TechTools

PicTools Manual 3.0

PicTools Manual 3.0

Printed: September 2007 in Garland, Texas U.S.A.

Publisher

TechTools
P.O. Box 462101
Garland, TX 75046-2101
U.S.A.

Technical Sales

(972) 272-9392

Fax

(972) 494-5814

Email

Sales@tech-tools.com

Support@tech-tools.com

On The Web

www.tech-tools.com

Table of Contents

Foreword 0

Part I TechTools PicTools Introduction 2

Part II QuickWriter MCU Programmer 4

1 HEX File 5

2 Setting Options 5

Setting Auto Options 6

Setting General Options 7

Setting HEX Options 9

Setting Serial Number Options 10

Setting Advanced Options 12

Setting I.D. Locations 13

Setting Options for Production Personnel 14

3 Operation 15

Selecting the MCU 15

Setting the Configuration Word (Fuses) 15

Using Gang Adapters 17

Using Serial Numbers 17

Reading the MCU Contents 19

4 Advanced 20

In-Circuit Programming 20

ICP with Self Powered Targets 21

Cable Connector 23

Command-line Operation 23

Exit Codes 24

Legend - Programming Tasks 25

Part III TechTools Design Environment 27

1 Sample Project 27

Starting TDE 28

Opening The Project 28

Loading a Source File 29

Hardware Settings, Project Info 31

Device Settings, Building the Project 32

Running the Code, Stopping Execution 34

Single-Stepping 35

Animation, Watching Variable Values 36

Watching Variables, Viewing Special Registers 38

Watching the Program Run 40

Resetting the Emulator, Setting Breakpoints 41

Finishing the Project 43

2 TDE Details 44

Project and File Operations 45

Creating a Single-file Project 46

Adding a File 48

Creating a Multiple-file Project 50

Adding Files 51
Mode of Operation 55
Watch Dog Timer 57
Break on Stack Over/Underflow 57
Device Settings 59
Trace Buffer 60
Environment Preferences 62
Editor Preferences 63
General preferences 64
Build Preferences 65
Debug Preferences 65
Programming Preferences 66
Resetting the Emulator 66
Printing 67
Compiling and Executing Code 68
Building the Project 68
Executing Code at Full Speed 70
Halting Execution 71
Single-Step Execution 71
Step OUT Execution 72
Step Over Execution 73
Animation-Mode Execution 74
Animate Over Execution 74
Showing the Execution Point 75
Resetting the Emulated PIC 76
Setting the Program Counter 77
Using Breakpoints 78
Using the Various TDE Windows 80
Window Basics 80
Editor Window 82
Enter source code 85
Set and clear breakpoints 85
Cut, copy, and paste 85
Undo and redo 88
Search and replace 89
Place markers 91
Build Log 93
Code Window 95
Symbols Window 96
Special Registers Window 97
Variables Window 99
Labels Window 103
Files Window 105
File Registers Window 105
Watch List Window 108
Adding Watch items by Highlighting 108
Add Watch Dialog 109
Modifying Watch item values 112
Watch List Options 115
Trace Buffer Window 116
Data Breakpoints 118
DBP Enable and Break On options 120
DBP File Register options 120
DBP Match On options 121

DBP Include Constants ... 123
Breaking on data ... 124
3 Compiler Integration Notes ... 125
How we do it ... 126
CVASM16 ... 127
SPASM ... 127
MPASM ... 127
MPLINK ... 128
Bytecraft C ... 128
Bytecraft LINKER ... 128
MPLABC ... 129
Hi-Tech C ... 129
CCS ... 130
COD ONLY ... 131
HEX ONLY ... 131
Part IV ClearView Assembler 133
1 CVASM16 Reference ... 134
Software Installation ... 134
Running the Assembler ... 134
Generating Assembly Listings ... 135
Command-Line Options ... 135
Assembler Basics ... 135
Addressing Definitions ... 136
Data Types ... 136
Expressions ... 138
Symbols & Labels ... 138
Local Labels ... 139
Default_Symbol_Tables ... 139
Comments ... 140
Assembler Directives ... 140
Source Code Formatting ... 142
CVASM16 Error Messages ... 142
2 TechTools Instruction Set ... 144
Microchip to CVASM16 ... 145
ADD, ADDB ... 148
AND ... 150
CALL ... 151
CJA, CJAE ... 152
CJB, CJBE ... 153
CJE ... 154
CJNE ... 155
CLR ... 156
CLRB, CLC, CLZ ... 157
CSA, CSAE ... 157
CSB, CSBE ... 159
CSE ... 160
CSNE ... 161
DEC, DECSZ, DJNZ ... 162
IJNZ, INC, INCSZ ... 163
JB, JC ... 164
JMP ... 164
JNB, JNC ... 165

JNZ, JZ 166
LCALL, LJMP, LSET 166
MOV [to register] 169
MOV [to OPTION] 169
MOV [to I/O Control] 171
MOV [to W] 172
MOV [using Expressions] 172
MOVB 174
MOVSZ 175
NOP 175
NOT 176
OR 176
RET, RETFI, RETW 178
RL, RR 179
SB, SC, SZ 179
SETB, STC, STZ 180
SKIP 181
SLEEP 181
SNB, SNC, SNZ 182
SUB, SUBB 183
SWAP 184
TEST 184
XOR 185

3 Sample Projects 186

Simple.tpr 187
Table.tpr 190
Bargraph.tpr 194

Part V ClearView Mathias Hardware 199

1 ClearView Mathias 199
2 System Requirements 200
3 Packing List 200
4 Hardware Features 201
5 Hardware Installation 201
6 Power Supply Considerations 203
7 MCLR Pull-Up Resistor 204
8 Optional Modules 205

Index 207

TechTools PicTools Introduction

Part

I

1 TechTools PicTools Introduction

This manual covers information for using the following TechTools products:

QuickWriter (MCU Programmer 4), **ClearView Mathias** (In-Circuit Emulator 199),

CVASM (ClearView Assembler 133), **TDE** (TechTools Design Environment 27)

QuickWriter and Mathias are hardware devices that use our CVASM and TDE software. This manual will reference both hardware products in the software sections, so not all features will be available unless you own both products. For example, the debugging features of TDE will not be available if you do not own a ClearView Mathias In-Circuit Emulator.

TechTools

Thank You for choosing TechTools for your development needs.

www.tech-tools.com
(972) 272-9392
Email Support
Email Sales

Part II

2 QuickWriter MCU Programmer

QuickWriter

Hex Files

Opening a HEX File 5

Setting Options

Setting Options 5
Setting Auto Options 6
Setting General Options 7
Setting HEX Options 9
Setting Serial Number Options 10
Setting Advanced Options 12
Setting I.D. Locations 13
Setting Options for Production Personnel 14

Operation

Operation 15
Selecting the MCU 15
Setting the Configuration Word (Fuses) 15
Using Gang Adapters 17
Using Serial Numbers 17
Reading the MCU Contents 19

Advanced

Advanced 20
In-Circuit Programming 20
ICP with Self Powered Targets 21
Cable Connector 23

Command-line Operation 23
Exit Codes 24
Legend - Programming Tasks 25

2.1 HEX File

Overview

QuickWriter accepts INHX8M (INTEL HEX 8 bit merged) and INHX32 HEX files. For the PICmicro MCU microcontroller, these are generally in one of two formats:

Microchip: Includes Code, Data, ID and Configuration Word information. This format is normally saved with a ".hex" extension.

TechTools: Includes Code, Data, ID, Device type, Configuration Word and sometimes Calibration information. This format is normally saved with a ".obj" extension.

QuickWriter will accept the above formats and adjust all relative data according to the type of information found in the HEX file. Any information that is missing from the HEX file (such as the Configuration Word) will be set to defaults unless manually overridden elsewhere in the program. If the MCU type is not specified, then the currently selected MCU will be used.

QuickWriter saves HEX files with the ".hex" extension following Microchip's format. However, it also includes the device type as a comment that can be read by QuickWriter. This allows other products that may not recognize the device type to use the HEX file without experiencing any errors.

Details

Select "**FILE - Open File**". Choose the desired file, then click "Open". The full filename and path will be displayed in the Title Bar of the program, the file will be loaded into memory and all displayed data will be updated with the contents of the HEX file.

Shortcuts

Alt+F, O

2.2 Setting Options

Overview

Options are divided into five categories:

Auto Run Options 6 - Enables and disables programming operations and sections of the MCU for "Auto Run" cycles. These selections will enable or disable speed buttons and the associated item in the "Task Tree" on the left of the main window, but will not affect the "MCU SECTION" menu items, which will always be available while "Option Editing" is enabled in General Options.

General Options 7 - General options that are not normally needed during development but help when preparing settings for production personnel.

HEX Options 9 - Provides HEX file Overrides for the specific HEX file and selection of records to include when saving HEX files.

Serial Numbers 10 - Enable and set Serial Numbering details such as number of locations, area of the MCU and address to place the serial number.

Advanced Options 12 - Advanced settings mainly related to In-Circuit Programming.

All options are dynamic and take effect immediately when changed. It is NOT necessary to "close" the options window or to "save option settings" in order to "apply" the new settings. QuickWriter was designed with developers in mind and frequent option changes between programming tasks or "auto run" cycles is expected.

Details

Option changes and Editing take effect immediately regardless of whether the Window is closed or remains open.

Options are saved permanantly when the application is closed and when "File - Save Option Settings" is selected.

Options are loaded whenever the application starts and when a HEX file is loaded. Options will be restored from the HEX file and the corresponding ".QWC" (QuickWriter Control) file if found, otherwise all options will remain unchanged from the current settings.

NOTE: Some options are Refreshed from permanent settings when the MCU selection is changed. Be sure to save any changes to options before changing the MCU selection.

Shortcuts

Alt+O

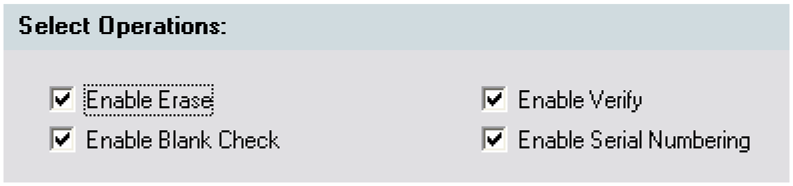
2.2.1 Setting Auto Options

Overview

Enables and disables programming operations and sections of the MCU for "Auto Run" cycles. These selections will enable or disable speed buttons and the associated item in the "Task Tree" on the left of the main window, but will not affect the "MCU SECTION" menu items, which will always be available while "Option Editing" is enabled in General Options.

Details

__Operations:__


Select Operations:

☑ Enable Erase
☑ Enable Verify
☑ Enable Blank Check
☑ Enable Serial Numbering

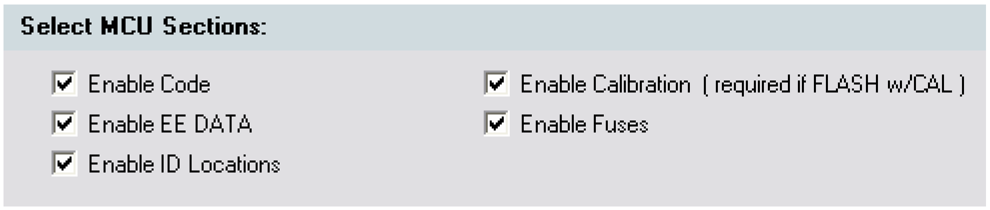
Enable Erase - If available for the MCU and selected, a bulk erase is performed on the FLASH device during Auto Run cycles.

Enable Blank Check - If selected, a blank check is performed on all enabled areas of the device during Auto Run cycles. We suggest leaving this option enabled unless reprogramming an MCU that already contains data.

Enable Verify - If selected, a verify will be performed during Auto Run cycles on all selected areas of the MCU (i.e. Code, EE Data, Calibration, ID) before the Configuration Word is programmed. Most programming operations automatically verify during programming, but because some MCU algorithms do not, we suggest always leaving this option enabled.

Enable Serial Numbering - If selected, the MCU will be serialized (just prior to programming the Configuration word) during Auto Run cycles according to the settings in Serial Number options 10 (also see: Using Serial Numbers 17).

MCU Sections:



Select MCU Sections:

- ✓ Enable Code
- ✓ Enable EE DATA
- ✓ Enable ID Locations
- ✓ Enable Calibration (required if FLASH w/CAL)
- ✓ Enable Fuses

Enable Code - If selected, the Code Space area will be blank checked, programmed and verified during Auto Run cycles according to Operation settings.

Enable EE Data - If selected, the EE Data area will be programmed and verified during Auto Run cycles.

Enable ID Locations - If selected, the ID locations will be blank checked, programmed and verified during Auto Run cycles. ID locations have no bearing on the operation of the MCU and can be ignored if desired.

Enable Calibration - If available for the MCU and selected, the Calibration area will be blank checked, programmed and verified during Auto Run cycles. Some FLASH MCUs with calibration require this option and QuickWriter will automatically select it for you. If the MCU is not a FLASH device, **we strongly advise against selecting this option** unless programming a JW (windowed) device. Attempting to program the calibration area of an OTP (one time programmable) device will result in corrupted calibration data.

Enable Fuses - If selected, the Configuration Word will be programmed and verified during Auto Run cycles.

Shortcuts

Alt+O, O

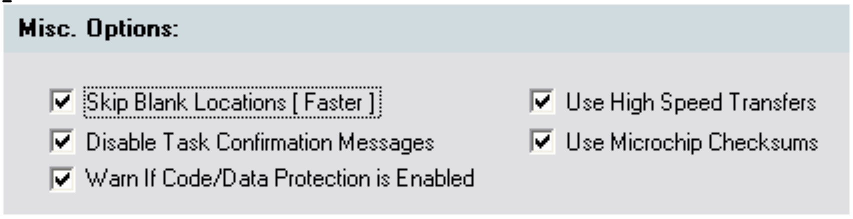
2.2.2 Setting General Options

Overview

General options that are not normally needed during development but help when preparing settings for production personnel.

Details

Misc. Options:


Misc. Options:

- [✓] Skip Blank Locations [Faster]
- [✓] Disable Task Confirmation Messages
- [✓] Warn If Code/Data Protection is Enabled
- [✓] Use High Speed Transfers
- [✓] Use Microchip Checksums

Skip Blank Locations - If selected, any "Blank" value will be skipped during programming, which can greatly reduce programming time on large devices that are only using small portions of the Code space. Keep in mind however, that after serializing an MCU in a "Blank" area of the code space, verifying will pass because of skipping over the blank area. Likewise, blank checking will fail because a "non-blank" value is detected. This normally is not an issue but could be confusing when manually selecting individual programming tasks out of sequence. NOTE: This setting has no effect on 18 series MCU's and some newer FLASH MCU's due to multi-word programming algorithms.

Disable Task Confirmation Messages - If selected, all "Are You Sure?" type messages will be disabled. We suggest enabling this option after you are thoroughly familiar with the programs operation.

Warn if CODE/Data Protection is Enabled - If selected, a warning dialog will appear if the Fuse Options are set to "Protect" any area of the MCU. (see: Setting the Configuration Word (Fuses) 15)

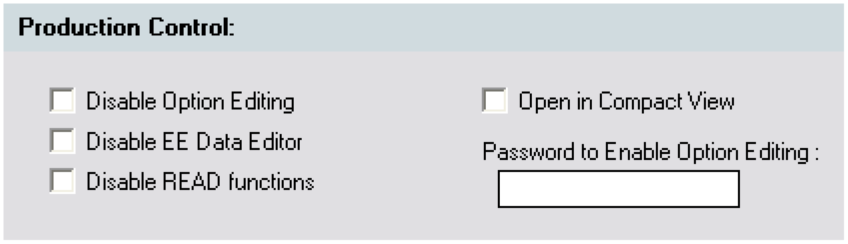
Use High Speed Transfers - Select this option to enable higher speed communication between the PC and Quickwriter. This will double the speed of Reading from the device and may reduce total programming time slightly on some MCU's. If any communication errors are detected while this option is enabled, QuickWriter will automatically switch back to the normal transfer speed until "Reset QuickWriter" is selected, a new port is selected, this option is re-selected or the program is restarted.

Use Microchip Checksums - If selected, QuickWriter's buffer checksum calculation will adhere to Microchip's method of calculation which does not include the EEDATA and only includes Code and User IDs depending on the values of the Fuses and the device selected. The resulting calculation is displayed as a 16 bit HEX value.

To match Microchip's "Code Protected" checksum value, enable this option and "Read" a code protected device.

If not selected, QuickWriter's checksum will include all enabled buffers (Code, EEDATA, User IDs and FUSES) and the resulting calculation is displayed as a 32 bit HEX value.

Production Control:


Production Control:

- [ ] Disable Option Editing
- [ ] Disable EE Data Editor
- [ ] Disable READ functions
- [ ] Open in Compact View

Password to Enable Option Editing :

Disable Option Editing - When this option is selected, all access to change options is hidden when the HEX file is opened. This prevents production personnel from accidentally changing settings and producing "unusable" devices.

To regain access, simply choose "Edit - Enable Option Editing". Entering the correct password will re-enable access to the options and the Editors.

If no password was specified when Option Editing was disabled, then leave the password field blank and select "OK" to re-enable access.

Disable EE Data Editor - When this option is selected, access to the EE Data editor will be disabled.

Disable Read Functions - If selected, all READ functions will be disabled. This will prevent production personnel from accidentally "Reading" the MCU and possibly changing loaded data.

Open in Compact View - When this option is selected, QuickWriter will open in "Compact View" only, regardless of the view selected previously. When this option is left unchecked, the view state is retreived from the operator's local settings (stored in the Operating System's Registry).

Password to Enable Option Editing - If a password exists in this field and the "Disable Option Editing" option is checked, then this password will be required to re-enable option editing when the current HEX file is re-opened. The password is limited to 32 Alpha-Numeric characters.

(also see: Setting Options for Production Personnel 14)

Shortcuts

Alt+O, G

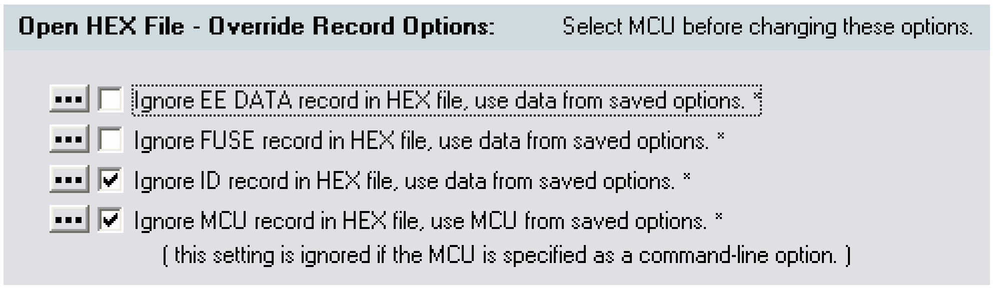
2.2.3 Setting HEX Options

Overview

Provides HEX file Overrides for the specific HEX file and selection of records to include when saving HEX files.

Details

Open HEX File - Override HEX Options:


Open HEX File - Override Record Options: Select MCU before changing these options.

- [ ] Ignore EE DATA record in HEX file, use data from saved options. *
- [ ] Ignore FUSE record in HEX file, use data from saved options. *
- [x] Ignore ID record in HEX file, use data from saved options. *
- [x] Ignore MCU record in HEX file, use MCU from saved options. *
  (this setting is ignored if the MCU is specified as a command-line option.)

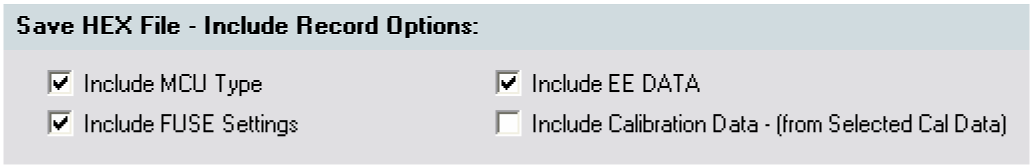
Ignore EE DATA - If selected, data currently displayed in the EE DATA editor will be saved and used whenever this HEX file is re-opened or the MCU selection changes, regardless of the data contained in the HEX file.

Ignore FUSE record - If selected, data currently displayed in the FUSES editor will be saved and used whenever this HEX file is re-opened or the MCU selection changes, regardless of the data contained in the HEX file.

Ignore ID record - If selected, data currently displayed in the ID Locations editor will be saved and used whenever this HEX file is re-opened or the MCU selection changes, regardless of the data contained in the HEX file.

Ignore MCU record - If selected, the currently displayed MCU will be saved and used whenever this HEX file is re-opened, regardless of the data contained in the HEX file.

Save HEX File - Include Record Options:


Save HEX File - Include Record Options:

- [x] Include MCU Type
- [x] Include FUSE Settings
- [x] Include EE DATA
- [ ] Include Calibration Data - (from Selected Cal Data)

Include MCU Type - If selected, a "device" record will be included in the generated HEX file during a save.

Include FUSE Settings - If selected, a "Configuration Word" record will be included in the generated HEX file during a save.

Include EE Data - If selected, the EE Data will be included in the generated HEX file during a save.

Include Calibration Data - If selected, the Calibration Data will be included in the generated HEX file during a save. When multiple MCU devices with configuration data are present (i.e using a 4-gang adapter), the configuration data will be saved from the current selection in the Calibration Editor.

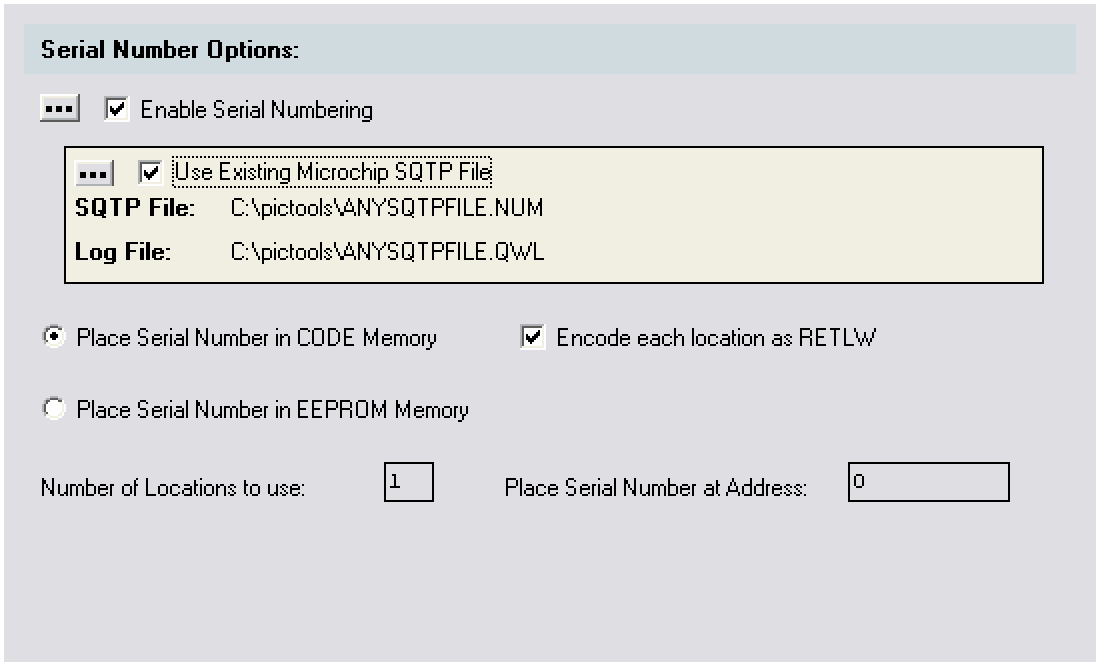
Shortcuts

Alt+O, H

2.2.4 Setting Serial Number Options

Overview

Enable and set Serial Numbering details such as number of locations, area of the MCU and address to place the serial number.


Serial Number Options:

[✓] Enable Serial Numbering

[✓] Use Existing Microchip SQTP File

SQTP File: C:\pictools\ANYSQTPFILE.NUM

Log File: C:\pictools\ANYSQTPFILE.QWL

(●) Place Serial Number in CODE Memory — [✓] Encode each location as RETLW

( ) Place Serial Number in EEPROM Memory

Number of Locations to use: 1 — Place Serial Number at Address: 0

Details

Serial Number Options:

Enable Serial Numbering - If selected, the MCU will be serialized during Auto Run cycles.

Use Existing Microchip SQTP file - When selected, use the button to the left of the option to choose an SQTP file (*.num). A log file will automatically be created with the same name but with a "QWL" extension.

Each serial number is marked as "used" when it is retreived from the SQTP file to prevent using the same number twice. As each number is programmed into a PICmicro MCU, the results are stored in the log file, including Pass or Fail, the QuickWriter used, the Socket used and the time in ms.

The SQTP file can reside anywhere on a local network as QuickWriter fully implements file sharing mechanisms for the SQTP file and the corresponding LOG file.

Microchip SQTP files can be generated using the MPLAB environment provided by Microchip Technology Inc. To generate this file from MPLAB, the appropriate tool (such as a Pro version programmer) must be selected (see the Help information provided with MPLAB). Or an equivelant HEX file can be generated manually with an identical address specified for each HEX line.

NOTE: When this option is selected, the "Locations" and "Address" options are disabled since this information is retreived from the SQTP file.

Place in Code Memory - If selected, the Serial number will be programmed into the Code area of the MCU. This also enables access to the Encode as RETLW option.

Encode as RETLW - This option is only available if the CODE memory option is selected. If selected, each Serial number location will be converted into the proper RETLW OPCODE (return literal in W) for the currently selected MCU.

Place in EEPROM Memory - If selected, the Serial number will be programmed into the EE DATA area of the MCU. This also Disables access to the Encode as RETLW option.

Number of Locations - Enter the maximum number of locations the serial number will need (from 1 to 8). Each byte of the serial number will use 1 location in the memory area specified.

Place at Address - Enter the address in the selected memory area to place the serial number. This can be from 0 to the highest available address, minus the number of locations specified. If an address higher than the maximum valid address is specified, QuickWriter will notify you and automatically reduce the address to the highest valid address for the selected memory area and number of locations.

(also see: Using Serial Numbers 17)

Shortcuts

Alt+O, N

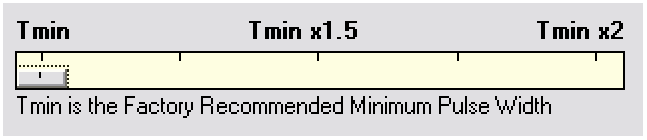
2.2.5 Setting Advanced Options

Overview

Adjust the Programming Pulse Width of FLASH MCUs and choose Target Power Options for ICP (In-Circuit Programming).

Details

FLASH Programming Pulse Width:



Tmin is the Factory Recommended Minimum Pulse Width

Modify this setting in "Less than Optimal" situations to ensure successful programming over voltage, temperature, device and circuit variations.

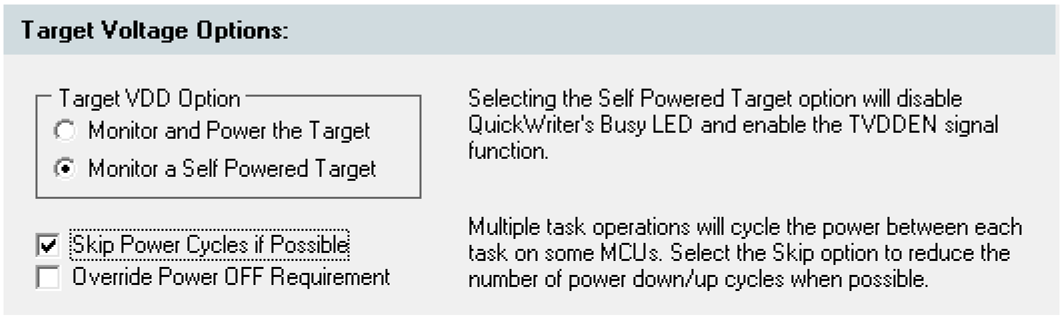
The default value of Tmin is preset to the Factory Recommended Value that already accounts for temperature and device variations. However if errors occur when programming In-circuit, this value may need to be adjusted to account for Voltage and Circuit variations.

Each step in the selector (shown above) will increase the duration of the programming pulse by 1/4 of the Factory Recommended Value. The 5 steps are as follows:

PPW = Tmin or **1 PPW**
PPW = (Tmin + (Tmin /4)) or **1.25 PPW**
PPW = (Tmin + ((Tmin /4) * 2)) or **1.50 PPW**
PPW = (Tmin + ((Tmin /4) * 3)) or **1.75 PPW**
PPW = (Tmin * 2) or **2 PPW**

If the calculated value exceeds factory specifications or a maximum of 255 ms, it will be adjusted apropriately.

Target Voltage Option:



Target Voltage Options:

Target VDD Option
- ○ Monitor and Power the Target
- ● Monitor a Self Powered Target

Selecting the Self Powered Target option will disable QuickWriter's Busy LED and enable the TVDDEN signal function.

- [✓] Skip Power Cycles if Possible
- [ ] Override Power OFF Requirement

Multiple task operations will cycle the power between each task on some MCUs. Select the Skip option to reduce the number of power down/up cycles when possible.

Modify this setting for In-Circuit Programming to select whether QuickWriter supplies the power for the target circuit or whether the Target circuit will supply its own power. (see: ICP with Self Powered Targets 21)

Skip Power Cycles:
Select this option to have QuickWriter reduce the number of Power OFF/ON cycles during "Auto" cycles. This option is not recommended, but may be needed for some in-circuit configurations.

Override Power Off Requirement:
Select this option if it is neccessary to have power present in the target circuit before starting any "Auto" cycle. When selected, this option also will not require the target power to be removed when the cycle has been completed. NOTE: This option is not recommended, but has been added due to popular request.

Shortcuts

Alt+O, A

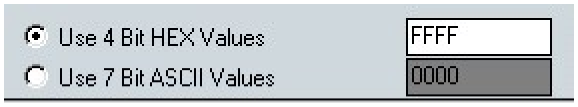
2.2.6 Setting I.D. Locations

Overview

PICmicro MCU's have four or eight ID locations reserved for the customer's use. Although each location may be 12, 14 or 16 bits in length (depending on the MCU), the manufacturer recommends using the 4 least significant bits of each location. Common industry practice has offered the ability to use the 7 least significant bits of each location therefore, we provide both options.

If your HEX file does not include ID information or you would like override the ID with a different value whenever the HEX file is opened, select the "Override" check box after changing the ID values.

Details



- ● Use 4 Bit HEX Values FFFF
- ○ Use 7 Bit ASCII Values 0000

Select "Edit - ID Locations". If the values displayed do not reflect those desired, simply select and edit the appropriate field (4 bit or 7 bit). Any changes take effect immediately and will be implemented with the next programming action (i.e. Autorun, program ID locations, etc.).

(see also: Setting HEX Options 9)

Shortcuts

Alt+E, I

2.2.7 Setting Options for Production Personnel

Overview

Sometimes it is desirable to block access to option settings for a HEX file in order to prevent undesired changes. Whether this is for archiving "released" code or for the production department, QuickWriter provides a simple method of protecting your settings from accidental changes. This is accomplished through a "control" file.

The control file contains all option settings for QuickWriter, is saved in the same directory as the HEX file and is created with the same name as the HEX file except the file extension is changed to ".QWC". One of the options in the control file instructs QuickWriter to DISABLE option editing.

The following will be disabled when a HEX file is opened, if the "Disable Option Editing for this file" option has been selected:

MCU selection.
Auto Options, HEX Options and General Options.

Code, FUSE, ID and Calibration editors.

Manual access to tasks in the "Programming Tasks" tree or MCU Section menu.

ALL programming functions except Auto Run, Erase, Blank Check and Verify.

A detailed procedure for setting options for production personnel is described below but the general procedure is to Open the HEX file, set all desired options, select "Disable Option Editing" in "General Options", then select "File - Save Option Settings". The control file has now been created and modified (resides in the same directory as the HEX file), and is now ready for the production department.

Details

To configure a HEX file for archiving or production use, follow the procedure described here.

1. Open the desired HEX file from the directory it will be used in if possible.
2. Verify the MCU, Configuration Word and ID location settings. If the MCU and Configuration Word are incorrect you will need to make the necessary changes, then **save the HEX file with ALL "Save Hex Options" enabled or select the proper overrides in HEX Options**. After saving the hex file, re-open it and re-verify the MCU, Configuration Word and etc. If everything is correct, proceed with the next step.
3. Go to the "Options - Auto Options" and select each option to be performed during programming. Each selected item will be performed when the production department chooses "Auto Run" (or F4 or Alt+A).
4. If you have enabled Serialization, verify the serial number length and address settings to prevent overlapping valid code or EE data. Also verify the "last serial number" and "Auto Increment" settings (see: Using Serial Numbers 17).

5. After all settings have been verified, select "Disable Option Editing.." in the "Misc. Options" section of the "General Options". You may also want to select "Disable Read Functions" in this same location.
6. Select "File - Save Option Settings" from the main menu, then close the program.
7. Copy the HEX file and the corresponding control file (hexfilename.QWC) to the desired location for production accessability. The production department can now open the HEX file without fear of accidental changes.

Shortcuts

Alt+E, i (disable option editing for testing)

Alt+E, E (enable option editing for further editing)

2.3 Operation

Operation

Selecting the MCU 15
Setting the Configuration Word (Fuses) 15
Using Gang Adapters 17
Using Serial Numbers 17
Reading the MCU Contents 19

2.3.1 Selecting the MCU

Overview

Some HEX files will not contain the MCU type or you may want to select a different MCU to program. When the MCU type changes, QuickWriter will adjust many aspects of the software to match the capabilities of the selected device. Because of this it is very important to select the proper MCU type (including prefixes such as 'A','B' and etc.).

Details

Select "Options - Select MCU Type". When the MCU dialog opens use the cursor keys or the mouse to select the desired MCU, then click the 'OK' button (or press the 'Enter' key) or "DBL-Click" on the desired MCU.

(see also: Setting HEX Options 9)

Shortcuts

Alt+O, S

2.3.2 Setting the Configuration Word (Fuses)

Overview

The configuration Word is set according to data stored in the HEX file when it is opened. However, you are free to change this information in the program before programming the MCU, provided "Option Editing" is enabled for this HEX file (see: Setting General Options 7 ; see also: Setting Options for Production Personnel 14).

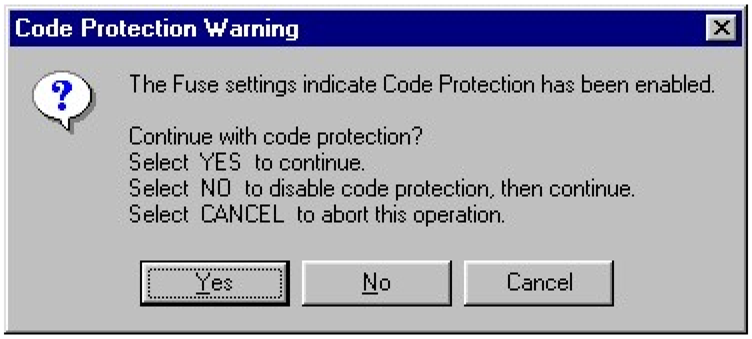
Specific bits in each Configuration Word control available MCU options when programmed.

QuickWriter identifies each set of bits by labels provided in the manufacturers datasheet. If you are not familiar with a particular reference, such as WDT (Watch Dog Timer), you will need to see the datasheet which will give a full explanation of the configuration option and its meaning. Do not change the configuration word unless you are fully aware of each options function. Selecting the wrong configuration can make the MCU operation undesirable or unreliable, which renders the device unusable.

Details

Select "Edit - Configuration Word". To change FUSES in the configuration word, use the selection arrow beside each Fuse option to select from its available settings (or use the TAB and CURSOR keys).

WARN if Code Protection Enabled - If selected and one of the Configuration Fuses indicate that the Code or Data Protection fuse(s) have been enabled, you will be presented with three options before programming the MCU.


Code Protection Warning

The Fuse settings indicate Code Protection has been enabled.

Continue with code protection?
Select YES to continue.
Select NO to disable code protection, then continue.
Select CANCEL to abort this operation.

This warning will help remind you that the protection fuses are about to be enabled.

Once the Code or Data protection fuses have been enabled on a device, the contents of the protected section is no longer accessible for READING or VERIFYING.

NOTE: Code Protection will prevent erasing on most JW (Windowed, eraseble) **devices.**

If setting the Protection fuse was unintentional, then select NO to have QuickWriter automatically disable the Protection and continue with the operation in progress. Otherwise choose YES to continue "AS IS" or CANCEL to completely abort the operation.

NOTE: Each MCU has different Configuration options. If the MCU type is changed, you will need to review changes in the Configuration Word options before programming the MCU.

(see also: Setting HEX Options 9)

Special Consideration

Some PICmicro MCUs have Fuse options to "turn off" the external MCLR signal and others may have options to set the Oscillator to "internal". On older MCUs with these options and a "Power First" requirement of VDD (such as the 12C508/12C509), QuickWriter may have problems re-programming or reading the contents once the fuses have been programmed. This normally does not become an issue since the devices are OTP (one time programmable).

Newer MCUs with these fuse options allow a "Power first" of VPP (programming Voltage), which ensures the device can re-enter programming mode and be read after the fuses have been programmed.

During an "Auto Run" programming cycle, QuickWriter always verifies all sections of the MCU before programming the Fuses. Programming the fuses last allows QuickWriter to perform a full verification of all areas of the MCU regardless of the fuse settings.

Shortcuts

Alt+E, W

2.3.3 Using Gang Adapters

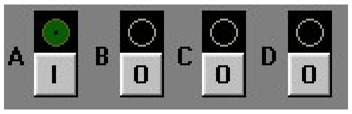
Overview

When using Gang adapters, each device is programmed and verified independently, but at the same time. Because of this, QuickWriter is able to save time while also tracking individual errors that may occur for each device being programmed.

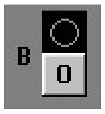
Gang adapters are available for all serially programmed devices that are supported by QuickWriter. Multiple devices can be programmed at once without costing additional time except for the minor delay while Serializing.

Details

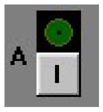
After connecting a gang adapter, you will want to enable multiple device support by selecting multiple sockets in the QuickWriter software.



The Socket LED Indicators (shown above), will change states when "clicked" as shown below.



Socket B disabled.



Socket A enabled

This can be done manually by toggling the button beneath each Socket LED Indicator or by using the Main Menu: "Options - Programming Sockets - Gang" , which will select all 4 sockets.

Shortcuts

Alt+O, P, G

2.3.4 Using Serial Numbers

Overview

QuickWriter programs serial numbers in Little Endian format. Any unused locations specified for serial numbering will be set to '00'. Therefore, the HEX serial number "FFFE" programmed into 8 locations of EE Data will be stored as "FE FF 00 00 00 00 00 00". If auto increment is selected, the next four incremented serial numbers will be stored as:

```
"FF FF 00 00 00 00 00 00"        (FFFFh or 65535)
"00 00 01 00 00 00 00 00"        (10000h or 65536)
"01 00 01 00 00 00 00 00"        (10001h or 65537)
"02 00 01 00 00 00 00 00"        (10002h or 65538)
```

Stored Format in EE DATA:
The serial number 62449988134764293 (DDDDEEEEFFFF05h) will be stored as:

```
"05 FF FF EE EE DD DD 00"
```

Stored Format in CODE memory:
The serial number 62449988134764293 (DDDDEEEEFFFF05h) will be stored as:

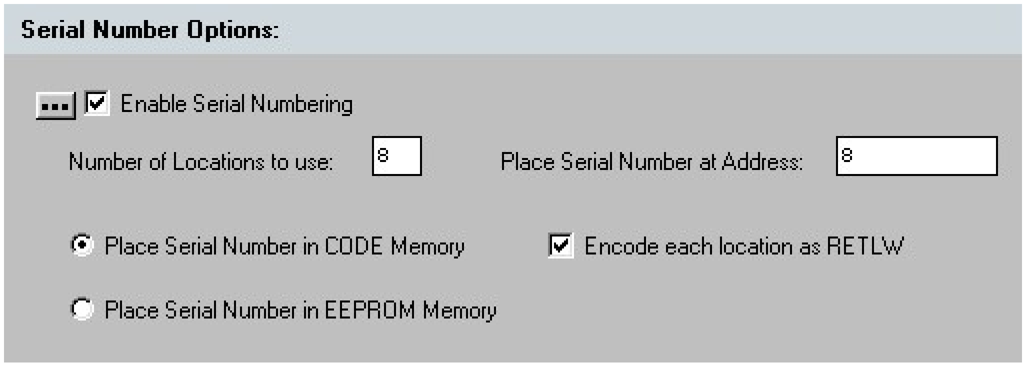
```
"0005 00FF 00FF 00EE 00EE 00DD 00DD 0000"
```

Stored Format in Code memory as RETLW in a PIC16F873:
The serial number 62449988134764293 (DDDDEEEEFFFF05h) will be stored as:

```
"3405 34FF 34FF 34EE 34EE 34DD 34DD 3400"
```

Details

To enable Serialization, select the "Serial Numbers" section of the "Options" window, then select the "Enable Serial Numbering" option (see: Setting Serial Number Options 10):


Serial Number Options:

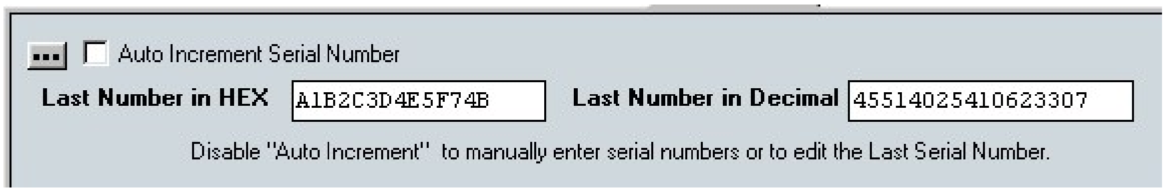
[✓] Enable Serial Numbering

Number of Locations to use: 8 Place Serial Number at Address: 8

(●) Place Serial Number in CODE Memory [✓] Encode each location as RETLW

( ) Place Serial Number in EEPROM Memory

The last step is to set the beginning serial number and whether to have QuickWriter "Auto increment" before programming each device.


[ ] Auto Increment Serial Number

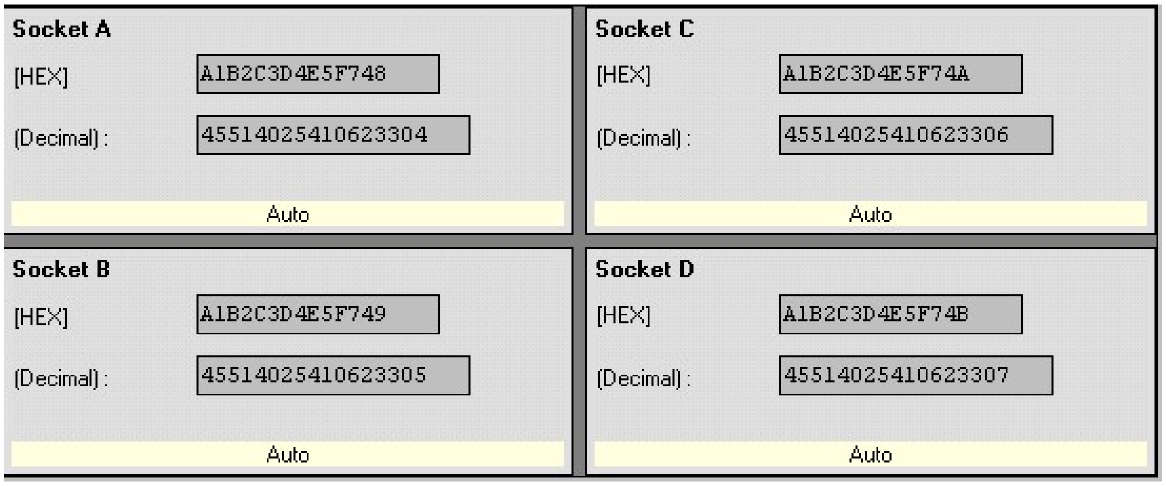
Last Number in HEX A1B2C3D4E5F74B Last Number in Decimal 45514025410623307

Disable "Auto Increment" to manually enter serial numbers or to edit the Last Serial Number.

Auto Increment - When selected, this option reads and stores the serial number indicated in the "**LAST HEX**" field as the last serial number used. The incremented value of this number will be programmed in the next device. The number used for each device will be displayed in HEX and Decimal beside the socket identifier and the results displayed on the right. The results field will indicate one of the following:

Manual - Serial numbering preferences have changed to Manual mode.
Auto - Serial numbering preferences have changed to Auto Mode.
Skipped - An earlier error has dictated skipping serialization for this socket. The serial number is not incremented for skipped sockets.

Passed - Serialization was successful for this socket.
Failed - An error occurred while serializing this socket. When a device fails to properly serialize, its serial number is not used for the next device.



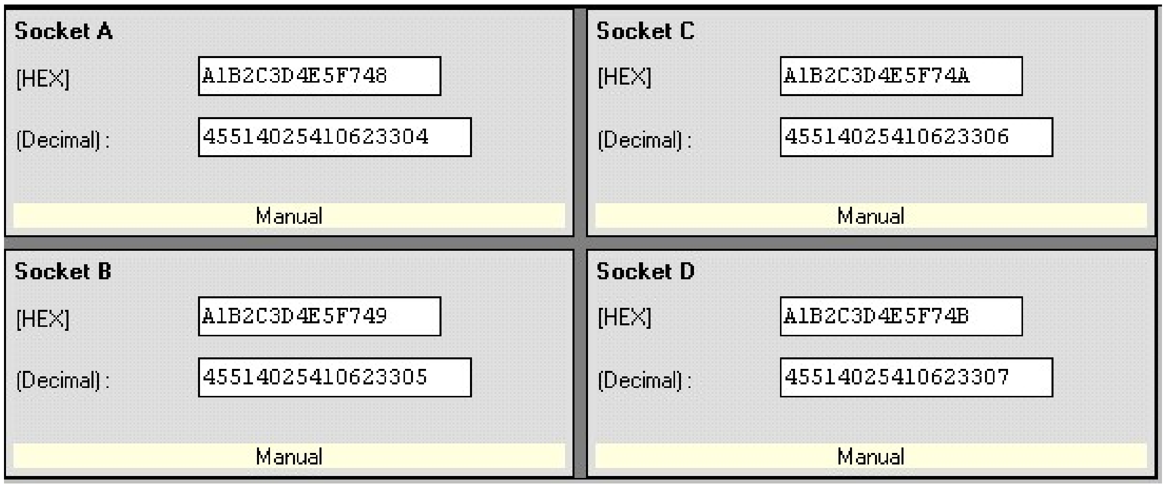
Socket A
[HEX] A1B2C3D4E5F748
(Decimal) : 45514025410623304
Auto

Socket C
[HEX] A1B2C3D4E5F74A
(Decimal) : 45514025410623306
Auto

Socket B
[HEX] A1B2C3D4E5F749
(Decimal) : 45514025410623305
Auto

Socket D
[HEX] A1B2C3D4E5F74B
(Decimal) : 45514025410623307
Auto

Manual Serial Numbers - Serial numbers can be entered manually for each device by unselecting the "Auto" check box and editing the HEX or Decimal field beside each socket identifier. The "Last HEX" and "Last Decimal" fields will reflect the last value used to serialize a device.



Socket A
[HEX] A1B2C3D4E5F748
(Decimal) : 45514025410623304
Manual

Socket C
[HEX] A1B2C3D4E5F74A
(Decimal) : 45514025410623306
Manual

Socket B
[HEX] A1B2C3D4E5F749
(Decimal) : 45514025410623305
Manual

Socket D
[HEX] A1B2C3D4E5F74B
(Decimal) : 45514025410623307
Manual

Shortcuts

Alt+E, S - Opens the Serial Number Editor (shown above).
Alt+O, N - Opens the Serial Number Options.
Alt+R, S - performs a "Serialize Only" programming task.

2.3.5 Reading the MCU Contents

Overview

Sometimes it may be desirable to "reproduce" an MCU that has already been programmed or to visually verify the contents. To do this, you will need to read the contents of the MCU instead of opening a HEX file.

Details

To read the contents of an MCU, choose "Run - Auto Read Only" from the main menu (or Ctl+R). If multiple sockets have been selected, a pop-up menu will appear for you to select the socket you wish to read from. After selecting the socket, the entire contents of the MCU (including I.D. locations and Configuration Word) will be loaded into the QuickWriter software.

Unless you intend to make exact duplicates of the MCU just read, remember to re-open the HEX file before programming another device.

(see also: Setting General Options 7; and Setting Options for Production Personnel 14)

Shortcuts

Ctl+R
Alt+R, R

2.4 Advanced

Advanced

In-Circuit Programming 20
ICP with Self Powered Targets 21
Cable Connector 23
Command-line Operation 23
Exit Codes 24
Legend - Programming Tasks 25

2.4.1 In-Circuit Programming

Overview

QuickWriter can be used with single socket and gang programming adapters, but one of its key features is the ability to program devices already incorporated in a working circuit. To program devices in-circuit, a few requirements must be met by the target.

For a detailed discussion on the key issues related to in-circuit programming, see the In-Circuit Programming Guide installed with the QuickWriter software.

NOTE: ICP does not support parallel programmed devices (i.e. PIC16C55, PIC16C57, etc.).

Details

When using the ICP cable (provided with QuickWriter), always select Socket A. If additional sockets are selected or if socket A is not selected, programming errors will occur.

If any error messages appear during ICP, pay special attention to any messages relating to voltage errors. QuickWriter performs extensive voltage testing to help prevent damage to itself and your target. All suggestions presented in the error dialog should be checked thoroughly before another programming attempt is made.

Target Design Recommendations

Use a 10K Ohm or larger pull up on MCLR Use 10uF or less capacitance on MCLR (most targets use none).

Avoid or isolate any other circuitry that touches the MCLR line.

Ideally, use RB6 (clock), RB7(data) and RB3 (LVP/PGM pin, RB4 or RB5 on some PICmicros) as OUTPUTs in your circuit design OR connect them to outside-world connections or NO switches so that nothing is driving them during programming. This avoids any need for isolation. If that is not desirable, design to isolate or tri-state anything that drives these lines.

Always connect the ground pin.

Decide who powers the target and address the trade-offs involved. (see: ICP with Self Powered Targets 21)

When programming, enable the PICmicro MCU's power-up timer Fuse Option if possible. Particularly if setting the device for internal MCLR or internal OSC.

In-Circuit Programming Cable Pinout

```
PIN Number | Signal Name           | KEY
---------------------------------------------
1.           VPP (MCLR)             RED
2.           VDD (+5)               White
3.           GND ( - )              White
4.           RB7 (data)             White
5.           RB6 (clock)            White
6.           NC                     Pin Block
7.           LVP (RB3/RB4/RB5) **   White
```

** Only necessary for some FLASH devices.
Holds LVP/PGM low to disable Low Voltage Programming.
(connect to RB3 on 16F87X devices, RB4 on PIC16F62X and RB5 on 18 series MCUs if needed).

Shortcuts

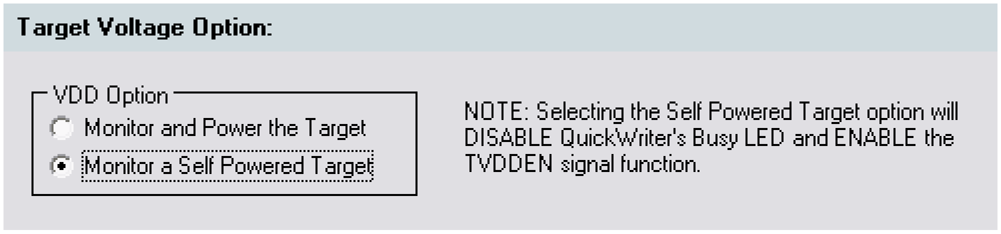
None

2.4.2 ICP with Self Powered Targets

Overview

QuickWriter can supply up to 100 ma of current on the VDD signal (5 volts) when programming MCUs in-circuit. This includes the current required by the MCU during programming (20 to 50 ma).

When programming MCUs in-circuit, it is sometimes desirable or even necessary for the circuit to supply its own power. If the target circuit has larger current requirements or if it is not feasible to power the circuit from the 5 volt power trace (i.e. when this will apply 5 volts to the output of a voltage regulator), then you should choose QuickWriter's "Monitor a Self Powered Target" option.



Target Voltage Option:

VDD Option
- ○ Monitor and Power the Target
- ● Monitor a Self Powered Target

NOTE: Selecting the Self Powered Target option will DISABLE QuickWriter's Busy LED and ENABLE the TVDDEN signal function.

When the circuit supplies its own power, we have to be concerned with voltage sequence and timing requirements of the MCU.

The programming algorithms for some devices require that the device's VDD be enabled/disabled several times during a
programming cycle. In self-powered target boards, this requires that we ask the operator to do the power cycling for us. This can become tedious for the operator. Even with devices that only require a single power up and down, it would still be nice to free the operator from this burden. To enable full automation of in-circuit programming, without violating programming specs, QuickWriter provides a control signal called TVDDEN.

Details

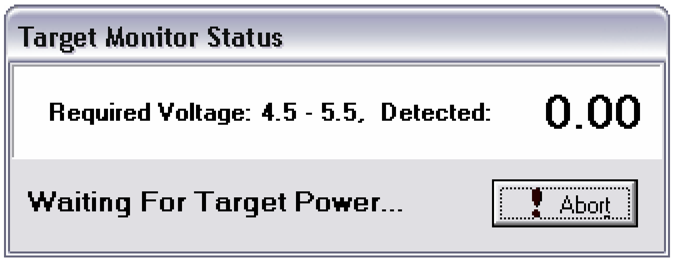
When using the "Self Powered Target" option, QuickWriter provides two ways of determining when to activate your target circuits power supply.

1. Operator Notification for manual activation/deactivation.

2. Hardware Control Signal (TVDDEN) for electronic activation/deactivation (automation).

Operator Notification

When QuickWriter is ready for the target's power supply to be turned on or off, it will display a notification dialog that includes the current voltage reading. Once the required voltage condition is met, this dialog will automatically close and QuickWriter will continue operation.



Target Monitor Status

Required Voltage: 4.5 - 5.5, Detected: 0.00

Waiting For Target Power...

If for some reason the voltage condition can not be met, the operator can use the ABORT button to close the dialog. Using the abort button will cause QuickWriter to discontinue the operation and display a relevant error message.

Hardware Control Signal (TVDDEN)

TVDDEN can be used to control your target's power supply (through an appropriate driver circuit). QuickWriter's TVDDEN signal will help automate control of the target's power and make it possible to meet the required timing for power sequence specifications.

TVDDEN is a new signal, available on QuickWriter's 26 pin interface connector. It is not available in the standard ICSP cable included with the unit. You may need to customize your in-circuit cable to make use of the new signal (and add your power control circuitry). (see: Cable Connector 23)

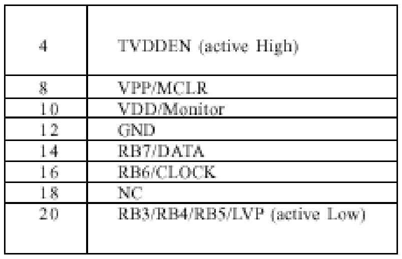
TVDDEN is an active high, logic level (+-20ma) signal. Typically, one would use it to drive a current limited LED within a solid-state relay to control your target's power supply.

2.4.3 Cable Connector

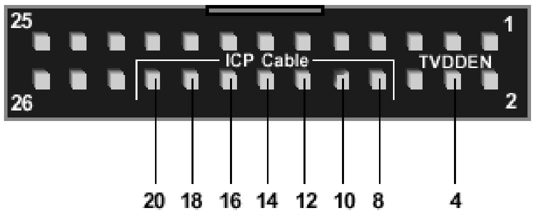
Overview

QuickWriter's programming cable connector is compatible with existing PICwriter and Parallax PIC16CXX-PGM programmer adapters while also providing Gang programming with TechTools Gang programming adapters and In-Circuit Programming with the included ICP Cable.

A newer signal, TVDDEN is also provided on this connector for automated control of the Target Circuit's power supply (see: ICP with Self Powered Targets 21).



4	TVDDEN (active High)
8	VPP/MCLR
10	VDD/Monitor
12	GND
14	RB7/DATA
16	RB6/CLOCK
18	NC
20	RB3/RB4/RB5/LVP (active Low)



2.4.4 Command-line Operation

Overview

Sometimes it is convenient to create a shortcut that will launch the QuickWriter software with a specific HEX file and device type, then have it program and close automatically. This can be accomplished from a BAT file or a Window's Shortcut using QuickWriter's command-line options.

Details

```
Usage - QW.exe  Filename [/Ddevicename]  [/A] [/X]
```

Filename = [optional] - Any valid filename including drive and path. If omitted, the last file used is opened. A Control File can also be specified (hexfilename.QWC), in which case QuickWriter will open the associated filename stored in the Control File.

/D = Device [optional - only valid if filename specified] - requires a valid, supported MCU number such as '18F2620'.

/A = AutoRun [optional - only valid if filename specified] - instructs the software to "Auto Run" and close if no errors occurred.

/X = Close with Exit Code [optional - only valid if filename and /A specified] - instructs the software to inhibit all non-critical notification dialogs during "Auto Run" and close when finished. The program will exit with an appropriate Error Code if an error occurred.

```
Example 1 - QW myhexfile.obj  /D12CE673 /A
```

The above example will launch QuickWriter, load the hex file "myhexfile.obj" from the same directory as the QuickWriter software, retrieve all option settings from "myhexfile.qwc" (if it exists), set the MCU to a `12CE673` then "**Auto Run**" and close. If any errors occur, the program will remain open for you to view the results. After viewing the error, you will need to manually close the program (Alt+X, or "File - Exit").

```
Example 2 - QW C:\myprojects\Blinkleds\myhexfile.obj
```

The above example will launch QuickWriter, load the hex file "myhexfile.obj" from the "C:\myprojects\Blinkleds\" directory and retrieve all option settings from "myhexfile.qwc" (if it exists). The MCU information will be set from the Control file if that option is specified in the control file. If the control file did not specify an MCU override, then the MCU is extracted from the hex file if a device record is present, else the last selected MCU will be used.

Shortcuts

None

2.4.5 Exit Codes

Overview

When the QuickWriter program terminates, an exit code is returned. This is normally ignored by the system but can be a great benefit when programatically launching the QuickWriter program from custom software (such as using the Window's API "ShellExecute" function). The exit codes can be used by the lauching program to make intelligent decisions.

Details

QuickWriter's Exit Codes are returned as a 2 byte bitmask, with each bit position representing a particular error code. A return value of 0 (zero) indicates no errors occurred.

Low byte of error codes

```
EF_port      = $1;   { 'Error communicating with Port'                  }
EF_timeout   = $2;   { 'Communication Timeout, Hardware not responding'  }
EF_RX        = $4;   { 'Communication Error Detected, BAD data received' }
EF_failed    = $8;   { 'Current Programming Task Failed'                 }
EF_badfw     = $10;  { 'Firmware update Required'                        }
EF_HIGHBAUD  = $20;  { 'Higher Transfer Speed Failed'                    }
EF_abort     = $40;  { 'User Aborted Task in progress.'                  }
EF_unknown   = $80;  { 'Unknown Error has Occurred.'                     }
```

High byte of error codes

```
EF_VDDLV     = $100;  { 'VDD Voltage TOO LOW (+5)' }
EF_VDDHV     = $200;  { 'VDD Voltage TOO HIGH'     }
EF_VDDTIME   = $400;  { 'VDD not rising'           }
EF_VPPLV     = $800;  { 'MCLR Voltage TOO LOW'     }
EF_VPPHV     = $1000; { 'MCLR Voltage TOO HIGH'    }
EF_VPPTIME   = $2000; { 'MCLR not rising'          }
EF_VDD5      = $4000; { 'Voltage detected on VDD'  }
```

```
        EF_VPPNZ     = $8000; { 'Can not pull MCLR low'    }
```

2.4.6 Legend - Programming Tasks

Parent Node

Select a parent node to perform the tasks listed beneath it. Some parent tasks will also perform a related sub task from another parent node (i.e. Program will also perform a Program/Verify of the Fuses node).

◨ - Child nodes are collapsed.
◢ - Child nodes are expanded. .
🟥 - An error has occurred in at least one child task.
◪ - This node has been disabled in Option Settings and will be skipped.
☒ - This node is not available for the selected MCU and will be skipped.

Child Nodes

A Child node can be selected to perform a single task.

● - Normal, available child task.
➡ - Node is selected and preparing to perform its task or the task was canceled.
🟥 - An error occurred for at least one socket while performing this task.
🟩 - This task was completed successfully.
⬜ - This task has been disabled in Option Settings and will be skipped.
⧄ - This task is not available for the selected MCU and will be skipped.

TechTools Design Environment

Part III

3 TechTools Design Environment

This portion describes the various feature s of TechTool’s TDE software. The first section is a brief step-by-step over-view, while the next section includes detailed descriptions of TDE's features.

The following TDE topics are covered :

Sample Project 27
TDE Details 44
Project and File Operations 45
Compiling and Executing Code 68
Using TDE's Windows 80
Data Breakpoints 118

3.1 Sample Project

If you already have emulator experience, then you may wish to skip to the next section, (TDE Details 44) which explains features in more detail. Otherwise, the following project should help you to become familiar with the basic functions of the emulator and its software.

The following pages describe a simple project that uses the most common features of ClearView Mathias and its TDE software. The project shows how to accomplish the following things :

Starting TDE 28
Open a project 28
Load a source file 29
Select ClearView type, serial port, and serial baud rate 31
Select PIC type, clock frequency, watchdog status, etc 32.
Compile and download code 34
Execute code in single-step mode 35
Execute code in animate mode 36
Watch variables during execution 36
Watch special registers during execution 38
Reset the emulator 41
Set breakpoints 41

The tutorial is based upon the PIC16C554.

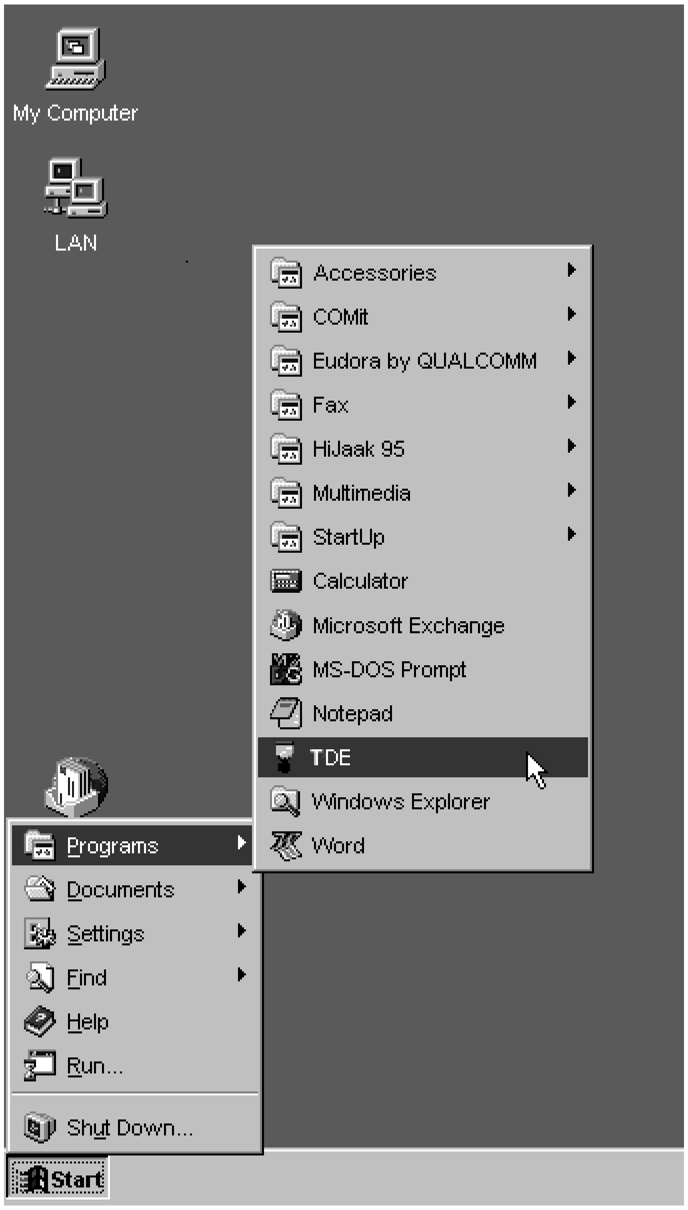
Because this project is meant as a quick introduction, some details have been left out. For instance, there is no mention of changing register values. Much more detail is included in the text that follows this project.

