multiMCU

Microcontroller Co-simulation

Electronics
WORKBENCH

A NATIONAL INSTRUMENTS COMPANY

MCU-E-1790 Rev. 1

Table of Contents

1. MultiMCU Overview

2. MultiMCU Sample Walkthroughs

Chapter 1
MultiMCU 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 RAM and Registers of the MCU, set code breakpoints, and single step through your code.

For detailed demonstrations of the MultiMCU functionality see:

- "2.1 Tutorial 1 - MCU Driven Blinking Lights" on page 2-2
- "2.2 Tutorial 2 - MCU Controlled Holding Tank" on page 2-13
- "2.3 Tutorial 3 - MCU Based Calculator" on page 2-22

Chapter 2
MultiMCU Sample Walkthroughs

This chapter details three tutorials that use MultiMCU's co-simulation functionality. The circuits for the tutorials are found in the folder where you installed Multisim 8, at `...\samples\MCU Sample Circuits`.

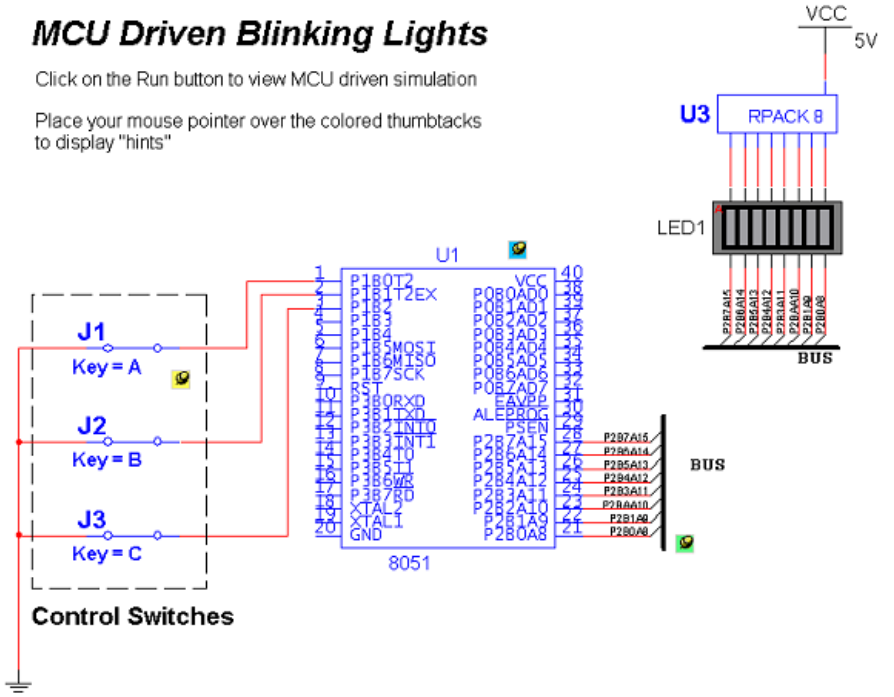
The following are described in this chapter.

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.



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 (U3) 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 full source code for this MCU sample can be found in the Blink.asm file. The following excerpt of that source code 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:

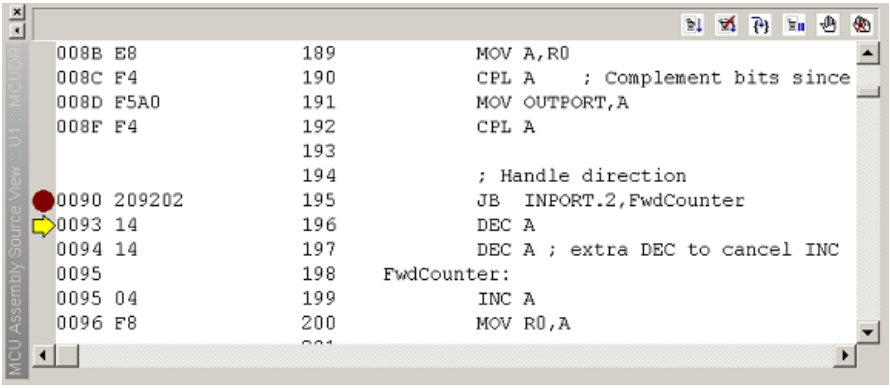
- A delay subroutine is called to slow down the pattern. The sample MCU is using an internal 12MHz oscillator. Without the delay the lights would be changing too quickly.
- A typical MCU component can sink more current than it can source so the bar LED was arranged to match. However, people usually equate 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 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.

2.1.3.1 MCU Assembly Source Window

The MCU Assembly Source Window 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 edit dialog will automatically scroll to the place in the assembly code where the simulation has paused and indicate the current instruction with an arrow.

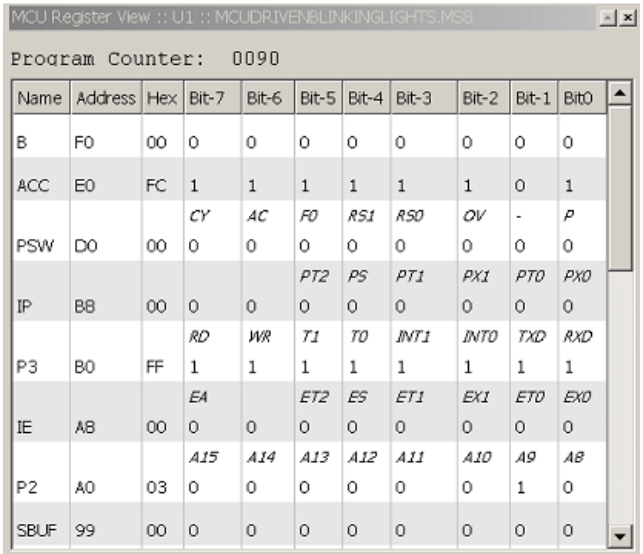


The figure here shows the MCU Assembly Source Window showing the annotated source code of part of the same Counter Loop code shown earlier. Only the annotated source is shown since editing assembly code is disabled in the sample.

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 Window are buttons to control execution. The buttons as shown (from left to right respectively) are **Go**, **Stop Debugging**, **Step Into**, **Break Execution**, **Insert/Remove Breakpoint**, and **Remove All Breakpoints**.

