CE 12 Lab Report 6

Ready, Set, Go!

Brandon Gomez

Lab Section: 2

12.10.14

Procedure:
The purpose of this lab is to understand interrupts and memory mapped I/O registers. In this lab, we will program a timer that is controlled by SW1 and SW2. SW1 will pause the timer while SW2 will reset the timer.

Materials:
Hardware:
- PICkit3
- Chipkit Uno32
- Chipkit Basic IO Shield
- Two USB cables (one for PICkit3 and one for Uno32)

Software:
- MPLAB (to code)

Instructions:
- add: $d = $s + $t
- addiu: $t = $s + SE(i)
- addi: $t = $s & ZE(i)
- and: $d = $s & $t
- sw: MEM [$s + i]:4 = $t
- lw: $t = MEM [$s + i]:4
- la: ui $rd, LabelAddr[31:16]; ori $rd,$rd, LabelAddr[15:0]
- jal: $31 = pc; pc += i << 2
- nop: Do nothing

T1_ISR:
The T1_ISR is were the counting of the timer will be held. Time will be equal to the equation we got in lab which is *Time = PR1*(1/frequency*256).* In order to do this I first load PR1, T1CON, and frequency into registers. I then use mult and div instructions in order to get the desired results. Once the operations are held milliseconds, the timer, is then incremented.

```
lw $t0, PR1
lw $t1, T1CON
/*PR1 * 256 */
mult $t0, $t1
/* getting the least sign */
mflo $t2
lw $t3, frequency
/* PR1 * 256 / frequency */
div $t2, $t3
/* get lower bits */
mflo $t4
/* add to milliseconds */
lw $t0, milliseconds
add $t0, $t0, $t4
sw $t0, milliseconds

/* clear TMR1 value clear the count if it was used */
la $t0, TMR1
li $t1, 0xFFFF
nop
sw $t1, 4($t0)
/* clear T1IF clear the interrrupt priority */
la $t0, IFS0
li $t1, 0x0010
nop
sw $t1, 4($t0)
```

T1Setup:

The T1Setup should enable the actual interrupt but before we do that, we must clear and set some things. First we have to set TRISD, then we have to clear T1CON, then we set the T1CKPS, then we clear theTMR1 value, then we set the PR1 value which I set it to 781250. Next, we have to set the IPC1 to 4, then we clear the T1IF and last we enable the actual interrupt to increment the timer.

```
T1Setup:
    la $t0, TRISD
    li $t1, 0x0C00
    nop
    sw $t1, 8($t0)
    /* clear T1CON put in known state */
    la $t0, T1CON
    li $t1, 0xFFFF
    nop
    sw $t1, 4($t0)
    /* set T1CKPS set the prescalar */
    la $t0, T1CON
    li $t1, 0x0030
    nop
    sw $t1, 8($t0)
    /* clear TMR1 value clear the count if it was used */
    la $t0, TMR1
    li $t1, 0xFFFF
    nop
    sw $t1, 4($t0)
    /* set PR1 value set the period you want */
    la $t0, PR1
    li $t1, 781250
    nop
    sw $t1, 8($t0)
    /* set T1IP set the interrupt priority */
    la $t0, IPC1
    li $t1, 4
    nop
    sw $t1, 8($t0)
    /* clear T1IF clear the interrrupt priority */
    la $t0, IFS0
    li $t1, 0x0010
    nop
    sw $t1, 4($t0)
```

```
/* Enable T1IE enable the interrupts */
la $t0, IEC0
li $t1, 0x10  /* 4th bit */
nop
sw $t1, 8($t0)

jr $ra
nop
```

T1Start/Stop:

The T1 Start should start the timer. We do this by setting the T1CON and turn on the 15th bit.

```
T1Start:
    /* s1 starts the timer */
    /* s2 clears the timer */
    /* s2 has priority */

    /* Set T1 ON actually turn on the timer */
    la $t0, T1CON
    li $t1, 0x8000 /* 15th bit */
    nop
    sw $t1, 8($t0)

    jr $ra
    nop
```

The T1 Stop should stop the timer. We do this by clearing the T1CON and turn off the 15th bit.

```
T1Stop:
     /* Set T1 ON actually turn on the timer */
    la $t0, T1CON
    li $t1, 0x8000 /* 15th bit */
    nop
    sw $t1, 4($t0)

    jr $ra
    nop
```

Results:

Once we have all the subroutines done, we are now able to code in C. We first have to initialize the OLED display, and then turn it on. We are then able to call the first subroutine, T1Setup(). We then have an array char buffer to hold the timer string. In the while loop, we check to see the values of the switches to determine what subroutine to do. If SW2 is on, we have to stop the timer, set the timer again, and set the time back to zero. If SW1 is on we start the timer, else we pause the timer.

```
OledInit();
OledDisplayOn();
T1Setup();
char timeString[6];
*timeString = "00:00";
OledPutString(timeString);
extern volatile int milliseconds;
// Either write your C program here or call your (new) assembler function
while(1) {
    // Display the least significant part of the time for debugging
    //PORTE = 0x00FF & milliseconds;
    // if SW2 is on
    if (PORTD == 0x210 || PORTD == 0x310){
        T1Stop();
        T1Setup();
        milliseconds = 0;
    }
    // if SW1 is on and nothing else
    else if (PORTD == 0x110){
        T1Start();
    }
    // if all off
    else if (PORTD == 0x010){
        T1Stop();
    }
    // reset position
    OledSetCursor(0,0);
    // display
    if(milliseconds%2 == 0){
        sprintf(timeString, "%d:%02d", milliseconds/60, milliseconds%60);
    }
    else{
        sprintf(timeString, "%d %02d", milliseconds/60, milliseconds%60);
    }

    OledPutString(timeString);

    OledClear(); // clears led to insert new time
```

Conclusion:

Overall I found this lab to be very satisfying getting the right I/O registers from the map and understanding the interrupts. I designed my program by first doing the functions that were easy like ISR. Then I just continued building from the bottom all the way to the top (Bottom up programming). My timer is just a few tenths off from an actual timer. Some bugs that I encountered were that the display would not reset the cursor, so the display will look like it was flickering. To make sure the program didn't flicker I put a OledSetCursor() function before the print.