It’s assumed that you have successfully installed the Mathias hardware and software, as described in the previous sections of this chapter. If not, please review the installation sections before trying this project.

3.1.1 Starting TDE

Step 1:

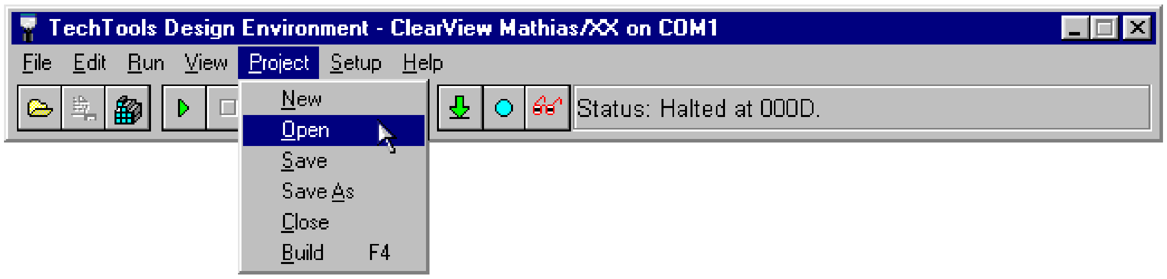
Start the TDE software. You should be able to select TDE from the Windows taskbar.



3.1.2 Opening The Project

Step 2:

Select Open from TDE's Project menu, or click the equivalent button as shown above.


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
New
Open
Save
Save As
Close
Build F4

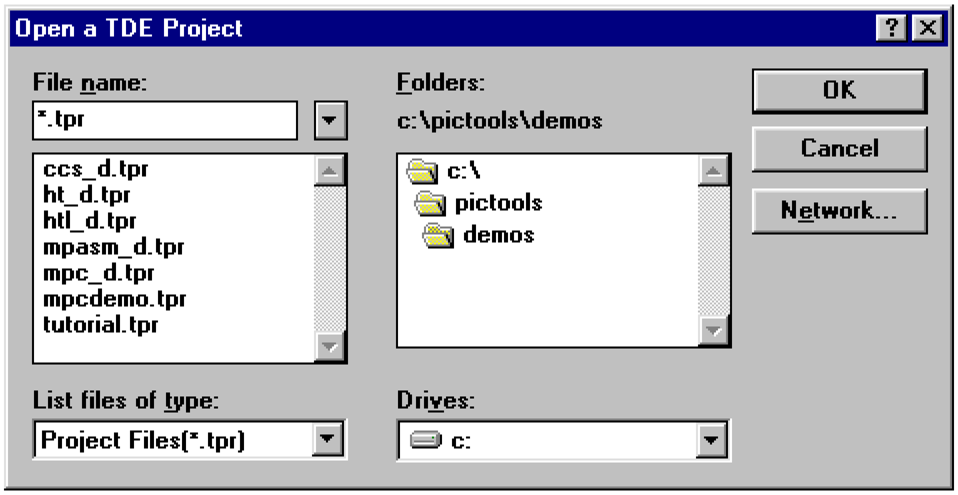
Status: Halted at 000D.

A "project" is a control file tha t stores all the information needed to integrate your work with the emulator. For example, the project file will store a reference to your source files (assembly language or C source files, for instance), and other useful setup and environment informat ion, such as Compiler options, emulator type, oscillator speed, editor settings, window positions, etc. When you close a project or exit from TDE, all of these settings are saved in a separate file with the ".tpr" extension.

3.1.3 Loading a Source File

Step 3:

Select the file "tutorial.tpr" from the newly-created TDE directory on your hard drive, then click OK.



File name: *.tpr

ccs_d.tpr
ht_d.tpr
htl_d.tpr
mpasm_d.tpr
mpc_d.tpr
mpcdemo.tpr
tutorial.tpr

Folders: c:\pictools\demos

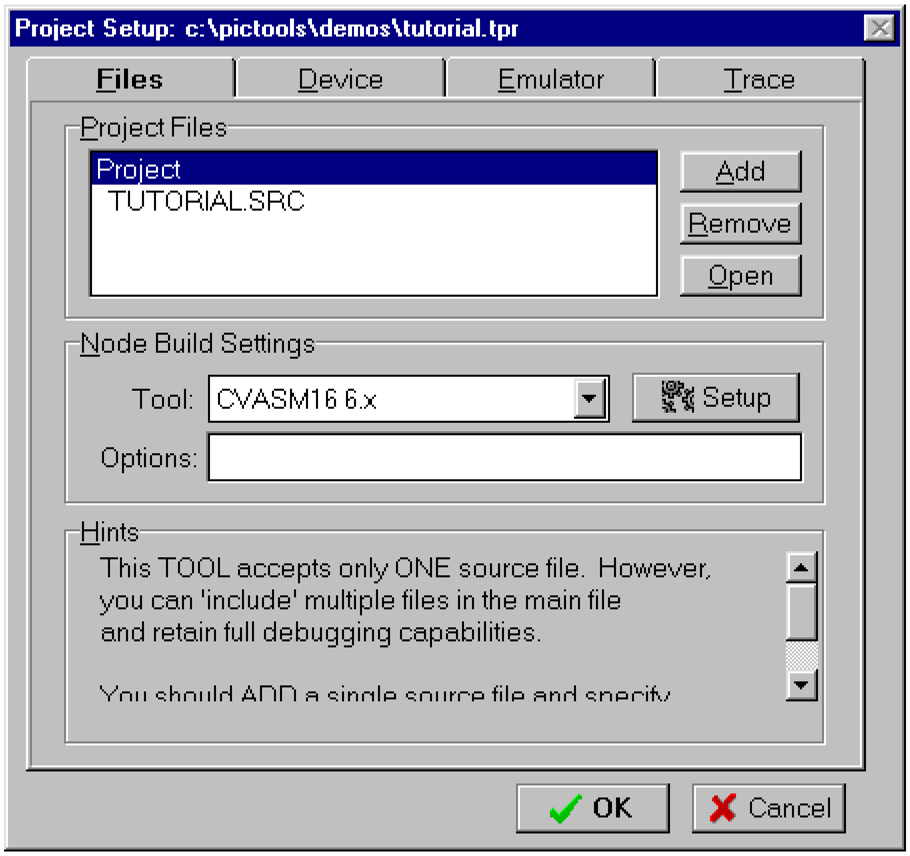
c:\
pictools
demos

List files of type: Project Files(*.tpr)

Drives: c:

Since this is an existing proj ect, the source file "tutorial.src" has already been associated and the "Project Tool" option has been set to "CVASM16". (For a detailed example of creating a new project and selecting project options, see the section "Project and File Operations" late r in this chapter.)

Now highlight the source file for this project and click the Open button.


Project Setup: c:\pictools\demos\tutorial.tpr

Project Files

Project
TUTORIAL.SRC

Node Build Settings

Tool: CVASM16 6.x

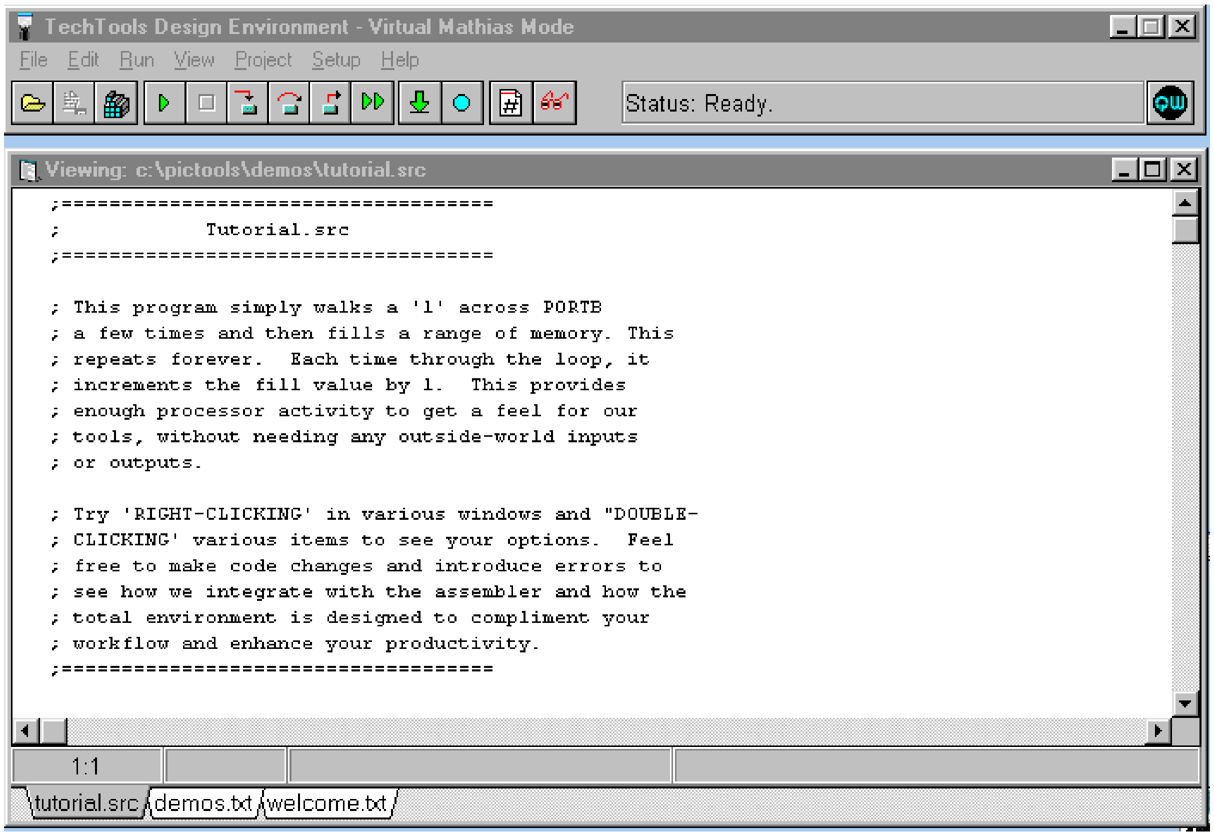
Options:

Hints

This TOOL accepts only ONE source file. However, you can 'include' multiple files in the main file and retain full debugging capabilities.

You should ADD a single source file and specify

After Step 3, you should see the fo llowing screen. The tutorial program appears in the text editor window.

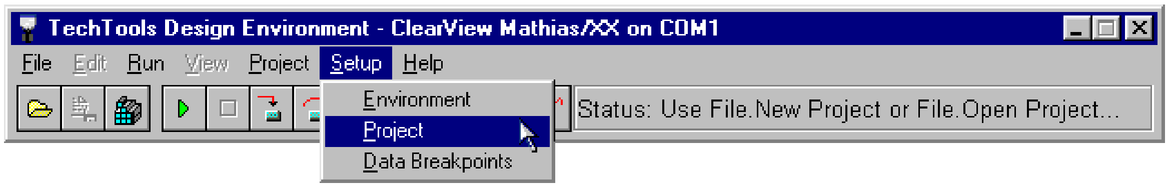

TechTools Design Environment - Virtual Mathias Mode
File Edit Run View Project Setup Help
Status: Ready.
Viewing: c:\pictools\demos\tutorial.src
1:1
tutorial.src demos.txt welcome.txt

```
;==================================
;            Tutorial.src
;==================================

; This program simply walks a '1' across PORTB
; a few times and then fills a range of memory. This
; repeats forever.  Each time through the loop, it
; increments the fill value by 1.  This provides
; enough processor activity to get a feel for our
; tools, without needing any outside-world inputs
; or outputs.

; Try 'RIGHT-CLICKING' in various windows and "DOUBLE-
; CLICKING' various items to see your options.  Feel
; free to make code changes and introduce errors to
; see how we integrate with the assembler and how the
; total environment is designed to compliment your
; workflow and enhance your productivity.
;==================================
```

3.1.4 Hardware Settings, Project Info

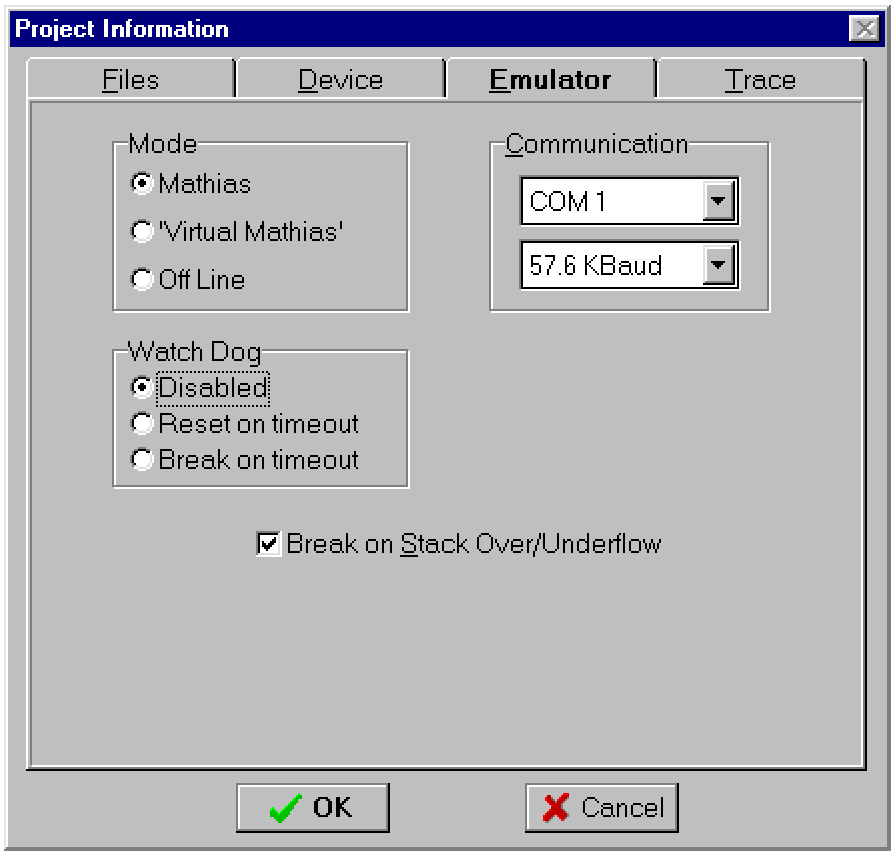
Step 4:

Select Project from the Setup menu.


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Environment
Project
Data Breakpoints
Status: Use File.New Project or File.Open Project...

Step 5:

Select the serial com port that the emulator is connected to, as well as the baud rate for communication with the PC.



Mode
- [✓ Mathias]
- 'Virtual Mathias'
- Off Line

Communication
- COM 1
- 57.6 KBaud

Watch Dog
- [✓ Disabled]
- Reset on timeout
- Break on timeout

[✓ Break on Stack Over/Underflow]

(Note: If the Emulator "Mode" is set to "Mathias", the software will attempt to find an emulator on the specified port when this dialog is closed. If an emulator is found, TDE will show the message "Status: Ready"; otherwise, it will indicate that no emulator was found.)

5X Note:
Stack Viewing and the "Break on Stack Overflow/Underflow" options are not available on 5X devices.

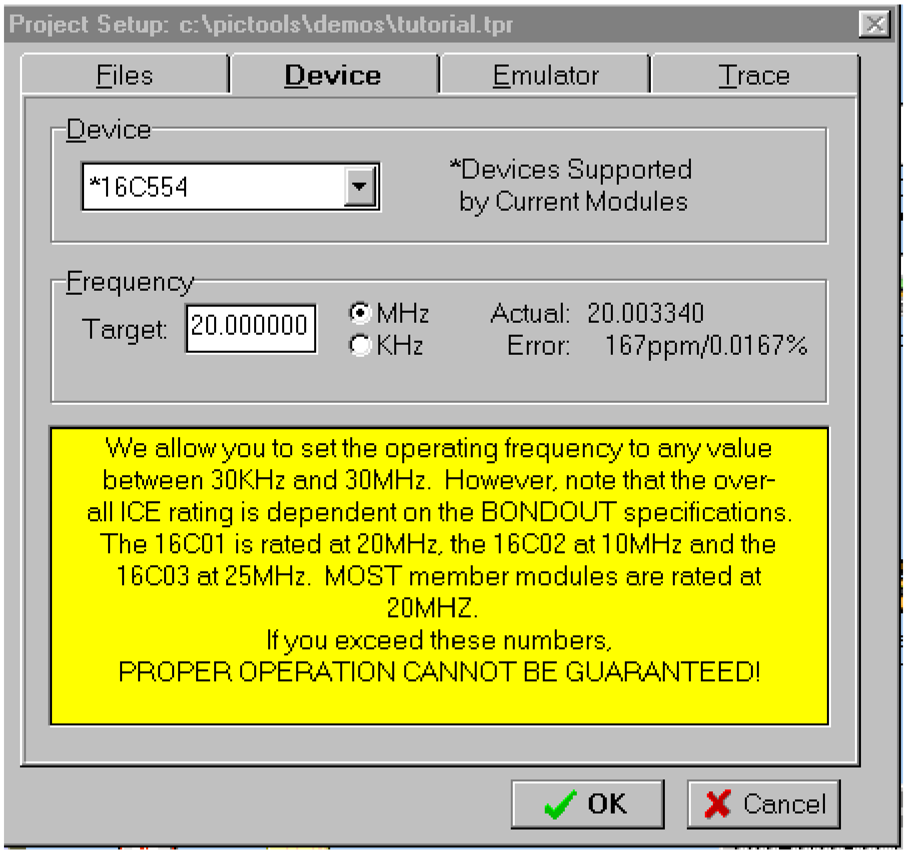
3.1.5 Device Settings, Building the Project

Step 6:

Under the Device heading, select the following settings:

Device: 16C554

Frequency : 20 MHz.



Device

*16C554

*Devices Supported by Current Modules

Frequency

Target: 20.000000 [✓ MHz] KHz

Actual: 20.003340
Error: 167ppm/0.0167%

We allow you to set the operating frequency to any value between 30KHz and 30MHz. However, note that the over-all ICE rating is dependent on the BONDOUT specifications. The 16C01 is rated at 20MHz, the 16C02 at 10MHz and the 16C03 at 25MHz. MOST member modules are rated at 20MHZ.
If you exceed these numbers,
PROPER OPERATION CANNOT BE GUARANTEED!

The device types available (16C71, 16C74,...) depend upon the PIC family and PIC member modules in use on the emulator. Before this window is shown, TDE checks the emulator hardware to see what options are available. All available devices will be marked with an "*". If you do not have the necessary modules for 16C554 emulation or if you are not in Virtual Mathias mode, then you will not see the asterisk beside the device type.

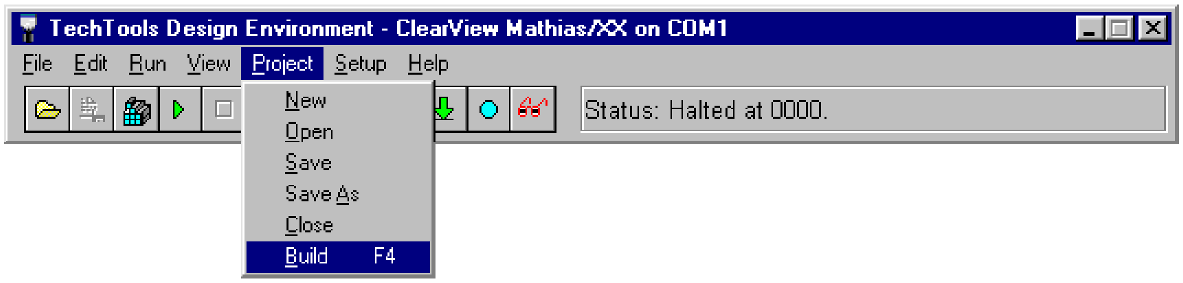
This project does not use the optional trace buffer, so there i s no need to adjust any settings under the fourth heading (Trace).

Now click the OK button to close this dialog.

Step 7:

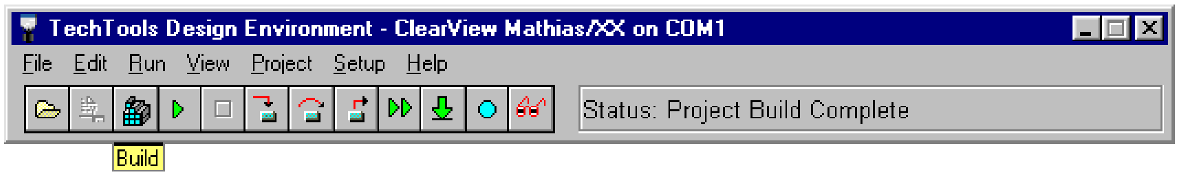
Select Build from the Project menu, or click the equivalent button.

F4


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
New
Open
Save
Save As
Close
Build F4

Status: Halted at 0000.

3.1.6 Running the Code, Stopping Execution


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Build

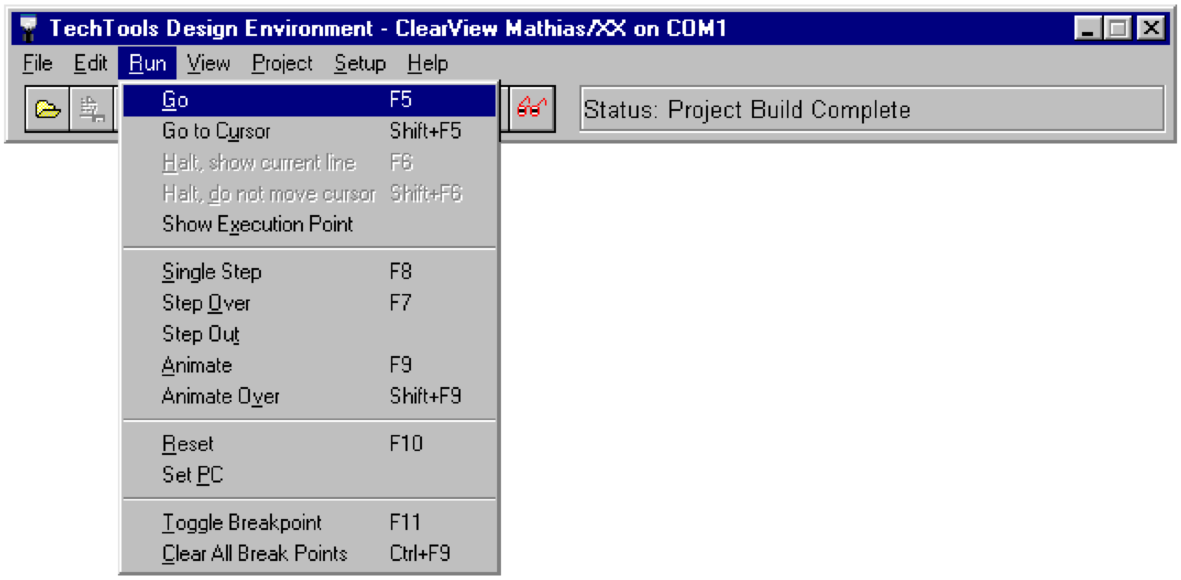
Status: Project Build Complete

If the build function worked properly, you should see the status bar as above. Note the status "Project Build Complete."

Step 8:

Select Go from the Run menu, or click the equivalent button.

F5


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

Status: Project Build Complete

This causes the emulator to run your assembled program.

If all goes well, the status bar should display several quick messages, such as "downloading code" and "resetting emulator." The final status message should be "Running..."

Selecting Go is the only way to make the emulator ru n at full hardware speeds (20 MHz, etc). During full-speed execution, the screen is not updated. To watch changes during execution, you must select single-step or animate, which are slower modes of execution. Single-step and animate are shown in the foll owing pages.

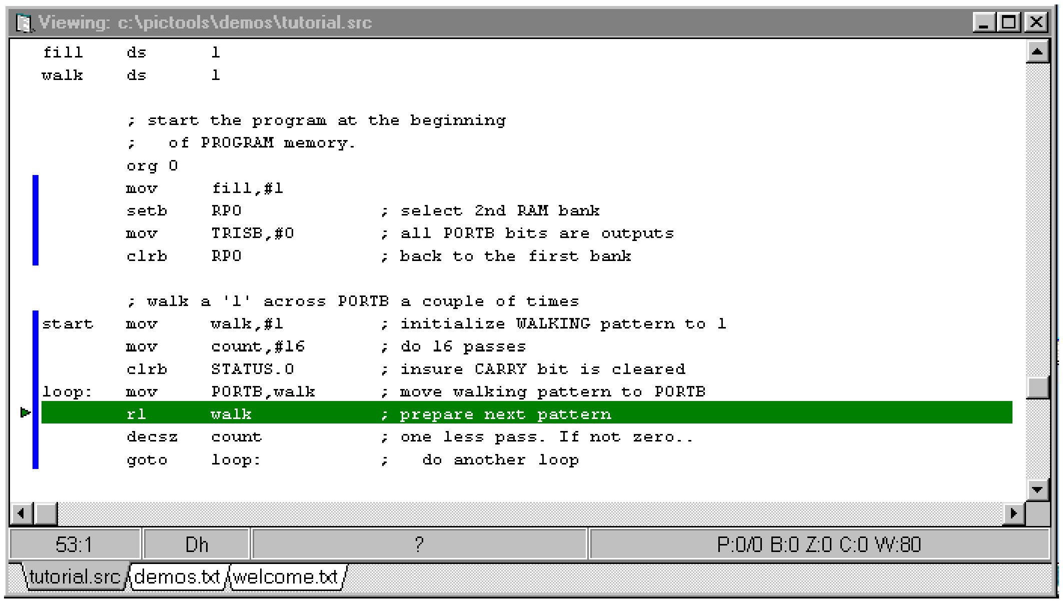
Step 9:

Let the emulator run for several seconds, and then click the stop button.

□ F6

After you click the Stop button, you should see something like the screen below. The highlighted line (green) indicates where program execution stopped.

The status line (which is in the toolbar, but not shown here) will indicate the current address, such as "User Halted at 000Dh".



```
fill    ds      1
walk    ds      1

        ; start the program at the beginning
        ;   of PROGRAM memory.
        org 0
        mov     fill,#1
        setb    RP0             ; select 2nd RAM bank
        mov     TRISB,#0        ; all PORTB bits are outputs
        clrb    RP0             ; back to the first bank

        ; walk a '1' across PORTB a couple of times
start   mov     walk,#1         ; initialize WALKING pattern to 1
        mov     count,#16       ; do 16 passes
        clrb    STATUS.0        ; insure CARRY bit is cleared
loop:   mov     PORTB,walk      ; move walking pattern to PORTB
        rl      walk            ; prepare next pattern
        decsz   count           ; one less pass. If not zero..
        goto    loop:           ;   do another loop
```

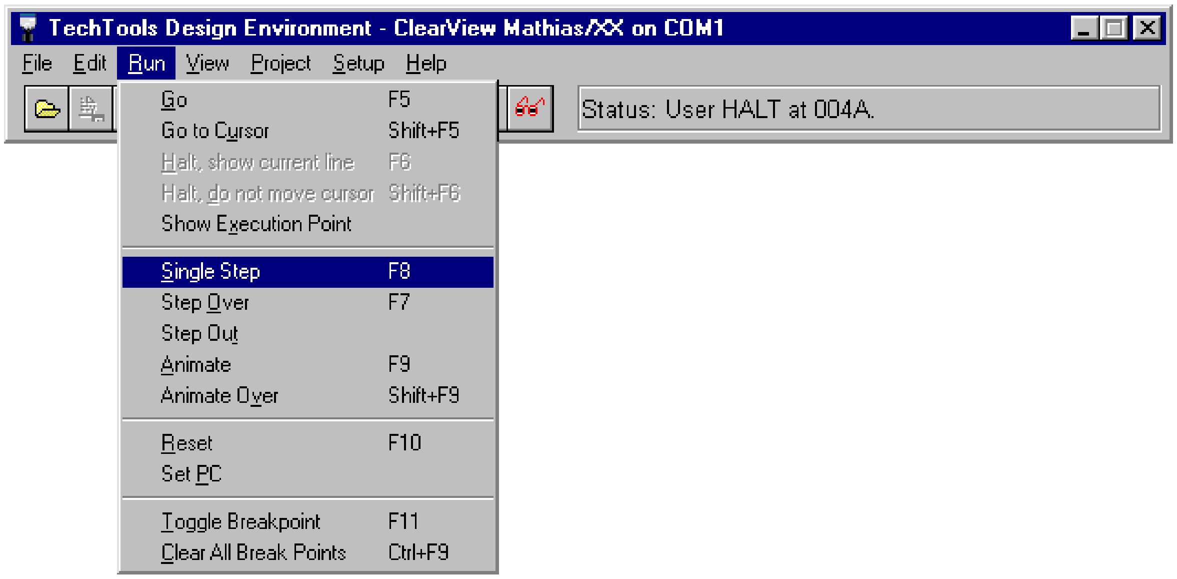
```
53:1 | Dh | ? | P:0/0 B:0 Z:0 C:0 W:80
```

3.1.7 Single-Stepping

Step 10:

Select Single Step from the Run menu, or click the equivalent button.

F8


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

Status: User HALT at 004A.

This causes the emulator to execute one instruction in your program, and then stop. As each instruction is executed, you can watch variables and registers, as well as your target circuit. This ability makes it easier to find the exact instruction that may be causing an error.

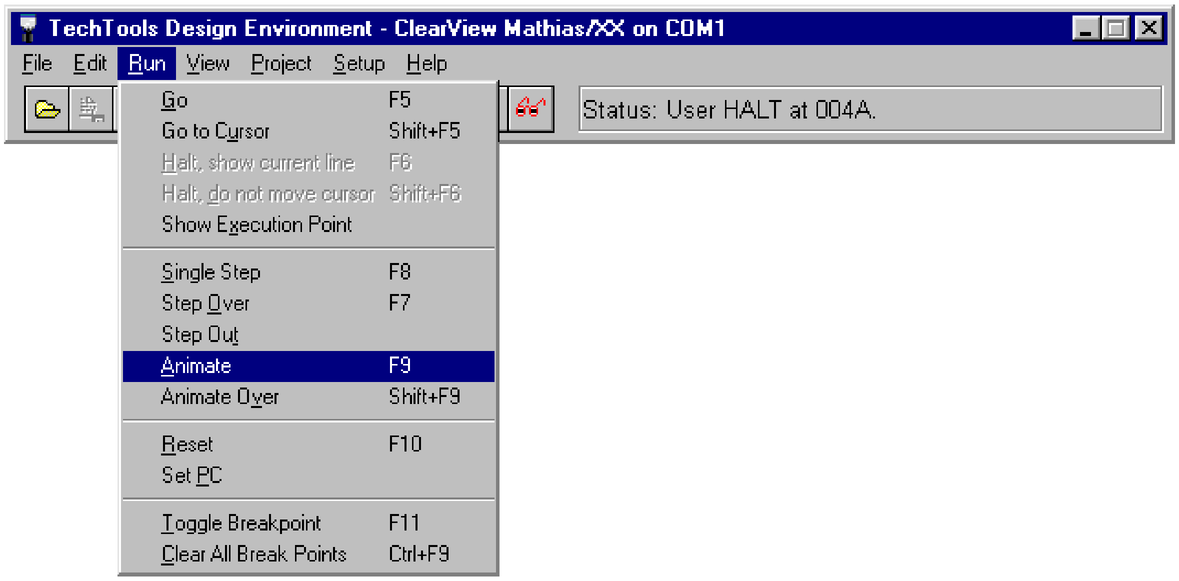
Try single-stepping several times. The green highlighted line should advance each time you step through the code.

3.1.8 Animation, Watching Variable Values

Step 11:

Select Animate from the Run menu, or click the equivalent button.

F9


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Status: User HALT at 004A.
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

This causes the emulator to execute one instruction (much like single-stepping), but execution does not stop after one. The screen is updated after each instruction, so you can see the status of variables, registers, etc. as your program runs.

Step 12:

After a few seconds, click the Stop button to stop execution of the program.

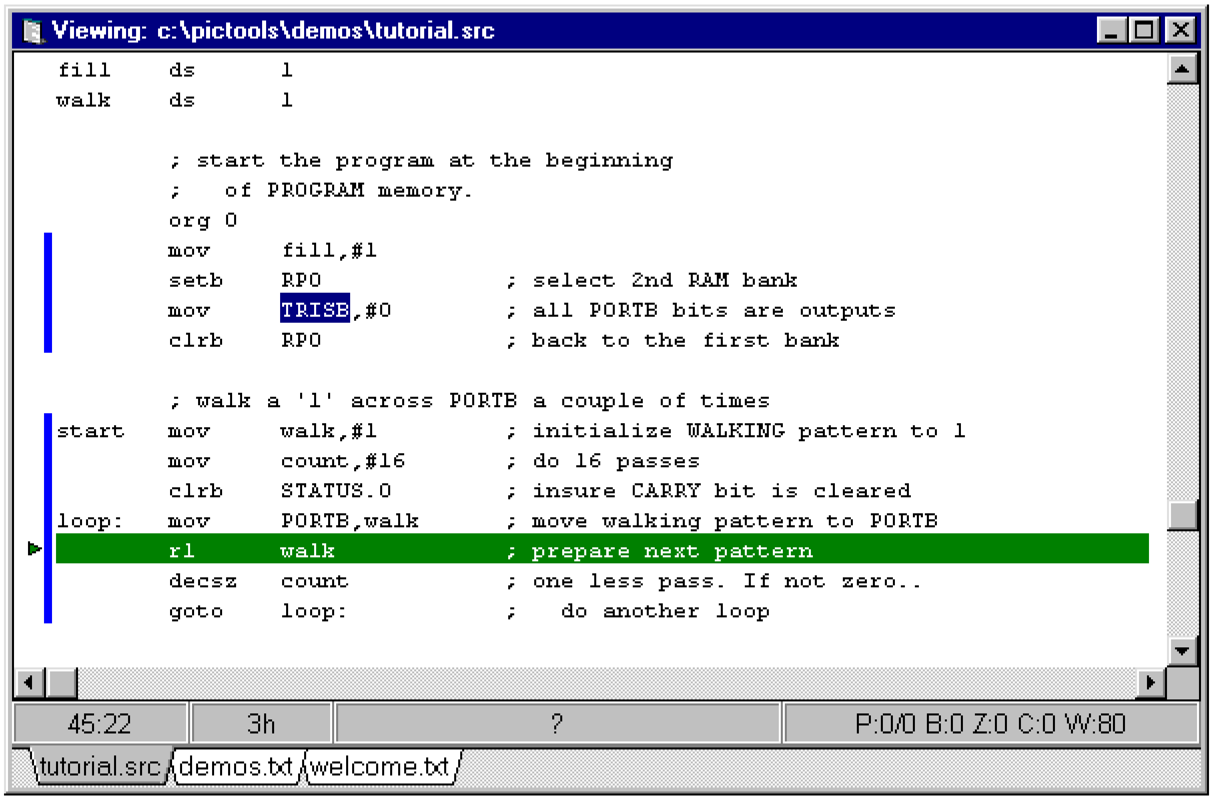
☐ F6

Step 13:

Add a Variable to the Watch Window

When you stopped execution, the program probably stopped on a line where you can see the variable called Value. Let’s add this variable to the watch list, so you can watch its value as the program runs.

Start by highlighting the variable’s name in the source code window.



```
fill     ds       1
walk     ds       1

         ; start the program at the beginning
         ;   of PROGRAM memory.
         org 0
         mov      fill,#1
         setb     RP0              ; select 2nd RAM bank
         mov      TRISB,#0         ; all PORTB bits are outputs
         clrb     RP0              ; back to the first bank

         ; walk a '1' across PORTB a couple of times
start    mov      walk,#1          ; initialize WALKING pattern to 1
         mov      count,#16        ; do 16 passes
         clrb     STATUS.0         ; insure CARRY bit is cleared
loop:    mov      PORTB,walk       ; move walking pattern to PORTB
         rl       walk             ; prepare next pattern
         decsz    count            ; one less pass. If not zero..
         goto     loop:            ;   do another loop
```

```
45:22    3h    ?    P:0/0 B:0 Z:0 C:0 W:80
```

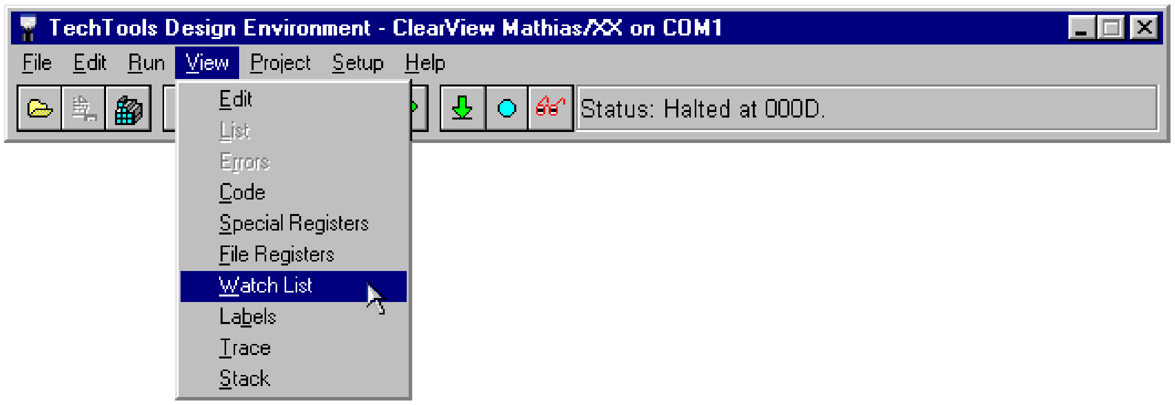
Watch

Now Right-Click in the editor and choose "Add Watch Selected". This will add the highlighted variable to the current watch group displayed in the Watch List. To open the Watch List use the Open Watch button or select "Watch List" from the VIEW menu.

3.1.9 Watching Variables, Viewing Special Registers

Step 14:

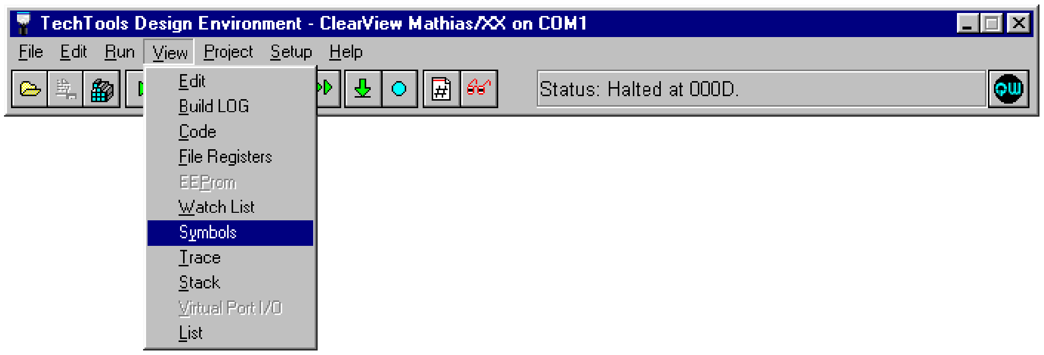
Now select the Watch List option from the View menu.



A small Watch window should appear. This window shows the values of all variables that have been selected for "watching." At this time, only one variable (Value) should appear in the window.

Step 15:

Select Special Registers from the View menu.

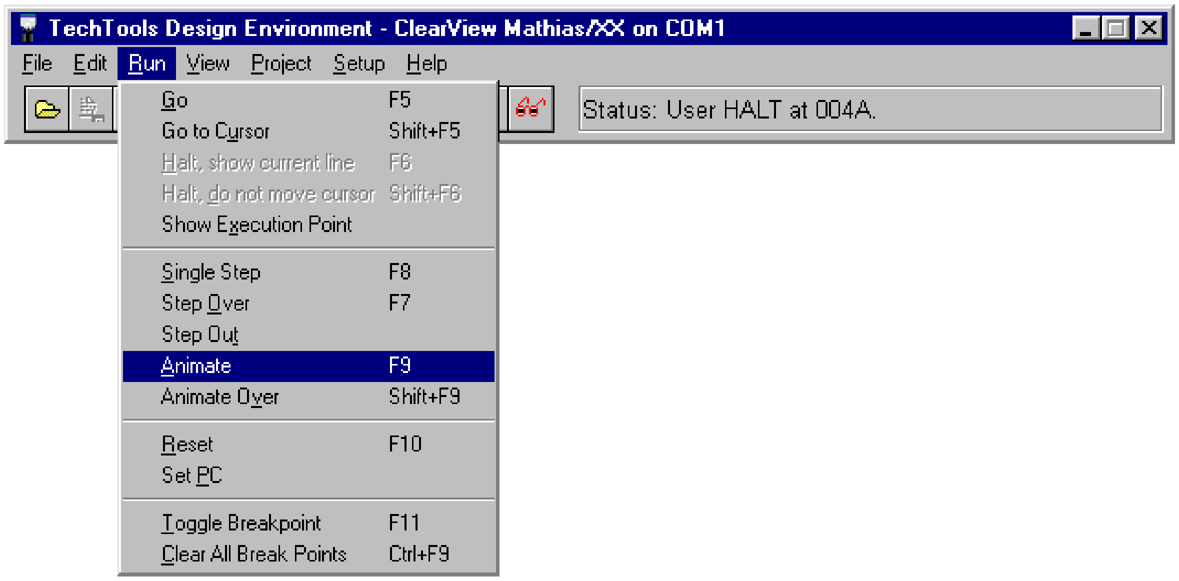


This opens the Symbols Window which displays the Special Registers, user variables and etc. The Special Registers portion displays the contents of the PIC's special register memory (status, fsr, option, etc).

Step 16:

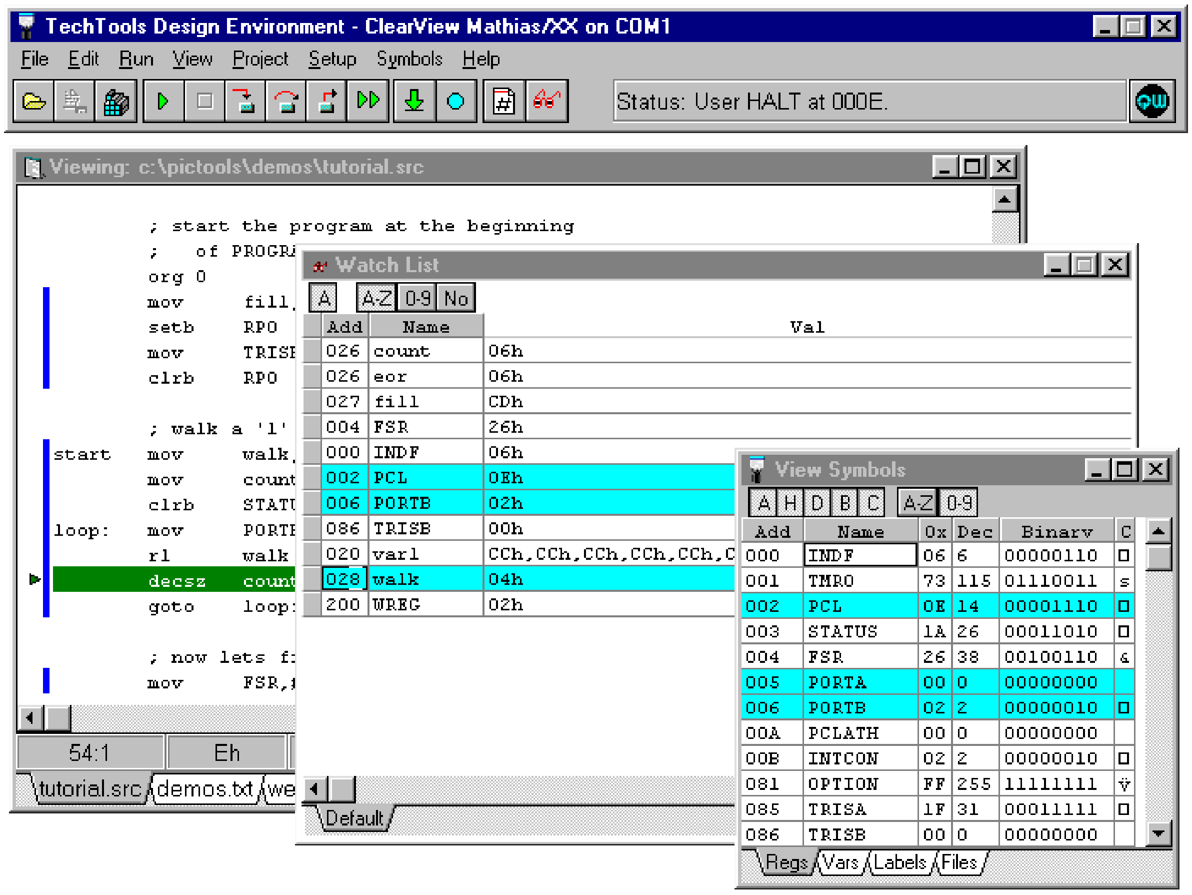
Select Animate from the Run menu, or click the equivalent button.

F9


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Status: User HALT at 004A.
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

3.1.10 Watching the Program Run

The screen should look something like this. The green highlighted line should be stepping through the program, and the two new windows (Watch & Special Registers) should be updated and values highlighted as they change.


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Symbols Help
Status: User HALT at 000E.
Viewing: c:\pictools\demos\tutorial.src
Watch List
View Symbols
54:1
Eh
tutorial.src
demos.txt
Default
Regs
Vars
Labels
Files

```
        ; start the program at the beginning
        ;   of PROGR
        org 0
        mov     fill,
        setb    RP0
        mov     TRISE
        clrb    RP0

        ; walk a '1'
start   mov     walk,
        mov     count
        clrb    STATU
loop:   mov     PORTE
        rl      walk
        decsz   count
        goto    loop:

        ; now lets f:
        mov     FSR,#
```

Add	Name	Val
026	count	06h
026	eor	06h
027	fill	CDh
004	FSR	26h
000	INDF	06h
002	PCL	0Eh
006	PORTB	02h
086	TRISB	00h
020	var1	CCh,CCh,CCh,CCh,CCh,C
028	walk	04h
200	WREG	02h

Add	Name	0x	Dec	Binary	C
000	INDF	06	6	00000110	□
001	TMR0	73	115	01110011	s
002	PCL	0E	14	00001110	□
003	STATUS	1A	26	00011010	□
004	FSR	26	38	00100110	&
005	PORTA	00	0	00000000	
006	PORTB	02	2	00000010	□
00A	PCLATH	00	0	00000000	
00B	INTCON	02	2	00000010	□
081	OPTION	FF	255	11111111	ÿ
085	TRISA	1F	31	00011111	□
086	TRISB	00	0	00000000	

Step 17:

After a few seconds, click the Stop button to stop execution of the program.

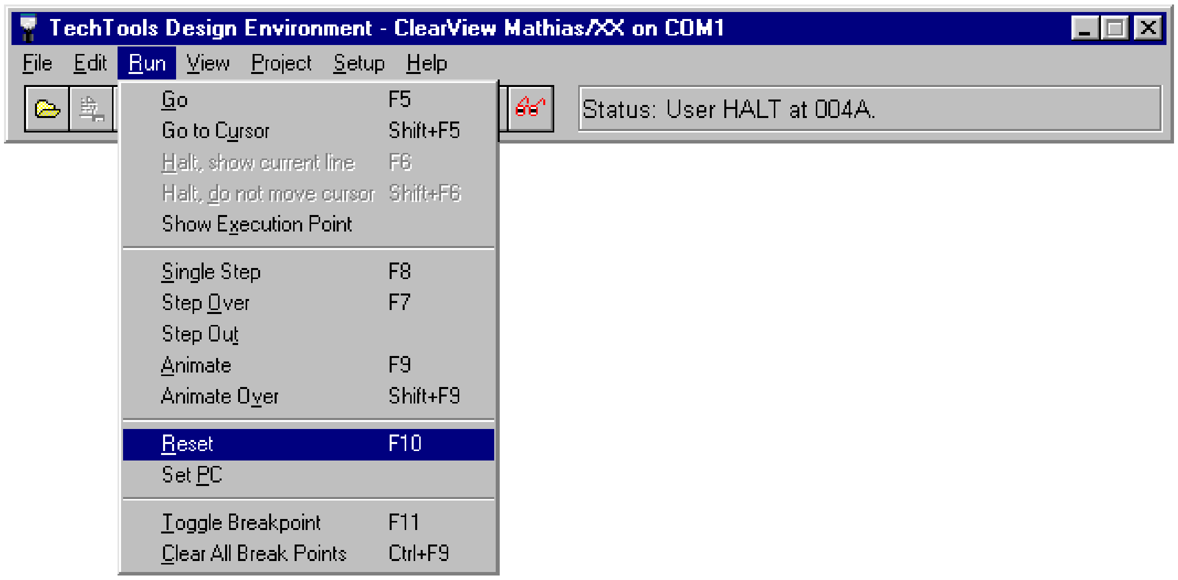
□ F6

3.1.11 Resetting the Emulator, Setting Breakpoints

Step 18:

Select Reset from the Run menu, or click the equivalent button.

○ **F10**


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Status: User HALT at 004A.
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

This causes the emulator to reset the PIC's Program Counter (PC). The reset value of the Program Counter may vary with different PICs. In the case of the 16C74, the reset value is 000h, so program execution will st art at 000h whenever a reset occurs.

After you select Reset, the green highlighted line should move to the beginning of the program.

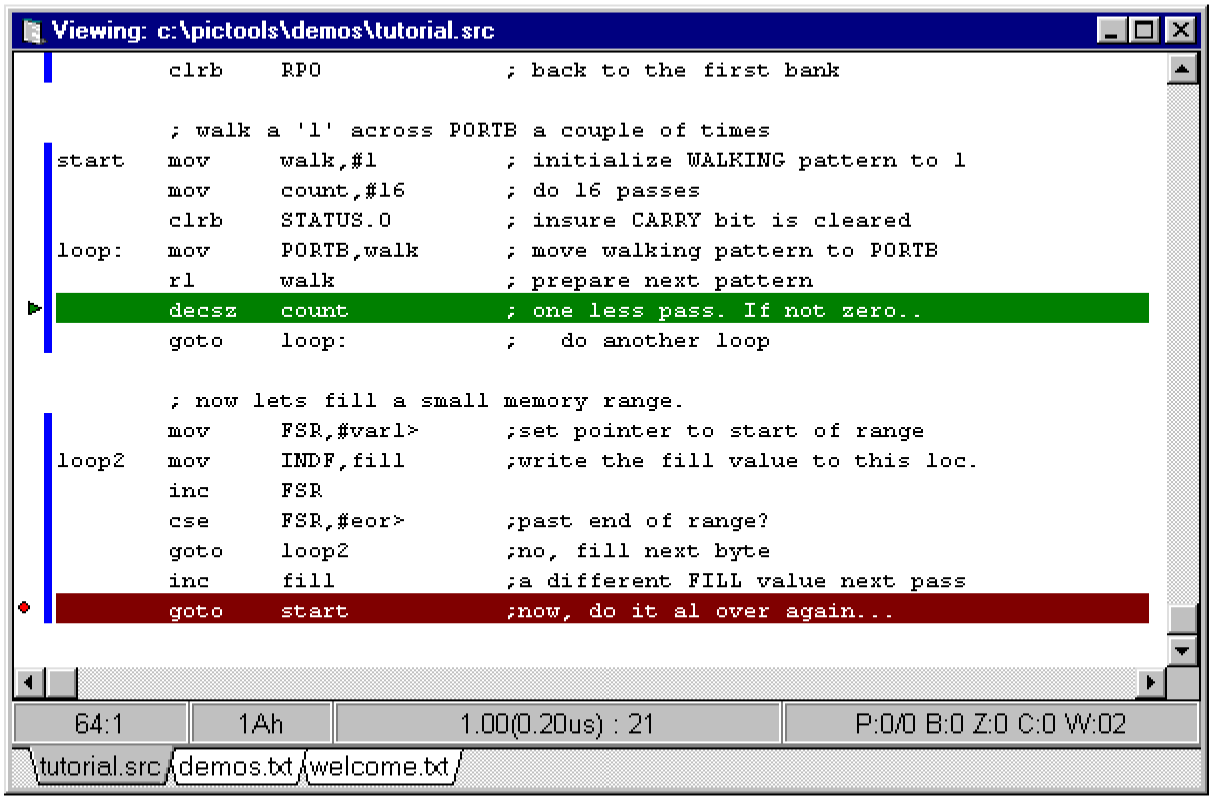
Step 19:

Scroll to the last line in the program, and then double-click to the left of the line.

This sets a breakpoint on the last line.

If the emulator enc ounters a breakpoint while running a program, it will stop execution, update any information on the screen and highlight any changed values (variables, registers, etc).

Note that lines with breakpoints are highlighted in red. If the "current execution line" (green) lands on a breakpoint line, the resulting color will be blue (green + red = blue).


Viewing: c:\pictools\demos\tutorial.src

```
        clrb    RP0             ; back to the first bank

        ; walk a '1' across PORTB a couple of times
start   mov     walk,#1         ; initialize WALKING pattern to 1
        mov     count,#16       ; do 16 passes
        clrb    STATUS.0        ; insure CARRY bit is cleared
loop:   mov     PORTB,walk      ; move walking pattern to PORTB
        rl      walk            ; prepare next pattern
        decsz   count           ; one less pass. If not zero..
        goto    loop:           ;   do another loop

        ; now lets fill a small memory range.
        mov     FSR,#var1>      ;set pointer to start of range
loop2   mov     INDF,fill       ;write the fill value to this loc.
        inc     FSR
        cse     FSR,#eor>       ;past end of range?
        goto    loop2           ;no, fill next byte
        inc     fill            ;a different FILL value next pass
        goto    start           ;now, do it al over again...
```

```
64:1 | 1Ah | 1.00(0.20us) : 21 | P:0/0 B:0 Z:0 C:0 W:02
```

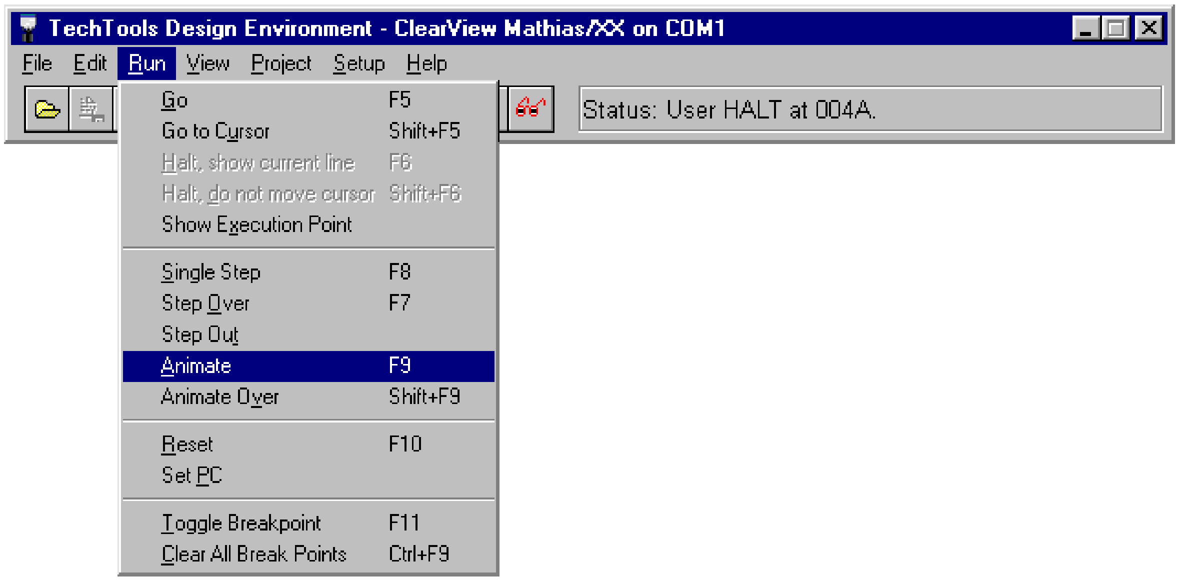
To disable a breakpoint, just double-click in the left column again.

3.1.12 Finishing the Project

Step 20:

Select Animate from the Run menu, or click the equivalent button.

F9


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

Status: User HALT at 004A.

The emulator should step through each line of the program, stopping on the last line (normally, it would loop back to the start of the program). Because the last line is the current execution line and a breakpoint line, it should be highlighted in blue.

Step 21:

At this point, the sample project is done.

If you want to take a coffee break, select Exit from the File menu. All project info will be saved automatically.

3.2 TDE Details

The following pages describe the various features of ClearView's TDE software. Unlike the previous pages, this section includes a detailed description of each software function. The following topics are covered :

Project and File Operations 45
- Creating a Single-file Project 46
- Creating a Multiple-file Project 50
- Adding Files 51
- Mode of Operation 55
- Watch Dog Timer 57
- Breaking on Stack Over/Underflow 57
- Device Settings 59
- Trace buffer settings 60
- Environment Settings 62
- Resetting the emulator 66

Printing 67

Compiling and Executing Code 68
Building the project 68
Executing code at full speed 70
Halting execution 71
Single-step , step-over and step-out execution 71
Animation-mode and "Animate Over" execution 74
Showing the execution point 75
Resetting the emulated PIC 76
Setting the program counter (execution point 77)
Using breakpoints 78

Using TDE's Windows 80
Window basics 80
Text editor window 82
Build Log Window 93
Code window 95
Special registers window 97
File registers window 105
Watch list window 108
Trace buffer window 116

Data Breakpoints 118
Enable and Break On options 120
File Register options 120
Match On options 121
Include Constants option 123
Breaking on Data 124

3.2.1 Project and File Operations

The following sections explain how to create single and multi-file projects, how to work with various file formats, and how to select device and emulator settings.

Creating a Single-file Project 46
Creating a Multiple-file Project 50
Adding Files 51
Mode of Operation 55
Watch Dog Timer 57
Breaking on Stack Over/Underflow 57
Device Settings 59
Trace buffer settings 60
Environment Settings 62
Resetting the emulator 66
Printing 67

All compile or debug sessions in TDE require a project name.

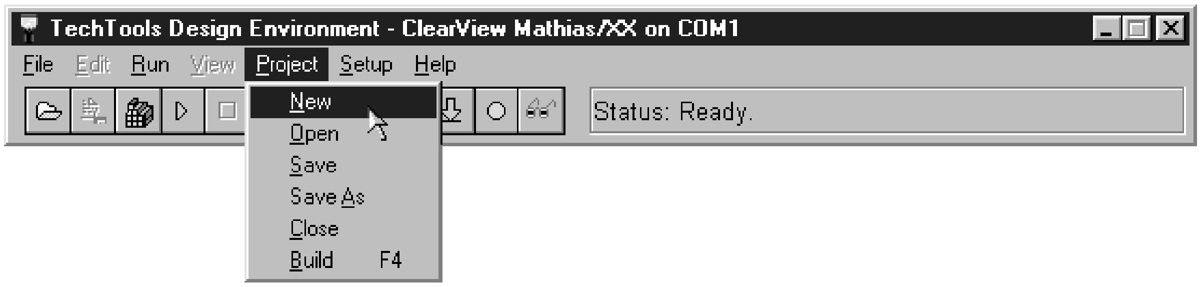
This next section explains how to create a single-file project using the included TechTools

PICmicro Assembler (CVASM16).

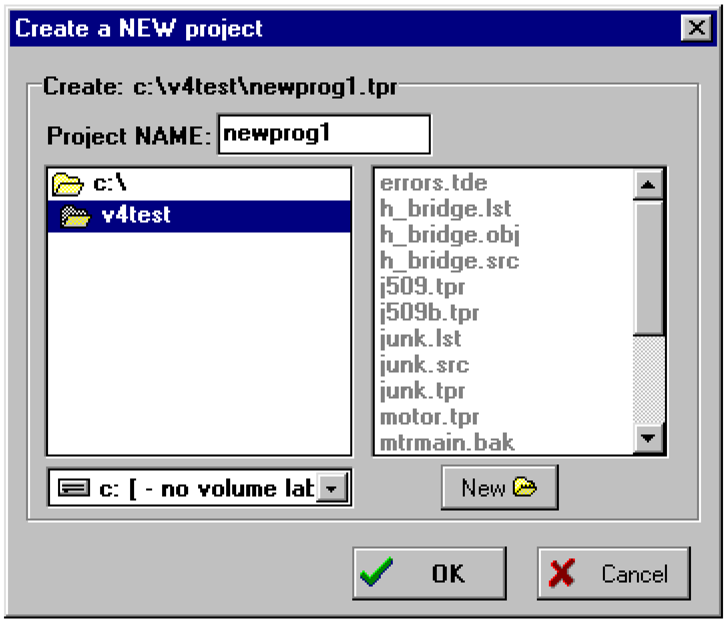
As mentioned in the sample project section, a "project" is a control file that stores all the information needed to integrate your work with the emulator. In addition to source code, the project file contains setup and environment information. When you close a project or exit from TDE, these extra settings are saved in a separate file with the ".tpr" extension. When you later open an existing project, TDE loads these settings, so you don't have to re-establish the emulator type, oscillator speed, editor settings, etc.

3.2.1.1 Creating a Single-file Project

To create a new project, select New Project from the Project menu.



TDE will open the Project file dialog. Here you can type a new project name. For this example we will use the name " Newprog1". If the file does not exist, TDE will ask if it is ok to create this new project filename.


