to implement in hardware (see following hardware examples). It is also a good, generic software solution that allows up to 32k of common space for interrupt vectors, string constants, and other frequently used routines. When using the Nohau emulator there is the possibility of having from one to seven 32k banks in addition to the root bank. This will cover most large applications.

In general the IAR banked memory model lets you use many -b declarations to add multiple modules to one bank definition.

Chapter 1: Emulator Board

Bank switching with the Keil/Franklin Banked Linker EMUL51™-PC Win-

Example 3:

```
-b(CODE)CODE0, MAIN, IN_OUT, PWM_UNIT, MAG_READ = 00004000,  
4000, 00010000
```

```
-b(CODE)SERIAL1, I2C_COM, DISK_IO
```

This will define a 16k bank system, with the modules both on line 1 and 2 included in the bank definition. This is easily done by just leaving out the special bank parameters in line 2.

The example above can also be written like this:

```
-b(CODE)CODE0, MAIN, IN_OUT, PWM_UNIT, MAG_READ \  
SERIAL1, I2C_COM, DISK_IO = 00004000, 4000, 00010000
```

For further information on memory bank-switching with the IAR compiler, please refer to the “The Banked Memory Model” section in the IAR manual.

Bank switching with the Keil/Franklin Banked Linker

The Keil / Franklin BL51 Bank Linker is an extension of the standard L51 Linker. The BL51 Linker is able to handle up to 16 64k-byte banks of code which will increase the 8051 code area to 1M-byte instead of the normal 64k byte. Writing code for a bankswitching application does not require any modifications of existing code. This applies to the IAR as well. The only actual difference to the IAR is the number of banks and the fact that no root bank is required. However, most applications require a root bank. The BL51 Linker supports all the different modes of the Nohau bank-switch emulators.

Keil / Franklin BL51 Linker Options

BANKAREA (start, end)	Determines the size of the banked code.
------------------------------	---

Example 1:

BANK AREA (0000H, FFFFH)	This is for a solution using all of the total 64kbyte as banked area.
--------------------------	---

Example 2:

BL51 BANKAREA (8000H, FFFFH)	This is for an application with a physical root bank from 0000H to 7FFFH and the banked area from 8000h to FFFFH.
COMMON (saddr seg)	Locates segments in the common area. The common area is available to all banks and does not require any bank-switching when accessed.

Example 3:

BL51 COMMON {MOD_INT, KEY_POLL, MOD_UART}, BANK0 {MOD_1, MOD2, MOD3}, BANK1 {MOD3}	This puts the interrupt module MOD_INT, the keyboard poll routine KEY_POLL, and the serial interface module MOD_UART in the common area.
BANK0 (saddr seg) / BANKn (saddr seg)	Locates segments in the bank 0 ... n. Up to 16 (0 to 15) banks are supported.

Example 4:

BL51 COMMON {MOD_INT.OBJ, MOD_ROOT.OBJ} BANK0 {MOD_1.OBJ, MOD_2.OBJ} BANK1 {MOD_DISP.OBJ, MOD_4.OBJ} BANKAREA (8000H TO 1FFFFH)

Note: When using an application with no physical common (root) area, the Keil BL51 does not duplicate the areas defined as common. By using the Object Converter OC51 you will get the code of the different banks in separate files. The Nohau EMUL51 supports this duplication automatically from version 5.7R. Please contact your Nohau distributor for updates if required.

For further information on parameters and the BL51 Linker please refer to the Keil / Franklin BL51 Linker / Locator manual and the L51 Linker / Locator manual.

Hardware Bank-Switching Examples

The following pages will show some examples on how to handle 8051 hardware bank-switching. These are only examples, so please use them as suggestions on how to deal with bank-switching when designing your hardware.

Different types of hardware designs can be supported when using the Nohau bank-switch emulators. However, some cases may require a specially configured emulator. If this manual and the different bank-switching schemes do not seem to meet your needs, please contact Nohau to see if we can work out a solution that will suit your requirements.

When designing hardware with multiple memory banks which is to be used with emulators, try to simplify the accessibility of the bank-switch signals since these must always be connected to the pod board (see the emulator descriptions further on in this manual). This is especially important if you use x data writes to switch banks. In these cases the bank-switching signals are only available in your target.

In general, one thing to consider when designing hardware to be used with emulators is to prepare for connecting a pod. Make sure the PCB layout does not put you in a corner. Try to have space around your CPU socket. If you have a surface mounted target,

mount some of your prototype boards with sockets. Surface mounted PLCC sockets are available today from most socket and connector manufacturers.

Configuring EMUL51 for Bank switching

At this time, bank-switching is not supported as completely under the Windows user interface as it is under the DOS user interface. For current status of a feature you need, please contact Nohau customer support.

The DOS EMUL51 commands BANKADDRESS and BANKBYTE have been replaced by a dialog box:

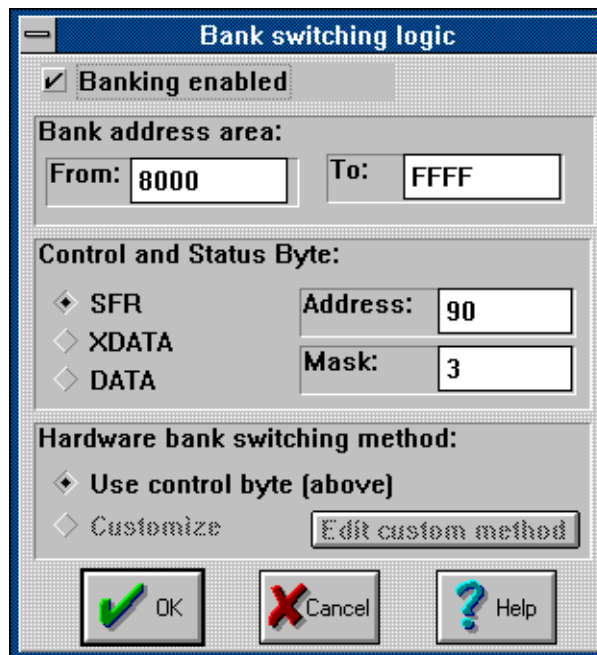
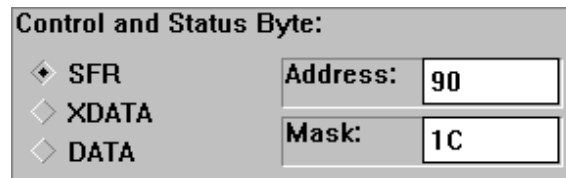


Figure 10: Bank switching Dialog Box

The fields shown in Figure 10 let you control the bank address range (and size), the control byte address and address space, and the bits within the control byte that are used to indicate the currently active bank. The “Customize” button is gray because it is not yet implemented. If your target design is not compatible with the control byte method, please contact Nohau Customer Support for assistance.

Note: *The bits in the control byte that indicate the current active memory bank must be consecutive within the byte and must not be inverted.*

Example 1:



Control and Status Byte:

☒ SFR Address: 90

☐ XDATA Mask: 1C

☐ DATA

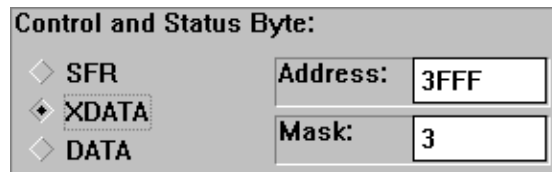
Figure 11: Bank switching Using Port P1

The address 90H refers to port P1 because the SFR button is selected, and the value 1CH is the mask used for “filtering” the port P1 value so that the correct bits are used. In this example the bits P1.2, P1.3, and P1.4 are used. The mask bits must be consecutive and not inverted.

Under the DOS interface, the command would be

BANKBYTE 90H RB 1CH

Example 2:



A screenshot of a software window titled "Control and Status Byte:". On the left, there are three radio buttons labeled "SFR", "XDATA", and "DATA". The "XDATA" radio button is selected. To the right of these buttons are two input fields. The first is labeled "Address:" and contains the value "3FFF". The second is labeled "Mask:" and contains the value "3".

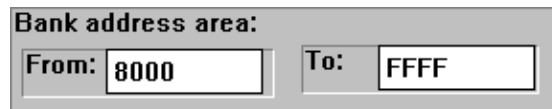
Figure 12: Bank switching Using Memory Map to Address

In this case, XDATA address 3FFFH is used. The two first bits in this byte are used.

Under the DOS interface, this command would be:

BANKBYTE 3FFFH XB 3H

Example 3:



A screenshot of a software window titled "Bank address area:". It contains two input fields. The first is labeled "From:" and contains the value "8000". The second is labeled "To:" and contains the value "FFFF".

Figure 13: Setting the Banked Area

These fields are for setting up the “banked area”. These fields must be set before loading the code. The emulator needs this information to correctly sort the code as it is loaded.

The default values are 8000H to FFFFH, which can be handled by all of the Nohau bank switch emulators.

Under the DOS user interface, the command would be:

BANKADDRESS [address1] TO [address2]

Tracing and bank switching

When using the Trace board with a bank-switching application, the trace can use E0 and E1 to trace the active bank. E0 and E1 are available on the POD board as “wire wrap posts”. When nothing is connected a “1” will be traced. The following combinations on E0 and E1 indicates current bank:

BANK	E0	E1
Bank 0	0	0
Bank 1	1	0
Bank 2	0	1
Bank 3	1	1

The result will be that the correct symbols will be shown according to the bank indicated by the values traced at E0 and E1.

While all pods have trace input pins labeled E0 and E1, different pods have signals as the default inputs to those pins. The table below shows, for most pods, the signals input to the E0 and E1 pins by default. If your pod is not listed in the table below and you wish to use bank switching, please contact Nohau Customer Support.

POD	Rev.	Default E0	Default E1
POD-C51B		CONSULT FACTORY	
POD-C51GB		P5.6/P3.6	P5.7/P3.7
POD-C51FX		CONSULT FACTORY	

POD	Rev.	Default E0	Default E1
POD-C51SL		CONSULT FACTORY	
POD-C52		CONSULT FACTORY	
POD-C152-PGA	B	P4.6/3.6	P4.7/3.7
POD-C152-PGA	A	P4.6/3.6	P4.7/3.7
POD-C152-DIP	C	P4.6/3.6	P4.7/3.7
POD-C152-DIP	B	P4.6/3.6	P4.7/3.7
POD-407		CONSULT FACTORY	
POD-C451B-PGA		IDS (0.6)	ODS (0.7)
POD-C451-DIP		IDS (4.6)	ODS (4.7)
POD-C451-PGA		IDS (4.6)	ODS (4.7)
POD-C452-PGA		P4.6/WR	P4.7/RD
POD-5001		CONSULT FACTORY	
POD-C528		CONSULT FACTORY	
POD-C515A-PGA		E0/P4.6	E1/P4.7
POD-C517B-PGA		P4.6/P5.6	P4.7/P5.7
POD-C517AB-PGA		P4.6/P5.6	P4.7/P5.7
POD-C535-PGA		P4.6	P4.7
POD-C550-PGA		CONSULT FACTORY	
POD-C552-PGA		PWM0/P5.6	PWM1/P5.7
POD-C552B		P4.6/3.6	P4.7/3.7
POD-C537-PGA		P4.6/P5.6	P4.7/P5.7
POD-C592-PGA		CONSULT FACTORY	

Bank switching and Breakpoints

On the Nohau bank-switch emulators there are no direct ways to specify in which bank a break should occur. This could result in breaking at the correct address but in the wrong bank. If you have a trace board, however, it is possible to make the break occur in the correct bank as the following examples will show. This example uses the external signals E0 and E1. The current software only supports E0 and E1 for tracing. That is, only four banks will allow the correct source file to be synchronized with the trace window display. If this is insufficient, please contact Nohau customer support.

In this example, two EZ-hooks must be connected between the bank-switch signals in your target system and E0 and E1 on the POD board. Use E0 as the LSB and E1 as MSB.

Open the Trace setup dialog box and when setting up your address, also set the two top bits of the P3 field to match the bank in which you want to break. For example, to set a breakpoint at address A022 in bank 2, set the **A** trigger condition to:

Address	FWR ST INT	Data	P1	P3
A022H	xxx xx xxxY	xxxxxxxxY	xxxxxxxxY	10xxxxxxxxY

This will work when a common (root) bank is used. If the Break Address is 0000 - 7FFF then the emulator will always break, regardless of the Bank Number. If the address is 8000 - FFFF the Bank Number # must be specified.

The trace can be set up for any type of bank switching supported by Nohau emulators, in addition to a bank-switch application with a configuration other than that where the above example is used. The possibility to set up with the trace breakpoints based either on specified port values or x data writes with a specific value will let you break in the bank you desire.

EMUL51-PC/E128-BSW Bank-switch Emulator

The 128k bank-switch emulator has two different modes of operation which will be discussed later. This emulator cannot emulate XDATA memory. Any POD board can be modified to work as a bank-switching unit. For most operations, one or two wires will be used to control the bank selection. These wires must always be connected to the bank-switch logic. See Figure 14 for the modifications required for the POD board.

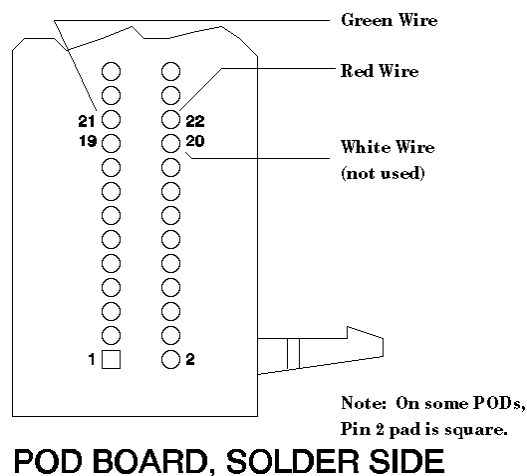


Figure 14: POD Board Modification for EMUL51-PC/E128-BSW

The EMUL51-PC/E128-BSW must be configured as a 32k emulator in the Config Emulator hardware dialog box.

Warning: Do not modify the switch settings shown in Figure 15. This may cause improper function and damage to the emulator.

Chapter 1: Emulator Board

EMUL51-PC/E128-BSW Bank-switch Emulator EMUL51™-PC Windows

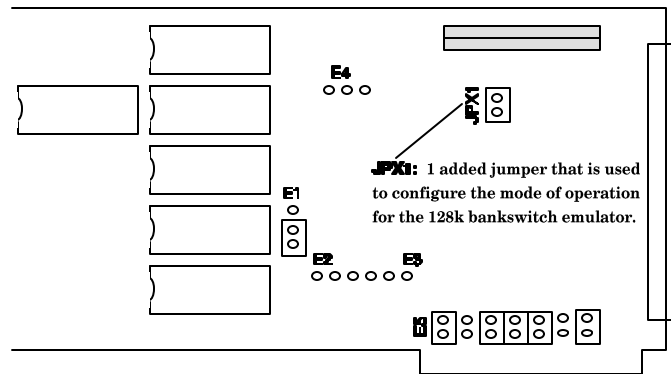


Figure 15: Layout of EMUL51-PC/E128-BSW Bank switch Emulator

JPX1



Mode 0	○	Banksizes = 64k
	○	128k of emulation memory
Mode 1	○	Banksizes = 32k
	○	128k of emulation memory

Memory map mode 0	
0000 - FFFF	Switched bank 0
0000 - FFFF	Switched bank 1

Memory map mode 1	
0000 - 7FFF	Root bank
8000 - FFFF	Switched bank 0
8000 - FFFF	Switched bank 1
8000 - FFFF	Switched bank 2

Figure 16: Jumper Description and Memory Map for the EMUL51-PC/E128-BSW

Chapter 1: Emulator Board

EMUL51™-PC Windows EMUL51-PC/E256-BSW Bank switch Emulator

Mode 0			Mode 1		
Green	Red	Bank	Green	Red	Bank
n/c	0	0	0	0	0
n/c	1	1	0	1	1
			1	0	2

Figure 17: Control Lines for the EMUL51-PC/E128-BSW Bank switch Emulator

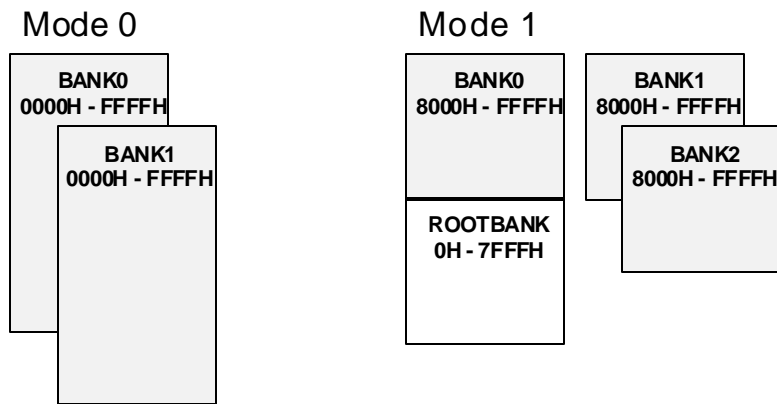


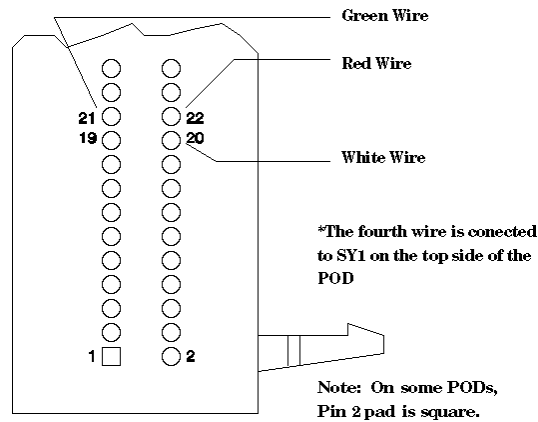
Figure 18: Bank switch Modes EMUL51-PC/E128-BSW Graphic Illustration

EMUL51-PC/E256-BSW Bank switch Emulator

The 256k bank-switch emulator has four different modes of operation which will be discussed later. This emulator can only emulate XDATA memory in mode 3. Any pod board can be modified to work as a bank-switching unit. For most operations, two or three wires will be used to control the bank selection. These wires must always be connected to the bank-switch logic. There is a special mode of operation which is customer specific mode and it requires that four wires are used to control its operation. See Figure 19 below for modifications for the POD board.

Chapter 1: Emulator Board

EMUL51-PC/E256-BSW Bank switch Emulator EMUL51™-PC Windows



POD BOARD, SOLDER SIDE

Figure 19: POD Board Modification for the EMUL51-PC/E256-BSW

The EMUL51-PC/E256-BSW must be configured as a 32k emulator in the Config Emulator Hardware dialog box.

Warning: Do not modify the switch settings shown in Figure 20. This may cause improper function and damage to the emulator.

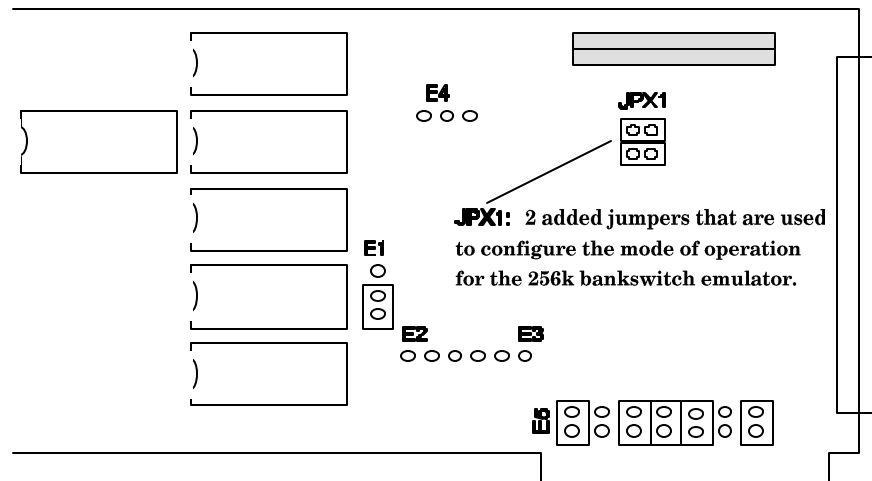


Figure 20: The EMUL51-PC/E256-BSW Bank-switch Emulator

Chapter 1: Emulator Board

EMUL51-PC/E256-BSW Bank switch Emulator EMUL51™-PC Windows

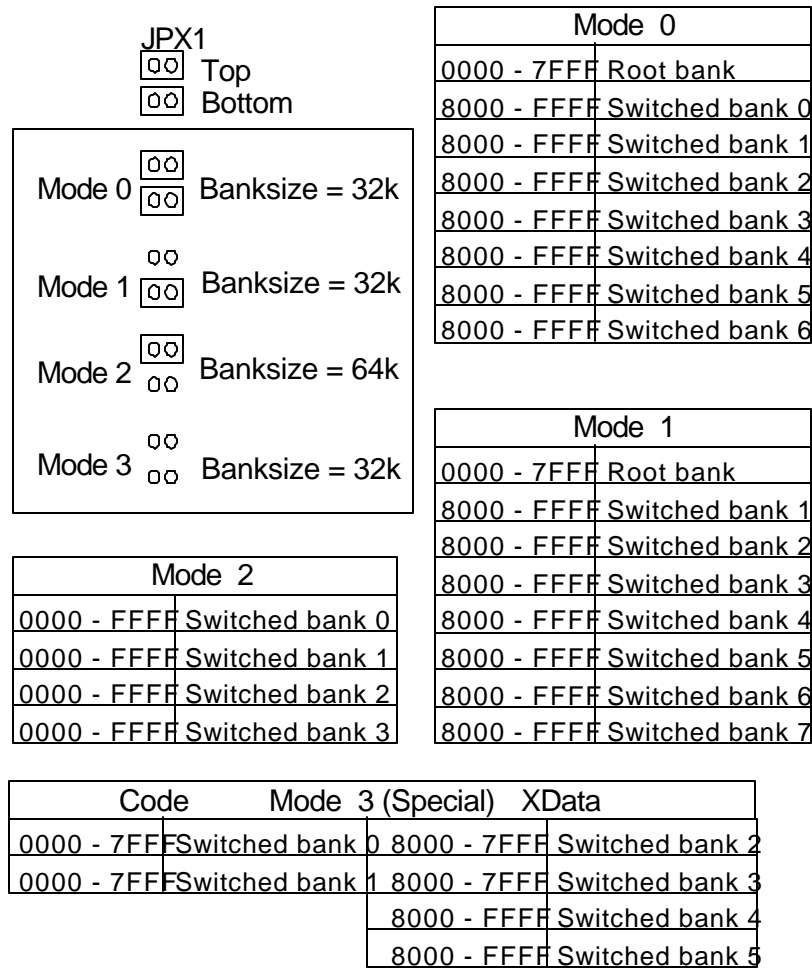


Figure 21: Jumper Description and Memory Map for the EMUL51-PC/E256-BSW

Chapter 1: Emulator Board

EMUL51™-PC Windows EMUL51-PC/E256-BSW Bank switch Emulator

Mode 0				
SY1	White	Green	Red	Bank
n/c	0	0	0	0
n/c	0	0	1	1
n/c	0	1	0	2
n/c	0	1	1	3
n/c	1	0	0	4
n/c	1	0	1	5
n/c	1	1	0	6
n/c	1	1	1	root

Mode 1				
SY1	White	Green	Red	Bank
n/c	0	0	0	root
n/c	0	0	1	1
n/c	0	1	0	2
n/c	0	1	1	3
n/c	1	0	0	4
n/c	1	0	1	5
n/c	1	1	0	6
n/c	1	1	1	7

Mode 2				
SY1	White	Green	Red	Bank
n/c	n/c	0	0	0
n/c	n/c	0	1	1
n/c	n/c	1	0	2
n/c	n/c	1	1	3

Mode 3				Code	Xdata
SY1	White	Green	Red	Bank	Bank
X	0	X	0	0	0
X	1	X	0	1	1
0	0	1	1	0	2
1	0	0	1	0	3
0	0	0	1	0	4
1	0	1	1	0	5
0	1	1	1	1	2
1	1	0	1	1	3
0	1	0	1	1	4
1	1	1	1	1	5

Figure 22: Control Lines for the EMUL51-PC/E256-BSW Bank-switch Emulator

Chapter 1: Emulator Board

EMUL51-PC/E256-BSW Bank switch Emulator EMUL51™-PC Windows

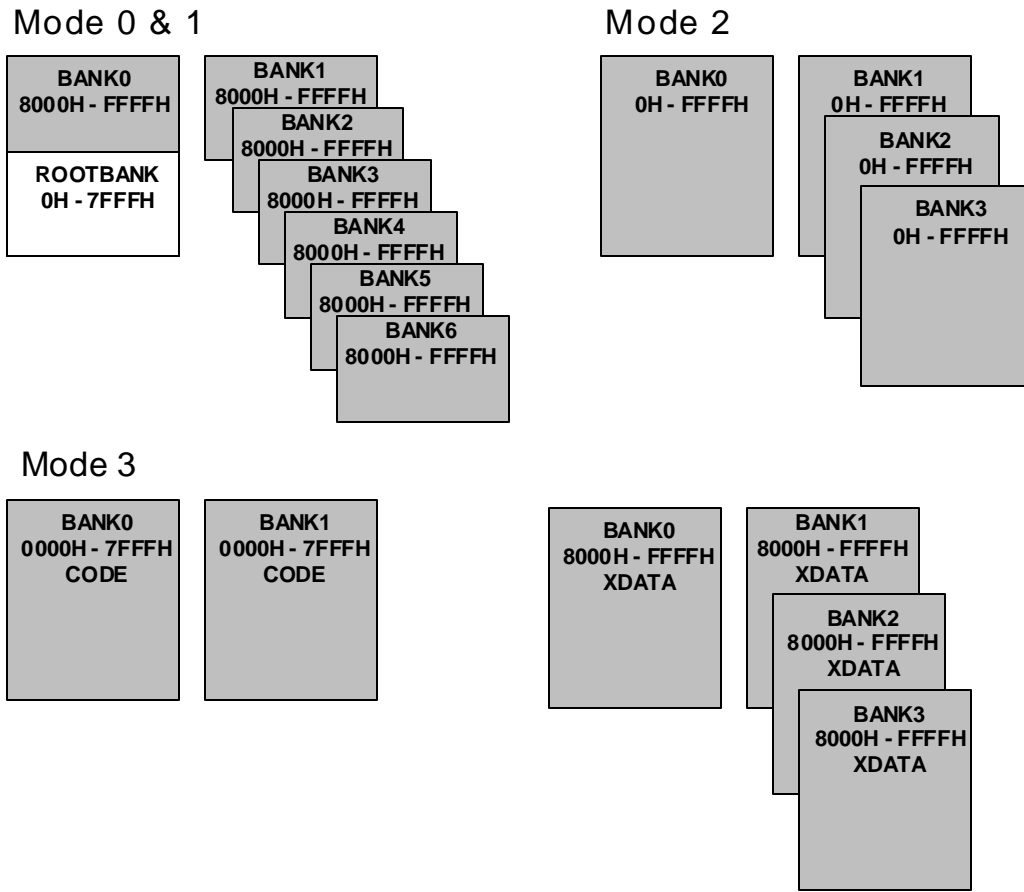


Figure 23: Bank switch Modes of EMUL51-PC/E256-BSW

EMUL51-PC/EA256-256k**Memory Configurations**

The EMUL51-PC/EA256¹ comes with bank-switch support for up to 256k bytes of memory and also supports the Dallas processors with speeded-up cycle times (DS80C320, DS87C520, and DS87C530).