2.1.3.2 MCU Register View

When the program is paused on a breakpoint, the Special Function Registers (SFRs) are displayed in the MCU Register View as shown below:

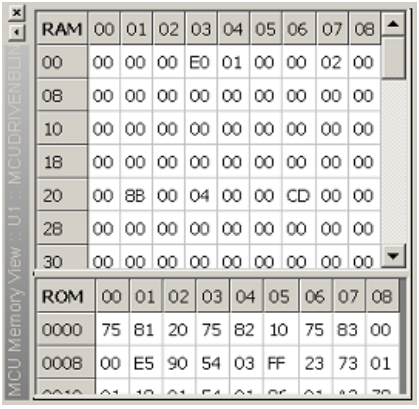


Name	Address	Hex	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
B	F0	00	0	0	0	0	0	0	0	0
ACC	E0	FC	1	1	1	1	1	1	0	1
			CY	*AC*	*F0*	*RS1*	*RS0*	*OV*	-	*P*
PSW	D0	00	0	0	0	0	0	0	0	0
					PT2	*PS*	*PT1*	*PX1*	*PT0*	*PX0*
IP	B8	00	0	0	0	0	0	0	0	0
			RD	*WR*	*T1*	*T0*	*INT1*	*INT0*	*TXD*	*RXD*
P3	B0	FF	1	1	1	1	1	1	1	1
			EA		*ET2*	*ES*	*ET1*	*EX1*	*ET0*	*EX0*
IE	A8	00	0	0	0	0	0	0	0	0
			A15	*A14*	*A13*	*A12*	*A11*	*A10*	*A9*	*A8*
P2	A0	03	0	0	0	0	0	0	1	0
SBUF	99	00	0	0	0	0	0	0	0	0

The first column contains the name of the Special Function Register (SFR) and the second column displays the address of the SFR. The third column shows the value contained in the SFR. For example, the accumulator (ACC) which is used very often, contains a value of 0FCH. When the current instruction DEC A is executed, the value in ACC will be decremented and become 0FBH. Some SFRs such as the program status word (PSW) contain bits that are used individually for different purposes. This can be seen in the remaining columns that display each bit individually.

2.1.3.3 MCU Memory View

The MCU Memory View displays the values in the internal RAM and ROM of the 8051 MCU.



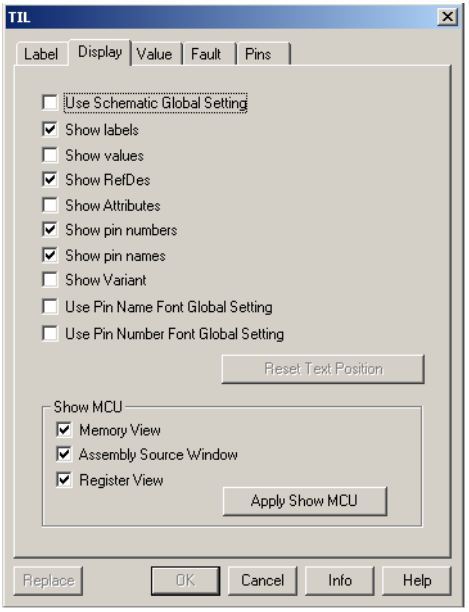
The ROM window shows the program memory code in hexadecimal format. These are the actual machine instructions that the simulation uses when it is activated. The left column shows the memory address and the header row shows the offset from the address on the left.

The internal RAM, displayed above the ROM, shows the data that is inside the MCU's memory and is modified as the program runs. RAM memory on the 8051 is used for four banks of registers R0 to R7, user variables, and stack memory. Addresses 00H to 19H contain the four register banks. By default, the 8051 uses the first register bank (00H to 07H) for the registers R0, R1 to R7, but this can be configured by the program. The Blinking Lights Sample uses registers in Bank 0 but begins the assembly code by setting the Stack Pointer (SP) to 20H which reserves all four register banks for future use. This is a standard and recommended practice for 8051 programming. In the figure you see that register R7 (at location 07H) contains the value 02H which is our current mode value.

2.1.3.4 Displaying Elements of the MCU Interface

➢ To show or hide elements of the MCU Interface:

1. Double-click on the 8051 MCU to display its properties dialog and click on the **Display** tab.



2. Enable the **Memory View**, **Assembly Source Window** and **Register View** checkboxes as desired and click on the **Apply Show MCU** button.

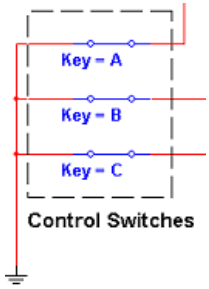
2.1.4 Advanced Features

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.

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.

2.1.4.1 Adding a Breakpoint

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



2. Scroll the MCU Assembly Source Window to the Counter Loop and move the cursor (by cursor keys or mouse click) to line 195. It shows:

   JB INPORT.2, FwdCounter

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

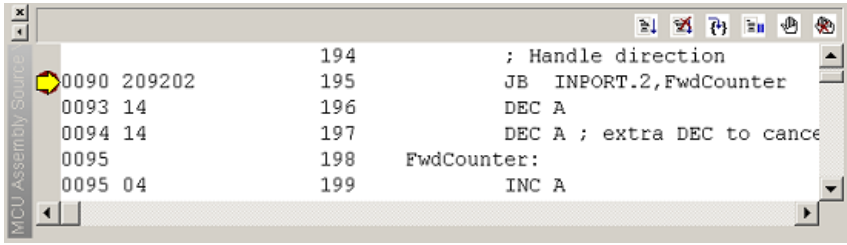
   You have now set a breakpoint at the branch instruction in the Counter mode loop that decides whether to increment or decrement the counter.

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 .

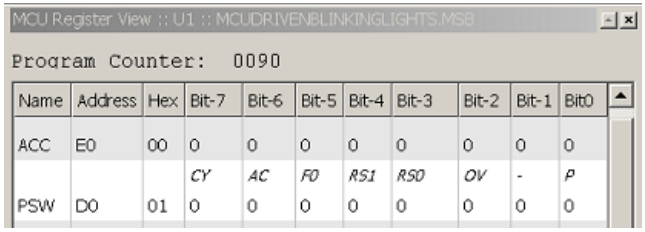
2.1.4.2 Break and Continue

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.

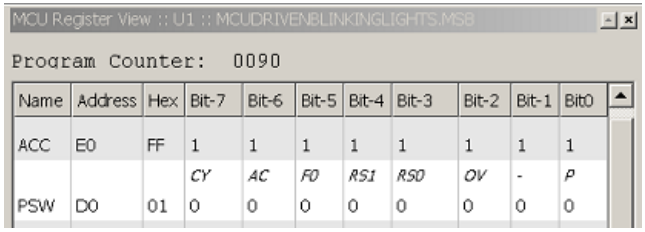
4. The simulator should be paused now and the Assembly Source Window will show the yellow arrow over our breakpoint.