Create a NEW project

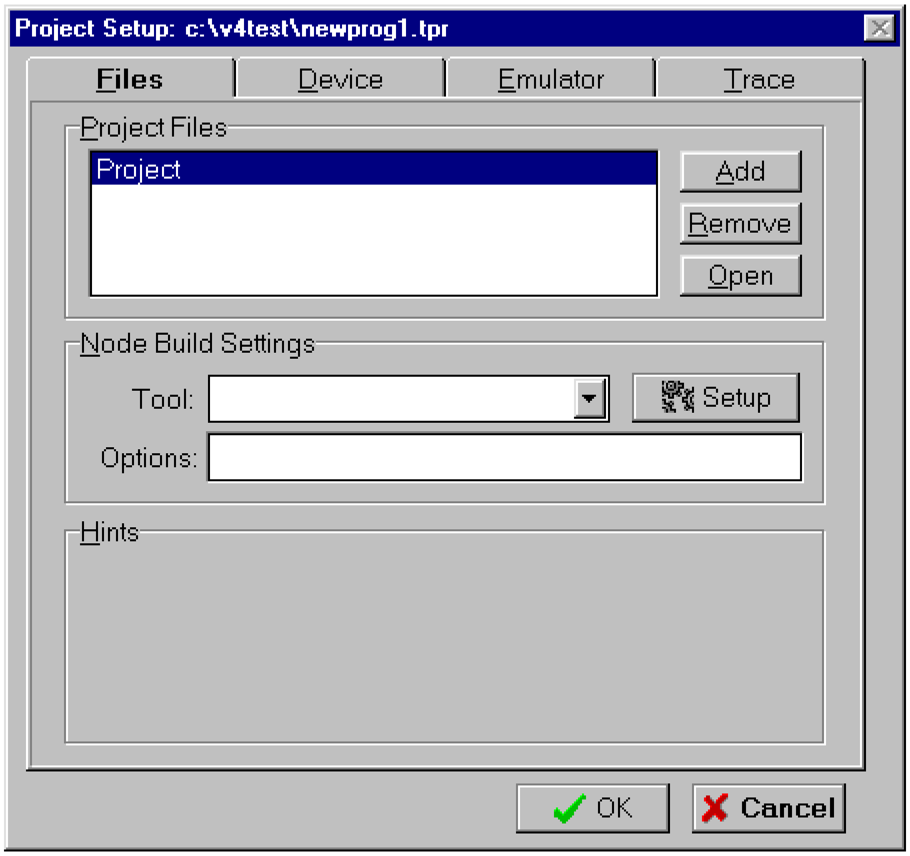
Create: c:\v4test\newprog1.tpr

Project NAME: newprog1

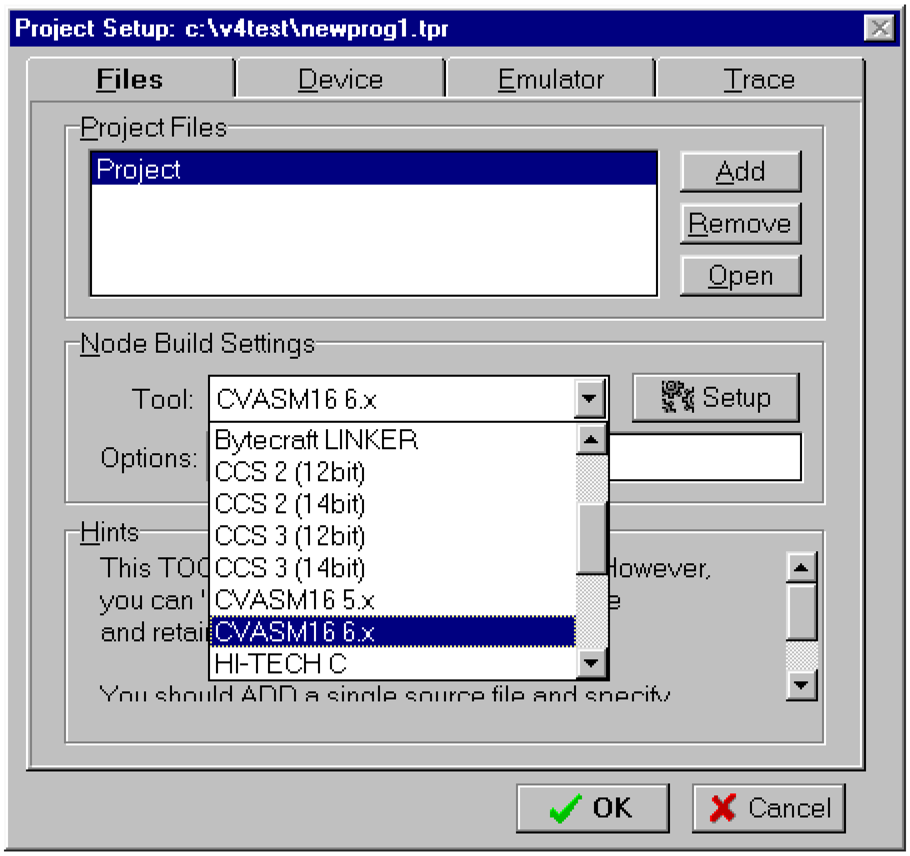
c:\
v4test

errors.tde
h_bridge.lst
h_bridge.obj
h_bridge.src
j509.tpr
j509b.tpr
junk.lst
junk.src
junk.tpr
motor.tpr
mtrmain.bak

After you click "OK", a dialog will appear like the one below.



The highlighted item "Project", is the main Node in your project file. You can now select the tool you would like to use for your project by clicking on the Tool drop-down list under Node Build Settings.This tool selection lets TDE know how to handle the build process.



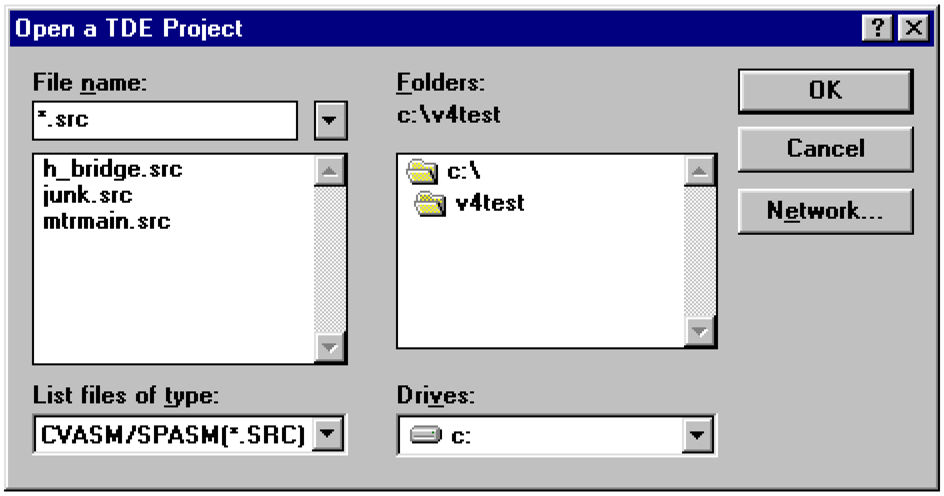
As shown above, you can select your favorite compiler or specify a particular file type such as a COD file.

3.2.1.1.1 Adding a File

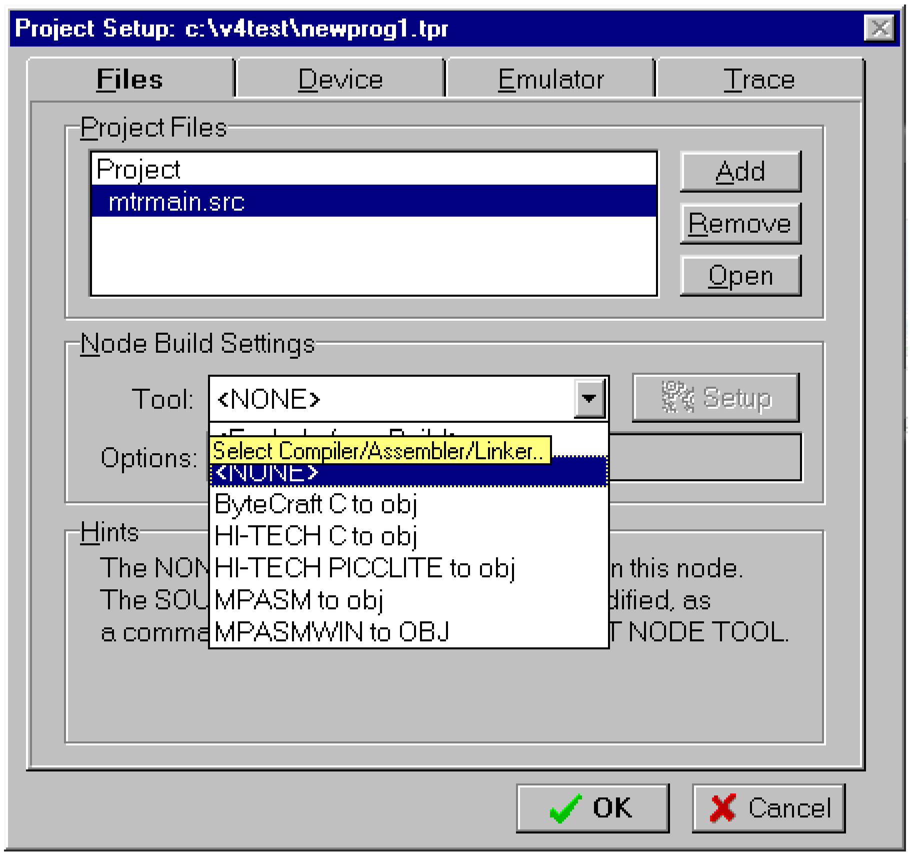
Project => Setup =>

Now that we have selected CVASM16 as the tool for our Project Node, lets add a file.

Click on the ADD button and a dialog will appear similar to the one shown below. Here you can select an existing file, or just type a new filename and TDE will create the file for you.



After selecting a file, your Project Dialog should resemble the graphic below.

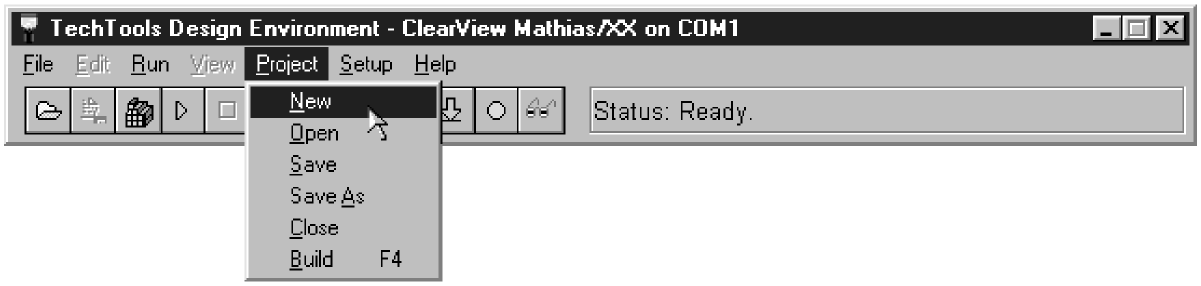


Now we can choose the appropriate tool for the selected file. Since we have chosen CVASM16 as our Project Node Tool, we will select NONE as the File Node Tool. This will pass the file to the Project Tool for processing.

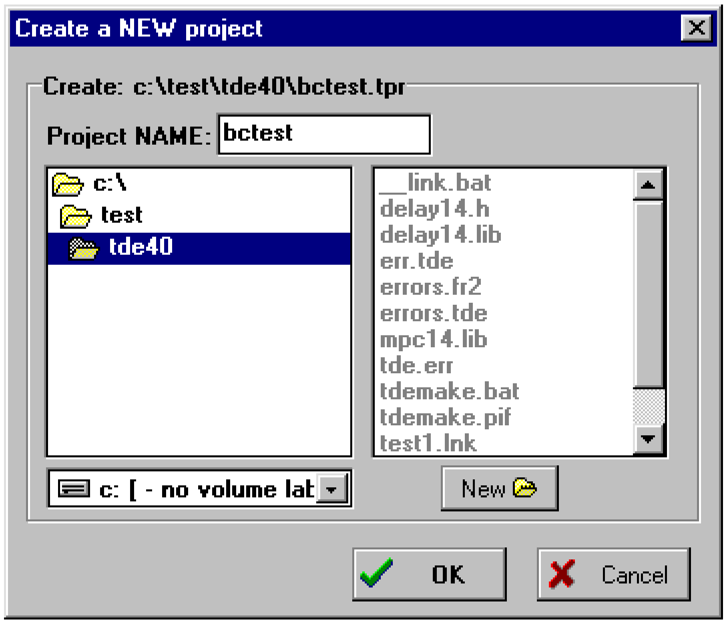
At this time you can either double-click on the File Node or select the Open button to open the file into TDE's editor window.

3.2.1.2 Creating a Multiple-file Project

To create a multi-file project, select New from the Project menu.



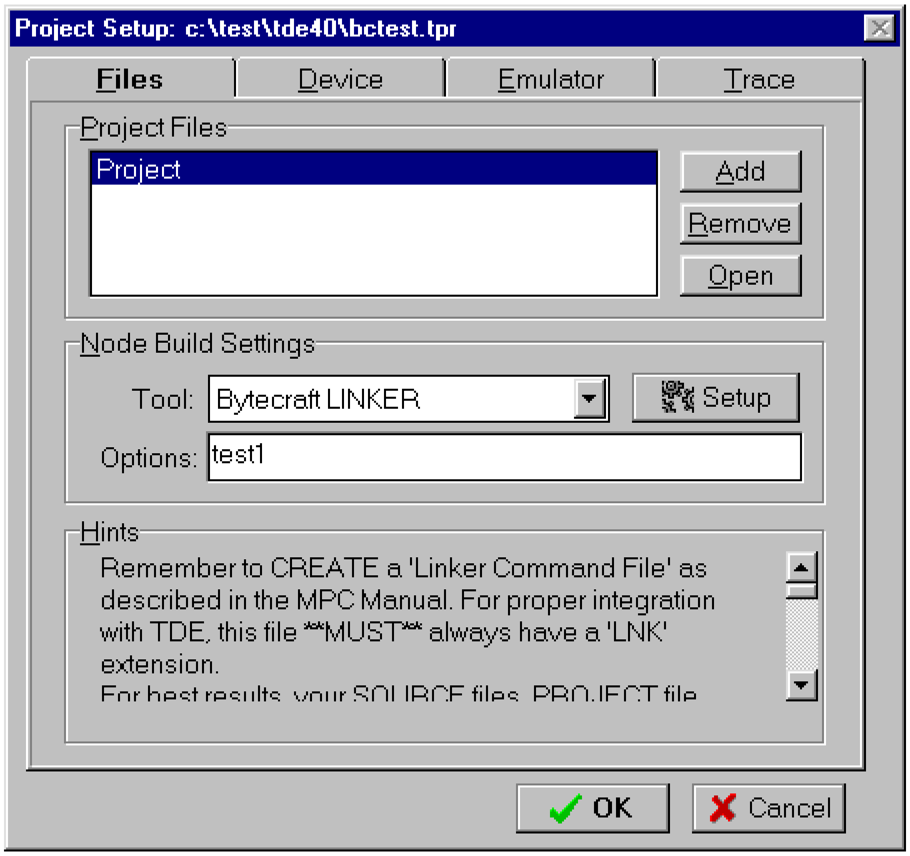
TDE will open the Project file dialog. Here you can type a new project name. For this example we will use the name " BCtest". TDE will verify that the name does not al ready exist in the selected directory.


Create a NEW project

Create: c:\test\tde40\bctest.tpr

Project NAME: bctest

After you click "OK", a dialog will appear like the one below. For this example we will select By tecraft Linker as our Project tool.


Project Setup: c:\test\tde40\bctest.tpr

Project Files

Project

Node Build Settings

Tool: Bytecraft LINKER

Options: test1

Hints

Remember to CREATE a 'Linker Command File' as described in the MPC Manual. For proper integration with TDE, this file **MUST** always have a 'LNK' extension.
For best results, your SOURCE files, PROJECT file

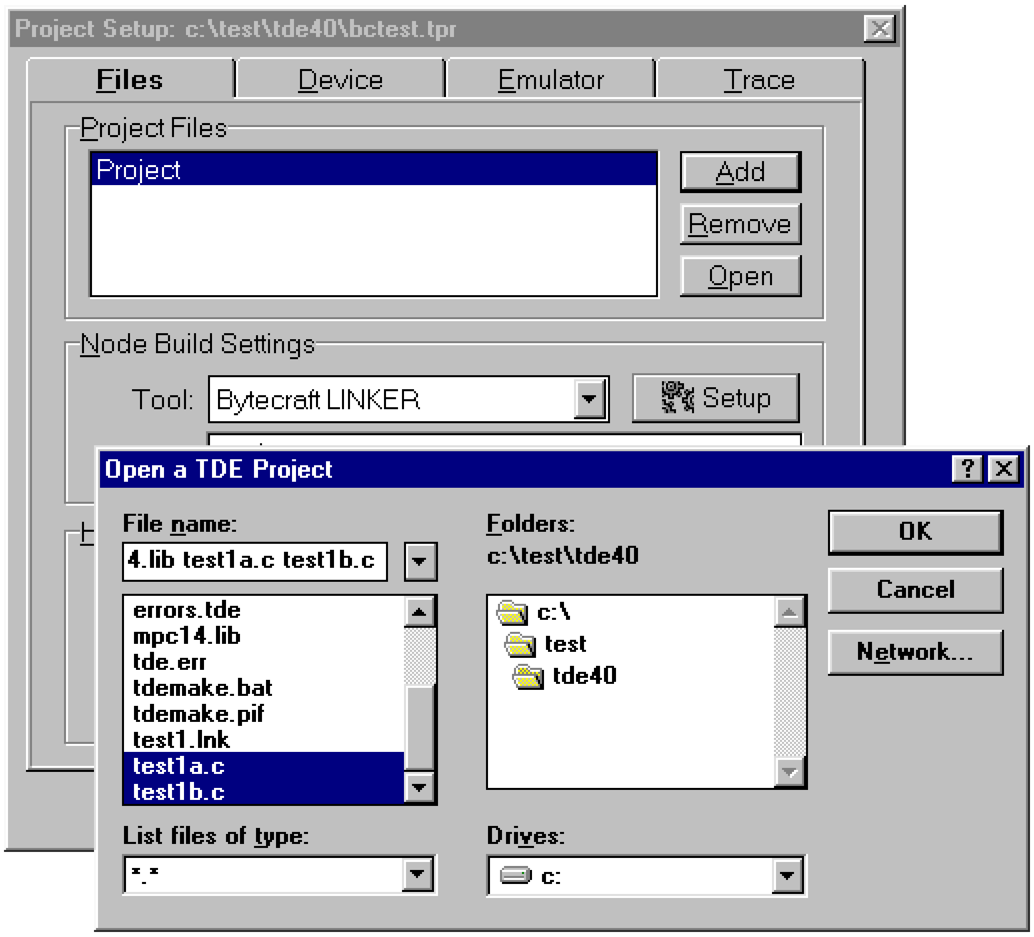
The highlighted item Project is the main Node in your project fi le. You can now select the tool you would like to use for your project by clicking on the Tool drop-down list under Node Build Settings. This tool selection lets TDE know how to handle the build process.

3.2.1.2.1 Adding Files

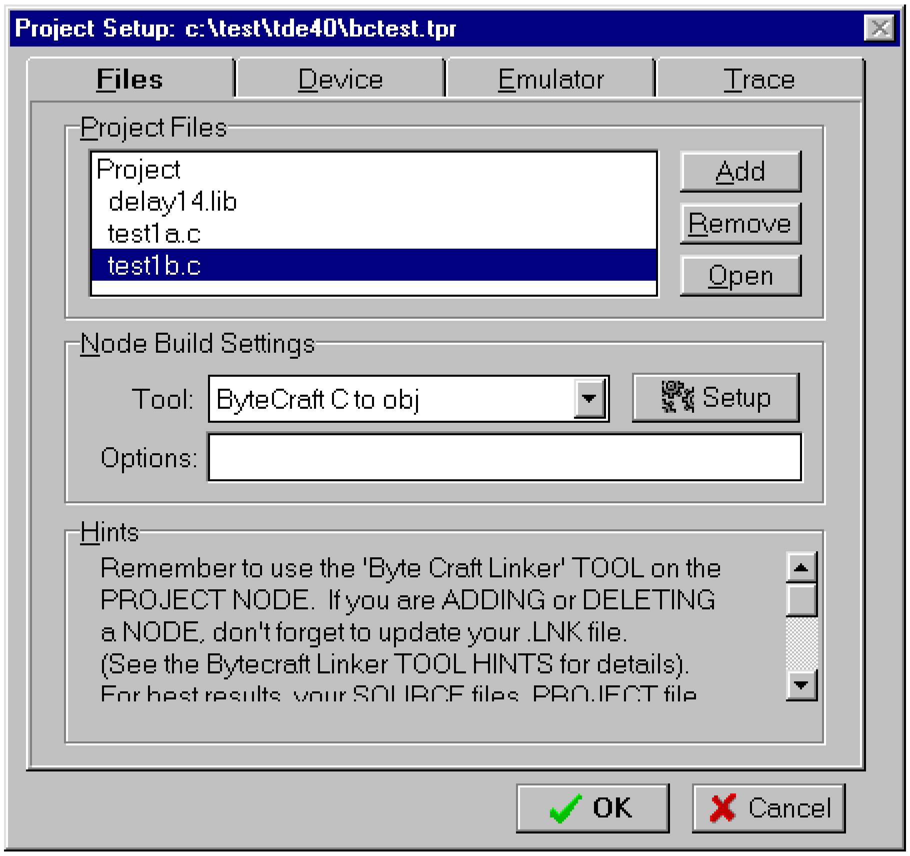
Setup => Project => **Files**

Now that we have selected Bytecraft Linker as the tool for our Project Node, let's add a couple of files.

Click on the ADD button and a dialog will appear similar to the one shown here. We have selected an existing file, but you can just type a new filename and TDE will create the file for you. Multiple files could also be selected by using the ctrl or shift keys.



After selecting a file, your Project Dialog should resemble the graphic below.
Now we can choose the appropriate tool for each file added to the project. Since we have chosen Bytecraft Linker as our Project Node Tool, AND added a 'C' source file, we will select Bytecraft C to OBJ as the File Node Tool.



Project
delay14.lib
test1a.c
test1b.c

Tool: ByteCraft C to obj

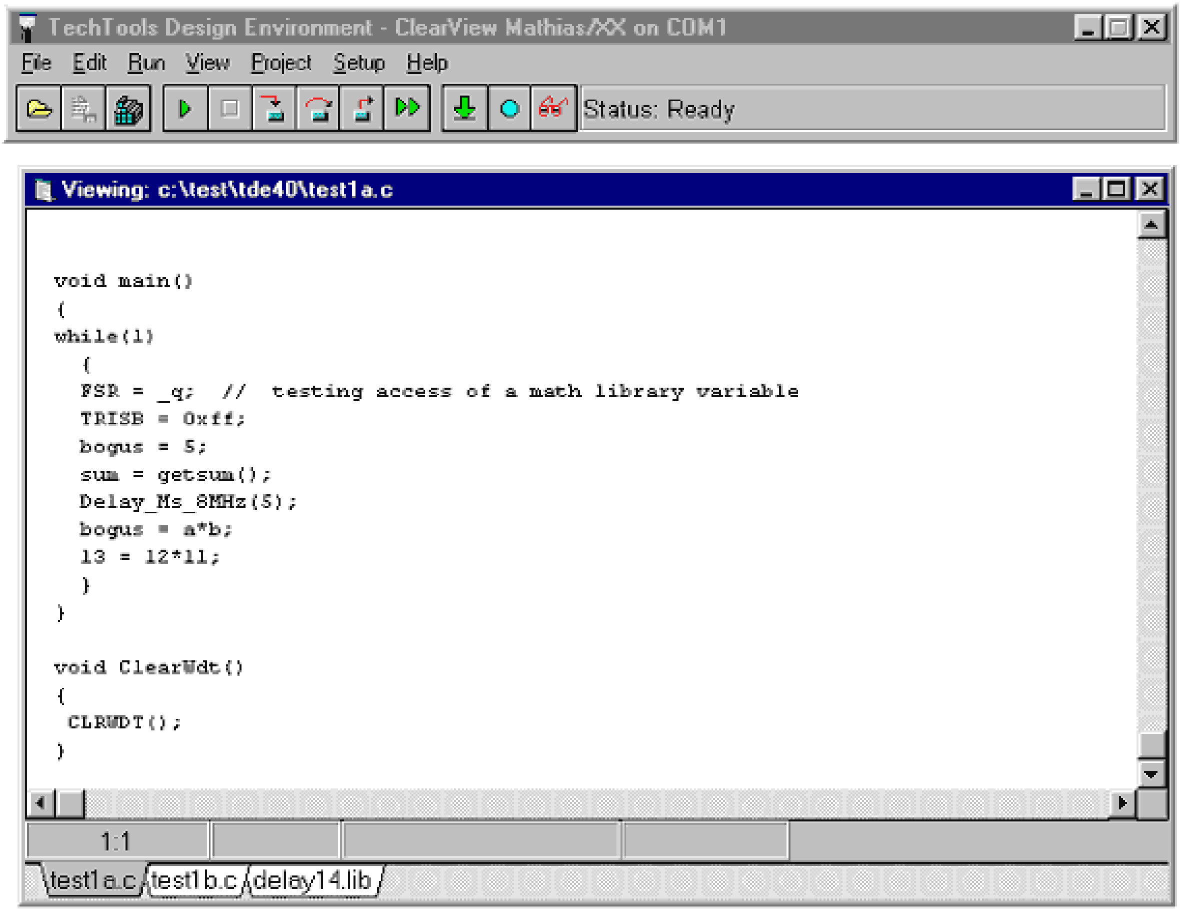
Options:

Remember to use the 'Byte Craft Linker' TOOL on the PROJECT NODE. If you are ADDING or DELETING a NODE, don't forget to update your .LNK file.
(See the Bytecraft Linker TOOL HINTS for details).
For best results, your SOURCE files, PROJECT file

At this time you can either double-click on the highlighted File Node or select the Open button to open the file into TDE's editor window.

NOTE:
The first file added to the project will be the last file processed during build operations. Likewise, the last file added to the project will be the first file processed.

After selecting OK, the Project Dialog will close and your Edit window should resemble the graphic below.

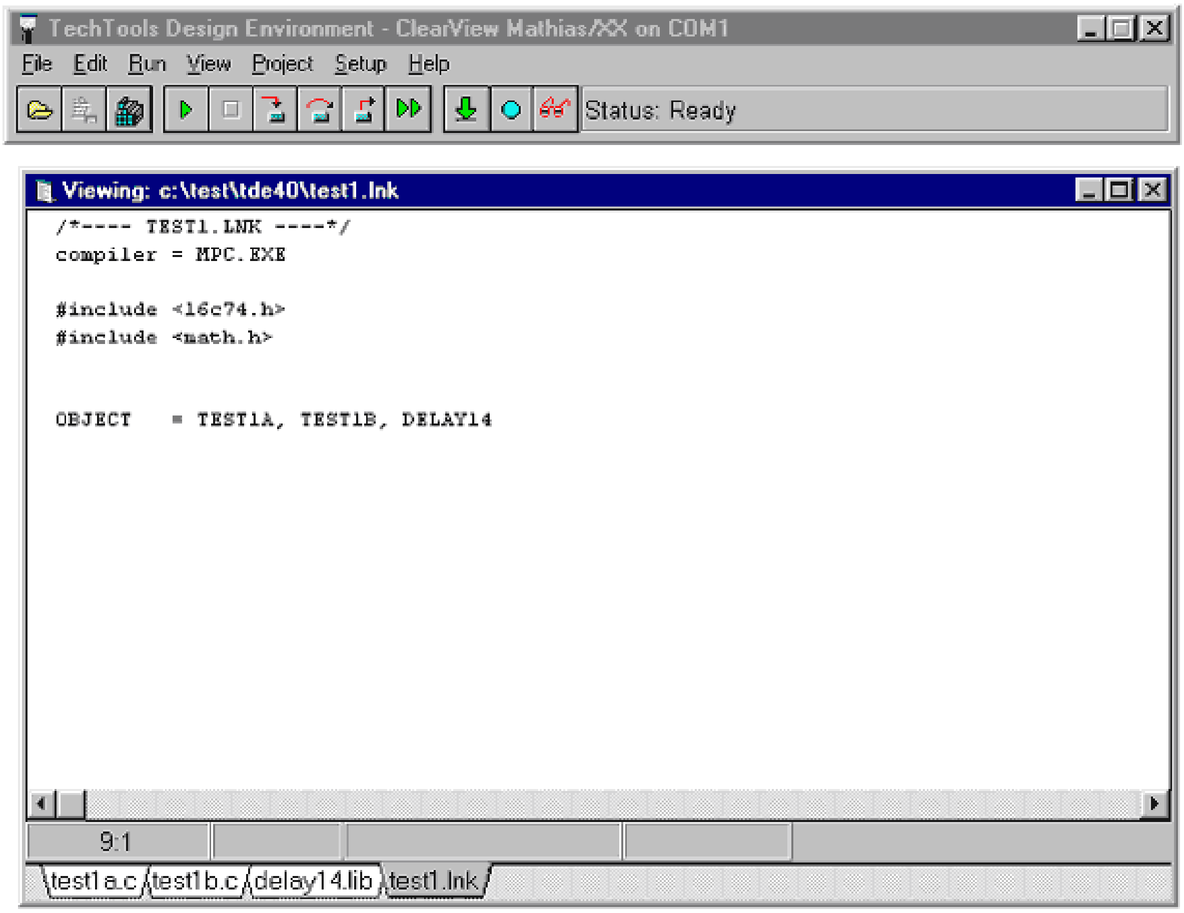

TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Status: Ready
Viewing: c:\test\tde40\test1a.c
1:1
test1a.c
test1b.c
delay14.lib

```
void main()
{
while(1)
  {
  FSR = _q;  //  testing access of a math library variable
  TRISB = 0xff;
  bogus = 5;
  sum = getsum();
  Delay_Ms_8MHz(5);
  bogus = a*b;
  l3 = l2*l1;
  }
}

void ClearWdt()
{
 CLRWDT();
}
```

Now we need to create a "Link" file for the Bytecraft Linker. Since the Link file is simply a text file, we can create it using the TDE editor.

Choose the New option from the File menu. This will popup a File Open Dialog. From here a new filen ame can be specified and TDE will create an empty text file and open it in the Editor window.

You will want to reference the Bytecraft Linker documentation for complete information regarding the requirements of the link file, however an exam ple is shown below.


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Status: Ready
Viewing: c:\test\tde40\test1.lnk
9:1
test1a.c
test1b.c
delay14.lib
test1.lnk

```
/*---- TEST1.LNK ----*/
compiler = MPC.EXE

#include <16c74.h>
#include <math.h>


OBJECT   = TEST1A, TEST1B, DELAY14
```

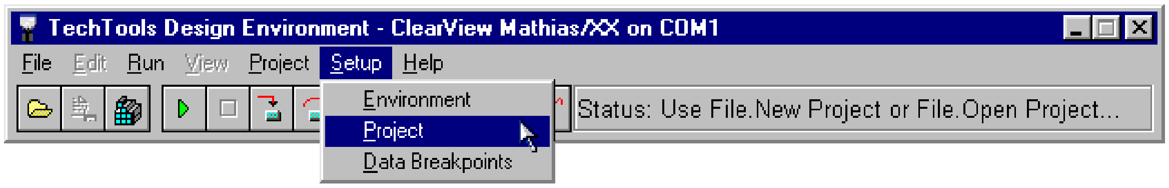
You can now edit, compile and debug your project.

NOTE: There is one restriction when using a link file with TDE:
Link files must have the extension of ".LNK".

3.2.1.3 Mode of Operation

Setup => Project => **Emulator**

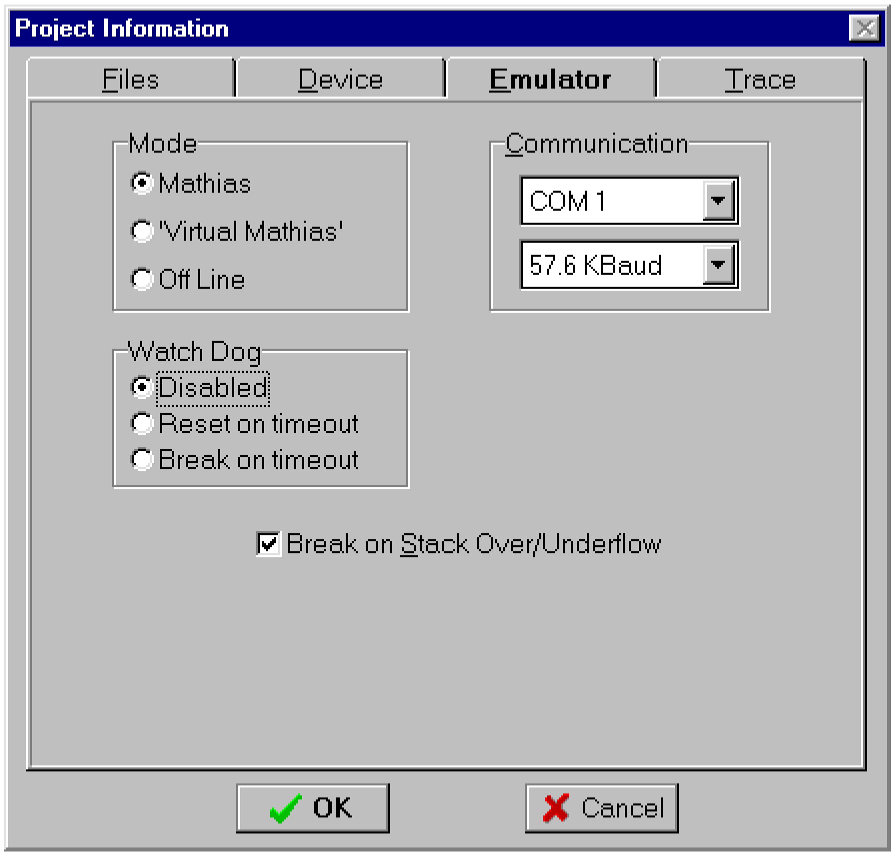
Before closing the Project Setup Window, you will want to verify a few other project options (These options can always be changed at a later time by selecting the Project option from the Setup menu in TDE).



Status: Use File.New Project or File.Open Project...

First, select the Emulator tab.

Now you can select the serial com port the emulator is connected to, as well as the baud rate for communication with the PC.



Mode: [✓ Mathias] 'Virtual Mathias' Off Line

Communication: COM 1; 57.6 KBaud

Watch Dog: [✓ Disabled] Reset on timeout Break on timeout

[✓ Break on Stack Over/Underflow]

Mode of Operation

Mathias:

Full Emulation mode with Mathias attached.

Virtual Mathias:

This allows you to start experiencing all of TDE and Mathias's features - even if you do not have a Mathias connected.

Off Line:

Editing and compiling mode.

Even if you have not designed your target yet, you can sti ll choose Mathias Mode and begin your development (Note: If the Emulator Mode is set to Mathias, the software will attempt to find an emulator on the specified port when this dialog is closed. If an emulator is found, TDE will show the message "Status: Ready"; otherwise, it will indicate that no emulator was found.)

If you do not have a Mathias connected, you can work in offline mode or use Virtual Mathias mode to start experiencing all of TDE and Mathias's features.

3.2.1.4 Watch Dog Timer

Setup => Project => Emulator

We recommend that you disable the WDT until the rest of your code is debug ged. When it is time to add the code necessary to keep the Watch Dog Timer satisfied (or to "kick the dog"), you will find the 'Break on Timeout' feature very time saving. This feature stops the program execution whenever the Watch Dog times out, al lowing you to evaluate the portion of code that needs to clear the WDT.

Disabled:

Turns the Watch Dog Timer OFF.

Reset on Timeout:

Allows the Watch Dog Timer to reset the PIC whenever a timeout occurs.

Break on Timeout:*

This new feature will halt program execution and display the current program line in the Editor window whenever a timeout occurs. (Always RESET the program after a Break on Timeout).

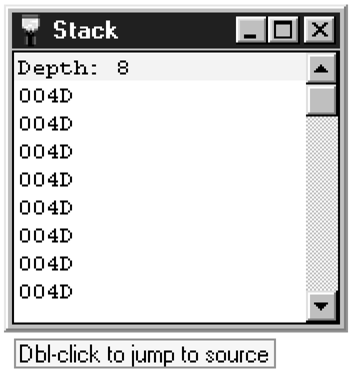
* Break on Stack Over/Underflow, and WDT Break on Timeout are not available on 12bit or 16C5X devices

3.2.1.5 Break on Stack Over/Underflow

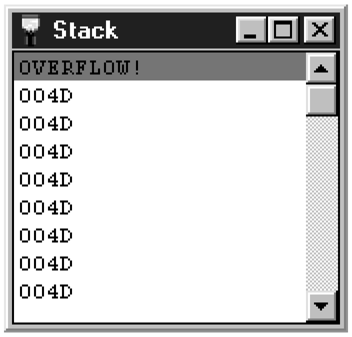
Setup => Project => Emulator

If enabled, this feature will halt program execution anytime the stack is "pushed" beyond its capacity or "popped" too many times.

The first line in the stack window keeps tra ck of the stack depth. When the stack is full, this line turns yellow. If an additional CALL is executed, this line turns RED, the program is halted and you receive the message "Stack OverFlow".

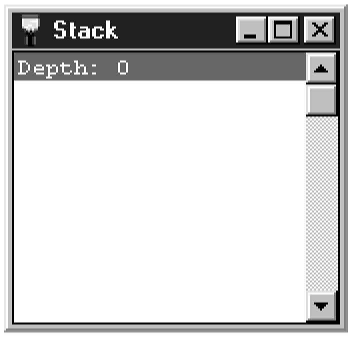


```
Depth:  8
004D
004D
004D
004D
004D
004D
004D
004D
```

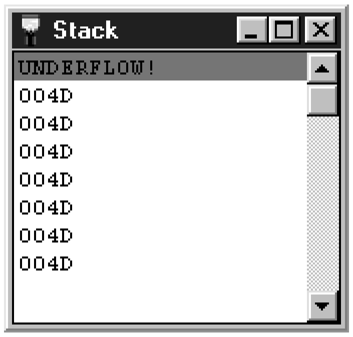


```
OVERFLOW!
004D
004D
004D
004D
004D
004D
004D
004D
```

If too many RETURNs are executed, the message will read "Stack UnderFlow", and the stack window may show new return addresses. These new addresses actually exist in the stack and are left-over from previous CALLs. When the stack is "empty" and a RET is executed, the stack pointer moves from the top of the stack to the bottom of the stack. So the underflow actually copies whatever value is located   at the bottom of the stack into the program counter; which could be garbage or a real address from a previous stack operation.



```
Depth:  0
```



```
UNDERFLOW!
004D
004D
004D
004D
004D
004D
004D
```

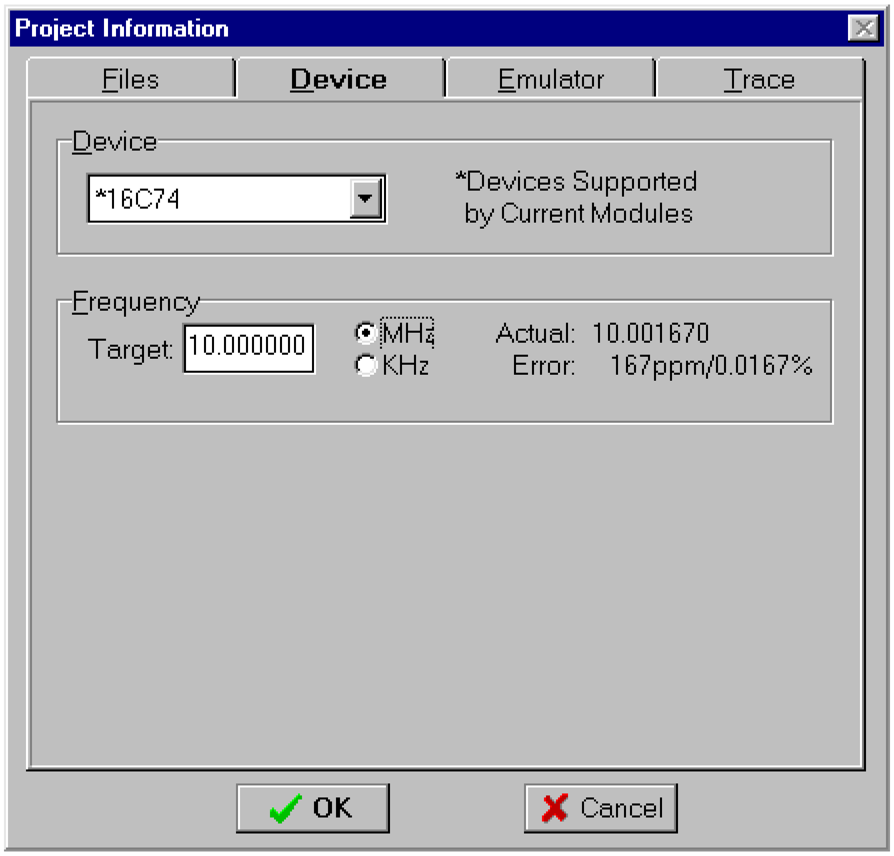
The items listed in the Stack Window beneath the Depth line are the RETURN ADDRESSES that have been pushed onto the stack. This will help you track down the cause of a stack overflow. However, if further analysis is required on a stack underflow, a Trace Buffer Module 116 is required (part #CVMT2). This will give you an execution history, making it possible to back-track the code flow.

Note:
Stack Viewing is only available with 14bit device emulation and Mathias firmware Version 2.1 or higher and Break on Stack Over/Underflow is not available on 12bit or 16C5X devices.

3.2.1.6 Device Settings

Setup => Project => Device

Next, select the Device tab. Here you can select the actual PIC device and oscillator frequency you wish to emulate.



Device

*16C74

*Devices Supported by Current Modules

Frequency

Target: 10.000000 [✓ MHz] KHz

Actual: 10.001670

Error: 167ppm/0.0167%

Device Type

The device pulldown lists all available device types supported by the Mathias hardware. The items marked with an asterisk (*) are the devices supported by your current Module combination. If you have changed your com port settings, and the TDE software has not communicated with the Mathias yet, you may not see any marked devices. In this case, just select the proper device that you wish to emulate and proceed with the rest of your settings.

Frequency

The Mathias hardware is capable of operating at any frequency from 32 KHz to 25 MHz, with certain limitations. Certain "Bondout" combinations may be limited to 20 or even 10 MHz.

Bondouts and Speed Limitations

Bondouts are specially manufactured forms of the PIC production silicon that make it possible to access internal control signals. These bondout chips are installed in the Mathias Family and Member modules, making it possible to accurately emulate the production PIC. Microchip Technology Incorporated produces these bondouts and specifies their speed ratings. This speed rating sets the maximum frequency at which the production PIC can be emulated.

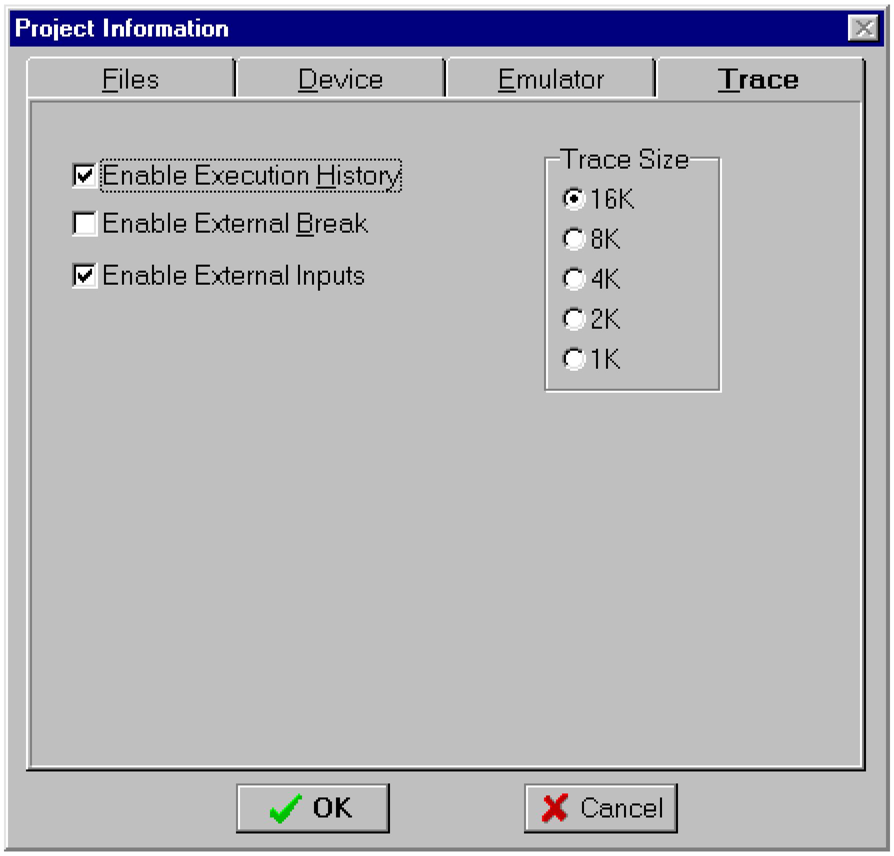
Currently all 12bit or 16C5x devices can be emulated at a full 20Mhz; the maximum production rating. However, depending on which 14bit Family Module you have installed, you may be limited to an emulation speed of 10Mhz on 16Cxx devices. Our original 14bit Family Module (part # CVMXXF) contained Microchip bondout part #16C02-ME/L, which is limited to 10 MHz operation.

If you require the ability to emulate a 20Mhz PIC, the newer 25Mhz PICs or would like the new feature of Data Breakpoints, we now offer an Advanced Family Module for 14bit devices: part #CVM03F.

3.2.1.7 Trace Buffer

Setup => Project => Trace

If you have a Trace Buffer Module (part #CVMT2), select the Trace tab.



☑ Enable Execution History
☐ Enable External Break
☑ Enable External Inputs

Trace Size
● 16K
○ 8K
○ 4K
○ 2K
○ 1K

Note: The functions shown here are only available if the trace buffer module is installed.

Execution History:

If this is enabled, TDE keeps a record of each instruction executed. This record is stored in memory on the optional trace buffer module.

For each instruction, the record shows the location in the trace buffer, the value of the program counter, the state of external trace inputs, and the original line of source code.

External Break:

If this is enabled, the emulator will halt execution if the external Break input goes low.

External Inputs:

There are eight external trace inputs, shown at the end of the hardware installation section.

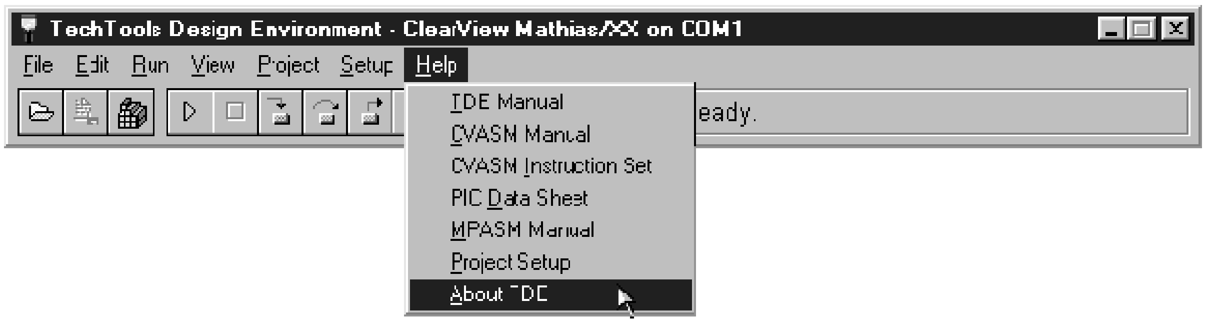
Trace Size:

To prevent extra delays when execution is halted, we have added a selectable Trace Size option.

Many of our customers requested the ability to view only the most recent trace information when a breakpoint was encountered, but wanted to keep the full trace ability for more "in depth" analysis when needed. As a result, you can now set the Trace Size as small as 1K or as large as 16K with several sizes in between.

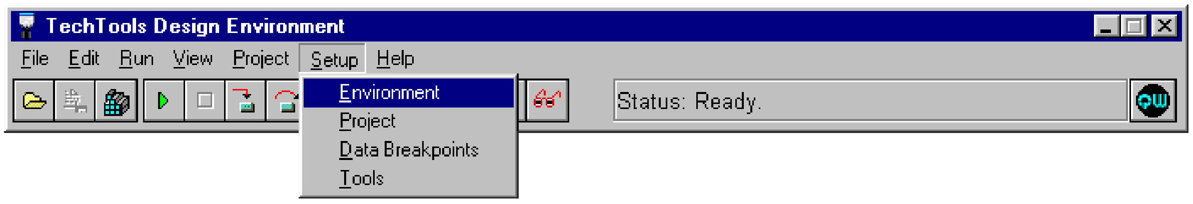
You can now click the OK button and TDE will save these settings.

The Trace Size option is only available with Mathias firmware Version 2.1 or higher. To determine the firmware version of your Mathias hardware, select the About option from the Help menu. When the About dialog is activated, TDE will retrieve the hardware version information from Mathias and display it at the bottom of the window.



3.2.1.8 Environment Preferences

Environment Preferences can be mod ified by selecting the Environment option from the Setup menu in TDE. This option allows you to set general features that affect all editing/debugging sessions.



Environment Option Groups

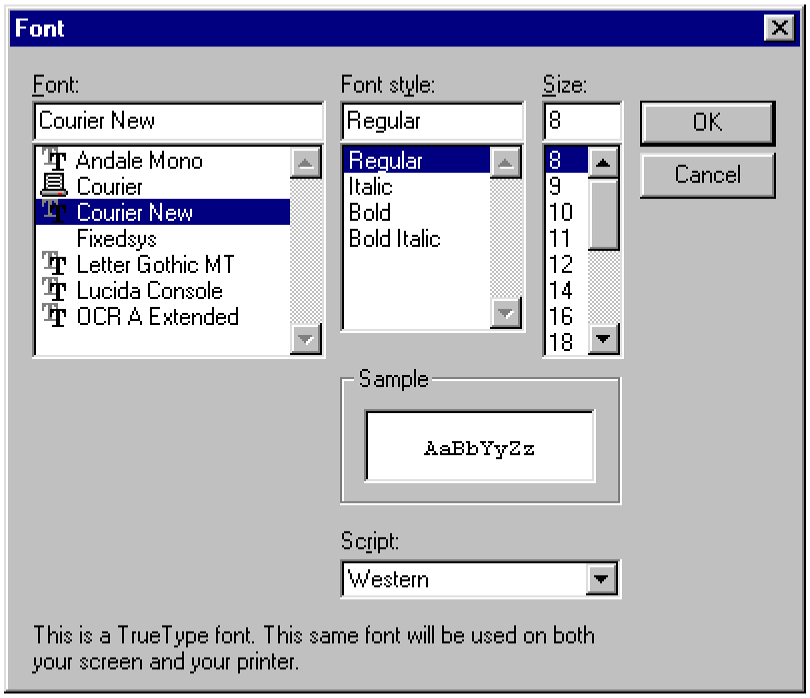
- Editor 63
- General 64
- Build 65
- Debug 65

Programmer_Preferences 66

3.2.1.8.1 Editor Preferences

Font Options

Clicking the Select button will open a Font options dialog. A list of available Fixed Width Fonts will be displayed on the left. You can select any of these fonts and other options to customize how your code is printed and displayed.



This is a TrueType font. This same font will be used on both your screen and your printer.

Tab Width

Choose the number of spaces to skip when the Tab key is pressed in the Editor window.

Auto Indent

When this option is enabled, a carriage-return will automatically space over the beginning of the next line to match the one above it.

Auto Backup

TDE will automatically create a backup copy of modified files and use the extension '.bak', before saving changes made from the editor window.

Tab Style

USE TAB - A TAB character is inserted in the editor when the Tab Key is pressed.

USE Spaces - Space characters are inserted in the editor when the Tab Key is pressed.

Smart Spaces - Space characters are inserted in the editor when the Tab Key is pressed. However, the Tab position is determined by th e text position in the previous line, providing "smart" tab positions that line up with existing text columns.

Current Line Position

This option chooses the positio ning of the current source line in relationship to the borders of the Debugging window. When single stepping, animating or halting program execution; TDE will use this setting to show the current execution line.

3.2.1.8.2 General preferences

Show Hints

This option toggles the extra 'fly-out' hints that appear when the mouse pauses over an item in TDE.

Retain Z Order

Retaining the Z order will force the active windows in TDE to reappear in the same 'front to back' order when TDE is opened, the program is reset or when a halt is encountered.

Randomize File Registers

TDE automatically sets all Special Registe r values to their default power-up or reset state. When the Randomize option is selected, TDE will randomize the General Purpose registers on a power-up or manual reset. This insures true PIC emulation and helps expose any hidden dependencies on previous register values.

Auto Load last Project

If this option is enabled, TDE will automatically load the project that was active the last time TDE was closed.

Beep on Errors

Whenever an error is encountered by TDE, the PC speaker will emit a short beep.

Beep on Stop

Whenever program execution is halted, the PC speaker will emit a short beep.

Animation Step Delay

During single-stepping and animating, it may be desirable to slow down the execution speed in order to view the changes taking place, without   having to halt the program. This option gives you a sliding scale to set the minimum delay from 100 msec to 999 msec. Note: Delays can be longer than this setting, depending on the speed of the computer running the TDE software. However, this option doe s set the minimum delay, preventing these modes from running faster; even if other, heavily updated windows are closed.

3.2.1.8.3 Build Preferences

Always Build All

When selected, ALL project files will be built whenever any of the Editor files have been updated. When NOT selected, only modified files will be re-built.

Verbose Build log

When selected, additional information will display in the build log during the build process, providing a detailed view of events.

Log Warnings

When selected, warning messages will be included in the Build Log.

Abort After

Build will abort if the number of warnings or error messages exceed the numbers specified here.

3.2.1.8.4 Debug Preferences

Run To MAIN on reset

When   Selected, Mathias will execute all code from the reset vector of the Emulated PICmicro MCU at Full speed until a "MAIN" function or Label is encountered. If NOT selected, Mathias will halt at the reset vector.

Halt ONLY on Source Lines

If Selected, Mathias will continue to run on a halt request until a source line reference is found for the code being executed. If NOT selected, Mathias will halt immediately when requeste d.

Step OUT of unknown code

When selected, any STEP operation will run full speed through compiled code that does not have a source line reference.

Step OVER calls to unknown code

When selected, any STEP operation will run full speed through calls that do not have a source line reference (effectively performing a STEP OVER when stepping).

Show File Register Mirrors

When selected, the General Purpose Register window (file registers), will display memory locations that are mirrored with other locations in ano ther bank. This also means that multiple locations can change when the data in a "mirrored" location changes . When NOT selected, mirrored locations will not be displayed.

Debugging with Stale Source Code

Ask Me What to Do - If the source code has been modified, TDE will ask whether to rebuild the source before continuing or continue without rebuilding (stale debugging information).

Build and restart debugging - If the source code has been modified, TDE will rebuild the source and reset the microcontroller before continuing. (fresh debugging information).

Continue Debugging with stale data - If the source code has been modified, TDE will continue without rebuilding or resetting the microcontroller (stale debugging information).

3.2.1.8.5 Programming Preferences

Enable Program Button

This option Enables or Disables the QuickWriter programming button on TDE's main Application bar, next to the Status area.

Make before Launch

When Selected, the current project will be rebuilt before passing the resulting HEX file to the QuickWriter application.

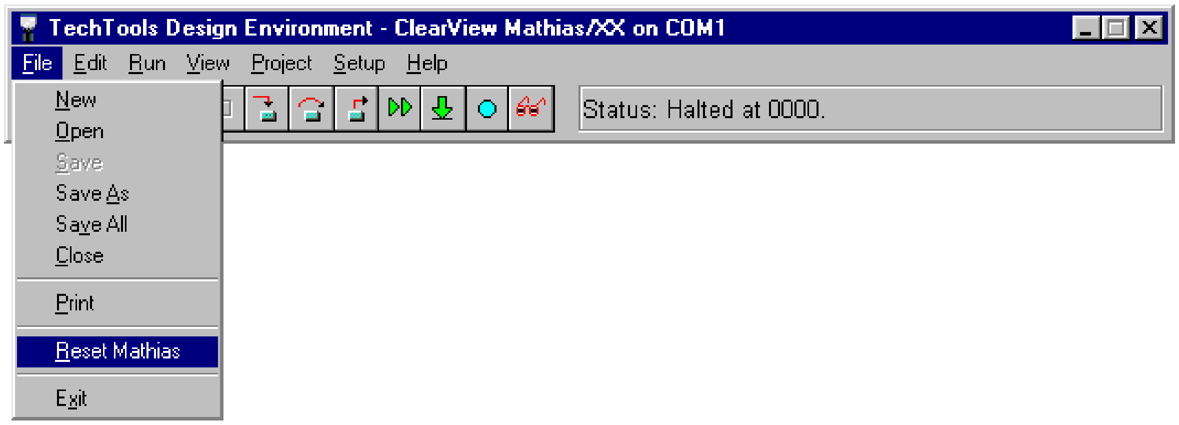
Auto Program and Close

When launched, the QuickWriter software will automat ically program and close. If an error occurred while programming, the application will stay open so that all error information can be seen.

3.2.1.9 Resetting the Emulator

Sometimes, it may be necessary to reset the emulator hardware. For instanc e, there may be a communications error between the PC and the emulator, which will require a reset.

To reset the emulator, select Reset Mathias from the File menu.

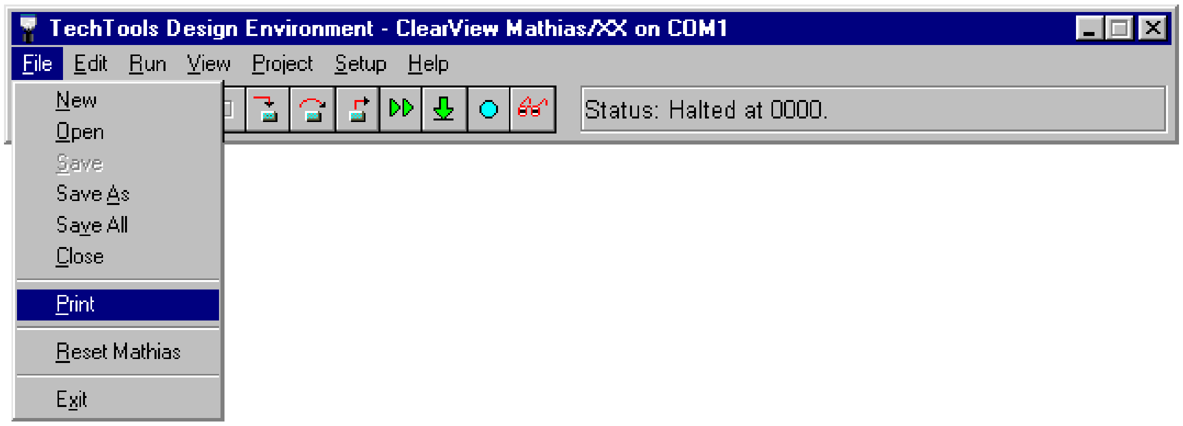

TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
New
Open
Save
Save As
Save All
Close
Print
Reset Mathias
Exit
Status: Halted at 0000.

Please note that this is much different from the Reset 76 option found in the Run menu. The reset described here resets all of the emulato r hardware, much like turning the emulator off and on. The reset option in the Run menu just resets the emulated PIC.
If you want to simulate a PIC reset cycle, the proper reset to use is the one in the Run menu.

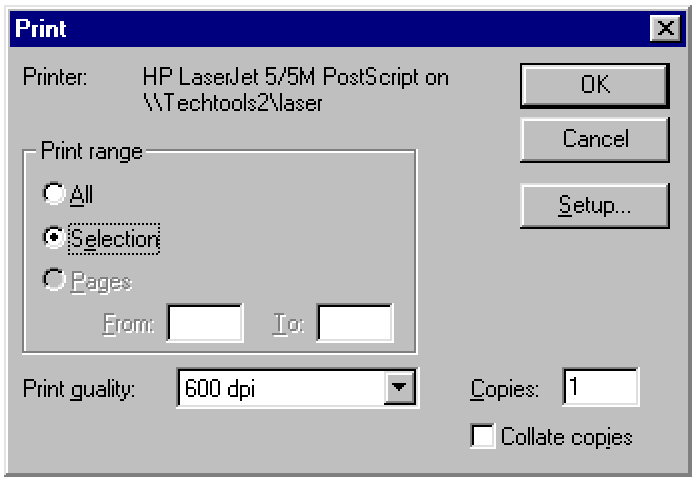
3.2.1.10 Printing

At times, it may be helpful to print a copy of your work. TDE allows you to print from two windows: the editor and the trace buffer.

To print from either window, just click on the window and then select Print from the File menu.


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
New
Open
Save
Save As
Save All
Close
Print
Reset Mathias
Exit
Status: Halted at 0000.

TDE will display the Print dialog box:



Printer: HP LaserJet 5/5M PostScript on \\Techtools2\laser

Print range: All, [✓ Selection], Pages From: To:

Print quality: 600 dpi

Copies: 1

Collate copies

From this dialog box, you can choose what you want to print, how man y copies to print, etc. If everything is correct, click OK to start printing.

If you need to change settings that aren't shown, click the Setup button. The Print Setup dialog box will appear, which allows you to select different printers, paper sizes, etc.

3.2.2 Compiling and Executing Code

The following sections explain how to assemble or compile source code, and then execute that code in the emulator.

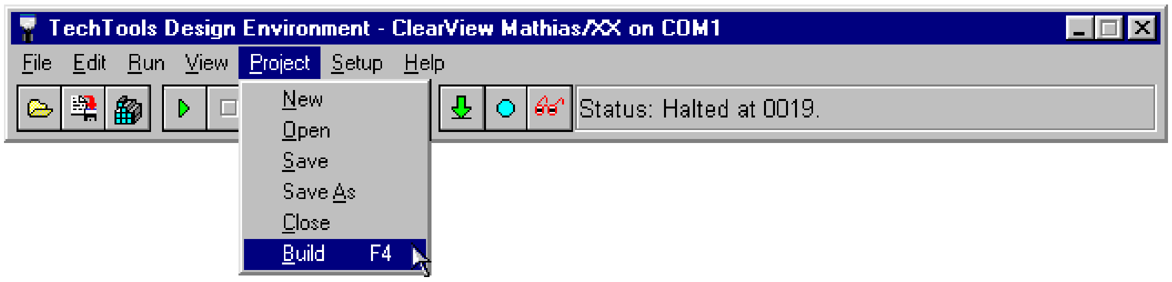
Building the project 68
Executing code at full speed 70
Halting execution 71
Single-step , step-over and step-out execution 71
Animation-mode and "Animate Over" execution 74
Showing the execution point 75
Resetting the emulated PIC 76
Setting the program counter (execution point 77)
Using breakpoints 78

3.2.2.1 Building the Project

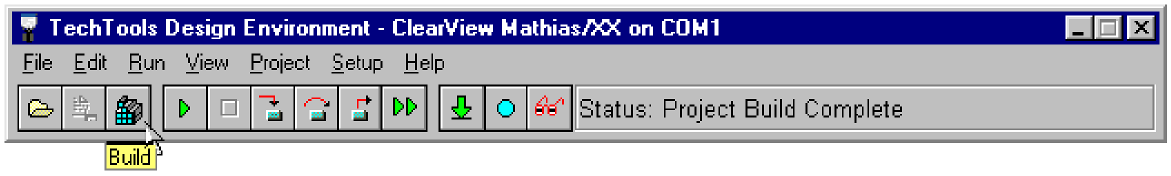
The process of assembling or compiling source code is called "building." When you tell TDE to build the project, it calls upon the selected tools to assemble/compile the each node in the project.*

To build the project, select Build from the Project menu, or click the corresponding button, or press F4.

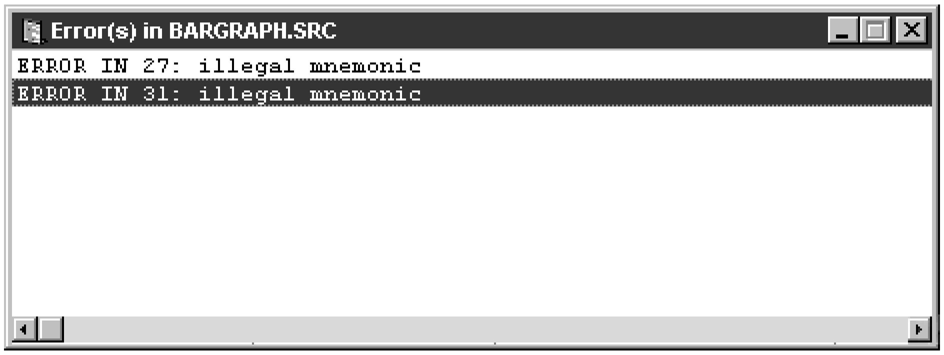
F4


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
New
Open
Save
Save As
Close
Build F4
Status: Halted at 0019.