◀

This emulator is configured by both headers and EPROMs. The following is a partial list of the standard EPROMs that will be used. The EPROM labeled COM1.4 is for all pod boards using Intel MC551 architecture; COM1.46 is specifically for the POD-C320; and COM1.47 is for the POD-C520 and POD-C530.

Figure 24 shows the locations of all the EA256 headers and their labels. Please refer to Figure 24 when reading the descriptions on the following pages.

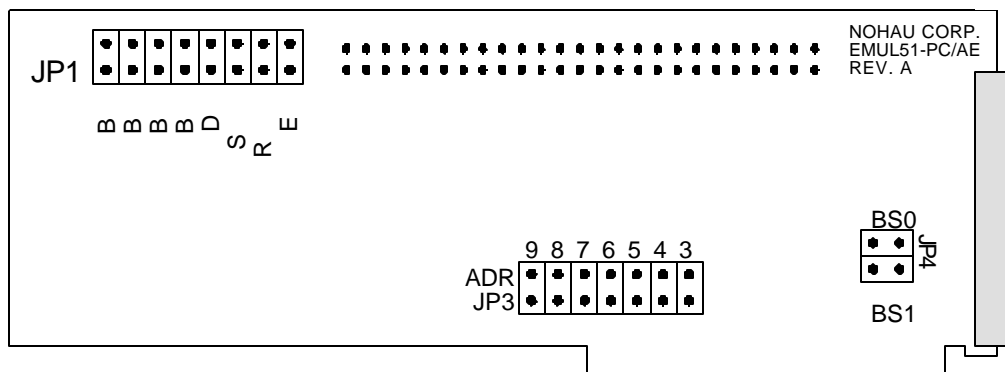


Figure 24: EMUL51-PC/EA256 Header Positions

1. The silkscreen on the board says EMUL51-PC/AE. The official part number is EMUL51-PC/EAxxx

Description of Headers:

JP3 These are the board's I/O addressing jumpers and have the same function as on the standard emulator board.

JP4 These two headers, **BS0** and **BS1**, are used to configure the board for one of the bankswitch modes.

For bankswitching: Both jumpers IN

For Trace ANB output function: both jumpers OUT

JP1 **BM0** through **BM3**

Used to set the memory mode for the emulator (to be described in more detail later).

DAL

This header selects the special mode for the Dallas microcontrollers: 320, 520, & 530 only.

Installed = Dallas, Removed = Normal

Also, remember to change the EPROM to match the type of pod being used.

SCNT

Install a jumper here if you are using a pod board that supports DMA (80C152 & 80C452).

RWEN

This jumper, when removed, allows you to use the R/W lines on the micro as general I/O. This mode will work ONLY with the POD-31A type of pod ("HF" and C320 pods¹), the 520 pod, and all hooks pods. (It MUST be out for POD-520.) If you use this mode, remove the **RE** and **WE** jumpers on the pod board and configure the emulator software for a pod that has a hooks-mode controller (e.g. 8752, 8051FC, 80528, or 87530).

1. Currently the I/O feature is not supported for the Dallas 80C320.

ENF

Untested feature; leave jumper out.

Bank switch / Memory Modes:

X = Don't care

1 = Jumper out

0 = Jumper in

(BM) Headers

3	2	1	0	Description	Configure As
0	0	0	0	No Bank switching; 64k CODE + 64k XDATA in separate memory areas.	128k Emulation memory
0	0	X	1	No Bank switching; 64k CODE with 64k XDATA overlaid on code memory	32k (or bank switched)

Banks size = 32k

3	2	1	0	Description	Configure As
1	0	0	0	32k in root page (0000 - 7FFF) + 3 pages in upper memory (8000 -FFFF) + 64k of XDATA memory. First upper 32k page is the same as root page for POD-31A(Figure 25). Last upper 32k page is same as root page for all other pods. (Figure 26).	128k Emulation memory

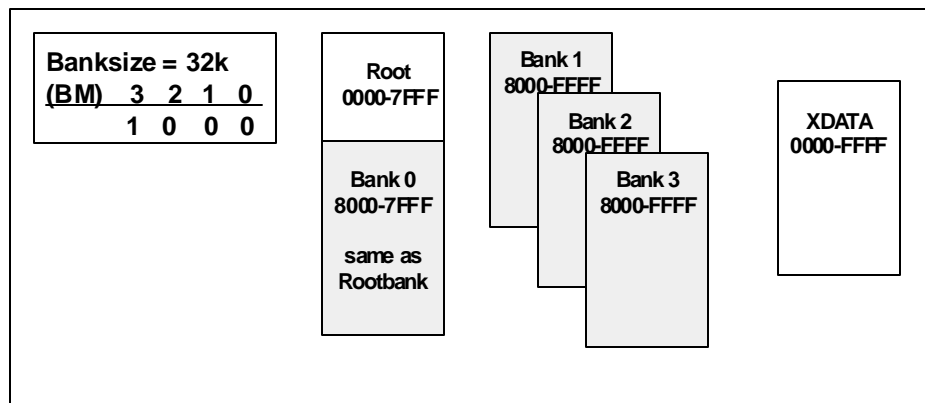


Figure 25: Four 32k-Byte Banks with POD-31A-type Pods¹

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

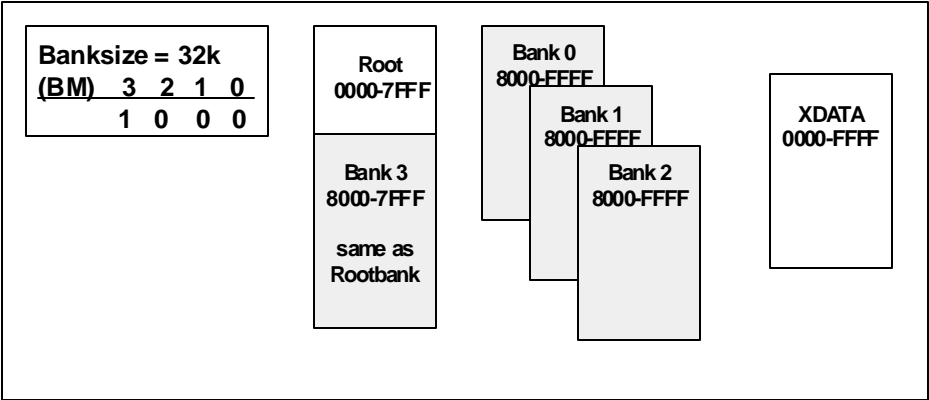


Figure 26: Four 32k-Byte Banks with Most Pod Types

3	2	1	0	Description	Configure As
1	0	0	1	32k in root page (0000 - 7FFF) + 3 pages in upper memory (8000 -FFFF) + 64k of XDATA memory. Last upper 32k page is the same as root page for POD-31A pods (Figure 27). First upper 32k page is the same a root page for all other pods. (Figure 28.)	128k Emulation memory

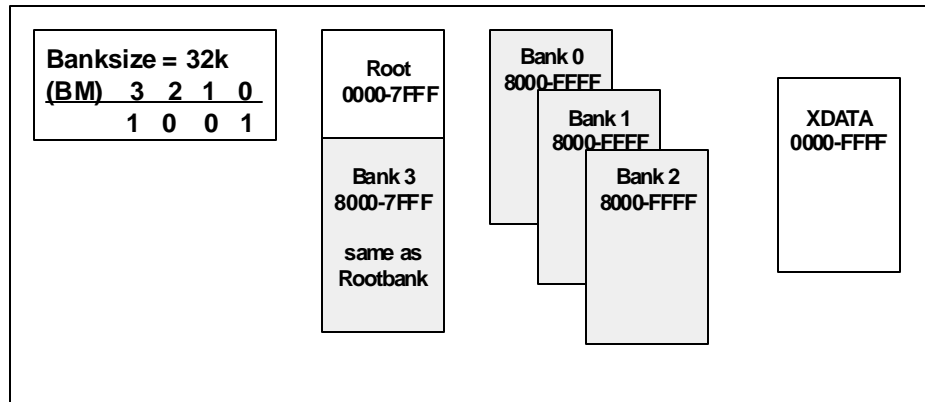


Figure 27: Four 32k Byte Banks with POD-31A-type Pods¹

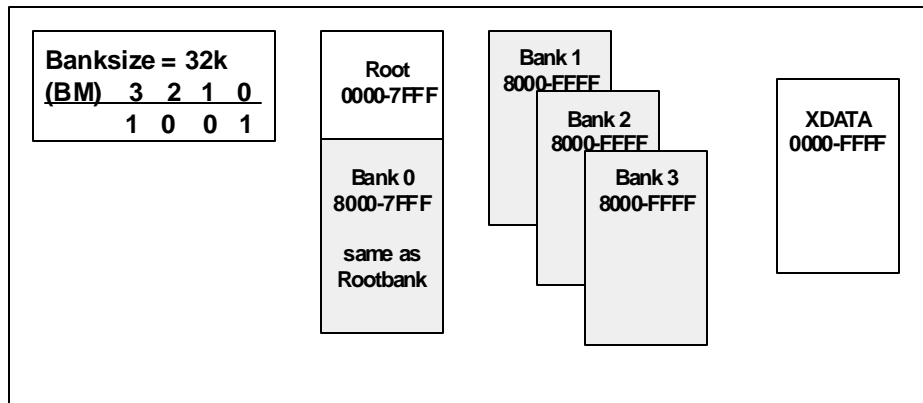
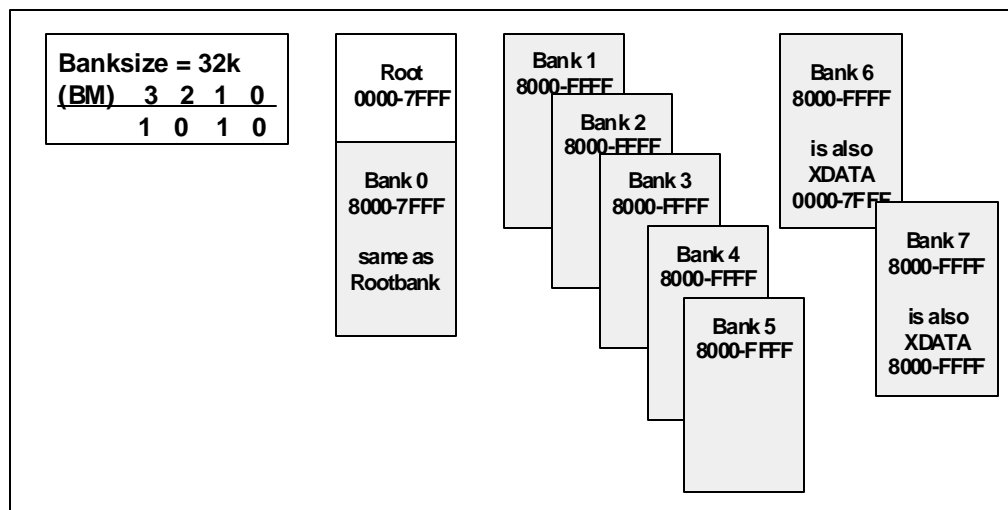


Figure 28: Four 32k Byte Banks with Most Pod Types

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

3	2	1	0	Description	Configure As
1	0	1	0	32k in root page (0000 - 7FFF) + 7 pages in upper memory (8000 - FFFF). XDATA is mapped to the emulator. For POD-31A type pods, the first upper 32k page is the same as the upper root page. Also, the last two upper pages contain the XDATA memory (Figure 29). For other pods, the last 32k page is the same as the upper root page and XDATA is mapped to the first two 32k pages (Figure 30).	128k Emulation memory

Figure 29: Eight 32k Byte Banks with POD-31A-type Pods¹

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

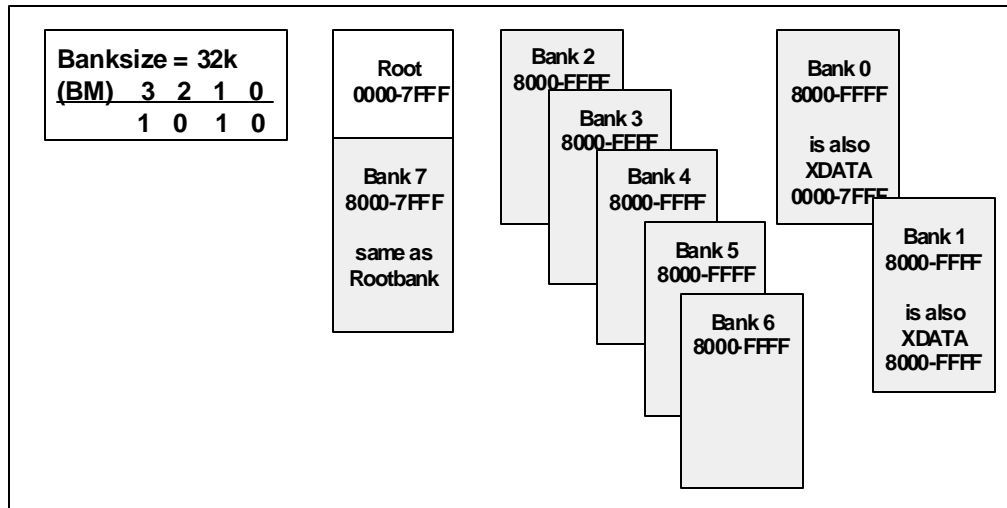
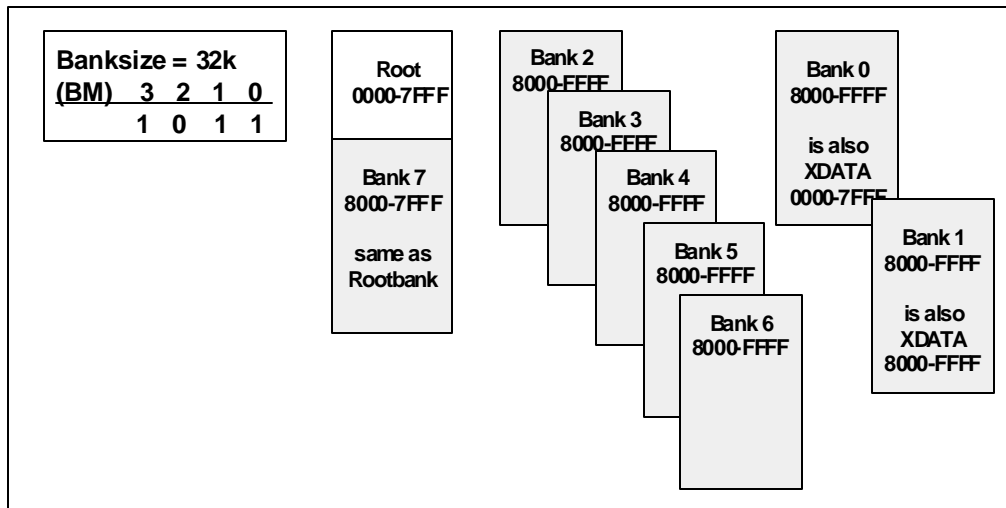


Figure 30: 32k Byte Banks with Most Pod Types

3	2	1	0	Description	Configure As
1	0	1	1	32k in root page (0000 - 7FFF) + 7 pages in upper memory (8000 -FFFF). XDATA is mapped to the emulator. For POD-31A type pods, the last upper 32k page is the same as upper root page. Also, the first two upper pages contain the XDATA memory (Figure 31). For other pods, the first upper 32k page is the same as upper root and the last two 32k upper pages contain XDATA memory (Figure 32).	128k Emulation memory

Figure 31: 32k Byte Banks with POD-31A-type Pods¹

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

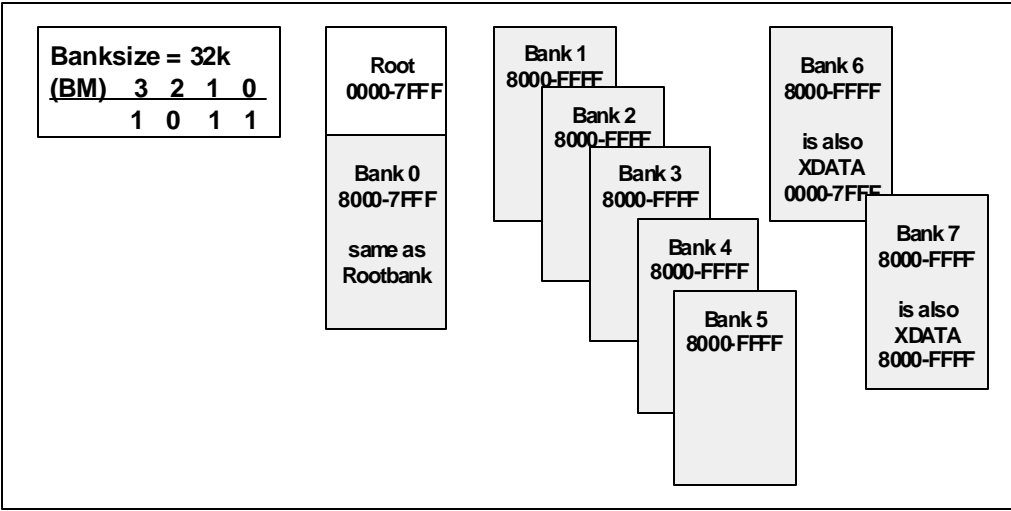


Figure 32: 32k Byte Banks with Most Pod Types

Bank size = 64k

3	2	1	0	Description	Configure As
0	1	0	X	2 banks of 64k for code + 64k for XDATA memory; separate memory areas (Figure 33).	128k Emulation memory

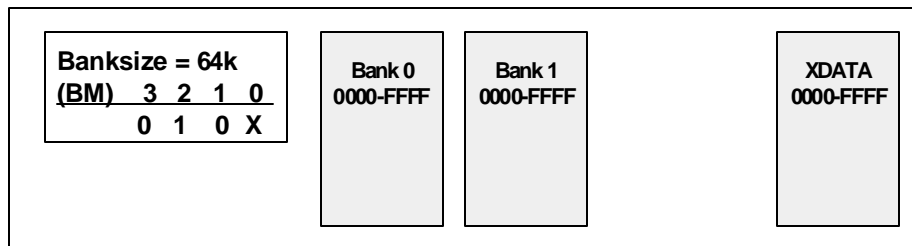


Figure 33: Two 64k Byte Banks (All Pod Types)

3	2	1	0	Description	Configure As
0	1	1	0	4 banks of 64k for code memory. If XDATA is mapped to emulator, XDATA memory appears in the first 64k bank of memory with POD-31A pod types (Figure 34). With other pod types, XDATA will overlay code bank 3 (Figure 35).	128k Emulation memory

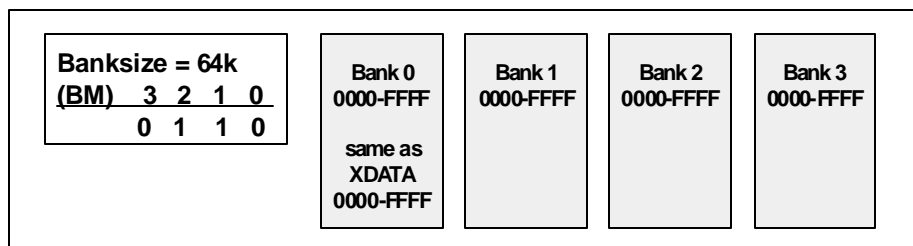


Figure 34: Four 64k Byte Banks with POD-31A-type Pods¹

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

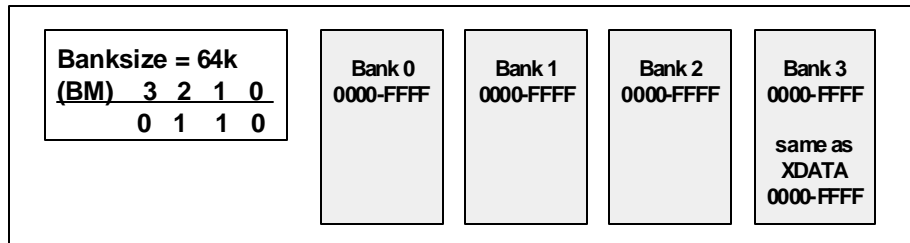
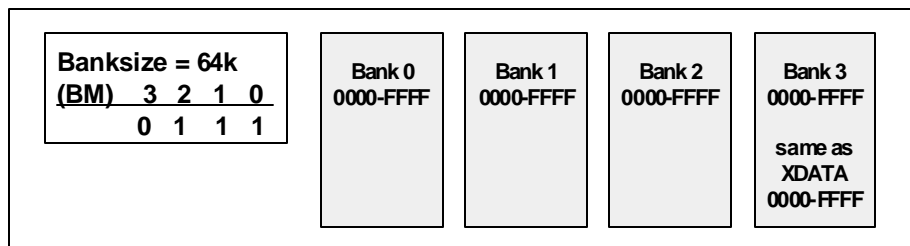


