

Electronics Workbench™

MultiMCU Microcontroller Co-simulation

User Guide

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada 800 433 3488, China 86 21 6555 7838, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00,
Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, India 91 80 41190000,
Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400,
Lebanon 961 0 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 0 348 433 466,
New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210,
Russia 7 095 783 68 51, Singapore 1800 226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197,
Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 02 2377 2222,
Thailand 662 278 6777, United Kingdom 44 0 1635 523545

For further support information, refer to the [Technical Support Resources and Professional Services](#) page. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code `feedback`.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THEREFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

The ASM51 cross assembler bundled with MultMCU is a copyrighted product of MetaLink Corp. (www.metaice.com).

Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on ni.com/legal for more information about National Instruments trademarks.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

MPASM™ macro assembler and related documentation and literature is reproduced and distributed by Electronics Workbench under license from Microchip Technology Inc. All rights reserved by Microchip Technology Inc. MICROCHIP SOFTWARE OR FIRMWARE AND LITERATURE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY ARISING OUT OF OR IN CONNECTION WITH THE SOFTWARE OR FIRMWARE OR THE USE OF OTHER DEALINGS IN THE SOFTWARE OR FIRMWARE.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or ni.com/patents.

Some portions of this product are protected under United States Patent No. 6,560,572.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

- (1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.
- (2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS.

BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Table of Contents

1. Introduction

1.1	Overview	1-1
1.2	MultiMCU Basics	1-2
1.3	Debugging Tools	1-2
1.3.1	MCU Assembly Source View	1-2
1.3.2	MCU Memory View	1-5
1.3.3	Advanced Features	1-6
1.4	Peripheral Devices	1-6
1.5	Installing MultiMCU	1-6
1.5.1	Entering the Release Code	1-7

2. MultiMCU Sample Walkthroughs

2.1	Tutorial 1 - MCU Driven Blinking Lights	2-2
2.1.1	Overview	2-2
2.1.2	About the Tutorial	2-2
2.1.3	Advanced Features - Blinking Lights Example	2-6
2.1.3.1	Adding a Breakpoint	2-6
2.1.3.2	Break and Continue	2-6
2.1.3.3	Break and Step	2-7
2.2	Tutorial 2 - MCU Controlled Holding Tank	2-8
2.2.1	Overview	2-8
2.2.2	About the Tutorial	2-8
2.2.3	Advanced Features - Holding Tank Example	2-12
2.2.3.1	Adding a Breakpoint	2-13
2.2.3.2	Break and Continue	2-13
2.2.3.3	Break and Step	2-13
2.3	Tutorial 3 - MCU Based Calculator	2-14
2.3.1	Overview	2-14
2.3.2	About the Tutorial	2-14
2.3.3	Advanced Features - Calculator Example	2-21
2.3.3.1	Adding a Breakpoint	2-21
2.3.3.2	Break and Continue	2-22
2.3.3.3	Break and Step	2-22

2.4	Tutorial 4 - MCU Serial Terminal	.2-23
2.4.1	Overview	.2-23
2.4.2	About the Tutorial	.2-24
2.4.3	Using the MCU Interface	.2-26

Appendix A - MultiMCU Parts

A.1	8051/8052 Microcontroller Units	A-1
A.2	PIC16F84/16F84A Microcontroller Units	A-4
A.3	RAM	A-6
A.4	ROM	A-7

Chapter 1

Introduction

The following are found in this chapter.

Subject	Page No.
Overview	1-1
MultiMCU Basics	1-2
Debugging Tools	1-2
Peripheral Devices	1-6
Installing MultiMCU	1-6

1.1 Overview

Microcontroller (MCU) components are useful for many circuit designs. A modern microcontroller typically combines a CPU, data memory, program memory, and peripheral devices on a single physical chip. The integration of these essential elements of a computer into a single chip reduces component counts and board size resulting in higher reliability with more capabilities. The MCU Co-simulation system provides software development features for writing and debugging code for embedded devices.

Embedded software development can be a challenging process for even the best programmers. MultiMCU helps you produce high quality code more quickly and easily. The MCU development interfaces allow you to pause a simulation, inspect the internal memory and registers of the MCU, set code breakpoints and single step through your code.

1.2 MultiMCU Basics

This and subsequent sections give a brief overview of MultiMCU's components. For detailed examples of circuits containing MCUs, refer to

- “2.1 Tutorial 1 - MCU Driven Blinking Lights” on page 2-2
- “2.2 Tutorial 2 - MCU Controlled Holding Tank” on page 2-8
- “2.3 Tutorial 3 - MCU Based Calculator” on page 2-14
- “2.4 Tutorial 4 - MCU Serial Terminal” on page 2-23.

➤ To place an MCU:



1. Select **Place/Component** to display the **Select a Component** dialog box.
2. Navigate to the MultiMCU **Group** and select the **Family** containing the desired MCU (e.g., 805x, PIC).
3. Select the desired MCU, click **OK** and click again to place the component on the workspace.

In addition to the MCU, the **MCU Assembly Source View** and the **MCU Memory View** also appear. To show/hide these views, or to change the MCU's values, see “A.1 8051/8052 Microcontroller Units” on page A-1 or “A.2 PIC16F84/16F84A Microcontroller Units” on page A-4.

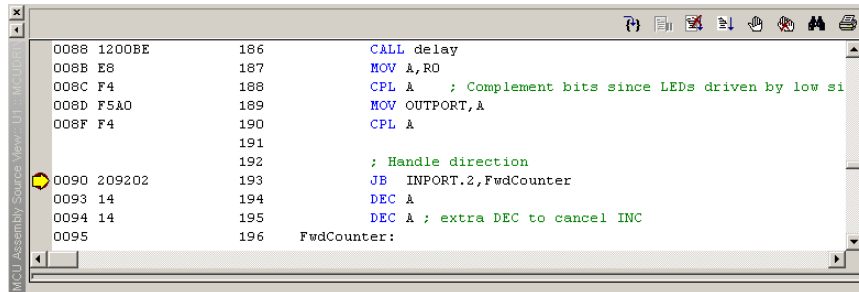
Note See also, “1.3.1 MCU Assembly Source View” on page 1-2 and “1.3.2 MCU Memory View” on page 1-5.

1.3 Debugging Tools

The MCU debugging tools provide the user with the ability to control execution at the instruction level (breakpoints and single-stepping) while also providing views of the memory and registers within the MCU.

1.3.1 MCU Assembly Source View

The **MCU Assembly Source View** shows the assembly source code for the MCU program. This is also where you can set breakpoints to have the simulation pause at a particular location in the code. The dialog will automatically scroll to the place in the assembly code where the simulation has paused and indicate the current instruction with an arrow.



```

0088 1200BE      186      CALL delay
008E E8         187      MOV A,RO
008C F4         188      CPL A ; Complement bits since LEDs driven by low si
008D F5A0       189      MOV OUTPORT,A
008F F4         190      CPL A
191
192      ; Handle direction
0090 2092D2     193      JB INPORT.2,FwdCounter
0093 14         194      DEC A
0094 14         195      DEC A ; extra DEC to cancel INC
0095           196      FwdCounter:





```












The numbers on the far left are the program memory addresses and the hexadecimal codes to their immediate right are the assembled codes for each mnemonic assembly instruction. The column of numbers in the middle shows the line number in the original assembly source. The remainder of the line to the right shows the assembly source and comments. The red dot indicates a breakpoint. The yellow arrow indicates the instruction that the program has paused at. On the top right on the **MCU Assembly Source View** are buttons to control execution. These buttons change depending on whether or not simulation is running. For details, see “MCU Assembly Source View Buttons” on page 1-3.

MCU Assembly Source View Buttons

The following buttons appear in the **MCU Assembly Source View** during edit mode (i.e., simulation is not running).

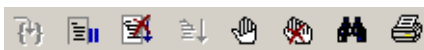










Button	Description
	New button. Clears the contents of the MCU Assembly Source View .
	Import button. Imports an assembly file (.asm) that you created in a text editor into the MCU Assembly Source View .
	Export button. Exports the contents of the MCU Assembly Source View into an assembly text file (.asm).
	Cut button. Removes the selection from the MCU Assembly Source View and places it on the clipboard.

Button	Description
	Copy button. Copies the selection onto the clipboard.
	Paste button. Pastes the contents of the clipboard into the insertion point in the MCU Assembly Source View .
	Undo button. Undoes the previous action.
	Redo button. Redoes the most recently performed undo action.
	Build button. Assembles the code in the MCU Assembly Source View .
	Go button. Starts the debugging process in the MCU Assembly Source View .
	Insert/Remove Breakpoint button. Inserts or removes a breakpoint from the code in the MCU Assembly Source View .
	Remove All Breakpoints button. Removes all breakpoints from the code in the MCU Assembly Source View .
	Show Line Numbers button. Displays the line numbers in the MCU Assembly Source View .
	Find button. Displays the Find dialog box, where you can search for text strings in the MCU Assembly Source View .
	Print button. Prints the contents of the MCU Assembly Source View .

Note When you click the **Build** button in the **MCU Assembly Source View**, a line appears in the lower pane stating the total number of errors, warnings and messages. You can view these by scrolling up. For information on errors and warnings, refer to the appropriate assembler manual, found on your Documentation CD.