If no errors are encountered, then the resulting hex code can be run in the emulator. Note the message "Project Build Complete" shown below:

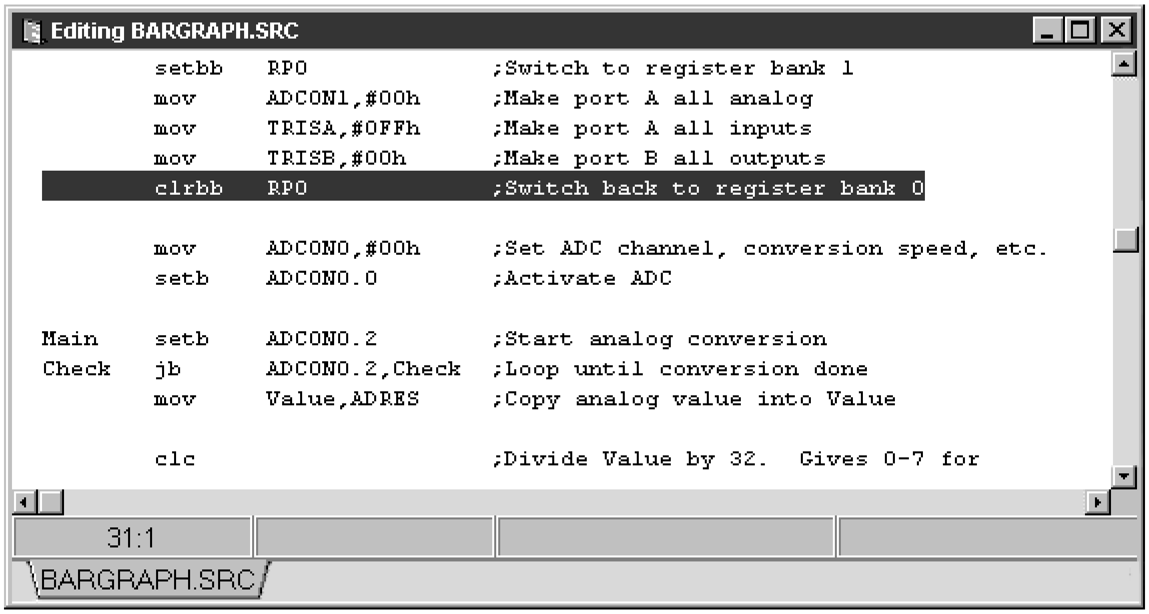

TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Status: Project Build Complete
Build

If errors are encountered, then TDE will display an error   list:



```
ERROR IN 27: illegal mnemonic
ERROR IN 31: illegal mnemonic
```

To see a corresponding error in the source code, just click on the error in the error listing; the original source code line will be highlighted:



Editing BARGRAPH.SRC

```
          setbb   RP0           ;Switch to register bank 1
          mov     ADCON1,#00h   ;Make port A all analog
          mov     TRISA,#0FFh   ;Make port A all inputs
          mov     TRISB,#00h    ;Make port B all outputs
          clrbb   RP0           ;Switch back to register bank 0

          mov     ADCON0,#00h   ;Set ADC channel, conversion speed, etc.
          setb    ADCON0.0      ;Activate ADC

Main      setb    ADCON0.2      ;Start analog conversion
Check     jb      ADCON0.2,Check ;Loop until conversion done
          mov     Value,ADRES   ;Copy analog value into Value

          clc                   ;Divide Value by 32.  Gives 0-7 for
```

31:1

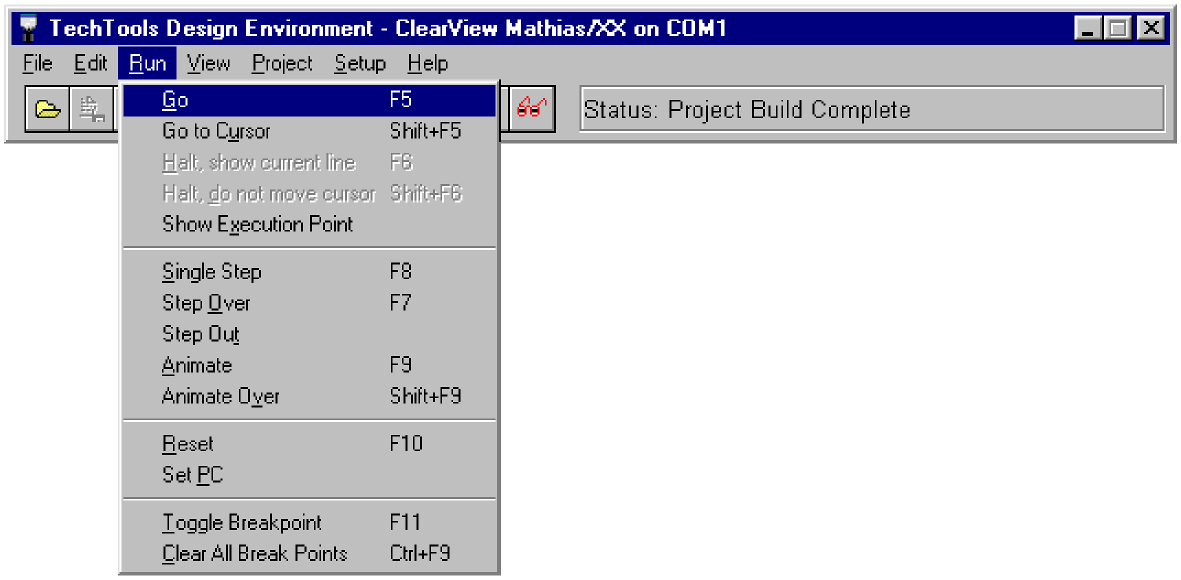
BARGRAPH.SRC

* The assembler/compiler is selected by using the Project 46 option in the Setup menu.

3.2.2.2 Executing Code at Full Speed

To execute code at full speed in the emulator, select Go from the Run menu, or click the corresponding button, or press F5.

F5



TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Status: Project Build Complete
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

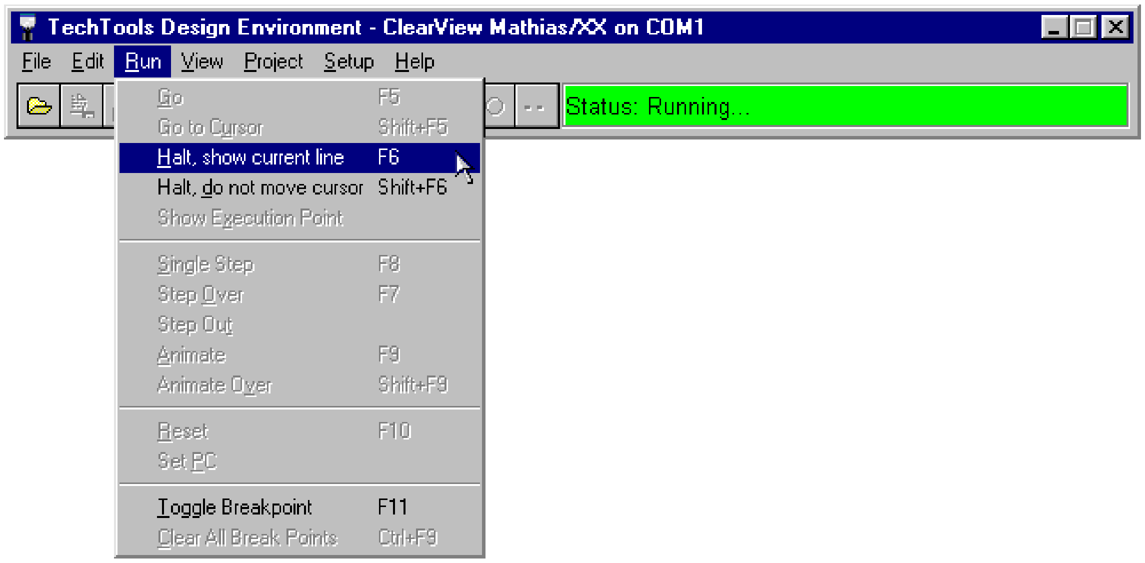
Code will be executed at the speed chosen when the project was started (for more information, see the Device 59 tab of the Project option of the Setup menu). Execution speeds vary from 30 kHz to 25 MHz or up to the maximum ratings of the modules.

During full-speed execution, very little occurs with TDE. After downloading the code to the emulator, TDE simply waits for a breakpoint or for user input. Because full-speed execution occurs too quickly for the PC to keep up, TDE does not update any of its windows until execution is stopped (by a breakpoint or by you).

3.2.2.3 Halting Execution

To halt code execution, select Halt from the Run menu, or click the corresponding button, or press F6.

F6

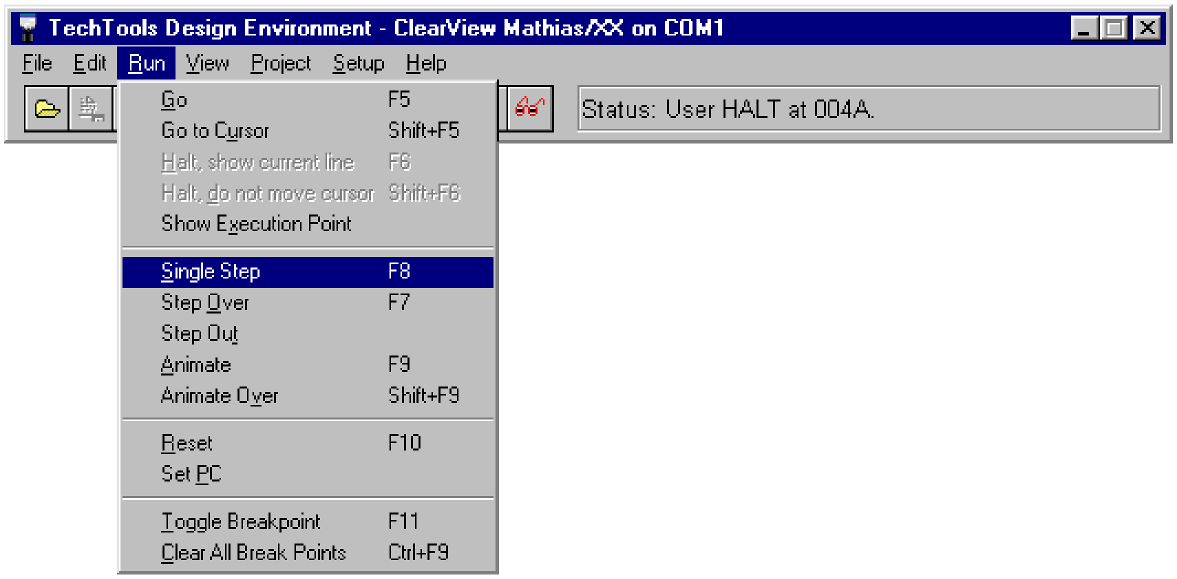


Execution will be halted and TDE will update information in any open windows.

3.2.2.4 Single-Step Execution

To execute one line or instruction, select Single Step from the Run menu, or click the corresponding button, or press F8.

F8

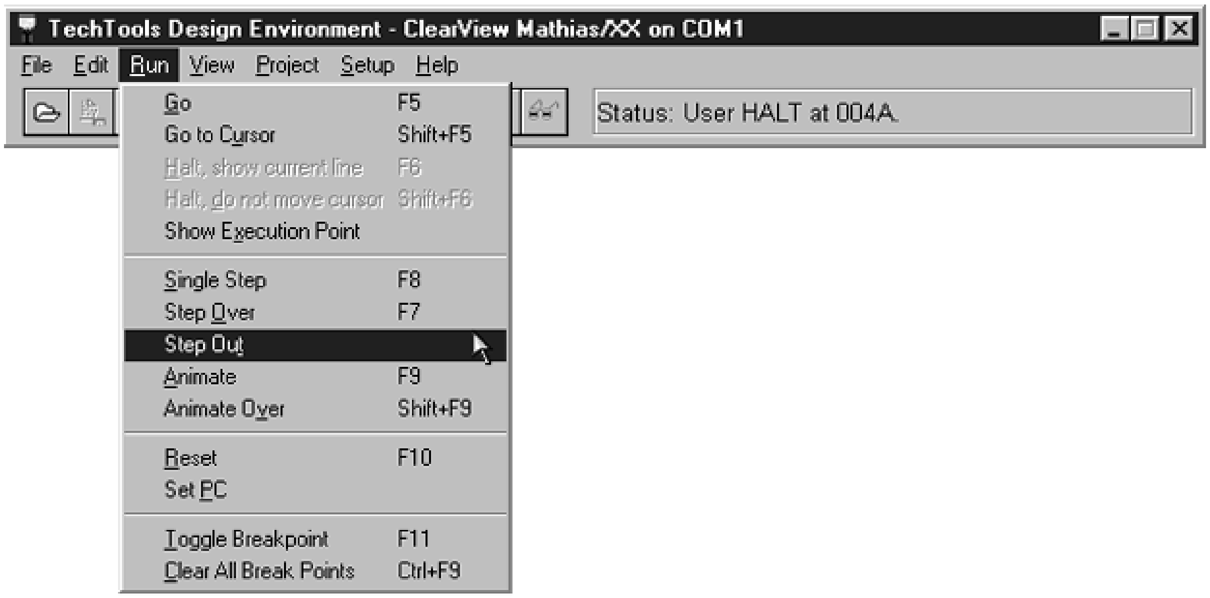

TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9
Status: User HALT at 004A.

If the editor window is the active window, then the emulator will execute one line of source code. If the code window is the active window, then the emulator will execute one instruction of code. When using the TechTools assembler or any C compiler, one line of source code may be multiple instructions.

Single-stepping is very useful when you suspect that a part of your code contains an error, but you're not sure which instruction is the cause.

3.2.2.5 Step OUT Execution

To execute a step-out command, select Step Out from the Run menu, or click the corresponding button.

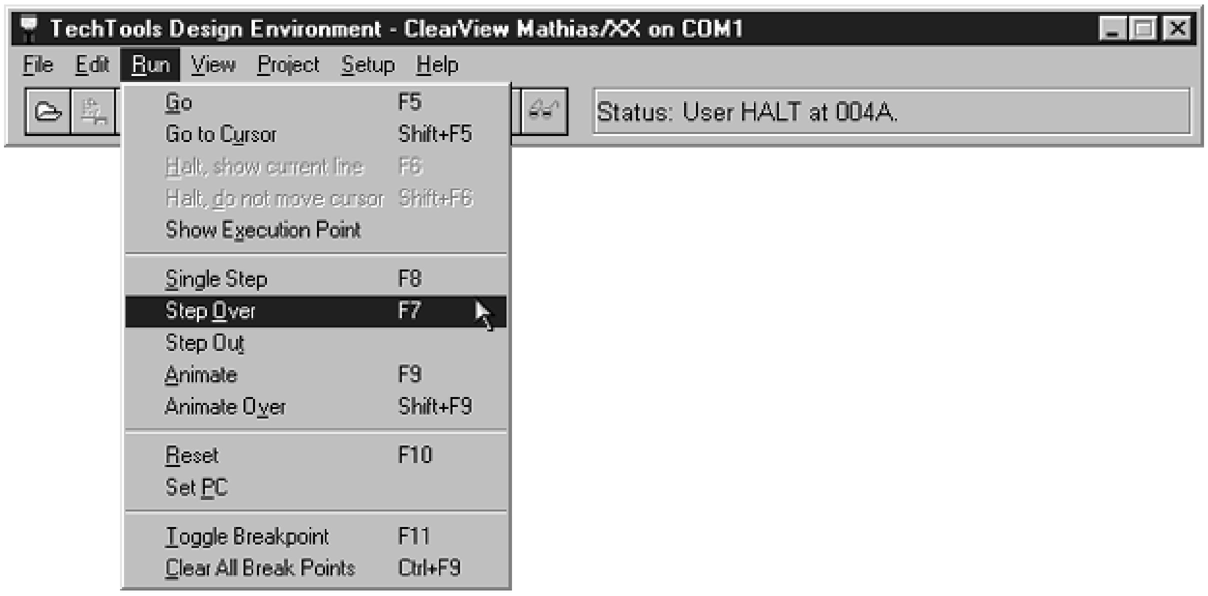

TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Status: User HALT at 004A.
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

Step-out execution will "Step Out" of a subroutine if the current execution point has already progressed into one. The remaining portion of the routine will execute at full speed until a return is encountered, then execution will be halted.

3.2.2.6 Step Over Execution

To execute in step-over mode, select Step Over from the Run menu, or click the corresponding button, or press F7. The emulator will execute one instruction, and then TDE will update any open windows.

F7

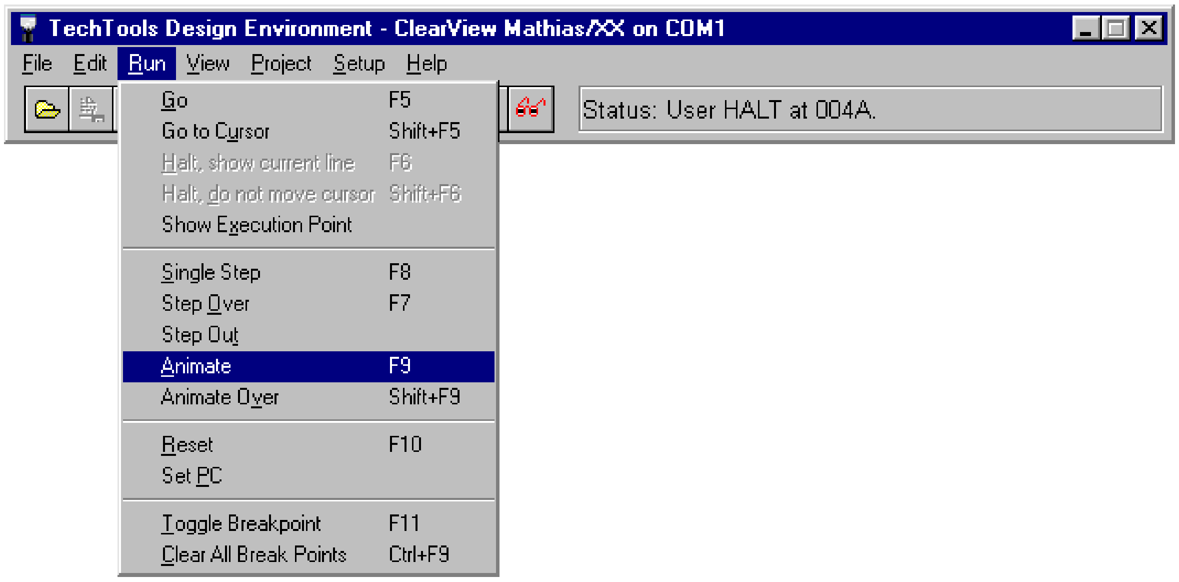

TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Status: User HALT at 004A.
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

Step-over execution is similar to single-step 71 execution. In this mode, the emulator executes one instruction at a time, just like single-step mode. However, step-over mode jumps over calls to subroutines (subroutines are still executed, but not shown on the screen). This is useful when you're debugging a particular routine, but you don't want to study each subroutine in the same detail.

3.2.2.7 Animation-Mode Execution

To execute in animation mode, select Animate from the Run menu, or click the corresponding button, or press F9.

F9


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

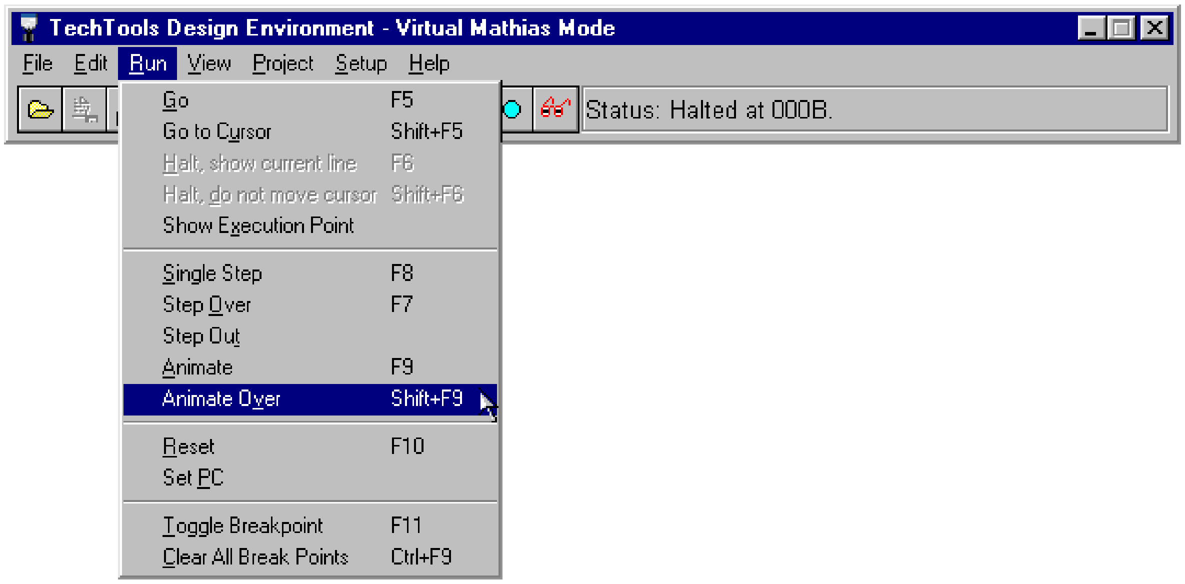
Status: User HALT at 004A.

Animation mode is a repeating version of single-stepping. In this mode, the emulator executes one instruction, then waits for TDE to update its windows, and then executes the next instruction. Execution speed varies with the speed your PC and the baud rate selected for communication with the emulator. On a typical 486-based PC, the emulator will execute 1-2 instructions per second. If Animate runs too fast, you can set the maximum rate by choosing the Environment 62 option from the Setup menu.

3.2.2.8 Animate Over Execution

To execute in animation mode, select Animate Over from the Run menu, or press "Shift" and click the Animate button, or press "Shift" and F9.

Shift + F9

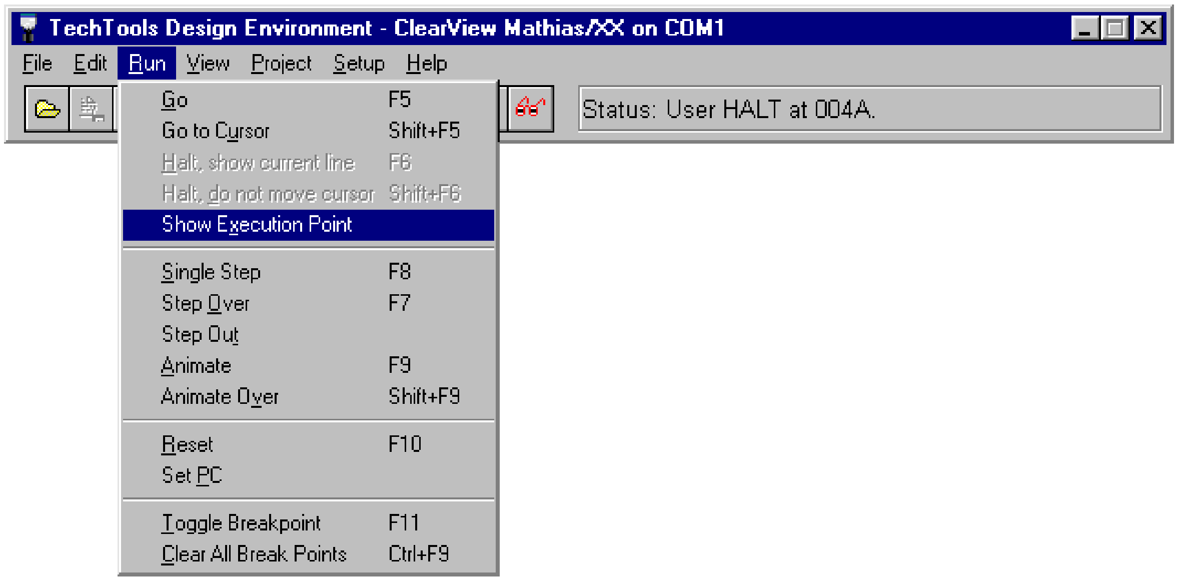


Animate Over is a modified Animate function. When Animate Over is selected, the emulator executes one instruction, waits for TDE to update its windows then executes the next instruction. However if the instruction is a "CALL", the emulator will execute at full speed until a "RET" is processed, then resume its single instruction stepping.

Since Animate Over relies on Stack Viewing capability, this mode of execution is not available when emulating a 16C5x device.

3.2.2.9 Showing the Execution Point

To show the execution point, select Show Execution Point from the Run menu. The execution point will be indicated by a green highlighted line in the text editor window.


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

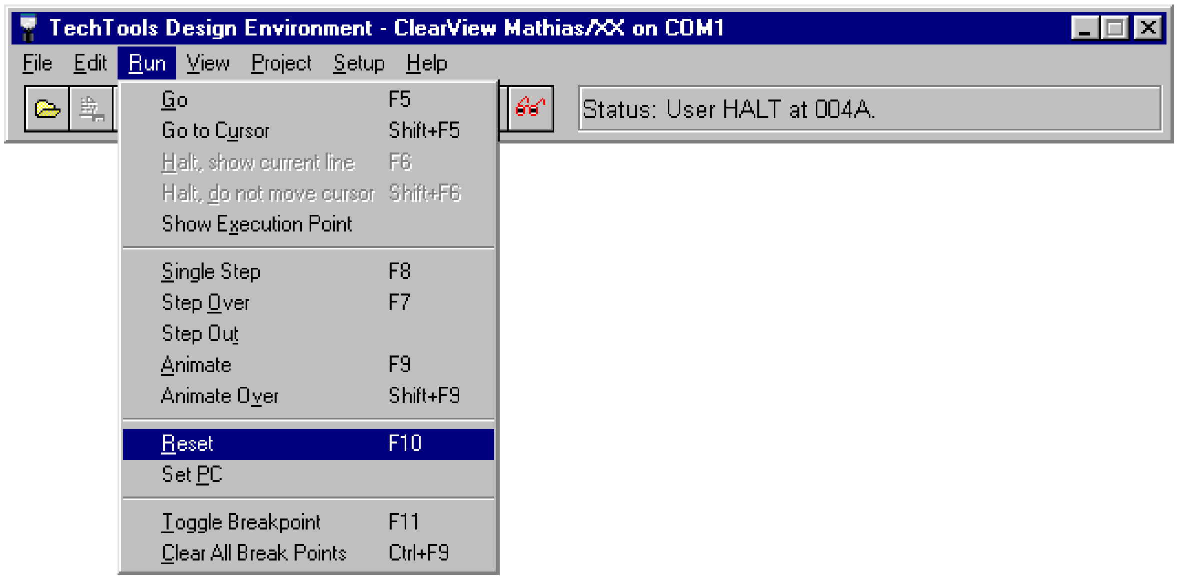
Status: User HALT at 004A.

At first, this option may seem a bit strange. The execution line is always indicated by a green highlighted line in the text editor window, so you might ask why you would need to specifically ask to see the execution line. Well, if execution has been stopped, and you've spent the las t few minutes scrolling around in a long program, you may have forgotten where the execution point is; in such a situation, you can ask TDE to quickly take you back.

3.2.2.10 Resetting the Emulated PIC

To reset the PIC, select Reset from the Run menu, or click the corresponding button, or press F10.

F10

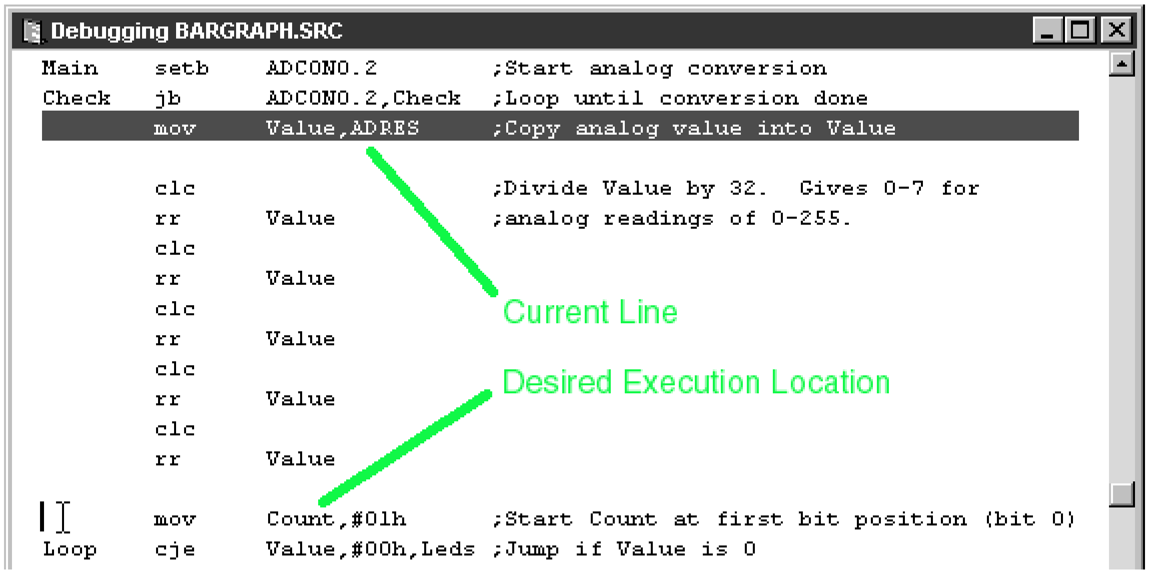

TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

Status: User HALT at 004A.

This resets the emulated PIC, as though a hardware reset had occurred. The program counter is reset, and all internal registers are set to their power-on state (or they are randomized, if you selected that option in the TDE Environment settings 62). This is different from the Reset ClearView 66 option in the File menu, which resets the emulator hardware itself.

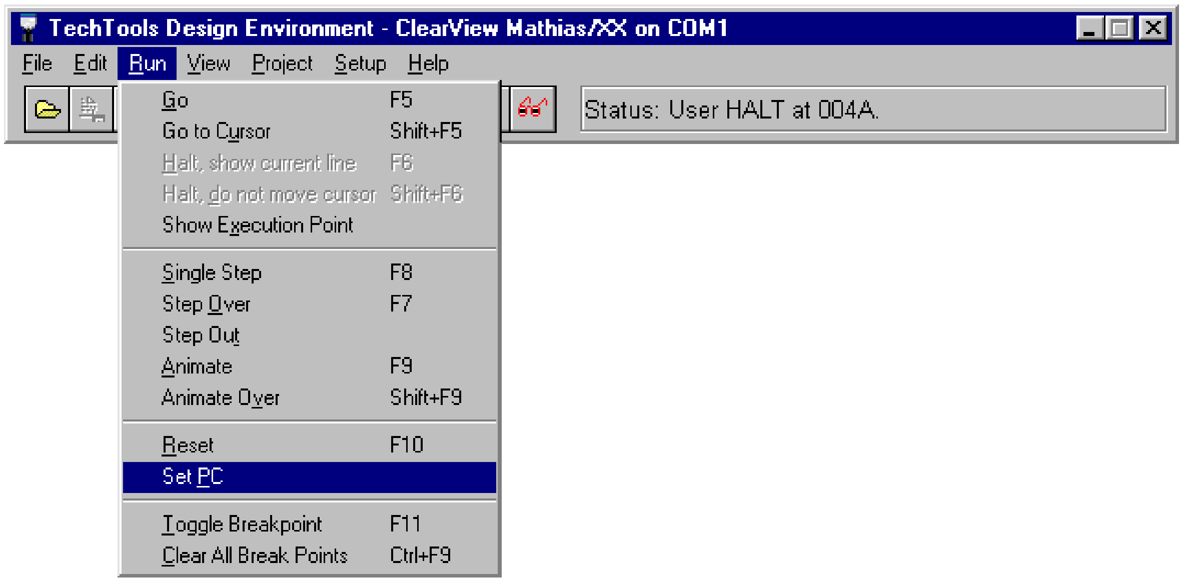
3.2.2.11 Setting the Program Counter

Sometimes, you may wish to set the program counter (the execution point) to an arbitrary location in your program. To do this, click on the line where you wish to set the program counter:

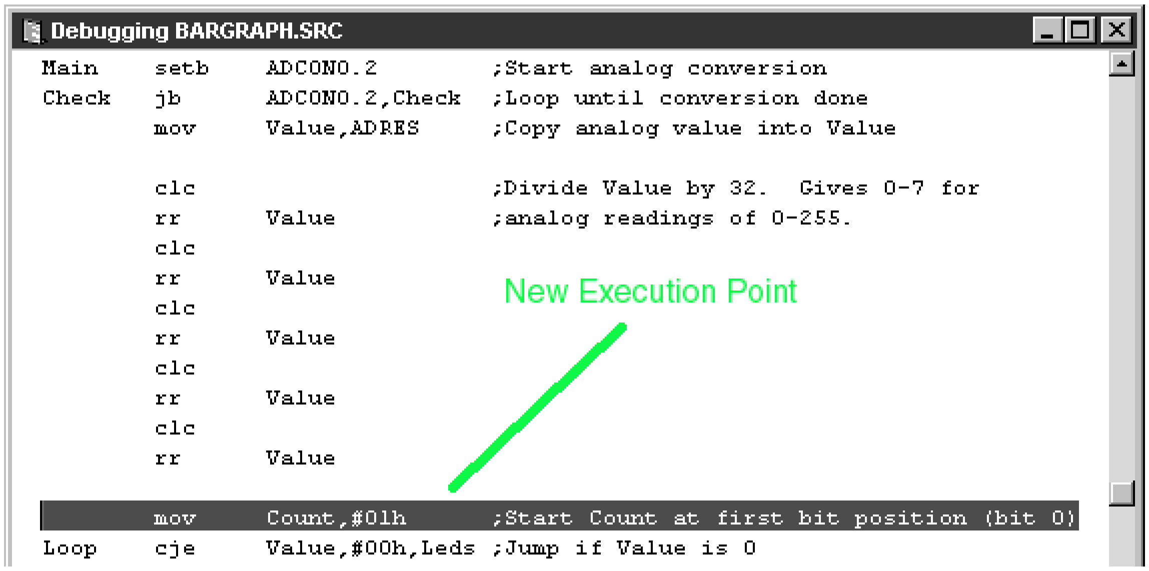


```
Main    setb    ADCON0.2        ;Start analog conversion
Check   jb      ADCON0.2,Check  ;Loop until conversion done
        mov     Value,ADRES     ;Copy analog value into Value

        clc                     ;Divide Value by 32.  Gives 0-7 for
        rr      Value           ;analog readings of 0-255.
        clc
        rr      Value
        clc
        rr      Value
        clc
        rr      Value
        clc
        rr      Value

        mov     Count,#01h      ;Start Count at first bit position (bit 0)
Loop    cje     Value,#00h,Leds ;Jump if Value is 0
```

After clicking on the new line, select Set PC from the Run menu.


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Status: User HALT at 004A.
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

The highlighted line will move to the new line, indicating the new program counter value:

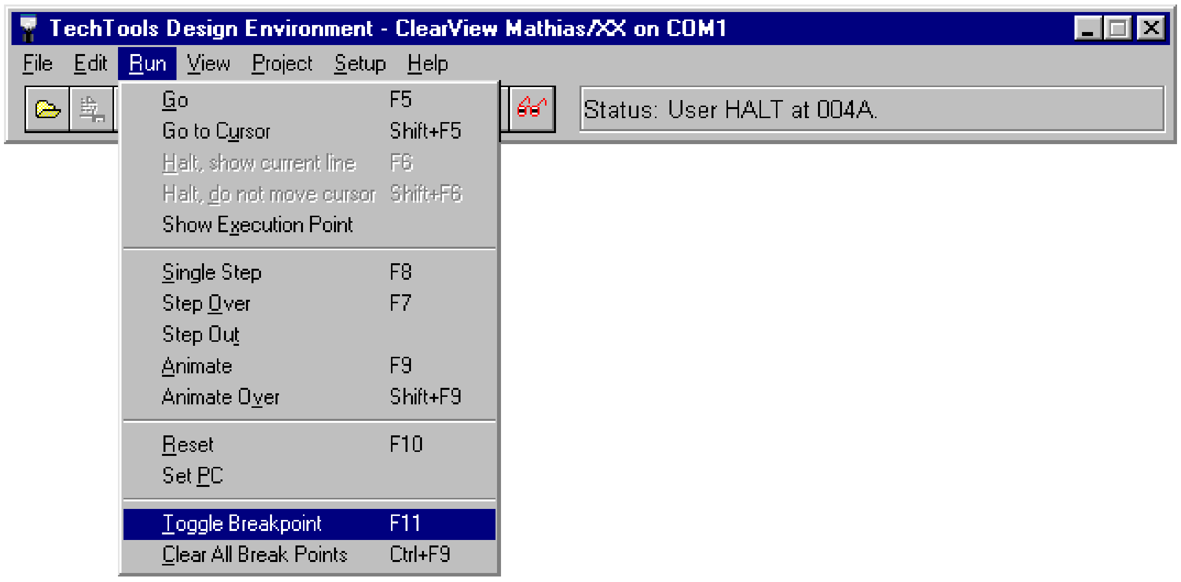


```
Main     setb    ADCON0.2         ;Start analog conversion
Check    jb      ADCON0.2,Check   ;Loop until conversion done
         mov     Value,ADRES      ;Copy analog value into Value

         clc                      ;Divide Value by 32.  Gives 0-7 for
         rr      Value            ;analog readings of 0-255.
         clc
         rr      Value
         clc
         rr      Value
         clc
         rr      Value
         clc
         rr      Value

         mov     Count,#01h       ;Start Count at first bit position (bit 0)
Loop     cje     Value,#00h,Leds  ;Jump if Value is 0
```

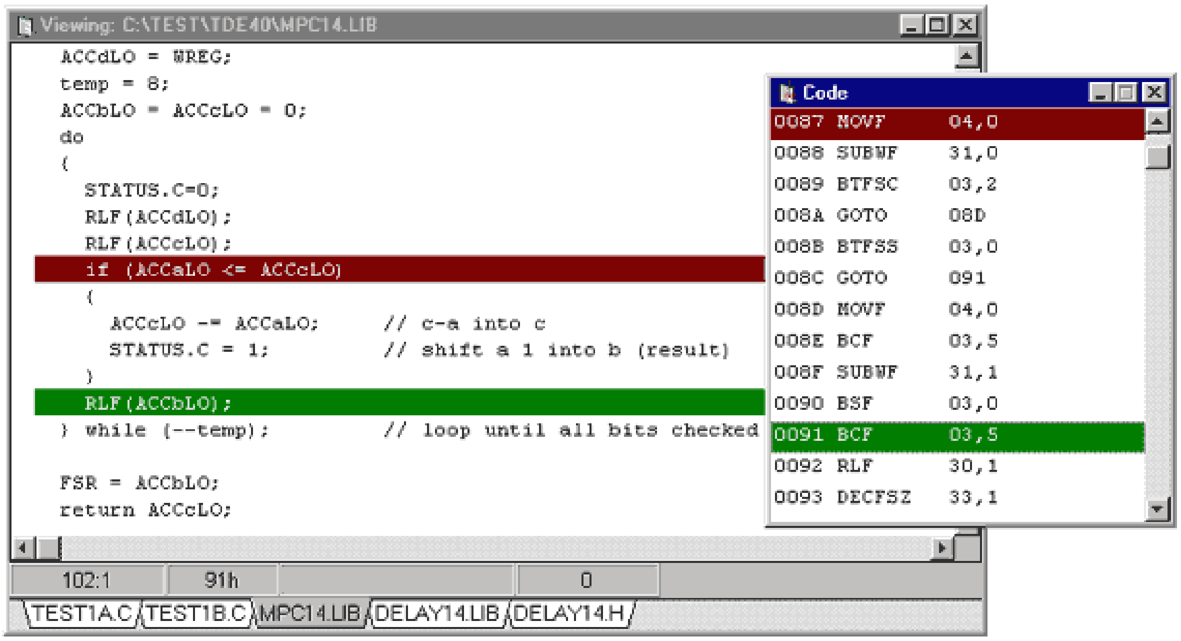
3.2.2.12 Using Breakpoints

One very useful debugging method is the use of breakpoints. Break points are flags that you set on one or more lines in your program. During execution, the emulator stops on breakpoints, allowing you to see the status of registers, I/O pins, etc.

To set a breakpoint, just double-click in the left margin of the desired line. The line will be highlighted in red, indicating the presence of a breakpoint. To remove the breakpoint, just double-click in the left margin of the same line. If you prefer, you can also use the keyboard and menu options shown below.


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Status: User HALT at 004A.
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

Note that you can set breakpoints in the text editor window and in the code window. Any change in one window will be reflected in the other window, as well.


Viewing: C:\TEST\TDE40\MPC14.LIB
Code
102:1
91h
0
TEST1A.C TEST1B.C MPC14.LIB DELAY14.LIB DELAY14.H

```
ACCdLO = WREG;
temp = 8;
ACCbLO = ACCcLO = 0;
do
{
  STATUS.C=0;
  RLF(ACCdLO);
  RLF(ACCcLO);
  if (ACCaLO <= ACCcLO)
  {
    ACCcLO -= ACCaLO;      // c-a into c
    STATUS.C = 1;          // shift a 1 into b (result)
  }
  RLF(ACCbLO);
} while (--temp);          // loop until all bits checked

FSR = ACCbLO;
return ACCcLO;
```

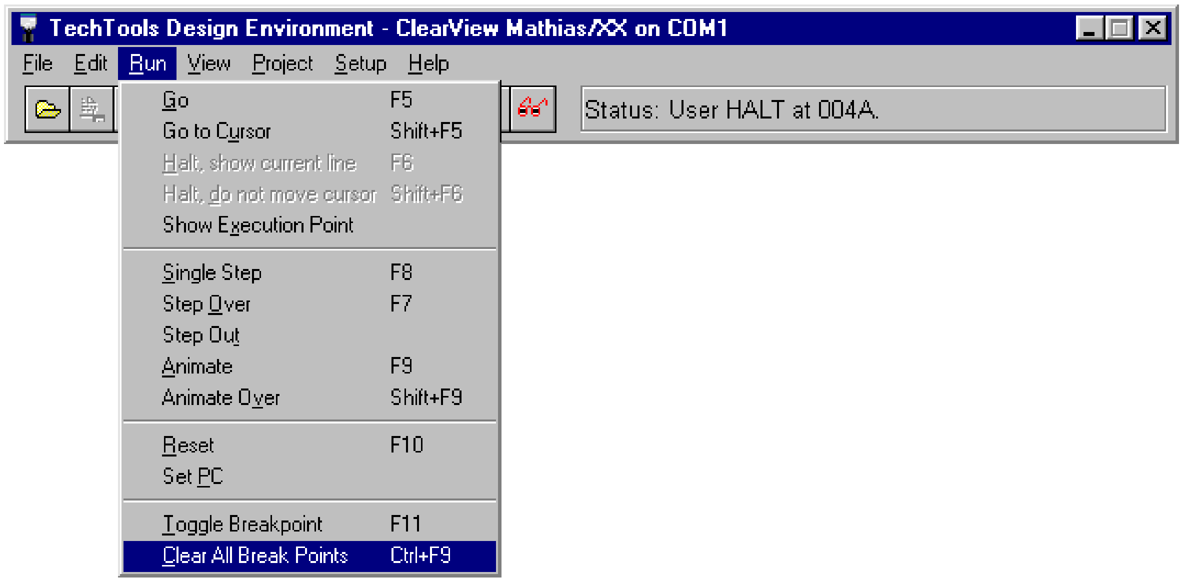
```
0087 MOVF    04,0
0088 SUBWF   31,0
0089 BTFSC   03,2
008A GOTO    08D
008B BTFSS   03,0
008C GOTO    091
008D MOVF    04,0
008E BCF     03,5
008F SUBWF   31,1
0090 BSF     03,0
0091 BCF     03,5
0092 RLF     30,1
0093 DECFSZ  33,1
```

Red highlighting shows the location of a breakpoint.

Clearing All Breakpoints

If you want to clear all breakpoints, there’s an easier way than clearing each one individually. Instead, you can use the Clear All Breakpoints function.

To clear all breakpoints, select Clear All Breakpoints from the Run menu, or press Ctrl-F9.


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Go F5
Go to Cursor Shift+F5
Halt, show current line F6
Halt, do not move cursor Shift+F6
Show Execution Point
Single Step F8
Step Over F7
Step Out
Animate F9
Animate Over Shift+F9
Reset F10
Set PC
Toggle Breakpoint F11
Clear All Break Points Ctrl+F9

Status: User HALT at 004A.

3.2.3 Using the Various TDE Windows

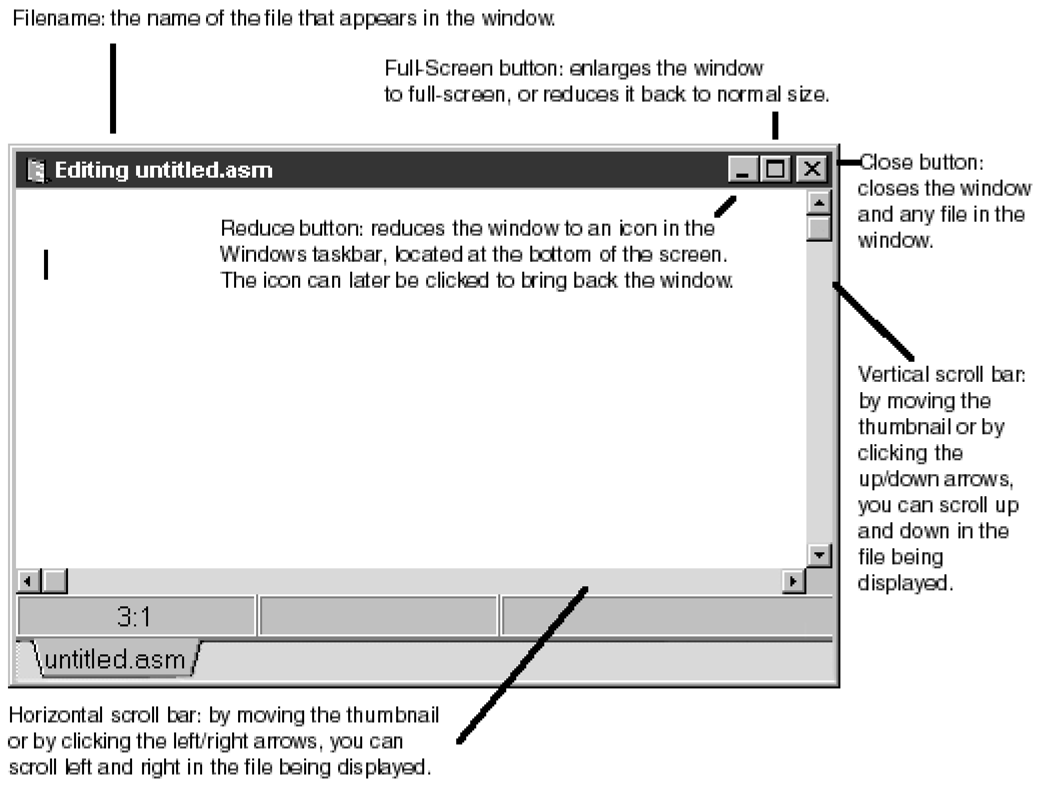
The following section begins with the "basics" of using TDE's windows, then covers the specifics of how to implement particular features of various windows.

Window basics 80
Text editor window 82
Build Log window 93
Code w indow 95
Symbols window 96
File registers window 105
Watch list window 108
Trace buffer window 116

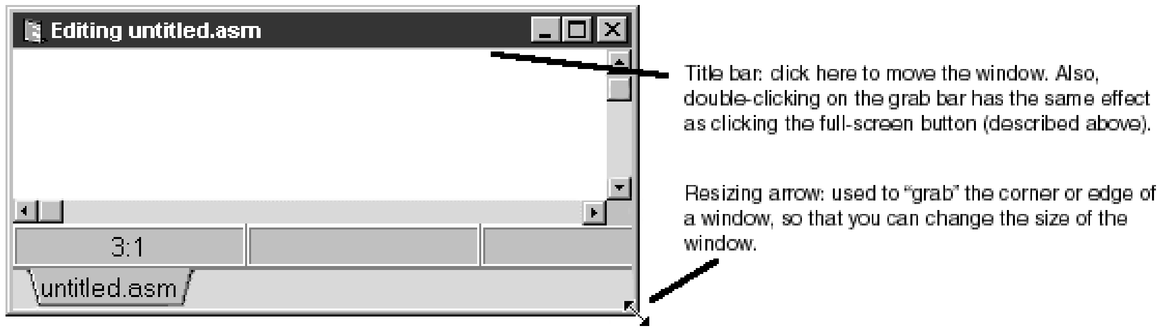
3.2.3.1 Window Basics

Before we examine any particular window, let’s review the basics concernin g all windows: things like scroll bars, close boxes, etc. If you already understand such things, you may want to skip to the Editor Window section.

The window below is like most windows in TDE; it happens to be the editor window, but for thi s discussion, it could be almost any window.


Filename: the name of the file that appears in the window.
Full-Screen button: enlarges the window to full-screen, or reduces it back to normal size.
Close button: closes the window and any file in the window.
Editing untitled.asm
Reduce button: reduces the window to an icon in the Windows taskbar, located at the bottom of the screen. The icon can later be clicked to bring back the window.
Vertical scroll bar: by moving the thumbnail or by clicking the up/down arrows, you can scroll up and down in the file being displayed.
3:1
untitled.asm
Horizontal scroll bar: by moving the thumbnail or by clicking the left/right arrows, you can scroll left and right in the file being displayed.

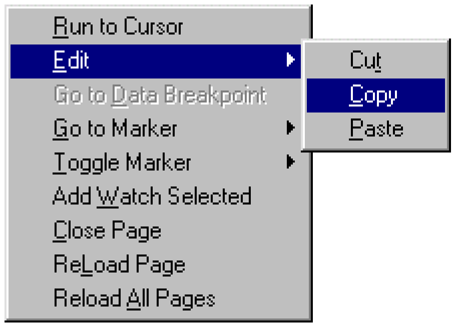
A couple of things that may not be obvious are how to move a window and how to change its size . To move a window, click and hold the mouse button on the window’s "Title bar" while moving the mouse. To change its size, click and hold the mouse button on one of the window’s edges or corners while moving the mouse.


Editing untitled.asm
Title bar: click here to move the window. Also, double-clicking on the grab bar has the same effect as clicking the full-screen button (described above).
Resizing arrow: used to “grab” the corner or edge of a window, so that you can change the size of the window.
3:1
untitled.asm

Pop-up menu:

Most TDE functions have one or two ways to access them. Some of the Window functions have a third way: by pop-up menu. To see the pop-up menu, press the right mouse button while the cursor is in the window.

Once the pop-up menu is visible, use the mouse (with the usual left button) to make your selection. The image below shows an example of the Editor pop-up menu:



3.2.3.2 Editor Window

The editor window is the central w indow in any project; it is used to enter and edit source code, to watch execution, and to set breakpoints. The editor acts like a simple word processor, complete with functions to cut/copy/paste, search/replace, mark routines, etc.

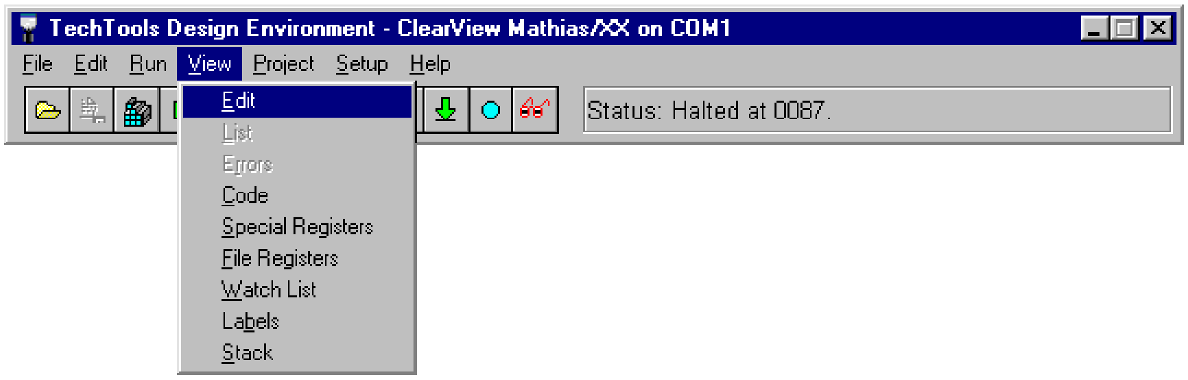
If you've read many of the preceding sections, you've already seen and worked with the editor window. So far, however, you haven't seen a detailed description of how the editor works. This section covers the editor in more detail.

Enter source code 85
Set and clear breakpoints 85
Cut, copy, and paste 85
Undo and redo 88
Search and replace 89
Place markers 91

There are several ways to see the editor window:

Open an existing or new project, OR open an existing or new file.

Then:



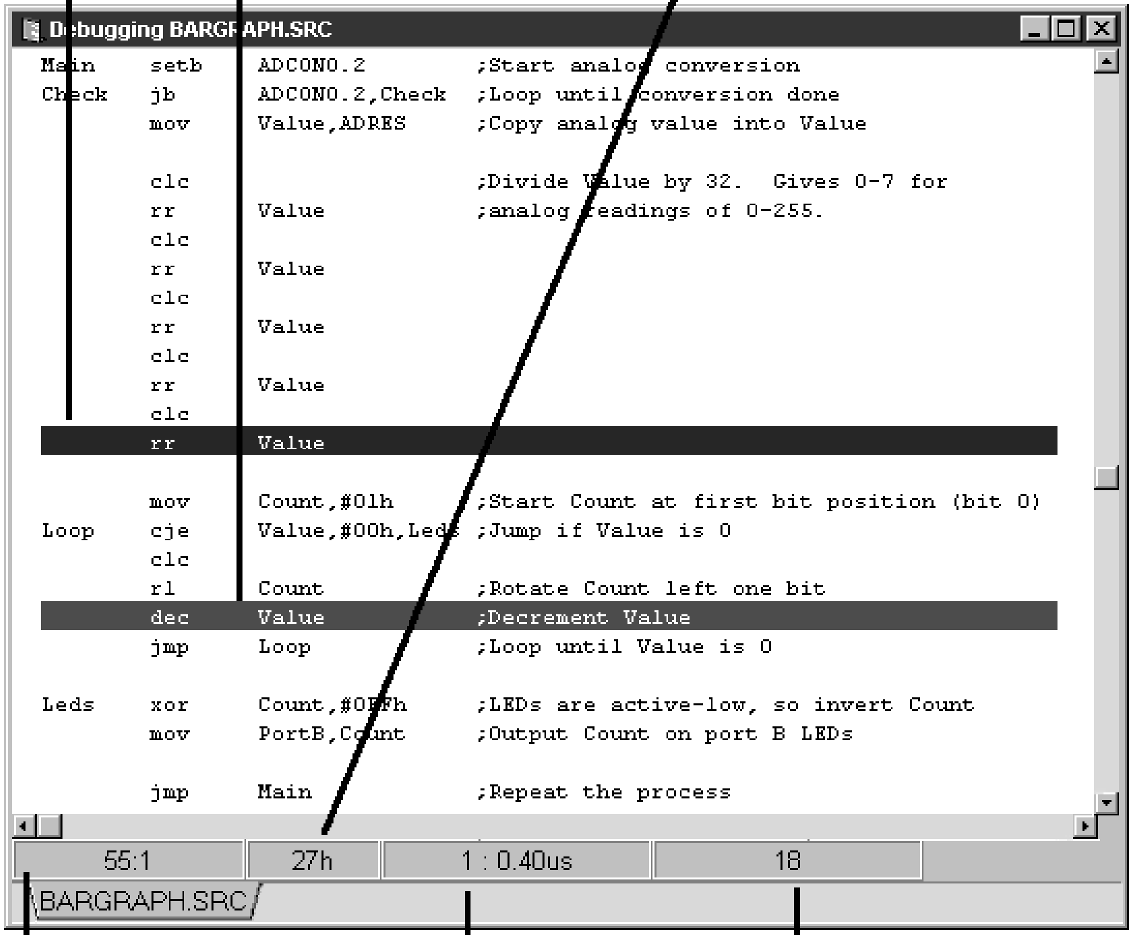
Select Edit from the View menu OR,
Select Open from the Files tab 51 of the Setup->Project option, OR...
Double-click on a File-node from the Files tab 51 of the Setup->Project option.

The open editor window looks like this:

Breakpoint (red): shows the location of a breakpoint. Double-click to set or clear a breakpoint.

Execution point (green): shows the instruction being executed.
If execution is stopped, then this line will be executed when execution is resumed.

Execution address: the address at the execution point.



```
Main    setb    ADCON0.2        ;Start analog conversion
Check   jb      ADCON0.2,Check  ;Loop until conversion done
        mov     Value,ADRES     ;Copy analog value into Value

        clc                     ;Divide Value by 32.  Gives 0-7 for
        rr      Value           ;analog readings of 0-255.
        clc
        rr      Value
        clc
        rr      Value
        clc
        rr      Value
        clc
        rr      Value

        mov     Count,#01h      ;Start Count at first bit position (bit 0)
Loop    cje     Value,#00h,Leds ;Jump if Value is 0
        clc
        rl      Count           ;Rotate Count left one bit
        dec     Value           ;Decrement Value
        jmp     Loop            ;Loop until Value is 0

Leds    xor     Count,#0FFh     ;LEDs are active-low, so invert Count
        mov     PortB,Count     ;Output Count on port B LEDs

        jmp     Main            ;Repeat the process
```

Row/column: shows the row and column numbers for the correct location of the cursor. The cursor is difficult to see here, because it is positioned at the beginning of the highlighted line.

Cycles and time: shows the number of processor }cycles and execution time for the last-executed instruction (in this case, the RL instruction just above the execution point). Cleared with each new instruction executed.

Total cycles: shows the total number of processor cycles used. Cleared by clicking on the value shown, or by loading a new project.

NOTE:
"Cycles and Time" and "Total cycles" are only available if the Trace Buffer 60 or Timing Module is installed.

3.2.3.2.1 Enter source code

The editor works much like a word processor. If you already have source code on disk, use New Project and ADD the existing file to the new project. Then select Open, for editing.

If you need to create a new source code file:
1. Open or Create a project,
2. Select ADD from the Files 51 tab of the Project Dialog

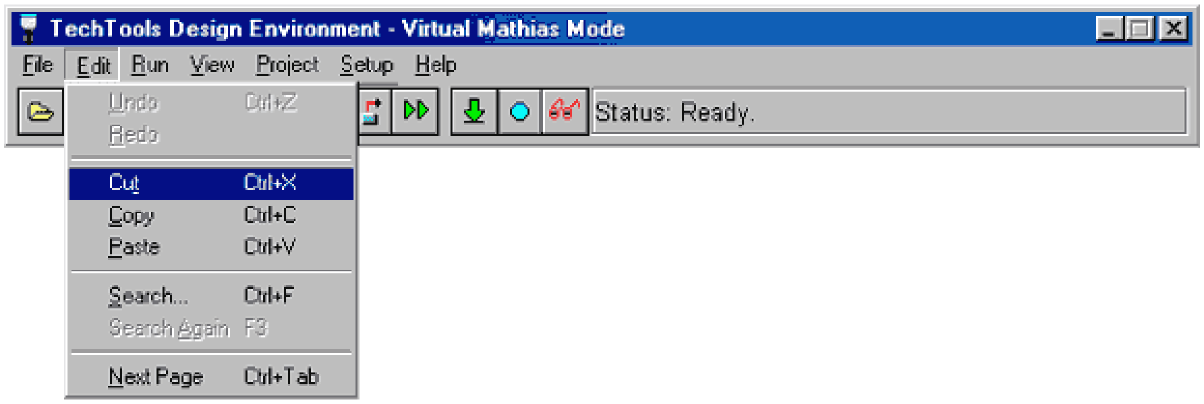
When prompted for the file name, enter the desired name and TDE will create the new file for you.

3.2.3.2.2 Set and clear breakpoints

To set or clear a Code breakpoint, just double-click in the left margin of the desired line. The line will be highlighted in red, indicating the presence of a breakpoint. If a breakpoint already exists on the line, then it will be cleared. Note that if the execution point (green) is present on the same line as a breakpoint (red), the resulting color is blue (green + red = blue). For more information on breakpoints, see the breakpoints 78 discussion at the end of the previous section.
Data Breakpoints (DBP) are only available if the Advanced xx Family Module is installed. For details on configuring Data Breakpoints 118, see the Data Breakpoints Section at the end of this Chapter.

3.2.3.2.3 Cut, copy, and paste

These functions are fairly standard on most editors, and work in the same manner. The editor maintains a "clipboard," which is a special buffer used for editing. When you cut or copy text, the text is stored in the buffer. Later, you can paste the buffer text into another part of your source code.



Cut:

Highlight the desired text, and then select Cut from the Edit menu, or select Cut from the pop-up menu, or press Ctrl-X. The highlighted text will be removed and stored in the buffer.

Copy:

Highlight the desired text, and then select Copy from the Edit menu, or select Copy from the pop-up menu, or press Ctrl-C. A copy of the highlighted text will be stored in the buffer (the text will not be removed from its current location, though).

Paste:

Place the cursor where you'd like to insert the text from the buffer (text that was previously cut or copied), and then select Paste from the Edit menu, or select Paste from the pop-up menu, or press Ctrl-V.

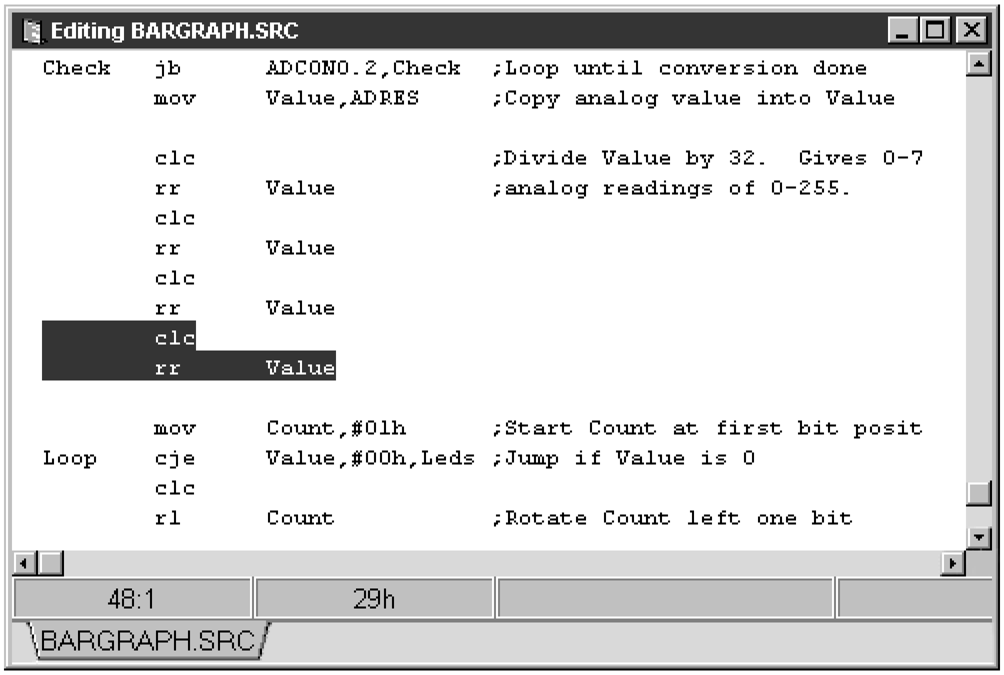
The following example shows a simple copy-and-paste operation:

Step 1: highlight text to copy

Place the mouse at the beginning of the text, and then click and hold the mouse button while moving the mouse to the end of the text.