Figure 35: Four 64k Byte Banks with Most Pod Types

3	2	1	0	Description	Configure As
0	1	1	1	4 banks of 64k for code memory. If XDATA is mapped to emulator, XDATA memory appears in last 64k bank of memory with POD-31A pod types (Figure 36). With other pod types, XDATA will overlay code bank 0 (Figure 37).	128k Emulation memory

Figure 36: Four 64k Byte Banks with POD-31A-type Pods¹

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

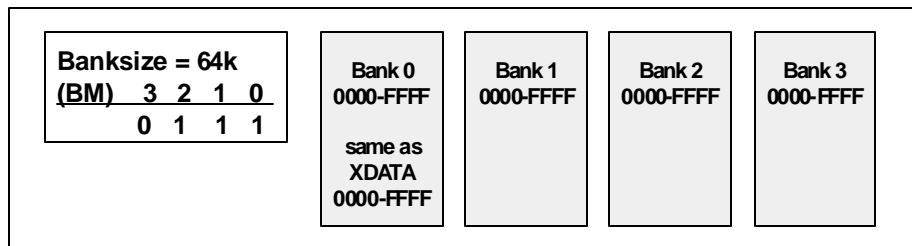


Figure 37: Four 64k Byte Banks with Most Pod Types

Banksize = 48k

3	2	1	0	Description	Configure As
1	1	X	X	16k in root page (0000 - 3FFF) + 3 pages of 48k in upper memory (4000 - FFFF), + lower 48k (0000 -BFFF) of XDATA available in separate memory area, and the upper 16k of XDATA memory is the same as the root 16k of code memory (Figure 38).	128k Emulation memory

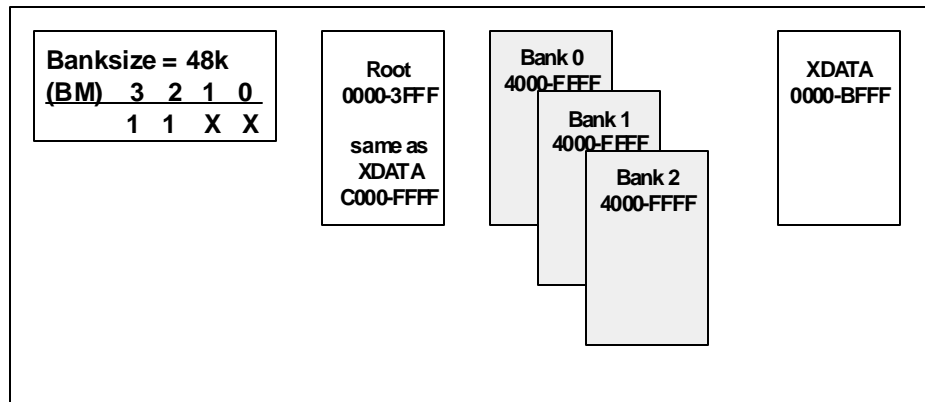


Figure 38: 48k Byte Banks for All Pod Types

Reprogramming an EPROM for another type:

- 1) Part required for this board is a Cypress part CY7C266-20WG. It is very important that this exact part is used due to the timing required by this board.
- 2) The EPROM files are available on the distribution disk and are installed to your hard disk during the installation process.
- 3) You can reprogram the part that came in the board or program another part so that you have a part to support both types. Please refer the the README.1ST file regarding the EPROMs for selecting the proper HEX file for your pod type.
- 4) If you do not have a way to program an EPROM, please contact Nohau's customer support and arrange for an EPROM to be sent to you.
- 5) If you do not have the EPROM files on your disk, you can obtain these files from Nohau on disk or download the files from the Nohau BBS:

PH: 1 (408) 626-7893
No Parity, 1 Stop bit, 1200 to 14.4k baud

This file is located under the "Library of Files" menu option, and in the 8051 library. File name = EPROMS.ZIP

EMUL51-PC/EA768-768k

Memory Configurations

This board comes with bankswitch support for up to 768k memory, and also supports the Dallas processors with speeded-up cycle times (DS80C320, DS87C520, and DS87C530).

The board can be configured for a particular mode of operation. The following is a partial list of the standard EPROMs that will be used. The EPROM labeled COM1.4 is for all pod boards using Intel MC551 architecture; the EPROM labeled COM1.46 is specifically for the POD-C320; and COM1.47 is for the POD-C520 and POD-C530. To reconfigure for each of the different modes, refer to the following jumpers (see Figure 39).

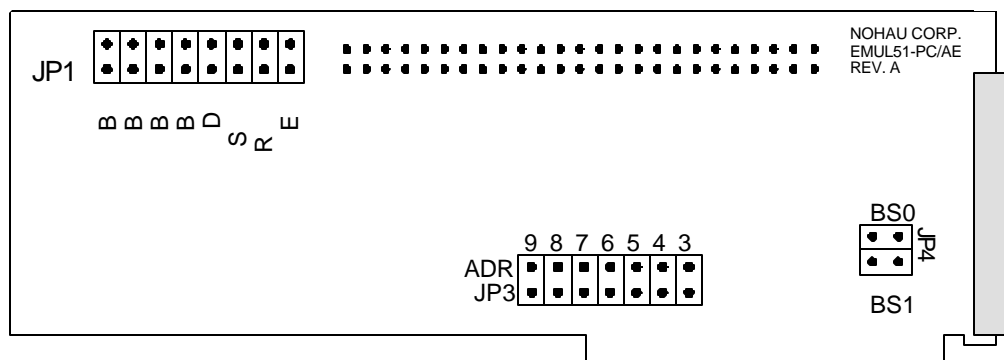


Figure 39: EA768 Emulator headers

Description of Headers:

JP3 These are the board's I/O addressing headers, and have the same function as on the standard emulator board.

JP4 These two headers, **BS0** and **BS1**, are used to configure the board for one of the bankswitch modes.

For bankswitching: both jumpers IN

For Trace ANB output function: both jumpers OUT

JP1 **BM0** through **BM3**:

Used to set the memory mode for the emulator (to be described in more detail later).

DAL:

This header selects the special mode for the following Dallas microcontrollers: 320, 520, & 530 only.

Installed = Dallas, Removed = Normal

Also, remember to change the EPROM to match the type of pod being used.

SCNT:

Install this jumper if you are using a pod board that supports DMA (80C152 & 80C452).

RWEN:

This jumper, when removed, allows you to use the R/W lines on the micro as general I/O. This mode will work ONLY with the POD-31A type of pod ("HF" and C320 pods¹), the 520 pod, and all hooks pods. It MUST be out for POD-520. If you use this mode, remove the **RE** and **WE** jumpers on the pod board and configure the emulator software for a pod with hooks-mode controller (e.g. 8752, 8051FC, 80528, or 87530).

1. Currently the I/O feature is not supported for the Dallas 80C320.

ENF:

Untested feature; leave jumper out.

Bank switch / memory modes:

X = Don't care

1 = Jumper out

0 = Jumper in

(BM) jumpers

3	2	1	0	Description	Configure As
0	0	0	0	No Bank switching; 64k CODE + 64k XDATA in separate memory areas.	128k Emulation memory

3	2	1	0	Description	Configure As
0	0	X	1	No Bank switching; 64k CODE with 64k XDATA overlaid on code memory.	32k Emulation memory

Bank size = 32k

3	2	1	0	Description	Configure As
1	0	0	0	32k in root page (0000 - 7FFF) + 3 pages in upper memory (8000 -FFFF) + 64k of XDATA memory. With POD-31A-type pods, the first upper 32k page is the same as the upper root page Figure 40). All other pods overlay the last upper page and the upper root page (Figure 41).	128k Emulation memory

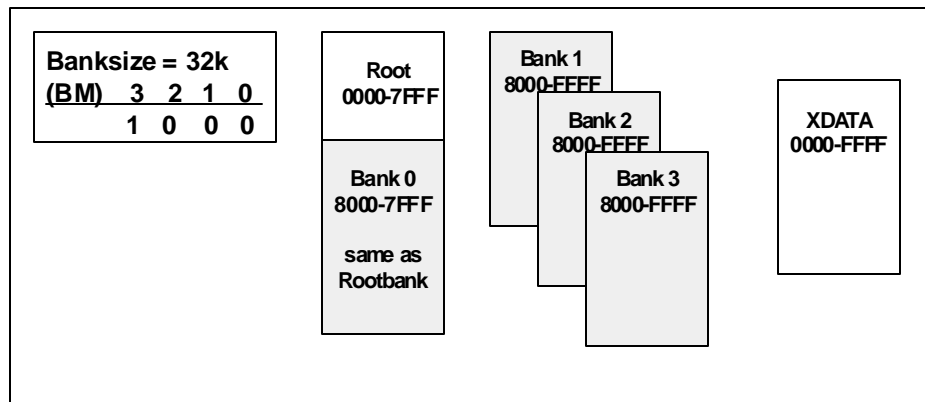


Figure 40: Four 32k Byte Banks for POD 31A-type Pods¹

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

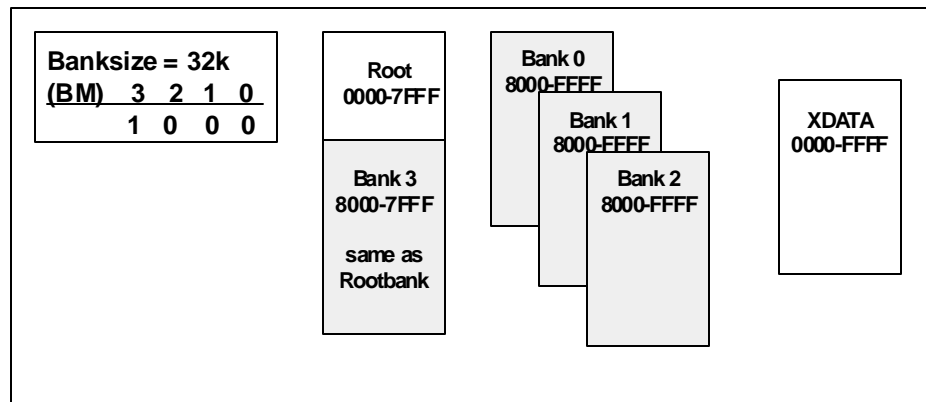


Figure 41: Four 32k Byte Banks for Most Pods

3	2	1	0	Description	Configure As
1	0	0	1	32k in root page (0000 - 7FFF) + 3 pages in upper memory (8000 -FFFF) + 64k of XDATA memory. With POD-31A-type pods, the last upper 32k page is the same as the upper root page (Figure 42). All other pods overlay the first upper page with the upper root page (Figure 43).	128k Emulation memory

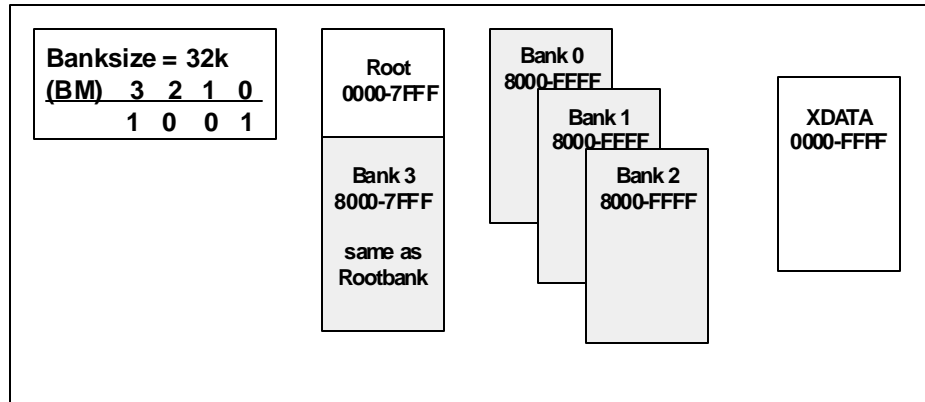


Figure 42: Four 32k Byte Banks for POD 31A-type Pods¹

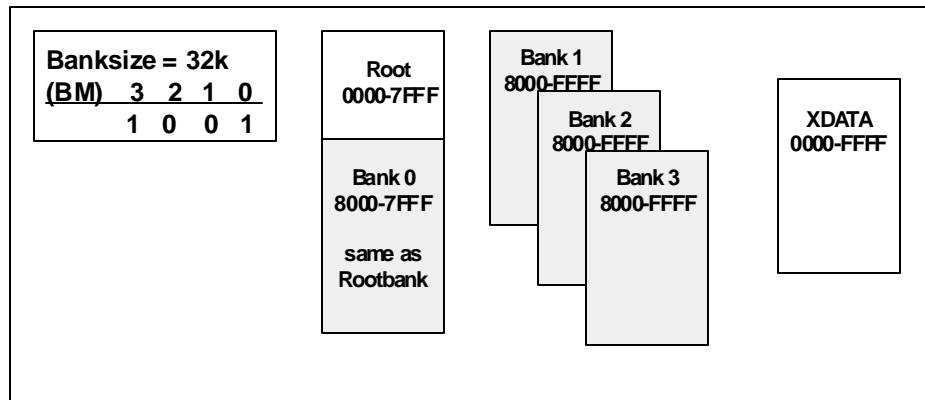


Figure 43: Four 32k Byte Banks for Most Pods

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

3	2	1	0	Description	Configure As
1	0	1	0	32k in root page (0000 - 7FFF) + 7 pages in upper memory (8000 -FFFF) + 64k of XDATA memory. With POD-31A-type pods, the first upper 32k page is the same as upper root page (Figure 44). All other pods overlay the upper root page with the last upper 32k page (Figure 45).	128k Emulation memory

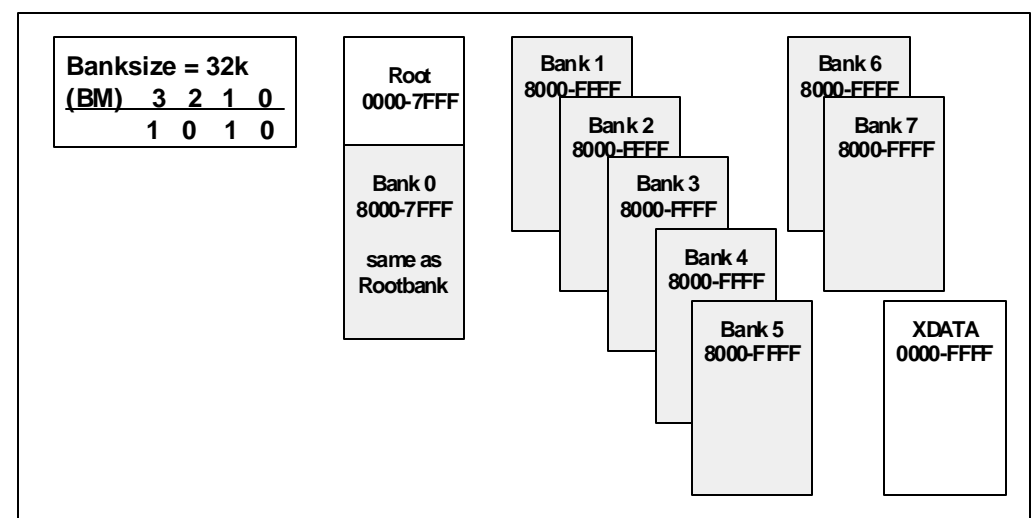


Figure 44: Eight 32k Byte Banks for POD-31A-type Pods¹

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

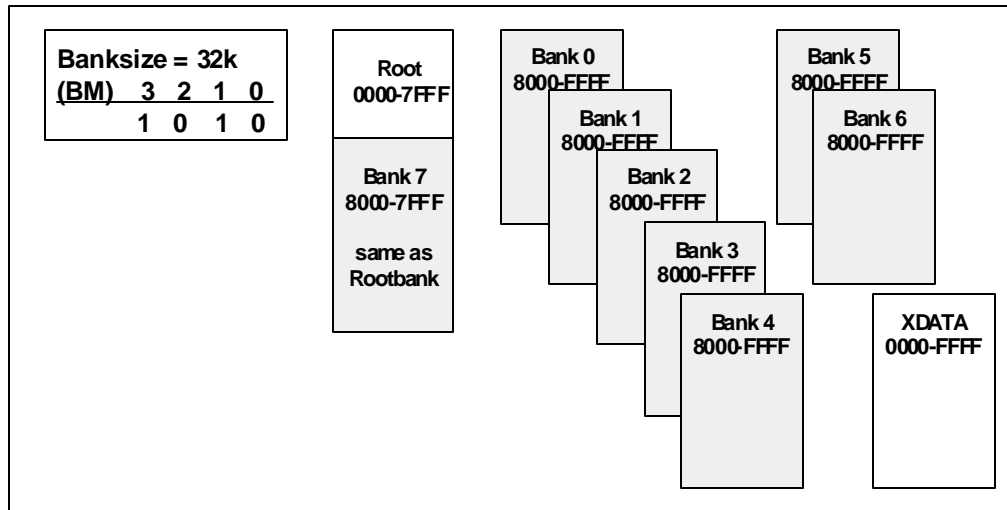
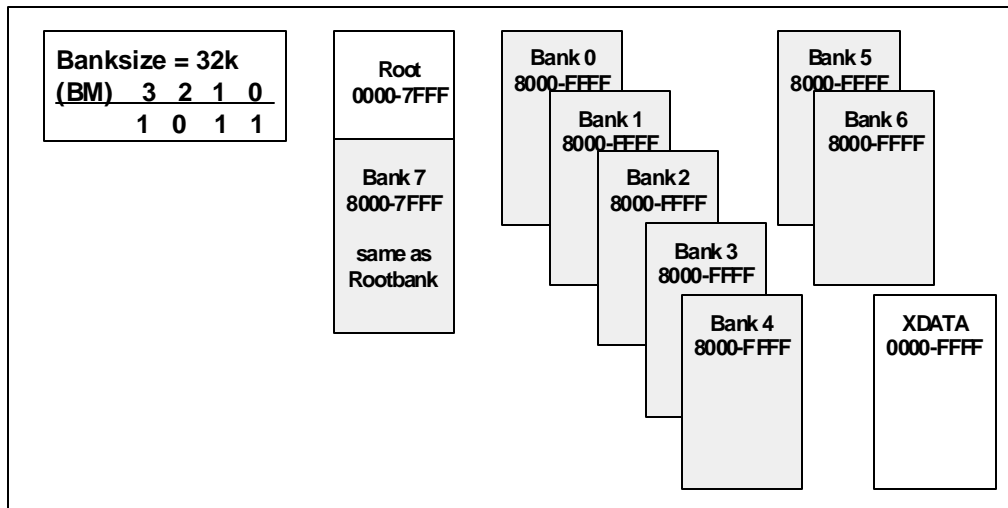


Figure 45: Eight 32k Byte Banks for Most Pods

3	2	1	0	Description	Configure As
1	0	1	1	32k in root page (0000 - 7FFF) + 7 pages in upper memory (8000 -FFFF) + 64k of XDATA. With POD-31A-type pods, the last upper 32k page is the same as root page (Figure 46). All other pods overlay the upper root page (Figure 47).	128k Emulation memory

Figure 46: Eight 32k Byte Banks for POD 31A-type Pods¹

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

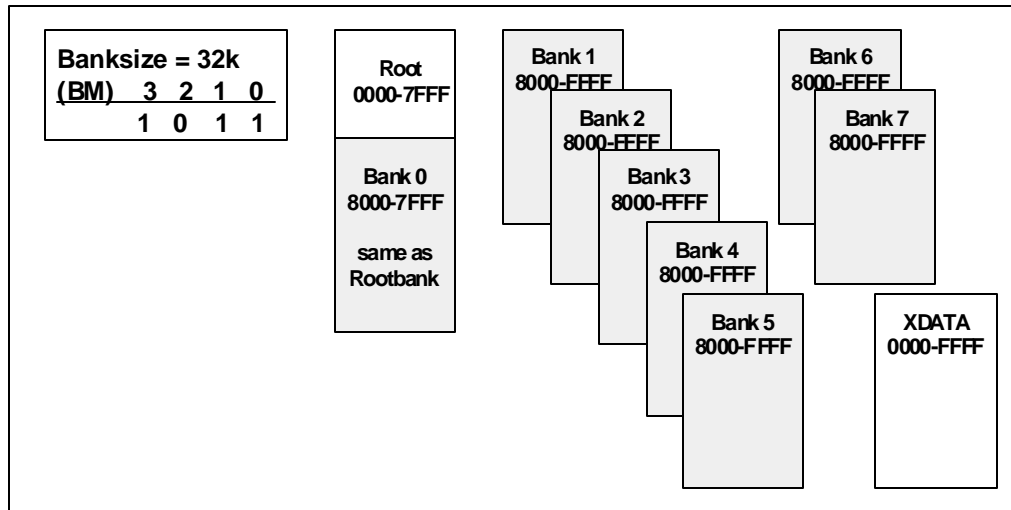
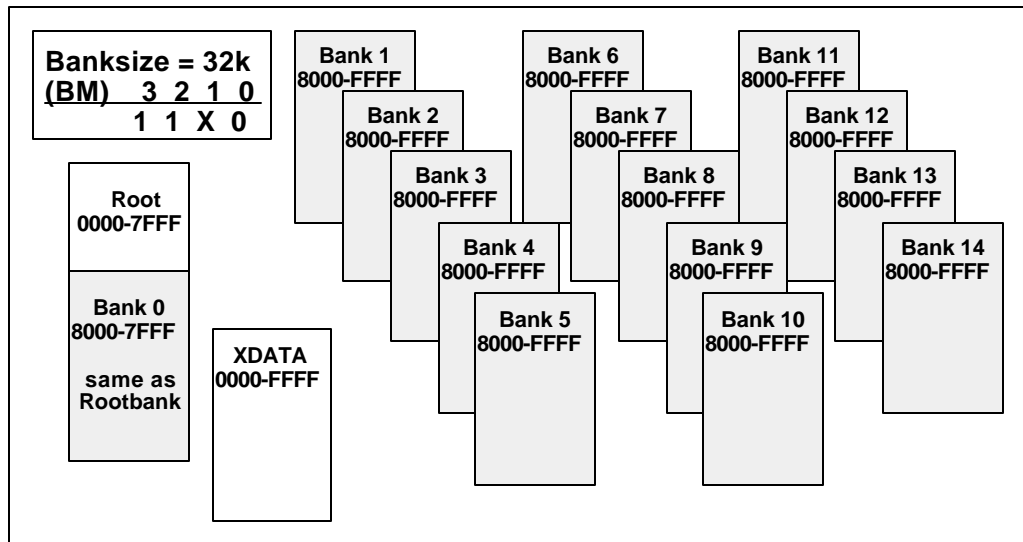


Figure 47: Eight 32k Byte Banks for Most Pods

3	2	1	0	Description	Configure As
1	1	X	0	32k in root page (0000 - 7FFF) + 15 pages in upper memory (8000 -FFFF) and 64k of XDATA memory. With POD-31A-type pods, the first upper 32k is the same as root page (Figure 48). All other pods overlay the upper root page with page 14 (Figure 49).	128k Emulation memory

Figure 48: 16 32k Byte Banks for POD-31A-type Pods¹

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

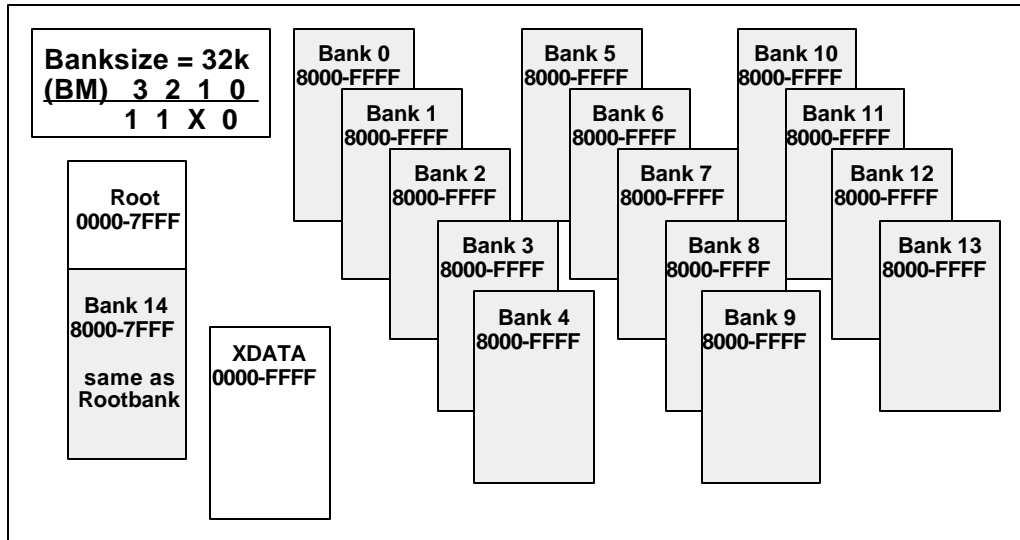


Figure 49: 16 32k Byte Banks for Most Pods

3	2	1	0	Description	Configure As
1	1	X	1	32k in root page (0000 - 7FFF) + 15 pages in upper memory (8000 -FFFF) and one 64k page of XDATA memory. With POD-31A type pods, the last upper 32k is the same as root page. (Figure 50). All other pods overlay bank 0 with the root page (Figure 51).	128k Emulation memory

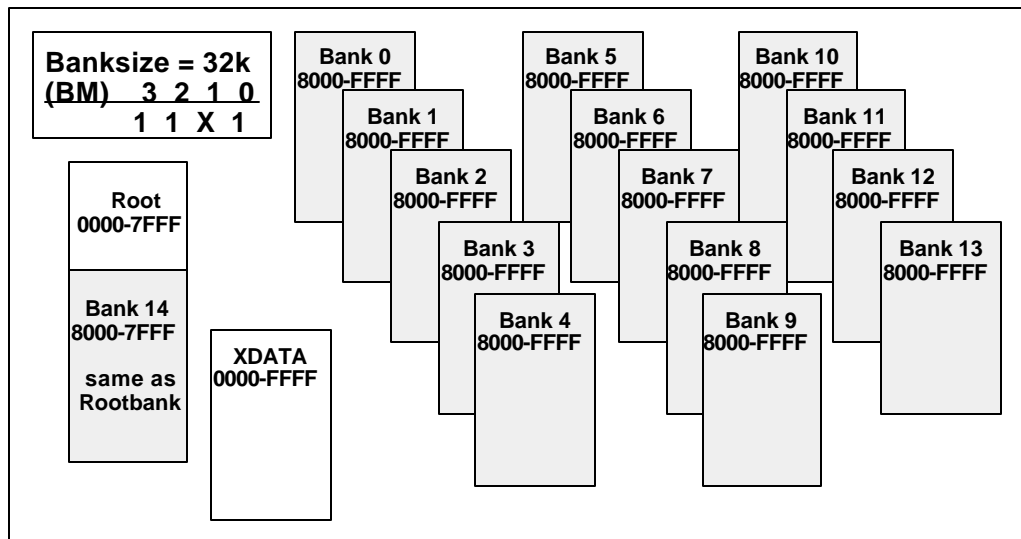


Figure 50: 16 32k Byte Banks for POD 31A-type Pods¹

1. "POD-31A" type refers to POD-C32HF-42, POD-C154HF-42, POD-C320-25, and any similar pod built with the POD-31A fabrication.