The following buttons appear in the **MCU Assembly Source View** during simulation/debugging (annotated source code only appears, as editing is not allowed during simulation).



Button	Description
	Step button. Executes the next logical instruction.
	Break Execution button. Pauses the simulation at the current instruction.
	Stop Debugging button. Stops the debugging process.
	Go button. Starts the debugging process in the MCU Assembly Source View .
	Insert/Remove Breakpoint button. Inserts or removes a breakpoint from the code in the MCU Assembly Source View .
	Remove All Breakpoints button. Removes all breakpoints from the code in the MCU Assembly Source View .
	Find button. Displays the Find dialog box, where you can search for text strings in the MCU Assembly Source View .
	Print button. Prints the contents of the MCU Assembly Source View .

1.3.2 MCU Memory View

The contents of the **MCU Memory View** change depending on the type of MCU. It may, for example, contain internal memory information, register views and configuration information.

For details on specific MCUs, see “A.1 8051/8052 Microcontroller Units” on page A-1 and “A.2 PIC16F84/16F84A Microcontroller Units” on page A-4.

1.3.3 Advanced Features

MultiMCU provides advanced debugging tools to make it easy to pause your circuit and explore the internal data and state of the MCU controlling your circuit. MultiMCU lets you set breakpoints and single step through assembly code while validating that the register contents are changing as expected. You can also manually edit most memory view while debugging.

Examples of MultiMCU's advanced debugging features can be found in the sample circuit descriptions found in Chapter 2, "MultiMCU Sample Walkthroughs".

1.4 Peripheral Devices

Along with its selection of MCUs, MultiMCU contains a number of peripheral devices.

The MultiMCU **Group** contains RAM and ROM devices that are designed to function specifically with the MCUs. For details on these components, see "A.3 RAM" on page A-6 and "A.4 ROM" on page A-7.

The Advanced Peripherals **Group** contains a selection of Keypads, LCDs, Terminals and Miscellaneous Peripherals like the Liquid Holding Tank.

For details, refer to the *Multisim 9 Component Reference Guide*, or the Component Helpfile.

1.5 Installing MultiMCU

You must install Multisim before installing MultiMCU. If you attempt to install MultiMCU before installing Multisim, MultiMCU will not install.

The MultiMCU 9 CD you received will autostart when inserted in the CD-ROM drive. Follow the instructions below and on the screen during the installation process.

➤ To install MultiMCU 9:

1. Copy down the serial number you have received with your MultiMCU 9 package.
2. Exit all Windows applications prior to continuing with the installation.
3. Insert the MultiMCU 9 CD into your CD-ROM drive. When the splash screen appears, click on MultiMCU 9 to begin the installation.
4. Follow the on-screen prompts to complete the installation.

1.5.1 Entering the Release Code

MultiMCU 9 requires you to enter a Release Code within five days of the date of installation. After the five day grace period has expired, MultiMCU 9 will not run until a Release Code is entered.

To obtain your Release Code, you must provide us with your MultiMCU Serial Number and Signature number, as displayed on the splash screen. Contact Electronics Workbench via our website (preferred method) at www.electronicsworkbench.com and select the Product Registration link, or call Customer Service at 1.800.263.5552. Customers outside North America should contact their local distributor.

Electronics Workbench recommends that you obtain your Release Code as soon as possible after you have installed MultiMCU 9.

Note The Release Code that you will be provided with is composed of 60-alphanumeric characters. Electronics Workbench recommends that you use one of the methods below to enter the Release Code.

➤ To enter the Release Code:

1. Launch Multisim. The MultiMCU 9 release code splash screen displays.

Note If you launch Multisim without a MultiMCU 9 release code, you may press **Cancel** to continue. Remember that after five days, MultiMCU 9 will not run until a valid release code is entered.

2. If you received your Release Code via email there are a few ways to easily enter it without the need to type each number or character one at a time. Select one of the following methods:

- Highlight the Release Code. Drag and drop it on one of the text boxes.
- Highlight the Release Code, right-click on it and select Copy. Click on the **Paste Release Code** button.
- Highlight the Release Code, right-click on it and select Copy. Right-click on one of the text boxes and click on Paste from the pop-up menu.

3. If you have received your Release Code over the phone, you must type it in the Release Code fields 5 characters at a time.

4. Click **OK** to continue.

Chapter 2

MultiMCU Sample Walkthroughs

This chapter details several tutorials that use MultiMCU's co-simulation functionality. The circuits for the tutorials are found in the folder where you installed Multisim 9, at `...\samples\MCU Sample Circuits`.

The following are described in this chapter.

Subject	Page No.
Tutorial 1 - MCU Driven Blinking Lights	2-2
Tutorial 2 - MCU Controlled Holding Tank	2-8
Tutorial 3 - MCU Based Calculator	2-14
Tutorial 4 - MCU Serial Terminal	2-23

2.1 Tutorial 1 - MCU Driven Blinking Lights

This tutorial leads you through the simulation of the Blinking Lights sample circuit.

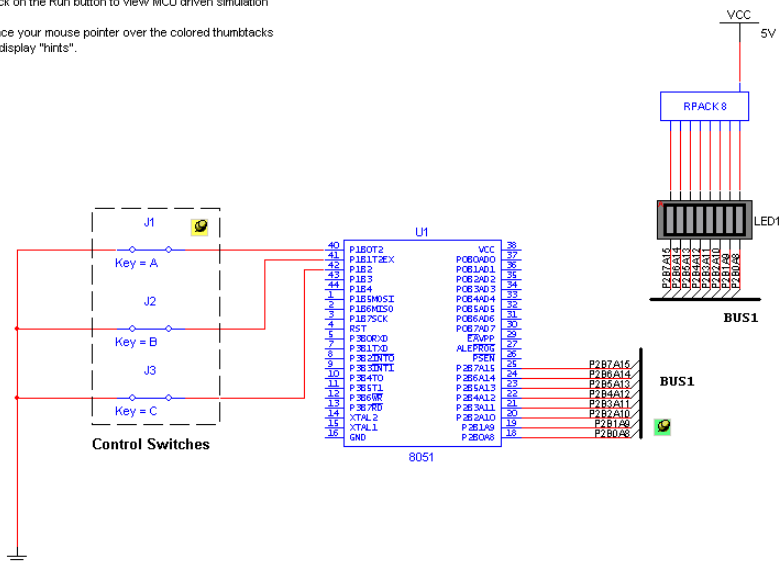
2.1.1 Overview

The Blinking Lights circuit example shows the use of a microcontroller to control a set of LEDs as might be found in a novelty toy. There are four operating modes controlled by the different combinations of switches A and B. The third switch, C, controls the direction for two of the modes.

MCU Driven Blinking Lights

Click on the Run button to view MCU driven simulation

Place your mouse pointer over the colored thumbtacks to display "hints".



2.1.2 About the Tutorial

The Blinking Lights circuit is composed of an 8051 MCU component (U1) connected to three switches (J1, J2, J3) on Port 1 of the 8051 and Bar LED (LED1) connected to Port 2 of the 8051 for display output. The resistor pack attached to the other side of the Bar LED completes the circuit. This circuit is a simple demonstration of using inputs to the MCU to control outputs.

The switches translate into the mode value as a simple mapping.

Mode	Switch J1 (A-key)	Switch J2 (B-key)
0 (Sweeping Eye)	Closed	Closed
1 (Meter)	Open	Closed
2 (Counter)	Closed	Open
3 (Marquis)	Open	Open

Note Port 1 (as well as Port 2 and Port 3) on the 8051 has internal pull-ups so an open switch will read as a High value. The closed switches will connect the pins to Ground.

The sample uses the Switch J1 (A-key) and Switch J2 (B-key) inputs to decide which algorithm to use.

Dispatch routine:

```
Read switches J1 and J2 (Converted to Modes 0 to 3)
Jump to the code for Mode m
```

The assembly code for this is:

```
Dispatch:
    ; The dispatch section reads switches A and B and
    ; runs the corresponding display pattern.
    MOV DPL,#LOW(DispatchJumpTable) ; set start of jump table
    MOV DPH,#HIGH(DispatchJumpTable)
    MOV A,INPORT ; Read input port
    ANL A,#003H ; Confine to 4 choices
    MOV R7,A ; Make copy in R7 for comparisons
    RL A ; multiply by two since each AJMP is two bytes
    JMP @A+DPTR
```

```
DispatchJumpTable:
    AJMP SweepingEyeBegin
    AJMP MeterBegin
    AJMP CounterBegin
    AJMP MarquisBegin
```

Each mode algorithm checks for changes in the states of Switches A and B. If the mode changes, they will abort and jump back to the dispatch routine.

The four modes display different patterns on the Bar LED. These are:

a) Sweeping Eye Pattern (Mode 0)

A four-wide group of lights are moved back and forth across the Bar LED.

b) Meter Pattern (Mode 1)

The light pattern grows and shrinks from the right like a level meter.

c) Counter (Mode 2)

The Bar LED shows an 8-bit counter value. Switch J3 (C-key) controls whether the value increases or decreases.

d) Marquis (Mode 3)

The marquis mode moves a pattern from left to right or right to left with the pattern wrapping to the other side as it shifts off the Bar LED. The direction of movement is controlled by Switch J3 (C-key). Left to right is chosen by Switch J3 being Closed. Right to left is chosen by Switch J3 being Open.