Step 2: copy text to clipboard memory

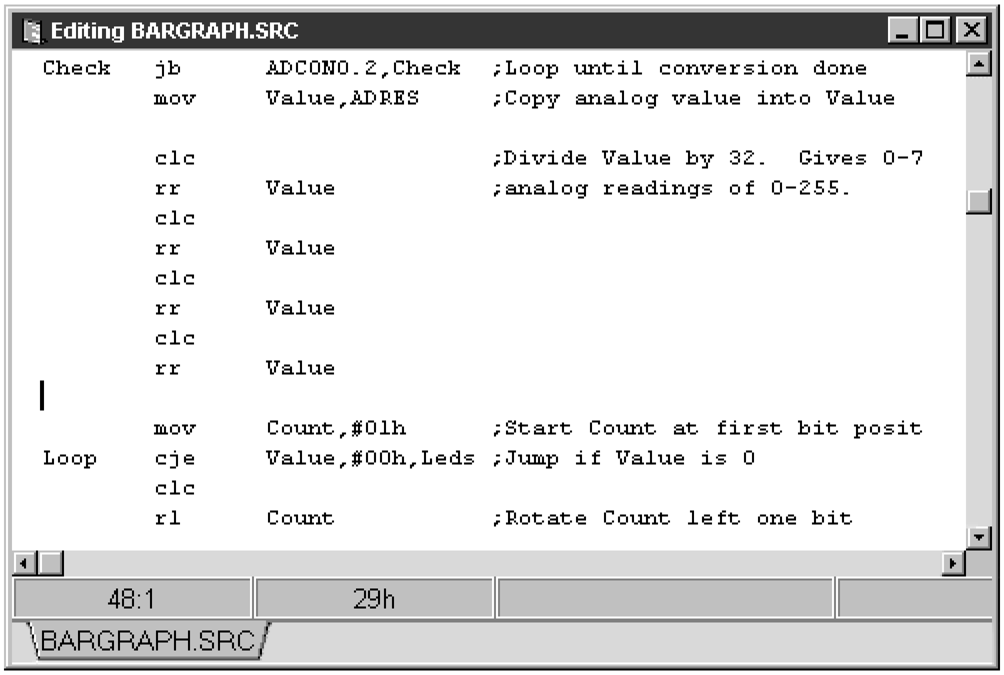
Select Copy from the Edit menu.



```
Check   jb      ADCON0.2,Check  ;Loop until conversion done
        mov     Value,ADRES     ;Copy analog value into Value

        clc                     ;Divide Value by 32.  Gives 0-7
        rr      Value           ;analog readings of 0-255.
        clc
        rr      Value
        clc
        rr      Value
        clc
        rr      Value

        mov     Count,#01h      ;Start Count at first bit posit
Loop    cje     Value,#00h,Leds ;Jump if Value is 0
        clc
        rl      Count           ;Rotate Count left one bit
```

Step 3: place cursor at new location

Move the mouse to the new location, and then click the mouse button. The cursor will appear at the new location.



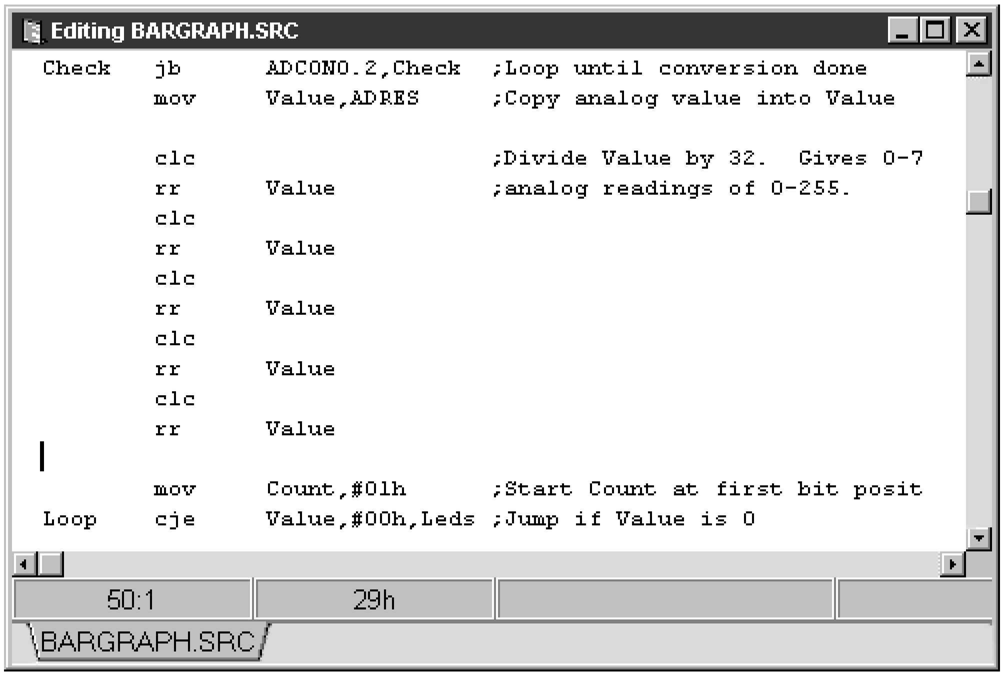
```
Check   jb      ADCON0.2,Check    ;Loop until conversion done
        mov     Value,ADRES       ;Copy analog value into Value

        clc                       ;Divide Value by 32.  Gives 0-7
        rr      Value             ;analog readings of 0-255.
        clc
        rr      Value
        clc
        rr      Value
        clc
        rr      Value

        mov     Count,#01h        ;Start Count at first bit posit
Loop    cje     Value,#00h,Leds   ;Jump if Value is 0
        clc
        rl      Count             ;Rotate Count left one bit
```

Step 4: insert the copied text

Select Paste from the Edit menu.

The copied text will appear where the cursor was placed in step 3.

Notice the new pair of CLC and RR instructions.



Editing BARGRAPH.SRC

```
Check    jb      ADCON0.2,Check   ;Loop until conversion done
         mov     Value,ADRES      ;Copy analog value into Value

         clc                      ;Divide Value by 32.  Gives 0-7
         rr      Value            ;analog readings of 0-255.
         clc
         rr      Value
         clc
         rr      Value
         clc
         rr      Value
         clc
         rr      Value

         mov     Count,#01h       ;Start Count at first bit posit
Loop     cje     Value,#00h,Leds ;Jump if Value is 0
```

50:1 | 29h

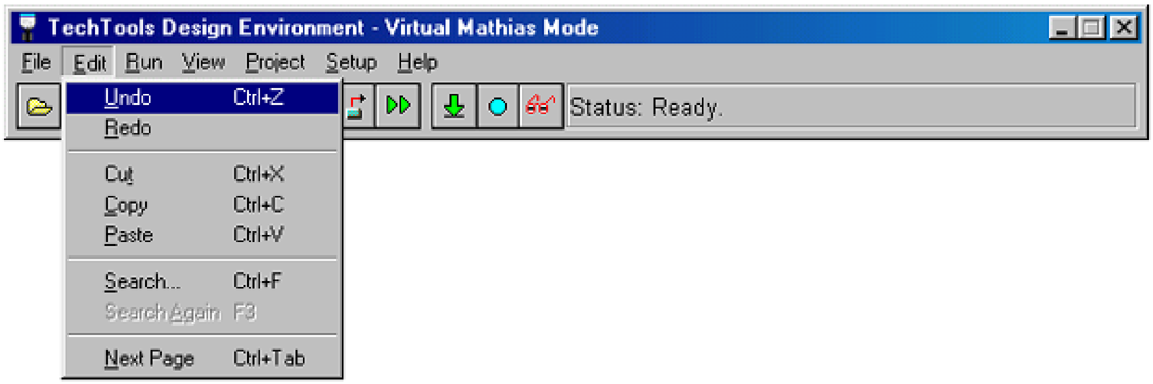
BARGRAPH.SRC

3.2.3.2.4 Undo and redo

Like cut, copy, and paste 85, these functions are fairly standard on most editors, and work in the same manner. The editor maintains a record of changes you make, so that changes can be reversed (undone) if you decide to do so.

A change is anything that was recently typed or deleted, but the return key separates one change from the next. Also, the return key itself is a change.

So, if you type "abc" on one line, and then you type "xyz" on the next line, there are three changes: 1) abc, 2) return key, and 3) xyz.



Undo:

Select Undo fro m the Edit menu, or press Ctrl-Z. The editor will undo the last change made in the editor.

Redo:

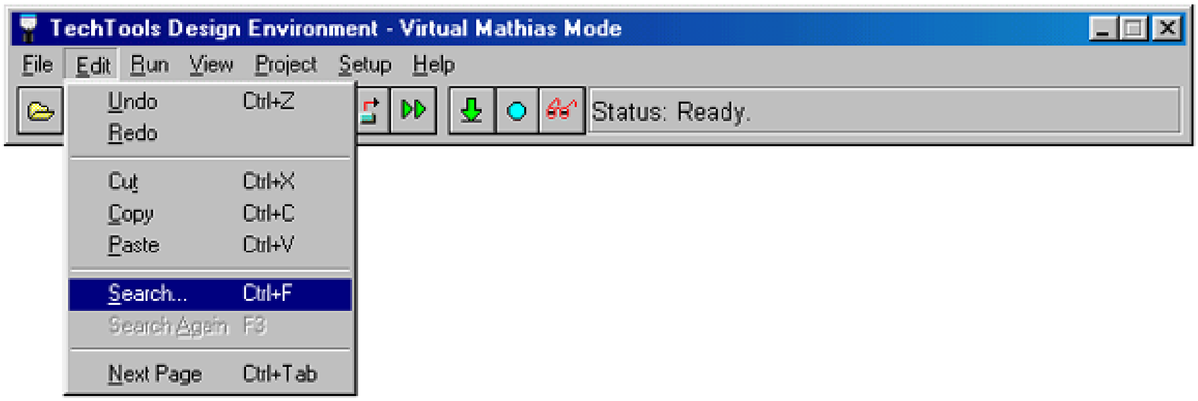
Select Redo from the Edit menu. The editor will redo (replace) the last "undone" item.

The editor maintains an 8K buffer for storing changes to the file, so a lot of things can be undone and redone. However, changes are stored sequentially, so it may take some time to reach a change that happened a long time ago. For instance, if you delete one line and then type ten new lines, and then decide that you want the first line restored, you will have to undo the last eleven changes (which will erase the last ten lines).

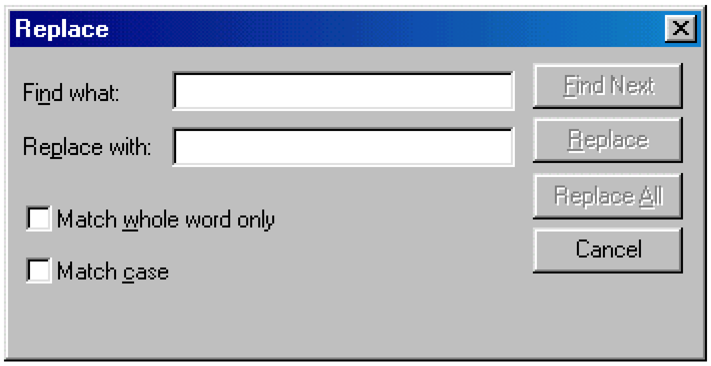
3.2.3.2.5 Search and replace

This function allows you to easily find text in the editor, and to replace the text with something new (if you want to). You can also perform global search and replace, in which the editor replaces every occurrence of the text.

The search function only searches text after the cursor location, so be sure to place the cursor a ccordingly. If you want to search the entire editor, you must place the cursor at the beginning of the file before starting the search.



To start a search, select Search from the Edit menu, or press Ctrl-F; after doing so, you'll see the following window:



The search window has the following controls:

Find what:

This is the text to search for.

Replace with:

This is the text which will replace the text found in the search. If you just want to find something without replacing it, you can leave this field blank.

Match whole word:

If this box is checked, then the search function will only find occurrences of the exact text to be found. If this box is not checked, then the search function may find words that simply contain the search text (for instance, if searching for "and," the function will find "band").

Match case:

If this box is checked, then the search function will only find occurrences with the same case. If this box is not checked, then the search function will ignore case (for instance, if searching for "value," the function will find "Value").

Find Next:

Click this button to search for the next occurrence of the search text. If a match is found, then it will be highlighted in the editor.

Replace:

Click this button to replace the last found text with the replacement text. This only applies to if a search has been performed, and if the search found a match.

Replace All:

Click this button to replace all occurrences of the search text with the replacement text. This is really useful if you decide to change the name of a variable or other item throughout your program.

Cancel:

Click this button to exit the window.

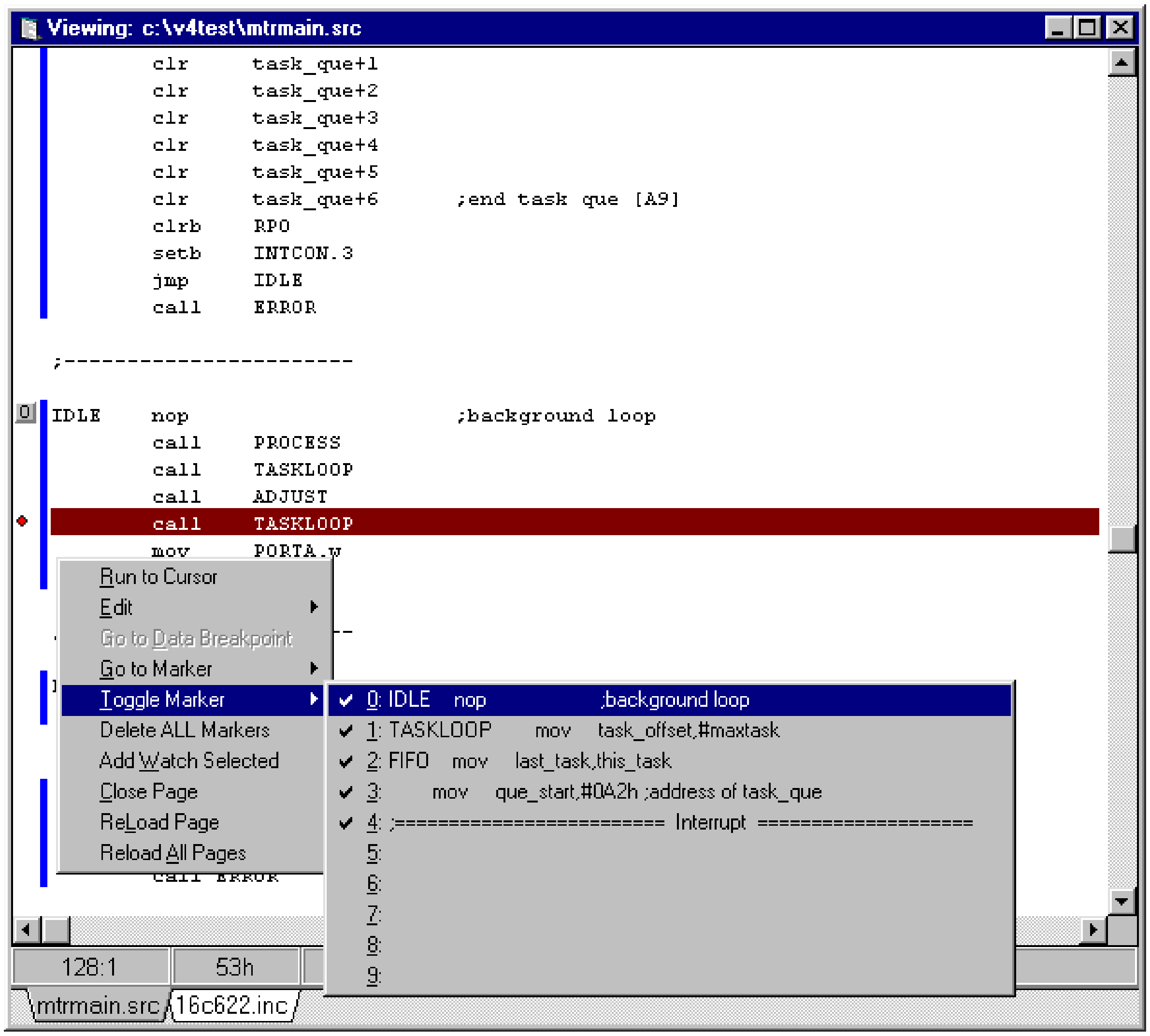
A related function is Search Again. This function uses settings from the most recent search, but does not show the normal search window. It’s useful for performing the same search over and over.

To activate the search-again function, select Search Again from the Edit menu, or press F3.

3.2.3.2.6 Place markers

Markers are used to mark locations in the editor, so that they may be found later. Only ten markers are allowed, so they are normally used to mark important routines which are under development. Later, when you want to find a marked location, you just use the pop-up menu 80.

To set or clear a marker on the current line (the line where the cursor is), click the right mouse button to see the editor’s pop-up menu, then select Toggle Marker, and then select the marker number you want to use (0-9).

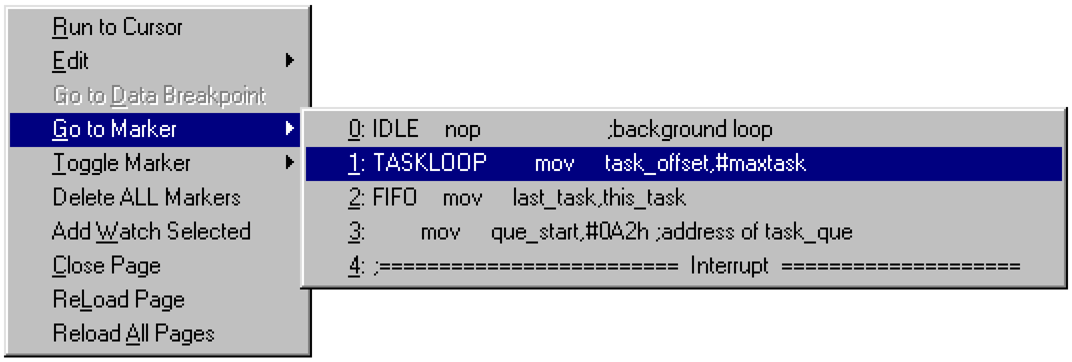

Viewing: c:\v4test\mtrmain.src

```
        clr     task_que+1
        clr     task_que+2
        clr     task_que+3
        clr     task_que+4
        clr     task_que+5
        clr     task_que+6      ;end task que [A9]
        clrb    RP0
        setb    INTCON.3
        jmp     IDLE
        call    ERROR

;-----------------------

IDLE    nop                     ;background loop
        call    PROCESS
        call    TASKLOOP
        call    ADJUST
        call    TASKLOOP
        mov     PORTA,w
```

Run to Cursor
Edit
Go to Data Breakpoint
Go to Marker
Toggle Marker
Delete ALL Markers
Add Watch Selected
Close Page
ReLoad Page
Reload All Pages
✓ 0: IDLE nop ;background loop
✓ 1: TASKLOOP mov task_offset,#maxtask
✓ 2: FIFO mov last_task,this_task
✓ 3: mov que_start,#0A2h ;address of task_que
✓ 4: ;========================= Interrupt =====================
5:
6:
7:
8:
9:
128:1
53h
mtrmain.src
16c622.inc

1. If the chosen marker is not already present, then it will be placed on the line.
2. If the chosen marker is already present, then it will be removed.

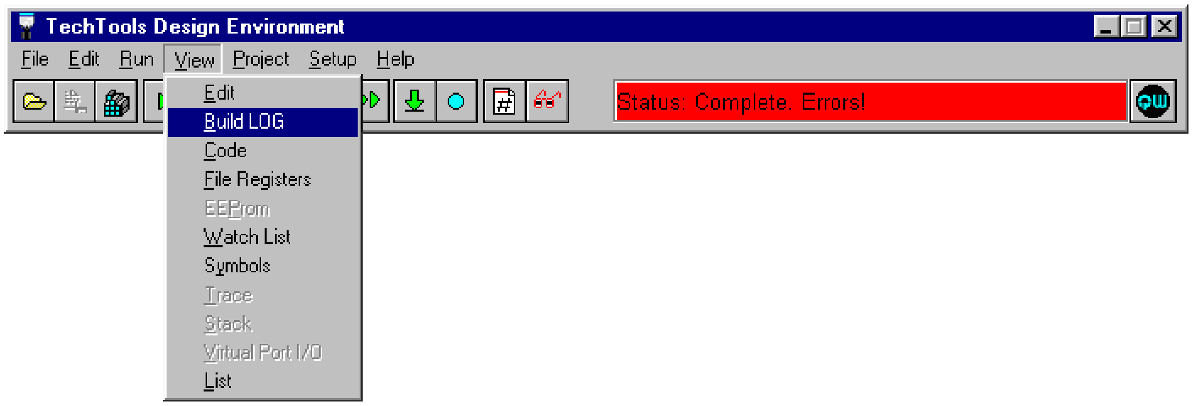
To find a marker that was placed earlier, select Go to Marker fr om the pop-up menu 80, and then select the marker you want to find. The editor will move to the line where the marker was placed.



```
0: IDLE    nop              ;background loop
1: TASKLOOP     mov    task_offset,#maxtask
2: FIFO    mov    last_task,this_task
3:       mov    que_start,#0A2h ;address of task_que
4: ;========================= Interrupt ====================
```

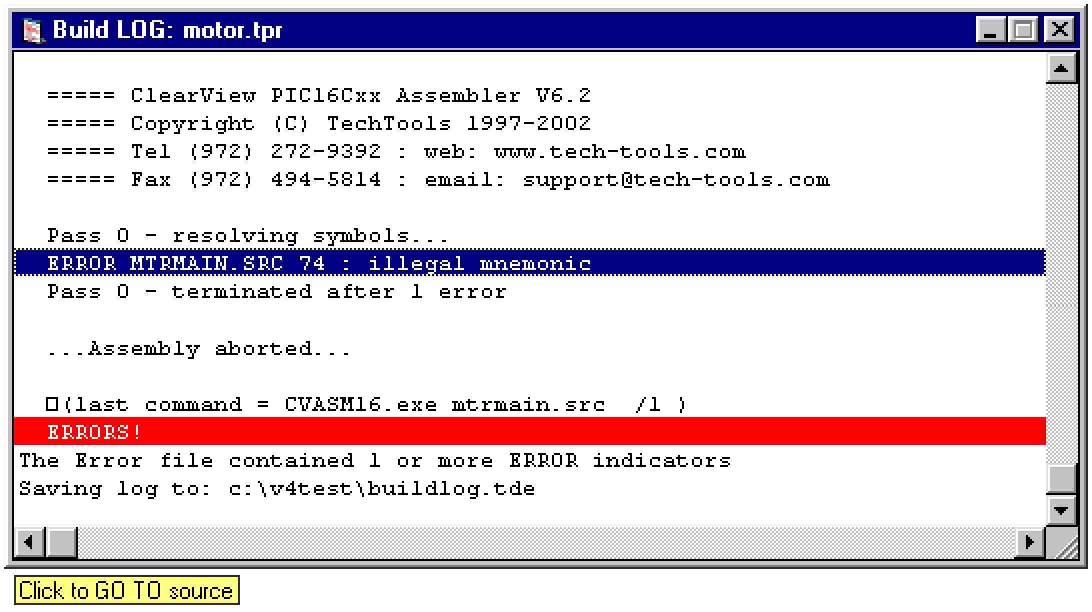
3.2.3.3 Build Log

If errors are encountered when TDE tries to assemble or compile your source code, then they will be shown in the Build Log window.



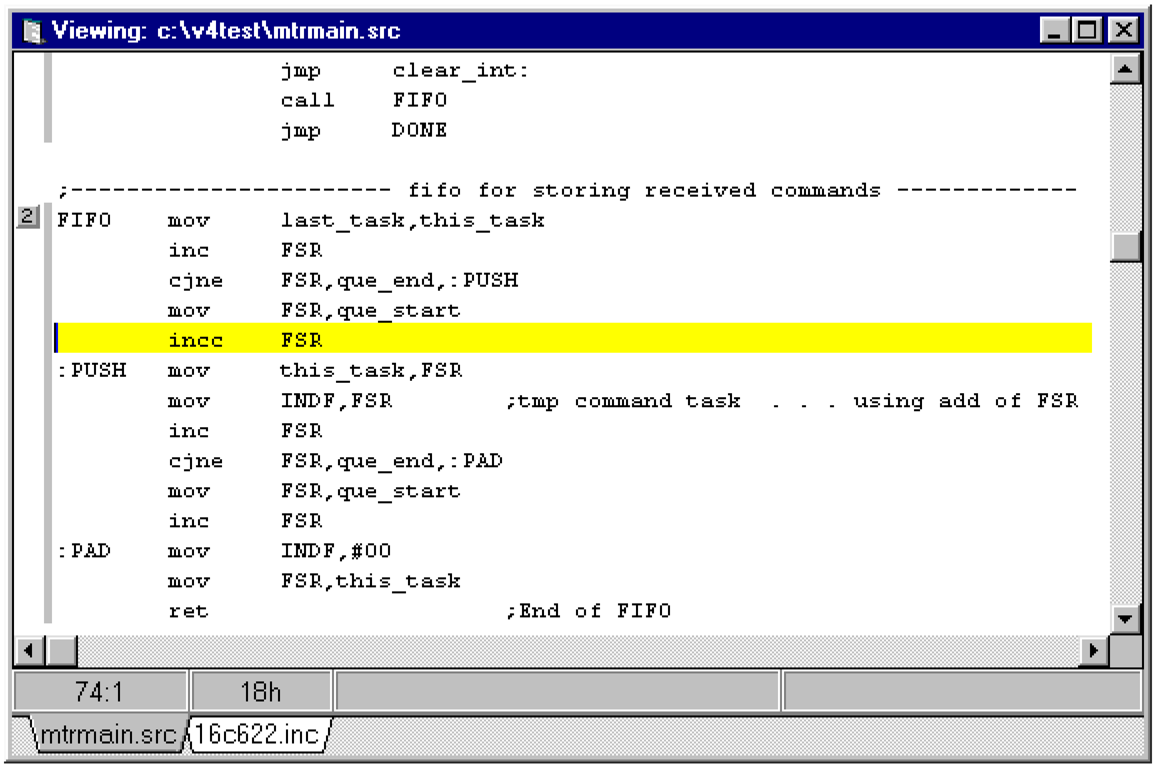
Status: Complete. Errors!

The Build Log is automatically shown if TDE encounters errors. However, if you would like to see the Build Log (if it's behind another window), just select Build Log from the View menu; the following window will appear:



```
 ===== ClearView PIC16Cxx Assembler V6.2
 ===== Copyright (C) TechTools 1997-2002
 ===== Tel (972) 272-9392 : web: www.tech-tools.com
 ===== Fax (972) 494-5814 : email: support@tech-tools.com

 Pass 0 - resolving symbols...
 ERROR MTRMAIN.SRC 74 : illegal mnemonic
 Pass 0 - terminated after 1 error

 ...Assembly aborted...

 □(last command = CVASM16.exe mtrmain.src  /l )
 ERRORS!
The Error file contained 1 or more ERROR indicators
Saving log to: c:\v4test\buildlog.tde
```

Each line in the Build Log marked "error", corresponds to an error in the source code. To see a corresponding error in the source code, just click on the error in the errors window; the original source code line will be highlighted:

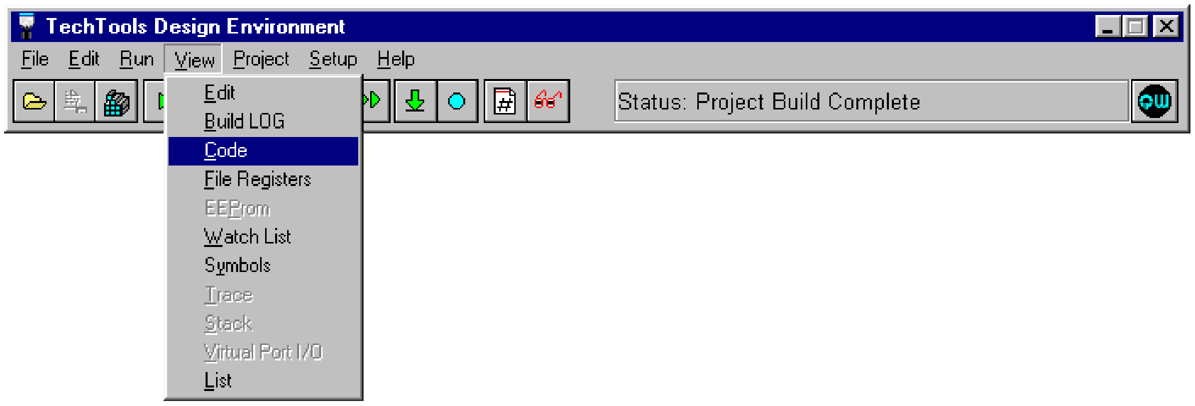


```
                jmp     clear_int:
                call    FIFO
                jmp     DONE

;----------------------- fifo for storing received commands -------------
FIFO    mov     last_task,this_task
        inc     FSR
        cjne    FSR,que_end,:PUSH
        mov     FSR,que_start
        incc    FSR
:PUSH   mov     this_task,FSR
        mov     INDF,FSR        ;tmp command task  . . . using add of FSR
        inc     FSR
        cjne    FSR,que_end,:PAD
        mov     FSR,que_start
        inc     FSR
:PAD    mov     INDF,#00
        mov     FSR,this_task
        ret                     ;End of FIFO
```

3.2.3.4 Code Window

The code window shows the actual Microchip hex code that runs in the emulator. Hex code is the result after source code is assembled or compiled.

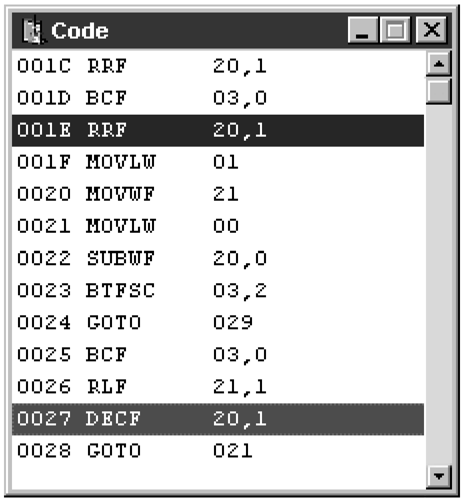
To see the code window, select Code from the View menu.


TechTools Design Environment
File Edit Run View Project Setup Help
Edit
Build LOG
Code
File Registers
EEProm
Watch List
Symbols
Trace
Stack
Virtual Port I/O
List
Status: Project Build Complete

In most cases, you'll work with source code in the editor window. Most users prefer to work with source code, since it's easier to read and edit. However, there may be times when you need to emulate code that only exists as hex code (the source code may have been lost, etc). In such cases, you can load the hex file by adding it to your project using the Project option of the Setup menu (explained earlier in " Creating a Single-file Project 46 ").

Whether the hex code came from an internal operation (assembly or compilation), or from an external file, it will be available in the code window. However, if the hex file was loaded from disk, then the code window will be the only window in which you can execute and manipulate the code.

Like the editor window, the code window shows the execution point and any breakpoints; breakpoints can be set and cleared by double-clicking on any executable line.



```
001C RRF     20,1
001D BCF     03,0
001E RRF     20,1
001F MOVLW   01
0020 MOVWF   21
0021 MOVLW   00
0022 SUBWF   20,0
0023 BTFSC   03,2
0024 GOTO    029
0025 BCF     03,0
0026 RLF     21,1
0027 DECF    20,1
0028 GOTO    021
```

Breakpoint (red):

Indicates a breakpoint; double-click to set or clear a breakpoint.

Execution point (green):

Shows the instruction being executed. If execution is stopped, then this line will be executed when execution is resumed.

Selected line (navy):

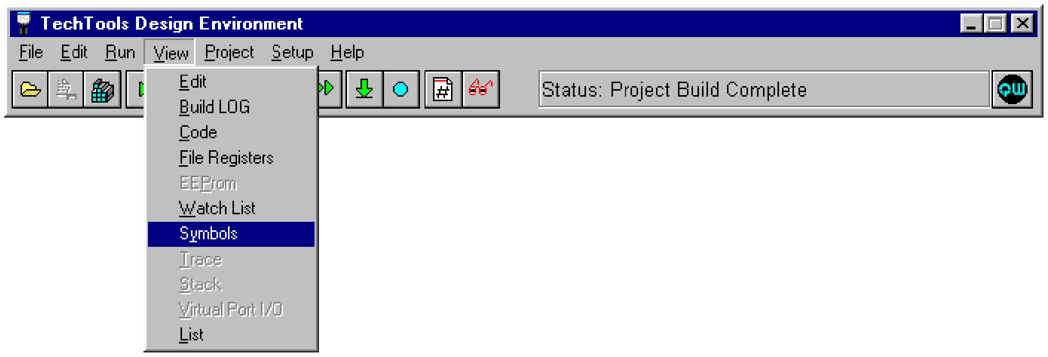
A Single-click will highlight the selected line in the Code Window as well as the corresponding source-line in the Editor Window 82. This feature helps the user to navigate and quickly cross-reference the source to the actual compiled Opcode/Operands.

3.2.3.5 Symbols Window

The Symbols window contains several special windows important to Editing and Debugging.

- Special Registers 97
- Variables 99
- Labels 103
- Files 105

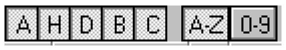
To see the symbols window, select Symbols from the View menu.


TechTools Design Environment
File Edit Run View Project Setup Help
Edit
Build LOG
Code
File Registers
EEProm
Watch List
Symbols
Trace
Stack
Virtual Port I/O
List
Status: Project Build Complete

3.2.3.5.1 Special Registers Window

This window allows you to see and modify the emulated PIC's special registers. Special registers include the W register, Indirect addressing register, Option register, Program Counter, Status register, I/O port registers, etc.

Special Register Viewing Options:



At the top of the special register window are several buttons that control what items are displayed and how they are displayed.

A - Toggles displaying of the Address Column.

H - Toggles displaying of the HEX Column.

D - Toggles displaying of the Decimal Column.

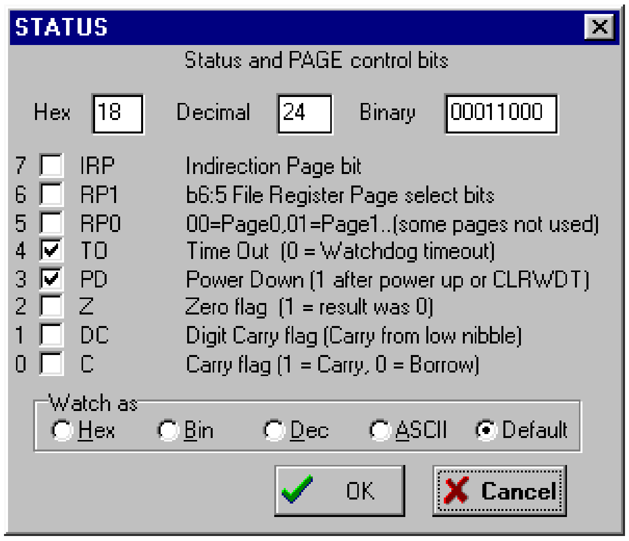
B - Toggles displaying of the Binary Column.

C - Toggles displaying of the ASCII (Character) Column.

A-Z Sorts the registers by NAME

0-9 Sorts the registers by ADDRESS

To modify the value in a register, just double-click on the value; the following register modification window will appear:



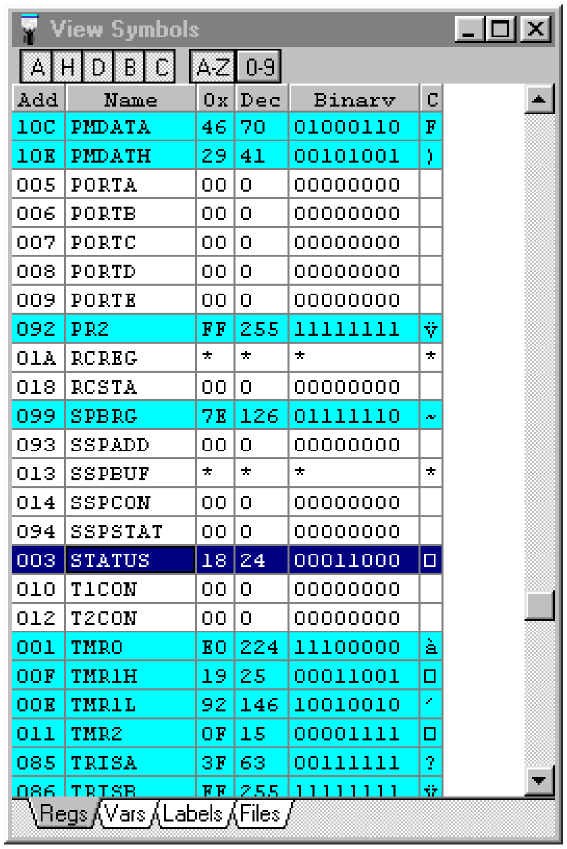
STATUS

Status and PAGE control bits

Hex 18 Decimal 24 Binary 00011000

7	☐	IRP	Indirection Page bit
6	☐	RP1	b6:5 File Register Page select bits
5	☐	RP0	00=Page0,01=Page1..(some pages not used)
4	☑	TO	Time Out (0 = Watchdog timeout)
3	☑	PD	Power Down (1 after power up or CLRWDT)
2	☐	Z	Zero flag (1 = result was 0)
1	☐	DC	Digit Carry flag (Carry from low nibble)
0	☐	C	Carry flag (1 = Carry, 0 = Borrow)

Watch as: Hex Bin Dec ASCII [✓ Default]

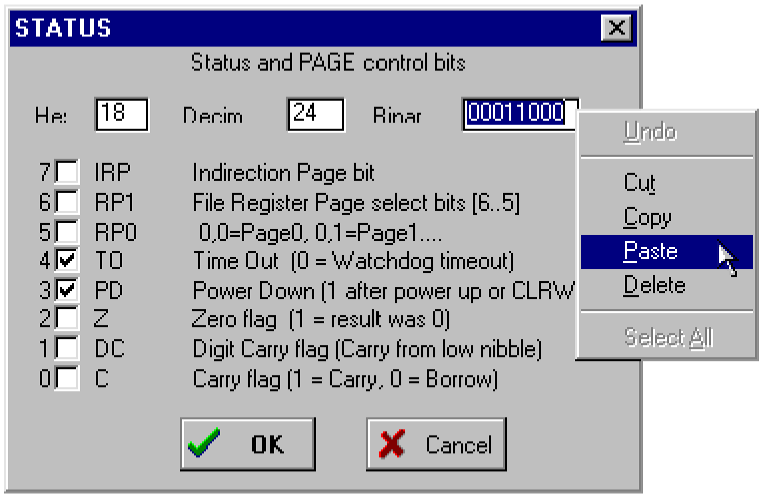


View Symbols

Add	Name	0x	Dec	Binary	C
10C	PMDATA	46	70	01000110	F
10E	PMDATH	29	41	00101001	)
005	PORTA	00	0	00000000	
006	PORTB	00	0	00000000	
007	PORTC	00	0	00000000	
008	PORTD	00	0	00000000	
009	PORTE	00	0	00000000	
092	PR2	FF	255	11111111	ÿ
01A	RCREG	*	*	*	*
018	RCSTA	00	0	00000000	
099	SPBRG	7E	126	01111110	~
093	SSPADD	00	0	00000000	
013	SSPBUF	*	*	*	*
014	SSPCON	00	0	00000000	
094	SSPSTAT	00	0	00000000	
003	STATUS	18	24	00011000	□
010	T1CON	00	0	00000000	
012	T2CON	00	0	00000000	
001	TMR0	E0	224	11100000	à
00F	TMR1H	19	25	00011001	□
00E	TMR1L	92	146	10010010	'
011	TMR2	0F	15	00001111	□
085	TRISA	3F	63	00111111	?
086	TRISB	FF	255	11111111	ÿ

Some registers and bits of registers are write-only. Such registers and bits cannot be modified.
Registers that will be affected by READING, display an ' * ' instead of the actual dat a (as seen above in the RCREG and SSPBUFF registers). This provides the user with the option of forcing a read by DBL-Clicking on the register.

From this window, you can enter a new register value in hex, binary, or decimal. You can also use the handy register modification pop-up menu. To use the pop-up menu, click the right mouse button; you'll see the pop-up menu as shown here.


STATUS

Status and PAGE control bits

He: 18 Decim 24 Binar 00011000

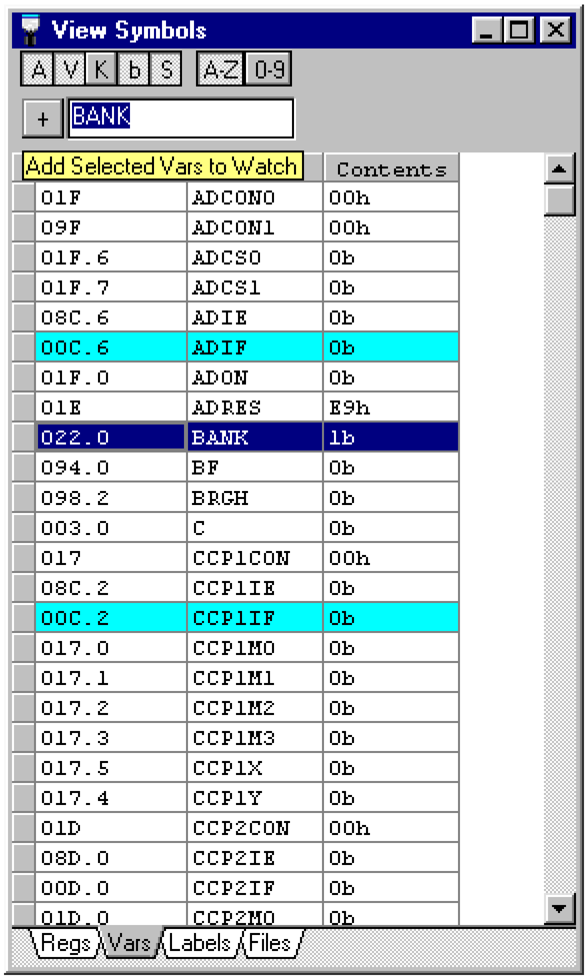
7	☐	IRP	Indirection Page bit
6	☐	RP1	File Register Page select bits [6..5]
5	☐	RP0	0,0=Page0, 0,1=Page1....
4	✓	TO	Time Out (0 = Watchdog timeout)
3	✓	PD	Power Down (1 after power up or CLRW
2	☐	Z	Zero flag (1 = result was 0)
1	☐	DC	Digit Carry flag (Carry from low nibble)
0	☐	C	Carry flag (1 = Carry, 0 = Borrow)

The pop-up menu gives you several useful editing functions, including undo, cut, copy, and paste. These functions work just as their counterparts in the editor window. In fact, the cut, copy, and paste functions use the same buffer memory as the cut, copy, and paste functions in the editor. Therefore, you can copy a value from the editor to a register, or from a register to the editor. Of course, you can also copy values from register to register. If you're not sure what these functions do, please see the editor window description 82 earlier in this section.

After making any necessary changes, click OK to accept the changes, or click Cancel to disregard them.

3.2.3.5.2 Variables Window

This window allows you to see and modify Defined RAM Locations, including the special registers if selected. From this window register values can be edited, variable display characteristics can be modified and items can be added to the Watch Window.



		Contents
01F	ADCON0	00h
09F	ADCON1	00h
01F.6	ADCS0	0b
01F.7	ADCS1	0b
08C.6	ADIE	0b
00C.6	ADIF	0b
01F.0	ADON	0b
01E	ADRES	E9h
022.0	BANK	1b
094.0	BF	0b
098.2	BRGH	0b
003.0	C	0b
017	CCP1CON	00h
08C.2	CCP1IE	0b
00C.2	CCP1IF	0b
017.0	CCP1M0	0b
017.1	CCP1M1	0b
017.2	CCP1M2	0b
017.3	CCP1M3	0b
017.5	CCP1X	0b
017.4	CCP1Y	0b
01D	CCP2CON	00h
08D.0	CCP2IE	0b
00D.0	CCP2IF	0b
01D.0	CCP2M0	0b

Variables Window Viewing Options:

At the top of the special register window are several buttons that control what items are displayed and how they are displayed.

A - Toggles displaying of the Address Column (Val).

V - Toggles displaying Defined General Purpose RAM locations (Variables).

K - Toggles displaying Defined Constants.

b - Toggles displaying of Defined Bits.

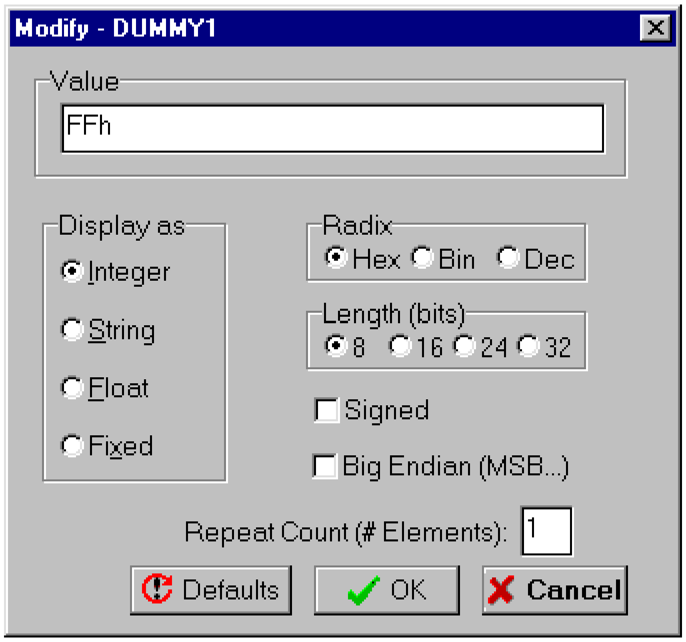
S - Toggles displaying of the Special Registers.

A-Z Sorts the registers by NAME

0-9 Sorts the registers by ADDRESS

Modifying a Register Value and Display Format:

To modify the value in a register or to change the display format, just double-click on the value and the modification dialog will appear:


Modify - DUMMY1

Value: FFh

Display as: [✓ Integer] String Float Fixed

Radix: [✓ Hex] Bin Dec

Length (bits): [✓ 8] 16 24 32

☐ Signed

☐ Big Endian (MSB...)

Repeat Count (# Elements): 1

Integer Options:

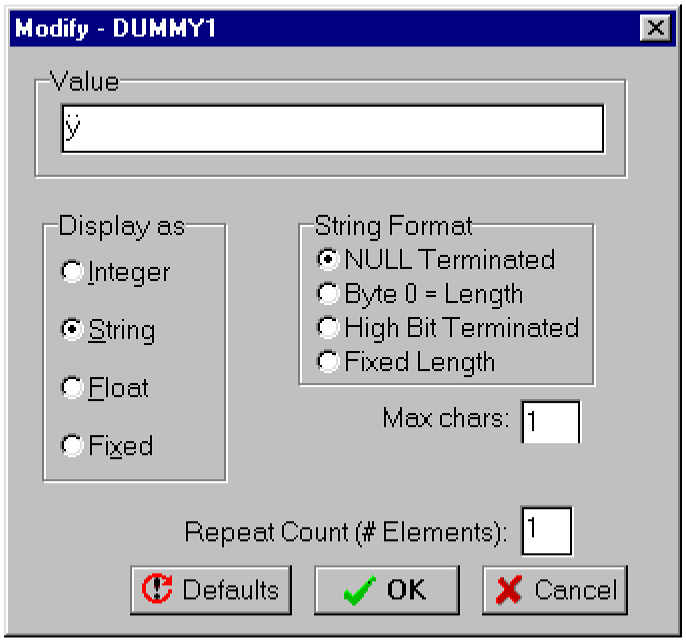
Radix: - Select HEX, Binary or Decimal formatting.

Length: - Group the selected number of bits as a single value (consecutive).

Signed: - Display as a signed value.

Big Endian: - If the Length is greater than 8 bits, this option will display the MSB first when selected.

Repeat Count - Treat as an array of the above settings with the specified number of elements.


Modify - DUMMY1

Value: ÿ

Display as: ( ) Integer [✓ String] ( ) Float ( ) Fixed

String Format: [✓ NULL Terminated] ( ) Byte 0 = Length ( ) High Bit Terminated ( ) Fixed Length

Max chars: 1

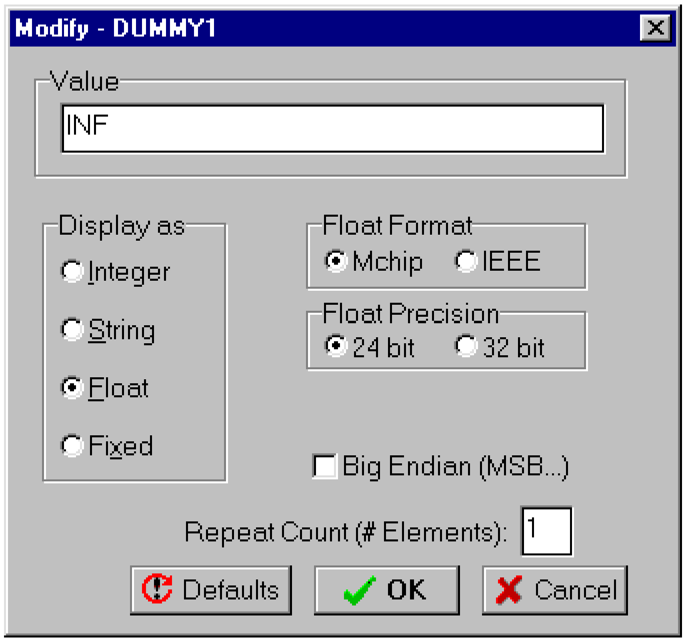
Repeat Count (# Elements): 1

String Options:

Format: - Select the desired string type.

Max Chars - Enter the maximum number of characters to associate with this string.

Repeat Count - Treat as an array of the above settings with the specified number of elements.


Modify - DUMMY1

Value: INF

Display as: ( ) Integer ( ) String [✓ Float] ( ) Fixed

Float Format: [✓ Mchip] ( ) IEEE

Float Precision: [✓ 24 bit] ( ) 32 bit

[ ] Big Endian (MSB...)

Repeat Count (# Elements): 1

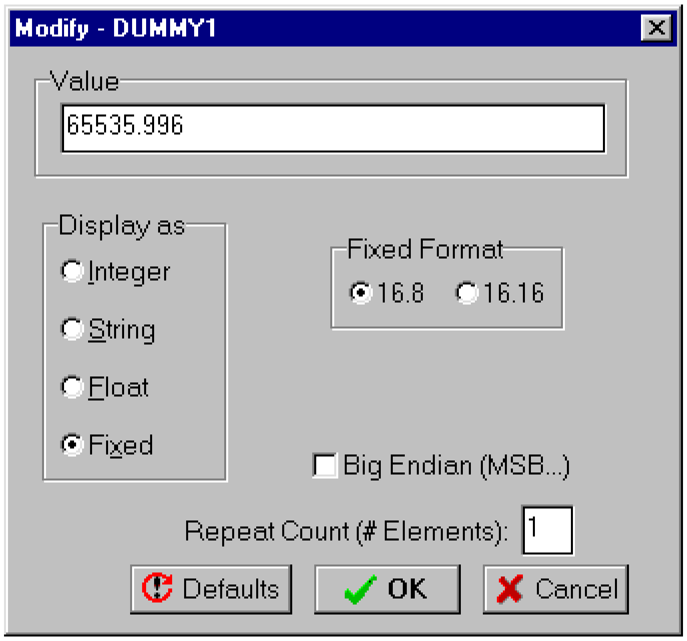
Float Options:

Format - Choose the desired float format.

Precision - choose from 24 or 32 bit precision.

Big Endian - If selected, value will be displayed MSB first.

Repeat Count - Treat as an array of the above settings with the specified number of elements.



Modify - DUMMY1

Value: 65535.996

Display as: ( ) Integer ( ) String ( ) Float (● Fixed)

Fixed Format: (● 16.8) ( ) 16.16

☐ Big Endian (MSB...)

Repeat Count (# Elements): 1

Fixed Options:

Format - Choose from 16.8 or 16.16 format.

Big Endian - If selected, value will be displayed MSB first.

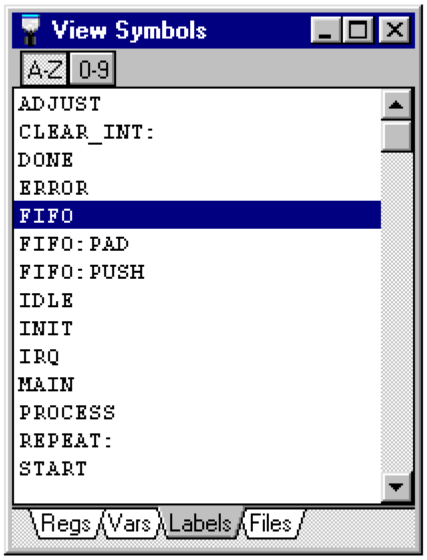
Repeat Count - Treat as an array of the above settings with the specified number of elements.

Adding Watches:

Watches can be added to the watch window three ways. First, select one or more registers by using the left mouse button (and optionally the Shift or Control keys for multiple selections). Then, either Drag & Drop the selected items onto the Watch Window, or click the Add button at the top of the window (+). The third method is to enter the address manually in the edit box at the top of the window and then press the Enter key (or click the add button). With this method, the address must be specified in HEX, proceeded by ' 0x '. For example, address 127 must be entered a s 0x7F.

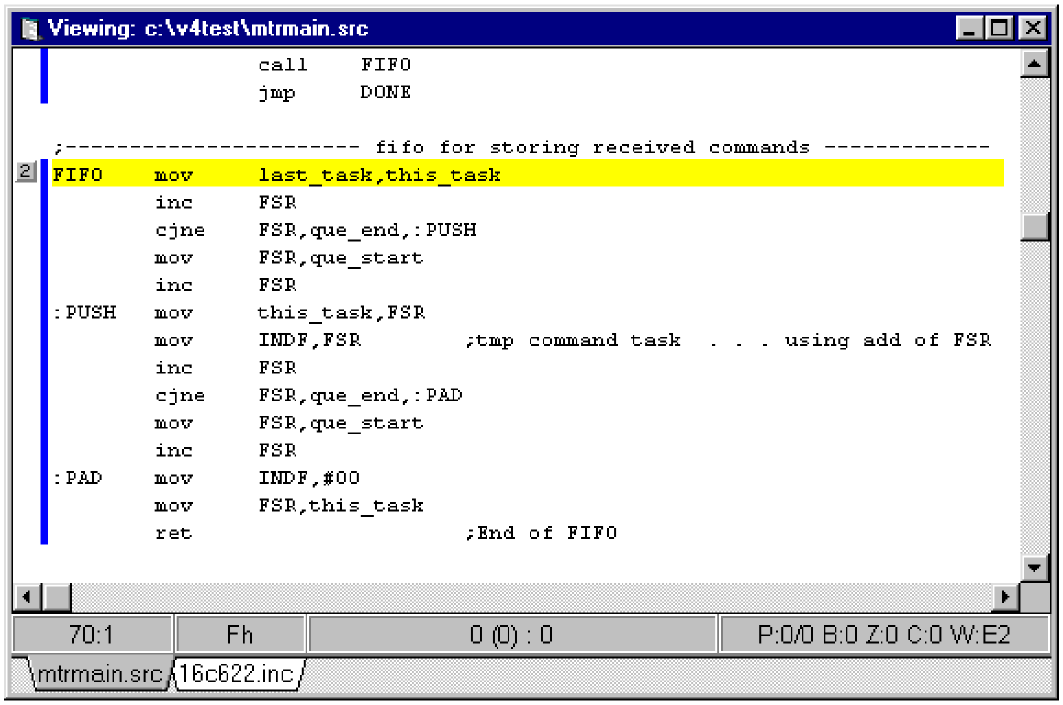
3.2.3.5.3 Labels Window

From this window, you can jump to the position of a specific label in your source code. Double-Click on any label in this window and the associated source code line will be brought into view and highlighted yellow.



```
ADJUST
CLEAR_INT:
DONE
ERROR
FIFO
FIFO:PAD
FIFO:PUSH
IDLE
INIT
IRQ
MAIN
PROCESS
REPEAT:
START
```

Local labels are grouped with its Global label as seen with the label ' FIFO ' above. FIFO is a global label that has two local labels defined as PAD and PUSH.



```
                call    FIFO
                jmp     DONE

;----------------------- fifo for storing received commands -------------
FIFO    mov     last_task,this_task
        inc     FSR
        cjne    FSR,que_end,:PUSH
        mov     FSR,que_start
        inc     FSR
:PUSH   mov     this_task,FSR
        mov     INDF,FSR        ;tmp command task  . . . using add of FSR
        inc     FSR
        cjne    FSR,que_end,:PAD
        mov     FSR,que_start
        inc     FSR
:PAD    mov     INDF,#00
        mov     FSR,this_task
        ret                     ;End of FIFO
```

```
70:1    Fh    0 (0) : 0    P:0/0 B:0 Z:0 C:0 W:E2
```

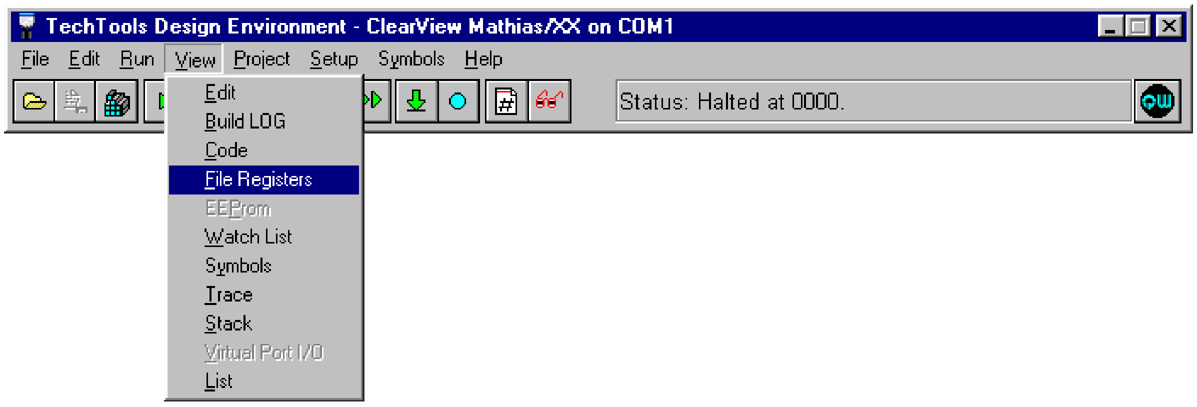
3.2.3.5.4 Files Window

The Files Window will reflect all files referenced in the debugging information provided by the assembler or compiler. This information is refreshed after each successful build.

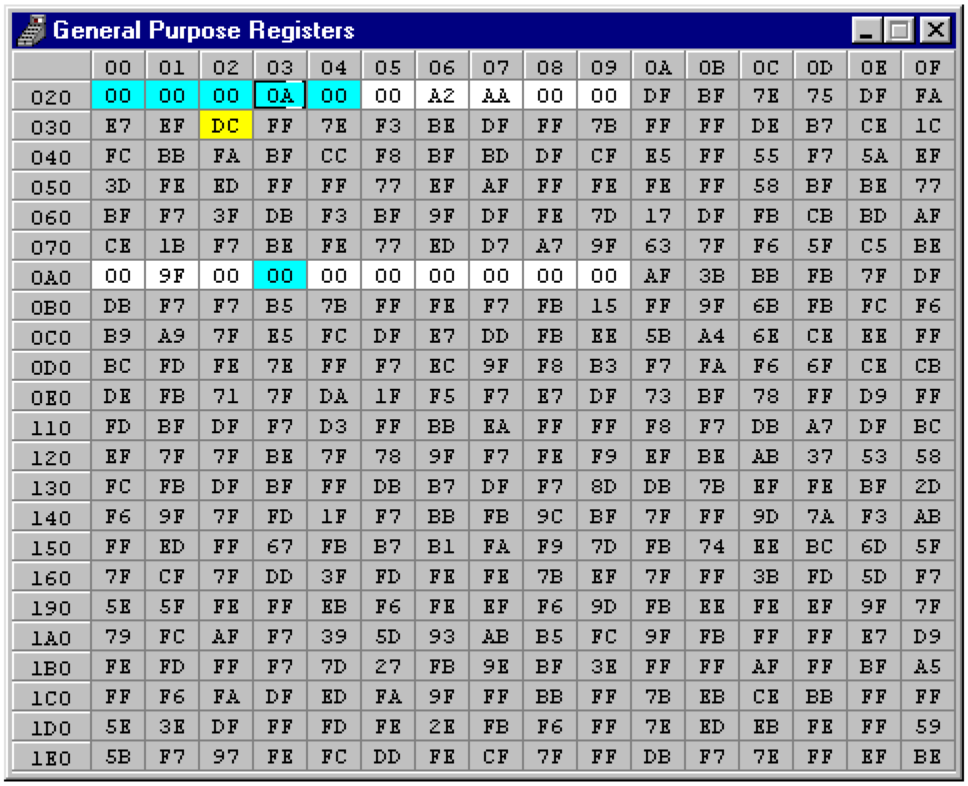
Double-Click on any file in this window and it will be displayed in the source code editor. If the file has not been opened since the start of your session, TDE will open the file for you.

3.2.3.6 File Registers Window

This window allows you to see and modify the emulated PIC's file registers. Most file registers serve as general-purpose storage space for variables and program data.



To see the file registers window, select File Registers from the View menu; the window below will appear.


General Purpose Registers

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
020	00	00	00	0A	00	00	A2	AA	00	00	DF	BF	7E	75	DF	FA
030	E7	EF	DC	FF	7E	F3	BE	DF	FF	7B	FF	FF	DE	B7	CE	1C
040	FC	BB	FA	BF	CC	F8	BF	BD	DF	CF	E5	FF	55	F7	5A	EF
050	3D	FE	ED	FF	FF	77	EF	AF	FF	FE	FE	FF	58	BF	BE	77
060	BF	F7	3F	DB	F3	BF	9F	DF	FE	7D	17	DF	FB	CB	BD	AF
070	CE	1B	F7	BE	FE	77	ED	D7	A7	9F	63	7F	F6	5F	C5	BE
0A0	00	9F	00	00	00	00	00	00	00	00	AF	3B	BB	FB	7F	DF
0B0	DB	F7	F7	B5	7B	FF	FE	F7	FB	15	FF	9F	6B	FB	FC	F6
0C0	B9	A9	7F	E5	FC	DF	E7	DD	FB	EE	5B	A4	6E	CE	EE	FF
0D0	BC	FD	FE	7E	FF	F7	EC	9F	F8	B3	F7	FA	F6	6F	CE	CB
0E0	DE	FB	71	7F	DA	1F	F5	F7	E7	DF	73	BF	78	FF	D9	FF
110	FD	BF	DF	F7	D3	FF	BB	EA	FF	FF	F8	F7	DB	A7	DF	BC
120	EF	7F	7F	BE	7F	78	9F	F7	FE	F9	EF	BE	AB	37	53	58
130	FC	FB	DF	BF	FF	DB	B7	DF	F7	8D	DB	7B	EF	FE	BF	2D
140	F6	9F	7F	FD	1F	F7	BB	FB	9C	BF	7F	FF	9D	7A	F3	AB
150	FF	ED	FF	67	FB	B7	B1	FA	F9	7D	FB	74	EE	BC	6D	5F
160	7F	CF	7F	DD	3F	FD	FE	FE	7B	EF	7F	FF	3B	FD	5D	F7
190	5E	5F	FE	FF	EB	F6	FE	EF	F6	9D	FB	EE	FE	EF	9F	7F
1A0	79	FC	AF	F7	39	5D	93	AB	B5	FC	9F	FB	FF	FF	E7	D9
1B0	FE	FD	FF	F7	7D	27	FB	9E	BF	3E	FF	FF	AF	FF	BF	A5
1C0	FF	F6	FA	DF	ED	FA	9F	FF	BB	FF	7B	EB	CE	BB	FF	FF
1D0	5E	3E	DF	FF	FD	FE	2E	FB	F6	FF	7E	ED	EB	FE	FF	59
1E0	5B	F7	97	FE	FC	DD	FE	CF	7F	FF	DB	F7	7E	FF	EF	BE

Color Legend:

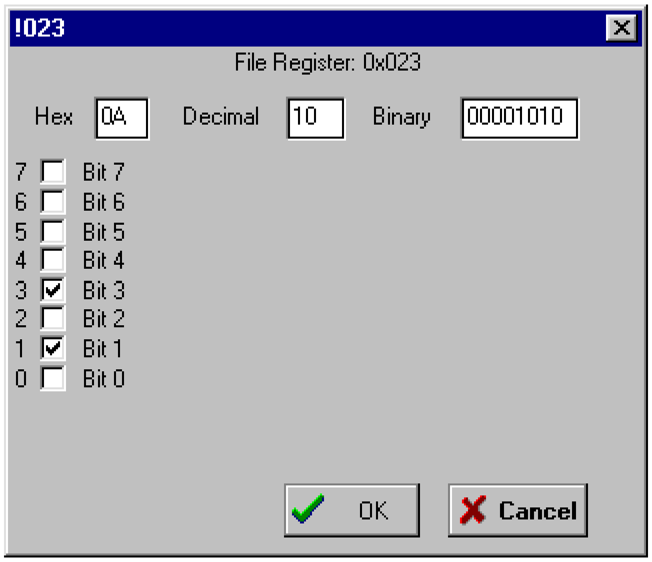
White - Defined ram location that has NOT changed since the last Halt.

