CS/CoE 0447 LED Simulator Details

February 11, 2015
Version 0.2

This document describes selected aspects of the LED display simulator for Mars.

Keypad and LED Display Simulator

The LED simulator requires a special version of Mars available from your TA. This version of Mars includes an LED display developed by a graduate student in the CS Department.

To run the display simulator, click on the Tools menu tab in Mars. Select the menu option "Keypad and LED Display Simulator". When the display opens, select Connect to MIPS. This action establishes communication between the MIPS simulator and the LED display simulator. Load your program and run it. The display should change.

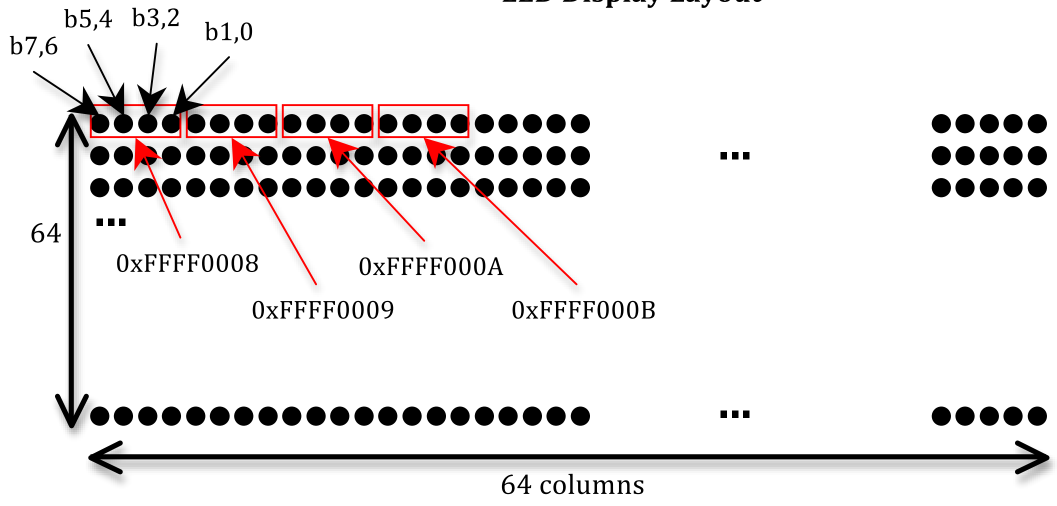
To end the LED Display Simulator, Disconnect and Close it. You can clear the LED display with the Reset button. You may need to Reset and Connect whenever you load or assemble a program. If you have problems (bugs!) with the LED Display Simulator, let your TA know.

Using the LED Display

The simulator models a display with 64 rows and 64 columns of LEDs. Each LED is associated with *two bits* in memory. An LED is turned off (black) when its value is 00. The LED is red when its value is 01, orange when its value is 10, and green when its value is 11. There are 4,096 LEDs, so 8,192 bits (1,024 bytes) in memory are needed. The values stored in the address range [0xFFFF0008, 0xFFFF0408) are used for the display. The bytes in the display's address range are mapped to specific LEDs. The display rows are numbered 0 to 63 (top to bottom) and the columns are 0 to 63 (left to right). The upper left corner is coordinate (row 0, column 0) and the lower right corner is coordinate (63, 63).

The upper left corner of the display is mapped to the lowest address and the lower right corner is mapped to the highest address. Bits 7 and 6 in the byte at 0xFFFF0008 control the LED at (0,0). Bits 5 and 4 in the byte at 0xFFFF0008 control the LED at (0,1), bits 3 and 2 control the LED at (0,2), and bits 1 and 0 control the LED at (0,3). The byte at 0xFFFF0009 controls the next four LEDs, and so forth.

LED Display Layout



LED display colors: 00 = off, 01 = red, 10 = orange/yellow, 11 = green
LED display memory starts at 0xFFFF0008
Each byte represents four LEDs (2 bits per LED)

The diagram above shows the layout of the LED display. Address 0xFFFF0008 is the start of the display; the byte value stored at this address corresponds to the 4 leftmost LEDs in the first row of the display (i.e., the upper left corner at positions (0,0), (0,1), (0,2) and (0,3)). The next four LEDs correspond to address 0xFFFF0009, followed by 0xFFFF000A and 0xFFFF000B. Bit 7 and 6 at address 0xFFFF00008 represent the "state" of the LED at position (0,0). For example, to set the LED to red, these bits would be 01. Bits 5 and 4 at address 0xFFFF0008 are position (0,1) and so forth. Bits 7 and 6 at address 0xFFFF0009 are position (0,4).