➤ To run this circuit:

1. Select **Simulate/Run** to begin simulation. The MCU will immediately begin flashing the lights in the pattern appropriate for the switch settings.
2. Change the mode switches (J1 and J2) using the A key and the B key on your keyboard and see the corresponding change in pattern of the LEDs.

The following excerpt shows the code for the Counter pattern.

```
CounterBegin:
    MOV R0,#000H
CounterLoop:
    CALL delay
    MOV A,R0
    CPL A    ; Complement bits since LEDs driven by low signals.
    MOV OUTPORT,A
    CPL A
```

```
        ; Handle direction
        JB  INPORT.2,FwdCounter
        DEC A
        DEC A ; extra DEC to cancel INC
FwdCounter:
        INC A
        MOV R0,A

        MOV A,INPORT    ; branch to beginning if config inputs change
        ANL A,#003H
        XRL A,R7
        JNZ CounterEnd

        JMP CounterLoop
CounterEnd:
        JMP Begin
```

Some areas of interest to note in this code fragment are:

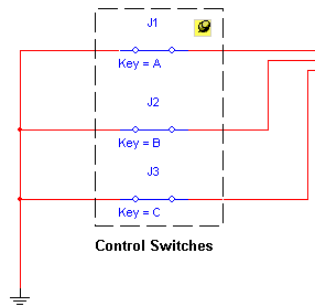
- A delay subroutine is called to slow down the pattern. The sample MCU is using an internal 12 MHz oscillator. Without the delay the lights would be changing too quickly.
- A typical MCU component can sink more current than it can source so the bar LED was arranged to match. However, people usually equate a lighted LED with a Logic One or High value. The bits written to the output port are complemented to produce LED results that match expectations. This is yet another convenience of using microcontrollers. It is often easier to transform inputs or outputs of an MCU in code than it would be to add circuit elements to perform the same effect.
- A bit-test branch is used directly on the port bit attached to Switch J3 (C-key). The bit-is-set branch (JB opcode) goes directly to *FwdCounter*. The 'else' clause to decrement the counter uses the fact that one can run an extra DEC instruction to offset the INC instruction at *FwdCounter* where execution will flow through.

2.1.3 Advanced Features - Blinking Lights Example

This section provides a step by step walkthrough of the MultiMCU debugging features. It is important to follow the steps exactly as scripted otherwise the descriptions will no longer apply. Once you understand how the breakpoint and single stepping features you can explore the possibilities of advanced MCU debugging.


2.1.3.1 Adding a Breakpoint


1. Load the MCU Driven Blinking Lights Example. The switches should all be closed.




2. Scroll the **MCU Assembly Source View** to the Counter Loop and move the cursor (by cursor keys or mouse click) to line 192. It shows:

```
JB INPORT.2, FwdCounter
```

Tip: If the line numbers are not showing click the **Show Line Numbers** button  .

3. Click the **Insert/Remove Breakpoint** button  . You should see a red dot appear in the left margin.

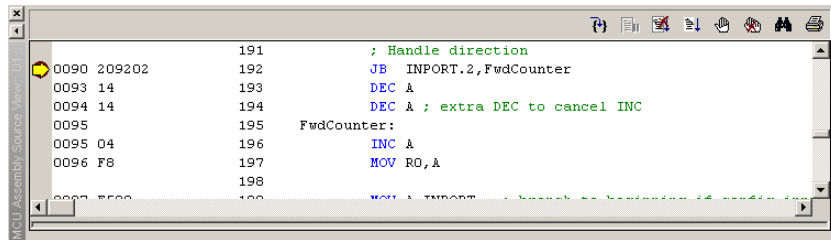
You have now set a breakpoint at the branch instruction in the Counter mode loop that decides whether to increment or decrement the counter.

4. You can remove this breakpoint by clicking on the same **Insert/Remove Breakpoint** button again or you can remove all of the break points in one step by clicking on the **Remove All Breakpoints** button  .

2.1.3.2 Break and Continue

1. Select **Simulate/Run** to begin simulation.
2. You should see the Sweeping Eye pattern on the Bar LED. Our simulation breakpoint does not get triggered because we are in Sweeping Eye mode instead of Counter mode.
3. Move into Counter mode by hitting the B key on the keyboard. The switch should change to the Open state.


- The simulator should be paused now and the Assembly Source Window will show the yellow arrow over our breakpoint.




```

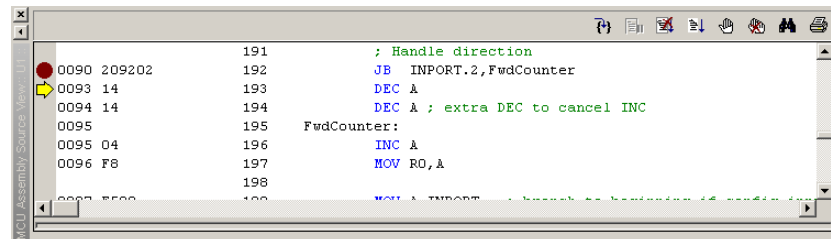
0090 209202    191      ; Handle direction
0093 14        192      JB  INPORT.2,FwdCounter
0094 14        193      DEC  A
0095         194      DEC  A ; extra DEC to cancel INC
0095 04        195      FwdCounter:
0096 F8        196      INC  A
0097 F8        197      MOV  R0,A
0098 F8        198      MOV  A,INPORT ; branch on hardware if switch is
0099 F8        199      MOV  A,INPORT ; branch on hardware if switch is

```

- Look at the Accumulator (ACC) in the SFR (Special Function Register). Its value is 00H.
- Click on the **Go** button  in the Assembly Source window. The Counter Loop will run for one iteration and stop again at our breakpoint. The new accumulator value is 0FFH.

2.1.3.3 Break and Step

- Click on the **Step** button . The simulator will run briefly and then return to the pause state. The yellow arrow has moved to the instruction shown below.





```

0090 209202    191      ; Handle direction
0093 14        192      JB  INPORT.2,FwdCounter
0094 14        193      DEC  A
0095         194      DEC  A ; extra DEC to cancel INC
0095 04        195      FwdCounter:
0096 F8        196      INC  A
0097 F8        197      MOV  R0,A
0098 F8        198      MOV  A,INPORT ; branch on hardware if switch is
0099 F8        199      MOV  A,INPORT ; branch on hardware if switch is


```

- Click on the **Step** button a few more times and watch the Accumulator value change as the DEC and INC instructions are executed.

Setting Breakpoints During Simulation

- Stop the Simulation by clicking on the **Stop Debugging** button .
- Click on the **Remove All Breakpoints** button . This will clear all user breakpoints. All of the red breakpoint dots should disappear.
- Select **Simulate/Run** to begin simulation.
- Hit the A and B keys on the keyboard until both switches are open.
- You should see the Marquis pattern scrolling across the Bar LED.
- Breakpoints can be set and cleared during a simulation. Scroll to the line in the Assembly Source View dialog corresponding to address 00AC in the left-most column. The

assembly instruction for that line decides between left or right scrolling of the marquis pattern.

7. Make sure your cursor is positioned on that line and click on the **Insert/Remove Breakpoint** button . The simulation will pause almost immediately at your new breakpoint since the MCU was looping through that code to display the marquis.

2.2 Tutorial 2 - MCU Controlled Holding Tank

This section details an 8051 MCU that controls a circuit example that fills and then empties a fluid holding tank.

2.2.1 Overview

The 8051 MCU emulates the behavior of the ladder logic diagram example to control the filling and emptying of the holding tank. The logic behind the MCU is contained inside an assembly program that is loaded when the circuit starts running. The circuit can be run using the same series of operations as the ladder logic circuit. In addition to the schematic capture interface, there is an MCU interface that allows you to view the instructions in the assembly code that are being executed by the MCU at the same time that you are running the simulation.

You can pause the simulation at any time and see the exact corresponding assembly instruction that the MCU is about to execute. You can also go in the opposite direction and set breakpoints inside the code to pause the simulation automatically when it reaches the desired point in the program of the MCU and see what is happening in your simulation. To understand the assembly code in even more detail, you can also step through the assembly instructions one by one to view the flow of control.