5. Look at the Accumulator (ACC) value in the MCU Register View. The ACC is 00H at this moment.

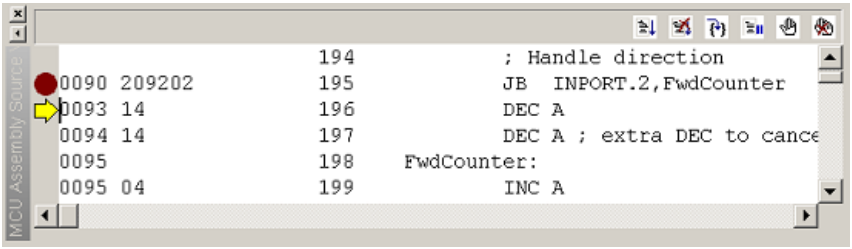


6. Click on the **Go** button  in the Assembly Source window. The Counter Loop will run for one iteration and stop again at our breakpoint. The new value of the accumulator is now 0FFH.



2.1.4.3 Break and Step Into

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



2. Since switch J3 (C-key) was closed, the input value of Port-1, Bit-2 will be zero. The test for bit-set failed as expected and execution continues after the JB instruction.
3. Click on the **Step Into** button a few more times and watch the Accumulator value change as the DEC and INC instructions are executed.

Setting Breakpoints During Simulation

1. Stop the Simulation by clicking on the **Stop Debugging** button .
2. Click on the **Remove All Breakpoints** button . This will clear all user breakpoints. All of the red breakpoint dots should disappear.
3. Select Simulate/Run to begin simulation.
4. Hit the A and B keys on the keyboard until both switches are open.
5. You should see the Marquis pattern scrolling across the Bar LED.
6. 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. At this time you can examine the memory and register views as demonstrated earlier.

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. Note the 8051 MCU completely replaces the ladder diagram in the same example (“Educational Sample Circuits\Ladder Diagrams\HoldingTankExample.ms8”) included in the Educational version of Multisim 8. The behavior of the ladder logic diagram is emulated by the 8051 MCU to achieve the same results.

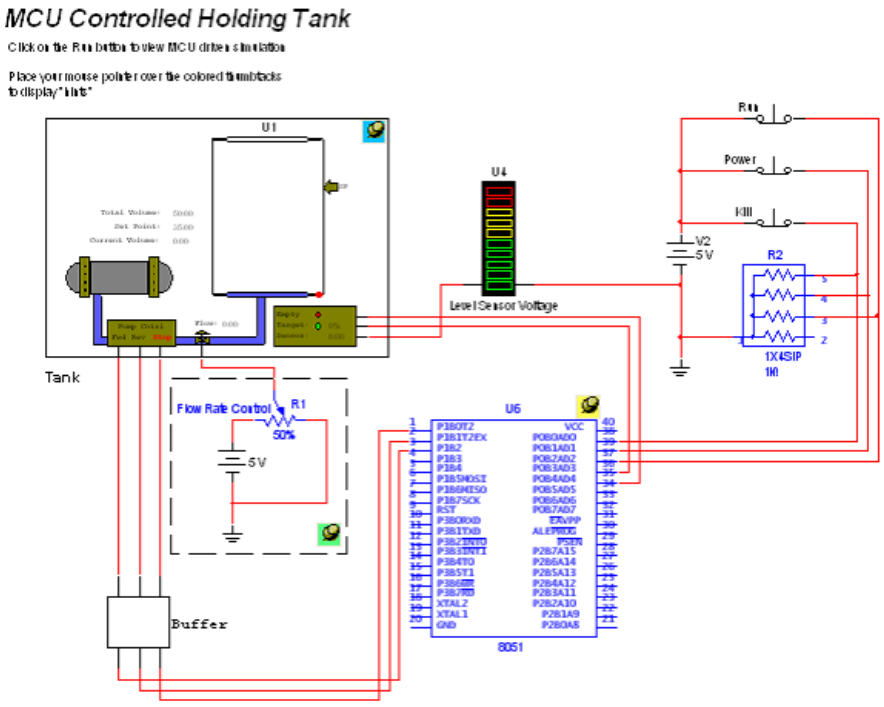
2.2.1 Overview

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.



➢ 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.

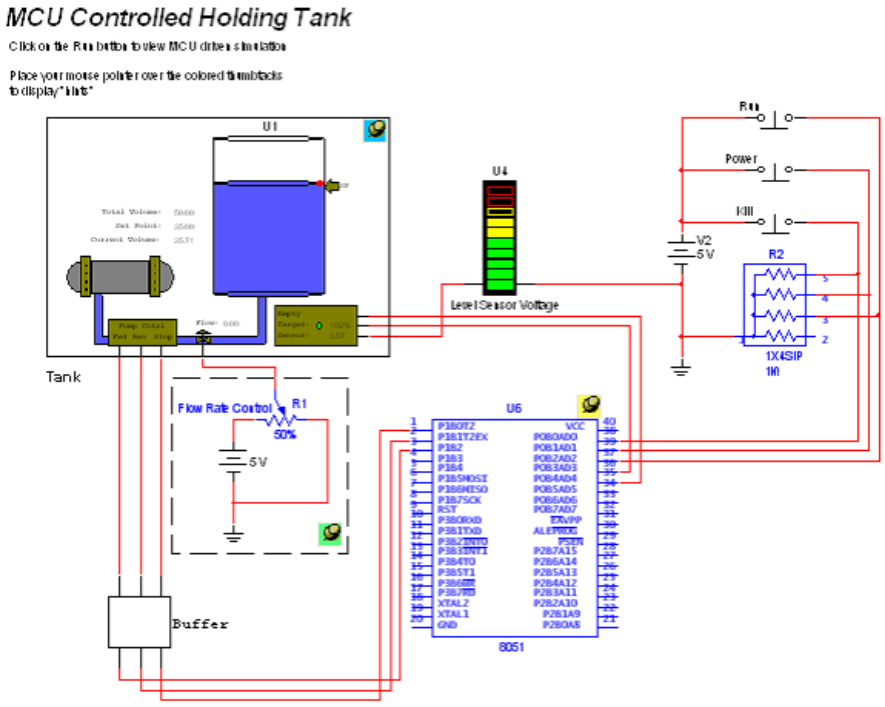
```
; Fill in forward direction
fillfwd:
      MOV P1,#004H          ; set fwd signal to high
      CALL outputdelay      ; hold fwd signal high
      CALL outputdelay
      MOV P1,#000H          ; set fwd signal back to low
```

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
```



4. A timer will start and after a delay of about 5 seconds, the tank begins to empty.

```
        CALL timerdelay    ; go to timer routine
        JMP fillrev        ; timer has finished, start draining
```

5. When the tank is empty, the MCU receives an empty signal on port P0 bit 4. The MCU sends a stop signal to the pump in turn and the flow stops.

```
; Fill in reverse direction (drain)
fillrev:
        MOV P1,#002H           ; set reverse signal to high
        CALL outputdelay       ; hold reverse signal
        CALL outputdelay
        MOV P1,#000H           ; set reverse signal to low
; Wait for tank to reach the empty point
fillrevloop:
        JB P0.0,fillrevkill    ; kill button pressed
```

```
        JB P0.4,fillrevend     ; empty point reached

        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 P0B0AD0. 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 occuring.