Aqua - Defined ram location that has changed since the last Halt.

Gray - Undefined ram location that has NOT changed since the last Halt.

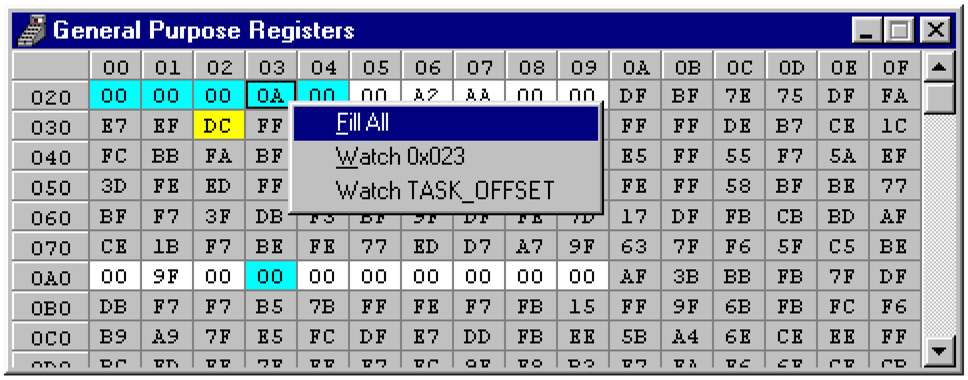
Yellow - Undefined ram location that has changed since the last Halt.

The file register window works just like the previously-discussed special registers window. To modify the value in a register, just double-click on the value; the following window will appear:



From this window, you can enter a new register value in hex, binary or decimal. You can also use the register modification pop-up menu to cut, copy or paste values. After making any necessary changes, click OK to accept the changes, or click Cancel to disregard them.

The File Registers window has additional options that are accessible by a POPUP Menu:



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
020	00	00	00	0A	00	00	A7	AA	00	00	DF	BF	7E	75	DF	FA
030	E7	EF	DC	FF	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	FF	FF	DE	B7	CE	1C
040	FC	BB	FA	BF	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	E5	FF	55	F7	5A	EF
050	3D	FE	ED	FF	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	FE	FF	58	BF	BE	77
060	BF	F7	3F	DB	F3	EF	9F	DF	FE	7D	17	DF	FB	CB	BD	AF
070	CE	1B	F7	BE	FE	77	ED	D7	A7	9F	63	7F	F6	5F	C5	BE
0A0	00	9F	00	00	00	00	00	00	00	00	AF	3B	BB	FB	7F	DF
0B0	DB	F7	F7	B5	7B	FF	FE	F7	FB	15	FF	9F	6B	FB	FC	F6
0C0	B9	A9	7F	E5	FC	DF	E7	DD	FB	EE	5B	A4	6E	CE	EE	FF

FILL ALL - Select this option to fill all general purpose ram locations with a specific value.

Watch - Select this option to add the ram location (or its label if defined) to the watch window.

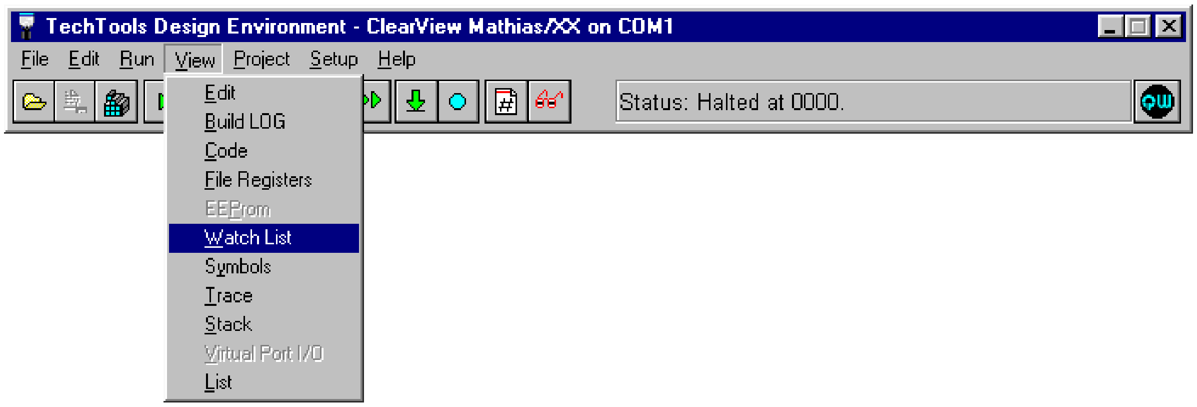
3.2.3.7 Watch List Window

This window maintains a list of variables and/or registers that you'd like to watch. By placing items in this window, you can easily see their values as your program runs. You could do the same thing by watching the appropriate locations in the Special Registers 97 , Variable Window 99 and File Registers windows 105, but using the Watch List window can speed up your debugging by grouping items of interest.

This section covers the following procedures:

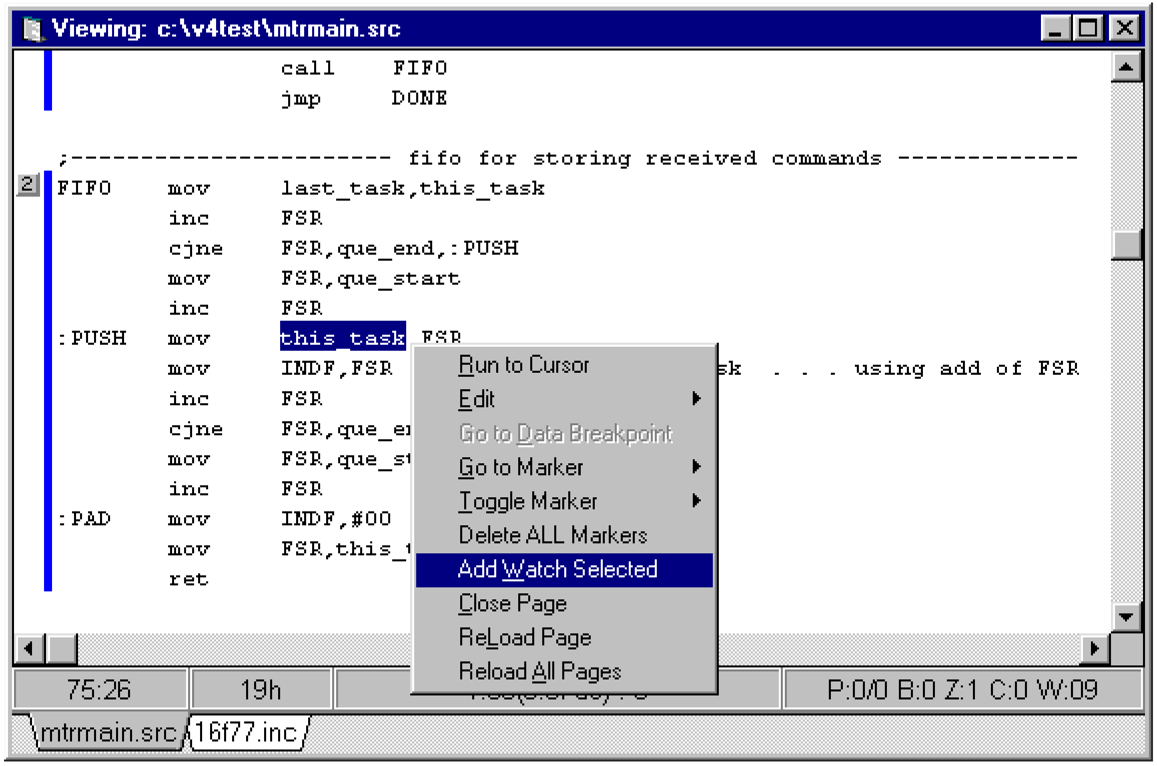
Adding items from Source Code 108
Adding items from the Variables Window 109
Modifying Values of watched items 112
Watch List Options 115

To see the watch list window, select Watch List from the View menu or click the View Watches Button.



3.2.3.7.1 Adding Watch items by Highlighting

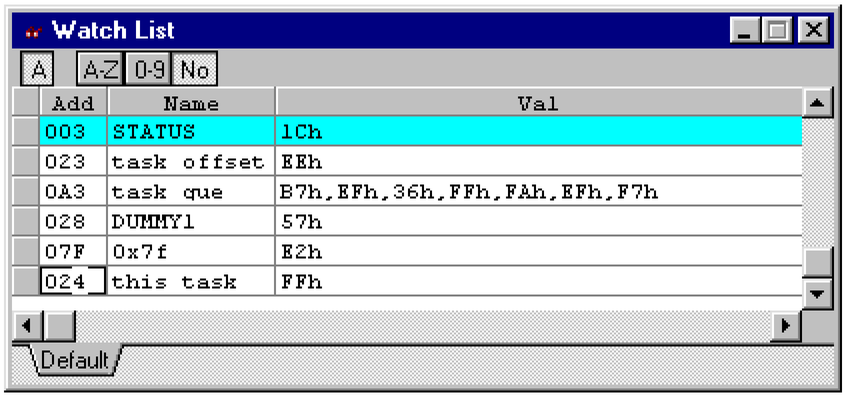
Items can be added to the Watch Window directly from the source code editor. To add a watch, simply Highlight the item to watch, then Right-Click in the editor and select Add Watch Selected from the popup menu.



Viewing: c:\v4test\mtrmain.src

```
                call    FIFO
                jmp     DONE

;----------------------- fifo for storing received commands -------------
FIFO    mov     last_task,this_task
        inc     FSR
        cjne    FSR,que_end,:PUSH
        mov     FSR,que_start
        inc     FSR
:PUSH   mov     this_task FSR
        mov     INDF,FSR                 sk  . . . using add of FSR
        inc     FSR
        cjne    FSR,que_e
        mov     FSR,que_st
        inc     FSR
:PAD    mov     INDF,#00
        mov     FSR,this_t
        ret
```

The item will be added to the currently visible Watch Group in the Watch Window. If you have not added a watch group, then the item will be placed in the Default Watch Group.



Watch List

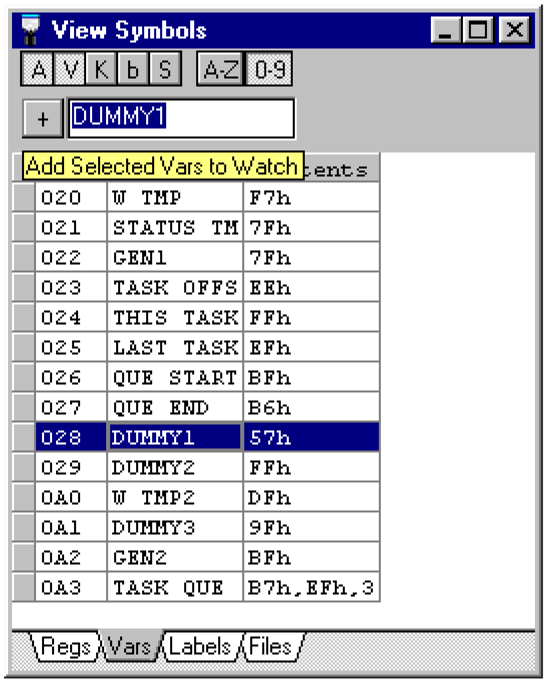
Add	Name	Val
003	STATUS	1Ch
023	task offset	EEh
0A3	task que	B7h,EFh,36h,FFh,FAh,EFh,F7h
028	DUMMY1	57h
07F	0x7f	E2h
024	this task	FFh

3.2.3.7.2 Add Watch Dialog

Adding Watches:

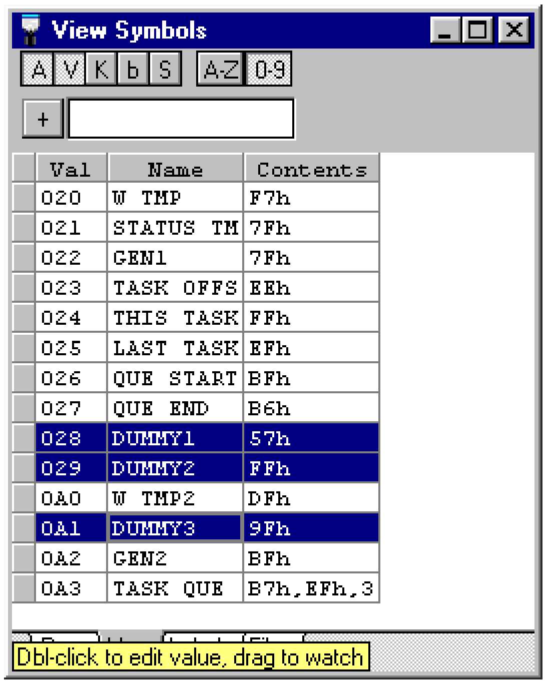
Watches can be added to the Watch wind ow from the Variables Window in the three ways described below. To select which items are visible in the Variables Window, see: Symbols - Variables Window 99 .

Select and Add - First, select one or more registers by using the left mouse button (and optionally the Shift or Control keys for multiple selections). Then, click the Add button at the top of the window (+).



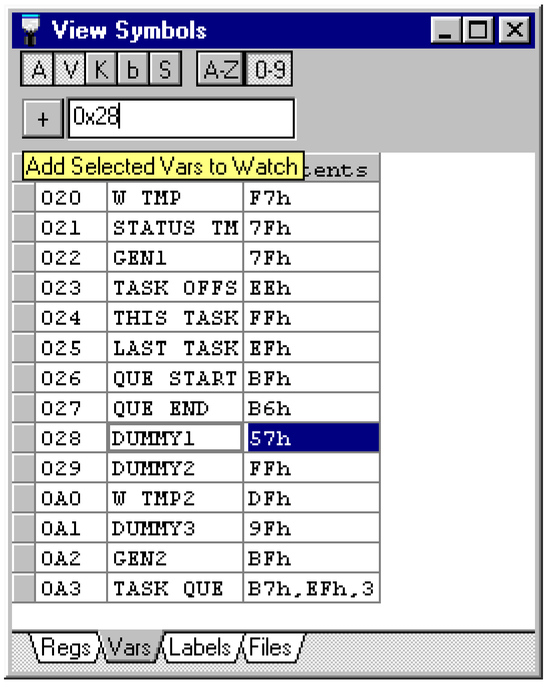
		ents
020	W TMP	F7h
021	STATUS TM	7Fh
022	GEN1	7Fh
023	TASK OFFS	EEh
024	THIS TASK	FFh
025	LAST TASK	EFh
026	QUE START	BFh
027	QUE END	B6h
028	DUMMY1	57h
029	DUMMY2	FFh
0A0	W TMP2	DFh
0A1	DUMMY3	9Fh
0A2	GEN2	BFh
0A3	TASK QUE	B7h,EFh,3

Select and Drag & Drop - First, select one or more registers by using the left mouse button (and optionally the Shift or Control keys for multiple selections). Then, Drag & Drop the selected items onto the Watch Window.



Val	Name	Contents
020	W TMP	F7h
021	STATUS TM	7Fh
022	GEN1	7Fh
023	TASK OFFS	EEh
024	THIS TASK	FFh
025	LAST TASK	EFh
026	QUE START	BFh
027	QUE END	B6h
028	DUMMY1	57h
029	DUMMY2	FFh
0A0	W TMP2	DFh
0A1	DUMMY3	9Fh
0A2	GEN2	BFh
0A3	TASK QUE	B7h,EFh,3

Edit and Add - Enter the Name of the item or its address in the edit box at the top of the window and press the Enter key (or click the add button). If entering an address, it must be specified in HEX, proceeded by ' 0x '. For example, address 127 must be entered as 0x7F.



		ents
020	W TMP	F7h
021	STATUS TM	7Fh
022	GEN1	7Fh
023	TASK OFFS	EEh
024	THIS TASK	FFh
025	LAST TASK	EFh
026	QUE START	BFh
027	QUE END	B6h
028	DUMMY1	57h
029	DUMMY2	FFh
0A0	W TMP2	DFh
0A1	DUMMY3	9Fh
0A2	GEN2	BFh
0A3	TASK QUE	B7h,EFh,3

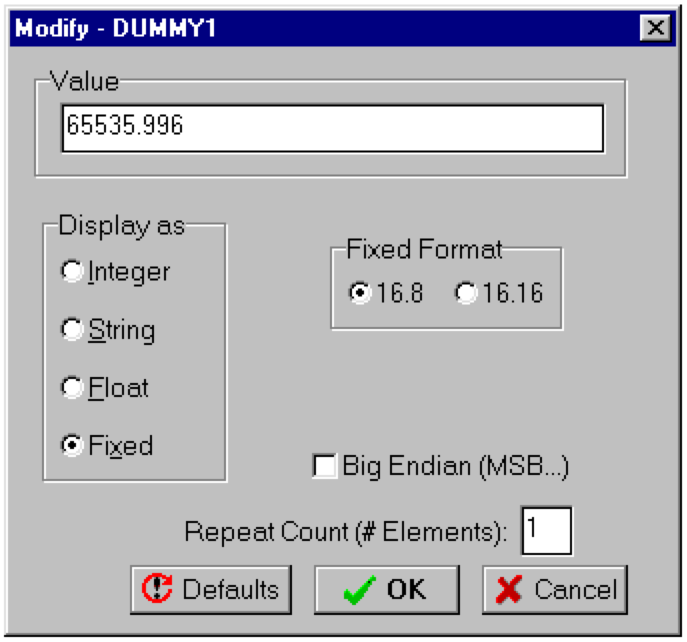
Note 1:
You can watch a specific bit of a symbol by declaring Symbol.Bit, where Symbol is the name and Bit is a bit number from 0-7.
Note 2:
You can watch a symbol by giving its address, rather than its name. To do so, declare it as 0xhhh, where 'hhh' is the hexadecimal address of the symbol. This must be done in some cases, such as when watching a local symbol from the MPC or CCS compilers. This notation is also valid when setting the File Reg ister options 120 in the Data Breakpoints Dialog 118.

3.2.3.7.3 Modifying Watch item values

In addition to watching items in the Watch Window, you can modify the item's value and display characteristics as well. To change the properties of an item, double-click on the item in the watch list:

Modifying a watched item or changing its display format:

To modify the value in a register or to change the display format, just double-click on the value and the modification dialog will appear:


Modify - DUMMY1

Value: 65535.996

Display as: ( ) Integer ( ) String ( ) Float [✓ Fixed]

Fixed Format: [✓ 16.8] ( ) 16.16

[ ] Big Endian (MSB...)

Repeat Count (# Elements): 1

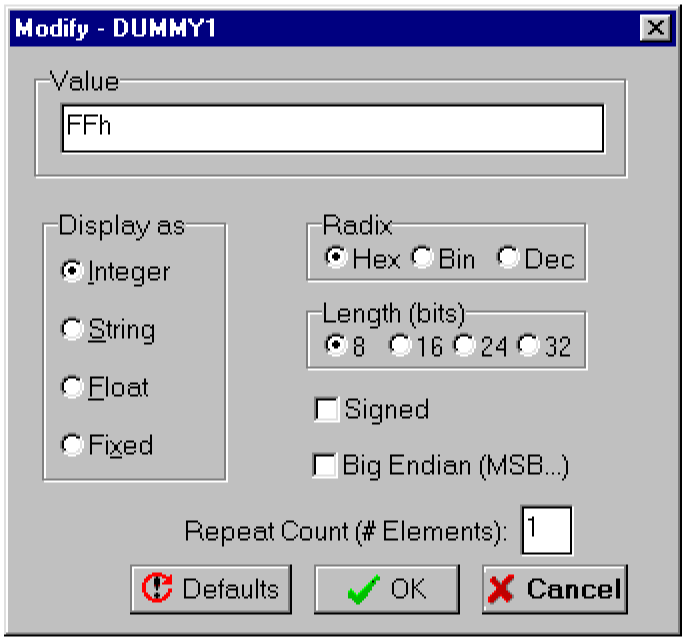
Integer Options:

Radix: - Select HEX, Binary or Decimal formatting.

Length: - Group the selected number of bits as a single value (consecutive).

Big Endian: - If the Length is greater than 8 bits, this option will display the MSB first when selected.

Repeat Count - Treat as an array of the above settings with the specified number of elements.


Modify - DUMMY1

Value: FFh

Display as: [✓ Integer] String Float Fixed

Radix: [✓ Hex] Bin Dec

Length (bits): [✓ 8] 16 24 32

☐ Signed

☐ Big Endian (MSB...)

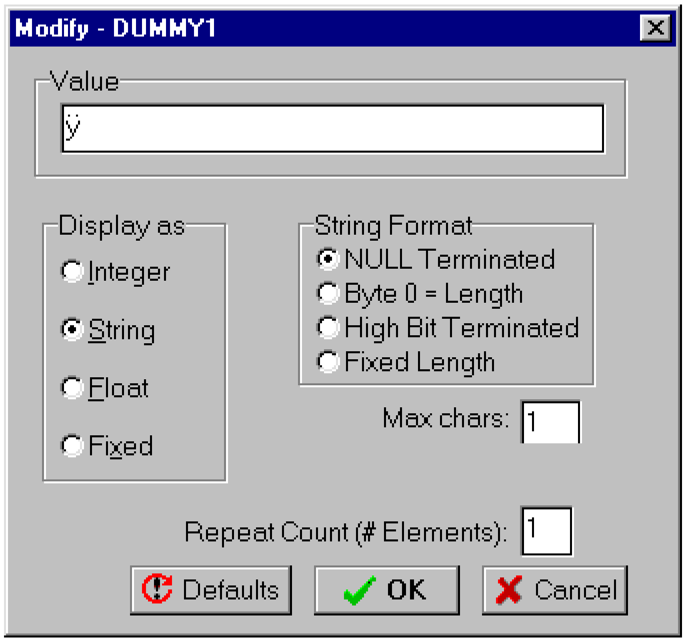
Repeat Count (# Elements): 1

String Options:

Format: - Select the desired string type.

Max Chars - Enter the maximum number of characters to associate with this string.

Repeat Count - Treat as an array of the above settings with the specified number of elements.


Modify - DUMMY1

Value: ÿ

Display as: Integer, [✓ String], Float, Fixed

String Format: [✓ NULL Terminated], Byte 0 = Length, High Bit Terminated, Fixed Length

Max chars: 1

Repeat Count (# Elements): 1

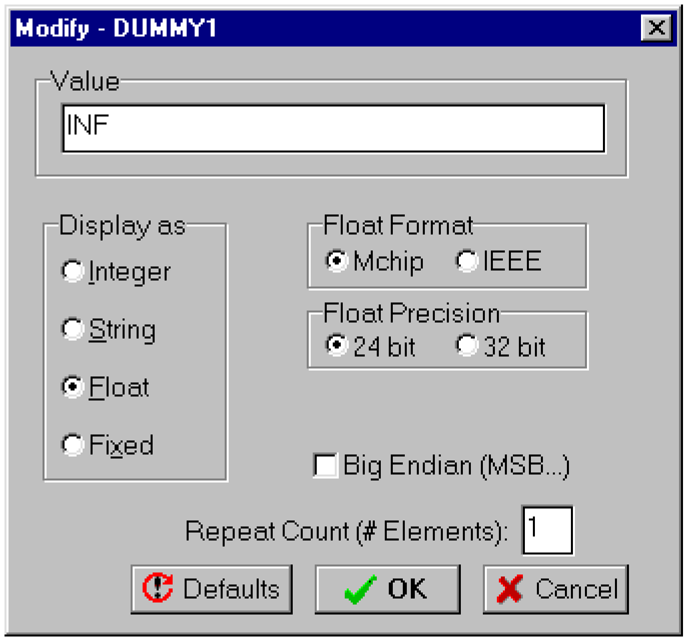
Float Options:

Format - Choose the desired float format.

Precision - cho ose from 24 or 32 bit precision.

Big Endian - If selected, value will be displayed MSB first.

Repeat Count - Treat as an array of the above settings with the specified number of elements.


Modify - DUMMY1

Value: INF

Display as: Integer, String, [✓ Float], Fixed

Float Format: [✓ Mchip] IEEE

Float Precision: [✓ 24 bit] 32 bit

Big Endian (MSB...) (unchecked)

Repeat Count (# Elements): 1

Fixed Options:

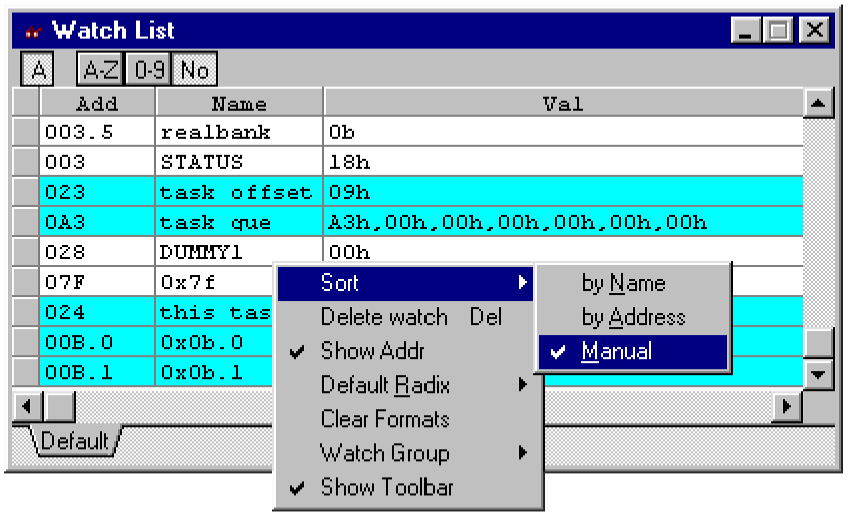
Format - Choose from 16.8 or 16.16 format.

Big Endian - If selected, value will be displayed MSB first.

Repeat Count - Treat as an array of the above settings with the specified number of elements.

3.2.3.7.4 Watch List Options

The Watch Window has several options for changing how items are displayed. All of these options are available from the popup menu. To access the popup menu, Right-Click in the Watch Window.



Add	Name	Val
003.5	realbank	0b
003	STATUS	18h
023	task offset	09h
0A3	task que	A3h,00h,00h,00h,00h,00h,00h
028	DUMMY1	00h
07F	0x7f	
024	this tas	
00B.0	0x0b.0	
00B.1	0x0b.1	

Sort

Items can be sorted by NAME, ADDRESS or No Sorting. These three options are also available as speed buttons on the toolbar (A-Z = Name Sort, 0-9 = Address Sort, No = No Sort). To sort manually, simply Drag & Drop the item to the desired position. This is accomplished by holding the mouse down on the column to the left of the item and then releasing the mouse when the cursor is at the desired destination.

Delete Watch

Deletes the currently selected watch item.

Show Address

Toggles displaying of the Address Column

Default Radix

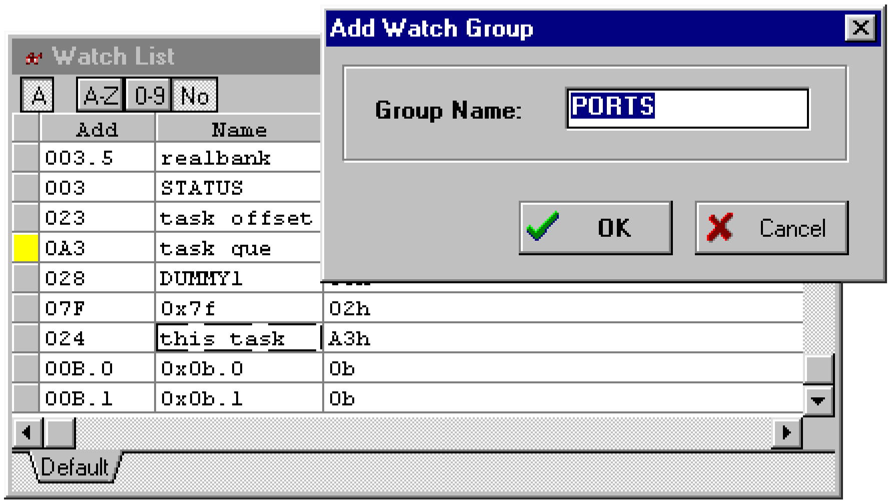
Sets the RADIX (HEX, Decimal or Binary) for all items that have not been customized (see: Modifying 112 Watch Item Values 112).

Clear Formats

Clears all special formatting on customized items (indicated by the color yellow in the first column) and sets them to default format (see: Modifying 112 Watch Item Values 112). The window may have to be closed and re-opened in order for the changes to take effect if the customized item is not already selected.

Watch Group

New groups can be added from this menu option or the current watchgroup can be deleted.



	Add	Name	
	003.5	realbank	
	003	STATUS	
	023	task offset	
🟨	0A3	task que	
	028	DUMMY1	
	07F	0x7f	02h
	024	this task	A3h
	00B.0	0x0b.0	0b
	00B.1	0x0b.1	0b

Group Name: PORTS

Show Toolbar

This option toggles the toolbar display.

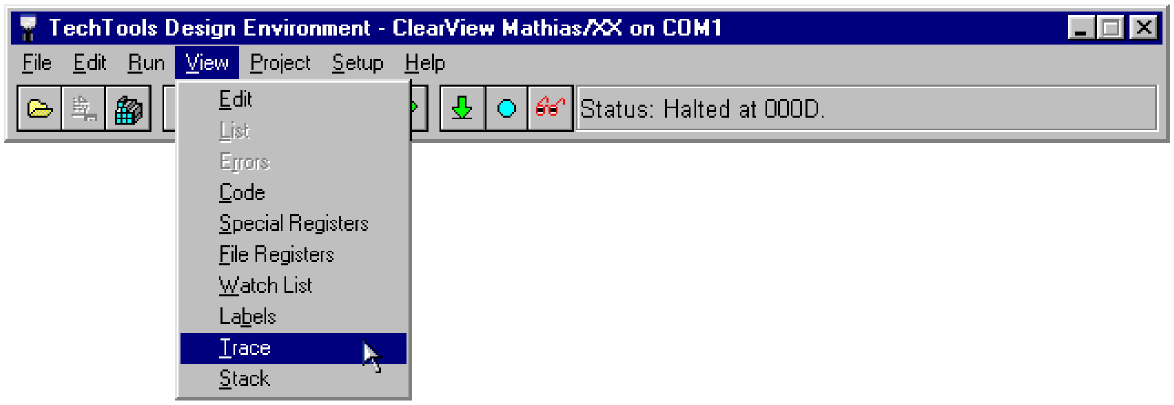
3.2.3.8 Trace Buffer Window

This window shows the contents of the trace buffer memory. It is only available if the optional trace buffer 60 module is installed.

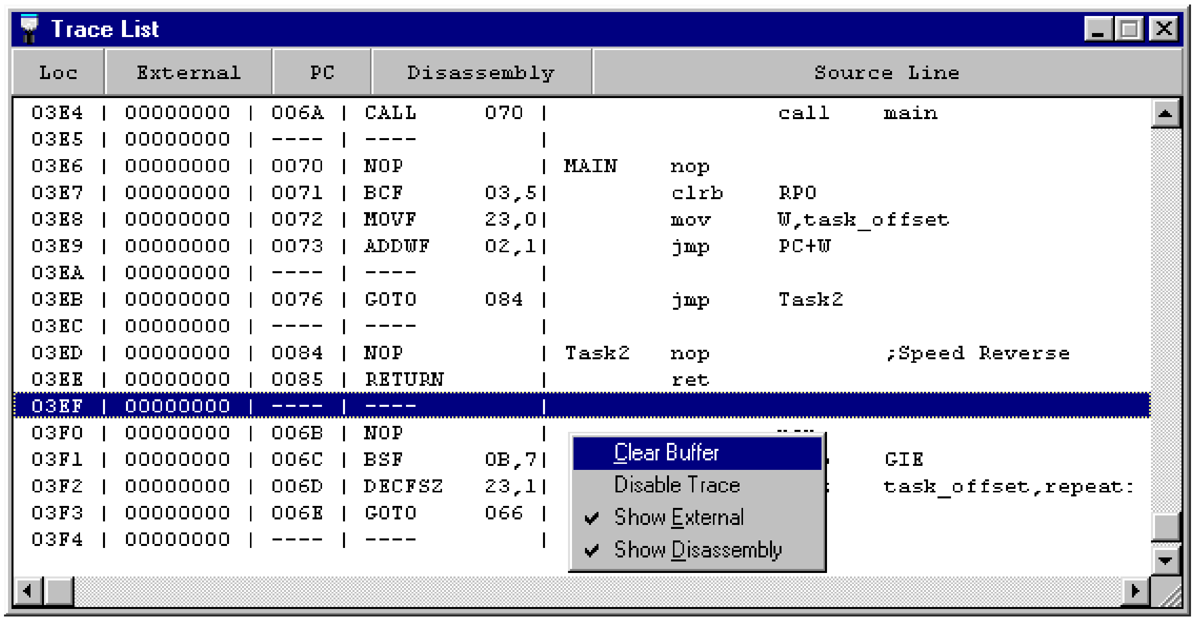
The trace buffer keeps a record of each instruction executed by the emulator. For each instruction, the record shows the locat ion in the trace buffer, the value of the program counter, the state of external trace inputs, and the original line of source code.

The trace buffer memory is 16K "deep," resulting in a capacity of approximately 16,384 Microchip instructions . Some instructions, such as JMP and GOTO, take two processor cycles, and therefore take two locations in the trace buffer memory. In addition, some TechTools instructions utilize two or three Microchip instructions, and will therefore take multiple locations in the trace buffer (this is also true of high-level compiled languages, such as C).

To see the trace buffer window, select Trace from the View menu.



The following window will appear:



```
 Loc    External      PC     Disassembly                    Source Line
03E4 | 00000000 | 006A | CALL      070 |                  call     main
03E5 | 00000000 | ---- | ----          |
03E6 | 00000000 | 0070 | NOP           | MAIN      nop
03E7 | 00000000 | 0071 | BCF      03,5 |           clrb     RP0
03E8 | 00000000 | 0072 | MOVF     23,0 |           mov      W,task_offset
03E9 | 00000000 | 0073 | ADDWF    02,1 |           jmp      PC+W
03EA | 00000000 | ---- | ----          |
03EB | 00000000 | 0076 | GOTO      084 |           jmp      Task2
03EC | 00000000 | ---- | ----          |
03ED | 00000000 | 0084 | NOP           | Task2     nop               ;Speed Reverse
03EE | 00000000 | 0085 | RETURN        |           ret
03EF | 00000000 | ---- | ----          |
03F0 | 00000000 | 006B | NOP           |
03F1 | 00000000 | 006C | BSF      0B,7 |                             GIE
03F2 | 00000000 | 006D | DECFSZ   23,1 |                             task_offset,repeat:
03F3 | 00000000 | 006E | GOTO      066 |
03F4 | 00000000 | ---- | ----          |
```

Like several other windows, the trace window has a pop-up menu; to see it, place the mouse over the trace window and then press the right mouse button:

The pop-up menu has four options; to select a function, just move the mouse to the desired option and then click the left mouse button. The options are described below:

Clear trace buffer

Select this option to erase the trace buffer.

Enable/disable trace

Select this option to enable or disable the trace function. If already enabled, then the pop-up menu will show "disable trace." If already disabled, then "enable trace" will be shown. The trace function can also be controlled from the Trace 60 tab in the Project option of the Setup menu , but the pop-up menu is usually much easier. If the trace function is disabled, t hen the window titles will be shown in italic letters.

Show Externals

This option toggles the External Input Display. An external input connector is located on each member module for attaching the trace cable (which has mini-clip connectors for attaching to your circuit). When selected, the eight external inputs are displayed in binary form.

Show Disassembly

This option toggles the Disassembly display. If selected, the trace window also displays the execution history in Microchip assembly format.

Note 1:
The trace window is only available if the trace buffer module is installed. For questions regarding the trace hardware, please see the end of the Hardware Installation section.
Note 2:
The trace buffer memory is arran ged as a FIFO (first-in first-out) buffer. After 16K of history is recorded, the oldest records are lost to make room for new records.

3.2.4 Data Breakpoints

Data Breakpointing is a very productive and timesaving feature available with the Advanced XX Family Module.

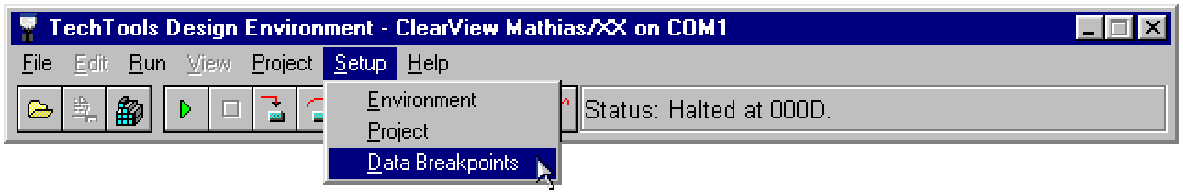
Sometimes a simple execution breakpoint and Watch window do not yield enough information to determine which line of code is changing a variable. It could take hours or even days to find the mistake.

TDE's Data Breakpoints will help you find this kind of problem almost immediately.

This section covers the following details:

Enable and Break On options 120
File Register options 120
Match On options 121
Include Constants option 123
Breaking on Data 124

To open the Data Breakpointing Dialog, select Data Breakpoints from the Setup menu.

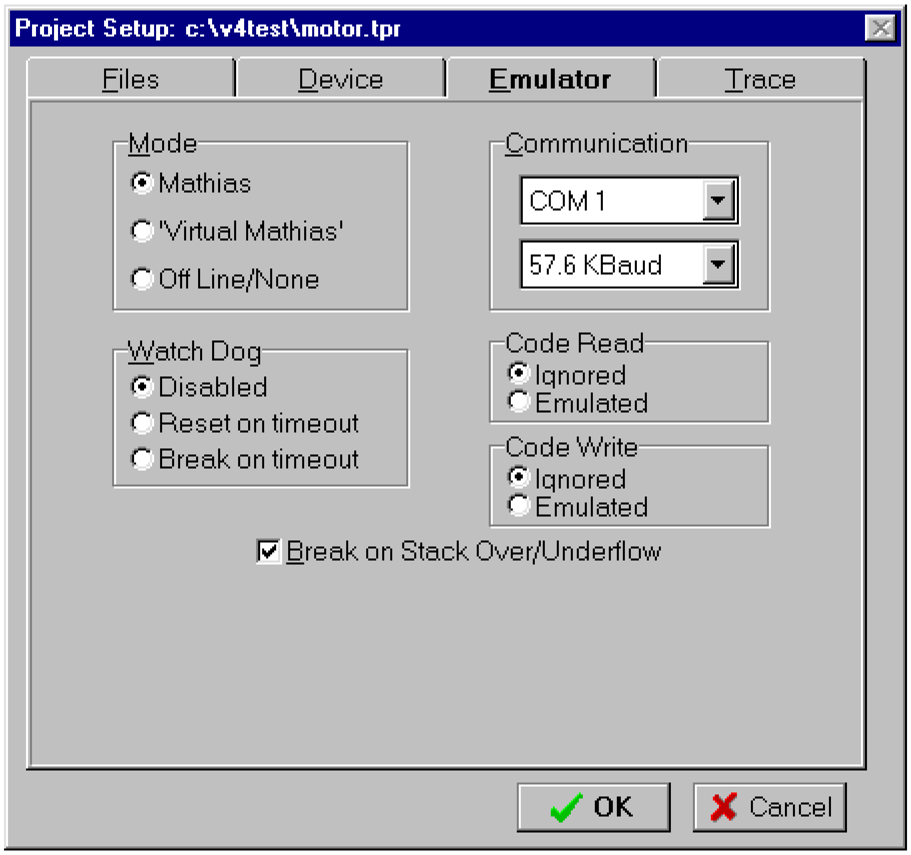


Status: Halted at 000D.

NOTICE:

Some MCUs have a "Self Modifying Code" feature. This option must be disabled in order to enable Data Breakpoints.

Disable the self modifying features by selecting "Setup - Project - Emulator", then selecting "Ignored" for the Code Read and Code Write options.



Project Setup: c:\v4test\motor.tpr

Mode: [✓ Mathias] 'Virtual Mathias' Off Line/None

Communication: COM 1; 57.6 KBaud

Watch Dog: [✓ Disabled] Reset on timeout Break on timeout

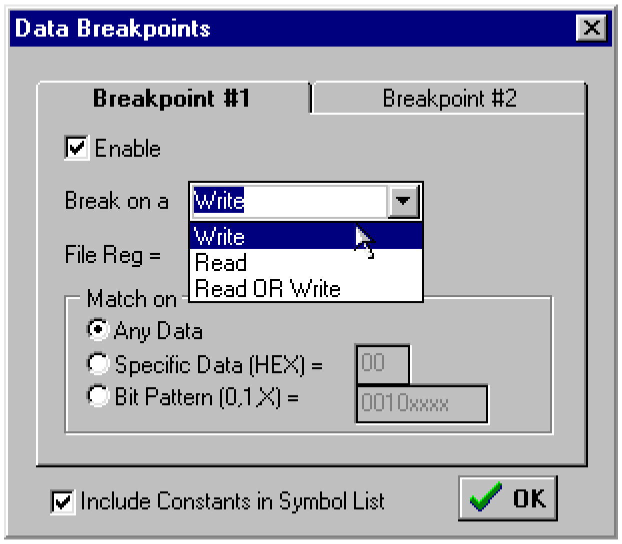
Code Read: [✓ Ignored] Emulated

Code Write: [✓ Ignored] Emulated

[✓] Break on Stack Over/Underflow

3.2.4.1 DBP Enable and Break On options

After Selecting Data Breakpoints from the Setup menu, a dialog similar to the one below should appear. If the options are not available, then you may not have the Advanced Family Module or you need to disable the Self Modifying Code Features (see: Data Breakpoints 118).


Data Breakpoints

Breakpoint #1 | Breakpoint #2

☑ Enable

Break on a: Write

Write / Read / Read OR Write

File Reg =

Match on:
● Any Data
○ Specific Data (HEX) = 00
○ Bit Pattern (0,1,X) = 0010xxxx

☑ Include Constants in Symbol List

OK

Enable:

Disables or Enables a specific Data Breakpoint.

Break On:

Select the type of activity to monitor:

Read - Break If a match occurs when your program reads the register.

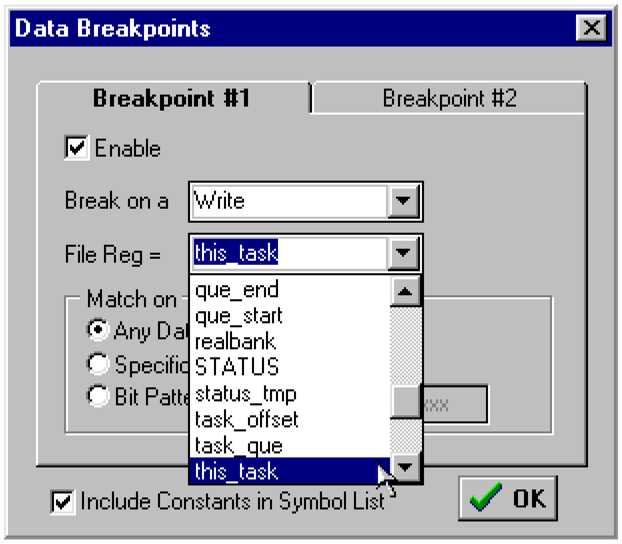
Write - Break If a match occurs when your program writes to the register.

Read or Write - Break If a match occurs when your program writes to OR reads from the register.

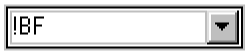
3.2.4.2 DBP File Register options

The "File Register" option refers to any Special Register 97 or General Purpose Register 105 (also see: Variables Window 99) your code has access to. The data at this location will be the focus of the criteria in the Match On 121 option.

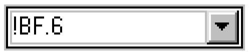
Select a predefined symbol from the drop down list as shown here, or enter an absolute address as shown in the examples below :



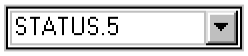
Monitor the data at the
absolute HEX address BF:


!BF

Monitor bit 6 of the
absolute HEX address BF:


!BF.6

Monitor bit 5 of the
STATUS register:


STATUS.5

Notice: An ' ! ' (exclamation point) or ' 0x ' should be used prior to the HEX value to indicate an Absolute address.
This notation is also used in the Add Watch Dialog 109.

3.2.4.3 DBP Match On options

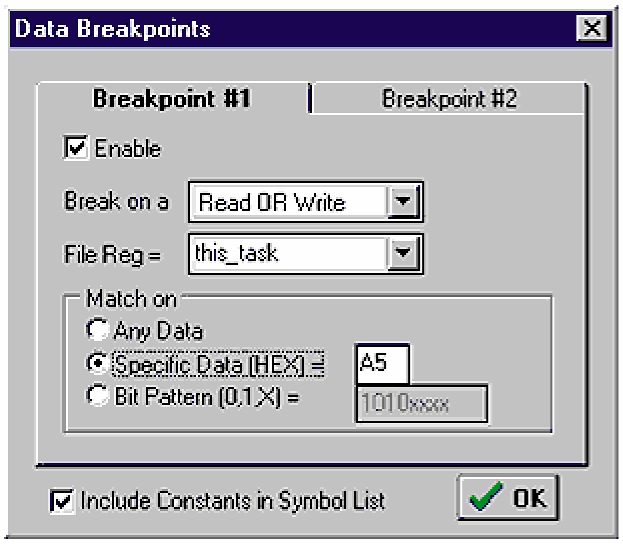
Select one of the three options to specify the data "Match" to break on.

Any Data:

When this option is selected, the data value is ignored. A Match is determined by the Break On 120 selection, without any further qualifiers.

Specific Data:

This option will qualify the breakpoint with a specific data value. When this option is selected, a break will occur ONLY if the data matches the specified value exactly.



[✓] Enable
Break on a: Read OR Write
File Reg = this_task
Match on
Any Data
[✓ Specific Data (HEX) =] A5
Bit Pattern (0,1,X) = 1010xxxx
[✓] Include Constants in Symbol List

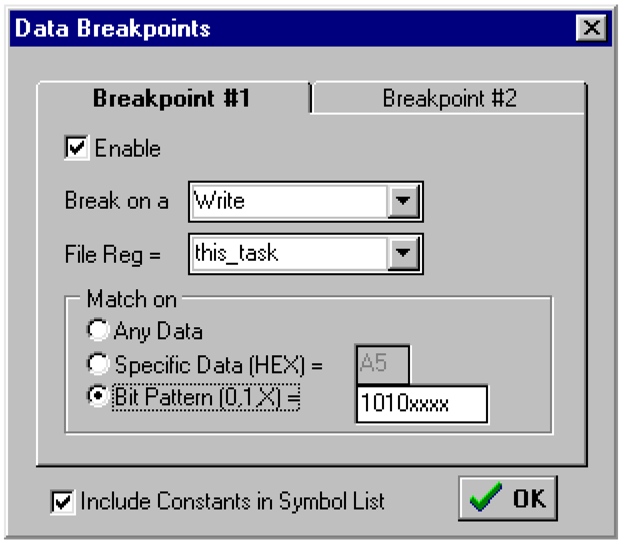
Bit Pattern:

This option allows you to define specific bits which should be ignored during the match evaluation. Each bit has three possible states; on, off and don't care.

1 = ON

0 = OFF

X = Don't Care



[✓] Enable

Break on a: Write

File Reg = this_task

Match on
Any Data
Specific Data (HEX) = A5
[✓ Bit Pattern (0,1,X) =] 1010xxxx

[✓] Include Constants in Symbol List

In the example on the above, we selected specific values for the bit positions in the upper nibble, and don't car e for each bit in the lower nibble. Using this bit pattern as a qualifier, a break will occur whenever your program writes any of the following HEX values to the "this_task" register:

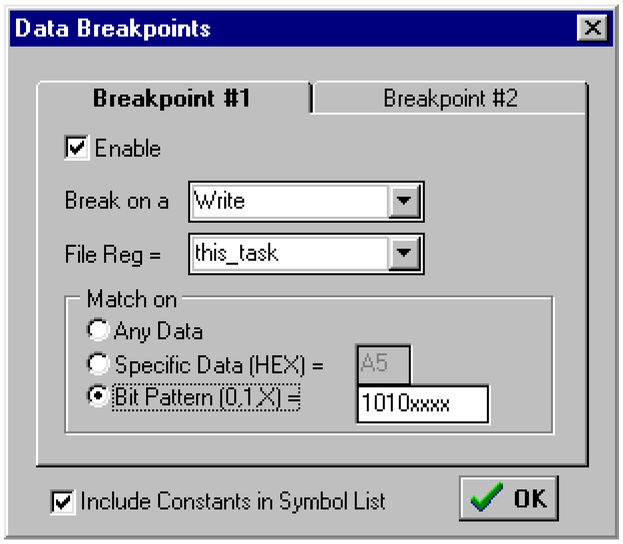
A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, AA, AB, AC, AD, AE, AF

Or in binary; any value from:

10100000 through 10101111.

Any combination of 1's, 0's and X's is valid.

3.2.4.4 DBP Include Constants


Data Breakpoints

Breakpoint #1 | Breakpoint #2

[✓] Enable

Break on a: Write

File Reg = this_task

Match on
- ( ) Any Data
- ( ) Specific Data (HEX) = A5
- (●) Bit Pattern (0,1,X) = 1010xxxx

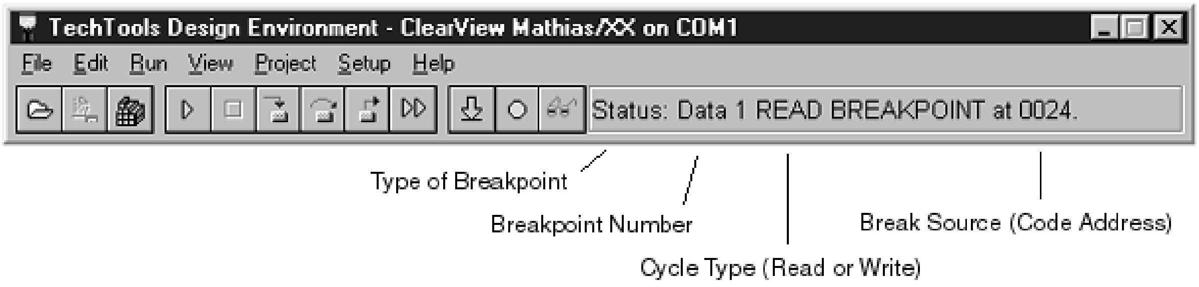
[✓] Include Constants in Symbol List

OK

When this opti on is not selected, TDE will filter out all data types defined as Constant by the compiler. Note that some compilers define all symbols as constants, and selecting this option forces TDE to display Constant Data Types in the File Register list 120.

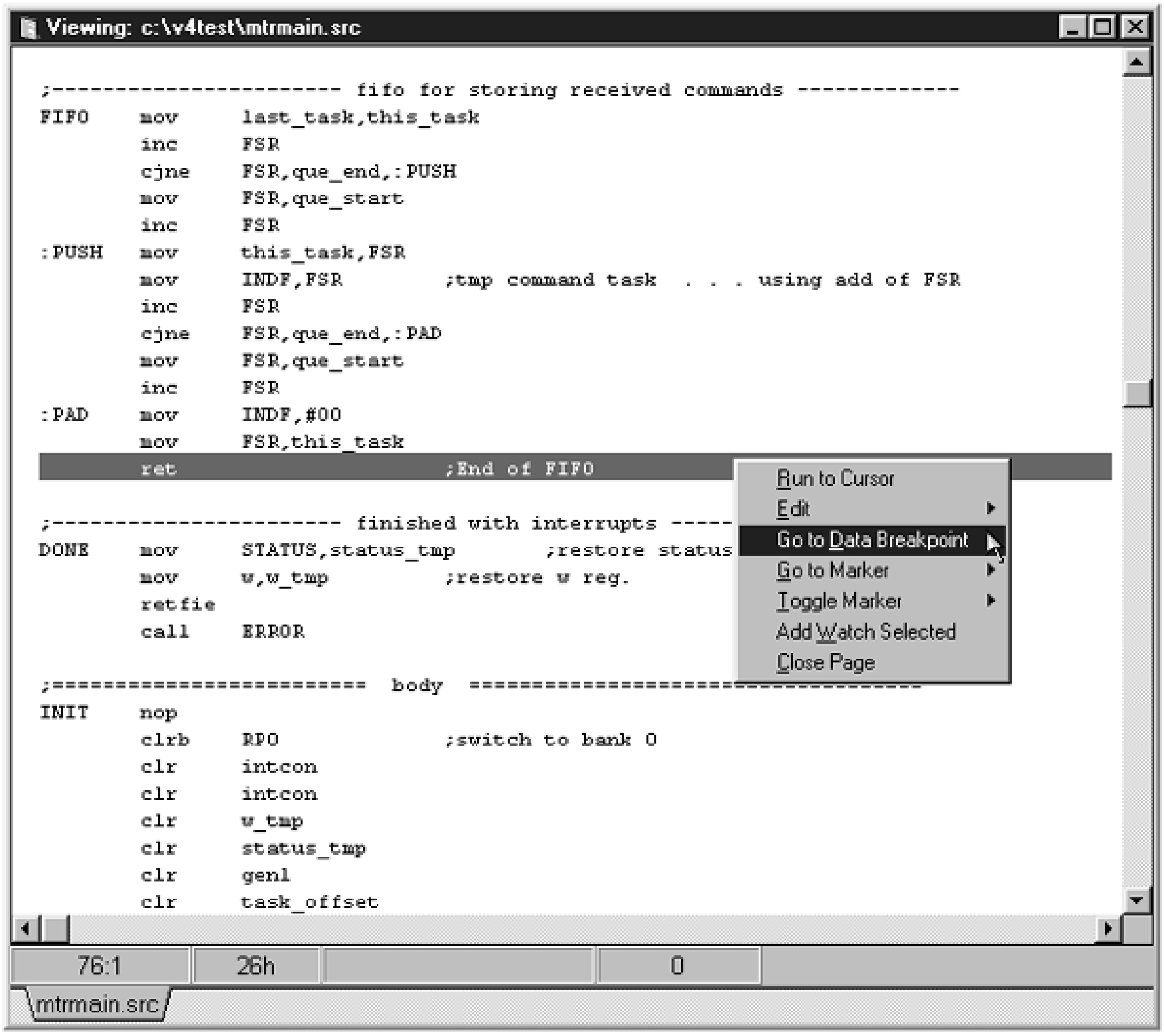
3.2.4.5 Breaking on data

When TDE encounters a valid Data Breakpoint 118, execution will be halted and the display line updated to reflect the breakpoint details as shown below.


TechTools Design Environment - ClearView Mathias/XX on COM1
File Edit Run View Project Setup Help
Type of Breakpoint
Breakpoint Number
Cycle Type (Read or Write)
Break Source (Code Address)

Status: Data 1 READ BREAKPOINT at 0024.

To view the actual code line responsible for the Data breakpoint, right-click in the editor window 82 and select Go to Data Breakpoint. This will move the cursor to the correct code line. If the code line is located in another file, TDE will bring that file into view before moving the cursor. The correct code line is also identified in the status bar as shown above.



```
;----------------------- fifo for storing received commands -------------
FIFO    mov     last_task,this_task
        inc     FSR
        cjne    FSR,que_end,:PUSH
        mov     FSR,que_start
        inc     FSR
:PUSH   mov     this_task,FSR
        mov     INDF,FSR        ;tmp command task  . . . using add of FSR
        inc     FSR
        cjne    FSR,que_end,:PAD
        mov     FSR,que_start
        inc     FSR
:PAD    mov     INDF,#00
        mov     FSR,this_task
        ret                     ;End of FIFO

;----------------------- finished with interrupts -----
DONE    mov     STATUS,status_tmp       ;restore status
        mov     w,w_tmp         ;restore w reg.
        retfie
        call    ERROR

;========================== body ====================================
INIT    nop
        clrb    RP0             ;switch to bank 0
        clr     intcon
        clr     intcon
        clr     w_tmp
        clr     status_tmp
        clr     genl
        clr     task_offset
```

Due to the PIC's pipeline architecture, one to two instructions will be executed after the data breakpoint occurs.

3.3 Compiler Integration Notes

This section contains Hints on integrating popular compilers and assemblers with TDE.

How We Do It 126
CVASM16 127
SPASM 127
MPASM 127
MPLINK 128
Bytecraft C 128
Bytecraft Linker 128

MPLABC 129
Hi-Tech C 129
CCS - PCB and PCM 130
COD Only 131
HEX Only 131

This information is included to give you some insight as to HOW we integrate with each of these compilers, any assumptions we make and any common installation mistakes we are aware of.

Most TOOLS strongly recommend that you include their directory in your path. If you forgot to (or chose not to) set your system PATH to include the compiler directory, you may optionally choose to specify the
path in the TOOL PATH option box in the project NODE setup. This MAY not work with some TOOLS.

NOTE:
We include CVASM16 with TDE.
We also include SUPPORT for all of these other compilers and assemblers. HOWEVER, WE DO NOT SUPPLY THESE THIRD PARTY TOOLS WITH TDE. YOU MUST OWN THESE OTHER TOOLS IN ORDER TO USE THEM.
This may seem obvious but we have had MANY tech-support calls asking why a particular compiler is not working. Eventually we find that the customer thought that we were somehow providing the functionality of all of these commercial compilers with our own software.

3.3.1 How we do it

In general, we spawn the third-party compiler or assembler in a separate console window and capture its output. We wait for it to complete and then

analyze the resultant files to determine if it was successful. If errors are found, we display them and show you the source line that caused the error.

If the TOOl was successful, we read the resulting symbol table, line references and HEX data into an internal database for debugging.

While we are building the project, you will see a console ICON on the task bar. If the project build seems to stall, click on this ICON and you will see the state of the project build. In a few cases, some tools will pause and ask you to hit a key before it continues. This causes TDE to appear to 'lock-up'. In reality, TDE is just waiting on the project build process to complete. The process is waiting on you to press a key but its window is minimized so you never see the message. Unfortunately, not all compilers and assemblers can be forced to ALWAYS log errors rather than pause for input from you. The good news is that

this situation is very rare. It usually only happens if the compiler has a critical error.

The following sections describe each compiler we currently support. If you are using a TOOL that we do not list here, let us know. TOOLs that generate standard symbol formats are easy to integrate.

NOTE:
TDE is preconfigured to work with these compilers. You should not have to make any concessions or exceptions to the normal compiler installation or configuration to operate with TDE. In other words, install and configure your compiler as directed by its manufacture and it should operate with TDE.

3.3.2 CVASM16

CVASM16 is include with the TDE installation. It should operate properly as installed. TDE expects CVASM16.EXE to reside in the same directory as TDE.exe. PLEASE LEAVE IT THERE.

CVASM16 is a SINGLE SOURCE FILE type tool. You can NOT specify multiple source file NODEs in the project. However, you can 'include' as many files as you wish into the main source file. This does not reduce your ability to debug. You can set breakpoints and single step through included files exactly the same as the main file.

A typical CVASM project would specify CVASM16 as the TOOL in the PROJECT node. A single file NODE would be added to indicate the main source file. Its TOOL would be set to NONE. No path or options are required.

NOTE: Beginning with the release of CVASM 6.0, an additional Project Tool selection has been added to distinguish between 6.0 and earlier versions of CVASM. To assemble source code written for pre 6.0 CVASM or any version of SPASM, select the CVASM 5.x Project Tool. This will call CVASM 5.8 which is included with the installation named as "CVASM_.EXE". You will find this file in TDE's directory ("C:\PicTools" by default).

3.3.3 SPASM

Starting with release 4.0, TDE will invoke CVASM16 when you select SPASM. CVASM16 is fully source code compatible with SPASM. In addition, CVASM16 supports newer PIC chips and generates more complete symbolic information for the debugger. This additional information is critical to proper debugging. See the CVASM16 127 section for details.

3.3.4 MPASM

MPASM can operate in two different modes. It can assemble a SINGLE SOURCE FILE to completion, generating an executable code image and debugging information. Alternatively, MPASM can also be used to create an OBJECT file suitable for LINKING with other modules. Each approach is discussed below.

SINGLE FILE MODE:

This is the traditional mode, similar to CVASM16. A typical project would specify MPASM as the TOOL in the PROJECT node. A single file NODE would be added to indicate the main source file. Its TOOL would be NONE.

COMPILE TO OBJECT:

This mode compiles a source NODE to an OBJECT module. This is used in conjunction with MPLINK 128 to produce a complete image file. In this mode, MPASM-to-OBJ is used as the TOOL on each SOURCE NODE to convert source files to OBJECT files. MPLINK is specified as the TOOL for the PROJECT NODE.

3.3.5 MPLINK

MPLINK is used to combine multiple OBJECT files to a single executable file. Microchip assemblers can generate object files. MPLINK links these together. Set the PROJECT NODE's TOOL to MPLINK. MPASM to OBJ, can be used as TOOLS for each source file NODE. TDE will compile each source file node to objects, then run the PROJECT node tool(MPLINK) to link them together.

3.3.6 Bytecraft C

The Bytecraft C compiler can operate in two modes. It can compile a single source file to a complete executable image, or it can generate OBJECT modules, suitable linking with the Bytecraft LINKER 128. Both modes are discussed below.

SINGLE FILE MODE:

In this approach, you would specify the Bytecraft C compiler as the tool selection for the project node. You would them add a single source file NODE to the project to indicate which file should be compiled. This NODE would have its TOOL selection set to NONE. You can INCLUDE additional files IN THE SOURCE file if you wish.

COMPILE TO OBJECT MODE:

This mode allows you to specify several source files in the project. You would select the Bytecraft C to obj TOOL for each source file NODE. You would then select the Bytecraft LINKER for the PROJECT NODE. This setup compiles each source file to an object file, and then uses the linker to combine them into an executable image.

A simpler approach to working with a multi-file project is to set up a single file project as discussed above. Then 'INCLUDE' multiple files into a single source file. This does not limit your ability to debug in any way.

It may affect the way you use GLOBALS and STATICS but eliminates the need for a LINKER file.

3.3.7 Bytecraft LINKER

When working with multiple source file projects, you must select the Bytecraft LINKER for the PROJECT NODE.

You must also create a LINKER file to tell the linker which header files the source files used, and which object files it should link. Failure to define a proper linker file will result in the linker failing to complete. An incorrect LINKER file may generate seemingly unrelated error messages or cause the build to stall, making TDE appear to hang. If the build seems to be taking too long, click on the console ICON on the task bar to see what the compiler is doing. It may be waiting on you to acknowledge that you have read a critical error message.

A typical LINKER file follows:

```
compiler = mpc.exe
#include <16c74.h>
#include <math.h>
OBJECT = file1, file2
OBJECT = file3, file4
```

When running under TDE, the LINKER file MUST have a '.LNK' extension. You also MUST type the link file NAME, WITHOUT EXTENSION in the OPTIONs box of the PROJECT NODE. For example, you might create a LINKER file called myprj.LNK. You would then type "myprj"

(without the quotes..of course) in the OPTIONs box of the PROJECT NODE.