2.2.2 About the Tutorial

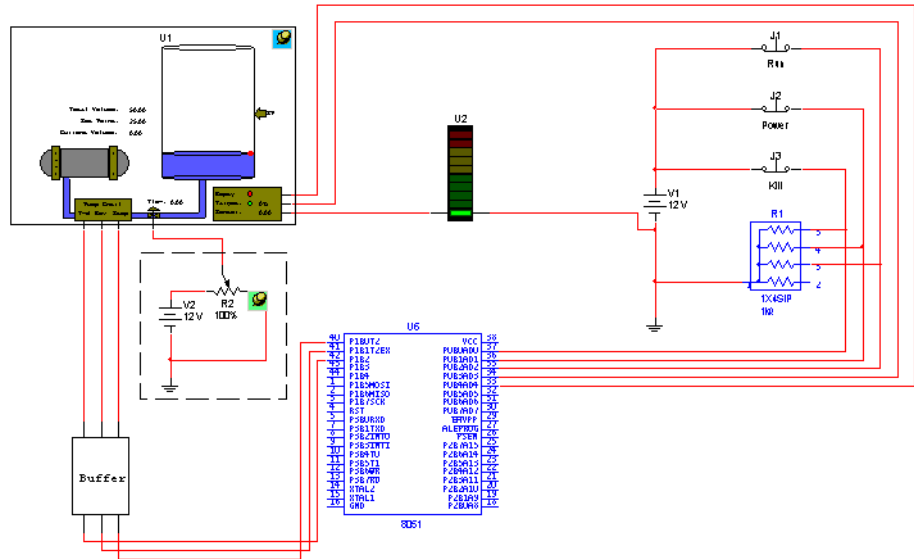
The input signals to the 8051 MCU are the push buttons and the empty and set point pins of the Holding Tank part. These input signals are connected to port P0 (pins P0B0AD0 to P0B4AD4). The MCU generates output signals on port P1 (pins P1B0T2 to P1B2). The output signals are connected to the Fwd, Rev and Stop pins of the Holding Tank. The

changing of the input signals will cause the MCU to generate output signals that emulate the behavior of the analog circuit in the ladder logic example.

MCU Controlled Holding Tank

Click on the Run button to view MCU driver simulation

Place your mouse pointer over the colored thumbnails to display "hints"



➤ To activate this circuit:

1. Select **Simulate/Run** to begin simulation. The program memory code is loaded at this point and the MCU is waiting for the power button to be pressed. The corresponding assembly code is as follows:


```

; Wait for power button to be pressed
startloop:
MOV P1,#000H
JB P0.1,ready; power button was pressed
JMP startloop
            
```
2. Press the 'P' key on the keyboard to activate the Power switch. This sends 5V to pin P0B1AD1 of U6 (MCU) which puts the MCU into the ready state to accept other input signals to start running the circuit.

ready:

```
MOV P1,#001H
```

; Wait for run button to be pressed to start filling tank

```
readyloop:
    JB P0.0,start ; kill button pressed
    JB P0.2,run   ; run button pressed
    JMP readyloop
```

➤ To run the holding tank circuit:

1. Press the 'R' key on the keyboard to activate the Run switch. The MCU will receive a high signal on port P0 bit 2 and set the output pin of P1 bit 2 to high for a moment to start filling the tank in the forward direction.

```
; Fill in forward direction
```

```
fillfwd:
```

```
    MOV P1,#004H      ; set fwd signal to high
    CALL outputdelay  ; hold fwd signal high
    CALL outputdelay
    MOV P1,#000H      ; set fwd signal back to low
```

2. When the tank reaches the set point, the MCU is notified by the high signal that it receives on port P0 bit 3.

```
; Wait for set point to be reached
```

```
fillfwdloop:
```

```
    JB P0.0,fillfwdkill ; kill button pressed
    JB P0.3,fillfwdend  ; set point reached
    JMP fillfwdloop
```

3. The MCU sends a high output signal to the stop pin of the pump control and the fluid stops being pumped.

```
; Stop filling in fwd direction and start timer for 5 seconds
```

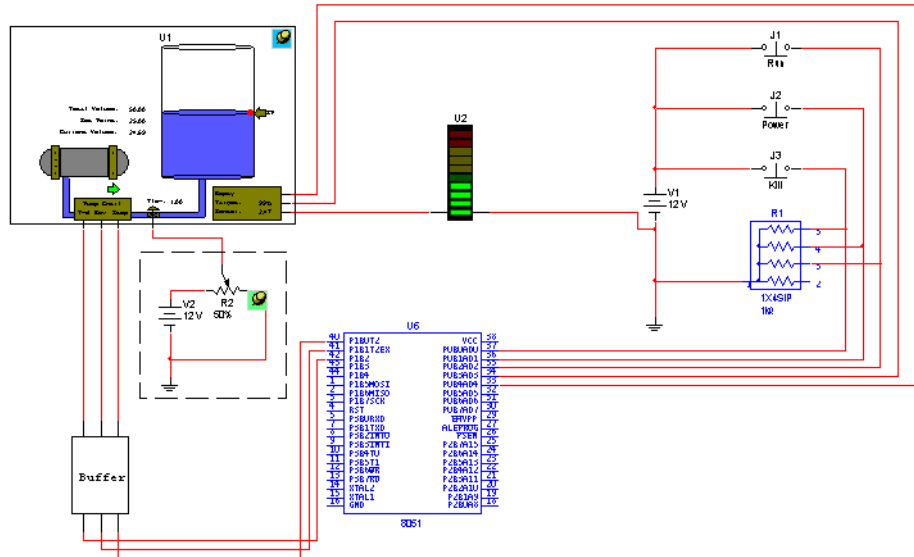
```
fillfwdend:
```

```
    MOV P1,#001H      ; set stop signal to high
```

MCU Controlled Holding Tank

Click on the R1s button to allow MCU drive a pin to box

Place your mouse pointer over the colored embeddings to display "list"



```
        JMP fillrevloop
; Finished draining, go back to ready state
fillrevend:
        MOV P1,#001H          ; set stop signal to high
        JMP ready
```

➤ To turn off the power at any point in the simulation:

1. Press the 'K' key on your keyboard to activate the Kill switch. This sends 5V to pin POB0AD0. There is code in each of the various states of the circuit to stop the filling, emptying of the tank or timer function depending on which is currently occurring.

```
; Kill button was pressed during filling in fwd direction
fillfwdkill:
        MOV P1,#001H          ; set stop signal to high
        CALL outputdelay      ; hold top signal high
        CALL outputdelay
        JMP start              ; go back to beginning of program




; Kill button was pressed during filling in reverse direction
fillrevkill:
        MOV P1,#001H          ; send stop signal
        CALL outputdelay
        CALL outputdelay
        JMP start

; Kill button was pressed during timer routine, wait for power button
timerdelaykill:
        JB P0.1,timerdelayready ; power button pressed
        JMP timerdelaykill
```


2.2.3 Advanced Features - Holding Tank Example

This section provides a step by step walkthrough of the MultiMCU debugging features. It is important to follow the steps exactly as scripted otherwise the descriptions will no longer apply. Once you understand how the breakpoint and single stepping features you can explore the possibilities of advanced MCU debugging.

2.2.3.1 Adding a Breakpoint


1. Load the MCU Controlled Holding Tank Example.
2. Scroll the **MCU Assembly Source View** and move the cursor (by cursor keys or mouse click) to line 45 (MOV P1,#001H).
Tip: If the line numbers are not showing click the **Show Line Numbers** button  .
3. Click on the **Insert/Remove Breakpoint** button  . You should see a red dot on the left margin.
 You have now set a breakpoint at the branch instruction.
4. Place a second break point on line 49 (MOV P1,#004H ; set fwd signal to high) .
5. You can remove this breakpoint by clicking on the **Insert/Remove Breakpoint** button again or you can remove all of the break points in one step by clicking on the **Remove All Breakpoints** button  .

2.2.3.2 Break and Continue

1. Select **Simulate/Run** to start the simulation.
2. Press ‘P’ on your keyboard to activate the circuit.
3. Press ‘R’ on your keyboard to start filling the tank.
4. The simulation will pause at the first breakpoint.
5. Look at the MCU Register View and scroll until you can see P1 and notice that bit 2 is 0.
6. Click on the **Go** button  in the **MCU Assembly Source View**. The program will execute until it reaches the next break point.
7. Notice that the MCU Register View has been updated with the current values and will display a value of 1 inside P1 bit 2 after executing the instruction “MOV P1, #004H”.
8. Click on the **Go** button again and the simulation will continue filling the tank.

2.2.3.3 Break and Step

1. Stop the simulation and clear any break points that you inserted earlier.
2. Place a break point on line 62 (CALL timerdelay). Line 62 is executed to start the 5 second timer when the tank is filled to the set point.
3. Select **Simulate/Run** to restart the simulation.
4. Press ‘P’ on your keyboard to activate your circuit.
5. Press ‘R’ on your keyboard to start filling.
6. When level of the fluid in the tank reaches the set point SP, the simulation will pause and the debugger will show the paused program execution at line 62 .

7. Click on the **Step** button  to enter the subroutine 'timerdelay'.
8. The yellow arrow shows the next instruction that will be executed at line 110 inside the "timerdelay" (JMP timerstart).
9. If you click on the **Step** button again, JMP timerstart will execute and jump to line 96 where the "timerstart" code begins (MOV P1,#001H).
10. Click on the **Step** button one more time. See that the value in P1 bit 0 was set to 1 to stop the filling of the tank.
11. You can step through more instructions to see how the rest of the routine functions.

2.3 Tutorial 3 - MCU Based Calculator

This section contains an example of a calculator application created using an 8051 MCU. All the logic for the arithmetic, input and output operations of the calculator, are handled by the MCU.