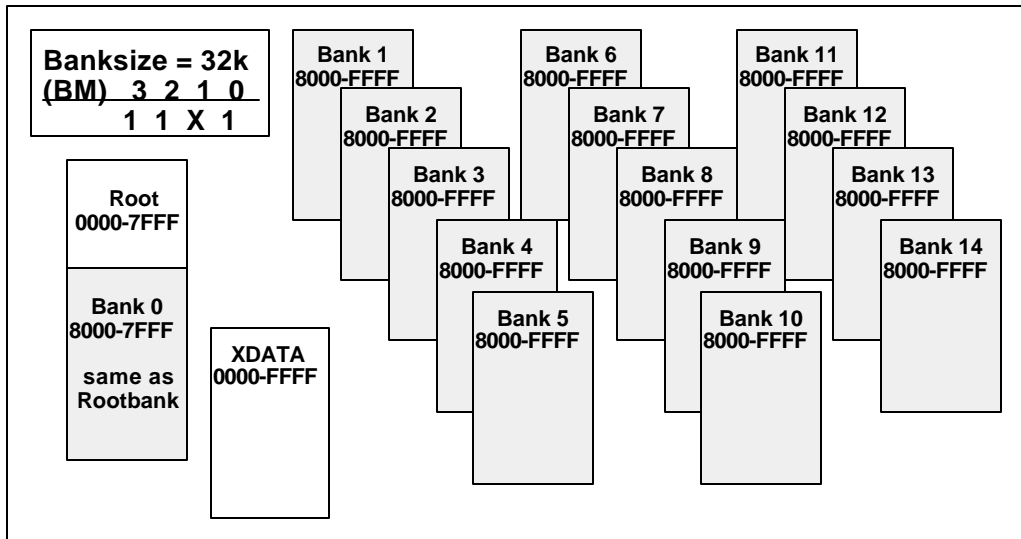


Figure 51: 16 32k Byte Banks for Most Pods

Bank size = 64k

3	2	1	0	Description	Configure As
0	1	0	X	2 banks of 64k for code + 64k for XDATA memory; separate memory areas (Figure 52).	128k Emulation memory

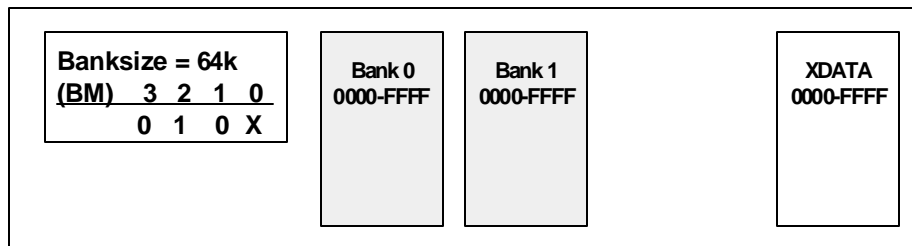


Figure 52: Two 64k Byte Banks for All Pods

3	2	1	0	Description	Configure As
0	1	1	0	4 banks of 64k for code memory + 64k for XDATA memory (Figure 53).	128k Emulation memory

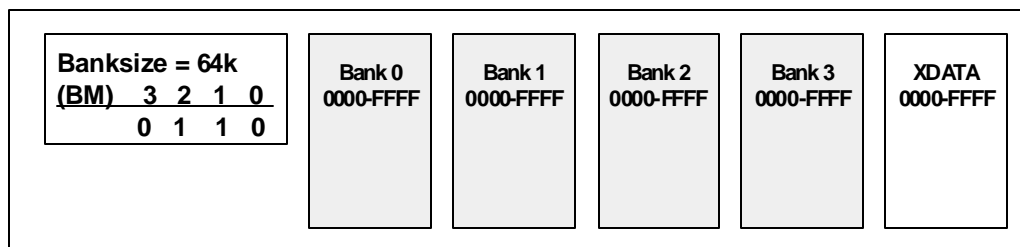


Figure 53: Four 64k Byte Banks for All Pods

3	2	1	0	Description	Configure As
0	1	1	1	8 banks of 64k for code memory + 64k for XDATA memory (Figure 54).	128k Emulation memory

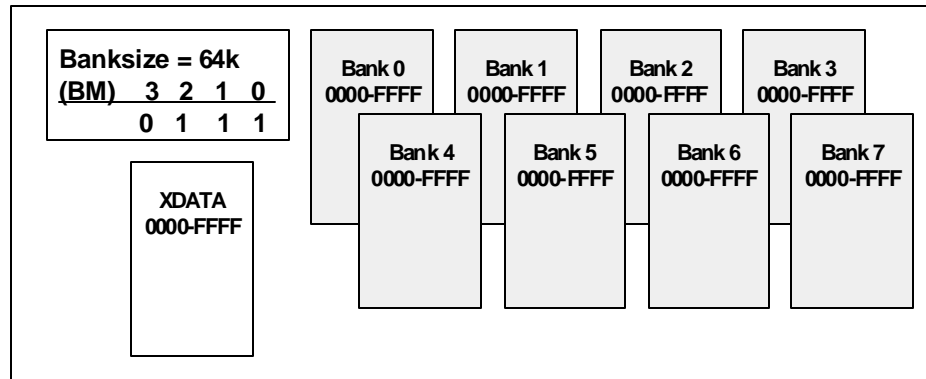


Figure 54: Eight 64k Byte Banks for All Pods

Reprogramming an EPROM for another type:

- 1) The part required for this board is a Cypress part CY7C266-20WG. It is very important that this exact part is used due to the timing required by this board.
- 2) The EPROM files are available on the distribution disk and are installed to your hard disk during the installation process.
- 3) You can reprogram the part that came in the board or program another part so that you have a part to support both types. Please refer the the readme.1st file regarding the EPROMs for selecting the proper HEX file for your pod type.
- 4) If you do not have a way to program an EPROM, please contact Nohau's customer support and arrange for an EPROM to be sent to you.
- 5) If you do not have the EPROM files on your disk, you can obtain these files from Nohau on disk or download the files from the Nohau BBS:

PH: 1 (408) 626-7893 or email support@icetech.com
No Parity, 1 Stop bit, 1200 to 14.4k baud

This file is located under the "Library of Files" menu option in the 8051 library. File name = EPROMS.ZIP

Modified bank-switch pods use a different logic than the POD-31A type of pod. Refer to the memory map information for each mode and pod type.

Chapter 2: Software User Interface

Detailed Installation Instructions

Before installing the software, it is important to have a basic understanding of how to operate *MS Windows*. For help mastering *MS Windows*, please refer to the *Microsoft Windows* User's Guide.

The EMUL51™-PC Windows floppy disk includes an *MS Windows*-compatible SETUP.EXE program. To install this software, from within , 3.1/3.11Windows run SETUP.EXE by typing "WIN A:SETUP" at the DOS prompt before entering *Windows* or, from within *MS Windows*, by selecting the **RUN** item in the **Program Manager File** menu and type **A:SETUP** as the file to run.

If your installation disk is not in drive A, replace the letter "A" with the letter corresponding to the appropriate drive.

From *Windows '95*, start the My Computer facility by double-clicking on the **My Computer** desktop icon, and selecting the disk drive that contains the installation disk: double-click on the **drive** icon, then double-click on the **Setup** icon. As an alternate, you may also select **Run** from the **Start** menu, then **Run** from the list, then type **A:SETUP**. Windows NT users must be logged in as administrator before they can install the software.

A dialog box will ask for a directory for the EMUL51™-PC Windows software. Either accept the suggested directory or type a different one. **Setup** will copy files from the floppy to the hard disk directory specified and modify the configuration information stored in the ".ini" files as needed.

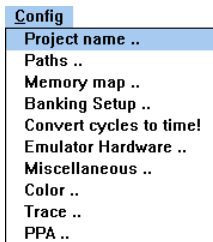
For *Windows '95* users, there will be a Nohau program group with one icon: WIN51. Double-clicking on the emulator icon will start the EMUL51™-PC Windows application. If you wish to move the

icon to another group, you may do so by using the **Move...** menu item in the **Program Manager's File** menu or by dragging the icon to the new group.

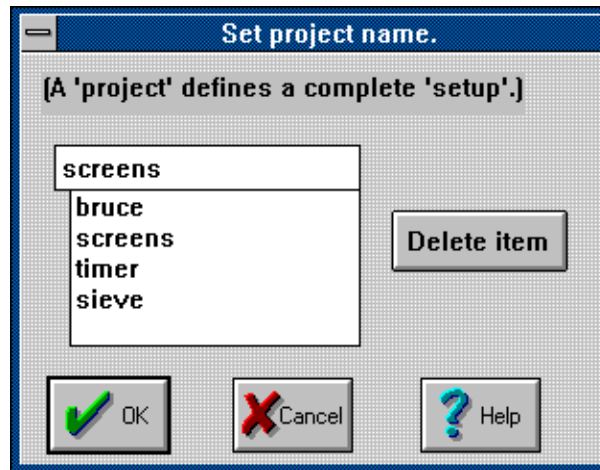
Configuring the Software

If the Quick Installation instructions do not work, you will most likely need to adjust either the hardware jumpers, the software configuration, or possibly both. Please refer to the appropriate chapters for setting the jumpers on any of: the Emulator board, the Trace board, or the Pod board. (For now, the POD board chapters are in the EMUL51 DOS manual.). The next few pages describe all of the items in the **Config** menu. Use these menu items to examine the software configuration in detail and to change it as needed.

Projects



A project is a collection of software configuration settings that are all associated with a specific person, target, or software development project. This menu item opens a dialog box that allows you to set up named configurations or projects. This is first in the menu and described first because all of the other **Config** menu item settings will be stored as settings for the current project in a file with a ".pro" suffix. All software configuration settings are written to disk every time you change projects or whenever you exit the emulator software. There is no "exit without saving changes" option. Also, all the open windows and their positions will be saved so that the current working environment can be easily restored.



:

Figure 55: Set Project Name Dialog Box

To add a project, open the **Set Project Name** dialog box and type the new name over the current name. Because the project name is used as the body of a DOS file name, do not use characters in the name that cannot be used in a file name (like a space character). The new project will inherit all the settings from the old project. Projects are deleted by highlighting the project name you want deleted and then clicking on the **Delete item** button.

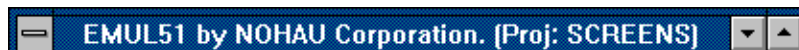
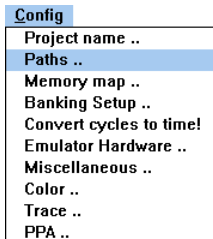


Figure 56: EMUL51™-PC Windows Title Bar

The name of the current project appears in the title bar, which appears at the top of the background window. Figure 56 is an example of the **EMUL51** title bar for the project named **SCREENS** (used to create the screen shots for this manual).

Setting the Paths ..



The next item in the **Config** menu is **Paths ..** , which opens the dialog box shown on page 78. The emulator uses these directories to find the files it needs.

Each of these fields can hold up to 1024 characters. Each directory in the path must be separated by a semi-colon (;) just like MS DOS path names. By default, The **User load modules:** field will contain the directory from the last loaded object file, and the **Emulator internal files:** field will contain the directory where the emulator files were installed.

The **User load modules** directory is the default directory searched for Intel Hex files and absolute object files. Any directory can be specified when loading a module, but the directory shown here is the default. The **.ext** field specifies the default file extension. Files in the default directory with this extension will be listed in the **Load code..** dialog box.

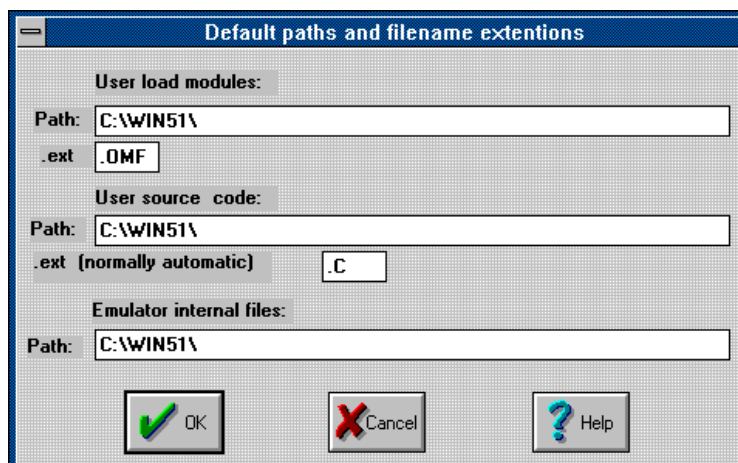
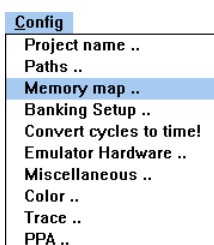


Figure 57: Paths Dialog Box

With many compilers, the full path name of each source file is contained within the object file. Linked object files consisting of several linked objects will, correspondingly, have several source file names and paths. If that source file name exists in the object file that EMUL51™-PC Windows is loading, the debugger will first look that source file when updating the **Source** window.

The second field, **User source code:**., identifies other directories to search for missing source files not identified in the object file or files moved since the compile. The directories in this field must be entered by you, the user. Multiple directories may be entered by using a semicolon between them. Once entered, directories will stay here until you remove it. The **Emulator internal files:** field will be set during the installation and probably will not need to be changed. **Emulator internal files:** is the directory the application uses to find the .STR and .SYM files that are part of the EMUL51™-PC Windows software. Normally, the installation program will set this to the proper directory. If you copy or move EMUL51™-PC Windows software to a new directory or disk drive, remember to change this field also.

Mapping memory



ROM and RAM on the target can be emulated by RAM on the emulator board. This RAM is called emulation RAM, or emulation memory. The entire address range for both ROM and RAM can be mapped to either the target or the emulator memory in 4K byte blocks. The **Memory map ..** menu item under the **Config** menu opens the dialog box that controls those address ranges.

When an address is mapped to emulation RAM on the emulator board, all READ, WRITE, and instruction fetch cycles at that address are directed to emulation RAM. Target RAM, target ROM, and memory mapped devices on the target at that address are ignored. If your target has a memory-mapped I/O device within a block mapped to emulation RAM, this mapping will prevent your application from accessing that device. To avoid this, map the blocks that contain target devices to the target.

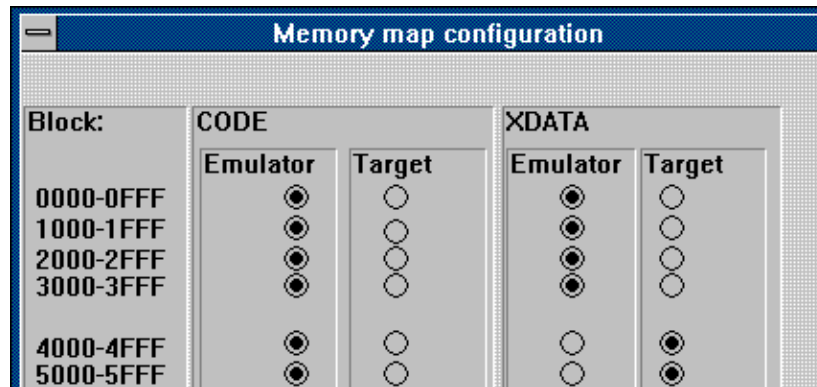


Figure 58: Mapping Memory

Both CODE and XDATA memory are divided into 1000 H byte blocks. The block size is not adjustable. Each block can be mapped to either emulation RAM or target RAM/ROM. Simply click on the button representing the kind of memory you want for each block.

For your convenience, at the bottom of the dialog box, there are buttons that map all addresses to the emulator or to the pod.

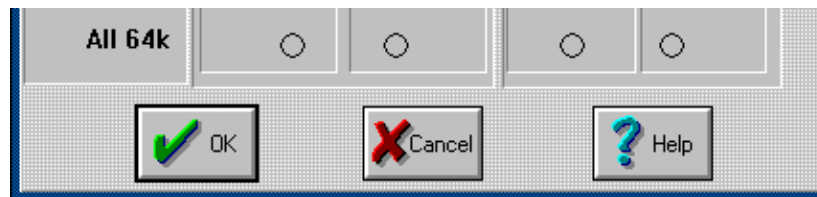
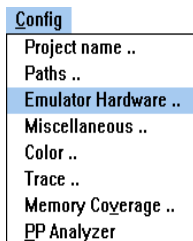


Figure 59: Exiting the Memory Map Dialog Box

When all the blocks are mapped as you want them, click on the OK button.

Emulator Hardware Configuration



The **Emulator Hardware ..** menu item configures the software to correctly communicate with the hardware. The **Emulator port** field contains the hexadecimal address of the emulator in the PC's I/O space. The value entered in the Emulator Port field must agree with the jumper settings on the emulator board header J1E5 (see "Setting the Emulator address" on page 18). If they do not agree, the EMUL51™-PC Windows software will not be able to communicate with the hardware, and a warning will automatically be displayed as a reminder that communication has failed and some change is needed.

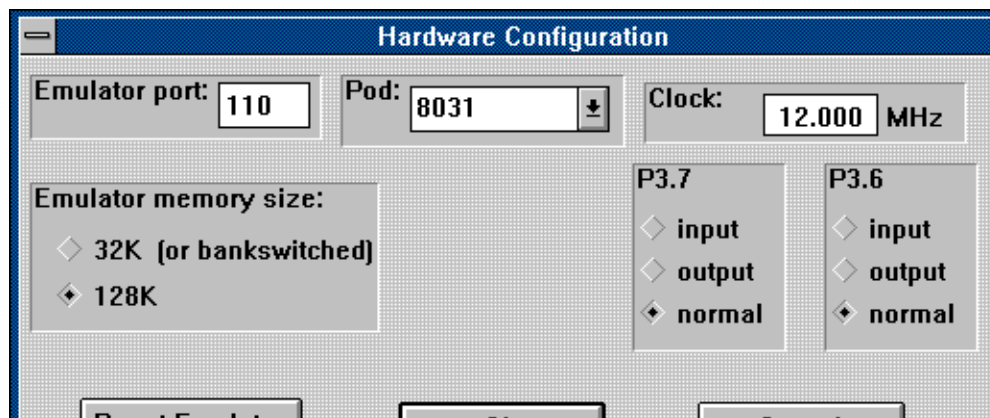


Figure 60: Hardware Configuration Dialog Box

Warning: The settings in this dialog box must agree with the emulator jumper settings, the pod processor type, and how the application start-up code sets up certain control registers. If this is not the case, EMUL51™-PC Windows will not work properly.

Click on the **Pod** field and you will see a long list of supported controllers. Changing the type of pod is as easy as selecting a new processor from the list then clicking on the **Reset Emulator** button. (This button will tell the emulator to read the .SYM and .STR files for that pod and controller.) See the POD Chapter in the EMUL51-PC DOS Manual for the correct choice (refer to the “-p” parameter).

The two sets of radio buttons labeled **P3.7** and **P3.6** support a special feature found on some special pods. For most of the pods, these two pins are used for READ and WRITE strobe signals during external bus cycles. This corresponds to the **Normal** button and under most circumstances, you must select the **Normal** button to ensure correct emulator operation.

Some pods have added circuitry and can support using these two pins for either input or output. ONLY IF you have a POD-31S (or one of its derivatives), or a port substitution pod, AND you have changed the jumpers and switches on that pod to support input or output on either of those pins, can you select either of the **Input** or **Output** buttons for either pin. If you have one of these special pods, please refer to the EMUL51 DOS interface manual, page 4-17 for more information about how to configure the switches and jumpers.

Miscellaneous Configuration

The **Miscellaneous** item in the **Config** menu opens a dialog box that controls special features of EMUL51™-PC Windows :

- when and if automatic resets occur
- optional startup address
- the source code address range for limiting where breakpoints are set

By default, the emulator resets the controller when the EMUL51™-PC Windows software is started and after an object file is loaded. The **Reset chip at start up:** and the **Reset chip after load file:** radio buttons can disable either of those resets which may be helpful during particularly difficult or unusual debugging circumstances.

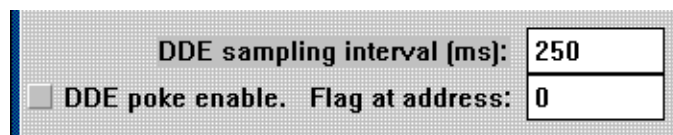


Figure 61: Miscellaneous Setup Dialog Box

Even though the next four fields appear to work, they are not implemented yet. Watchpoints are not yet saved and writes to memory are always read back. There is no DDE support in EMUL51™-PC Windows yet and thus there is no DDE Poke. For information about the status of these features, call Nohau customer support.