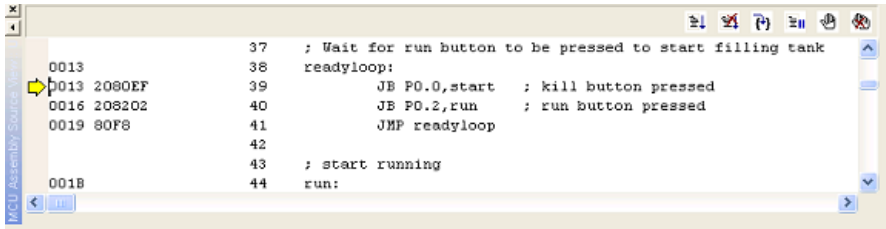
```
; Kill button was pressed during filling in fwd direction
fillfwdkill:
      MOV P1,#001H             ; set stop signal to high
      CALL outputdelay         ; hold top signal high
      CALL outputdelay
      JMP start                ; go back to beginning of program

; 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 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.

2.2.3.1 MCU Assembly Source Window

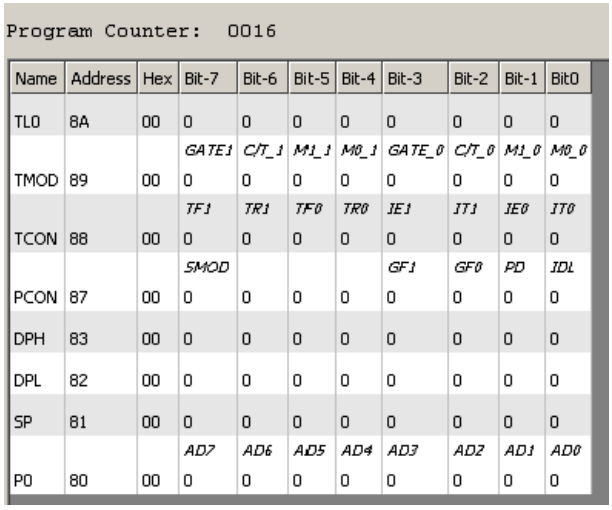
The MCU Assembly Source Window shows the assembly source code for the MCU program. This is also where you can set breakpoints to have the simulation pause at a particular location in the code. The edit dialog will automatically scroll to the place in the assembly code where the simulation has paused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



The figure here shows the MCU Assembly Source Window showing the annotated source code of part of the ‘readyloop’ code shown earlier. Only the annotated source is shown since editing assembly code is disabled in the sample. 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 yellow arrow indicates the instruction that the program has paused at. On the top right on the MCU Assembly Source Window are buttons to control execution. The buttons as shown (from left to right respectively) are **Go**, **Stop Debugging**, **Step Into**, **Break Execution**, **Insert/Remove Breakpoint**, and **Remove All Breakpoints**.

2.2.3.2 MCU Register View

When the program is paused on a break point, the Special Function Registers (SFRs) are displayed in the MCU Register View as shown below:



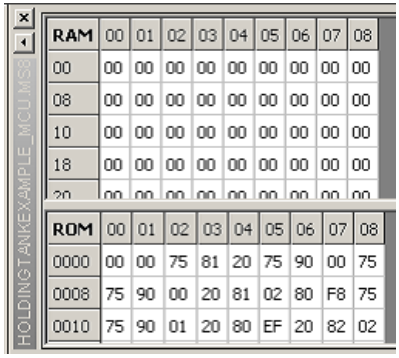
Program Counter: 0016

Name	Address	Hex	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
TL0	8A	00	0	0	0	0	0	0	0	0
			GATE1	C/T_1	M1_1	M0_1	GATE_0	C/T_0	M1_0	M0_0
TMOD	89	00	0	0	0	0	0	0	0	0
			TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
TCON	88	00	0	0	0	0	0	0	0	0
			SMOD				GF1	GF0	PD	IDL
PCON	87	00	0	0	0	0	0	0	0	0
DPH	83	00	0	0	0	0	0	0	0	0
DPL	82	00	0	0	0	0	0	0	0	0
SP	81	00	0	0	0	0	0	0	0	0
			AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
P0	80	00	0	0	0	0	0	0	0	0

The first column contains the name of the Special Function Register (SFR) and the second column displays the address of the corresponding SFR. The third column shows the value contained in the SFR. When you are paused at line 39 of the Holding Tank example, the input signals at port P0 should contain zeros in all its bits since no buttons have been pressed.

2.2.3.3 MCU Memory View

The MCU Memory View displays the values in the internal RAM and ROM of the 8051 MCU.



HOLDINGTANKEXAMPLE_MCU.MSE

RAM	00	01	02	03	04	05	06	07	08
00	00	00	00	00	00	00	00	00	00
08	00	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00
18	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00

ROM	00	01	02	03	04	05	06	07	08
0000	00	00	75	81	20	75	90	00	75
0008	75	90	00	20	81	02	80	F8	75
0010	75	90	01	20	80	EF	20	82	02

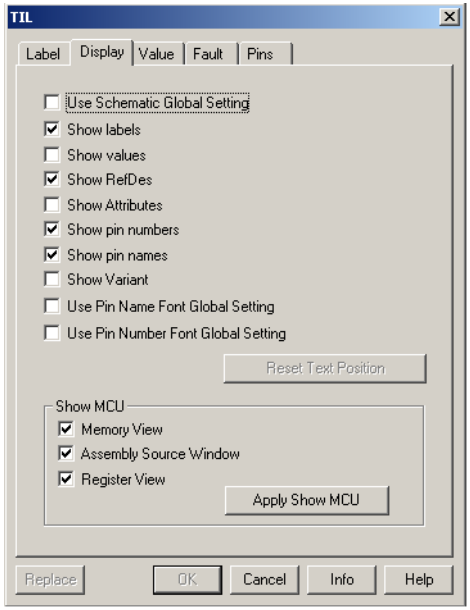
The ROM window shows the program memory code that is loaded in hexadecimal format. These are the actual machine instructions that the simulation is using when it is activated. The left column indicates the memory address and the header row indicates the offset from the address on the left.