2.3.1 Overview

This circuit behaves like a normal calculator that performs operations of the form **operand1 operator operand2 = result**, where:

- Operand1 and operand2 are positive integers between 0 and 9999, and
- The operator can be +, -, * or /.

For example, a typical operation would be $3 * 4 = 12$.

Numbers and operators can be entered via the keypad and displayed on the HEX Displays attached to the MCU. As the equation is entered into the calculator, the results are calculated as soon as enough information is entered to perform a calculation. The result is displayed on the HEX Displays and will be used as the first operand in the next calculation. All of these operations are performed by the 8051 MCU. The logic for the MCU is programmed in assembly and loaded at the start of simulation.

2.3.2 About the Tutorial

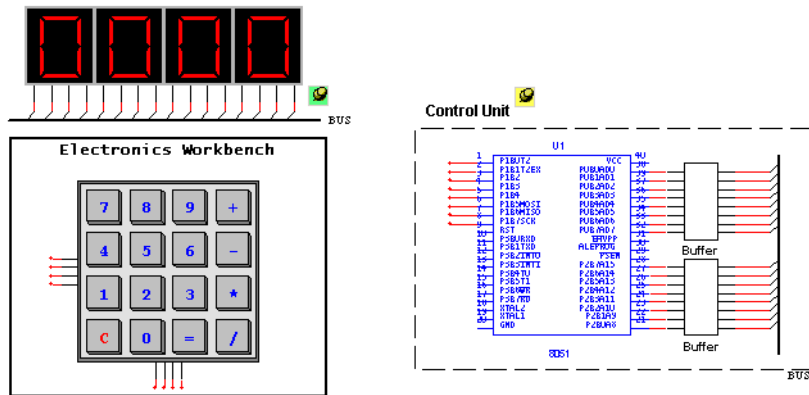
The calculator circuit consists of an 8051 MCU that is hooked up to a keypad via port P1 and 4 LEDs via ports P0 and P2. The keypad is an interactive part used for entering input values into the calculator. The keypad can be used by pressing keys on the keyboard that correspond to the characters on the keypad. These characters are fed into the MCU, manipulated and the resulting values are displayed on the HEX Displays from a range of 0 to 9999.

Instead of building a calculator circuit using electrical components, the logic for the calculator is controlled by the 8051 MCU. The MCU can be programmed to perform virtually any operation based on the inputs that it receives from its ports. In this example, the MCU is used to keep track of input values in its memory and the current state of its operation. It also performs arithmetic operations on 16-bit numbers that include addition, subtraction, multiplication and division. Since the 8051 assembly instructions operate on hexadecimal values, the MCU is also programmed to perform hexadecimal to BCD conversions and back again in order to display the input data and results in BCD format for the user.

- To activate the circuit and perform a simple calculation (12+3) on the calculator:
 1. Select **Simulate/Run** to begin simulation. The assembled code for the 8051 MCU is loaded at this point and the HEX displays are set to 0000. The MCU is in the “ready” state and is scanning its input port P1, waiting for a key press.

MCU Based Calculator

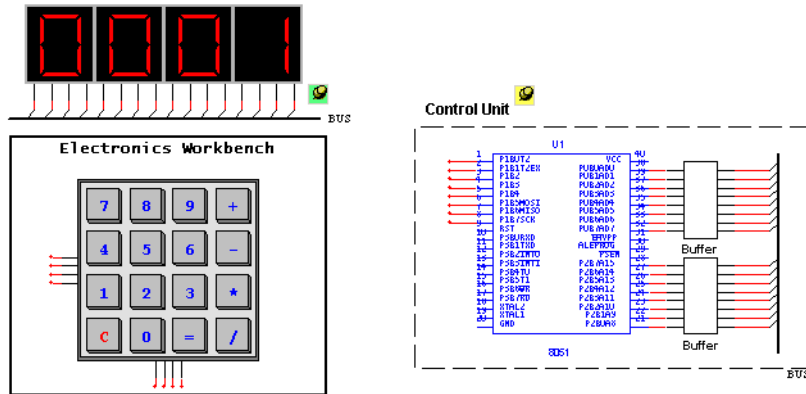
Place your mouse pointer over the colored thumbtacks for hints.



- Press '1' on the keyboard. The MCU detects this value on P1 and starts processing the high and low input values. The number that was pressed is determined and is displayed on the HEX displays.

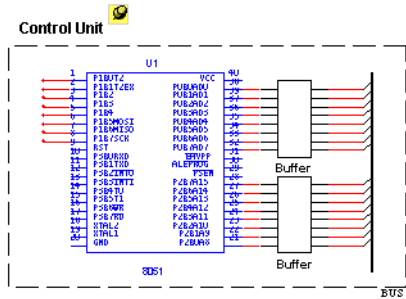
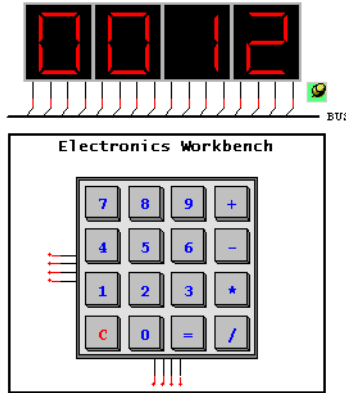
MCU Based Calculator

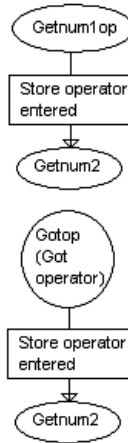
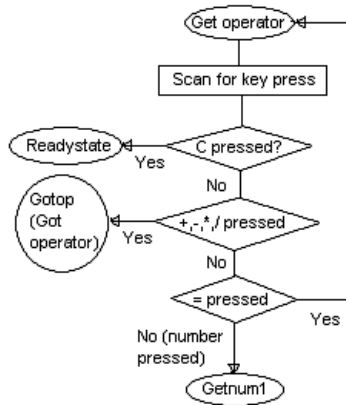
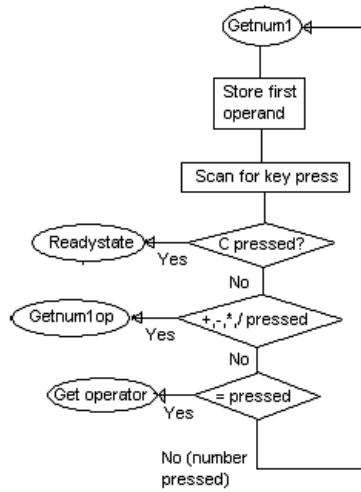
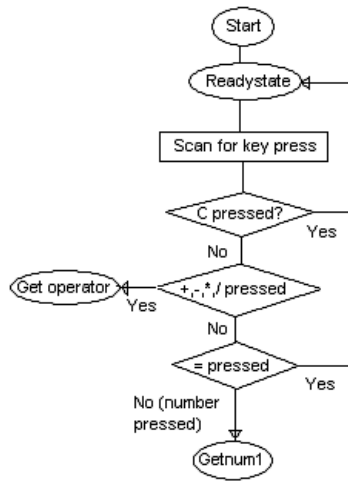
Place your mouse pointer over the colored thumbtacks for hints.

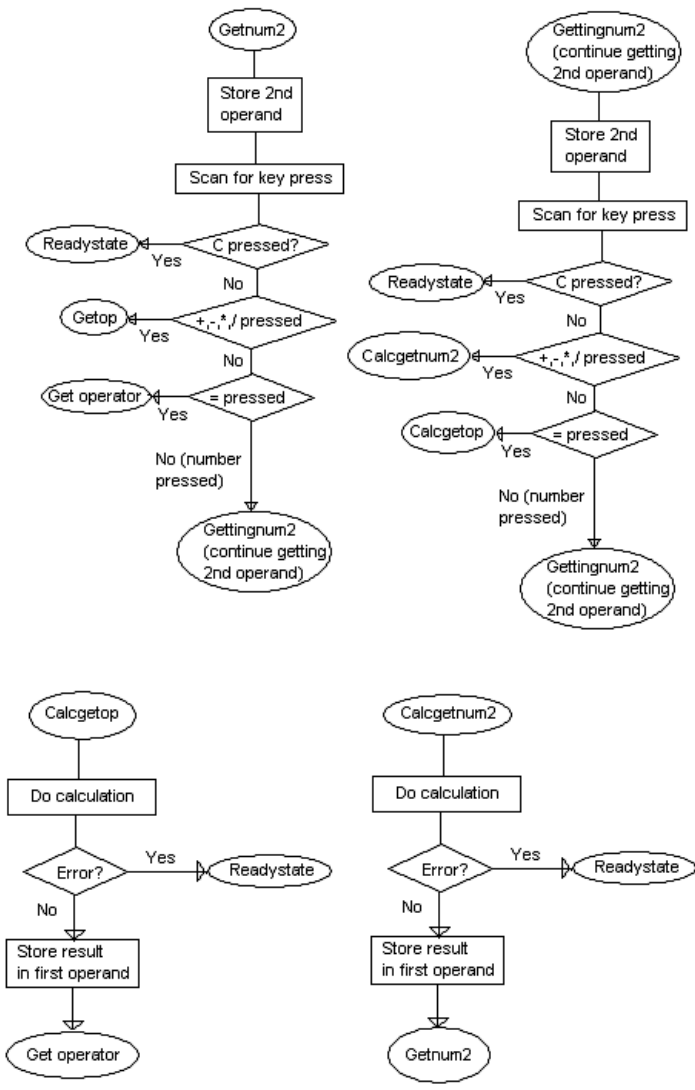


MCU Based Calculator

Place your mouse pointer over the colored thumbtacks for hints.