Figure 62: Two Features Not Yet Implemented



In some symbol files, symbols are not identified as Program space variables or as Data space variables. If this is true of your file, you may see the EMUL51™-PC Windows software try to set breakpoints in your data address range. To prevent this, check the **Enable code space limits** box and set the **Low** and **High** addresses to encompass just the instructions. Configured this way, EMUL51™-PC Windows will not put breakpoints outside that address range.

Override at Reset	<input checked="" type="checkbox"/> Program Counter	400
	<input checked="" type="checkbox"/> Stack Pointer:	1FFFE

☐ Start address override [at Reset]:

☒ OK ☒ Cancel ☒ Help

Figure 63: Forcing a New Starting Address

When the **Start Address override (at Reset):** box is checked and the field contains an address (in hexadecimal notation), that value will be written to the controller's program counter every time EMUL51™-PC Windows resets the controller. If you have patched your startup code or if you want to only test a certain subroutine, the ability to start at an arbitrary address can be quite helpful.

Window Colors

Under the **Config** menu is the **Color ..** menu item. Open this dialog box to set the colors of different kinds of EMUL51™-PC Windows child windows. For example, all **Program** windows can be set to have a dark blue background with white text to differentiate them from other kinds of windows. At the same time, all **Data** windows can be green with black text, and all **Source** windows set to have white background and red text. It is possible to make each window very distinctive.

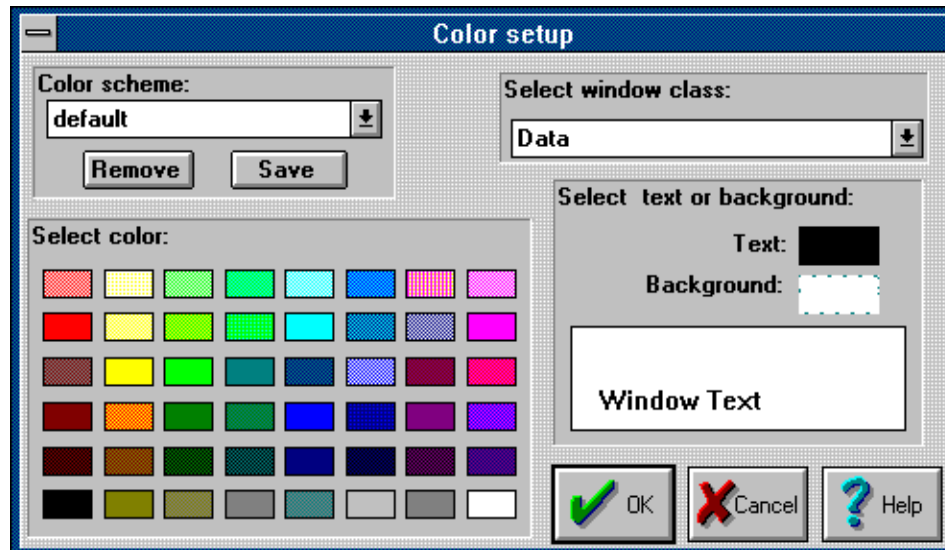


Figure 64: The Color Setup Dialog Box

For each window class that you wish to change, select it from the **Select window class** drop list. While that class name is showing in that field, the colors you select will be assigned to that class of windows.

After you have set all the colors the way you want them, you can name your creation (“color scheme”) by typing the name in the **Color scheme** field and then click on the **Save** button. This color scheme can then be recalled by selecting it from the drop list of color schemes.

Note: Not all combinations of background and foreground colors are possible due to constraints imposed by MS Windows and your video configuration. EMUL51™-PC Windows is constrained by the same limits as MS Windows itself, and is affected by the color palette chosen in the Windows Control Panel program. No matter what colors you select from this palette, the example text pane in this dialog box will show you the colors that will actually be used by EMUL51™-PC Windows.

Trace Config Menu

For information about the **Trace ..** menu item, please refer to “Chapter 3: Trace Board” on page 113 .

Performance Analysis

What portion of your application uses most of the CPU cycles? This is the question that Performance Analysis is designed to answer. You set up address ranges or bins, run your program, and then look at the results to see where (or which bin) the statistics say your program spent the most time.

Performance Analysis is a statistical analysis of execution behavior. Once every second, 12,345 sequential bus cycles are collected, sorted into their respective bins, and the results are displayed on the screen. As you might guess, 12,345 cycles is a rather small percentage of the cycles that occur in one second.

To get more accurate results, run your program for longer periods of time (or at slower clock rates). If you watch the statistics on the screen, you will see them change quickly at first, then more slowly. When they change very slowly, you know that the statistics will probably not get any more accurate.

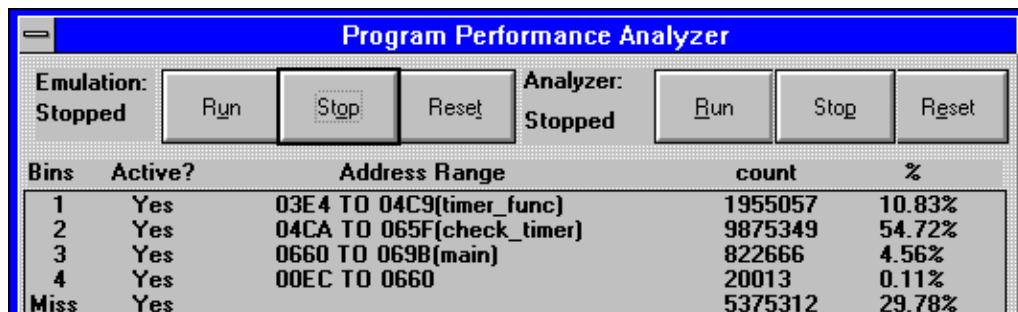


Figure 65: Performance Analysis Control Window

When you select **PPA Analyzer** from the **Config** menu, you will see a control window that looks like Figure 65. The application and the data collection will automatically be started every time you open the PPA control window. The six buttons at the top of the window control the application and the data collection separately. Each bin is really an address range. If the address range corresponds exactly to the address range of a function, that function name will be displayed next to the address.

A fetch from an address that doesn't fall within any address range will be counted in the **Miss** bin. A fetch from within an existing but inactive address range bin will not be counted at all. It will not count in the inactive range and it will not be counted within the **Miss** bin. Statistics will not be kept for that inactive bin at all. The very first time you configure Performance Analysis you will find only one bin: the **Miss** bin, which will count all the bus cycles that do not fall into any of the bins you set up. This bin cannot be deleted, or edited, or be made inactive.



Figure 66: Performance Analysis Control Options

Figure 66 shows the rest of the statistics from Figure 65, plus a few options. This data set took 25 minutes and 42 seconds to collect. A total of 18,048,397 frames were recorded. Figure 66 shows that data read and write cycles were ignored; the **Code only** option is checked. Only instruction fetches are counted in the statistics.

Figure 66 also shows that the **Bargraph** option is turned off. If you prefer a graphic display, you may turn on the **Bargraph** option and see the data displayed in a form similar to that shown in Figure 66.

Bins	Active?	Address Range	count	%
1	Yes		798964	4.68%
2	Yes	(check_timer)	11325917	66.34%
3	Yes	(timer_func)	4156092	24.34%
Miss	Yes		792162	4.64%

Figure 67: Bargraph Display Option

To add bins of your own, double-click on the empty line below the **Miss** bin. In Figure , this is the lighter gray line below the **Miss** bin. Double-clicking on this line will open a dialog box with a list of functions shown in Figure . You can add any of the functions as an address range, or you may create an address range not on the list.

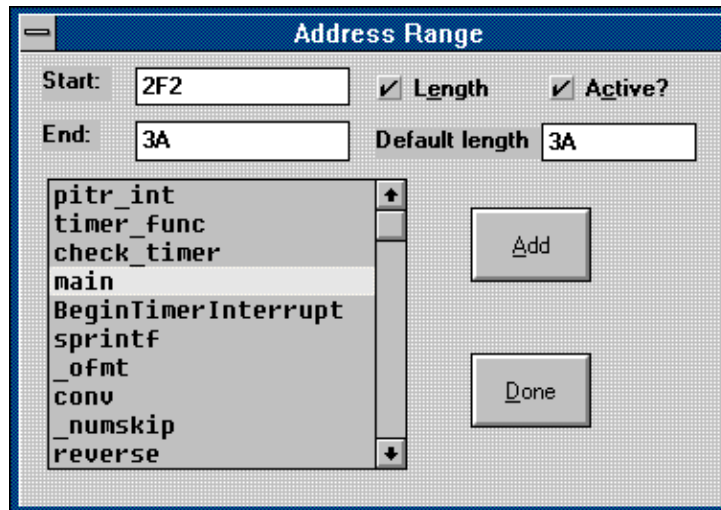


Figure 68: Adding a Bin

To add a bin corresponding to the `main ()` function, double click on **main** in the function list, then click on the **Add** button. Note that clicking once on **main** will highlight it but not update the **Start** and **End** fields with the values for `main ()`. Also note that double-clicking does not actually add the bin. You must click on the **Add** button to actually add the bin. With the **Address Range** dialog box open, you may add as many bins as you like before clicking on the **Done** button to close it.

If you wish to have a bin but not collect statistics for it, remove the check from its **Active** field. The **Length** field controls how the **End** field is used. With a check, the **End** field displays the length (as does default length after **Add**). Without a check in the **Length** field, the **End** field displays the end address in hexadecimal notation.

Using a very similar screen, any bin can be edited by double clicking on that line in the PPA Control window (Figure 65). This is how you activate and deactivate bins.

Once you have collected the data you want, EMUL51™-PC Windows allows you to either save or discard the changes you just made to the list of bins. Only one bin configuration can be saved, not one per project like most configuration settings. This bin configuration will be automatically restored the next time you use performance analysis.

Tool Bar

Just below the menu bar is the “Tool Bar” containing icons or buttons that, like Hot Keys, execute frequently needed menu options when clicked. The **Help** button opens the MS Windows Help application to the page that describes the current context. The **Reset** button resets the controller. The **Step** button emulates one source line or opcode depending upon which window was last active. When the program counter points to a function call, The **Stepover** button will emulate until the highlighted function call returns. The **Go** button starts full speed emulation that will continue until a break occurs. While emulating, the **Go** button changes to **Break**, and halts emulation when clicked. The **Pos** button acts just like typing CTRL-A and will open a dialog box that lets you jump the window to a specified address.



Figure 69: The Tool Bar

The last button affects the **cy** or cycle field at the far right of the speed bar. As you execute instructions, the **cy** field displays the number of clock cycles since the counter was last reset. You can reset the **cy** field either by clicking on the **Clock=0** button, or by resetting the controller.

Dialog Boxes

Many menu selections open dialog boxes that allow you to input more specific information. Some of these dialog boxes are described above next to their menu items. The rest are described in this section.

Child Windows

There are seven primary child windows created by EMUL51™-PC Windows: **Program** windows, **Data** or Memory windows, **Inspect** windows, **Source** windows, a **Registers** window, a **SpecialRegs** window, **Trace** windows (even if you have no Trace board), **Watch** windows, and a **Call Stack** window. All of these windows are opened by selecting the corresponding item in the **Window** menu.

Any number of child windows may be open at the same time. Any number of child windows can overlap but only one child window is active (has the focus) at a time. Some may be scrolled and resized to view any address desired. Their locations and sizes are saved to the current project file when EMUL51™-PC Windows exits, and will be restored when the software restarts.

Each child window has a corresponding menu that appears between the **Config** menu and the **Window** menu. The menu contains items that only make sense within the context of that window. This window-specific menu will also appear at the cursor when you click with the Right mouse button in the body of the active window.

Register Windows

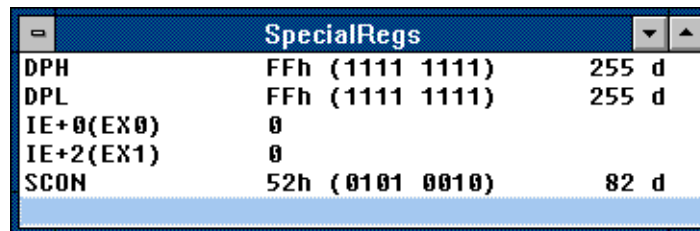


The **Registers** window displays the CPU registers. All registers are displayed in hexadecimal notation except the bottom three: **CAF**, **RS**, and **O-P**, which are fields of the PSW and are displayed in binary form. Clicking anywhere in the **Registers** window will

select that window (make it the active window) and right-clicking brings up the **Registers** menu. The only operation supported in the **Registers** window is editing register contents.

SpecialRegs Window

In addition to the **Registers** window, EMUL51™-PC Windows has a customizable window for special function registers called the **SpecialRegs** window.



SpecialRegs		
DPH	FFh (1111 1111)	255 d
DPL	FFh (1111 1111)	255 d
IE+0(EX0)	0	
IE+2(EX1)	0	
SCON	52h (0101 0010)	82 d

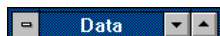
Figure 70: Example SpecialRegs Window

Initially, the **SpecialRegs** window is empty. Menu items in the **SpecialRegs** menu let you add, delete, or edit any special function register defined for the processor you are using. When you select **Add ..** you are shown a list of known registers. Double clicking on one will then add that register to the **SpecialRegs** window. Every time you exit EMUL51™-PC Windows or change Projects, the list of registers in the **SpecialRegs** window will be saved and then restored when that Project is opened once again.

When adding a register, there is also a button to add a single bit from a register to the **SpecialRegs** window. In Figure 70, two bits from the IE register are shown, EX0 and EX1. Because these bits have a special meaning, they can be added to the window just like any 8 bit register.

Note: *Not all special function registers have bits with a special meaning.*

Data Windows



Use **Data** windows to examine or modify emulation or target memory directly. EMUL51™-PC Windows uses the controller to read and write RAM, so the **Data** window cannot be updated while the emulation is running. Instead, asterisks will be displayed until data can be read from memory.

Data can be displayed or modified in a variety of formats as shown in Figure 71. Keep adding new windows for each display format and starting address.

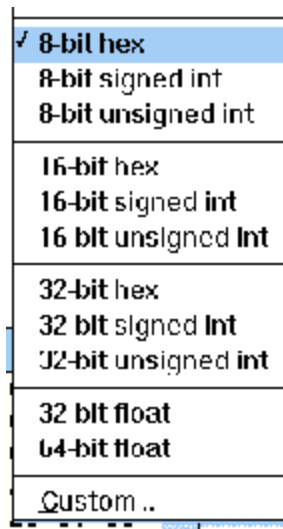


Figure 71: Data Display Formats

Note: 32 and 64 bit IEEE_754 floating point numbers must be word aligned. Some compilers support packed structures that can have floating point fields that start on an odd address. These fields will not be displayed properly in a Data window.

Selecting any **Data** window displays the **Data** menu which supports filling memory, jumping the selected window to a specific address, moving blocks of data, setting an address space, and setting the display mode (hex, ASCII, etc.) options.

Changing a value at any memory location is as easy as selecting the bit, byte, word, or long word to change and then typing the new value.

The first character you type will open a small data entry window, shown in Figure 72.



Figure 72: Editing Memory with a Data Window

To avoid confusion, always enter the new data in the same format that the data is displayed. If the **Data** window is displaying ASCII characters, type the new character (not a string) in ASCII. If the **Data** window is displaying signed integers, enter the new value as a decimal number. Symbols are supported and their type is irrelevant.

If you display the data in bytes, only a byte will be written to memory for each update. In other words, updating one byte uses a single bus cycle that is one byte wide.

On the far right side of the Edit data dialog box is a small check box labeled with the letter **C**. This check box impacts how the emulator interprets the data you enter. If you have a symbol named “abcd” and you are displaying in 16 bit hex, it is not clear whether to interpret “abcdef” as a symbol name or as a hex number. With the box checked, the emulator uses C syntax first, so it will be treated

as a symbol name. Without the **C** box checked, assembler rules apply first, and it will be interpreted as a hex number (see far right edge of Figure 72).

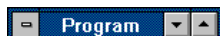
EMUL51™-PC Windows can access 6 address space types: Bit, Data, Xdata, Code, SFR, and NBYT. The first 5 are well known to 8051 users. The NBYT address space is the Nohau Byte address space. Do not use this address space unless directed to by Nohau Customer Support.

The **Bit** address space is displayed in ascending bit order which is different than when bytes are displayed with a binary format. Also, each row in the **Bit** window will contain a multiple of 8 bits and some bits may be obscured by the edge of the window. When editing bits in a **Bit** window, pressing the <Enter> key will change the selected bit. And remember, some bits in the Special Function registers are not implemented or have a special purpose. Those bits may not change when you press <Enter> or they may change state due to external inputs. Many bits above bit 80 also change when the controller is reset, unlike RAM.

Custom Display Format

Selecting the custom format option opens a dialog box that lets you input a C **printf** format string. All standard C formats are allowed, including the newline character. If you are trying to display odd address integers or floating point numbers, you must use the custom display format

Program Windows



A **Program** window disassembles and displays code memory. One line in the **Program** window is always highlighted. This is the cursor. The color of the highlighting and the window depend upon how you have configured your color settings. (See page 84 for information about how to change the color settings.) Use the cursor to set and disable breakpoints, set the program counter, and invoke the in-line assembler.

The first column is the hexadecimal address. If the address is highlighted, there is a breakpoint at that address. You may set or inactivate a breakpoint by clicking on the address. The second column is the hexadecimal value at that address. Between the address and the hexadecimal data may be an arrow pointing to the right, indicating the current program counter. The third column contains the disassembled instructions and operands.

Program windows can control the emulation. To set a breakpoint, click once on the address portion of the instruction where you want the break. Or, you may click once on the desired instruction (to highlight that instruction) and then click on it again to highlight the address. A breakpoint is indicated by displaying the address with white letters on a black or dark background¹. This second mouse click (not a double click) creates the breakpoint. To deactivate (not delete) that breakpoint, click again on the same instruction. The address will no longer be highlighted and the breakpoint will be inactive. To delete the breakpoint, use the **Setup ..** dialog box from the **Breakpoints** menu. Any highlighted instruction can be a temporary breakpoint. The **Run** menu item **Go until cursor** will use the cursor as a temporary breakpoint.

In-line Assembler

The in-line assembler is easy to use; simply highlight the instruction or address you wish to change in the **Program** window and type. The first character typed will open an edit dialog box to display the characters you type and allow you to edit your assembler source line. Once the source line is as you want it, press <Enter>.

The in-line assembler will translate the input line according to the syntax described in the 8051 data books and replace the former opcode(s) and data with the new opcode(s) and data. Note that the assembler will write as many bytes as required for the new

1. The colors used to indicate a breakpoint may be different if you have changed the color scheme as described in the next section.

instruction. This may overwrite part or all of subsequent instructions. Be sure to examine the subsequent instructions as well as the new instructions for correctness.

Source Windows



The **Source** window displays the C source (or assembler source if the assembler supports source line debugging) of the module containing the Program Counter. Like a **Program** window, a **Source** window displays the source text, line numbers, a cursor (the blinking underline), and a small arrow between the line numbers and the source text to indicate the current Program Counter value.

After each single step, and during each animation pause, the **Source** window scrolls to show the source line that generated the instruction pointed to by the new Program Counter, if it was generated by a source line.

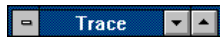
Displaying and toggling breakpoints in **Source** windows is different than in **Program** windows. In **Source** windows, breakpoints are displayed by inverting (or highlighting) the entire source line. In **Program** windows, only the address is highlighted. In **Source** windows, a single click on any line number (or address in the Program window) will toggle the breakpoint. A double-click on the source line itself will also toggle the breakpoint status for that line. In both kinds of windows, pressing F2 will toggle a breakpoint on the highlighted instruction.

When a **Source** window appears blank with the window title "Source", it usually means that the program counter is pointing to instructions derived from a module with no debugging information. As soon as the PC points to an instruction from a C module or assembly module with line number symbols, the **Source** window will show that text, and the title on the window will change from "Source" to the name of the source file being displayed.

The simplest way to find the first line of source is to reset the controller, click on the **Source** window title bar to select it, and then execute a single step by pressing the F7 key (or by clicking on the **Step** button on the speed bar).

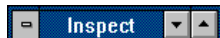
When the **Program** window is selected, a single step means a single opcode. The same is true for animated execution: a pause occurs after every opcode is executed. When the **Source** window is selected, a single step means a single source line. Animation will execute faster when the **Source** window is selected than when the **Program** window is selected because most source lines compile into more than one machine instruction. If the animation is running faster or slower than you expect, or if single stepping executes more or fewer instructions than you expect, visually confirm that the selected window is the one you want to be selected. If in doubt about which window is selected, click on the title bar of the window you wish to be selected.

Trace Window



For information about the **Trace** window, please refer to “Chapter 3: Trace Board” on page 113.

Other Windows



Three more child windows used for high level debugging in C are available: the **Stack** window, the **Inspect** window, and the **Watch** window. These windows are opened by selecting their respective items in the **View/Edit** menu. Like the other child windows, selecting one of these open windows will bring a corresponding menu up between the **Config** and **Window** menus.

Inspect Window

The **Inspect** window displays a single variable, or possibly modifies that variable. To open an **Inspect** window, either select the **Inspect ..** menu item in the **View/Edit** menu or click in the **Source** window on the variable you would like to inspect and type

CTRL-I. Double-click in an open **Inspect** window on a structure member or array element to open an **Inspect** window detailing that field.

The **Inspect** window can stay open just like a **Data** or **Watch** window, and it will be updated whenever the application stops. The variable being displayed may be part of an equation written following the rules of C that produces a single scalar answer.

***Note:** If you have an open **Inspect** window with an assignment statement, every time the emulator stops executing, the expression will be evaluated and the variable will be updated. The variable will appear as though your application is not changing it while the emulator is running.*

Watch Window

The **Watch** window displays multiple variables being watched, one variable per line. Any local variable in the **Watch** window that is not in scope will be displayed with three question marks instead of its value.

Stack Window

The **Stack** window displays the "call stack," or the list of functions called to reach the current point in the application, and the current value of parameters passed to them.

Addresses are displayed and entered using hexadecimal notation or global symbol names. In all windows (excluding **Inspect** windows,) values may be edited by selecting that value (with the mouse or cursor keys) and then typing.

***Note:** Symbol names are case sensitive. If a symbol cannot be found, try the same name with a different case. Also note that some assemblers shift all symbols to uppercase.*

Help Line

At the bottom of the EMUL51™-PC Windows window is a line of text that, depending upon the context, explains what the selected item is or what it does. This kind of context-sensitive help is turned on and off with the **Toggle help line** item in the **Windows** menu.

Menus

The primary means of controlling the debugger, thus the emulation, is through menus. The EMUL51™-PC Windows menus conform to the Microsoft MDI standard. Only those menu items that have meaning or can be used with the current selection will highlight when the mouse is pointing to them. Menus are organized to hide items that are out of context.

Most menu items have "Hot Key" equivalents. That is, there is some combination of function keys, character keys, and modifier keys (Control, Shift, or Alt keys) to select most menu items. The Hot Key for each menu item is shown in that menu to the right of the item name, and are also shown below. Where you see "<Alt>FS" as the keyboard shortcut, you should type <Alt>F (hold the Alt key down while you then press the F key) to open the **File** menu, then press the S key (without the Alt key) to activate the portion of EMUL51™-PC Windows that writes Hex files. Holding down the Shift key or turning on CapsLock is not necessary. Even though the keyboard shortcuts are all shown in capital letters, the shortcuts are not case-sensitive.

File Menu

Load code .	F3	Load an Intel Hex record file or one of the supported object files. See the Accessories chapter for more information.
--------------------	----	---

Save code as ..	<Alt>FS	Write the contents of RAM or ROM to a HEX record file.
Remove Symbols	<Alt>FR	Delete all line number and symbolic information.
Show load info ..		Display a window describing the object file last loaded including number of variables, address range loaded, etc.
Preferences	<Alt>FP	Controls the way the emulator loads object files.