It may be convenient (more efficient!) to operate on a whole word (or halfword) of LEDs all at once, say using load-word (lw) and store-word instructions. In this case, you need to be careful about byte ordering, i.e., endianness. The x86 processor is little endian. Therefore, the byte for the first four LEDs in the diagram above are located in the least significant byte of the word at address 0xFFFF0008. For instance, suppose you want the first four LEDs to be red, the next four to be off, the next four to be orange and the final four to be green. The state (color) of these 16 LEDs uses 32 bits, that is, one word at address 0xFFFF0008. Storing the word value 0xFFAA0055 will set the colors of the LEDs to red, off, orange and green.

If you want to set only certain LEDs, you should first load the byte (halfword or word) of the LED you want to set. You should set only the bits in the loaded value corresponding to the LED. Store the updated value to the address for the LED. For example, suppose the LED at position (0,3) is green and all other LEDs are off. Thus, the byte value at address 0xFFFF0008 is 0x03. Now, suppose we want to set the first LED (upper left corner) to red without affecting any other LED's state. We first load the byte at address 0xFFFF0008 and set bit 7 and 6 in the loaded value to 01. This value is then stored to address 0xFFFF0008. In this example, we would load 0x03 (current value), and set bit 7 and 6 to 01. The new value is 0x43, which is stored to address 0xFFFF0008.

Using the Keypad

The LED Display simulator also has a keypad. To get input from the keypad, two memory locations are loaded. One memory location indicates *when* a key (arrow) has been pressed. The other memory location indicates *which* arrow key was pressed.

To get a value from the keypad takes two steps. In the first step, the keypad status is loaded from memory byte address 0xFFFF0000. If the byte loaded is 0, then an arrow key was not pressed. If the byte is 1, then an arrow key was pressed. When the loaded byte is 1, the second step is done.

In the second step, a byte is loaded from memory byte address 0xFFFF0004 to get which arrow was pressed. The value read determines which arrow key was pressed. The values are:

Key	**Value**
Up arrow	0xE0
Down arrow	0xE1
Left arrow	0xE2
Right arrow	0xE3
Middle key (b)	0x42

Your program should use a “polling loop” to check for a key press. A “polling loop” repeatedly loads the byte at memory address 0xFFFF0000 until the loaded value is 1. Once a key is pressed (i.e., the loaded word is 0x1), your program should determine which key was pressed and react as appropriate to the keypress. After handling the keypress, go back to polling for input.

Suppose the user presses the Left arrow. The byte at memory address 0xFFFF0000 is set to 0x1 by the simulator. Your program loads the byte at this address and finds the byte is 1. Next, your program loads the byte at address 0xFFFF0004. Because the left arrow was pressed, the byte will be 0xE2. After the value at 0xFFFF0004 is read, the simulator resets the byte at 0xFFFF0000 to 0.

Example Program to Draw a Line

This program draws a vertical line in the leftmost column of the LED display.

```
      li    $t0,0x40          # RED in leftmost LED
      la    $t1,0xFFFF0008    # base address of LED display
      sb    $t0,0($t1)
      sb    $t0,16($t1)
      sb    $t0,32($t1)
      sb    $t0,48($t1)
      sb    $t0,64($t1)
      sb    $t0,80($t1)
      sb    $t0,96($t1)
      sb    $t0,112($t1)
      li    $v0,10
      syscall
```

Example Program for Polling the Keypad

This program increments (right arrow pressed) or decrements (left arrow pressed) the current color of a "box" on the LED display. The box is represented by a group of 6x6 LEDs in the center of the display. The full program is available from your TA.

```
         li   $s1,1              # $s1 holds current box color
         move $a0,$s1            # draw initial box
         jal  _drawBox
poll:    la   $v0,0xffff0000     # address for reading key press status
         lw   $t0,0($v0)         # read the key press status
         andi $t0,$t0,1
         beq  $t0,$0,poll        # no key pressed
         lw   $t0,4($v0)         # read key value
lkey:    addi $v0,$t0,-226       # check for left key press
         bne  $v0,$0,rkey        # wasn't left key, so try right key
         addi $s1,$s1,-1         # decrement current color
         andi $s1,$s1,3          # mask high bits after decrement
         move $a0,$s1            # argument for call
         jal  _drawBox           # redraw box in new color
         j    poll
rkey:    addi $v0,$t0,-227       # check for right key press
         bne  $v0,$0,bkey        # wasn't right key, so check for center
         addi $s1,$s1,1          # increment current color
         andi $s1,$s1,3          # mask high bits after increment
         move $a0,$s1            # argument for call
         jal  _drawBox           # redraw box in new color
         j    poll
bkey:    addi $v0,$t0,-66        # check for center key press
         bne  $v0,$0,poll        # invalid key, ignore it
         li   $v0,10             # terminate program
         syscall
```