The internal RAM, above the ROM, shows the data that is inside the MCU's memory and is modified as the program runs. RAM memory on the 8051 is used for four banks of registers R0 to R7, user variables, and stack memory. Addresses 00H to 19H contain the four register banks. By default, the 8051 uses the first register bank (00H to 07H) for the registers R0, R1 to R7, but this can be configured by the program. The Holding Tank example uses registers in Bank 0 but begins the assembly code by setting the Stack Pointer (SP) to 20H which reserves all four register banks for future use. This is a standard and recommended practice for 8051 programming. In the above figure, the program is paused and all the registers contain a value of zero.

2.2.3.4 Displaying Elements of the MCU Interface

➢ To show or hide elements of the MCU Interface:

1. Double-click on the 8051 MCU to display its properties dialog and click on the **Display** tab.



2. Enable the **Memory View**, **Assembly Source Window** and **Register View** checkboxes as desired and click on the **Apply Show MCU** button.

2.2.4 Advanced Features

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.

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.

2.2.4.1 Adding a Breakpoint

1. Load the MCU Controlled Holding Tank Example.
2. Scroll the MCU Assembly Source Window and move the cursor (by cursor keys or mouse click) to line 49.
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 51.
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  .

2.2.4.2 Break and Continue

1. Select Simulate/Run to begin 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 line 49.
5. Look at the MCU Register View and scroll until you can see P1 and notice that bit 2 is zero.
6. Click on the **Go** button  in the MCU Assembly Source Window. The program will execute until it reaches the next break point on line 51.
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 now after executing the instruction “MOV P1, #004H”.
8. Click on the **Go** button again and the simulation will continue filling the tank.

2.2.4.3 Break and Step Into

1. Place a break point on line 62. Line 62 is executed to start the 5 second timer when the tank is filled to the set point.
2. Select Simulate/Run to restart the simulation
3. Press ‘P’ on your keyboard to activate your circuit.
4. Press ‘R’ on your keyboard to start filling.
5. Eventually, the simulation will pause and the debugger will show the paused program execution at line 62.
6. Click on the **Step Into** button  to step into the subroutine 'timerdelay'.
7. The yellow arrow shows the next instruction that will be executed at line 110 inside the “timerdelay”.
8. If you click on the **Step Into** button again, JMP timerstart will execute and jump to line 96 where the “timerstart” code begins.
9. Click on the **Step Into** button one more time. See that the value in P1 bit 0 was set to 1 to stop the filling of the tank.
10. 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.

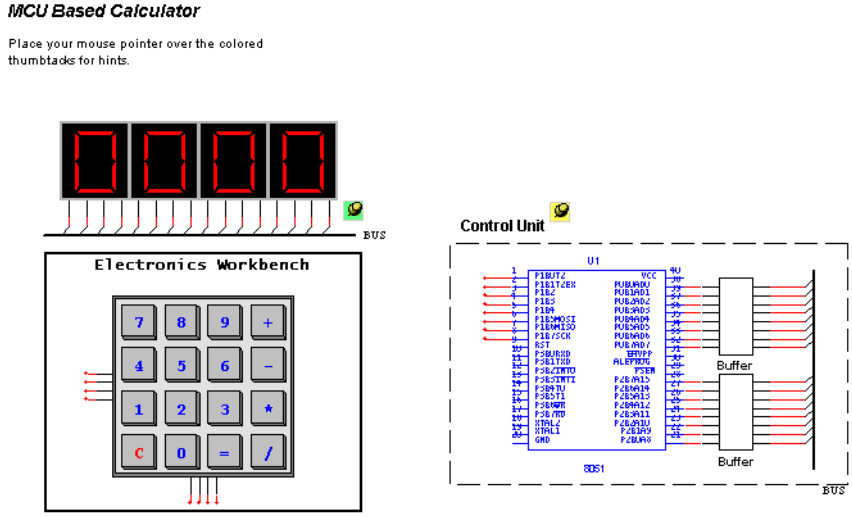
2.3.2 About the Tutorial

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:

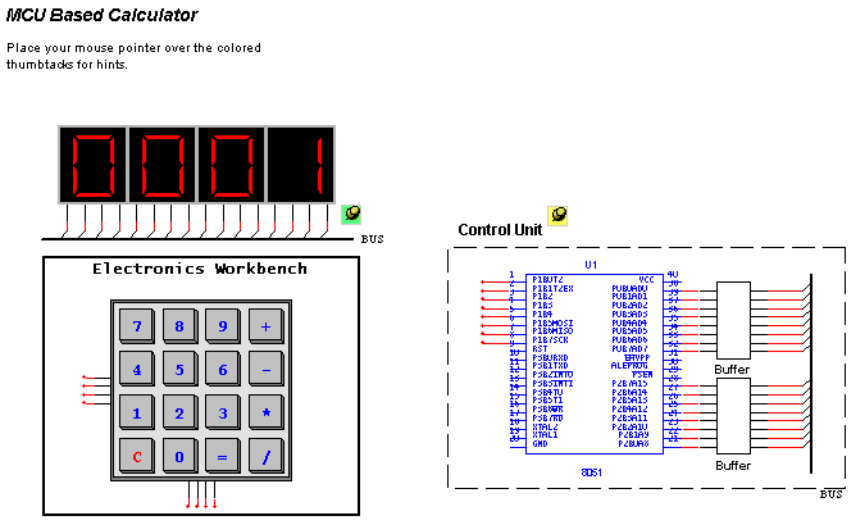
1. Select Simulate/Run to begin simulation. The assembled code for the 8051 MCU is loaded at this point and the HEX displays are set to 0000. The MCU is in the "ready" state and is scanning its input port P1, waiting for a key press.



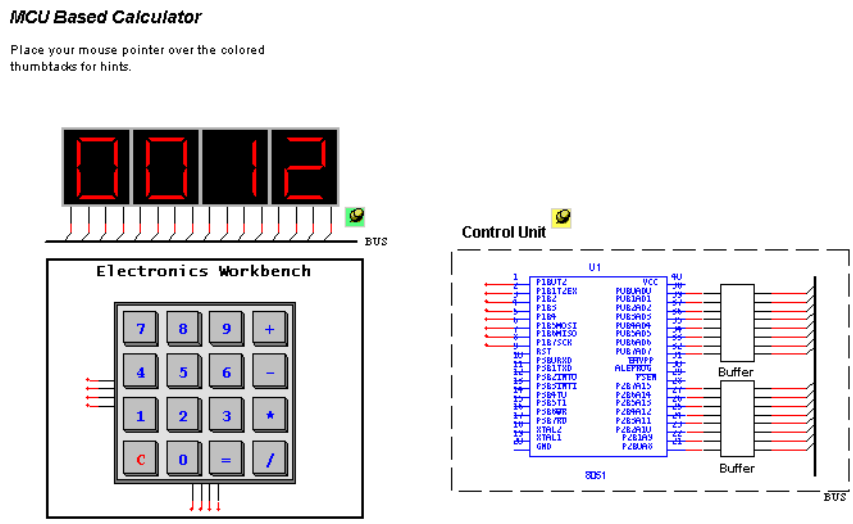
The lower 4 bits of input port P1are normally driven high and the higher 4 bits of the port are normally driven low. The MCU's assembly code waits in a loop polling those input values and waits for changes in them as shown below. It exits the loop as soon as a change is detected.