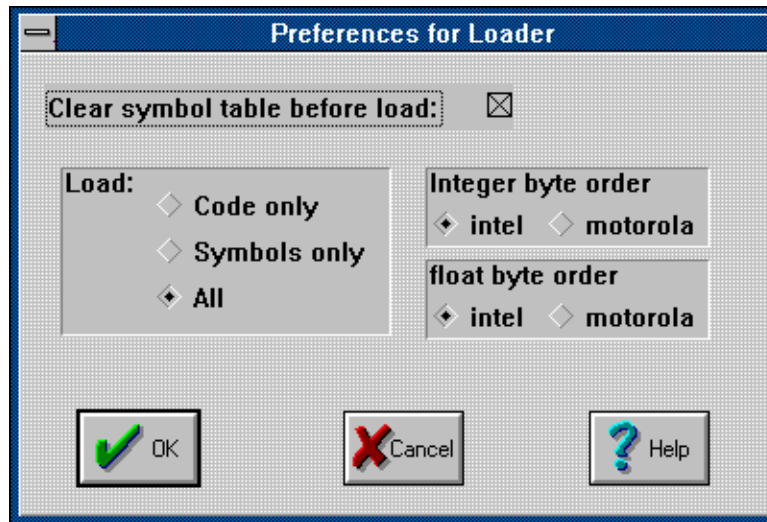


Figure 73: Loader Preferences Dialog Box

Exit	<Alt>X	Quit EMUL51™-PC Windows.
-------------	--------	--------------------------

View/Edit Menu

Copy to clipboard	<Ctrl><Ins>	Copy the text (without formatting or font information) of the entire active window to the clipboard.
User defined symbols	<Alt>VS	This item opens a dialog box that lets you select the module from which you can view symbols.
C call stack ..	<Ctrl>SS	Opens a child window that displays the C call stack and passed parameters needed to reach the current Program Counter.
Evaluate ..	<Ctrl>E	Open a dialog box that evaluates C expressions. Expressions may contain variables. Assignment expressions may change the values of variables.
<i>Hint: To change the value of a variable, use the Evaluate window to evaluate a C assignment expression such as "i=75".</i>		
Inspect ..	<Ctrl>I	Open a dialog box that displays the contents of a single variable, structure, or array in detail.
Add a watch point ..	<Ctrl>W	Open a child window that displays groups of variables that is updated every time emulation halts.

Animate ..	<Ctrl>F7	Execute instructions continuously and slowly, highlighting each instruction or each line as it is executed.
Go	F9	Begin executing instructions from the current PC at full speed until the next breakpoint.
Go to cursor	F4	Execute the instructions from the PC to the current cursor position.
Go to ..	<Ctrl>F9	Execute the instructions from the PC to the specified address.
Go to return address	<Alt>F9	Execute the instructions from the PC to the next found function return. Due to certain optimizations, this feature may not always be available.
Go FOREVER	<Alt>RF	Execute instructions from the current PC after disabling all breakpoints.
Break Emulation	F9	Suspend execution as if a breakpoint was encountered.
Reset Emulator!	<Alt>RR	Load the .STR file to the emulator and the .SYM file into the debugger software.
Soft reset (get vector)	<Alt>RSS	Load the reset vectors into the program counter and stack pointer.

Reset Chip!	<Ctrl>F2	Reset CPU without executing any instructions.
--------------------	----------	---

Breakpoints Menu

Toggle	F2	Disable or enable existing breakpoints.
---------------	----	---

At ..	<Alt>F2	Set a breakpoint by address.
--------------	---------	------------------------------

Setup ..	<Alt>BS	Open a breakpoint editing dialog box.
-----------------	---------	---------------------------------------

Disable all	<Alt>BI	Disable all breakpoints.
--------------------	---------	--------------------------

Special Conditions ..		Configure the SY0 header and the bus cycle type to cause a simple hardware break.
------------------------------	--	---

Delete All	<Alt>BD	Clear all existing breakpoints.
-------------------	---------	---------------------------------

Break now!	<Ctrl>C	Immediately halt the emulation.
-------------------	---------	---------------------------------

Config Menu

Project name ..		Choose a configuration or project from a list of existing projects, or create a new one.
------------------------	--	--

Paths ..	<Alt>CP	Sets the default directories for finding load files, source files, and emulator files.
-----------------	---------	--

Memory map ..	<Alt>CM	Assign memory in 4K blocks to either emulation RAM or the target.
----------------------	---------	---

Banking setup ..	<Alt>CB	Configure the emulator software to expect bank switching, what scheme, bank size, etc.
Emulator Hardware ..	<Alt>CE	Sets the emulator board address, controller type, and emulator memory size.
Miscellaneous ..	<Alt>CM	Sets automatic PC & SP reset value, DDE sampling interval, and memory scroll range values.
Color ..	<Alt>CC	Assign colors to windows.
Trace ..	<Alt>CT	Please refer to page 119 in the Trace Chapter for information about the Trace Config dialog box.
PPA ..	<Alt>CPP	Open a Performance Analysis control window and start recording addresses.

These next eight menus share one location in the menu bar. The menu displayed corresponds to the kind of child window selected. Selecting a different kind of child window will change which menu is displayed. To do this, either use the **Window** menu, or just click the mouse on any part of the desired window.

Program Menu

Address..	<Ctrl>A	Scroll the selected Program window to the specified address.
Origin (at program counter)	<Ctrl>O	Scroll the Program window to display the PC address.

Set new PC value at cursor	<Ctrl>N	Set the Program Counter to the address at the cursor.
Module	<Ctrl>F3	Open a dialog box that allows quickly scrolling the Program window to the start of any module.
Function	<Ctrl>F	Open a window listing all the functions in all modules loaded. Selecting one will scroll the Program window to the start of that function.
View source window	<Ctrl>V	Scroll (or open) a Source window to show the source at the current Program window cursor.
Toggle breakpoint	F2	Enable or disable a breakpoint at the cursor.

Source Menu

Address..	<Ctrl>A	Scroll the selected Source window to the specified address, which may be a function name or a label.
Origin (at program counter)	<Ctrl>O	Scroll the Source window to display the Program Counter address.
Set new PC value at cursor	<Ctrl>N	Set the Program Counter to the address at the cursor.

Module	<Ctrl>F3	Open a dialog box that allows quickly scrolling the Source window to the start of any module.
Function	<Ctrl>F	Open a window listing all the functions in all modules loaded. Selecting one will scroll the Source window to the start of that function.
Call stack ..	<Ctrl>SC	Opens a window that displays the C call stack and passed parameters to reach the current Program Counter.
View assembly code	<Ctrl>V	Scroll (or open) a Code window to the current program counter (not source window cursor).
Toggle breakpoint	F2	Enable or disable a breakpoint at the cursor.

Data Menu

Address..	<Ctrl>A	Scroll the selected Data window to the specified address.
Original Address	<Ctrl>O	Scroll the selected Data window to the last address used in an Address.. menu command.
Edit ..	<Enter>	Alter the contents of the highlighted location.
Block move..	<Ctrl>B	Move a segment of RAM to another location (in RAM).

Fill..	<Ctrl>F	Fill RAM with the specified value or pattern.
Data Display as..	<Ctrl>D	Set the data display mode (ASCII, hexadecimal bytes, long integers, etc. See page93 for the complete list of formats).
Address space ..	<Alt>DA	Set the address space for the selected Data window.

Register Menu

Either select a register then select this menu item, or more simply, select a register and type a new value. The first character typed will open the same dialog box as selecting the **Edit** menu.

Trace Menu

Please refer to Chapter 3 for all information regarding the Trace board and user interface.

Stack Menu

Parameters in Hex	<ALT>SP	Display the function parameters in hex instead of in their declared type.
Show function	<ALT>SS	Not implemented at this time.

Watch Menu

Add ..	<Insert>	Open a dialog box for adding a variable to the Watch window.
---------------	----------	---

Edit ..	<Enter>	Open a dialog box for editing an existing variable in the Watch window.
Remove ..	<Delete>	Delete the selected variable from the Watch window.

Window Menu

The **Window** menu items open new windows, close existing windows, select windows, and arrange windows on the screen.

Open a new window	<Alt>IO	Selecting this menu item opens a sub-menu that lists all of the daughter windows you can open.
Toggle help line	<Alt>IT	Turn on or off text at the bottom of the EMUL51™-PC Windows window.
Repaint	<Ctrl>R	Repaints the screen.
Tile windows	<Alt>IT	Resize and arrange the windows within the EMUL51™-PC Windows application.
Cascade windows	<Alt>IC	Resize and overlap the windows within the EMUL51™-PC Windows application.
Arrange Icons	<Alt>IA	Line up any closed EMUL51™-PC Windows icons at the bottom of the main window.
Zoom	F5	Expand selected window to fill the EMUL51™-PC Windows window.

Next window	<Ctrl>F6	Change the currently selected (highlighted) window.
Local system menu	<Alt>-	Opens the local system menu for manipulating the selected window.
Close	<Alt>F4	Close the currently selected window.

Below the **Close** menu item, there is one menu item for each open window, and the active window will be checked. Selecting one of these items will open the window if it is closed down to icon size, and activate it.

Help Menu

Selecting the **Info ..** menu item will open a box that displays the application version number and date. Please have this information handy when calling for support.

Chapter 3: Trace Board

Introduction

EMUL51™-PC Windows needs RAM to record a history of the data used and instructions executed. The trace board contains this RAM. The emulator board has the logic and connector necessary to support a trace board. The trace board is a full-length ISA-style bus card. It may occupy any slot as long as the ribbon cable can reach from the emulator card to the trace card. The standard trace board includes 48 bits of RAM for each trace record and can hold either 4096 or 16384 records. The detailed installation instructions for the standard Trace board follows immediately below. The advanced trace board is very different and it is described elsewhere. If you only have an advanced trace board, please go directly to page 127 for all information about installing and configuring the advanced trace board.

Detailed Installation Instructions

Power Requirements

The trace board typically requires 1.3A. Before proceeding, check that the PC's 5V power supply is sufficient to deliver the necessary current. If it can't, a larger power supply will have to be installed. Your computer dealer can give you information on where to purchase one.

I/O Address

Each pair of pins in the address header W5 represents one bit in the 10 bit address. Address bits 0, 1, 2, and 3 represent addresses within the 16 consecutive addresses and do not have pin pairs to represent them. This leaves 6 address bits (pin pairs) to set with jumpers: A4 through A9. Shorting two pins represents a 0 in the address. A pair of pins with no jumper represents a 1.

The trace board address jumpers have been factory preset to 100 (HEX). These address settings for a typical system. The table below shows how a typical system uses its address locations. If your system is presently using location 100 (HEX), an alternate address location must be found and the appropriate changes must be made to the jumpers and software.

Hex Location	Typical Use
000 - 0FF	Used by system
1F0 - 1F8	Fixed Disk
200 - 207	Game Adaptor
210 - 213	Expansion Unit
278 - 27F	Parallel Printer Port 2
2F8 - 2FF	Secondary Asynchronous Printer Adaptor
300 - 31F	Prototype Card
320 - 323	Fixed Disk Controller
360 - 36F	Reserved
378 - 37A	Printer Adaptor
380 - 38F	Alternate Binary Synchronous Communications Adapter, SDLC Adaptor
3A0 - 3AF	Primary Binary Synchronous Communications Adaptor
3B0 - 3BF	Monochrome Display and Printer Adaptor
3C0 - 3CF	Reserved
3D0 - 3DF	Color/Graphics Monitor Adaptor
3F0 - 3F7	Floppy Disk Controller
3F8 - 3FF	Primary Asynchronous Printer Adaptor

If the current trace board address conflicts with any other hardware, find free address space between 000 and 3FF (HEX). The trace board requires 16 (or 10 HEX) consecutive addresses. If you change the address and/or memory jumpers, the software settings must be changed accordingly in the “Standard Trace Board Configuration” on page 119.

Addressing Examples

The table below shows several examples of how to set the jumpers on Header W5. Jumper columns are arranged in the same order that the jumpers are physically located on the board.

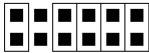
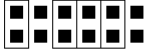





9 8 7 6 5 4	Hex Adr.
	100 - 10F
	110 - 11F
	120 - 12F
	140 - 14F
	180 - 18F
	200 - 20F
	300 - 30F

Figure 74: Trace Addressing Example

Installing the Standard Trace Board

With the address jumpers in place and after you have inspected the boards for any damage, it is time to install the Trace board in your PC. Perform the following steps in the order shown.

1. Turn power off.

<p>Warning: Power must always be off when you plug in the trace board.</p>

2. Insert the trace board with the short ribbon cable connected on to it.
3. Connect the ribbon cable to the emulator board.
4. Make sure the connector fingers of the board(s) are secured into the female connector of the PC's motherboard.
5. Don't forget to put the screws back on the brackets.

Factory Settings

Board jumpers have been preset at the factory prior to shipment. Compare the jumper configurations on the boards you receive against the configurations described in this manual. If a jumper is installed other than as shown, refer to the "Jumper Descriptions" information before changing its position. Jumpers may be installed for operating characteristics required at the time of order.

On the trace board the locations of jumper pins are designated on the board artwork as W1, W2, W3, W4 and W5, (see specified rectangles in Figure 75).

Other Miscellaneous Standard Trace Jumpers

Both the TR4 and TR16 trace boards are shipped in one of two memory configurations. There is either one large memory chip soldered into the bottom row at position U42, or else there are four small memory chips soldered into the bottom row.

The one chip version has the lower jumper, W2, connected. The four-chip version has the upper jumper, W1, connected. Both variations are functionally the same. Since the W1/W2 jumper depends on the memories soldered at the time of manufacture, it does not get changed even if the trace frame size is changed.

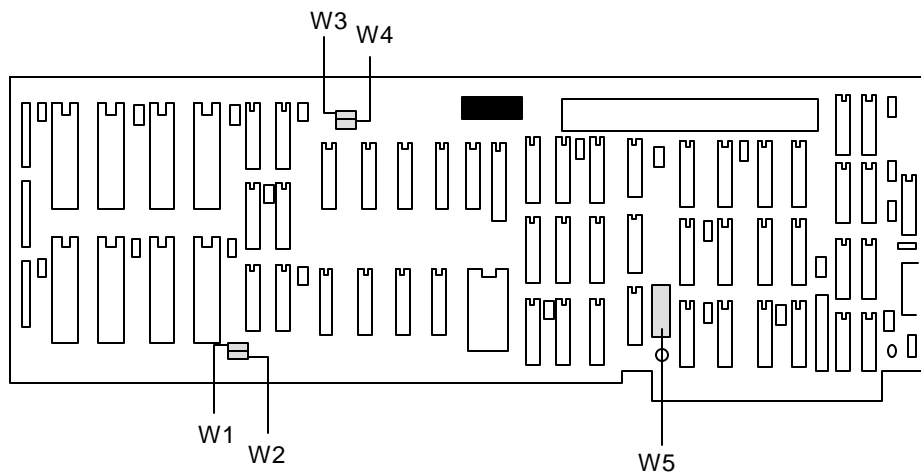


Figure 75: Locations of Jumpers on Trace Board

W1/W2 Setting	U39 - U41	U42
W1	8k x 8	8k x 8
W2	empty	32k x 8

These two factory build configurations both have the same function and size.

4K Trace

Here is the trace board configured for 4k (with 8k by 8 RAM chips) in sockets U3, U4, and U5), and the I/O Port Address is set for 100H (see Figure 76).

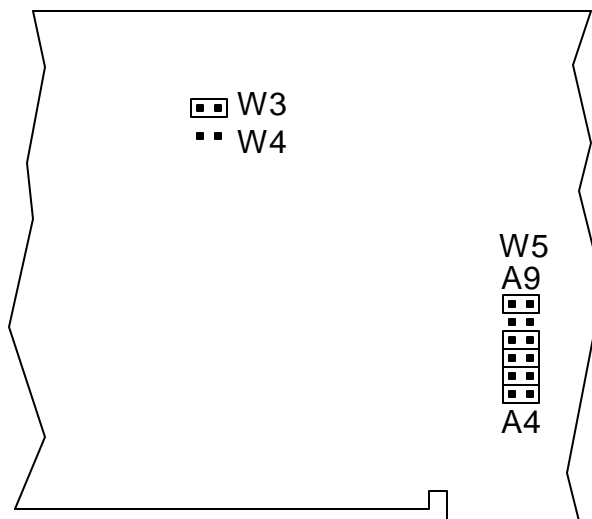


Figure 76: Trace Board with 4K installed, Address 100H

16K Trace

For a 16K trace board, 32K by 8 RAM chips have to be inserted in sockets U3, U4 and U5. Figure 77 shows the jumper configuration, again with I/O Port Address set for 100H.

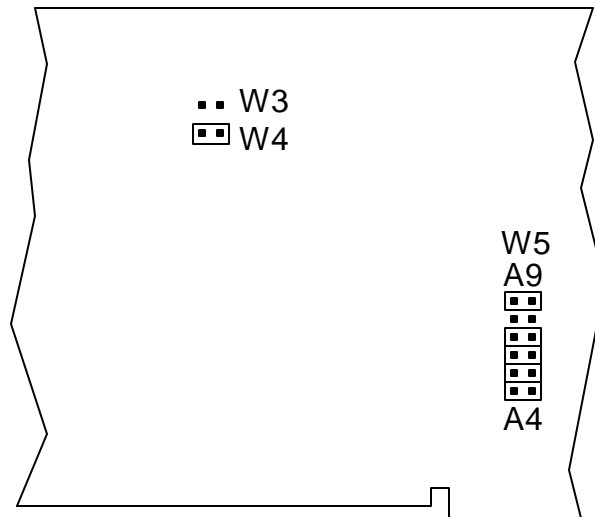


Figure 77: Trace Board with 16K installed, Address 100H

Standard Trace Board Configuration

The trace board makes it possible for you to record what the microprocessor is doing. However, if *everything* is recorded you may end up with lots of information that does not interest you. To prevent this, the EMUL51™-PC Windows Trace facility lets you set up the trace board to record only the information you want. This part of the manual describes how to program the trace board¹ to selectively record frames or cause triggers. By default ALL frames are collected and no frames cause a trigger.

The first field to configure has the buffer size buttons. Select **NONE** if the trace board is not installed. Selecting the **None** button is likely to cause problems if a trace board is present. Selecting a

1. For more information about what tracing is and what triggers are, please refer to "Introduction to Tracing" on page 145.

size other than the size of your hardware is likely to give unreliable trace results. A setting of **4K** or **16K** will display the configuration screen customized for the standard trace board. A setting of **64K** or **256K** will change the dialog box to include fields that are only appropriate for the advanced trace.



Figure 78: Standard Trace Configuration Buffer-Size Buttons

The second field to set is the I/O address field. The value in this field must agree with the address jumper setting on your trace board. If the hardware setting is different from this setting, the trace buffer may look like Figure 79 or show some other erroneous display.

trace buffer: -16279 - 101				
frame#	address	data	instruction	
0:	FF FF	***	processing interrupt	***
1:	FF FF	***	processing interrupt	***
2:	FF FF	***	processing interrupt	***
3:	FF FF	***	processing interrupt	***

Figure 79: No communication with Trace board

Triggering and Filtering

Each machine cycle of the 8051's execution will present different address, data and port information to the trace board. We call the information from one such cycle a "frame." One frame consists of these 48 bits: 16 address bits, 8 miscellaneous signal bits, 8 data bits, 8 bits from P1, and 8 bits from P3.

The largest features in the Standard Trace Setup dialog box are the **A** and **B** lists of conditions. These are lists of values to look for in each frame as it is collected. Each condition in list A or B consists of 48 bits that are either a "1", "0", or "don't care". For each machine cycle executed by the 8051, the trace board compares the 48 bits for the current frame with the conditions in list **A** and list **B**. These comparisons produce boolean results: either a "true" or a "false".

A - conditions:				
ADDRESS	MISC.	DATA	P1	P3
0122,	1xx xx xxxY,	xxxx xxxxY,	xxxx xxxxY,	xxxx xxxxY

Figure 80: The A List of Conditions

Actually each condition consists of five separate fields: **ADDRESS**, **MISC**, **DATA**, **P1** and **P3**. Comparisons are made for each field separately and OR'ed with the other comparisons for that field in the other members of the list. To be true, each "FIELD" comparison has to match exactly with the current data from the active machine cycle. X's denote "don't care," and the corresponding signal can then be either "0" or "1". The "Y" denotes binary notation is used when displaying the value. Values written into the fields without the ending Y are assumed to be hex.

The result for each field is then AND'ed with the other fields (**ADDRESS * MISC * DATA * P1 * P3**). So, if (and only if)

- Any member of the list has an address that matches the current frame, and
- Any list member has a match in the **MISC** field, and

3. One member of the list has a **DATA** field that matches the current frame, and
4. P1 from the current frame matches any one of the conditions in list A., and
5. The same for P3,

THEN the frame will cause a trigger or pass through the filter. This is done independently for both the A and B conditions.

To add or change a condition from either list, click on it (or click on a blank line to add a condition) and then click on the **Add** button. This will open the dialog box shown in Figure 81.

Figure 81: "TRUE" When Instruction is Fetched From 0122H

The address is shown in hexadecimal notation. The other fields, the ones with the "Y", are shown in binary notation. The bits in the **Data**, **P1**, and **P3** fields are the 8 bits you expect. The **Misc** field briefly described in the lower left corner of the dialog box, has 8 bits, comprised of:

- F Bus cycle is first byte of a valid instruction Fetch
- W Bus cycle is a data Write cycle

- R Bus cycle is a data Read cycle
- SY State of the SY pins on the pod
- INT Current INTerrupt level (INT0, INT1 and INT2)

The rest of the Trace Setup dialog box is concerned with controlling what is recorded and when recording (and possibly the application) stops. The Filter and Trigger controls both use the A and B condition lists.

Record Filtering

Filtering is a key feature of the trace board. The **Filter** field controls what frames are recorded by the trace board. Record filtering means that the trace buffer will contain **ONLY** the records you are interested in. You may not need to visually scan through thousands of records to find the one you want.

Figure 82 shows the choices available. You may record all frames, just the frames that match the list of conditions in list A, the frames that match list B, or only those that match both A and B.

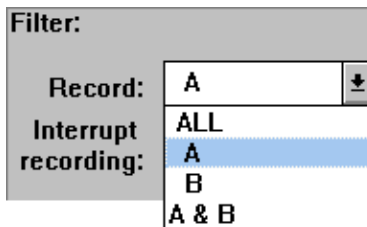


Figure 82: Record filtering

The other field in the **Filter** box controls yet another way to select which frames to record. The **Interrupt recording:** field gives you one convenient place to either record interrupts or not, without

using the A or B condition lists. You may choose between recording every record, just the frames from interrupts (the **INT** choice), or only frames *not* from interrupts (the **MAIN** choice).

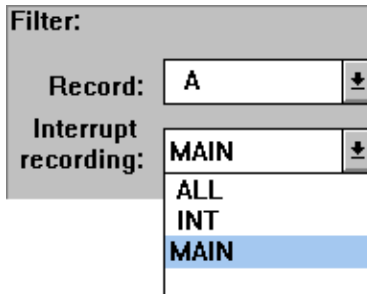


Figure 83: Recording No Interrupt Handlers

The last feature to configure is trace triggering. By default the trigger is off. Clicking on the drop list arrow brings down these additional choices:

A	Trigger on a true A-condition.
B	Trigger on a true B-condition.
A THEN B	First find a true A-condition, then trigger on the next true B-condition.
A LOOP	Trigger when LOOP+1 true A-conditions have been found.
A LOOP THEN B	First find LOOP+1 true A-conditions, then trigger on the next true B-condition.