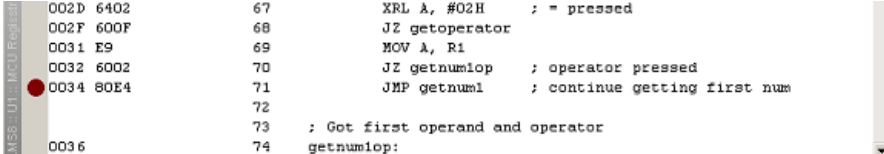


2.3.3 Advanced Features - Calculator Example

This section provides a step by step walkthrough of the MultiMCU debugging features. It is important to follow the steps exactly as scripted otherwise the descriptions will no longer apply. Once you understand how the breakpoint and single stepping features you can explore the possibilities of advanced MCU debugging.

2.3.3.1 Adding a Breakpoint

1. Select **Simulate/Run** to begin simulation.
2. Place your cursor on line 71 and then place a break point on the same line by clicking on the **Insert/Remove Breakpoint** button .




```

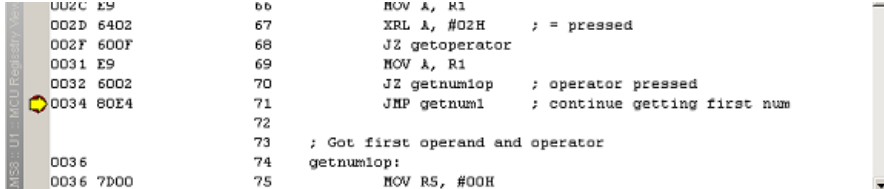
002D 6402      67      XRL A, #02H      ; = pressed
002F 600F      68      JZ getoperator
0031 E9        69      MOV A, R1
0032 6002      70      JZ getnum1op    ; operator pressed
0034 80E4      71      JMP getnum1     ; continue getting first num
              72
              73      ; Got first operand and operator
0036          74      getnum1op:

```

This will cause the program execution to stop when it reaches the JMP getnum1 instruction.

Tip: If the line numbers are not showing click the **Show Line Numbers** button .

3. Enter a number on your keyboard to cause a value to be entered on the keypad part.
4. Enter a second number on your keyboard.
5. The simulation will now pause at line 71.




```

002C E9        66      MOV A, R1
002D 6402      67      XRL A, #02H      ; = pressed
002F 600F      68      JZ getoperator
0031 E9        69      MOV A, R1
0032 6002      70      JZ getnum1op    ; operator pressed
0034 80E4      71      JMP getnum1     ; continue getting first num
              72
              73      ; Got first operand and operator
0036          74      getnum1op:
0036 7D00      75      MOV R5, #00H



```

6. As soon as the debugger has paused, the values in the MCU Register View and the **MCU Memory View** are updated to reflect the current values in the SFRs and the memory. Notice that R1 in the RAM contains a value of 1. It was moved into the accumulator on line 69. Also, notice that the accumulator (ACC) contains a value of 1.


Line 70, JZ getnum1op, tests the value of the ACC to determine if it is zero. If it is zero, then it jumps to the “getnum1op” label. Since the ACC is 1, it proceeds to the next line instead. This is what happened, otherwise line 71 would never have been reached and the simulation would not have paused.


7. In this way you can find the relationship between events that occur in the simulation and the routines that handle them in the assembly program. This is useful in figuring out how the program works for learning purposes or in debugging a problem. It can also be used to test and verify the correctness of your code.
8. You can remove a particular break point by placing the cursor on the line at that break point and clicking on the **Insert/Remove Breakpoint** button again or you can remove all break points that you have placed by clicking on the **Remove All Break points** button  .

2.3.3.2 Break and Continue

1. The debugger can be paused during simulation by clicking on the **Pause Simulation** button in the schematic capture tool bar. Do this and you will see the yellow arrow point to the instruction where the code execution has stopped in the **MCU Assembly Source View**.
2. Another way of pausing the simulation is to click on the **Break Execution** button  . The **MCU Assembly Source View** will show the instruction that it has stopped at in the same way. For this particular example, you will break inside the key scanning code since during idle time, that's what the calculator is doing.
3. To continue code execution after the simulation is paused, click on the **Go** button  , or click on the **Pause Simulation** button again in the schematic capture tool bar.
4. Another useful way of using the break and continue feature is for looping routines. Restart the simulation and enter “32 / 8” into the calculator on your keyboard.
5. Place a break point on line 775 just inside the DIV_LOOP label in the UDIV16 subroutine that will be called when you perform a divide operation and a break point on line 807 where it stops looping back to the DIV_Loop label.
6. Press the “=” key on your keyboard and see how it breaks at line 775.
7. Click on the **Go** button to continue and see that it breaks at line 775 again. You can look at the Register and Memory Views to see the updated values used in the UDIV16 subroutine to understand how it works.
8. You can also see how many times this loop is executed, by pressing the **Go** repeatedly until the DIV_LOOP loop exits and breaks at the second break point on line 807. (In general, if your program never exits a loop, then you may have an infinite loop on your hands.)
9. Remove all break points and click on the **Go** button one more time to see the answer “4” displayed on the HEX displays.

2.3.3.3 Break and Step

1. Select **Simulate/Run** to begin simulation again and enter “1 + 4” in to the keypad.
2. Clear all the break points by clicking on the **Remove All Break points** button  .

3. Go to the **MCU Assembly Source View** and scroll to line 229 where it is about to call the ADD16 subroutine begins.
4. Place a break point there by clicking on the **Insert/Remove Breakpoint** button.
5. Press the “=” key on the keyboard.
6. The program should now break at line 229 as it is about to do the add calculation.
7. You can enter the ADD16 subroutine by clicking on the **Step** button . This feature allows you to go into a call to a subroutine such as ADD16 and see what’s going on instead of executing the ADD16 subroutine as one step.
8. The arrow now jumps from line 229 to line 662 where the ADD16 subroutine starts.
9. You can step through the instructions by clicking on the **Step** button each time.
10. The ADD16 subroutine adds the value in R0, R1 to the value in R2, R3. Watch as the contents of these input registers are moved to the accumulator one by one and the sums obtained as you step through more instructions. The values shown in the internal RAM at 00H to 03H show the values inside R0, R1, R2 and R3 respectively. You can also watch the accumulator value change as the contents of the registers are moved and added to it in the data window. The final result is returned in R0 and R1. R0 should contain 05H and R1 should contain 00H.
11. Step all the way to line 673 RET.
12. Step one more time and the routine will return from the call, back to line 231 just after the call to ADD16.

2.4 Tutorial 4 - MCU Serial Terminal

This example uses an 8052 MCU to communicate with a virtual terminal via the serial ports. The virtual terminal detects characters entered by the user on the keyboard and transmits them to the MCU. On reception of a character, the MCU echos it back to the virtual terminal to be displayed in the virtual terminal window.

2.4.1 Overview

This circuit demonstrates the use of timers, interrupts and serial communication features of the 8052 MCU. The logic for the MCU is contained in the assembly program that is loaded into the MCU when the simulation begins.

The circuit consists of an 8052 MCU hooked up to a virtual terminal via serial ports P2B0RxD and P2B1TxD. The virtual terminal is a special part that consists of a virtual terminal window where you can type characters on your keyboard. When the simulation is

running, the virtual terminal does not normally display the characters that you type into it. The terminal just sends the characters that are typed into its window through its TxD pin at the baud rate that it is set to in its properties dialog. The terminal displays any characters that it receives through its RxD pin. Please note that characters are transmitted and received in the same format used for serial communication in mode 1 of the 8052 MCU. In this mode, a start bit is sent first, followed by the 8 bits of the character with the least significant bit being sent first, and finally a stop bit. In this example, the MCU is programmed to echo characters that it receives back to the virtual terminal to be displayed.

2.4.2 About the Tutorial

In order to communicate with the virtual terminal hooked up to the 8052 MCU, the input and output pins of the virtual terminal are hooked up to the TxD and RxD pins of the MCU respectively. The P2B1TxD and P2B0RxD pins on the 8052 are the UART pins used for serial communication.