```
anykeyloop:
        MOV A, P1
        ANL A, #00FH
        XRL A, #00FH
        JZ anykeyloop
        MOV R0, A
```

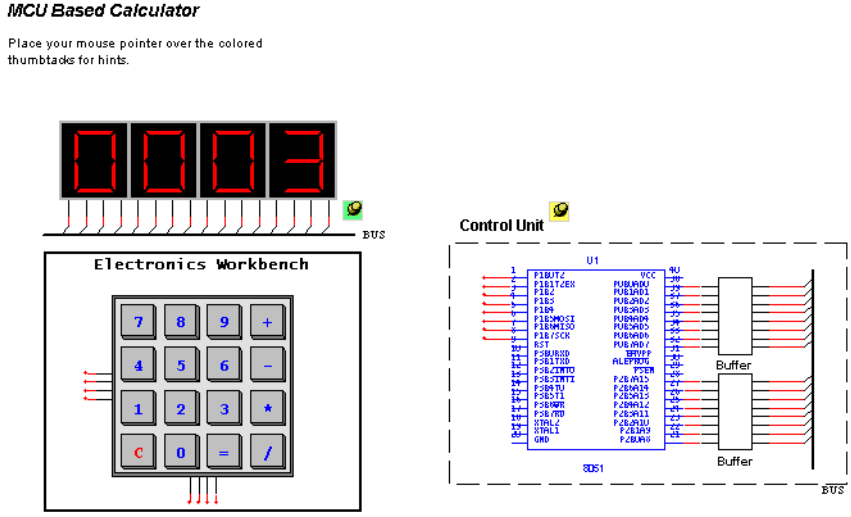
2. Press ‘1’ on the keyboard. The MCU detects this value on P1 and starts processing the high and low input values. The number that was pressed is determined and is displayed on the HEX displays.



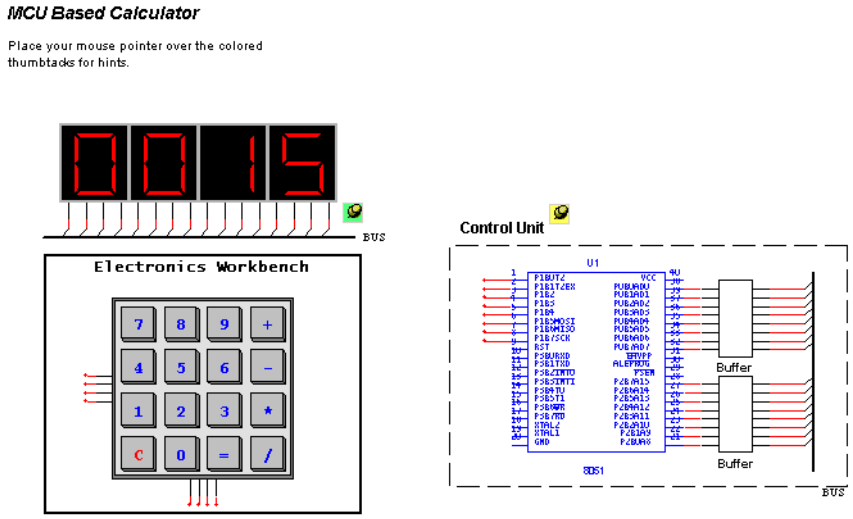
3. Press ‘2’ on the keyboard and see the number ‘2’ on the keypad depress. The MCU knows that the number 2 is still part of the first number that is being entered and shifts the ‘1’ displayed on the HEX display one to the left and displays the ‘2’ in the right most HEX display.



4. Press ‘+’ on the keyboard. The display remains the same since an operator was entered. The MCU stores the first complete number and the operator that was entered into its memory for later use.
5. Enter ‘3’ into the keyboard. Notice that the display clears and displays the new number 3.



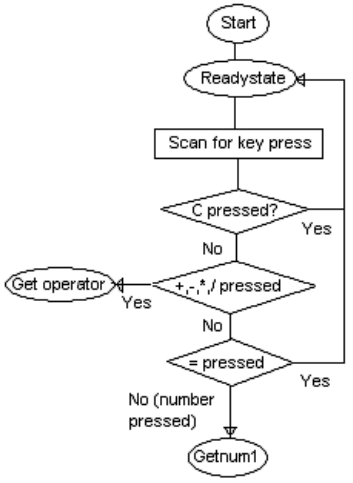
6. Press ‘=’ on the keyboard. The MCU retrieves the values of the operands and operator that it had stored previously. It calculates the result of 12+3 using a 16-bit addition function and converts the result from hexadecimal to the BCD value ‘15’, which is displayed on the HEX display.

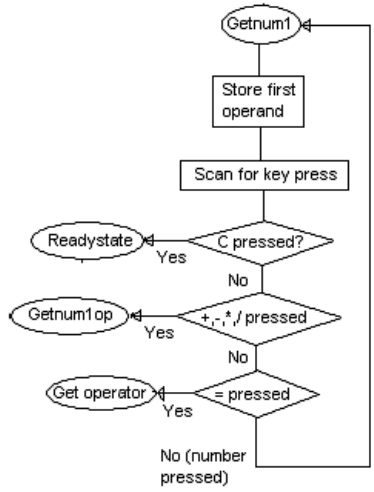


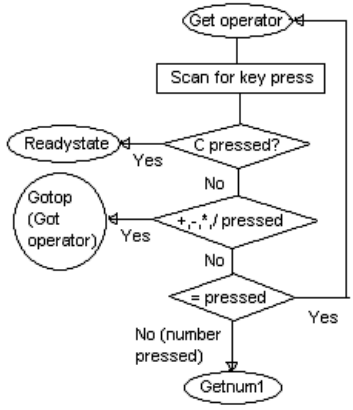
Understanding the calculator’s assembly code routines

The flow chart below provides a general overview of how the calculator works. States are shown in ovals, decisions are contained in diamonds, and operations are in rectangles. The names of the states correspond to actual labels in the assembly code.

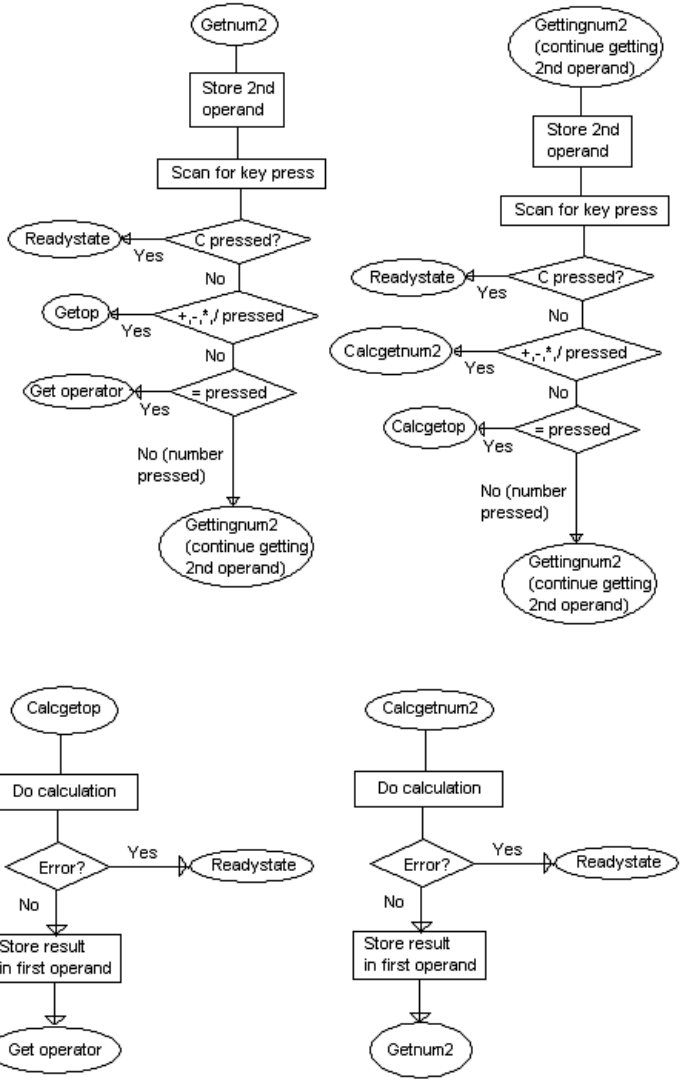
The operation of the calculator beings at the “Start” state and after some initializing, goes to the “readystate”. In the ready state, it scans for key presses. As soon as a character is received, it determines which one was pressed as shown in the decision diamonds and jumps to the next appropriate state. For example, if a number between 0 and 9 was entered, it will go to the “Getnum1” state. Find the corresponding “Getnum1” state bubble in the top left flow chart and follow the arrows to see what happens next. In this way you can examine each of the flow charts and see how the program flows. If you wish to understand a section in more detail, please refer to the Calc.asm file and find the label for the state that you are currently in and step through the assembly instructions.


Start
Readystate
Scan for key press
C pressed?
Yes
No
Get operator
+,-,*,/ pressed
Yes
No
= pressed
Yes
No (number pressed)
Getnum1


Getnum1
Store first operand
Scan for key press
Readystate
C pressed?
Yes
No
Getnum1op
+,-,*,/ pressed
Yes
No
Get operator
= pressed
Yes
No (number pressed)


Get operator
Scan for key press
Readystate
C pressed?
Yes
No
Gotop (Got operator)
+,-,*,/ pressed
Yes
No
= pressed
Yes
No (number pressed)
Getnum1


Getnum1op
Store operator entered