HINT:
You can use TDE's FILE/OPEN to create and maintain this file just like any other file in your project.

3.3.8 MPLABC

MPLABC is a SINGLE SOURCE FILE type tool. You can NOT specify multiple source file NODEs in the project. However, you can 'include' as many files as you wish into the main source file. This does not reduce your ability to debug. You can set breakpoints and single step through included files exactly the same as the main file.

A typical MPLABC project would specify MPLABC as the TOOL in the PROJECT NODE. A single file NODE would be added to indicate the main source file. Its TOOL would be set to NONE. No path or options are required.

3.3.9 Hi-Tech C

The Hi-Tech C compiler can be used in several modes. If you are compiling a single file, you can set up a simple project that compiles and links in a single step. In this case, you would specify Hi-Tech C for the PROJECT NODE TOOL. Then ADD your source file and specify 'NONE' for its tool.

If your project consists of several files, you have two choices.

The first choice is to specify multiple source files, each with a TOOL setting of 'NONE'. You can also add LIBRARY or OBJECT files to the project, again specifying tool settings of 'NONE'. This causes all source files to be passed to the PROJECT NODE TOOL. The PROJECT NODE TOOL (Hi-Tech C) would then compile and link them all together.

The second approach is to specify the "Hi-Tech C to OBJ" tool for each SOURCE FILE NODE. This approach causes the Hi-Tech C compiler to be invoked for EACH source file. The resulting OBJECT files are then

LINKED together by the Hi-Tech C compiler TOOL specified on the PROJECT NODE.

You can even mix the two approaches; compiling some of the source NODEs directly and others at LINK time. What makes this all work is the fact that the Hi-Tech C compiler will accept a mixture of source, library and OBJECT files. It compiles what needs compiled and then links them all together.

3.3.10 CCS

CCS - PCB (Base: 12 bit compiler)

CCS - PCM (Mid-range: 14 bit compiler)

These are SINGLE SOURCE FILE type tools. You can NOT specify multiple source file NODEs in the project. However, you can 'include' multiple files into the main source file. This does not reduce your ability

to debug. You can set breakpoints and single-step through included files exactly the same as the main file.

A typical project would specify PCB or PCM as the TOOL in the PROJECT node (depending on whether your are compiler for a 12bit or a 14bit device). A single file NODE would be added to indicate the main source file. Its TOOL would be set to NONE. No path or options are required unless you failed to set your system path during installation of the compiler. If your system path is not set, you can specify the PATH

to the compiler executable in the TOOL PATH option box.

We can not force PCB or PCM to generate COD format debugging information with command-line parameters. Instead, the compiler looks for this information in a file called PC.DEF in the project's local directory each time it is invoked. Each time we start a BUILD, we copy a PC.DEF file from the TDE install directory into your project directory, insuring that the compiler is properly instructed to generate the necessary information for debugging. If CCS changes the format of this file in the future, simply replace the one in the TDE directory with an updated version. TDE will then use the new PC.DEF file for all future builds.

3.3.11 COD ONLY

This tool allows full debugging from a COD file. This is useful if you only have the COD file or if you wish to do your compiling from some other environment and then use TDE for debugging. Some prefer to use their own MAKE programs or IDEs.

If your source files are available when the COD file is read in, TDE will do full source-level debugging. You will be able to step through the code, set breakpoints on source lines,..etc. To setup this work flow, simply select the COD ONLY tool for the PROJECT NODE and add a single source file node. The source file node should be the COD file. Set its tool to NONE. Now, each time TDE 'rebuilds' the project, he will copy your COD file to his own internal databases and begin debugging.

3.3.12 HEX ONLY

This TOOL allows you to do debugging, even if you have nothing more than a HEX file. Of course, HEX files contain NO DEBUGGING information, so TDE will not be able to show you symbols or let you single-step through source code. It WILL allow you to watch registers, single-step through CODE and set breakpoints in the CODE window (a disassembled dump of the code space).

To use this workflow, specify 'HEX ONLY' as the tool for the PROJECT NODE. ADD your HEX file to the project and specify 'NONE' for its TOOL.

Whenever you update the HEX file, simply press TDE's BUILD button or press F4 or select the BUILD menu item and TDE will update Mathias.

ClearView Assembler

Part IV

4 ClearView Assembler

If you already know how to use an assembler, you'll probably find our's to be quite simple. However, if you have little or no experience with assemblers, you should read the following pages before attempting to use the assembler.

Assembler Reference 134
Instruction Set 144
Sample Projects 186

When you write programs for PIC microcontrollers, you will use a text editor to create source code. Source code is the format that you're accustomed to looking at; it contains English-like labels, instructions, and data.

Before your source code can be used by a programmer, emulator, or other development tool, it must be converted into hex code. Hex code is the "machine-readable" version of source code; it contains instructions and data in the form of hexadecimal data, which can be executed by the PIC.

An assembler is a piece of software that converts source code into hex code. For instance, this line of source code:

```
CALL        SENDBYTE     ;Call send routine
```

assembles into just two bytes of hex code:

```
2420h
```

It's possible to write programs directly in hex code, using the individual machine codes that make up each instruction. However, most people find it preferable to use an assembler.

The TechTools assembler (CVASM16) is unique in its ability to accept two instruction sets: our own 8051-like instruction set, and the original Microchip instruction set. Many customers appreciate the TechTools instruction set, because it resembles other processors. However, there are certainly customers who prefer the Microchip instructions.

If you plan to use source code in the Microchip format, please note that while our assembler will accept their basic instructions, it will not accept various other aspects of their source code. The syntax of numbers is sometimes different, and our assembler does not recognize Microchip assembler directives and macros. If you plan to write a lot of code in the Microchip format, then you may wish to use the Microchip assembler (MPASM).

4.1 CVASM16 Reference

Software Installation 134
Running the Assembler 134
Generating Assembly Listings 135
Command-Line Options 135
Assembler Basics 135
Addressing Definitions 136
Data Types 136
Expressions 138
Symbols & Labels 138
Local Labels 139
Default Symbol Tables 139
Comments 140
Assembler Directives 140
Source Code Formatting 142
CVASM16 Error Messages 142

4.1.1 Software Installation

If you installed our TDE or PICwriter software, CVASM was automatically installed into the TDE directory (PicTools), and you can skip this step.

Insert the CD in your CDROM drive. If you have "auto run" enabled, the CD menu will launch automatically, or run the "Tech_cd.exe" program to launch the CD menu. After the CD menu has started, select the "PICTools" option , this will install TDE, the assembler and Include files, PICwriter software and create a Program Group called "PIC Tools". The CVASM16 assembler is a DOS executable which requires command-line parameters and can be executed by opening a DOS prompt and typing CVASM16 or selecting CVASM as your Project Tool in TDE.

4.1.2 Running the Assembler

To assemble your source code into a hex file, type the following command at the DOS prompt:

```
CVASM16  filename       Assembles text file filename.src into hex file filename.obj.
```

As shown above, only the name of the source file is given. The assembler uses the same name for the hex file, but replaces the extension with .obj. If you do not specify an extension as part of the source file name, the assembler will assume that the extension is .src.

An example is shown below:

```
CVASM16  EXAMPLE
```

The assembler would produce a hex file called EXAMPLE.OBJ from the source file called EXAMPLE.SRC.

4.1.3 Generating Assembly Listings

It is possible to have the assembler create an "assembly listing" of your program. An assembly listing is a duplicate of the source code, but with hex code information (line number, address, opcode, & data) preceding each original line. To have the assembler create a listing file, simply add "/L" after the filename:

```
CVASM16  filename /L
```

Assembles text file filename.src into hex file filename.obj and creates a listing file called filename.lst.

4.1.4 Command-Line Options

The PIC assembler has several options which can be invoked when it is run. These command-line options are shown below:

```
CVASM16  filename
```

Assembles text file file name.src into hex file filename.obj.

```
CVASM16  filename.xxx
```

Assembles text file filename.xxx into hex file filename.obj.

```
CVASM16  filename /L
```

Assembles text file filename.src into hex file filename.obj and creates a listing file called filename.lst.

```
CVASM16  filename /M
```

Assembles text file filename.src into hex file filename.hex. This option will output a Microchip compatible HEX file suitable for Microchip tools and third-party programmers.

```
CVASM16  filename /S
```

Assembles text file filename.src into hex file filename.obj but suppresses additional information. This option is use ful for other tools that won’t accept any Device or Fuse information embedded in the hex file.

4.1.5 Assembler Basics

The purpose of the assembler is to convert assembly language source code into hex code. The assembler accomplishes its task in two passes:

Pass 0 - The source code is scanned in an attempt to resolve all symbols. This is possible if all origin, define space, and equate directives can be resolved (equated symbols may be referenced by origin or define space). All other symbols can be resolved by byte-offsets which are determined by the mnemonic/operand combinations. If Pass 0 is successful, the assembler will advance to Pass 1. If Pass 0 is unsuccessful, a list of errors will be shown and assembly will be aborted.

Pass 1 - The source code is scanned once more in order to assemble the hex code. Since all symbols were resolved in Pass 0, all instructions and miscellaneous directives can be fully resolved in Pass 1. If Pass 1 is successful, a hex file containing the assembled code will be created. If Pass 1 is not successful, a list of errors will be shown and assembly will be aborted.

In addition to a hex file, the assembler can be used to generate an assembly listing. An assembly listing shows line numbers, equated values, addresses, data, and original source code (for more information, see the section Generating Assembly Listings 135).

4.1.6 Addressing Definitions

Throughout your programs, you'll refer to bits and bytes by their addresses. Depending on the instruction being used and the item being referred to, the address will be given in one of the following forms:

addr8	An 8-bit address (on 16C5x devices, the lower half of the current 512-word page).
addr9	A 9-bit address (on 16Cxx devices, within the current 512-word page).
addr11	An 11-bit address (anywhere in program memory in a device with 2K of memory).
addr12	A 12-bit address (anywhere in program memory in a device with 4K of memory).
addr13	A 13-bit address (anywhere in program memory in a device with 8K of memory).
bit	An address for bitwise operations. Example: PortC.3 = bit 3 of port C
fr	A file register (RAM) address.
rel	A relative address ranging from -7Fh to +80h.
literal	An immediate 8-bit value.

4.1.7 Data Types

Eight data types are allowed in the assembler. These data types are:

· Symbol/label · Binary value

· Local symbol/label · ASCII value

· Decimal value · Assembly address (origin)

· Hex value · EEPROM address (origin)

The examples below show various data types:

100	Decimal value 100
18h	Hex value 18
0A7h	Hex value A7
1011b	Binary value 1011
'A'	ASCII value for the letter A (65 decimal)
Start	Label called Start
:loop	Local label called :loop
$	Current assembly address (Program Counter)
%	Current EEPROM address

In addition to single-character text, entire strings can be generated using the RETW instruction:

```
retw                'The fox jumped over the lazy dog'
```

NOTE: The above example will assemble into multiple RETLW instru ctions. This instruction can be used as a "Look-up" table by moving a value into 'W', then doing a 'CALL' to the beginning of the table.

The table should begin with a JMP PC+W (at this point, W should hold the intended offset into the string of cha racters). It is recommended to begin the table at a '00' address (such as 100h, 200h, 300h, etc.) to avoid jumping to an unexpected address. PC+W is a calculated jump which is limited to an 8 bit calculation. If the calculated result is greater than 256 then the value in PCL will be rolled over causing execution to begin at an unexpected address.

EXAMPLE:

```
  org 00h
GETCHAR
  ;
  MOV           W,myoffset       ; RAM location 'myoffset' is the 'index' of the desired
character.
  CALL          TABLE1           ; calls code at label 'TABLE1'
  MOV           achar,W          ; stores the character in RAM location 'achar'
  ;
  ;

  org 200h
TABLE1
  JMP           PC+W
  RETW  'The fox jumped over the lazy dog'
```

4.1.8 Expressions

Mathematical expressions are used in many instructions. These expressions may be created using the following operators:

&	Logical AND	/	Divide
|	Logical OR	<<	Shift left
^	Exclusive OR	>>	Shift right
+	Add	<	High byte
-	Subtract	>	Low byte
*	Multiply	.	Bit address

Some example expressions:

```
setb     PortA.0          ;Set bit 0 on port A.

mov      Count+3,#88h             ;Store 88h in location   Count+3.

ds       N*2                      ;Define empty space of Nx2 bytes.
```

All expressions are resolved strictly from left to right. Please make note of th is, since it may affect the result of expressions. For instance, the expression 5+2*4 would normally be resolved as (2*4)+5, for a result of 13. Since our assemblers resolve expressions strictly from left to right, however, the example would be resolved as (5+2)*4, for a result of 28.

4.1.9 Symbols & Labels

Symbols are used to name locations and values within your program.

Many people refer to address symbols as "labels", but both have the same effect. For instance, by assigning a symbol to the start of an important routine, you can later call that routine by its name, rather than its address. And by giving a symbol to a common value, you can refer to it by its name. Rather than typing "212" many times in your program, you can type HOT = 212 at the beginning of your program, and then use HOT wherever you need it. This is also quite handy if your idea of what's "hot" changes. By changing the symbol definition, you can easily redefine HOT to be a different temperature.

Symbols may be up to 32 character long. They must begin with a letter or underscore (_)

and must contain only letters, numbers, underscores, and colons. Further, if you're labeling an address (such as the start of a routine), the label must start at the beginning of the line.

Here are some examples of valid symbols:

```
min_count          =      20h
maximum_count      =      21h


begin       mov    min_count,#05h
```

4.1.10 Local Labels

By default, labels are global, which means that they can be "seen" from anywhere in your program. Sometimes, however, you may want to use a local label, which can only be "seen" within a limited part of your program (the area in which the local labe l can be seen starts at the preceding global label, and continues up to the following global label).

Local labels have the same syntax rules as global labels, except they must begin with a colon (:), and must be referenced with a colon. Local labels can be referenced from outside their normal area by referring to preceding global label name:local label name.

The following code demonstrates how to use the local label :loop for common looping purposes within two globally-labeled routines.

```
Routine1    mov    count,#100      (global label Routine1)
:loop       call   send_a (local label :loop)
            djnz   count,:loop     (jump to line 2)
            ret
Routine2    mov    count,#200      (global label Routine2)
:loop       call   send_b (local label :loop)
            djnz   count,:loop     (jump to line 6)
            ret

Routine3    mov    count,#250      (global label R  outine3)
            jmp    Routine2:loop   (jump to line 6)
```

A Local label can be called from outside its usual area as demonstrated in the line:

```
jmp    Routine2:loop    (jump to line 6)
```

4.1.11 Default_Symbol_Tables

When the assembler is started, its symbol table is initialized with various PIC symbols, such as C for the Carry register, RA for Port A, etc. This saves you from having to define every register and bit in the PIC.

As the number of different PICs has grown, it has become necessary to have separate symbol tables for each PIC. At the beginning of your program, you must tell the assembler which PIC is being used. This is done with the INCLUDE directive (explained later in this chapter).

If you'd like to see the symbol table for a particular PIC, please refer to the device include files installed with the assembler (default location = C:\PicTools\*.inc). These files are named according to the targeted device.

4.1.12 Comments

Comments can be placed anywhere in your source code, beginning at any point on a line and continuing to the end.

A semicolon initiates a comment. The following are example comments:

```
;
; Move literal value 61h into w
;
Input           =           10h
Output  =           11h
mov     Input,#61h          ; Load 61h into Input
mov     Output,#10h         ; Load 10h into Output
call    talk_host           ; Call routine to communicate with host
```

Blank lines can be used to provide space between lines and make the code more readable.

4.1.13 Assembler Directives

Assembler directives are instructions that direct the assembler to do something.

Directives do many things; some tell the assembler to set aside space for variables, others tell the assembler to include additional source files, and others establish the start address for your program. T he directives available are shown below:

=	Assigns a value to a symbol (same as EQU)
EQU	Assigns a value to a symbol (same as =)

ORG Sets the current origin to a new value. This is used to set the program or register address during assembly. For example, ORG 0100h tells the assembler to assemble all subsequent code starting at address 0100h.

DS Defines an amount of free space. No code is generated. This is sometimes used for allocating variable space.

ID Sets the PIC’s identification bytes. PIC16C5x chips have two ID bytes, which can be set to a 2-byte value. Newer PICs have four 7-bit ID locations, which can be filled with a 4-character text string.

INCLUDE Loads another source file during assembly. This allows you to insert an additional source file into your code during assembly. Included source files usually contain common routines or data. By using an INCLUDE directive at the beginning of your program, you can avoid re-typing common information. Included files may not contain other included files. NOTE: The Device Include directive (i.e. INCLUDE 'C:\PicTools\16F877.inc') for the targeted device MUST be at the beginning of your source code.

FUSES NOTE that FUSE CONFIGURATIONs can be '&' together on a single line and/or spread between multiple lines. ALL FUSES directives are ANDed together to create the composite FUSE CONFIGURATION. (view the device "include" file for specific fuse syntax)

IF <expression> Assembles code if expression evaluates to TRUE.

IFNOT <expression> Assembles code if expression evaluates to FALSE.

ELSE Assembles code if preceeding evaluation is rejected.

ENDIF Ends conditional evaluation.

RESET Sets the reset start address. This address is where program execution will start following a reset. A jump to the given address is inserted at the last location in memory. After the PIC is reset, it starts executing code at the last location, which holds the jump to the given address. RESET is only available for PIC16C5x chips.

EEORG Sets the current data EEPROM origin to a new value. This is used to set the data EEPROM address during assembly. This directive usually precedes EEDATA. EEORG is only available for PICs that have EEPROM memory .

EEDATA Loads data EEPROM with given values. This provides a means of automatically storing values in the data EEPROM when the PIC is programmed. This is handy for storing configuration or start-up information. EEDATA is only available for PICs that have EEPROM memory.

Assembler Directive Examples

```
Include 'C:\PICTOOLS\16C877.inc' ; loads default symbols for the targeted device.
FUSES  _WD_OFF&_LP_OSC  ; specify multiple fuse settings using the '&' operator.
FUSES _CP_ON            ; Specifies 1 fuse s  etting per line.
Digit   =       43h     ; Assign value 43h to Digit
Max     EQU     1Ah     ; Assign value 1Ah to Max

ORG     10h             ; Set assembly address to 10h

Count   DS      2       ; Define 2 bytes at 10h & 11h
                        ; Bytes can be referred to
                        ; later as Count and Count+1

ID      1234h           ; Set 16C5x ID to 1234h
ID      'ABCD'          ; Set newer PIC ID to 'ABCD'

INCLUDE         'KEYS.SRC'      ; Include KEYS.SRC file at
                                ; point of insertion

RESET Start             ; Set 16C5x   reset jump to
                                ; location at Start

Start   mov     Count,#00               ; This will be executed when a '5X PIC is reset

EEORG           10h             ; Set EEPROM address to 10h
EEDATA          02h,88h,34h     ; Store 3 bytes in EEPROM
```

4.1.14 Source Code Formatting

We recommend that you format your source code with evenly spaced tabs, pr eferably 8 spaces each, since this will lend consistency to assembly listings. The assemblers are not case sensitive (except in the instance of strings), so you may follow your own convention for using upper and lower case.

4.1.15 CVASM16 Error Messages

During assembly, any syntax errors will be brought to your attention.

You can pause an error list by typing CTRL-S;   you can also press ESC or CTRL-C to abort the error list.

The following error messages may occur during assembly. They will always be preceded by "ERROR filename xxx", where filename is the file and xxx is the line number where the error occurred.

Address limit of xxxxh was exceeded:
Data was assembled at an address which exceeded the limit for the given PIC.

Attempt to divide by 0:
An attempt was made to divide a quantity by zero.

Bit number must be from 0 to 7:
A bit-address expression attempted to use a bit number greater than 7.

Data was already entered at location xxxxh:
An object code location which had   already been assigned data, was written to again.

Equate directive must be preceded by a symbol:
An "EQU" or "=" directive was not preceded by a necessary symbol.

Illegal mnemonic:
The assembler encountered an unknown directive   or instruction.

Include files cannot be nested:
An INCLUDE directive was found in an included file.

Syntax error in operand:
The operand contained an expression which did not follow proper syntax.

Invalid filename for include file: An invalid filename was given in an INCLUDE directive.

Line cannot exceed 256 characters:
Line length exceeded the line limit of 256 characters.

Mnemonic field cannot exceed 7 characters:
More than 7 characters were in the mnemonic (instruction) field.

Illegal mnemonic/operand combination:
The operand structure did not match the instruction’s or directive’s possibilities.

Literal value must be from 0 to 0FFh:
A value which needed to be from 0 to 255, was greater than 255.

Operand field cannot exceed 256 characters:
More than 256 characters were in the operand field after expansion by the assembler.

Redefinition of symbol xxxx:
An attempt was made to redefine a symbol that was already defined.

Symbol field cannot exceed 32 characters:
More than 32 characters were in the symbol field.

(operand) symbol must contain only letters, numbers, '_', and ':':
A symbol contained illegal characters.

(operand) symbol is a reserved word:
A symbol was identical to a reserved word. The symbols you define must not be reserved words.

(operand) symbol is too long:
A symbol referenced in the operand exceeded 32 characters.

(operand) symbol must begin with a letter or '_':
A symbol started with an illegal character.

Symbol table full:
The symbol table's 16K limit was exceeded.

Use of unknown symbol xxxx:
A symbol was referenced in the operand, which was never declared earlier in your program.

4.2 TechTools Instruction Set

For a listing of the Microchip Instruction set and the equivalent CVASM16 instruction; see the next section, "Microchip to CVASM16 145"

ADD, ADDB 148
AND 150
CALL 151
CJA, CJAE 152
CJB, CJBE 153

CJE 154
CJNE 155
CLR 156
CLRB, CLC, CLZ 157
CSA, CSAE 157
CSB, CSBE 159
CSE 160
CSNE 161
DEC, DECSZ, DJNZ 162
IJNZ, INC, INCSZ 163
JB, JC 164
JMP 164
JNB, JNC 165
JNZ, JZ 166
LCALL, LJMP, LSET 166
MOV [to register] 169
MOV [to OPTION] 169
MOV [to I/O Control] 171
MOV [to W] 172
MOV [using Expressions] 172
MOVB 174
MOVSZ 175
NOP 175
NOT 176
OR 176
RET, RETW 178
RL, RR 179
SB, SC, SZ 179
SETB, STC, STZ 180
SKIP 181
SLEEP 181
SNB, SNC, SNZ 182
SUB, SUBB 183
SWAP 184
TEST 184
XOR 185

4.2.1 Microchip to CVASM16

This table is designed to help the CVASM16 user to understand source code written in Microchip format. It can also be a valuable reference when converting source to TechTools format. When doing this, keep in mind that CVASM16 will accept ALL of Microchip's instruction set, but not Microchip's directives and macros.

f = File Register w = Working Register or "W"

Microchip		Description	TechTools Equivalent	
Byte-oriented File Register operations				
ADDWF	f,0	Add f into w	ADD 148	W,f

ADDWF	f,1	Add W into f		ADD 148	f,w
ANDWF	f,0	And f into w		AND 150	W,f
ANDWF	f,1	And W into f		AND 150	f,w
CLRF	f	Clear f		CLR 156	f
CLRW		Clear W		CLR 156	W
COMF	f,0	Complement f, store in w		MOV 172	W,/f
COMF	f,1	Complement f, store in f		NOT 176	f
DECF	f,0	Decrement f, store in w		MOV 172	W,--f
DECF	f,1	Decrement f, store in f		DEC 162	f
DECFSZ	f,0	Decrement f into w, skip if 0		MOVSZ 175	W,--f
DECFSZ	f,1	Decrement f into f, skip if 0		DECSZ 162	f
INCF	f,0	Increment f into w		MOV 172	W,++f
INCF	f,1	Increment f into f		INC 163	f
INCFSZ	f,0	Increment f into w, skip if 0		MOVSZ 175	W,++f
INCFSZ	f,1	Increment f into f, skip if 0		INCSZ 163	f
IORWF	f,0	Or W and f into w		OR 176	W,f
IORWF	f,1	Or W and f into f		OR 176	f,W
MOVF	f,0	Move f into w		MOV 172	W,f
MOVF	f,1	Move f to itself		TEST 184	f
MOVWF	f	Move W to f		MOV 169	f,W
NOP		No operation		NOP 175	
RLF	f,0	Rotate f left w/carry, in w		MOV 172	W,<<fr
RLF	f,1	Rotate f left w/carry, in f		RL 179	f
RRF	f,0	Rotate f right w/carry, in w		MOV 172	W,>>f
RRF	f,1	Rotate f right w/carry, in f		RR 179	f
SUBWF	f,0	Subtract W from f in w		MOV 172	W,f-W
SUBWF	f,1	Subtract W from f in f		SUB 172	f,W
SWAPF	f,0	Swap nibbles of f in w		MOV 172	W,<>f
SWAPF	f,1	Swap nibbles of f in f		SWAP 184	f
XORWF	f,0	Xor W and f in w		XOR 185	W,f
XORWF	f,1	Xor W and f in f		XOR 185	f,W

Bit-oriented File Register operations*

BCF	f,b	Bit clear f		CLRB 157	bit
BSF	f,b	Bit set f		SETB 180	bit
BTFSC	f,b	Bit test f, skip if clear		SNB 182	bit
BTFSS	f,b	Bit test f, skip if set		SB 179	bit

Literal and Control operations

ADDLW	lit	Add literal into W	('xx)	ADD 148	W,#lit
ANDLW	lit	And literal into W		AND 150	W,#lit
CALL	addr	Call to address	('5x)	CALL 151	addr8

			('xx)	CALL 151	
addr11					
CLRWDT		Clear WDT and prescaler		CLR 156	WDT
GOTO	addr	Go to address	('5x)	JMP 164	
addr9					
			('xx)	JMP 164	
addr11					
IORLW	lit	Or literal into W		OR 176	
W,#lit					
MOVLW	lit	Move literal into W		MOV 172	
W,#lit					
OPTION		Move W into OPTION	('5x)	MOV 169	
!OPTION,W					
RETFIE		Return from interrupt	('xx)	RETI 178	
RETLW	lit	Return with literal in W		RETW 178	lit
		Return, clearing W	('5x)	RET 178	
RETURN		Return from subroutine	('xx)	RET 178	
SLEEP		Clear WDT and enter sleep mode		SLEEP 181	
SUBLW	lit	Subtract W from literal	('xx)	MOV 172	
W,#lit-W					
TRIS	port	Move W into port's TRIS	('5x)	MOV 171	
!port,W					
XORLW	lit	Xor literal into W		XOR 185	
W,#lit					

Special Instructions

ADDCF	f,0	Add carry to f, store in w [1]		ADDCF	f,0
ADDCF	f,1	Add carry to f, store in f [1]		ADDB 148	f,C
ADDDCF	f,0	Add digit carry to f,store in w [1]		ADDDCF	f,0
ADDDCF	f,1	Add digit carry to f,store in f [1]		ADDB 148	f,DC
B	k	Branch	('5x)	JMP 164	
addr9					
			('xx)	JMP 164	
addr11					
BC	k	Branch on Carry [1]	('5x)	JC 164	
addr9					
			('xx)	JC 164	
addr11					
BDC	k	Branch on Digit Carry [1]	('5x)	JB 164	
DC,addr9					
			('xx)	JB 164	
DC,addr11					
BNC	k	Branch on No Carry [1]	('5x)	JNC 165	
addr9					
			('xx)	JNC 165	
addr11					
BNDC	k	Branch on No Digit Carry [1]	('5x)	JNB 165	
DC,addr9					
			('xx)	JNB 165	
DC,addr11					
BNZ	k	Branch on No Zero [1]	('5x)	JNZ 166	
addr9					
			('xx)	JNZ 166	
addr11					
BZ	k	Branch on Zero [1]	('5x)	JZ 166	
addr9					

			('xx)	JZ 166	addr11
CLRC		Clear Carry		CLC 157	
CLRDC		Clear Digit Carry		CLRB 157	DC
CLRZ		Clear zero		CLZ 157	
LCALL	k	Long Call [1]	('5x)	LCALL 166	addr11
			('xx)	LCALL 166	addr13
LGOTO	k	Long Goto [1]	('5x)	LJMP 166	addr11
			('xx)	LJMP 166	addr13
MOVFW	k	Move f into W		MOV 172	W,f
NEGF	f,0	Negate f, store in w [1]		NEGF	f,0
NEGF	f,1	Negate f, store in f [1]		NEGF	f,1
SETC		Set carry		STC 180	
SETDC		Set digit carry		SETB 180	DC
SETZ		Set zero		STZ 180	
SKPC		Skip if carry		SC 179	
SKPDC		Skip if digit carry		SB 179	DC
SKPNC		Skip if not carry		SNC 182	
SKPNDC		Skip if not digit carry		SNB 182	DC
SKPNZ		Skip if not zero		SNZ 182	
SKPZ		Skip if zero		SZ 179	
SUBCF	f,0	Subtract carry from f, in w [1]		SUBCF	f,0
SUBCF	f,1	Subtract carry from f, in f [1]		SUBB 183	f,C
SUBDCF	f,0	Sub digit carry from f, in w [1]		SUBDCF	f,0
SUBDCF	f,1	Sub digit carry from f, in f [1]		SUBB 183	f,DC
TSTF	f	Test f		TEST 184	f

* (alternate for bit operand) bit = file register.bit
[1] multiple opcode instruction

4.2.2 ADD, ADDB

ADD fr,#literal Add literal into fr

Words: 2 Cycles: 2 Affects: W, C, DC, Z

Operation: Literal is added into fr via W. C will be set if an overflow occurs; otherwise, C will be cleared. DC will be set or cleared depending on whether or not an overflow occurs in the lower nibble. Z will be set if the result is 0; otherwise, Z will be cleared. W is left holding the literal value.

Coding: MOV W,#lit (MOVLW lit)
ADD fr,W (ADDWF fr,1)

ADD fr1,fr2 Add fr2 into fr1

Words: 2 Cycles: 2 Affects: W, C, DC, Z

Operation: Fr2 is added into fr1 via W. C will be set if an overflow occurs; otherwise, C will be cleared. DC will be set or cleared depending on whether or not an overflow occurs in the lower nibble. Z will be set if the result is 0; otherwise, Z will be cleared. W is left holding the contents of frb.

Coding: MOV W,fr2 (MOVF fr2,0)
ADD fr1,W (ADDWF fr1,1)

ADD fr,W Add W into fr

Words: 1 Cycles: 1 Affects: C, DC, Z

Operation: W is added into fr. C will be set if an overflow occurs, otherwise C will be cleared. DC will be set or cleared depending on whether or not an overflow occurs in the lower nibble. Z will be set if the result is 0, otherwise Z will be cleared.

Coding: ADDWF 145 fr,1

ADD W,fr Add fr into W

Words: 1 Cycles: 1 Affects: C, DC, Z

Operation: Fr is added into W. C will be set if an overflow occurs, otherwise C will be cleared. DC will be set or cleared depending on whether or not an overflow occurs in the lower nibble. Z will be set if the result is 0, otherwise Z will be cleared.

Coding: ADDWF 145 fr,0

ADDB fr,bit Add bit into fr

Words: 2 Cycles: 2 Affects: Z

Operation: If bit is set, fr is incremented. If fr is incremented, Z will be set if the result is 0; otherwise, Z will be cleared. This instruction is useful for adding the carry into the upper byte of a double-byte sum after the lower byte has been computed.

Coding: SNC bit (BTFSC bit)
INC fr (INCF fr,1)

4.2.3 AND

AND fr,#literal AND literal into fr

Words: 2 Cycles: 2 Affects: W, Z

Operation: Literal is AND'd into fr via W. Z will be set if the result is 0; otherwise, Z will be cleared.

Coding: MOV W,#lit (MOVLW lit)
AND fr,W (ANDWF fr,1)

AND fr1,fr2 AND fr2 into fr1

Words: 2 Cycles: 2 Affects: W, Z

Operation: Fr2 is AND'd into fr1 via W. Z will be set if the result is 0; otherwise, Z will be cleared.

Coding: MOV W,fr2 (MOVF fr2,0)
AND fr1,W (ANDWF fr1,1)

AND fr,W AND W into fr

Words: 1 Cycles: 1 Affects: Z

Operation: W is AND'd into fr. Z will be set if the result is 0; otherwise, Z will be cleared.

Coding: ANDWF 145 fr,1

AND W,#literal AND literal into W

Words: 1 Cycles: 1 Affects: Z

Operation: Literal is AND'd into W. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: ANDLW 145 literal

AND W,fr AND fr into W into fr

Words: 1 Cycles: 1 Affects: Z

Operation: Fr is AND'd into W. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: ANDWF 145 fr,0

4.2.4 CALL

CALL addr8 Call subroutine ('5x)

Words: 1 Cycles: 2 Affects: none

'5x Operation: The next instruction address is pushed onto the stack and addr8 is moved to the program counter bits 0-7(PC). The 9th bit of the program counter will always be cleared. Therefore, calls are only allowed to the first half of any 512-word page. The CALL instruction can be anywhere, but the address being called MUST be an 8bit address. If the device you are targeting has more than 512 words of code space, you will need to load the page pre-select bits (status 5,6) to the proper value before executing the CALL. See LCALL 166 , LSET 166 for '5x devices.

Coding: CALL 145 addr8

CALL addr11 Call subroutine ('xx)

Words: 1 Cycles: 2 Affects: none

'xx Operation: The next instruction address is pushed onto the stack and addr11 is moved to the program counter bits 0-10 (PC). The upper bits (11,12) of the program counter will be loaded from PCLATH (3,4). Therefore, it is up to the user to pre-load PCLATH with the correct values BEFORE a CALL is initiated. This can be done easily by using the LSET 166 instruction (for 'xx devices) before CALL, or simply using the LCALL 166 instruction (in place of CALL), which will set the PCLATH bits for you. If your program appears to 'jump' to the wrong code page when executing a call, your PCLATH bits are probably incorrect.

Coding: CALL 145 addr11

4.2.5 CJA, CJAE

CJA fr,#literal,addr9 Compare fr to literal and jump if above

Words: 4 Cycles: 4 or 5 (jump) Affects: C, DC, Z

Operation: Fr is compared to literal via W. If fr is greater than literal, a jump to addr9 is executed.

```
Coding: MOVLW      literal^0FFh
  ADDWF            fr,0
  BTFSC       3,0
  GOTO        addr9
```

CJA fr1,fr2,addr9 Compare fr1 to fr2 and jump if above

Words: 4 Cycles: 4 or 5 (jump) Affects: W, C, DC, Z

Operation: Fr1 is compared to fr2 via W. If fr1 is greater than fr2, a jump to addr9 is executed.

```
Coding: MOVF       fr1,0
  SUBWF            fr2,0
  BTFSS       3,0
  GOTO        addr9
```

CJAE fr,#literal,addr9 Compare fr to literal and jump if above or equal

Words: 4 Cycles: 4 or 5 (jump) Affects: C, DC, Z

Operation: Fr is compared to literal via W. If fr is greater than or equal to literal, a jump to addr9 is executed.

Coding:
```
MOVLW     literal
SUBWF     fr,0
BTFSC     3,0
GOTO      addr9
```

CJAE fr1,fr2,addr9 Compare fr1 to fr2 and jump if above or equal

---

Words: 4 Cycles: 4 or 5 (jump) Affects: C, DC, Z

Operation: Fr1 is compared to fr2 via W. If fr1 is greater than or equal to fr2, a jump to addr9 is executed.

Coding:
```
MOVF      fr2,0
SUBWF     fr1,0
BTFSC     3,0
GOTO      addr9
```

4.2.6 CJB, CJBE

CJB fr,#literal,addr9 Compare fr to literal and jump if below

---

Words: 4 Cycles: 4 or 5 (jump) Affects: C, DC, Z

Operation: Fr is compared to literal via W. If fr is less than literal, a jump to addr9 is executed.

Coding:
```
MOVLW     literal
SUBWF     fr,0
BTFSS     3,0
GOTO      addr9 ad
```

CJB fr1,fr2,addr9 Compare fr1 to fr2 and jump if below

Words: 4 Cycles: 4 or 5 (jump) Affects: C, DC, Z

Operation: Fr1 is compared to fr2 via W. If fr1 is less than fr2, a jump to addr9 is executed.

```
Coding:   MOVF        fr2,0
  SUBWF               fr1,0
  BTFSS       3,0
  GOTO        addr9b a
```

CJBE fr,#literal,addr9 Compare fr to literal and jump if below or equal

---

Words: 4 Cycles: 4 or 5 (jump) Affects: C, DC, Z

Operation: Fr is compared to literal via W. If fr is less than or equal to literal, a jump to addr9 is executed.

```
Coding:   MOVLW               literal
  ADDWF               fr,0
  BTFSS       3,0
  GOTO        addr9
```

CJBE fr1,fr2,addr9 Compare fr1 to fr2 and jump if below or equal

---

Words: 4 Cycles: 4 or 5 (jump) Affects: C, DC, Z

Operation: Fr1 is compared to fr2 via W. If fr1 is less than or equal to fr2, a jump to addr9 is executed.

```
Coding:   MOVF        fr1,0
  SUBWF               fr2,0
  BTFSS       3,0
  GOTO        addr9
```

4.2.7 CJE

CJE fr,#literal,addr9 Compare fr to literal and jump if equal

---

Words: 4 Cycles: 4 or 5 (jump) Affects: C, DC, Z

Operation: Fr is compared to literal via W. If fr is equal to literal, a jump to addr9 is executed.

```
Coding:   MOVLW              literal
  SUBWF            fr,0
  BTFSC     3,2
  GOTO      addr9
```

CJE fr1,fr2,addr9 Compare fr1 to fr2 and jump if equal

Words: 4 Cycles: 4 or 5 (jump) Affects: C, DC, Z

Operation: Fr1 is compared to fr2 via W. If fr1 is equal to fr2, a jump to addr9 is executed.

```
Coding:   MOVF             fr2,0
  SUBWF            fr1,0
  BTFSC     3,2
  GOTO      addr9
```

4.2.8 CJNE

CJNE fr,#literal,addr9 Compare fr to literal and jump if not equal

Words: 4 Cycles: 4 or 5 (jump) Affects: C, DC, Z

Operation: Fr is compared to literal via W. If fr is not equal to literal, a jump to addr9 is executed.

```
Coding:   MOVLW              literal
  SUBWF            fr,0
  BTFSS     3,2
  GOTO      addr9
```

CJNE fr1,fr2,addr9 Compare fr1 to fr2 and jump if not equal

Words: 4 Cycles: 4 or 5 (jump) Affects: C, DC, Z

Operation: Fr1 is compared to fr2 via W. If fr1 is not equal to fr2, a jump to addr9 is executed.

Coding:
```
MOVF    fr2,0
SUBWF   fr1,0
BTFSS   3,2
GOTO    addr9
```

4.2.9 CLR

CLR fr Clear fr

Words: 1 Cycles: 1 Affects: Z

Operation: Fr is cleared to 0. Z is set to 1.

Coding: CLRF 145 fr

CLR W Clear W

Words: 1 Cycles: 1 Affects: Z

Operation: W is cleared to 0. Z is set to 1.

Coding: CLRW 145

CLR WDT Clear watchdog timer

Words: 1 Cycles: 1 Affects: TO, PD

Operation: The watchdog timer is cleared, along with the prescaler, if assigned. TO and PD are set to 1.

Coding: CLRWDT 145

4.2.10 CLRB, CLC, CLZ

CLRB bit Clear bit

Words: 1	Cycles: 1	Affects: none

Operation: Bit is cleared to 0.

Coding: BCF 145 bit

Note: The TechTools assemblers define a bit as port.bitposition, as in the following examples:

RA.3 = bit 3 of port A
PortB.0 = bit 0 of port B

CLC Clear carry

Words: 1	Cycles: 1	Affects: C

Operation: C is cleared to 0.

Coding: BCF 145 3,0

CLZ Clear zero

Words: 1	Cycles: 1	Affects: Z

Operation: Z is cleared to 0.

Coding: BCF 145 3,2

4.2.11 CSA, CSAE

CSA fr,#literal Compare fr to literal and skip if above

Words: 3	Cycles: 3 or 4 (skip)	Affects: C, DC, Z

Operation: Fr is compared to literal via W. If fr is greater than literal, the following instruction word is skipped.

```
Coding: MOVLW     literal
  ADDWF      fr,0
  BTFSS      3,0
```

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSA is a single-word instruction.

CSA fr1,fr2 Compare fr1 to fr2 and skip if above

Words: 3 Cycles: 3 or 4 (skip) Affects: C, DC, Z

Operation: Fr1 is compared to fr2 via W. If fr1 is greater than fr2, the following instruction word is skipped.

```
Coding: MOVF      fr1,0
  SUBWF      fr2,0
  BTFSC      3,0
```

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSA is a single-word instruction.

CSAE fr,#literal Compare fr to literal and skip if above or equal

Words: 3 Cycles: 3 or 4 (skip) Affects: C, DC, Z

Operation: Fr is compared to literal via W. If fr is greater than or equal to literal, the following instruction word is skipped.

```
Coding: MOVLW     literal
  SUBWF      fr,0
  BTFSS      3,0
```

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSAE is a single-word instruction.

CSAE fr1,fr2 Compare fr1 to fr2 and skip if above or equal

Words: 3 Cycles: 3 or 4 (skip) Affects: C, DC, Z

Operation: Fr1 is compared to fr2 via W. If fr1 is >= fr2, the following instruction word is skipped.

Coding:
```
MOVF            fr2,0
SUBWF           fr1,0
BTFSS       3,0
```

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSAE is a single-word instruction.

4.2.12 CSB, CSBE

CSB fr,#literal Compare fr to literal and skip if below

Words: 3 Cycles: 3 or 4 (skip) Affects: C, DC, Z

Operation: Fr is compared to literal via W. If fr is less than literal, the following instruction word is skipped.

Coding:
```
MOVLW                   literal
SUBWF           fr,0
BTFSC       3,0
```

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSB is a single-word instruction.

CSB fr1,fr2 Compare fr1 to fr2 and skip if below

Words: 3 Cycles: 3 or 4 (skip) Affects: C, DC, Z

Operation: Fr1 is compared to fr2via W. If fr1 is less than fr2, the following instruction word is skipped.

```
Coding:   MOVF        fr2,0
  SUBWF               fr1,0
  BTFSC       3,0
```

Note:	Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSB is a single-word instruction.

CSBE fr,#literal	Compare fr to literal and skip if below or equal

---

Words: 3	Cycles: 3 or 4 (skip)	Affects: C, DC, Z

Operation:	Fr is compared to literal via W. If fr is less than or equal to literal, the following instruction word is skipped.

```
Coding:   MOVLW               literal
  ADDWF               fr,0
  BTFSC       3,0
```

Note:	Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSBE is a single-word instruction.

CSBE fr1,fr2	Compare fr1 to fr2 and skip if below or equal

Words: 3	Cycles: 3 or 4 (skip)	Affects: C, DC, Z

Operation:	Fr1 is compared to fr2 via W. If fr1 is less than or equal to fr2, the following instruction word is skipped.

```
Coding:   MOVF        fr1,0
  SUBWF               fr2,0
  BTFSS       3,0
```

Note:	Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSBE is a single-word instruction.

4.2.13 CSE

CSE fr,#literal	Compare fr to literal and skip if equal

Words: 3 Cycles: 3 or 4 (skip) Affects: C, DC, Z

Operation: Fr is compared to literal via W. If fr is equal to literal, the following instruction word is skipped.

```
Coding: MOVLW          literal
  SUBWF          fr,0
  BTFSS     3,2
```

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSE is a single-word instruction.

CSE fr1,fr2 Compare fr1 to fr2 and skip if equal

Words: 3 Cycles: 3 or 4 (skip) Affects: C, DC, Z

Operation: Fr1 is compared to fr2 via W. If fr1 is equal to fr2, the following instruction word is skipped.

```
Coding: MOVF      fr2,0
  SUBWF          fr1,0
  BTFSS     3,2
```

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSE is a single-word instruction.

4.2.14 CSNE

CSNE fr,#literal Compare fr to literal and skip if not equal

Words: 3 Cycles: 3 or 4 (skip) Affects: C, DC, Z

Operation: Fr is compared to literal via W. If fr is not equal to literal, the following instruction word is skipped.

```
Coding: MOVLW          literal
  SUBWF          fr,0
  BTFSC     3,2
```

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSNE is a single-word instruction.

CSNE fr1,fr2 Compare fr1 to fr2 and skip if not equal

Words: 3 Cycles: 3 or 4 (skip) Affects: C, DC, Z

Operation: Fr1 is compared to fr2 via W. If fr1 is not equal to fr2, the following instruction word is skipped.

```
Coding:   MOVF         fr2,0
  SUBWF                fr1,0
  BTFSC         3,2
```

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following CSNE is a single-word instruction.

4.2.15 DEC, DECSZ, DJNZ

DEC fr Decrement fr

Words: 1 Cycles: 1 Affects: Z

Operation: Fr is decremented. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: DECF 145 fr,1

DECSZ fr Decrement fr and skip if zero

Words: 1 Cycles: 1 or 2 (skip) Affects: none

Operation: Fr is decremented. The next instruction word will be skipped if the result was 0.

Coding: DECFSZ 145 fr,1

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following DECSZ is a single-word instruction.

DJNZ fr,addr9 Decrement fr and jump if not zero

Words: 2 Cycles: 2 or 3 (jump) Affects: none

Operation: Fr is decremented. If the result is not 0, a jump to addr9 is executed.

Coding: DECFSZ fr,1
GOTO addr9

4.2.16 IJNZ, INC, INCSZ

IJNZ fr,addr9 Increment fr and jump if not zero

Words: 2 Cycles: 2 or 3 (jump) Affects: none

Operation: Fr is incremented. If the result is not 0, a jump to addr9 is executed.

Coding: INCFSZ fr,1
GOTO addr9

INC fr Increment fr

Words: 1 Cycles: 1 Affects: Z

Operation: Fr is incremented. Z will be set if the result was 0, otherwise Z will be cleared.

Coding: INCF 145 fr,1

INCSZ fr Increment fr and skip if zero

Words: 1 Cycles: 1 or 2 (skip) Affects: none

Operation: Fr is incremented. The next instruction word will be skipped if the result was 0.

Coding: INCFSZ 145 fr,1

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following INCSZ is a single-word instruction.

4.2.17 JB, JC

JB bit,addr9 Jump if bit

Words: 2 Cycles: 2 or 3 (jump) Affects: none

Operation: If bit is set, a jump to addr9 is executed.

Coding: BTFSC bit
GOTO addr9

JC addr9 Jump if carry

Words: 2 Cycles: 2 or 3 (jump) Affects: none

Operation: If the carry bit is set, a jump to addr9 is executed.

Coding: BTFSC 3,0
GOTO addr9

4.2.18 JMP

JMP addr9 Jump to address

Words: 1 Cycles: 2 Affects: none

Operation: The lower 9-bits of the literal addr9 is moved into the program counter. The upper bits of the Program counter are loaded from bits 3 & 4 of PCLATH.

Coding: GOTO 145 addr9

JMP PC+W Jump to PC+W

Words: 1 Cycles: 2 Affects: C, DC, Z

Operation: W+1 is added into the program counter. The 9th bit of the program counter is always cleared to 0, so the jump destination will be in the first 256 words of any 512-word page. This instruction is useful for jumping into lookup tables comprised of RETW data, or jumping to particular routines. The flags are set as they would be by an ADD instruction.

Coding: ADDWF 145 2,1

JMP W Jump to W

Words: 1 Cycles: 2 Affects: none

Operation: W is moved into the program counter. The 9th bit of the program counter is always cleared to 0, so the jump destination will be in the first 256 words of any 512-word page. This instruction is useful for jumping into lookup tables comprised of RETW data, or jumping to particular routines.

Coding: MOVWF 145 2

4.2.19 JNB, JNC

JNB bit,addr9 Jump if not bit

Words: 2 Cycles: 2 or 3 (jump) Affects: none

Operation: If bit reads 0, a jump to addr9 is executed.

Coding: BTFSS bit
GOTO addr9

Note: The TechTools assemblers define a bit as port.bitposition, as in the following examples:

RA.3 = bit 3 of port A

PortB.0 = bit 0 of port B

JNC addr9 Jump if not carry

Words: 2	Cycles: 2 or 3 (jump)	Affects: none	

Operation: If C is 0, a jump to addr9 is executed.

```
Coding: BTFSS       3,0
  GOTO              addr9
```

4.2.20 JNZ, JZ

JNZ addr9 Jump if not zero

Words: 2 Cycles: 2 or 3 (jump) Affects: none

Operation: If Z is 0, a jump to addr9 is executed.

```
Coding: BTFSS       3,2
  GOTO              addr9
```

JZ addr9 Jump if zero

Words: 2 Cycles: 2 or 3 (jump) Affects: none

Operation: If Z is 1, a jump to addr9 is executed.

```
Coding: BTFSC       3,2
  GOTO              addr9
```

4.2.21 LCALL, LJMP, LSET

LCALL addr11 Long call ('5x)

Words: 1-3 Cycles: 2-4 Affects: none

'5x Operation: Depending on the device size, from zero to two BCF/BSF instructions will be assembled to point the page pre-select bits (STATUS 5,6) to addr11's page. The bit set/clear instructions are followed by a CALL to addr8. Long calls can be made to any page, but the routine that is being called MUST be located in the lower half of that page.

This is due to the fact that bit 9 of the program counter is ALWAYS cleared when processing a CALL instruction on '5x devices (see CALL 151 for '5x devices). This instruction is only useful for PICs with more than 512 words.

Coding: (BCF/BSF 3, 5)

(BCF/BSF 3, 6)

CALL addr8

Note: Please note that LCALL does not set the page select bits upon returning to the calling routine. Therefore, your program must set these bits. This can be done using LSET $, which sets the page select bits to the current page.

LCALL addr13 Long call ('xx)

Words: 1-3 Cycles: 2-4 Affects: none

'xx Operation: Depending on the device size, from zero to two BCF/BSF instructions will be assembled to load the PCLATH bits to addr13's page. The bit set/clear instructions are followed by a CALL to addr11. This instruction is only useful for PICs with more than 2K words.

Coding: (BCF/BSF 0A ,3)

(BCF/BSF 0A , 4)

CALL addr11

Note: Please note that LCALL does not set the PCLATH bits upon returning to the calling routine. Therefore, your program must set these bits. This can be done using LSET $, which sets the page select bits to the current page.

LJMP addr11 Long jump ('5x)

Words: 1-3 Cycles: 2-4 Affects: none

'5x Operation: Depending on the device size, from zero to two BCF/BSF instructions will be assembled to point the page pre-select bits (STATUS 5,6) to addr11's page. The bit set/clear instructions are followed by a jump to addr8. This instruction is only useful for PICs with more than 512 words.

Coding: (BCF/BSF 3, 5)

(BCF/BSF 3, 6)

GOTO addr8

LJMP addr13 Long jump ('xx)

Words: 1-3 Cycles: 2-4 Affects: none

'xx Operation: Depending on the device size, from zero to two BCF/BSF instructions will be assembled to load the PCLATH bits to addr13's page. The bit set/clear instructions are followed by a jump to addr11. This instruction is only useful for PICs with more than 2K words.

```
Coding:   (BCF/BSF      0A ,3)
  (BCF/BSF      0A , 4)
  GOTO  addr11
```

LSET addr11 Long set ('5x)

Words: 0-2 Cycles: 0-2 Affects: none

'5x Operation: Depending on the device size, from zero to two BCF/BSF instructions will be assembled to point the page pre-select bits (STATUS 5,6) to addr11's page. This instruction is only useful for PICs with more than 512 words.

```
Coding:   (BCF/BSF      3, 5)
  (BCF/BSF      3, 6)
```

LSET addr13 Long set ('xx)

Words: 0-2 Cycles: 0-2 Affects: none

'xx Operation: Depending on the device size, from zero to two BCF/BSF instructions will be assembled to load the PCLATH bits to addr13's page. This instruction is only useful for PICs with more than 2K words.

```
Coding:   (BCF/BSF      0A ,3)
  (BCF/BSF      0A , 4)
```

4.2.22 MOV [to register]

MOV fr,#literal Move literal into fr

Words: 2	Cycles: 2	Affects: none

Operation: Literal is moved into fr via W.

Coding:
```
MOVLW   literal
MOVWF   fr
```

MOV fr1,fr2 Move fr2 into fr1

Words: 2	Cycles: 2	Affects: Z

Operation: Fr2 is moved into fr1 via W. Z will be set to 1 if the value moved was 0, otherwise Z will be cleared to 0.

Coding:
```
MOVF    fr2,0
MOVWF   fr1
```

MOV fr,W Move W into fr

Words: 1	Cycles: 1	Affects: none

Operation: W is moved into fr.

Coding: MOVWF 145 fr

4.2.23 MOV [to OPTION]

NOTE: Since the OPTION register on 'xx devices is readable and writable, it can be accessed as a normal File Register. To maintain upward compatibility with future devices, these instructions should not be used when coding for 'xx devices.

MOV !OPTION,#literal Move literal into OPTION

Words: 2	Cycles: 2	Affects: none

Operation: Literal is moved into OPTION via W.

Coding: MOVLW literal
OPTION

Note: On 16C5x parts, you must use the explanation mark (!) before the word OPTION; this causes the assembler to assemble an OPTION instruction. On 16Cxx parts, the mark (!) is optional. Microchip added an OPTION register to the newer parts, so a regular MOV is possible, as long as your program is in the proper bank (usually bank 1). Microchip may remove the OPTION instruction in newer PICs, so they recommend against using the ' ! '

MOV !OPTION,fr Move fr into OPTION

Words: 2 Cycles: 2 Affects: Z

Operation: Fr is moved into OPTION via W. Z will be set to 1 if the value moved was 0, otherwise Z will be cleared to 0.

Coding: MOVF fr,0
OPTION

Note: On 16C5x parts, you must use the explanation mark (!) before the word OPTION; this causes the assembler to assemble an OPTION instruction. On 16Cxx parts, the mark (!) is optional. Microchip added an OPTION register to the newer parts, so a regular MOV is possible, as long as your program is in the proper bank (usually bank 1). Microchip may remove the OPTION instruction in newer PICs, so they recommend against using the ' ! '

MOV !OPTION,W Move W into OPTION

Words: 1 Cycles: 1 Affects: none

Operation: W is moved into OPTION.

Coding: OPTION 145

Note: On 16C5x parts, you must use the explanation mark (!) before the word OPTION; this causes the assembler to assemble an OPTION instruction. On 16Cxx parts, the mark (!) is optional. Microchip added an OPTION register to the newer parts, so a regular MOV is possible, as long as your program is in the proper bank (usually bank 1). Microchip may remove the OPTION instruction in newer PICs, so they recommend against using the ‘ ! ’

4.2.24 MOV [to I/O Control]

NOTE: Since TRIS registers on 'xx devices are readable and writable, they can be accessed as a normal File Register. To maintain upward compatibility with future devices, the " ! " qualifier should not be used when coding for 'xx devices.

MOV !port_fr,#literal Move literal into port_fr’s I/O control register

---

Words: 2	Cycles: 2	Affects: none

Operation: Literal is moved into the I/O control register of port_fr via W. A "1" bit in W disables the corresponding port pin’s output buffer, allowing input use, while a "0" bit enables the output buffer for high or low output. Port_fr must be (at Address) 5, 6, or 7.

```
Coding: MOVLW     literal
        TRIS      port_fr
```

MOV !port_fr,fr Move fr into port_fr’s I/O control register

Words: 2	Cycles: 2	Affects: Z

Operation: Fr is moved into the I/O control register of port_fr via W. A "1" bit in W disables the corresponding port pin’s output buffer, allowing input use, while a "0" bit enables the output buffer for high or low output. Z will be set to 1 if the value moved was 0, otherwise Z will be cleared to 0. Port_fr must be (at Address) 5, 6, or 7.

```
Coding: MOVF  fr,0
  TRIS        port_fr6
```

MOV !port_fr,W Move W into port_fr’s I/O control register

Words: 1	Cycles: 1	Affects: none

Operation: W is moved into the I/O control register of port_fr. A "1" bit in W disables the corresponding port pin’s output buffer, allowing input use, while a "0" bit enables the output buffer for high or low output. Port_fr must be (at Address) 5, 6, or 7.

Coding: TRIS 145 port_fr

4.2.25 MOV [to W]

MOV W,#literal Move literal into W

Words:	1	Cycles:	1	Affects:	none

Operation: Literal is moved into W.

Coding: MOVLW 145 literal

MOV W,fr Move fr into W

Words:	1	Cycles:	1	Affects:	Z

Operation: Fr is moved into W. Z will be set to 1 if the value moved was 0, otherwise Z will cleared to 0.

Coding: MOVF 145 fr,0

4.2.26 MOV [using Expressions]

MOV W,/fr Move not fr into W

Words:	1	Cycles:	1	Affects:	Z

Operation: The one's complement of fr is moved into W. Z will be set to 1 if the result was 0, otherwise Z will cleared to 0.

Coding: COMF 145 fr,0

MOV W,fr-W Move fr-W into W

Words:	1	Cycles:	1	Affects:	C, DC, Z

Operation: W is subtracted from fr and the result is stored in W. C will be cleared to 0 if an underflow occurred, otherwise C will be set to 1. DC will be cleared or set depending on whether or not an underflow occurred in the least-significant nibble. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: SUBWF 145 fr,0

MOV W,++fr Move the incremented value of fr into W

Words: 1 Cycles: 1 Affects: Z

Operation: The incremented value of fr is moved into W. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: INCF 145 fr,0

MOV W,--fr Move the decremented value of fr into W

Words: 1 Cycles: 1 Affects: Z

Operation: The decremented value of fr is moved into W. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: DECF 145 fr,0

MOV W,<<fr Move the left-rotated value of fr into W

Words: 1 Cycles: 1 Affects: C

Operation: The left-rotated value of fr is moved into W. On entry, C must hold the value to be shifted into the least-significant bit of the fr value. On exit, C will hold the previous most-significant bit of the fr value.

Coding: RLF 145 fr,0

MOV W,>>fr Move the right-rotated value of fr into W

Words: 1 Cycles: 1 Affects: C

Operation: The right-rotated value of fr is moved into W. On entry, C must hold the value to be shifted into the most-significant bit of the fr value. On exit, C will hold the previous least-significant bit of the fr value.

Coding: RRF 145 fr,0

MOV W,<>fr Move the nibble-swapped value of fr into W

Words: 1 Cycles: 1 Affects: none

Operation: The nibble-swapped value of fr is moved into W.

Coding: SWAPF 145 fr,0

4.2.27 MOVB

MOVB bit1,bit2 Move bit2 to bit1

Words: 4 Cycles: 4 Affects: none

Operation: Bit2 is moved to bit1.

Coding:
```
BTFSS bit2
BCF   bit1
BTFSC bit2
BSF   bit1
```

Note: The TechTools assemblers define a bit as port.bitposition, as in the following examples:

RA.3 = bit 3 of port A PortB.0 = bit 0 of port B

MOVB bit1,/bit2 Move not bit2 to bit1

Words: 4 Cycles: 4 Affects: none

Operation: The complement of bit2 is moved to bit1.

Coding: BTFSC bit2
  BCF bit1
  BTFSS bit2
  BSF bit1

Note: The TechTools assemblers define a bit as port.bitposition, as in the following examples:

RA.3 = bit 3 of port A PortB.0 = bit 0 of port B

4.2.28 MOVSZ

MOVSZW,++fr Move the incremented value of fr into W and skip if zero

---

Words: 1 Cycles: 1 or 2 (skip) Affects: none

Operation: The incremented value of fr is moved into W. The next instruction word will be skipped if the result was 0.

Coding: INCFSZ 145 fr,0

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following MOVSZ is a single-word instruction.

MOVSZW,--fr Move the decremented value of fr into W and skip if zero

---

Words: 1 Cycles: 1 or 2 (skip) Affects: none

Operation: The decremented value of fr is moved into W. The next instruction word will be skipped if the result was 0.

Coding: DECFSZ 145 fr,0

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following MOVSZ is a single-word instruction.

4.2.29 NOP

NOP No operation

Words: 1 Cycles: 1 Affects: none

Operation: none

Coding: NOP 145

4.2.30 NOT

NOT fr Not fr

Words: 1 Cycles: 1 Affects: Z

Operation: Fr is converted into its one’s complement value. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: COMF 145 fr,1

NOT W Not W

Words: 1 Cycles: 1 Affects: Z

Operation: W is converted into its one’s complement value. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: XORLW 145 0FFh

4.2.31 OR

OR fr,#literal OR literal into fr

Words: 2 Cycles: 2 Affects: Z

Operation: Literal is OR’d into fr via W. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: MOVLW literal
IORWF fr,1

OR fr1,fr2 OR fr2 into fr1

Words:	2	Cycles:	2	Affects: Z

Operation: Fr2 is OR'd into fr1 via W. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: MOVF fr2,0
IORWF fr1,1

OR fr,W OR W into fr

Words:	1	Cycles:	1	Affects: Z

Operation: W is OR'd into fr. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: IORWF 145 fr,1

OR W,#literal OR literal into W

Words:	1	Cycles:	1	Affects: Z

Operation: Literal is OR'd into W. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: IORLW 145 literal

OR W,fr OR fr into W

Words:	1	Cycles:	1	Affects: Z

Operation: Fr is OR'd into W. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: IORWF 145 fr,0

4.2.32 RET, RETFI, RETW

RET Return from subroutine

Words: 1 Cycles: 2 Affects: none

Operation: The next stack value is moved into the program counter. On '5x devices, W is cleared to 0.

Coding: '5x RETLW 145 0
'xx RETURN 145

RETI Return from Interrupt

Words: 1 Cycles: 2 Affects: none

Operation: The next stack value is moved into the program counter. Interrupts are enabled by setting the Global Interrupt Enable bit, GIE (INTCON.7).

Coding: 'xx RETFIE 145

RETW literal1,literal2,... Assemble RET's which load W with literal data

---

Words: ? Cycles: 2 per RETLW Affects: none

Operation: A list of RET's with literal data in the W area is assembled, which can be accessed by JMP PC+W or JMP W instructions. This is useful for lookup tables.

```
Coding:  RETLW 145   literal1
  (RETLW         literal2)
  (RETLW         ...)
```

```
Example: jmp     pc+w    ;Jump to byte at location pc+w

  retw    00100011b      ;Return with w holding appropriate
  retw    00h,01h,02h,03h         ;value
  retw    'Enter cycle count'
```

4.2.33 RL, RR

RL fr Rotate fr left

Words: 1 Cycles: 1 Affects: C

Operation: Fr is rotated left. On entry, C must hold the value to be shifted into the least-significant bit of the fr value. On exit, C will hold the previous most-significant bit of the fr value.

Coding: RLF 145 fr,1

RR fr Rotate fr right

Words: 1 Cycles: 1 Affects: C

Operation: Fr is rotated right. On entry, C must hold the value to be shifted into the most-significant bit of the fr value. On exit, C will hold the previous least-significant bit of the fr value.

Coding: RRF 145 fr,1

4.2.34 SB, SC, SZ

SB bit Skip if bit

Words: 1 Cycles: 1 or 2 (skip) Affects: none

Operation: If bit reads 1, the following instruction word is skipped.

Coding: BTFSS 145 bit

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following SB is a single-word instruction.

Note: The TechTools assemblers define a bit as port.bitposition, as in the following examples:

RA.3 = bit 3 of port A PortB.0 = bit 0 of port B

SC Skip if carry

Words: 1 Cycles: 1 or 2 (skip) Affects: none

Operation: If C is 1, the following instruction word is skipped.

Coding: BTFSS 145 3,0

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following SC is a single-word instruction.

SZ Skip if zero

Words: 1 Cycles: 1 or 2 (skip) Affects: none

Operation: If Z is 1, the following instruction word is skipped.

Coding: BTFSS 145 3,2

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following SZ is a single-word instruction.

4.2.35 SETB, STC, STZ

SETB bit Set bit

Words: 1 Cycles: 1 Affects: none

Operation: Bit is set to 1.