This example uses serial communication mode 1 of the MCU set for a baud rate of 1.2 Kbps to send and receive data from the virtual terminal, which is also configured to send and receive data at the same rate. To set up the MCU for 1.2 Kbps communication, timer 1 is set to 8-bit auto-reload mode with a reload value of E5 in hexadecimal to do the timing.

```
; start timer 1 and then start serial rx in mode 1
CLR    RCLK          ; rx uses timer 1
MOV    TMOD, #020H  ; 8-bit auto reload
MOV    TH1, #0E5H   ; 1.2K baud rate
MOV    TL1, #0E5H
SETB   TR1          ; start timer 1
CLR    SM0           ; set serial mode to 1
SETB   SM1
CLR    RI
CLR    TI
SETB   REN
```

The MCU is also configured to use serial interrupts to notify it whenever a whole character has been received from the virtual terminal.

```
SETB   ES           ; enable serial interrupt
SETB   EA           ; enable interrupts in general
```

When a serial interrupt occurs, the MCU executes an interrupt service routine whose purpose is to echo the character back to the virtual terminal. This is achieved by sending the character

out through the TxD pin of the UART at the same 1.2 Kbps baud rate. The virtual terminal displays everything that it receives in the virtual terminal window. After the interrupt service routine finishes transmitting the character to the virtual terminal, it returns from the routine and waits in a loop until the next character is received and the whole process of echoing the character back is repeated.

```
ORG    0023H    ; serial ISR address
LJMP   int_serial
```

```
int_serial:
```

```
PUSH   PSW
PUSH   ACC
```

```
JNB    RI, exit_int ; check whether RI was set
; received data on pin RxD
MOV    A, SBUF ; SBUF contains received data
CLR    RI      ; reset RI bit in order to receive more data
```

```
; echo data received on RxD through TxD
MOV    SBUF, A
JNB    TI, $
CLR    TI
```

```
exit_int:
```

```
POP    ACC
POP    PSW
RETI   ; return from interrupt service routine
```

To activate the circuit, assemble the program by clicking on the **Build** button. Once the code has been assembled successfully, click on the **Go** button in the **MCU Assembly Source View**. Open up the window of the attached oscilloscope. The green line represents the data sent from the TxD pin of the virtual terminal to the MCU and the red line represents the data sent from the MCU back to the RxD pin of the virtual terminal. No data should be transmitted at this time.

Go to the virtual terminal window and click on it to set the focus. Type the letter 'h' into your keyboard and watch the oscilloscope window. You will see the bits of the letter 'h' being transmitted to the MCU in the green line. As soon as the letter 'h' finishes being transmitted,

the MCU sends back the letter 'h' to the virtual terminal as shown in the red line. The virtual terminal then displays the letter 'h' in its window. Try typing more letters into the window to spell out “hello world”. You will see that the subsequent letters are transmitted, echoed back and displayed in the virtual terminal as well.

Notice that the length of each bit shown in the oscilloscope window is about 833 microseconds in duration. This verifies that the data is being transmitted at a baud rate of 1.2 Kbps.


2.4.3 Using the MCU Interface

The MCU debugging tools provide the user with the ability to control execution at the instruction level (breakpoints and single-stepping) while also providing views of the data memory and registers within the MCU. It can be useful for understanding the echo example better. Here are a few things that you can try:

1. Place a breakpoint inside the serial interrupt service routine on line 43 by clicking on that line and clicking on the **Insert/Remove Breakpoint** button in the **MCU Assembly Source View**:

```
LJMP int_serial
```

When no data is being transmitted or received by the MCU, the program should never break here.

Tip: If the line numbers are not showing click the **Show Line Numbers** button  .

2. Type a character into the Virtual Terminal Window. Once it has finished transmitting the character to the MCU, the program will pause at the break point that you've set on the line indicated above since the interrupt should be triggered. Remove the break point by clicking on the **Insert/Remove Breakpoint** button again. Continue executing the program by clicking on the **Go** button.
3. Now try to pause the program by clicking on **Break Execution** button in the **MCU Assembly Source View**. You should see the program has paused in the loop. It will stay in this loop until another interrupt occurs at which point it will jump into the interrupt service routine again.
4. Take a look at the MCU Register View and find the TH1 and TL1 SFRs(Special Function Registers). The TL1 SFR is used by Timer 1 to count up.

When it reaches FF hexadecimal, then it rolls over and reloads itself with the value in TH1 since the MCU is configured for Timer 1 mode 2. Take note of their values. Now place a breakpoint on the “JMP endloop” line. Press the **Go** button and when the program breaks again at the same line, look at the values of TH1 and TL1. TL1 should have changed since some time has passed. This is one way to observe the timer feature in action.

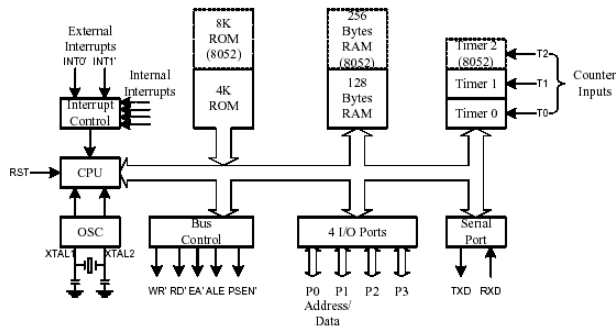
Appendix A - MultiMCU Parts

This appendix contains information on components that are added to the Multisim database during the MultiMCU installation process.

A.1 8051/8052 Microcontroller Units

40	P1B0T2	VCC	38	40	P1B0T2	VCC	38
41	P1B1T2EX	P0B0AD0	37	41	P1B1T2EX	P0B0AD0	37
42	P1B2	P0B1AD1	36	42	P1B2	P0B1AD1	36
43	P1B3	P0B2AD2	35	43	P1B3	P0B2AD2	35
44	P1B4	P0B3AD3	34	44	P1B4	P0B3AD3	34
1	P1B5M0SI	P0B4AD4	33	1	P1B5M0SI	P0B4AD4	33
2	P1B6M1SO	P0B5AD5	32	2	P1B6M1SO	P0B5AD5	32
3	P1B7SCK	P0B6AD6	31	3	P1B7SCK	P0B6AD6	31
4	RST	P0B7AD7	30	4	RST	P0B7AD7	30
5	P3B0RxD	EA/PP	29	5	P3B0RxD	EA/PP	29
7	P3B1TxD	ALE/PROG	27	7	P3B1TxD	ALE/PROG	27
8	P3B2INT0	PSEN	26	8	P3B2INT0	PSEN	26
9	P3B3INT1	P2B7A15	25	9	P3B3INT1	P2B7A15	25
10	P3B4T0	P2B6A14	24	10	P3B4T0	P2B6A14	24
11	P3B5T1	P2B5A13	23	11	P3B5T1	P2B5A13	23
12	P3B6WR	P2B4A12	22	12	P3B6WR	P2B4A12	22
13	P3B7RD	P2B3A11	21	13	P3B7RD	P2B3A11	21
14	XTAL2	P2B2A10	20	14	XTAL2	P2B2A10	20
15	XTAL1	P2B1A9	19	15	XTAL1	P2B1A9	19
16	GND	P2B0A8	18	16	GND	P2B0A8	18

The 8051 and 8052 microcontrollers combine a CPU, data memory, program memory, and built-in external memory on a single chip.

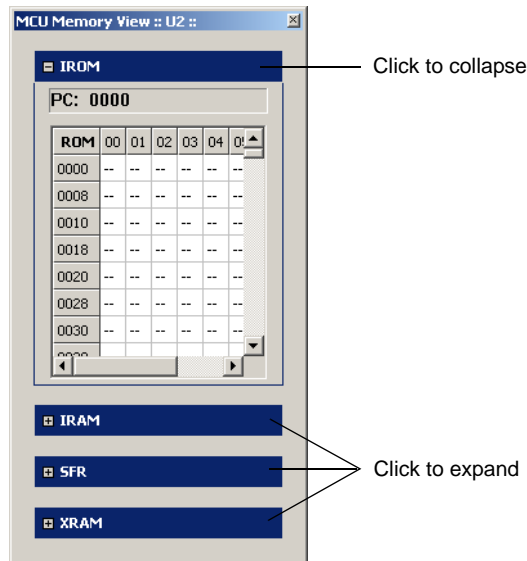


-
- To show/hide the elements of the 805x:
 1. Double-click on the placed MCU to display its properties dialog box and click on the **Display** tab.
 2. Enable the **Memory View** and **Assembly Source Window** checkboxes as desired and click on the **OK** button.

Note For details on the **MCU Assembly Source View**, see “1.3.1 MCU Assembly Source View” on page 1-2.
 - To change the values for the placed 805x:
 1. Double-click on the placed MCU to display its properties dialog box and click on the **Value** tab.
 2. Change the values as desired:
 - **Assembler** — the name of the assembler that is used to assemble the MCU source code. (Refer to the Documentation CD to view the user guide for the Metalink ASM51 cross assembler that is bundled with MultiMCU. The guide is also installed onto your system with MultiMCU).
 - **Built-in External RAM** — the external on-chip RAM for the MCU, as displayed in the **XRAM** section of the **MCU Memory View**.
 - **ROM Size** — the ROM for the MCU, as displayed in the **IROM** section of the **MCU Memory View**.
 - **Clock Speed** — the speed of the MCU’s internal clock.
 3. Click **OK** to close the dialog and accept the changes.