Getnum2
Gotop (Got operator)
Store operator entered
Getnum2


Getnum2
Store 2nd operand
Scan for key press
Readystate
C pressed?
Yes
No
Getop
+,-,*,/ pressed
Yes
No
Get operator
= pressed
Yes
No (number pressed)
Gettingnum2 (continue getting 2nd operand)
Gettingnum2 (continue getting 2nd operand)
Store 2nd operand
Scan for key press
Readystate
C pressed?
Yes
No
Calcgetnum2
+,-,*,/ pressed
Yes
No
Calcgetop
= pressed
Yes
No (number pressed)
Gettingnum2 (continue getting 2nd operand)
Calcgetop
Do calculation
Error?
Yes
Readystate
No
Store result in first operand
Get operator
Calcgetnum2
Do calculation
Error?
Yes
Readystate
No
Store result in first operand
Getnum2

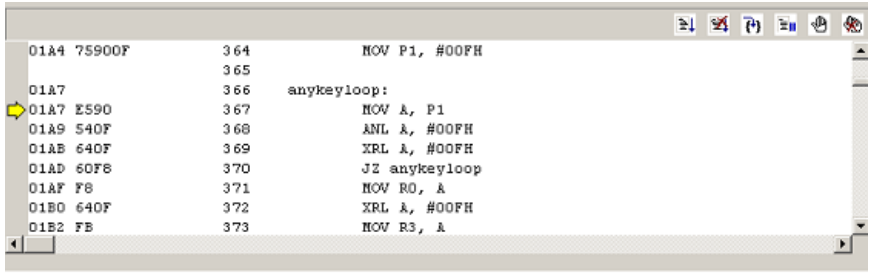
2.3.3 Using the MCU Interface

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.

2.3.3.1 MCU Assembly Source Window

The MCU Assembly Source Window 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 edit dialog will automatically scroll to the place in the assembly code where the simulation has paused and indicate the current instruction with an arrow.

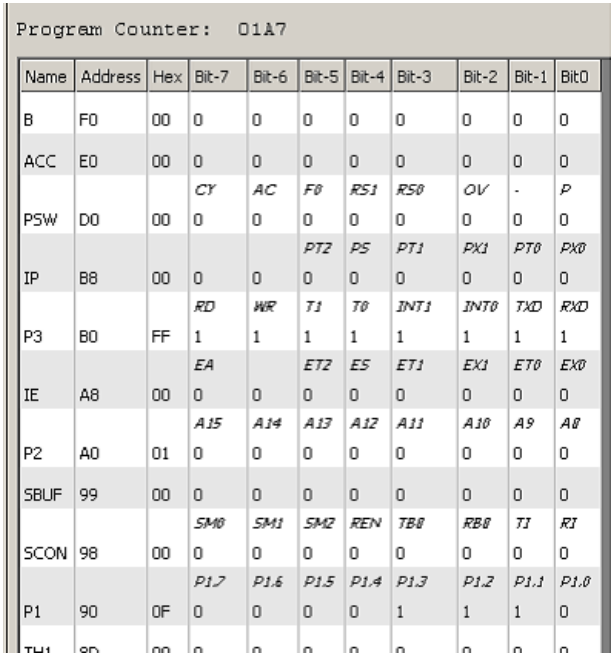
For this example, you will probably break in the key scanning routine as shown below:



The arrow indicates the instruction that the program has paused at. The numbers on the left are the opcodes for each assembly instruction. The second column shows the line number, line 367 in figure above. On the right side, the assembly instructions are displayed. The program is about to read the data on port P1 and move it into the accumulator (A) to be analyzed. It will determine whether any of the inputs have changed from the idle state.

2.3.3.2 MCU Register View

When the program is paused on a breakpoint, the Special Function Registers (SFRs) are displayed in the MCU Register View window as shown below:



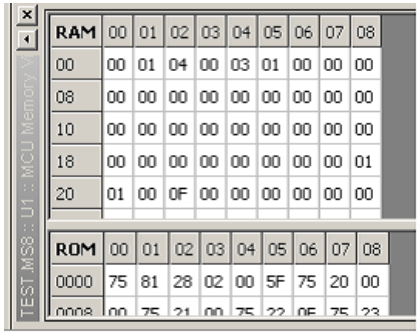
Program Counter: 01A7

Name	Address	Hex	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
B	F0	00	0	0	0	0	0	0	0	0
ACC	E0	00	0	0	0	0	0	0	0	0
			CY	*AC*	*F0*	*RS1*	*RS0*	*OV*	-	*P*
PSW	D0	00	0	0	0	0	0	0	0	0
					PT2	*PS*	*PT1*	*PX1*	*PT0*	*PX0*
IP	B8	00	0	0	0	0	0	0	0	0
			RD	*WR*	*T1*	*T0*	*INT1*	*INT0*	*TXD*	*RXD*
P3	B0	FF	1	1	1	1	1	1	1	1
			EA		*ET2*	*ES*	*ET1*	*EX1*	*ET0*	*EX0*
IE	A8	00	0	0	0	0	0	0	0	0
			A15	*A14*	*A13*	*A12*	*A11*	*A10*	*A9*	*A8*
P2	A0	01	0	0	0	0	0	0	0	0
SBUF	99	00	0	0	0	0	0	0	0	0
			SM0	*SM1*	*SM2*	*REN*	*TB8*	*RB8*	*TI*	*RI*
SCON	98	00	0	0	0	0	0	0	0	0
			P1.7	*P1.6*	*P1.5*	*P1.4*	*P1.3*	*P1.2*	*P1.1*	*P1.0*
P1	90	0F	0	0	0	0	1	1	1	0

The first column contains the name of the Special Function Register (SFR) and the second column displays the address of the corresponding SFR. The third column shows the value contained in the SFR. For example, the accumulator (ACC), contains a value of 00H. The instruction MOV A, P1, places the current value in P1 to register ACC. Some SFRs such as the program status word (PSW) contain bits that are used individually for different purposes. This can be seen in the remaining columns that display each bit individually as well.