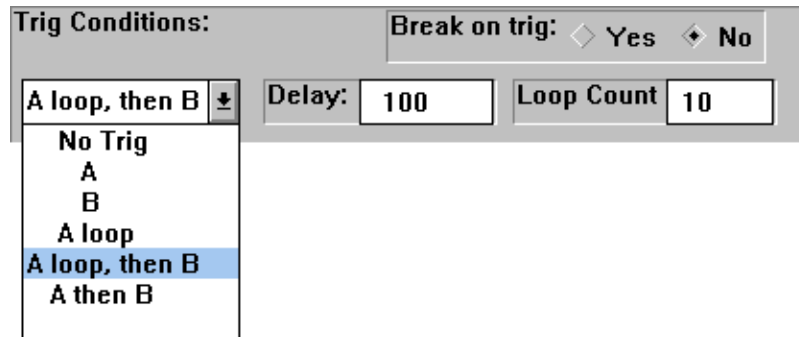


Figure 84: Types of Triggering

The **Delay** field starts counting frames once the trace board has triggered. When the trace board has collected as many frames after the trigger as is shown in the **Delay** field, recording finally stops. When the **Yes** button in the **Break on trig:** box is marked, the trace board will break emulation with the trigger occurs¹.

When all the fields are set to your satisfaction, click on the **OK** button. The emulator will close the dialog box, write the changes to the trace board, and await your next command. If you click on the **Cancel** button, the changes will not be written to the trace board.

This concludes the instructions for setting up the standard trace board. The next section describes setting up the advanced trace board. If you do not have an advanced trace board, you may skip the next section and resume reading about the Trace window and Trace menu on page 145.

1. There may be several bus cycles between the bus trigger and when emulation actually stops.

Advanced Trace Board

Introduction

EMUL51™-PC Windows needs RAM to record a history of the data used and instructions executed. The trace board contains this RAM. The emulator board has the logic and connectors necessary to support a trace board. The trace board is a full-length ISA-style bus card. It may occupy any slot as long as the ribbon cable can reach from the emulator card to the trace card. The standard trace board includes 64 bits of RAM for each trace record and can hold either 64K or 256K records. The detailed installation instructions for the advanced Trace board is described in the following section. The standard trace board is very different and it is described in the previous section, starting on page 113. If you only have a standard trace board, please refer only to that section for installation and configuration information.

Detailed Installation Instructions

Power Requirements

The trace board typically requires 1.3A. Before proceeding, check that the PC's 5V power supply is sufficient to deliver the necessary current. If it can't, a larger power supply will have to be installed. Your computer dealer can give you information on where to purchase one.

I/O Address

Each pair of pins in header J2 represents one bit in the 10 bit address. Address bits 0, 1, 2, and 3 represent addresses within the 16 consecutive addresses and do not have pin pairs to represent them. This leaves 6 address bits (pin pairs) to set with jumpers: A4 through A9. Shorting two pins represents a 0 in the address. A pair of pins with no jumper represents a 1.

The trace board address jumpers have been factory preset to 100 (HEX). These address settings are fine for a typical personal computer system. The table below shows how a typical system uses its address locations. If your system is presently using location 100 (HEX), an alternate address location must be found and the appropriate changes must be made to the jumpers and software.

Hex Location	Typical Use
000 - 0FF	Used by system
1F0 - 1F8	Fixed Disk
200 - 207	Game Adaptor
210 - 213	Expansion Unit
278 - 27F	Parallel Printer Port 2
2F8 - 2FF	Secondary Asynchronous Printer Adaptor
300 - 31F	Prototype Card
320 - 323	Fixed Disk Controller
360 - 36F	Reserved
378 - 37A	Printer Adaptor
380 - 38F	Alternate Binary Synchronous Communications Adapter, SDLC Adaptor
3A0 - 3AF	Primary Binary Synchronous Communications Adaptor
3B0 - 3BF	Monochrome Display and Printer Adaptor
3C0 - 3CF	Reserved
3D0 - 3DF	Color/Graphics Monitor Adaptor
3F0 - 3F7	Floppy Disk Controller
3F8 - 3FF	Primary Asynchronous Printer Adaptor

If the current trace board address conflicts with any other hardware, find free address space between 000 and 3FF (HEX). The trace board requires 16 (or 10 HEX) consecutive addresses. If you change the address and/or memory jumpers, the software settings must be changed accordingly. These settings are described in the section “Advanced Trace Board Configuration” on page 131.

Addressing Examples

The table below shows several examples of how to set the jumpers on header J2. The jumper columns are arranged in the same order that the jumpers are physically located on the board.

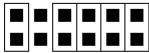
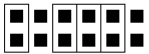
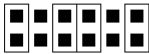
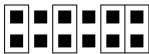
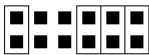
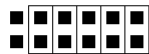
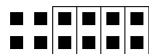
9 8 7 6 5 4	Hex Adr.
	100 - 10F
	110 - 11F
	120 - 12F
	140 - 14F
	180 - 18F
	200 - 20F
	300 - 30F

Figure 85: Trace Addressing Example

Installing the Advanced Trace Board

With the address jumpers in place and after you have inspected the boards for any damage, it is time to install the Trace board in your PC. Perform the following steps in the order shown.

1. Turn power off.

Warning: Power must always be off when you plug in the trace board.
--

2. Insert the trace board with the short ribbon cable connected on to it.
3. Connect the ribbon cable to the emulator board.
4. Make sure the connector fingers of the board(s) are secured into the female connector of the PC's motherboard.
5. Don't forget to put the screws back on the brackets.

Factory Settings

Board jumpers have been preset at the factory prior to shipment. Compare the jumper configurations on the boards you receive against the configurations described in this manual. If a jumper is installed other than shown, refer to the "Jumper Descriptions" information before changing its position. Jumpers may be installed for operating characteristics required at the time of order.

Other Miscellaneous Advanced Trace Jumpers

Header JP1 on the advanced trace board must match the size of the RAM chips. For the ATR64 trace board, make sure that the header on JP1 is on the right pair of pins as shown in Figure 86.

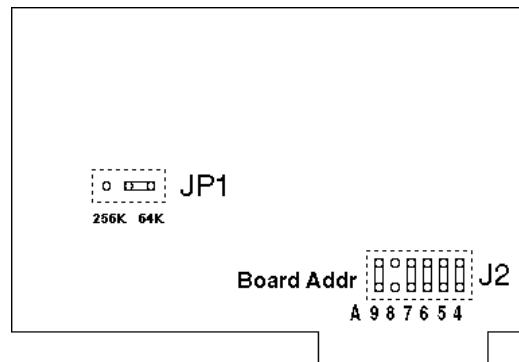


Figure 86: 64K Trace Board Jumpers and Headers

For ATR256, the jumper must be on the left pair of pins:

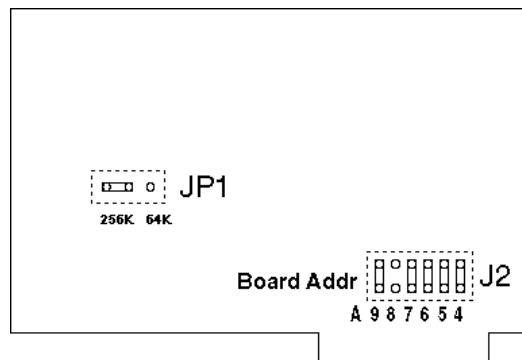


Figure 87: 256K Trace Board Jumpers and Headers

Advanced Trace Board Configuration

The trace board makes it possible for you to record what the microprocessor is doing. However, if *everything* is recorded you may end up with lots of information that does not interest you. To prevent this, the EMUL51TM-PC Windows Trace facility lets you

set up the trace board to record only the information you want. This part of the manual describes how to program the trace board¹ to selectively record frames or cause triggers. By default ALL frames are collected and no frames cause a trigger.

Advanced Trace Board "Program Fields"

These "program fields" define an event or events that control the trace capture. A more detailed description of a "true event" will follow after the description of the "program fields".

The first field to configure are the buffer size buttons. Selecting the **None** button will disable your trace board. A setting of **4K** or **16K** will display the configuration screen customized for the standard trace board. A setting of **64K** or **256K** will change the dialog box to include fields that are only appropriate for the advanced trace.

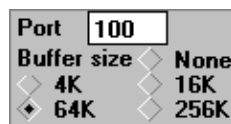


Figure 88: Trace Buffer Size and I/O Address

The second field to set is the I/O address or **Port** field. The value in this field must agree with the address jumper setting on your trace board. If the hardware setting is different from this setting, the trace buffer will look like Figure 79.

1. For more information about what tracing is and what triggers are, please refer to "Introduction to Tracing" on page 145.

Each machine cycle of the 8051's execution will present different address, data and port information to the trace board. We call the information from one such cycle a "frame." One frame consists of these 64 bits: 16 address bits, 8 miscellaneous signal bits, 8 data bits, 8 bits from P1, 8 bits from P3, and a 16 bit timestamp.

Trig

The **Trig** field is either active (checked) or inactive (not checked). The field to the right of **Trig** should be a boolean expression operating on the qualifying registers (A-H), S0 - S5, and the output of the loop counter: CO. The boolean expression may also contain "THEN".

Trigger Examples:

```
A
NOT A
A OR B
A AND B
A THEN B
A AND NOT B
A THEN B THEN C THEN D THEN E THEN F
      THEN G THEN H
A OR B THEN C AND NOT D OR E
A OR S3
A AND CO
A THEN B OR S3
A THEN B THEN CO AND C
```

Delay

When emulation starts, the trace starts tracing automatically. When the trace triggers (according to the conditions in the **Trig** field), it continues to trace for the number of cycles programmed in the **Delay** field. This means that if you have a small number in **Delay**, the trace will stop near the trig point. If you have a larger number,

the trace will continue for a longer time before it stops. The number can be between 2 and 2,147,483,647 (7FFFFFFF hex, in the 32-bit counter). This means that the trig point can be outside the actual trace that you are able to display. The default value of Trig Delay is half of trace memory size (32768 for the 64k ATR board).

Loop Count

This field determines how many times the S3 condition must be TRUE before the CO flag is set to TRUE. CO is an flag that can be used in the **Trig**, **S0-S5**, and **Record** expression fields. The number in this field can be between 0 and 65,535. The default is 0. For more information, read the section titled "Loop counter condition S3=" on page 138.

Break Emulator

This set of radio buttons controls when the trace board halts emulation. **No** means that the trace will not affect emulation. **On Trig** means that a breakpoint will be forced when the trig occurs. **When done** means that a breakpoint will be forced when the trace stops recording, therefore the break will occur **Delay** frames after **Trig**. **On S2** means that the break will occur when **S2** becomes TRUE.

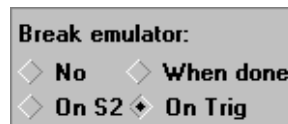


Figure 89: Advanced Trace Break Options

Record

The **Record** field can either be **ALWAYS**, **IF** or **SRC ONLY**. The field below the of **Record** label must be a boolean expression operating on the events A-H, S0 to S5 and CO. If **ALWAYS** is chosen, all frames will be recorded. If **IF** is chosen, only frames that are TRUE according to the boolean expression will be recorded. If **SRC ONLY** is chosen, only frames that have a corresponding line number will be recorded. When the trace is displayed later, high level statements from the source file can be shown. Since only one frame for each statement is recorded, up to 65535 source lines can be recorded! (262,143 if you have a 256k trace.) The "THEN" operator is not allowed here.

Examples:

```
A
A OR B
A AND B
S3 (where S3 could be SET A CLEAR B to
    record everything between two
    events).
```

Filter Delay

When **Record** is in **IF** mode, frames will not be recorded when the boolean expression is FALSE. However, in some cases you may want to continue to record a number of frames after **Record** goes from TRUE to FALSE. This is done by entering a number between 0 and 15 in the **Delay** field next to **Record**. This is particularly helpful when recording MOVX instructions. The third frame of a MOVX instruction emits an address to the location where XDATA is read or written. This address can be outside the **Record** expression, but if you have entered a 1 in the **Delay** field the read / write external data will be recorded. Another use of this feature is to continue to trace after conditional jumps in the code.

Time Stamp Prescaler

The time stamp mechanism consists of two 16-bit counters. Only the 16-bit value from one counter is recorded in the trace. The second counter is the prescaler. If the prescaler is **NONE**, you have full accuracy, which has a resolution of half a cycle. Therefore, you can count up to 65,535 "half cycles" or 32,767 cycles before the counter rolls over to 0. With a 12-MHz crystal, this is equivalent to 32.767 ms. With the prescaler programmed to 2, you would be able to count double the amount of time, or 65,535 ms, but with half the resolution; i.e., 1 cycle or 1 μ s. With the prescaler at 65,535, you would count 65,536 x 65,535 half cycles or for approximately 2,147 seconds (or 35 minutes) at the resolution of 32 ms. The default is **NONE**. Integers between 1 and 131070 are accepted. The prescaler can only use *EVEN* integers. For example, if you enter a 3 the program will round down the number to 2 (1 to **NONE**, 5 to 4, 99 to 98, etc.)

Time Stamp Overflow

This field can be either off (unchecked) or on (checked). When it is on, extra frames will be recorded when the time stamp counter rolls over from FFFF to 0. This will happen even when the **Record** field has determined that no frame will be recorded. In other words, a frame is saved in order to collect a time stamp at the rollover point. If no other frames are recorded (due to the **Record** field), 64k of "overflow" frames can be recorded (256k with the ATR256).

Let's say that you want to find out how long it takes between two addresses. Assume that the first address was not encountered again until the second address was encountered. With full resolution (**Prescaler** = **NONE**), you would be able to record up to 65,535 - 2 "overflow" frames. Since the time between two "rollovers" at full resolution (prescaler = none) is about 32ms (at 12MHz), up to 35 minutes (32 x 65533 / 1000 x 60) can be recorded with a resolution of half a cycle (500 ns). With the 256k ATR, this can be four times more, or almost 2.5 hours. With less resolution you could go for years!

Since it takes a while to calculate the time stamp, there will be a delay of several seconds between issuing a trace display command until the display actually appears on the screen.

State Flags S0 - S5

Boolean expressions

A boolean expression consists of a number of "Boolean operators" working on "Boolean operands". The Boolean operators are:

AND, OR, NOT:

AND between two operands means that both must be TRUE for the result to be TRUE.

OR between two operands means that one must be TRUE for the result to be TRUE.

NOT before an operand means that the operand must be FALSE for the result to be TRUE, or the operand must be TRUE for a result to be FALSE.

THEN between two operands means that the first operand must come true first, then the second operand follows sometime thereafter. This command is only allowed in the **Trig** and the **Record** fields.

SET-CLEAR This function will allow you to tell the trace when to start recording on a given condition (SET), and then to stop recording (CLEAR) on a different condition.

The hierarchy of the boolean operators is as follows:

NOT, / Highest

AND

OR, THEN

SET-CLEAR Lowest

***Note:** The SET-CLEAR function will have a delay of a single memory cycle when the SET becomes true until it is actually recorded in the trace buffer. To capture the actual SET frame or first byte of the operation, you need to use something like the following example:*

```
RECORD: Yes, if S0 or A
S0 = Set A Clear B
```

Cycle Count Enable S5=

This field consists of a <boolean expression> of <events>, S0 to S5 and CO. When the expression is true, a 32 bit counter is enabled to count cycles. This can be used to measure cycles or time "on the fly". This is displayed during emulation in the Trace Speed Bar (described later in this document). On PODS with the FLF pin, the S5 state is available. It can also be found on pin 22 of the 50 pin ribbon cable connector.

POD Signal "ANB" S4=

This field consists of a <boolean expression> of <events> and S0 to S5 and CO (loop count). When the expression is TRUE, the ANB pin of the POD board is low. (ANB is not available on all PODs, but this signal can be found on pin 21 of the 50 pin ribbon cable connector.)

Loop counter condition S3=

This field consists of a <boolean expression> of <events>, where an event is one of **S0** to **S5** and/or qualifiers A-H. When the condition goes from FALSE to TRUE, the loop counter will count down.

Note: Examples using all these fields start on page 142.

S2=, S1=, S0=

These fields can only be used if the **THEN** operator is not used in the **Trig** field above. If **THEN** is used, these fields will implement the "state machine" to handle the **THEN**. If **THEN** is not used in the **Trig**, **S0 - S2** can be freely employed as boolean expressions utilizing events and S0 to S5 and CO (loop count). You could then implement your own "state machine" of up to eight "states". If you don't use **S3 - S5** for loop counter or pod signals, they can be employed for other conditions that you wish to set up. In this case, you could implement a state machine of up to 64 states.

Conditions A through H

The last fields to describe are the individual condition fields **A** through **H**. Each field is actually a pair of fields that each contain bits to compare to the current trace frame. Once the comparison is made, the trace uses the result to control triggering, filtering, cycle counting, or loop counting, depending upon where that condition is used in the previously described boolean expressions.

ACTIVE?		ADDRESS	&	FWR SY INT	&	DATA	&	P1	&	P3
<input checked="" type="checkbox"/>	A	xxxx xxxx xxxx xxxY	01	xx xxxY		xxxx xxxY		xxxx xxxY		xxxx xxxY
<input type="checkbox"/>	OR	xxxx xxxx xxxx xxxY	xx	xx xxxY		xxxx xxxY		xxxx xxxY		xxxx xxxY
<input checked="" type="checkbox"/>	B	0511H		xx xx xxxY		xxxx xxxY		xxxx xxxY		xxxx xxxY

Figure 90: Control Conditions A through H

A condition without a check mark is not active and will return a FALSE to any of the boolean expressions where it is used. The box must be checked for the condition to be active. The rest of the fields are represented by either an "x", a 0 or a 1. If a bit has an x then that bit in the condition is not compared to the corresponding bit in the current frame and so it is a "Don't Care" condition.

Condition Fields

Active?	A check mark means the condition will be evaluated for each frame
Address	Used to specify an address, address range, or a wild card address to qualify the capture for the trace buffer.
FWR	This field is employed to specify if the address is a valid fetch (F), write (W), or a read(R). _ F = 01 _ R = 10 _ W = 11 _ None of the above = 00
SY	This column is used to specify logic HIGH (1) or Low (0) on the SY1 and SY0 pins, respectively.
INT	This column specifies the interrupt level. 000 interrupt level 0. (Main) 001 interrupt level 1. 010 interrupt level 2. 011 interrupt level 3. 100 interrupt level 4. 101 interrupt level 5. 110 interrupt level 6. 111 interrupt level 7.
DATA	This column specifies the data value on the bus.
P1	Here you specify the value on the probes labeled P1 (by default Port 1). These pins on the POD can also be removed and connected to points on the target system.
P3	This field is like P1, except for bits 6 & 7, (which on most PODS are for the inputs E0 and E1, respectively).

The lower 8 bits of the address are sampled on the trailing edge of ALE. The higher 8 bits of the address and P3 are sampled on the leading edge of PSEN, READ or WRITE. P1, data and the 8 miscellaneous signals are sampled on the trailing edge of PSEN,

READ or WRITE.

Each control condition is actually two sets of bits that are OR'ed together, field by field. After both sets of fields have been compared, if all five of the pairs of fields result in a TRUE, the entire condition results in a TRUE which is then used when evaluating all the boolean expressions that use the control condition.

If a field contains a 1 or a 0, the corresponding bit in the current frame must match for that portion of the condition to be TRUE. The one exception are the bits in the **FWR** field. This field contains two bits but represents three bits in the current frame. The encoding is described in the lower left corner of the **Trace qualifier** dialog box, which is shown in Figure 91.

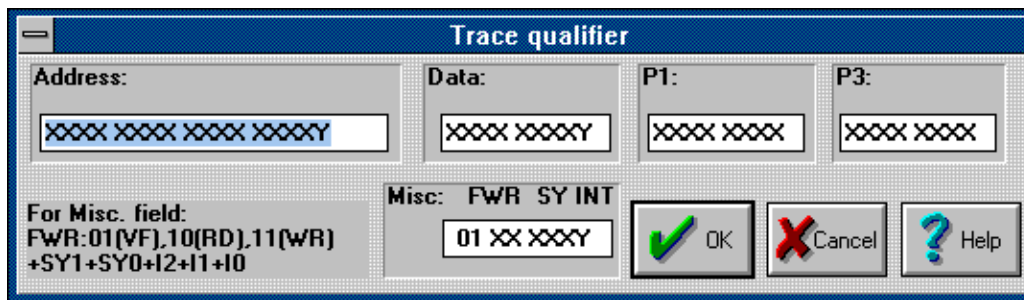


Figure 91: Trace Qualifier Dialog Box

In the above example, this qualifier will be TRUE when the frame is the start of a legitimate opcode fetch from any address. A **FWR** value of 10 becomes TRUE during the frame data READ cycle. A **FWR** value of 11 indicates a data WRITE cycle. A value of 00 responds to none of the above, i.e. operand fetches or blind prefetches.

Using the Sx Flags

Here is an example showing how **S2 - S0** can be used as a three bit counter (8 states) to count up each time the A Event equals a First Opcode fetch (FO) at address 200H. In this case, we use the result of **S2 - S0** to trigger the trace when **S2, S1** and **S0** are all TRUE. You can expand this example by using more events (A - H) in the equations. The result could be used to turn on and off the **Record** field.

Use TEST.A03 where address 0 is the first address in the interrupt service routine. The trace setup fields are left in their default values except the following:

S2=	S2 and /(S0 and S1 and A) or /S0 and (S0 and S1 and A)					
S1=	S1 and /(S0 and A) OR /S1 and (S0 and A)					
S0=	(S0 and /A) or (S0 and A)					
ACTIVE?	ADDRESS	&	FWR SY INT	&	DATA	&
✓ A	200		01 xx xxxY		xxxx xxxxY	xx

Figure 92; 3-Bit Counter Example

Each expression can be much longer than will fit in the visible portion of the field. The left and right arrow keys will move the cursor to areas not visible, and the hidden portion will be scrolled into view as necessary. The complete expression in S2 is

S2 and /(S0 and S1 and A) or /S0 and (S0 and S1 and A)

Figure 92 also shows the value to be used in control condition A, with the unseen bits set to "x" or "Don't Care".

The expressions in **S0**, **S1** and **S2** could have been written using the boolean XOR, but the XOR is not implemented. The equivalent expressions are:

S2=S2 xor (S0 and S1 and A)
S1=S1 xor (S0 and A)
S0=S0 xor A

Using SET and CLEAR with the Sx functions:

S0= SET A CLEAR B

This means that **S0** will be set to TRUE when **A** becomes TRUE. It will then stay TRUE (even if **A** becomes FALSE) until **B** becomes TRUE at which time it becomes FALSE. A and B in the example above may be boolean expressions. This can be used in a number of ways but here we will show an example where it is used to create a "window" for recording.

The trace setup fields are left in their default values except the following:



Record IF Delay 0

S0 OR A

Figure 93: Set and Clear Example Record Field Setting

S0=		SET A CLEAR B					
ACTIVE?		ADDRESS	&	FWR SY INT	&	DATA	&
✓	A	10C		xx xx xxxY		xxxx xxxxY	xx
	OR	xxxx xxxx xxxx xxxxY		xx xx xxxY		xxxx xxxxY	xx
✓	B	10E		xx xx xxxY		xxxx xxxxY	xx
	OR	xxxx xxxx xxxx xxxxY		xx xx xxxY		xxxx xxxxY	xx

Figure 94: Other Set and Clear Example Settings

The reason **A** must be in the **Record** expression is that the result of **SET A** will appear in **S0** delayed for one frame, otherwise you would miss address 10C.

The outcome is that all activities taking place between execution of address 10C and address 10E will be recorded. This is obviously different from saying that all addresses between 10C and 10E will be captured. If, for example, there is a subroutine call between 10C and 10E, it will be recorded even if the addresses of the subroutine are outside the range.

If SET / CLEAR is used under **S5**, you get a convenient way of measuring exactly how long it takes to execute a certain portion of your code!

This example is shown just to help you see how the state machine can work as a counter, and is an expansion of the previous example with the three bit state machine. There is no reason that you really need to analyze how each of the equations in the (S) registers affect the other.

S5 is also exercised, so when S5 becomes 1 the cycle counter will start to count. Since the delay in TW2.HEX is about one second, it will take about 64 seconds for all six states to all become 1. Half of this time S5 is 1, therefore the cycle counter will reach 32,xxx,xxx before trigger occurs. (TRIG is AND'ed between all six "S" bits.)