Coding: BSF 145 bit

Note: The TechTools assemblers define a bit as port.bitposition, as in the following examples:

RA.3 = bit 3 of port A PortB.0 = bit 0 of port B

STC	Set carry	
Words: 1	Cycles: 1	Affects: C

Operation: C is set to 1.

Coding: BSF 145 3,0

STZ	Set zero flag	
Words: 1	Cycles: 1	Affects: Z

Operation: Z is set to 1.

Coding: BSF 145 3,2

4.2.36 SKIP

SKIP	Skip the following instruction word	
Words: 1	Cycles: 2	Affects: none

Operation: The following instruction word is skipped.

Coding: BTFSS 145 4,7

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following SKIP is a single-word instruction.

4.2.37 SLEEP

SLEEP	Enter sleep mode	
Words: 1	Cycles: 1	Affects: TO, PD

Operation: The watchdog timer is cleared and the oscillator is stopped. TO is set to 1. PD is cleared to 0.

Coding: SLEEP 145

4.2.38 SNB, SNC, SNZ

SNB bit Skip if not bit

Words: 1 Cycles: 1 or 2 (skip) Affects: none

Operation: If bit reads 0, the following instruction word is skipped.

Coding: BTFSC 145 bit

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following SNB is a single-word instruction.

Note: The TechTools assemblers define a bit as port.bitposition, as in the following examples:

RA.3 = bit 3 of port A PortB.0 = bit 0 of port B

SNC Skip if not carry

Words: 1 Cycles: 1 or 2 (skip) Affects: none

Operation: If C is 0, the following instruction word is skipped.

Coding: BTFSC 145 3,0

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following SNC is a single-word instruction.

SNZ Skip if not zero

Words: 1 Cycles: 1 or 2 (skip) Affects: none

Operation: If Z is 0, the following instruction word is skipped.

Coding: BTFSC 145 3,2

Note: Only one word is skipped by this instruction. To avoid strange results, make sure that any instruction following SNZ is a single-word instruction.

4.2.39 SUB, SUBB

SUB fr,#literal Subtract literal from fr

Words: 2 Cycles: 2 Affects: C, DC, Z

Operation: Literal is subtracted from fr via W. C will be cleared to 0 if an underflow occurred, otherwise C will be set to 1. DC will be cleared or set, depending on whether or not an underflow occurred in the least-significant nibble. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: MOVLW literal
SUBWF fr,1

SUB fr1,fr2 Subtract fr2 from fr

Words: 2 Cycles: 2 Affects: C, DC, Z

Operation: Fr2 is subtracted from fr1 via W. C will be cleared to 0 if an underflow occurred, otherwise C will be set to 1. DC will be cleared or set, depending on whether or not an underflow occurred in the least-significant nibble. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: MOVF fr2,0
SUBWF fr1,1

SUB fr,W Subtract W from fr

Words: 1 Cycles: 1 Affects: C, DC, Z

Operation: W is subtracted from fr. C will be cleared to 0 if an underflow occurred, otherwise C will be set to 1. DC will be cleared or set, depending on whether or not an underflow occurred in the least-significant nibble. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: SUBWF 145 fr,1

SUBB fr,bit Subtract bit from fr

Words: 2 Cycles: 2 Affects: Z

Operation: If bit reads 0, fr is decremented. If fr was decremented, Z will be set to 1 if the result was 0, else Z will be cleared to 0. This instruction is useful for subtracting the carry from the upper byte of a double-byte value after the lower byte has been subtracted.

Coding: BTFSS 3,0
DECF fr,1

Note: The TechTools assemblers define a bit as port.bitposition, as in the following examples:

RA.3 = bit 3 of port A PortB.0 = bit 0 of port B

4.2.40 SWAP

SWAP fr Swap nibbles in fr

Words: 1 Cycles: 1 Affects: none

Operation: The high- and low-order nibbles in fr are swapped.

Coding: SWAPF 145 fr,1

4.2.41 TEST

TEST fr Test fr for zero

Words: 1 Cycles: 1 Affects: Z

Operation: Fr is read and copied back to itself. Z will be set to 1 if the value moved was 0, otherwise Z will be cleared to 0.

Coding: MOVF 145 fr,1

TEST W Test W for zero

Words: 1 Cycles: 1 Affects: Z

Operation: Z will be set to 1 if W is 0, otherwise Z will be cleared to 0.

Coding: IORLW 145 0

4.2.42 XOR

XOR fr,#literal XOR literal into fr

Words: 2 Cycles: 2 Affects: Z

Operation: Literal is XOR'd into fr via W. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: MOVLW literal
XORWF fr,1

XOR fr1,fr2 XOR fr2 into fr1

Words: 2 Cycles: 2 Affects: Z

Operation: Fr2 is XOR'd into fr1 via W. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: MOVF fr2,0
XORWF fr1,1

XOR fr,W XOR W into fr

Words: 1 Cycles: 1 Affects: Z

Operation: W is XOR'd into fr. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: XORWF 145 fr,1

XOR W,#literal XOR literal into W

Words: 1 Cycles: 1 Affects: Z

Operation: Literal is XOR'd into W. Z will be set if the result was 0, otherwise Z will be cleared.

Coding: XORLW 145 literal

XOR W,fr XOR fr into W

Words: 1 Cycles: 1 Affects: Z

Operation: Fr is XOR'd into W. Z will be set to 1 if the result was 0, otherwise Z will be cleared to 0.

Coding: XORWF 145 fr,0

4.3 Sample Projects

This Section contains three sample projects:

Simple Project 187
Table Project 190
Bargraph Project 194

These are not complex projects meant to show off the capabilities of PICs. Instead, they are designed to help new PIC users become familiar with PICs.

The first two projects use the venerable PIC16C54, and show how to accomplish very simple tasks, such as monitoring a button and lighting an LED Display.

The third project uses the more complex, and very popular, PIC16C74. This last project shows how to read an analog input and then display the result on an 8-segment bargraph.

Each sample project is located in the APPS subdirectory of the TDE installation. After opening the project in TDE (Project->Open), you can select the Project option from the Setup menu to customize project settings.

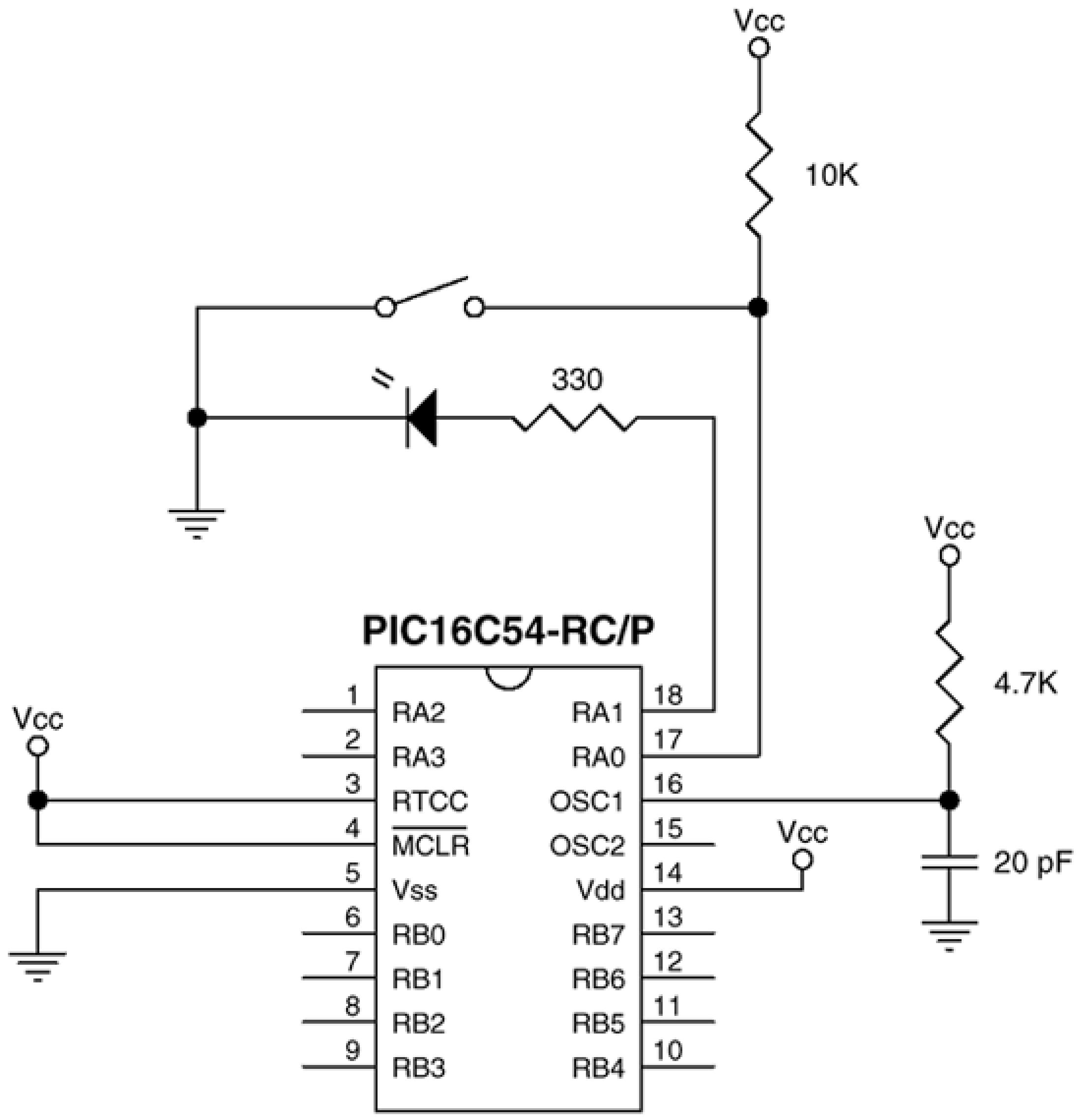
4.3.1 Simple.tpr

PIC Project #1: "Simple.tpr"

This project gives you a very simple introduction to the PIC. The PIC flashes an LED at one speed and then, if the button is pressed, speeds up the flashing.

The PIC itself requires only two parts to operate: a resistor and capacitor to form its RC oscillator. The remaining four parts are the button, the LED, and their resistors.

Before continuing, build the circuit shown below. Only seven parts are necessary, so the task shouldn’t be too laborious. If you don’t have exactly what’s called for, you can substitute a close match (the 4.7K resistor and 20 pF capacitor should not be substituted with smaller values, however).



Note:
When using the ClearView Mathias emulator, you can select any frequency from 32 KHz to 25 MHz using its programmable on-board oscillator.
(For this project we suggest using a frequency between 1 and 4 MHz.)

The following listing is the source code for the project Simple.tpr. It's contained in a file called SIMPLE.SRC, which is located in the APPS subdirectory of the TDE installation.

```
; PIC EXAMPLE PROGRAM: SIMPLE.SRC
; June 8, 1992
;
; This 16-word program is a very simple PIC application.  Its only purpose
; is to flash an LED at one of two rates.  Normally, the LED flashes slowly.
; However, if bit 0 of port A (RA0) is grounded, the LED flashes roughly twice
; as fast.
;
; As the program is written, it expects to run on a PIC16C54-RC/P.  A push
; button connects RA0 to ground (don’t forget a pull-up resistor from RA0 to
; Vdd, perhaps 10K).  A current-limited LED is connected from RA1 to ground
; (330 ohms work well).  For an oscillator, a 20 pF capacitor to ground and
; a 4.7K resistor to Vcc are connected to OSC1 (OSC2 is left open).  RTCC and
; MCLR should be tied high.  Lastly, power and ground are connected to Vdd
; and Vss, respectively.
;
; If you’re looking for a simple introduction to the PIC, this program should
; help.  Among other things, it shows the following basic concepts:
;
;      * Setting device options
;      * Setting the reset vector
;      * Setting I/O pins as inputs or outputs
;      * Using global labels
;      * Using local labels (see “:Loop” below)
;
; After you build the circuit described above, open the project file and select Run, the
; LED should start flashing as soon as TDE's  status line displays "Running".

; START OF PROGRAM

; Set the device type, oscillator type, watchdog timer status, and code
; protect status

             DEVICE PIC16C54,RC_OSC,WDT_OFF,PROTECT_OFF

             RESET  Start        ;Set reset vector to address at Start
                                 ;(PIC will jump to this when reset)

Count0 equ   10h                 ;Assign labels to registers 10h & 11h
Count1 equ   11h

             clr    Count0
             clr    Count1
             clr    RA           ;Clear port before setting direction
                                 ;register

Start  mov   !RA,#00000001b ;Set data direction register for port A
                                 ;(make bit 0 an input)
;
; Loop 65536 times, then invert the LED
;

:Loop  djnz  Count0,:Loop  ;Decrement Count0 until it reaches zero
             djnz   Count1,:Loop ;Decrement Count1.  If it’s not zero,
                                 ;jump back to :Loop

             xor    RA,#00000010b;Invert the LED (bit 1 of port A)

;
; Check button status.  If it’s pressed, skip the additional delay and jump
; back to the first loop
;

ChkBtn jnb   RA.0,Start:Loop ;Jump to 1st loop if button is pressed
```

```
                                ;(button is low when pressed)

:Loop  djnz   Count0,:Loop  ;Decrement Count0 until it reaches zero
              djnz   Count1,:Loop  ;Decrement Count1.  If it’s not zero,
                                ;jump back to :Loop

              jmp    Start:Loop    ;Jump back to first loop
```

The first step is to assemble the source code into hex code. To assemble the program, click the Build button on the menu bar or select the build option from the Run menu.

If the assembler finds any errors, it will give the line number and description of each error. Before continuing, you’ll need to correct any errors and reassemble the program (of course, this should not occur, unless you typed the program yourself and perhaps made a typo).

If the program assembles correctly, the assembler will produce a hex file, called SIMPLE.OBJ, in the Apps sub-directory.

After running your program satisfactorily, you are ready to program your PIC with the HEX file :

SIMPLE.OBJ.

Once you have successfully programmed the PIC, you're ready to try it in your circuit. Make sure the power to your circuit is off, and then plug in the PIC.

When you turn the power on, the LED should be flashing. If you push the button, the LED should blink more quickly.

If your project appears "dead", double-check the circuit (power, RC oscillator, connections, etc).

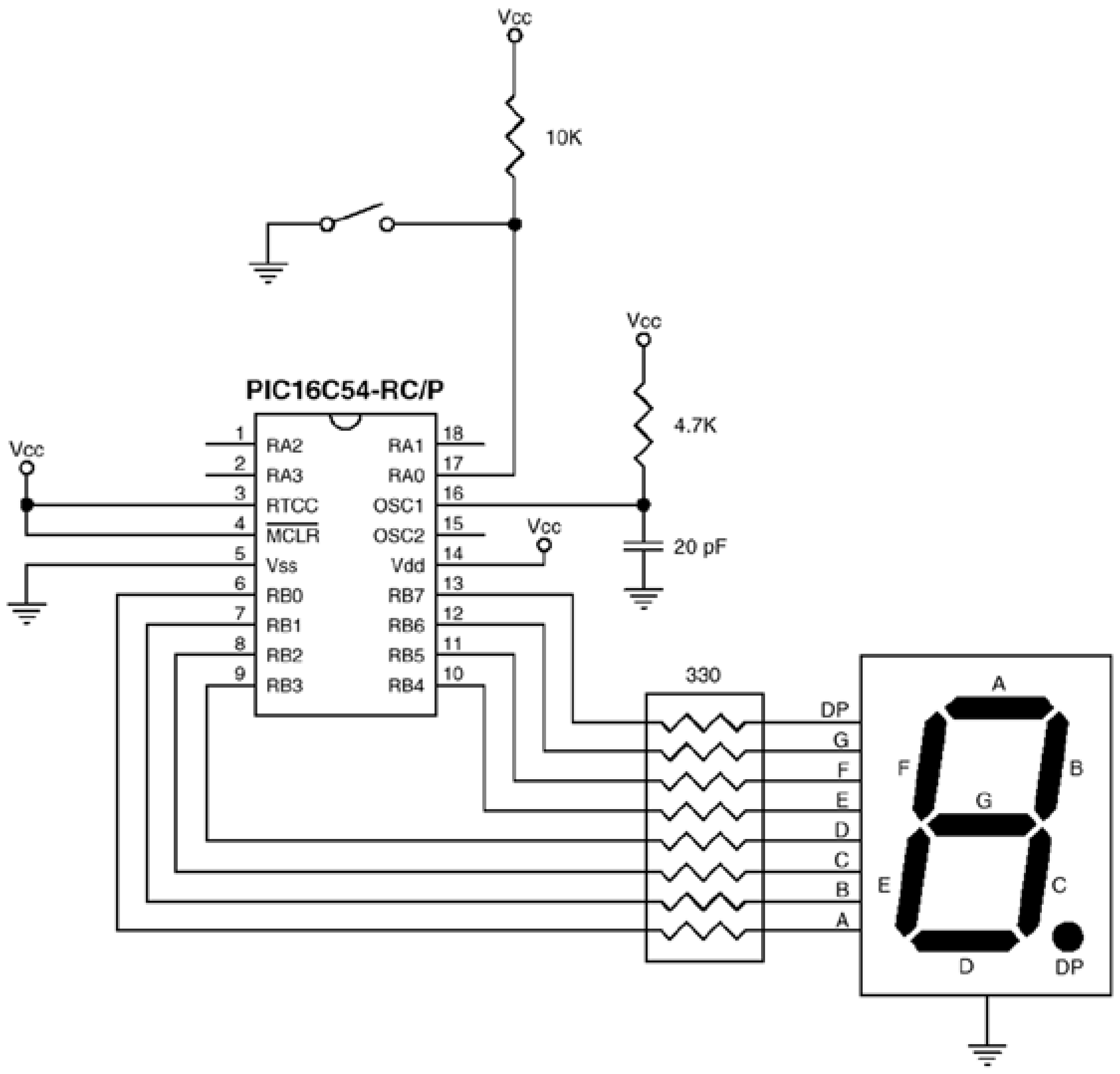
4.3.2 Table.tpr

PIC Project #2: "Table.tpr"

Like the first project, this project gives you a simple introduction to the PIC. But, it also demonstrates the useful topics of how to debounce push button inputs and how to make a lookup table.

The schematic below shows the circuit. Whenever the button is pressed, the value shown on the 7-segment display increments.

The PIC itself requires only two parts to operate: a resistor and capacitor to form its RC oscillator. The remaining four parts are the button, the 7-segment display (common cathode), a 10K resistor, and a resistor pack (330-ohm, 8 isolated resistors).



Note:
When using the ClearView Mathias emulator, you can select any frequency from 32 KHz to 25 MHz using its programmable on-board oscillator.
(For this project we suggest using a frequency between 1 and 4 MHz.)

The following listing is the source code for the project Table.tpr. It's contained in a file called TABLE.SRC, which is located in the APPS sub-directory of the PIC Tools Diskette.

```
; PIC EXAMPLE PROGRAM: TABLE.SRC
; August 19, 1992
;
; This 44-word program is a simple PIC application that shows some useful
; routines.  Its purpose is to monitor a button and display a digit on a
; 7-segment display.  When the button is pressed, the digit increments.
;
; As the program is written, it expects to run on a PIC16C54-RC/P.  A push
; button connects RA0 to ground (don’t forget a pull-up resistor from RA0 to
; Vcc, perhaps 10K).  A 7-segment display (common cathode) is connected to
; port B (segment A to bit 0, segment B to bit 1, etc.).  For an oscillator,
; a 20 pF capacitor to ground and a 4.7K resistor to Vcc are connected to
; OSC1 (OSC2 is left open).  RTCC and MCLR should be tied high.  Lastly,
; power and ground are connected to Vdd and Vss, respectively.
;
; Among other things, this program shows the following basic concepts:
;
;      * Setting device options
;      * Setting the reset vector
;      * Setting I/O pins as inputs or outputs
;      * Using labels
;      * Debouncing a push button input
;      * Reading from a lookup table
;
; If you build the circuit as described above,  the display should show a “0” when you select t

; START OF PROGRAM

; Set the device type, oscillator type, watchdog timer status, and code
; protect status

             DEVICE       PIC16C54,RC_OSC,WDT_OFF,PROTECT_OFF

             RESET  Start  ;Set reset vector to address at Start
                                  ;(PIC will jump to this when reset)

Count0 equ   10h                  ;Register labels
Count1 equ   11h
Number equ   12h

Flag         equ    13h.0         ;Bit label (bit 0 of register 13h)
Button equ   RA.0                 ;Port labels (bit 0 of port A)
Display      equ    RB                   ;(entire port)

Start        clr    Number
             clr    Flag
             clr    RA
             clr    RB

             mov    !RA,#00000001b       ;Set data direction register for port A
                                         ;(make bit 0 an input)

             mov    !RB,#00000000b       ;Set data direction register for port B
                                         ;(make all bits outputs)

; Main loop

Digit        mov    W,Number      ;Move Number into W

             call   GetPattern           ;Get bit pattern for digit
             mov    Display,W            ;Show digit

             jnb    Flag,UpLoop   ;If flag is clear, check for 2048 reads
```

```
                                        ;of button not pressed

             jb     Button,Digit  ;Jump if flag set, but button not
                                        ;pressed

; If button pressed after not being pressed for 2048 reads, then increment number

             inc    Number        ;Increment number
             cjne   Number,#10,Clear

             clr    Number        ;If Number reached 10, reset to 0

Clear  clr   Flag
             jmp    Digit

; Set flag if button not pressed for 2048 reads.  Faster oscillator speeds
; may require more than 2048 reads to properly debounce button inputs.

UpLoop clr   Count0
             mov    Count1,#08

Loop         jnb    Button,Digit  ;If button pressed before 2048 reads,
             djnz   Count0,Loop   ;jump back to main loop
             djnz   Count1,Loop
             setb   Flag                ;Set flag if button not pressed
             jmp    Digit

; 7-segment lookup table

GetPattern   jmp    PC+W                ;Jump to appropriate bit pattern,
                                        ;load W with pattern, and then return
                                        ;to calling routine.

             retw   3Fh,06h,5Bh   ;Bit patterns for digits 0-9
             retw   4Fh,66h,6Dh
             retw   7Dh,07h,7Fh
             retw   6Fh


;            retw   'This is text'      ;Sample text lookup table (not used by
                           ;this program)
```

The first step is to assemble the source code into hex code by selecting Build from TDE's Run menu.

If the assembler finds any errors, it will give the line number and description of each error. Before continuing, you'll need to correct any errors and re-assemble the program (of course, this should not occur, unless you typed the program yourself and perhaps made a typo).

If the program assembles correctly, the assembler will produce a hex file, called TABLE.OBJ, in the APPS directory.

You can now select Run, and your circuit will operate as if you had already inserted a programmed PIC.

Once everything is operating to your satisfaction, you can program your PIC with the HEX file :

TABLE.OBJ.

Once you have programmed the PIC, you're ready to try it in your circuit. Make sure the power to your circuit is off, and then plug in the PIC. When you turn the power on, the display should show "0". If you push the button, the value on the display should increment.

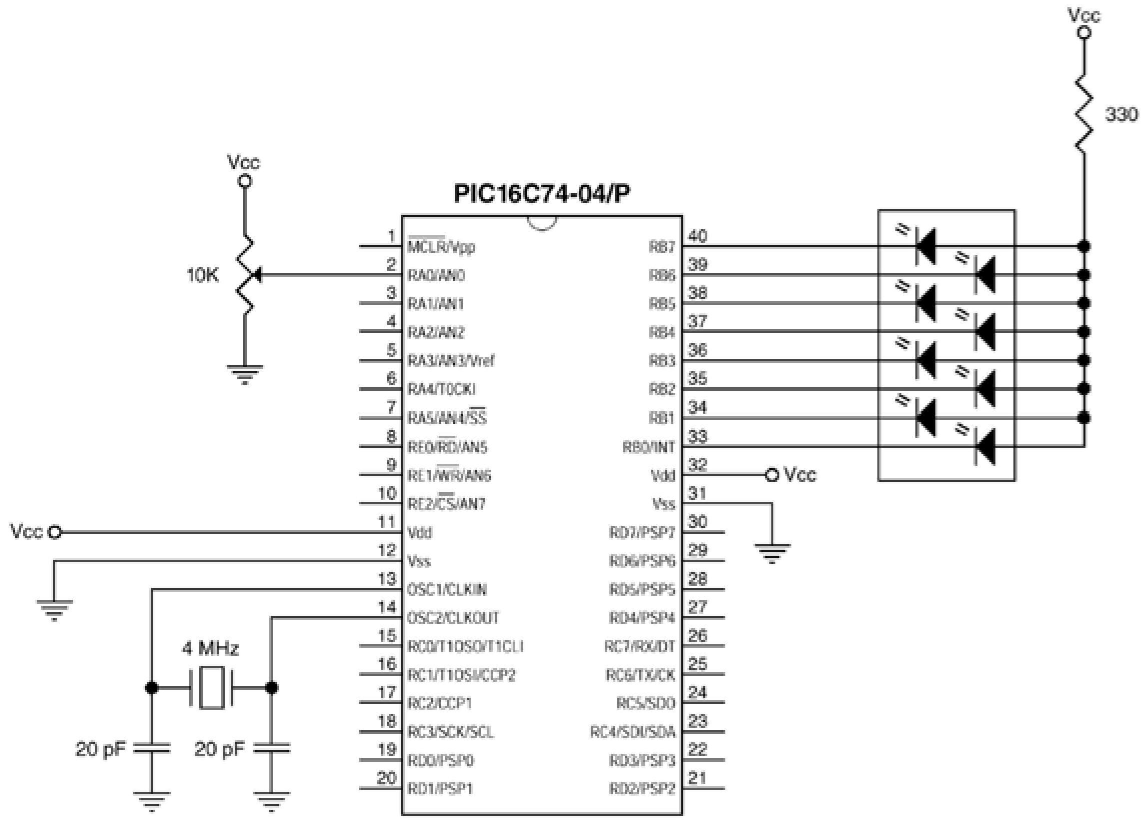
If your project appears "dead", double-check the circuit (power, RC oscillator, connections, etc).

4.3.3 Bargraph.tpr

PIC Project #3: "Bargraph.tpr"

This project is relatively simple, but it uses the more complex PIC16C74. The ’74 has many good features, which have made it a popular device. This project demonstrates how to read a single analog input and then display the result on an 8-segment LED bargraph.

The schematic below shows the circuit. The PIC itself requires just three parts to operate: a 4-MHz crystal and two capacitors to form its oscillator. The remaining three parts are a 10K potentiometer, a 330-ohm resistor, and an 8-segment LED bargraph (common anode).



Note:
When using the ClearView Mathias emulator, you can select any frequency from 32 KHz to 25 MHz using its programmable on-board oscillator.
(For this project we suggest using a frequency of 4 MHz.)

The following listing is the source code for the project. It's contained in a file called BARGRAPH.SRC, which is located in the APPS directory of the PIC Tools Diskette.

```
; SAMPLE PROGRAM: BARGRAPH.SRC
;
; February 9, 1996
;
; This program uses a PIC16C74 to convert an analog value into a reading
; on an 8-segment LED bargraph.  The analog input is connected to
; AN0 (pin 2) on the '74.  The LEDs are active-low, and are connected to
; Port B (pin 33 is first LED, pin 34 is second LED, etc).
;
; The program is written to only light one LED at a time, but could easily
; be changed to light multiple LEDs (all LEDs up to the reading).  Having
; only one LED on insures low current draw, which may be important if
; you're powering the circuit with an emulator.
;

            DEVICE PIC16C74,WDT_OFF,XT_OSC
```

```
Value  equ    20h                 ;Define two bytes for later use
Count  equ    21h

              org    000h                ;Set up reset vector at location 000h
              jmp    Init                ;On reset, jump to beginning of program

Init          org    005h                ;Program must begin at/after location 005h

              setb   RP0                 ;Switch to register bank 1
              mov    ADCON1,#00h  ;Make port A all analog
              mov    TRISA,#0FFh  ;Make port A all inputs
              mov    TRISB,#00h          ;Make port B all outputs
              clrb   RP0                 ;Switch back to register bank 0

              mov    ADCON0,#00h  ;Set ADC channel, conversion speed, etc.
              setb   ADCON0.0            ;Activate ADC

Main          setb   ADCON0.2            ;Start analog conversion
Check  jb     ADCON0.2,Check      ;Loop until conversion done
              mov    Value,ADRES  ;Copy analog value into Value

              clc                        ;Divide Value by 32.  Gives 0-7 for
              rr     Value        ;analog readings of 0-255.
              clc
              rr     Value
              clc
              rr     Value
              clc
              rr     Value
              clc
              rr     Value

              mov    Count,#01h          ;Start Count at first bit position (bit 0)
Loop          cje    Value,#00h,Leds     ;Jump if Value is 0
              clc
              rl     Count        ;Rotate Count left one bit
              dec    Value        ;Decrement Value
              jmp    Loop                ;Loop until Value is 0

Leds          xor    Count,#0FFh  ;LEDs are active-low, so invert Count
              mov    PortB,Count  ;Output Count on port B LEDs

              jmp    Main                ;Repeat the process
```

The first step is to assemble the source code into hex code. To assemble the program, choose Build from TDE's Run menu.

If the assembler finds any errors, it will give the line number and description of each error. Before continuing, you'll need to correct any errors and re-assemble the program (of course, this should not occur, unless you typed the program yourself and perhaps made a typo).

If the program assembles correctly, the assembler will produce a hex file, called BARGRAPH.OBJ, in the APPS directory.

You can now select Run, and your circuit will operate as if you had already inserted a programmed PIC.

Once everything is operating to your satisfaction, you can program your PIC with the HEX file :

BARGRAPH.OBJ.

Once you have programmed the PIC, you're ready to try it in your circuit. Make sure the power to your circuit is off, and then plug in the PIC.

When you turn the power on, any one of the LEDs might be turned on, depending upon the position of the potentiometer. If you turn the potentiometer, then you should see a corresponding change on the LEDs. For a quick test, you can also just connect the analog input to GND or to +5V, in which case the first or last LED should turn on.

If your project appears "dead", double-check the circuit (power, RC oscillator, connections, etc).

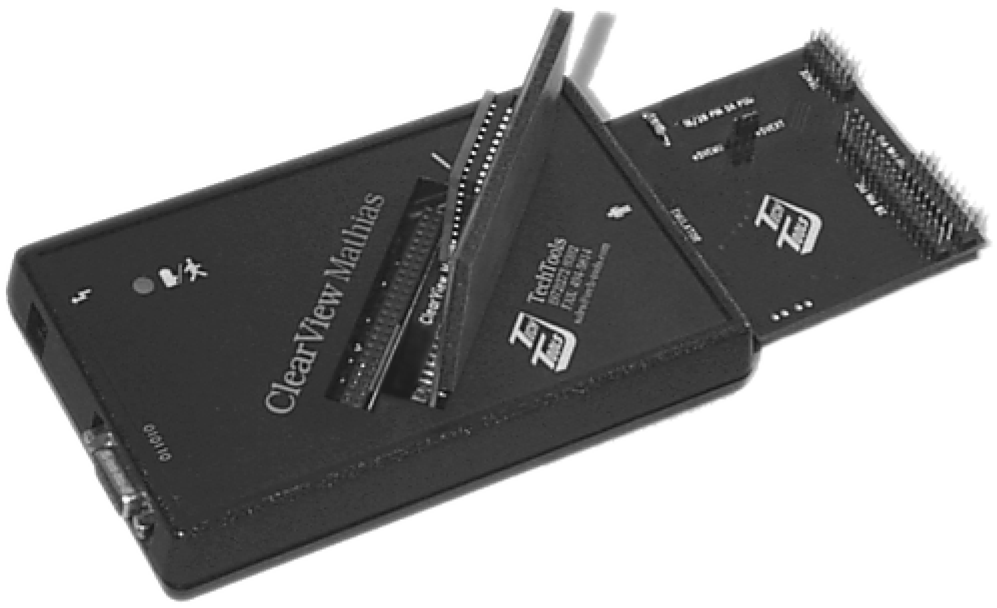
ClearView Mathias Hardware

Part V

5 ClearView Mathias Hardware

ClearView Mathias 199
System Requirements 200
Packing List 200
Hardware Features 201
Hardware Installation 201
Power Supply Considerations 203
MCLR Pull-Up Resistor 204
Optional Modules 205

5.1 ClearView Mathias



ClearView Mathias is an in-circuit emulator for PIC16Cxx micro-controllers. It plugs in place of a PIC in a target circuit, and allows you to run your code in-circuit at full hardware speeds.

ClearView Mathias supports full debugging features, such as stepping, breakpoints, register modification, and trace. These features allow you to see and affect the internal operation of the PIC, while it executes code.

ClearView Mathias supports 16C5x and 16Cxx parts. This is accomplished through the use of modules that select the family (5x/xx) and family "member" (16C54, 16C57, 16C74, 16F877, etc).

The use of modules allows us to keep the emulator affordable, while still offering the options people want. If you started with the 5x module, for instance, you can later add 16C74 support without spending too much.

5.2 System Requirements

To use ClearView Mathias, you will need the following items:

IBM AT or compatible computer with mouse
VGA or higher Display
CDROM drive
Serial port
2 megabytes of RAM (minimum + O/S requirements)
Windows 9x, NT, 2000, XP, XPpro

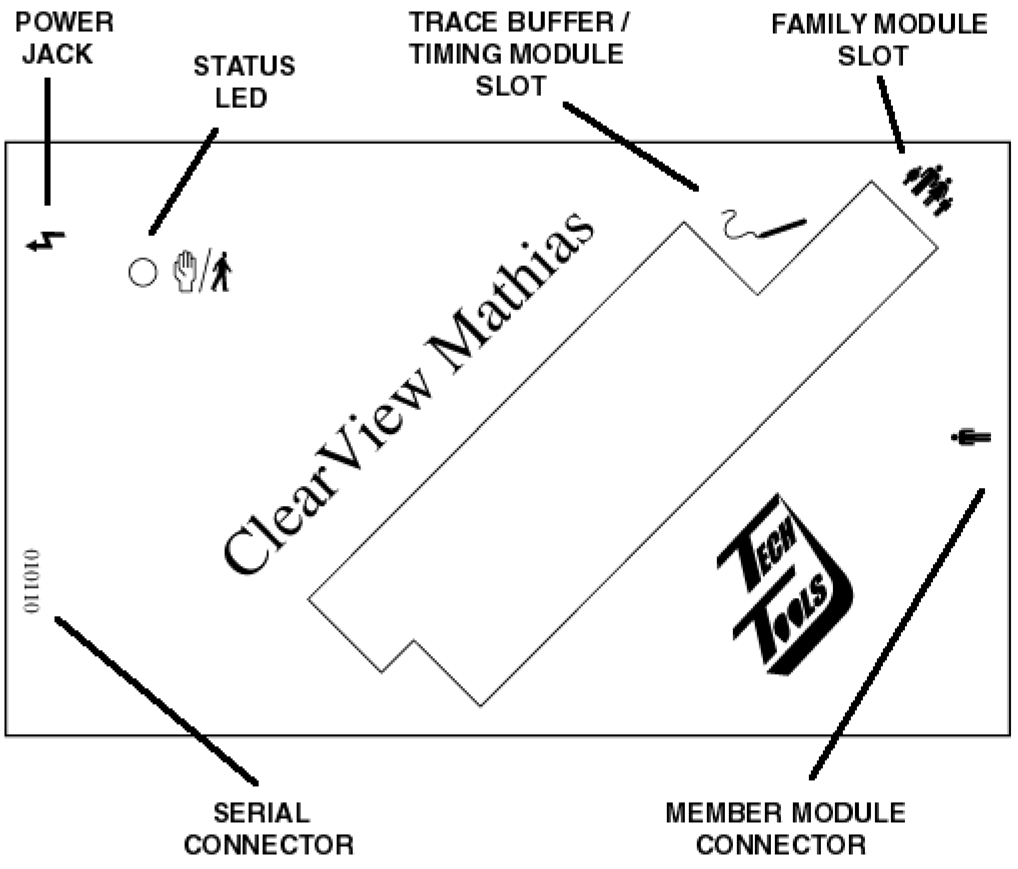
5.3 Packing List

The ClearView package should contain the following items. If any are missing, please let us know.

1. ClearView Mathias base unit
2. 1 or more PIC family modules (5x or xx family module) as ordered.
3. 1 or more PIC member modules (5x, 61, 74, 84, etc...) as ordered.
4. Trace buffer module (optional)
5. Timing module (optional)
6. 18-pin ribbon cable
7. 28-pin ribbon cable
8. 40-pin ribbon cable
9. Serial cable (DB-9 male to DB-9 female)
10. DB25 to DB9 serial adapter
11. Power supply *
12. TechTools CDROM
13. PIC16Cxx Development Tools Manual (this manual)

* Power supplies are only shipped with orders to the United States, Canada, and Japan. If your order was shipped to another country, you will need to obtain a power supply with the proper output voltage and current: 7-8 VDC, 500 mA (outer conductor = Ground)

5.4 Hardware Features



Power Jack: Accepts power from power supply 203 (7-8 VDC, 500 mA).

Status LED: Red when emulation is stopped, green when running. Also used in power-up test sequence: green when power is applied, red when unit is ready to use.

Trace Buffer Slot: Connector for optional trace buffer and timing modules 205.

Member Module Connector: Connector for PIC member module. Provides emulation for specific PIC (5x, 61, 71, 84, etc).

Family Module Slot: Connector for PIC family module. Family module determines if ClearView will emulate 16C5x (12 bit core) or 16Cxx (14 bit core) parts.

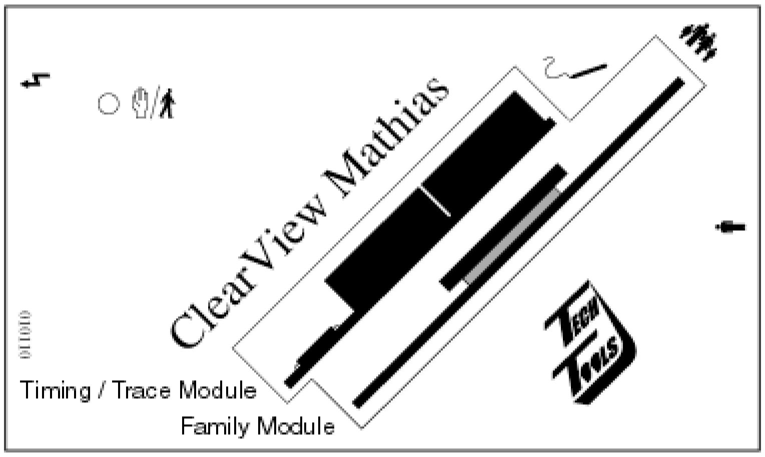
DB-9 Connector (female): Connects to PC serial port.

5.5 Hardware Installation

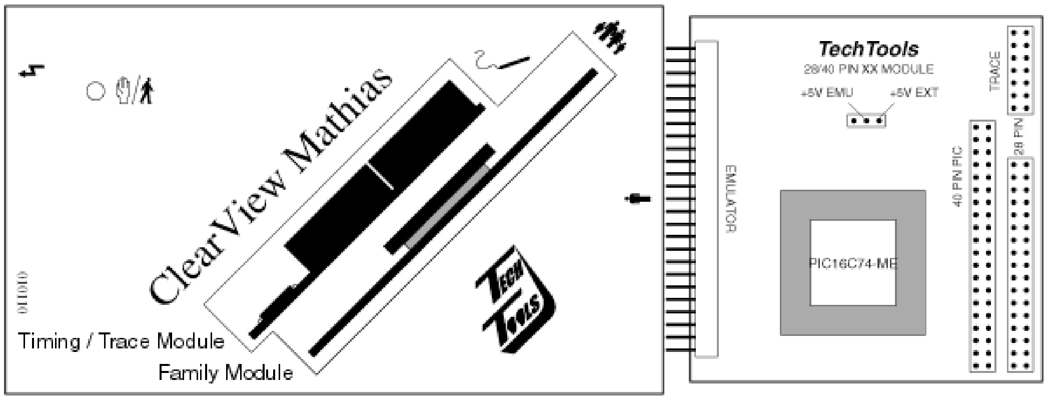
To install ClearView, follow these steps:

1) If you purchased the optional trace buffer module or timing module, plug it into ClearView's trace buffer slot. When plugged in properly, the components on the module should face the words "ClearView Mathias" on the emulator.

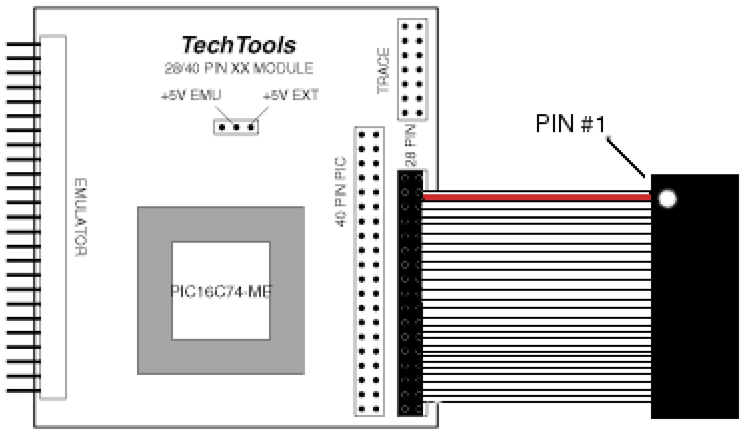
2) Plug a "5x" or "xx" family module into ClearView's family module slot. As with the trace buffer module, the components on the module should face the words "ClearView Mathias."



3) Plug a PIC member module into the right side of the emulator. If the member module is for "5x" PICs, then you must have a "5x" family module plugged into the family slot (see step #2). If the member module is for any "xx" PIC (61, 71, 84,...), then you must have an "xx" module plugged into the family slot.



4) Plug an appropriate ribbon cable onto the member module. The module will have connectors for 18-, 28-, and/or 40-pin cables. The type of cable used will depend on which PIC you plan to emulate (for 16C61, an 18-pin cable; for 16C73, a 28-pin cable; etc).



5) Plug the power supply into an AC outlet.

6) Plug the power supply cord into ClearView's power jack.

The LED on ClearView should be green when power is applied. After a few seconds of internal testing, the LED should change to red; this indicates that the unit is in emulation mode, but that execution is stopped.

7) Plug the female end of the serial cable into an available serial port on your PC. If your serial port has a DB-25 connector, you'll need to use a DB-25 to DB-9 adapter.

8) Plug the male end of the serial cable into ClearView's DB-9 connector.

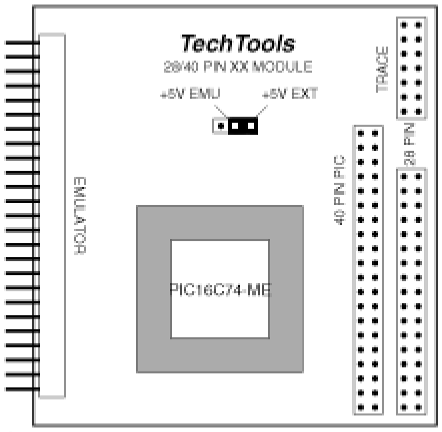
Your ClearView should now be ready to use.

If you need to change any of the plug-in modules, please be sure to disconnect any power source before making the change.

Failure to disconnect power before replacing modules can result in damage to the modules, emulator, and/or target circuit.

5.6 Power Supply Considerations

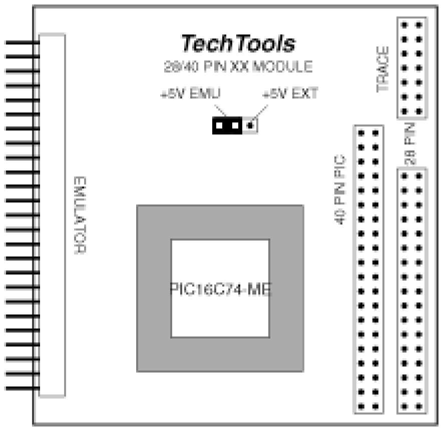
Every Mathias Member Module provides a simple jumper which may be used to short the power plane of both emulator and target together. This may be useful if you need to power a small target circuit from the emulator, or if you want to power the emulator from the target circuit. In either case, please note the following current limitations and requirements: when powering a target circuit, the emulator cannot provide more than approximately 50 mA; conversely, when powering the emulator from a target circuit, the emulator needs approximately 500 mA.



To power target circuit from emulator or to power emulator from target circuit:
(power supplies connected)

To connect the power planes of both devices together, insert the power jumper in the position on the right.

If both systems are connected and providing power, damage to the emulator and/or target circuit may result.



To power each device independently:
(power supplies not connected)

To power the emulator and target independently, isolate their power planes. Either remove the jumper or store it in the left position.

5.7 MCLR Pull-Up Resistor

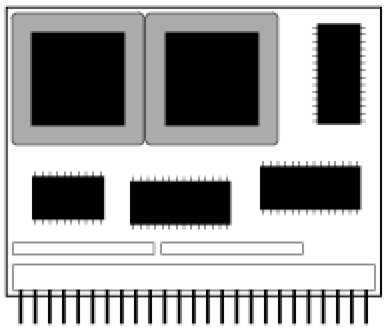
ClearView Mathias has a 100K pull-up resistor on the MCLR input. This ensures that the MCLR input will be inactive when it’s not specifically driven low, thus preventing random resets of the emulated PIC.

5.8 Optional Modules

As mentioned in the hardware installation text, there are two optional modules for ClearView Mathias; these modules are described below:

Trace Buffer Module:

This module keeps a record of each instruction executed by the emulator. For each instruction, the record shows the location in the trace buffer memory, the value of the program counter, the state of external trace inputs, and the original line of source code. The trace module also provides information about how many cycles and how much time is taken by each instruction.

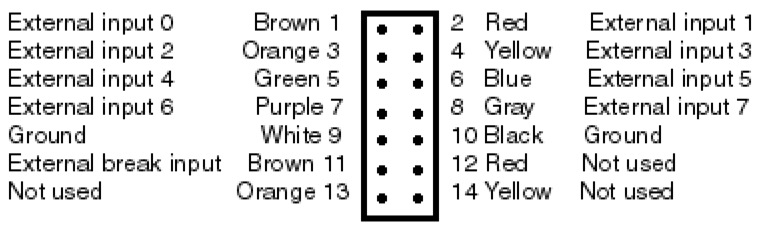


The trace buffer memory is 16K "deep," resulting in a capacity of approximately 16,384 Microchip instructions. Some instructions, such as JMP and GOTO, take two processor cycles, and therefore take two locations in the trace buffer memory. In addition, some TechTools instructions utilize two or three Microchip instructions, and will therefore take multiple locations in the trace buffer (this is also true of high-level compiled languages, such as C).

On the hardware side, the trace buffer adds eight external trace inputs. The status of the eight trace inputs is recorded along with each instruction in the trace buffer. By connecting these inputs to your circuit, you can record the activity of your circuit as it relates to the emulated PIC.

In addition to eight trace inputs, the trace module adds one more signal: an external break input. This input acts like an interrupt, and may be connected to your circuit. If the external break input transitions from high to low, then the emulator will halt execution. Please note, however, that it takes the emulator two or three cycles to react to an external break. This is because it takes one or two cycles to recognize the input, and another cycle to halt the emulator (to be recognized in the first instruction cycle, the break input must be valid at least 60ns before the start of the cycle).

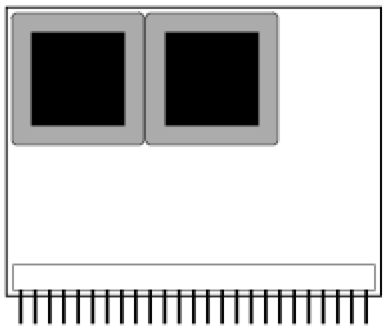
The hardware inputs described above are present on the trace input header, which is found on every PIC member module 201. A pin-out of this header is shown below:


External input 0 Brown 1 2 Red External input 1
External input 2 Orange 3 4 Yellow External input 3
External input 4 Green 5 6 Blue External input 5
External input 6 Purple 7 8 Gray External input 7
Ground White 9 10 Black Ground
External break input Brown 11 12 Red Not used
Not used Orange 13 14 Yellow Not used

The trace buffer module includes micro-grabber test leads which can be attached to the trace header on any of our member modules.

Timing Module:

This module is a subset of the trace module; it provides execution cycle and time information, just like the trace module. However, the timing module does not include any of the other trace functions (execution history, external inputs, etc).



Since it does not support any external inputs, the timing module does not include a 14-conductor cable.

The timing module is much less expensive than the trace module, and many customers are happy with the functions it provides.

Index

- ! -

! 120

- . -

.obj 5
.QWC 5

- 7 -

7-segment display 190

- 8 -

8-segment LED bargraph 194

- A -

ADD 148
Add a Varible to the Watch Window 36
Add fr2 into fr1 148
Add literal into fr 148
Add W into fr 148
Add Watch button 108
Add Watch Selected 108
ADDB 148
ADDCF 145
ADDDCF 145
Adding a File 48
Adding Files 51
Adding items by Highlighting 108
Adding items from Source Code 108
Adding Items from the Variables Window 108, 109
ADDLW 145
ADDWF 145, 148, 164
Advanced 20
Alt+E
- I 13
- S 17
- W 15

Alt+O
- G 17
- H 9
- N 17
- O 6
- S 15

Alt+R
- R 19
- S 17

analog input 194
AND 150
AND fr2 into fr1 150
AND literal into fr 150
AND literal into W 150
AND W into fr 150
ANDLW 145
ANDWF 150
Animate 74
Animate Over Execution 74
Animation 36
Animation Step Delay 62
Animation-Mode Execution 74
Any Data 121
Assemble RET's which load W with literal 178
assembler 126
Auto Backup 62
Auto Increment 17
Auto Indent 62
Auto Load last Project 62
Auto Run 23
AutoRun 23

- B -

B 145
basic functions 27
baud rate 55
BC 145
BCF 145, 157
BDC 145
Beep on Errors 62
Beep on Stop 62
binary 97, 105, 112
Bit Pattern 121
bit-oriented file register operations 145
BNC 145
BNDC 145
BNZ 145
Bondouts and Speed Limitations 59

Break On 120
Break on Stack Over/Underflow 57
Break on Timeout 57
Breaking on data 124
Breakpoint 95
Breakpoints 78
BSF 145, 180
BTFSC 145, 182
BTFSS 145, 179, 181
BUILD 68, 131
Building the Project 32, 68
button 187
Bytecraft C 128
Bytecraft C to obj 128
Bytecraft LINKER 128
byte-oriented file register operations 145
BZ 145

- C -

cable connector 23
Cable Pinout 20
call 145, 151, 190
Call subroutine 151
CCS 130
Changing the radix of items 115
Child Nodes 25
CJA 152
CJAE 152
CJB 153
CJBE 153
cje 154, 194
cjne 155, 190
clc 157, 194
Clear 156
Clear bit 157
Clear carry 157
Clear trace buffer 116
Clear W 156
Clear watchdog timer 156
Clear zero 157
Clearing All Breakpoints 78
clr 156, 187, 190
clrb 157, 194
CLRC 145
CLRDC 145
CLRF 145, 156
CLRW 145, 156
CLRWDT 145, 156
CLRZ 145
CLZ 157
COD 130, 131
COD ONLY 131
Code Protection 15
Code Window 95
com port 55
COMF 145, 172, 176
Command-line Operation 23
COMPILE TO OBJECT 127
COMPILE TO OBJECT MODE 128
compiler 126, 128, 129
Compiling and Executing Code 44, 68
Configuration Word 15
control file 14
Copy 85
Creating a Multiple-file Project 50
Creating a Single-file Project 46
CSA 157
CSAE 157
CSB 159
CSBE 159
CSE 160
CSNE 161
Ctl+R 19
Ctrl-C 85
Ctrl-F 89
Ctrl-F9 78
Ctrl-V 85
Ctrl-X 85
Ctrl-Z 88
current execution line 62
Current Line Position 62
Cut 85
  and paste 85
CVASM_OLD 127
CVASM16 127

- D -

Data Breakpoints 44, 118, 120
DBP 118, 120, 121, 123, 124
debounce push button inputs 190
Debouncing a push button input 190
dec 162, 194
DECF 145, 162, 172
DECFSZ 145, 162, 175

decimal 97, 105, 112
Decrement fr 162
Decrement fr and jump if not zero 162
Decrement fr and skip if zero 162
DECSZ 162
DEVICE 23, 32, 187, 190, 194
Device Settings 32, 59
Device Type 59
Disable EE Data Editor 7
Disable Option Editing.. 14
Disable Option Editing... 7
Disable Read Functions 7
Disable Task Confirmation Messages 7
Disabled 57
djnz 162, 187, 190

- E -

Edit 82
Edit - Configuration Word 15
Edit - ID Locations 13
Edit and Add 109
Edit menu 85, 88, 89
Editor Window 82
Emulator 55
Enable 120
Enable and Break On options 120
Enable Blank Check 6
Enable Calibration 6
Enable Code 6
Enable EE Data 6
Enable Erase 6
Enable Fuses 6
Enable ID Locations 6
enable option editing 14
Enable Serial Numbering 10
Enable Verify 6
Enable/disable trace 116
Encode as RETLW 10
Enter sleep mode 181
Enter source code 85
Environment Preferences 62
equ 187, 190, 194
error list 68
Error List Window 93
Errors 93
exclamation point 120
Executing Code at Full Speed 70
Execution History 60
Execution point 77, 95
Exit Code 23
Exit Codes 24
External Break 60
External Inputs 60

- F -

F10 41, 76
F3 89
F4 32, 68, 131
F5 34, 70
F6 34, 36, 40, 71
F7 73
F8 35, 71
F9 36, 38, 43, 74
File menu 51, 66, 67
file node 51, 127, 128, 129, 130, 131
File Node Tool 48, 51
File Register options 120
File Registers Window 105
Filename 23
Files Window 105
FILL ALL 105
Find Next 89
Find what 89
Finishing the Project 43
FLASH Programming Pulse Width 12
Font Options 62
forcing a read 97
Frequency 59

- G -

gang 17
Go 70
Go to Data Breakpoint 124
Go to Marker 91
GOTO 145, 164

- H -

Halt 71
Halting Execution 71
Hardware Control Signal (TVDDEN) 21
Hardware Settings 31

hex 5, 97, 105, 112
HEX ONLY 131
highlighted yellow 103
Hi-Tech C 129

- I -

ICP 20
ID locations 7, 13
Ignore EE DATA 9
Ignore FUSE record 9
Ignore ID record 9
Ignore MCU record 9
IJNZ 163
inc 163, 190
INCF 145, 163, 172
INCFSZ 145, 163, 175
In-Circuit Programming 20
In-Circuit Programming Cable Pinout 20
Include Calibration Data 9
Include Constants 123
Include EE Data 9
Include FUSE Settings 9
Include MCU Type 9
Increment fr 163
Increment fr and jump if not zero 163
Increment fr and skip if zero 163
INCSZ 163
INHX32 5
INHX8M 5
INS 108
IORLW 145, 176, 184
IORWF 145, 176

- J -

jb 164, 190, 194
JC 164
jmp 164, 187, 190, 194
jnb 165, 187, 190
JNC 165
JNZ 166
jump if above 152
jump if above or equal 152
jump if below 153
jump if below or equal 153
Jump if bit 164
Jump if carry 164
jump if equal 154
Jump if not bit 165
Jump if not carry 165
jump if not equal 155
Jump if not zero 166
Jump if zero 166
Jump to address 164
Jump to PC+W 164
Jump to W 164
JW 15
JZ 166

- L -

Labels Window 103
LAST HEX 17
LCALL 145, 166
LED 187
Legend - Programming Tasks 25
LGOTO 145
LIBRARY 129
Link 51
LINKER file 128
LINKING 127, 128, 129
literal and control operations 145
Little Endian 17
LJMP 166
Loading a Source File 29
Long call ('5x) 166
Long call ('xx) 166
Long jump ('5x) 166
Long jump ('xx) 166
Long set ('5x) 166
Long set ('xx) 166
Low Voltage Programming 20
LSET 166

- M -

Manual Serial Numbers 17
Match case 89
Match On options 121
Match whole word 89
Mathias 55
MCLR 15
MCU SECTION 6

MCU Sections 6
Microchip to CVASM16 145
Misc. Options 7
Mode 31
Mode of Operation 55
Modifying item values 112
Modifying Values of watched items 108
Monitor a Self Powered Target 21
mov 169, 171, 172, 187, 190, 194
MOV [to I/O Control] 171
MOV [to OPTION] 169
MOV [to register] 169
MOV [to W] 172
MOV [using Expressions] 172
MOVB 174
Move bit2 to bit1 174
Move fr into OPTION 169
Move fr into W 172
Move fr2 into fr1 169
Move fr-W into W 172
Move literal into fr 169
Move literal into OPTION 169
Move literal into W 172
Move not bit2 to bit1 174
Move not fr into W 172
Move the decremented value of fr into W 172
Move the incremented value of fr into W 172
Move the left-rotated value of fr into W 172
Move the nibble-swapped value of fr into W 172
Move the right-rotated value of fr into W 172
Move W into fr 169
Move W into OPTION 169
MOVF 145, 172, 184
MOVFW 145
MOVLW 145, 172
MOVSZ 175
MOVWF 145, 164, 169
MPASM 127
MPASM to OBJ 128
MPASM-to-OBJ 127
MPLABC 129
MPLINK 127, 128
multi-file project 50
multiple opcode instruction 145

- N -

NEGF 145
new project 46
No operation 175
Node Build Settings 46, 50
NOP 145, 175
NOT 176
Not fr 176
Not W 176
Number of Locations 10

- O -

OBJECT 129
Off Line 55
Open HEX File - Override HEX Options 9
Opening the Data Breakpoints Dialog 118
Opening The Project 28
Operation 15
Operations 6
Operator Notification 21
OPTION 145, 169
Option Editing 6
Options - Programming Sockets - Gang 17
Options - Select MCU Type 15
OR 176
OR fr into W 176
OR fr2 into fr1 176
OR literal into fr 176
OR literal into W 176
OR W into fr 176
org 194
oscillator frequency 59
OTP 15
Override Power Off Requirement 12

- P -

Parent Node 25
password 7
Password to Enable Option Editing 7
Paste 85
PC.DEF 130
PCB 130
PCM 130
PIC16C54 187
PIC16C74 194
Place at Address 10
Place in Code Memory 10

Place in EEPROM Memory 10
Place markers 91
popped 57
POPUP Menu 105
pop-up menu 80, 115, 116
Power First 15
power-on state 76
Print 67
Printing 67
Production Control 7
Program Counter 76, 77
Project 50, 95
Project and File Operations 44
Project Info 31
Project menu 28, 32, 46, 50, 68
project name 45
PROJECT NODE 48, 51, 127, 128, 129, 130, 131
Project Node Tool 48, 51
Project Tool 29
protecting your settings from accidental changes 14
pushed 57

- Q -

QuickWriter 4
QuickWriter Control 5

- R -

radix 115
Randomize File Registers 62
RC oscillator 187, 190
RC_OSC 190
Read 120
Read or Write 120
Reading from a lookup table 190
Reading the MCU Contents 19
redo 88
register modification window 97
Replace 89
Replace All 89
Replace with 89
RESET 76, 187, 190
Reset Mathias 66
Reset on Timeout 57
Resetting the Emulated PIC 76
Resetting the Emulator 41, 66
RET 178
Retain Z Order 62
RETFIE 145, 178
RETI 178
RETLW 17, 145, 178
RETLW OPCODE 10
RETURN 145
RETURN ADDRESSES 57
Return from Interrupt 178
Return from subroutine 178
retw 178, 190
rl 179, 194
RLF 145, 172, 179
Rotate fr left 179
Rotate fr right 179
rr 179, 194
RRF 145, 172, 179
Run menu 34, 35, 36, 38, 41, 43, 70, 71, 72, 73, 74, 75, 76, 77, 78
Running TDE 44
Running the Code 34

- S -

Sample Project 27
Sample Projects 186
Save HEX File - Include Record Options 9
SB 179
SC 179
Search 89
Search Again 89
Search and replace 89
Select and Add 109
Select and Drag & Drop 109
Select MCU Type 15
Selected line 95
Selecting the MCU 15
Self Modifying Code 118
Serial Number Options 10
Set and clear breakpoints 85
Set bit 180
Set carry 180
Set PC 77
Set zero flag 180
setb 180, 190, 194
SETC 145
Setting Breakpoints 41

Setting device options 187, 190
Setting General Options 7
Setting HEX Options 9
Setting I.D. Locations 13
Setting I/O pins as inputs or outputs 187, 190
Setting Options for Production Personnel 14
Setting Serial Number Options 10
Setting the Configuration Word (Fuses) 15
Setting the Program Counter 77
Setting the reset vector 187, 190
Setup menu 31, 55, 62, 95, 118, 120, 186
SETZ 145
Shift + F9 74
Shortcuts 14
Show Execution Point 75
Show Hints 62
Showing the Execution Point 75
SIMPLE 187
SINGLE FILE MODE 127, 128
SINGLE SOURCE FILE 127, 128, 129, 130
Single Step 71
Single-Step Execution 71
Single-Stepping 35
SKIP 181
Skip Blank Locations 7
skip if above 157
skip if above or equal 157
skip if below 159
skip if below or equal 159
Skip if bit 179
Skip if carry 179
skip if equal 160
Skip if not bit 182
Skip if not carry 182
skip if not equal 161
Skip if not zero 182
Skip if zero 179
Skip Power Cycles 12
Skip the following instruction word 181
Skipped 17
SKPC 145
SKPDC 145
SKPNC 145
SKPNDC 145
SKPNZ 145
SKPZ 145
SLEEP 145, 181
sleep mode 181
SNB 182
SNC 182
SNZ 182
sockets 17
SPASM 127
special instructions 145
Special Register Viewing Options 97
Special Registers Window 97
Specific Data 121
Stack OverFlow 57
Stack UnderFlow 57
Starting TDE 28
STC 180
Step OUT Execution 72
Step Over Execution 73
Stop button 34, 36, 40
stop execution 36
Stopping Execution 34
STZ 180
SUB 183
SUBB 183
SUBCF 145
SUBDCF 145
SUBLW 145
Subtract bit from fr 183
Subtract fr2 from fr 183
Subtract literal from fr 183
Subtract W from fr 183
SUBWF 145, 172, 183
SWAP 184
Swap nibbles in fr 184
SWAPF 145, 172, 184
SZ 179

- T -

Tab Stops 62
Target Design Recommendations 20
Target Voltage Option 12
TDE Details 44
TechTools Equivalent 145
TEST 184
Test fr for zero 184
Test W for zero 184
third-party compiler or assembler 126
Tmin 12
Toggle Marker 91
tool 51

Trace Buffer 60
Trace Buffer Module 57
Trace Buffer Window 116
Trace Size 60
TRIS 145
TRIS registers 171
TSTF 145
TVDDEN 12, 23

- U -

Undo 88
Undo and redo 88
Use High Speed Transfers 7
Use Microchip Checksums 7
Using Breakpoints 78
Using Gang Adapters 17
Using global labels 187
Using labels 190
Using local labels 187
Using Serial Numbers 17
Using TDE's Windows 44
Using the Various TDE Windows 80

- V -

VDD 21
VDD First 15
View menu 38, 82, 93, 95, 97, 105, 108
Viewing Special Registers 38
Virtual Mathias 55

- W -

WARN if Code Protection Enabled 15
Warn if CODE/Data Protection is Enabled 7
Watch 105
Watch button 36
Watch Dog Timer 57
watch list 112
Watch List Options 108
Watch List Window 108
Watching the Program Run 40
Watching Variable Values 36
Watching Variables 38
WDT 57
WDT_OFF 194
Window Basics 80
Write 120

- X -

xor 185, 187, 194
XOR fr into W 185
XOR fr2 into fr1 185
XOR literal into fr 185
XOR literal into W 185
XOR W into fr 185
XORLW 145, 176, 185
XORWF 145, 185
XT_OSC 194

TechTools

(972) 272-9392, www.tech-tools.com