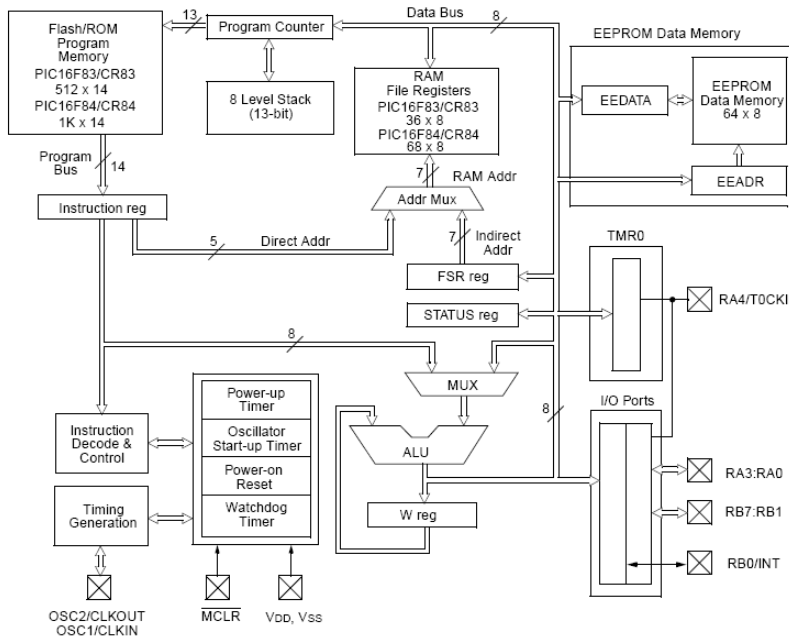
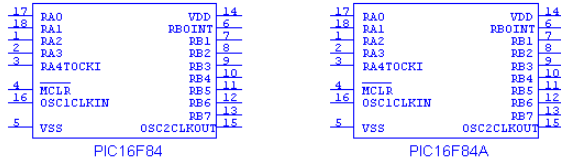
MCU Memory View

You can collapse/expand the fields in the **MCU Memory View** as shown below.



- **IROM** is the internal ROM (Read Only Memory) of the MCU. Shows the program memory code in hexadecimal format. These are the actual machine instructions that the simulation uses when it is activated. The left column shows the memory address and the header row shows the offset from the address on the left.
- **IRAM** is the internal RAM (Random Access Memory) of the MCU. Shows the data that is inside the MCU's memory and is modified as the program runs. The green shaded area shows the four "R" register banks; the currently selected one is brighter than the other three.
- **SFR** is the MCU's Special Function Registers.
- **XRAM** is the MCU's external on-chip RAM.

A.2 PIC16F84/16F84A Microcontroller Units



➤ To show/hide the MCU Memory View:

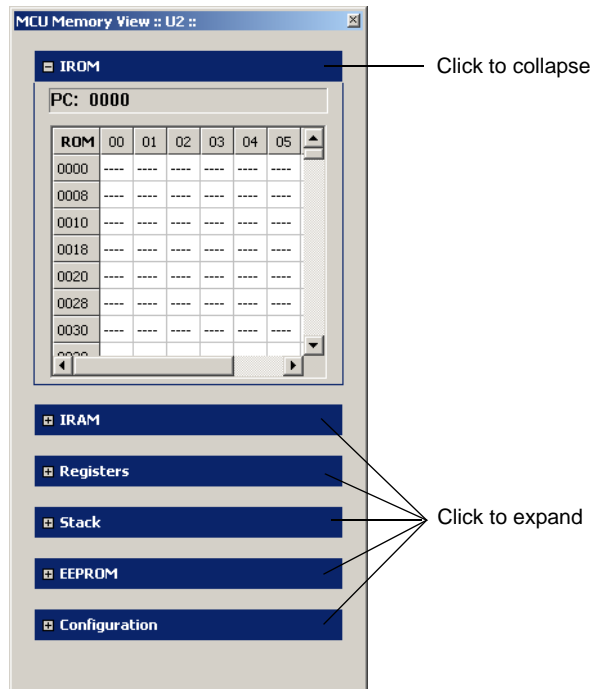
1. Double-click on the placed MCU to display its properties dialog box and click on the **Display** tab.
2. Enable the **Memory View** and **Assembly Source Window** checkboxes as desired and click on the **OK** button.

Note For details on the **MCU Assembly Source View**, see “1.3.1 MCU Assembly Source View” on page 1-2.

- To change the values for the placed PIC16F84/A:
1. Double-click on the placed MCU to display its properties dialog box and click on the **Value** tab.
 2. Change the values as desired:
 - **Assembler** — the name of the assembler that is used to assemble the MCU source code. (Refer to the Documentation CD to view the user guide for the MPASM cross assembler that is bundled with MultiMCU. The guide is also installed onto your system with MultiMCU).
 - **Clock Speed** — the speed of the MCU's internal clock.
 3. Click **OK** to close the dialog and accept the changes.

MCU Memory View

You can collapse/expand the fields in the **MCU Memory View** as shown below.



- **IROM** is the internal ROM (program memory) of the MCU. Shows the program memory code in hexadecimal format. These are the actual machine instructions that the simulation uses when it is activated. The left column shows the memory address and the header row shows the offset from the address on the left.

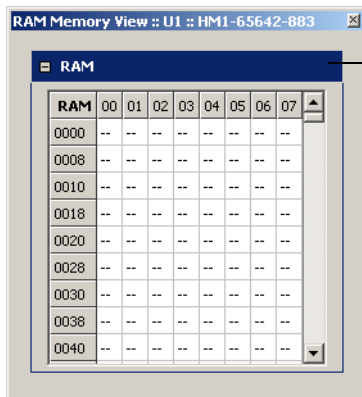
- **IRAM** contains the internal RAM (Random Access Memory) of the MCU. Shows the data that is inside the MCU's memory and is modified as the program runs. The colored areas represent SFRs (Special Function Registers), GPRs (General Purpose Registers) and different memory banks.
- **Registers** contains the SFRs.
- **Stack** contains the processor stack view.
- **EEPROM** (Electrically Erasable Programmable Read Only Memory) contains the data EEPROM.
- **Configuration** contains the PIC configuration bits.

A.3 RAM

The **RAM Family** in the **MultiMCU Group** contains a number of RAM devices for use with MultiMCU's MCU devices.

10	A0	D00	11
9	A1	D01	12
8	A2	D02	13
7	A3	D03	14
6	A4	D04	15
5	A5	D05	16
4	A6	D06	17
3	A7	D07	18
25	A8		19
24	A9		
21	A10		
23	A11		
2	A12		
27	W		
22	E		
20	E1		
26	E2		

HM1-65642-883



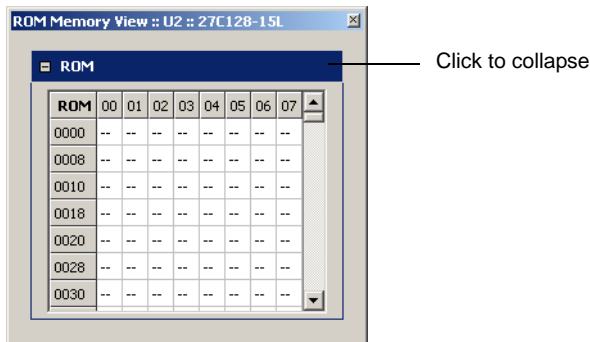
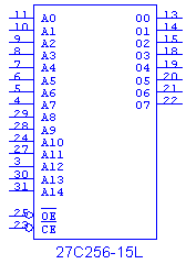
Click to collapse

The **RAM Memory View** is used to view the contents of the RAM chip while debugging.

- To show/hide the **RAM Memory View**:
 1. Double-click on the placed RAM device and click on the **Display** tab.
 2. Enable/disable the **Memory View** checkbox as desired and click on the **OK** button.













A.4 ROM

The **ROM Family** in the **MultiMCU Group** contains a number of ROM devices for use with MultiMCU's MCU devices.



The **ROM Memory View** is used to view the contents of the ROM chip while debugging.

The **XROM Assembly Source View** shows the assembly source code for the ROM chip and contains the following buttons:

Button	Description
	New button. Clears the contents of the external ROM.
	Import button. Imports an assembly file (.asm) that you created in a text editor into the XROM Assembly Source View .
	Export button. Exports the contents of the XROM Assembly Source View into an assembly text file (.asm).
	Cut button. Removes the selection from the XROM Assembly Source View and places it on the clipboard.
	Copy button. Copies the selection onto the clipboard.
	Paste button. Pastes the contents of the clipboard into the insertion point in the XROM Assembly Source View .
	Undo button. Undoes the previous action.
	Redo button. Redoes the most recently performed undo action.
	Build button. Assembles the code in the XROM Assembly Source View .
	Show Line Numbers button. Displays the line numbers in the XROM Assembly Source View .
	Find button. Displays the Find dialog box, where you can search for text strings in the XROM Assembly Source View .
	Print button. Prints the contents of the XROM Assembly Source View .

➤ To show/hide the elements of a ROM device:

1. Double-click on the placed ROM device and click on the **Display** tab.
2. Enable/disable the **Memory View** and **Assembly Source Window** checkboxes as desired and click on the **OK** button.

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Online technical support resources at ni.com/support include the following:
 - **Self-Help Resources**—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
 - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at ni.com/exchange. National Instruments Application Engineers make sure every question receives an answer.

For information about other technical support options in your area, visit ni.com/services or contact your local office at ni.com/contact.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.