Introduction to Tracing

A trace history is a time ordered recording of bus cycles (with some other helpful information). Events that do not affect the CPU external bus, such as testing a CPU internal data register, will not get recorded. Events that do affect the bus will only get recorded if the "recorder" is turned on and set up to record those types of events. All tracing emulators record bus events and not actual instruction execution, so they all must have some way to deal with the effects of the instruction pipeline. The trace board for EMUL51™-PC Windows includes pipeline decoding and marks opcode fetches that are not executed. As a result, the display software can show the trace records as though the pipeline did not exist, but it can also display the uncorrected bus cycles just as they were recorded.

Tracing starts automatically every time emulation starts. Even single-stepping will turn on the trace recording during that step. Clicking on the **Trace Beg** button or pressing the **F10** key will also start recording (but until emulation starts, there will be nothing to record). Once trace recording has started, the **Trace Beg** button changes to the **Trace End** button, and will stop recording when clicked. The trace buffer will continue to collect records until recording is stopped, either by a trigger, by stopping emulation, by pressing the **F10** key, or by clicking on the **Trace End** button.

Once emulation has started and bus cycles are "being recorded," every bus cycle is examined to see if it meets the conditions in the **Filter:** field of the **Trace Setup** dialog box. If it does, then it will be recorded. If it does not, that bus cycle will not be recorded in the trace buffer. Bus cycles that are not the correct type (opcode fetch, data read, or data write), or that fall outside the address range(s) specified in the **Filter:** field, will be examined to see if they meet any trigger conditions but will not be added to the buffer.

Every time tracing starts the buffer is cleared. After recording a single step, the trace buffer will only contain the records for that one instruction or source line. As long as trace recording continues,

records will be added to the buffer. Once the buffer is full, the new records will begin to overwrite the oldest records. The trace buffer is a ring buffer that will continue to collect new records and replace old records until recording is stopped. Triggers without an address qualifier will be made inactive.

Triggers and Hardware breakpoints

The trace board can do more than just record what happens on the controller bus. A "trigger" can occur when certain conditions on the bus are met. For example, you can program a trigger to occur when the instruction at 4FE Hex has been fetched for the fourth time. Triggers can stop trace buffer recording, and can cause breakpoints. These are useful if you are executing out of ROM or need to break on certain hardware conditions. For information about how to create standard trace board triggers and hardware breakpoints, see the section titled "Triggering and Filtering" on page 120. See "Advanced Trace Board "Program Fields"" on page 132 for information about how to trigger the advanced trace board.

Trace Speed Bar

Like the speed bar that lets one click start emulation, reset the controller, etc. there is a speed bar for the trace board features. There are two buttons and 8 values displayed in the trace speed bar for your convenience.

Done	Trace	Setup	Not Triggered	frames	loop count	trig delay	state	0	cy
				0	0	10	000000	0	us

Figure 95: Trace Speed Bar

From left to right, the first field displays the current trace status which is either **Tracing** or **Done**. The **Trace** button, which is labeled **Stop** when the trace is recording, starts and stops tracing. The **Setup** button opens the **Trace setup** dialog box. The field displaying **Not Triggered** in Figure 95 displays the trigger status as

the trace board is recording. Next to that, the **frames** field shows the number of frames currently captured. The **loop count** field and the **trig delay** field also display the current state of those counters on the trace board. The **state** field shows the current state of the **S0** through **S5** flags and is only accurate if you have the advanced trace board.

Trace Window

The contents of the trace buffer are displayed in the **Trace** window. If there is no **Trace** window open, you may open one using the **Window** menu item and selecting **Open a new window > trace buffer**. Most of the **Trace** window features are controlled by the trace menu, and are described in the **Trace Menu** section below. Please refer to both this section and the Trace Menu section for a complete description of the **Trace** window.

trace buffer: -26873 - 1004			
frame#	address	data	instruction
-5:	EC	E0	MOUX A,@DPTR
-4:	ED	FF	
-3:	2	00	
-2:	ED	FF	MOV R7,A
-1:	EE	A3	
0:	EE	A3	INC DPTR
1:	EF	E0	
2:	EF	E0	
3:	EF	E0	
4:	EF	E0	MOUX A,@DPTR
5:	F0	90	
6:	3	01	
7:	E0	00001B	MOVL DPTR,#0001B

Figure 96: A Basic Trace Buffer Window

As you can see, the first or left-most column displays the frame number. These numbers can range from 1-<buffer size> to <buffer size>. Frame 0 always represents the trigger frame. If there was no trigger, frame 0 will be the last frame in the buffer.

The second column, the **address** column displays the address of that bus cycle in hexadecimal notation. The **data** column, as you would expect, displays either one or two bytes of data from the displayed bus cycle.

Instruction Size

By default, multiple bus cycles that fetch a single opcode are displayed on one line, as if the bus were always big enough to fetch the entire instruction in one bus cycle. In this display mode, you will see that some frame numbers will be missing. If the display mode is set to **Show all frames**, the frame numbers will be in sequence and no frames will be missing. The disassembled instruction (opcode and operands) will be displayed with the first trace frame for that instruction and the other fetches needed to completely fetch the instruction will be displayed in the trace records that follow. For example, see record number 67240 in Figure 95.

Trace Menu

Like the other window-specific menus, the **Trace** menu only appears when the trace window is selected. The **Trace** menu contains items that control how the trace is displayed.

Toggle trace speed bar!

Done	Trace	Setup	Not Triggered	frames	loop count	trig delay	state	0	cy
				0	0	10	000000	0	us

Figure 97: Trace Speed Bar

The speed bar just below all the menus has two halves. The top half controls the emulator and the lower half controls the Trace board. (See Figure 97) If you find you do not use the trace half of the speed bar, you may hide it by selecting this Trace menu item. Selecting this menu item when there is no check mark will restore the trace half of the speed bar.

Go to beginning of trace buffer

(See next paragraph.)

Go to end of trace buffer

These two menu items do exactly what you think they do. "Go to the beginning" means jump the cursor to the frame with the least positive or most negative frame number. "The end of the trace buffer" means the most recently recorded frame, or the last frame recorded before recording stopped.

Find Frame number ..

When this menu item is selected, it opens a dialog box to get the desired frame number. Once the trace buffer has records, this menu item scrolls the trace window to the record entered in the dialog box.

Search Address ..

This menu item opens a dialog box, shown in Figure 98, to get the desired address, then searches from the beginning in the frame buffer for the first record that {bmc manual\graphics\tr_srch.bmp} matches the specified conditions. The **Data, P1, P3, SY, and INT** fields are all displayed in binary form. A lower case "x" represents a "don't care" condition for that bit. The "Y" is a reminder that the value is displayed in a "binaryY" form.

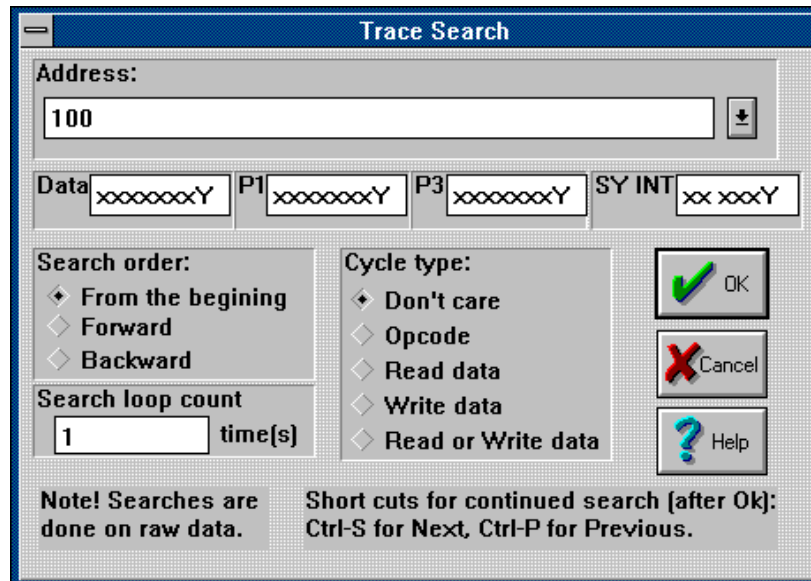


Figure 98: Trace Search Dialog Box

By default, the search starts at the first (oldest) frame in the buffer (not necessarily frame 0). By selecting options in the dialog box, you can choose the search direction and limit the search to only certain kinds of bus cycles. Every condition is AND'ed with the other conditions. A frame must meet all the criteria to be found using this search feature. To search for conditions regardless of the address, leave the address field blank.

Search Next Address

From the current frame, this menu item searches forward for the next occurrence of the last conditions searched. If a search has not yet been specified, no frame will be found.

Search Previous Address

From the current frame, this menu item searches backward for the next occurrence of the last conditions searched. If a search has not yet been specified, no frame will be found.

Find Trig point

This menu item will scroll the **Trace buffer** window to show frame 0, which is the trigger point.

Save trace as text ..

With this menu item, you can save any portion of the trace buffer to a text file suitable for inclusion in documents or processing with text manipulation or word processing tools. Selecting this menu item will open a dialog box that lets you set a range of frames and the name of the file where the text goes.

The file text will be formatted in the same manner and with the same options as the text in the **Trace buffer** window. If you want the text file to include timestamps, arrange for the **Trace buffer** window to show them as well.

Save trace image to file!

Occasionally, you may want to preserve the contents of the trace buffer for examination later. This menu item lets you do just that. Selecting this menu item will create (or overwrite) a file called TRACEBUF.SAV in the directory with the EMUL51™-PC Windows software. The name is not changeable and the entire contents of the buffer are saved, every frame, every bit. The display options have no effect on this feature.

Note: The size of the saved buffer must not exceed 64K bytes or about 1300 frames for the standard trace and 1000 frames for the advanced trace. If it does, you will see an error message and the buffer will not be saved.

Show trace image from file

Selecting this menu item will load the file TRACEBUF.SAV in the EMUL51™-PC Windows software directory into the trace buffer, erasing the previous contents. From that point you may do anything with the trace buffer you can do normally after filling the buffer.

Show misc data

The miscellaneous data columns consists of 8 bits displayed in this order, from left to right. The first three bits: Valid Fetch, Write to external memory, Read from external memory, are displayed beneath the **FWR** column heading and because they are mutually exclusive, are displayed as either an **F**, a **W**, or an **R**. Beneath the **SY** column heading are two bits that display the state of the two SY test points on the pod, SY0 and SY1. You may connect these test points to any signal you wish to record in these two bits. Each bit is displayed as either a 1 or 0. The next column to the right, under the **INT** column heading, displays the interrupt level. The remaining 3 bits record the state of the three interrupt level bits, INT0, INT1, and INT2 from the processor. They are displayed as a single number - either a 0, a 1, or a 2.

Show ports

With a check next to this menu item, the **Trace** window will show, in binary form, the 16 bits found in ports 1 and 3 with each frame. Most pods have a two-row header with one pin pair for each of these bits. This header lets you remove the jumper and record in the trace buffer, any signal you wish in these 16 bits. Some pods have

14-pin pairs for these 16 bits, leaving out P3.6 and P3.7. Please see the section in the PODS chapter that corresponds to your pod for more details about these header pins.

Show P3.6 and P3.7

Almost all pods have two test points labeled E0 and E1. Selecting this menu item will add those two bits to the Trace window display. You may connect any signals to these two test points. If your pod has 6 pins for port 3, you may want to connect these two test pins to P3.6 and P3.7.

Show all frames

Without a check mark, all the bus cycles that occur to fetch an opcode and its operands are condensed to make one instruction per line, even though fetching some instructions actually required 2 or more bus cycles. With a check mark, every bus cycle is shown on its own line, even if it was a part of an instruction fetch.

Show timestamp

***Note:** Only the advanced trace board has a time stamp field. Selecting this menu item when using the standard trace board will have no effect. The Trace window format will not change.*

The **Show timestamp** menu is a hierarchical menu with 4 sub menus: **Absolute cycle**, **Relative cycle**, **Absolute time**, **Relative time**. Selecting any one of these adds a column of numbers that show the time stamp associated with each frame.

Absolute cycle

Every time stamp represents the number of instruction cycles¹ that has occurred since recording was started.

Relative cycle

When displaying the relative number of cycles between successive frames, the top line in the **Trace** window shows the cycles that have elapsed since recording started. All other frames show the number of clock cycles that have occurred since the preceding frame.

This feature is especially useful for timing loops or response latencies in or your application, or how long it takes to execute a particular piece of code. Set up the trace board to record very specific events like the instruction fetch that marks the beginning and end of an even you want timed, and nothing else. Then when you display the trace buffer, the number you want will be displayed. No calculations will be needed, and the timing will be as accurate as your target clock.

Absolute time

With this sub menu checked, the time stamp column will display the elapsed time since recording started for each frame, similar to the Absolute cycles option. In fact, it is the absolute number of cycles divided by the clock frequency set in the Trace Config dialog box.

1. A 12 MHz clock will result in 1 million NOP instructions per second and two million ALE signals. Thus, succeeding bus cycles will be shown as only 1/2 of an instruction cycle apart if the prescale is set to **NONE**.

Relative time

For each frame, if you take the **Relative cycles** displayed and divide by the number of cycles-per-second, you will have the relative time for that frame.

Synchronize program window

When this menu item is checked, as you move the cursor around the **Trace** window from opcode cycle to opcode cycle, the cursors in the **Program** and **Source** windows will also move to point to the instruction fetched and it's context. If the application is running, only the **Source** window will scroll.

Filter: When addressing 16-bit wide memory, triggering on a single byte WRITE to an odd address occurs on the high byte on the data bus. To set up a trigger for this condition, set the mask to FF00.

Note: *Future versions of the software will make this feature more intuitive.*

Index

Symbols

.ext field 78
.STR files 21, 79, 82, 104
.SYM files 21, 79, 82, 104

Numerics

8031.STR 21

A

Absolute cycle 154
Absolute time 154
Add .. 109
Add a watch point 102
ADDRESS 121
Address
 Jumpers 19
 Space 30
Address space .. 109
Address.. 106–108
Addressing 19
 Header 129
 Trace board 115, 129
Advanced Trace Board
 Jumpers and Headers 131
Animation 98
Arrange Icons 110
At .. 105

B

Bank

 Address range 30
 Number 34
BANKADDRESS command
 Replacement of 29
BANKBYTE command
 Replacement of 29
Banked area 31
Banking setup .. 106
Bankswitching 22, 28–29, 32, 34
 Dialog Box 29
Bargraph display 88
Basic Skills 1
Block move.. 108
Break
 Address 34
 Button 90
 Emulation 104
 Now! 105
Breakpoints 8–9, 34, 84, 96
 Clearing 105
 Hardware 146
 Menu 96, 105
 Setting 96–97

C

C call stack 102
C check box 94
Call stack .. 108
Call Stack window 91
Cascade windows 110
Changing the Controller 82
Child Windows
 Call Stack 91
Child windows 1, 91, 98
 Data 85, 91
 Program 85, 91
 Registers 91

- Source 85, 91
- Watch 91
- Clipboard, copy to 102
- Clock
 - Cycles 154
- Clock=0 button 90
- Close 111
- Code window 108
- Color
 - Config 85
 - Menu 85
 - Scheme field 86
 - Setup dialog box 85
 - Window 84
- Color .. 106
- Compilers
 - IAR / Archimedes 22
 - Keil / Franklin 22
- Conditions A and B 121
- Config 106
 - Color 85
 - Hardware 81
 - Menu 76, 78–80, 82, 84–87, 89, 105
 - Miscellaneous 82
 - PPA 86
 - Trace 86
- Config | Memory map.. 79
- Configurations
 - 128k 16
 - 32K 14
- Control byte address 30
- Copy
 - To clipboard 102
- Current program counter 96
- Customize button 30
- Cy field in tool bar 90
- Cycles
 - Clock vs Instruction 154

D

- DATA 121
- Data Window 9, 91, 93–95, 108–109
 - Menu 94, 108
 - READ cycles 79
 - WRITE cycles 79
- Delay field 125
- Delete 9
 - All 9, 105
 - Breakpoint 9, 96
 - Item 77
 - Projects 77
 - Watchpoint 110
- Dialog Box
 - Bankswitching 29
- Disable
 - All breakpoints 105
- Disassembled instructions 96
- Display as.. 109

E

- E0 140
- E1 140
- Edit
 - Registers 109
- Edit .. 108, 110
- EMUL51-PC/E128-BSW 22, 35
 - Bankswitch Modes 37
 - Control Lines 37
 - Jumper Description/Memory Map 36
 - Layout 36
 - POD Board Modification 35
- EMUL51-PC/E256-BSW 22, 37, 39
 - Bankswitch Modes 42
 - Control Lines 41
 - Jumper Description/Memory Map 40

- POD Board Modification 38
- Emulation
 - Memory Mapping 20
 - RAM 80
- Emulator
 - Address Setting 18
 - Hardware .. 106
 - Installing 6
 - Internal files 79
 - Port field 81
- Emulator Hardware ..
 - Menu 81
- Enable
 - Code space limits box 84
- Enter
 - Data dialog box 9
 - Instruction dialog box 9
- Evaluate 102
- Exit 101
- EZ-hooks 34

F

- F10 key 145
- File 8
 - Menu 75–76, 100
- Files Provided 21
- Fill.. 109
- Filter field 145, 155
- Filtering 123
- Find
 - Trig point 151
- Frame 120, 133
 - Numbers 148
- Function 107–108
- FWR
 - Advanced Trace 140–141
 - Standard Trace 122

G

- GO 8, 104
 - Button 90
 - FOREVER 104
 - To cursor 104
 - Until cursor 96
- Go to .. 104
- Go to beginning... 149
- Go to end of trace buffer 149

H

- Hardware
 - Config 81
- Header
 - J2 129
- Headers
 - Advanced Trace Board 131
 - Emulator Board 13
- Help
 - Button 90
 - Line 100
 - Menu 111
- Hex
 - Record files 78
- Hexadecimal address 96
- High address 84
- Hot Keys 100

I

- I/O
 - Device 79
 - Port Address 14
- IAR / Archimedes 22, 26
 - Compilers 22
 - ICC8051 Compiler Options 24

- XLINK Linker Options 24
- Icons, Arrange 110
- Info .. 111
- In-line assembler 9, 96
- Input button 82
- Inspect 102
 - Window 98–99
- Installing
 - Emulator 6, 11
 - Pod 7
 - Software 8
- Instruction
 - Cycles 154
 - Pipeline 145
- Intel Hex files 78

J

- J2 header 129
- Jumper configuration
 - Four memory 15
 - Single memory 15
- Jumpers 12
 - Address 19
 - Advanced Trace Board 131
 - Location on Trace Board 117
 - Setting 12
 - Trace board 116, 130

K

- Keil / Franklin 23
 - BL51 Banked Linker 26
 - Compilers 22

L

- Linked Object files 79

- Lists of Conditions 121
- Load
 - Path 78
 - User Modules 78
- Load code .. 8, 78
- Loader Preferences Dialog Box 101
- Local system menu 111
- Loop timing 154
- Low address 84

M

- Mapping
 - Both Code & External Data 20
 - Emulation Memory 20, 79
- MDI 1
- Memory
 - Code 20
 - External Data 20
 - Map
 - Installation 79
 - Trace 113, 127
- Memory Banks
 - Active 30
 - Multiple 28
 - Switching 22
- Memory map .. 105
- Menu
 - Config 80
- Menus 90–91
 - Breakpoints 105
 - Color 85
 - Config 76, 78–79, 82, 84–87, 89, 105
 - Data 94, 108
 - File 75–76, 100
 - Help 111
 - Program 106

- Registers 92, 109
- Run 8, 96
- Source 107
- View/Edit 98, 102
- Watch 109
- Windows 100, 110
- Microsoft Windows 1, 75
- MISC 121
- Miscellaneous 106
 - Config 82
 - Item 82
- Miss bin 87
- Module 107–108
- Move .. 76
- MS Windows 1, 75
- Multiple Document Interface Standard 1

N

- Next window 111
- Normal
 - Button 82

O

- Object Converter OC51 28
- Object files, linked 79
- Odd
 - Address floating point 95
 - Address integers 95
- On-line help 2
- Open a new child window 110
- Origin (at program counter) 106–107
- Original Address 108–109
- Output button 82

P

- P1 121
- P3 121
- Parameters in Hex 109
- Path
 - Dialog box 78
 - Internal files 79
 - Load 78
 - Setting 78
 - Source 105
- Paths .. 78, 105
- PC I/O Address space 114, 128
- PCB layout 28
- Performance Analysis 86–87, 106
 - Adding a Bin 89
 - Bargraph option 88
 - Miss bin 87
- Pipeline Decoding 145
- POD-31S 82
- PODs
 - Available for EMUL51-PC 2
- Pods
 - Installing 7
- Port
 - Values 34
- Power Requirements
 - Trace board 113, 127
- PPA 86–87
- PPA .. 106
- Preferences 101
- Program 8–9
 - Loading and Executing 8
 - Menu 106
 - Window 85, 91, 95–98, 106–107, 155
- Project name .. 105

R

- RAM value 9
- READ cycles, Data Window 79
- Record filtering 123
- Registers
 - Menu 92, 109
 - Window 91–92
- Relative cycle 154
- Relative time 155
- Remove .. 110
- Remove Symbols 101
- Repaint 110
- Reset
 - Button 90
 - Chip after load file 83
 - Chip and Break 8, 105
 - Chip at start up 83
 - Chip! 105
 - Emulator button 82
 - Emulator! 104
 - Override start address 84
- Response latency 154
- Run
 - Menu 8, 96
 - Program Manager menu item 75

S

- Save
 - Button 86
 - Trace as text .. 151
 - Trace image to file! 151
- Search 103
 - Next 103
 - Next Address 150
 - Previous 103
 - Previous Address 151

- Select window class 85
- Set
 - New PC value at cursor 107
 - Project Name Dialog box 77
 - Software breakpoint 8
- Setting the Paths .. 78
- Setup
 - Dialog box 96
- Setup .. 8–9, 105
- SETUP.EXE 75
- SFR button 30
- Show
 - All frames 153
 - Function 109
 - Load info 101
 - Timestamp 153
 - Trace image from file 152
- Single step 98
- Soft reset 104
- Software
 - Configuration 76
 - Configuring 76
 - Installation 8, 75
- Source 8–9
 - Menu 107
 - Paths 105
 - Window 85, 91, 97–98, 103, 107–108, 155
- Space
 - Address 30
- Special Conditions .. 105
- SpecialRegs window 91
- Stack
 - Window 99
- Start Address override 84
- Step
 - Button 90, 98
 - Into 103

- Over 103
- STR files 21, 79, 82
- SYM files 21, 79, 82
- Symbol 21
- Symbol Files 84
- Synchronize
 - Program window 155
- System Requirements 5, 11

T

- Target
 - Memory access 79
 - RAM/ROM 80
- Tile windows 110
- Time stamp 154
- Toggle 8–9, 105
 - Breakpoint 9, 107–108
 - Help line 100, 110
 - Trace speed bar! 148
- Trace
 - 16k 118
 - 4k 118
 - Beg 145
 - Config 86, 106
 - End 145
 - Frame 120, 133
 - Frame size 117
 - Menu 109, 147–148
 - Record 120, 133
 - Selective Recording 145
 - Setup 145–146
 - Setup Dialog Box 127
 - Speed Bar 148
 - Triggering 124
 - Window 91, 98, 147, 151, 155
- Trace .. 86, 106
- Trace Board

- Installing Advanced Trace 130
- Installing Standard Trace 116
- Tracing 32
- Triggering 124
 - And Filtering 120

U

- User
 - Defined symbols 102
 - Load modules 78

V

- View
 - Assembly code 108
 - Source window 107
- View/Edit Menu 98, 102

W

- Watch
 - Menu 109
 - Window 91, 99, 109–110
- Watch point 83
 - Add 102
 - Delete 110
- Windows 147
 - Call Stack 91
 - Code 108
 - Colors 84
 - Data 9, 93–95, 108–109
 - Menu 100, 110
 - Program 85, 91, 95–98, 106
 - Registers 91–92
 - Source 91, 97–98, 103, 107–108, 155
 - Watch 91, 99, 109–110

WRITE cycles

 Data Window 79

X

Xdata

 Memory 35, 37

 Write 34

XLINK 24

Z

Zoom 110