2.3.3.3 MCU Memory View

The MCU Memory View displays the values in the internal RAM and ROM of the 8051 MCU.



RAM	00	01	02	03	04	05	06	07	08
00	00	01	04	00	03	01	00	00	00
08	00	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00
18	00	00	00	00	00	00	00	00	01
20	01	00	0F	00	00	00	00	00	00

ROM	00	01	02	03	04	05	06	07	08
0000	75	81	28	02	00	5F	75	20	00
0009	00	75	21	00	75	22	0E	75	23

The ROM window shows the program memory code that is loaded in hexadecimal format. This is the actual format of the code that the simulation is using when it is activated.

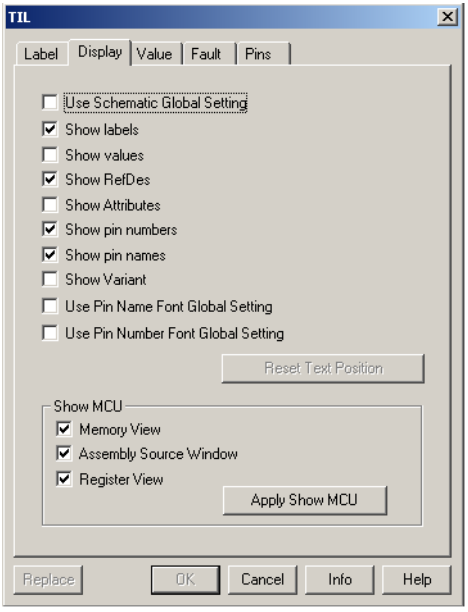
The internal RAM display, above the ROM, shows the data that is inside the 8051's memory and is modified as the program runs. Addresses 00H to 19H contain the four register banks. By default, the 8051 uses the first register bank (00H to 07H) for registers R0, and R1 to R7, but this can be configured by the program. The calculator example uses register bank '0' only. In this example, the program is paused and the following is displayed: R0 = 0h, R1 = 01H, R2 = 04H, R3 = 00H, R4 = 03H, R5 = 01H, R6 = 00H and R7 = 00H.

The memory space in the internal RAM, starting from 20H onwards, is general user space that is also used by the stack.

2.3.3.4 Displaying Elements of the MCU Interface

➢ To show or hide elements of the MCU Interface:

1. Double-click on the 8051 MCU to display its properties dialog and click on the **Display** tab.



2. Enable the **Memory View**, **Assembly Source Window** and **Register View** checkboxes as desired and click on the **Apply Show MCU** button.

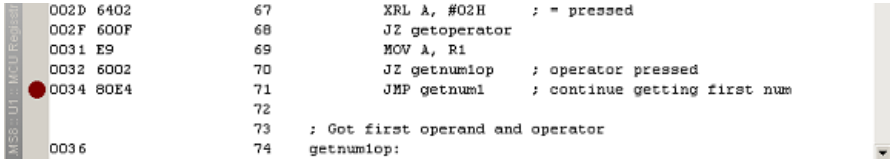
2.3.4 Advanced Features

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.

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.

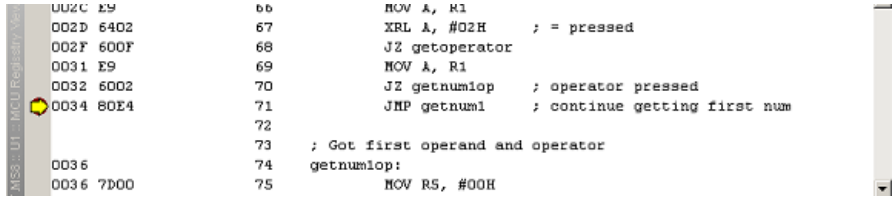
2.3.4.1 Adding a Breakpoint

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 .

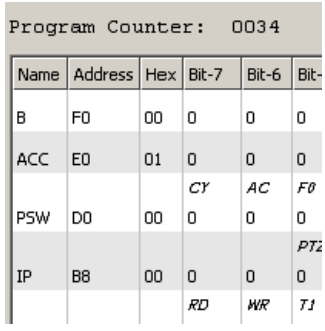


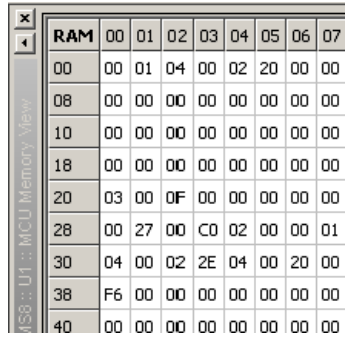
This will cause the program execution to stop when it reaches the JMP getnum1 instruction.

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.



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 value in the accumulator (ACC) in the CPU Data dialog now 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  .

2.3.4.2 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 MCUAssembly Source Window.
2. Another way of pausing the simulation is to click on the **Break Execution** button  . The MCU Assembly Source Window 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" be displayed on the HEX displays.

2.3.4.3 Break and Step Into

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 Window 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 step into the ADD16 subroutine by clicking on the **Step Into** 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 Into** 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.