EE333 Microcontroller Engineering
Oregon Tech Portland, Fall 2013

Lab Assignment #2 - *Temperature Readout*
Due October 31

Objectives:

Implement an algorithm as the processing step of a microcontroller system. Develop a program that actually does something useful.

Equipment Required:

Everything you needed for Lab Assignment #1. File ee333template.asm and registers.inc, or ee333template.c (C language)

Background Information:

All microcontroller programs consist of inputs (usually sensors of some type), processing of the input data, and outputs (usually controllers of some type). In this lab assignment you will continuously read the temperature and show it on the 4-digit, 7-segment LED display on the Dragon12. Because we haven't yet covered I/O devices nor other needed features (particularly interrupts) a starting point program has been provided, ee333template.asm. This program runs the analog to digital converter (the temperature sensor has an analog voltage output) and the LED display. Load and run the program as-is you will see "HELP" displayed on the 7-segment LED display and a pattern of lights on the row of LEDs below the 7-segment display.

You can also run the program on the simulator if you start the simulator in "Dragon12 & minidbug12" mode. When you do this you will need to press "Go" twice to start the program. Look under View-->Parallel Ports to see the the simulation of the Dragon12 LED displays and push buttons. Look under View-->Analog Inputs to be able to set the analog input voltage. Remember you can use the simulator for development, but always finish by running the program on the Dragon12.

The analog to digital converter has 8 inputs and puts the value representing the voltage on the input into 8 separate 16-bit word variables. The values are refreshed roughly every 64 milliseconds. These values all range from 0 (for 0 volts in) to 1023 (for 5 volts in). The particular variables of interest are:

- ADR04H — the light sensor (Q1)
- ADR05H — the temperature sensor (U14)
- ADR07H — the trimmer pot VR2. This allows adjusting the input voltage over range with a small screwdriver

These variables are part of the analog to digital interface and are declared in registers.inc. There are variables in the program that will be needed as well:

- displ — A byte variable where each bit will represent an LED in the row of LEDs, '1' turns the LED on and '0' turns the LED off.
- disptn — Four bytes where each byte represents a digit in the 7-segment display. Values 0 through 9 will display '0' through '9' but there are 22 other values that drive the display meaningfully. These are listed in the comments in the segm_ptrn table in ee333template.asm.

Look at the ee333template.asm file. It's a complete program that you will add to during the term. So that you don't accidentally "mess up" the original program, sections have been marked of with triple asterisks - *** - that are regions where you are to add your own variables and code. Code must not jump between these regions. These sections are:

- *Add your variables here* — put any variables for your program here. Use "ds 1" to declare a byte variable and "ds 2" to declare a word variable.
- *Any code you need to initialize should go here* — initialize your variables, if needed, here, using movb and movw instructions. When you start initializing I/O devices, that code will go here as well. Note that this code will only be executed once, when the program is started.
- *Code you need to execute repeatedly should go here* — code in this area is in a loop that gets executed repeatedly, forever, after initialization completes. Execution is limited to no more than once per millisecond, but depending on what you do there could take much longer.
- *Any code you place here will execute every 1.024 milliseconds* — Execution here is triggered by a timer every 1.024 milliseconds. Don't put anything here that would take lots of time.
- *Add any tables or other constant values here* — tables and constants go here, declared with db (for bytes) or dw (for words).

You should be able to understand all of the existing code by the end of the term.

Assignment Part 0:

Using "C" is optional for this course, and only if you are already familiar with the language. There is a document provided in the distribution on using the C compiler and the compiler and IDE are also provided.

For this assignment you should only need the area "Code you need to execute repeatedly should go here". You don't need to add any variables, but if you feel you need to then add them, go ahead.

Make a copy of the template file in the folder you are using for this assignment, renaming it "lab1.asm" or something similar, and copy the registers.inc file there as well. Assemble it and verify that it works ("HELP" on the 7 segment display and some of the LED row lit).

Assignment Part 1:

Let's get some quick feedback that we can actually write code that does something! Drive the row of LEDs with the value of the light intensity on the Dragon12 board. If you cover the sensor the intensity drops. If you shine a flashlight at the sensor, the intensity increases.

Load accumulator D with the value of the light sensor output. This value will be in the range of 0 to 1023. The row of LEDs can only display (in binary) values 0 to 255. So divide accumulator D by 4. Do this easily with two consecutive *lsrd* instructions. Then store the least significant byte of accumulator D (namely accumulator B) into variable *displ*.

Run the program to verify that it works. Note that the display will flicker and it may be difficult to read the actual value. We will fix that later.

Assignment Part 2:

Continue with the program you developed for part 1, you will now add a temperature readout to the 7-segment display.

The temperature code is readable as a word from ADR05H. This is a scaled value. Since 5 volts will give a code of 1023, The units are 1023/5 volts^{-1}. The sensor output is 0.01 volts/˚C. Combining these, we get that the temperature code is in units 10.23/5 ˚C^{-1}. The temperature needs to be converted to digits for display. This means we need to multiply by 500 and then divide by 1023 (we can't multiply by 5 and divide by 1.023 since values must be integers). The display digits need to be stored as bytes in locations disptn, disptn+1, disptn+2, and disptn+3, going from left to right. Since you only need two digits (why is this??), you could put a space (no light) in front of the number (use the value 32 for this) a the character "C" on the right of the number to indicate Fahrenheit. You can use the convert to digits algorithm in the text or come up with one of your own.

You will find that the display will flicker as the temperature changes between degrees. Don't worry about that now. It will be fixed in lab assignment 3. Also, the value will be somewhat higher than the actual room temperature because of self-heating of the sensor and the Dragon12 board.

To turn in:

- Documented program listing.
- Description of how you tested the program.
- Discussion of any problems you had

This should all be placed in a single file (PDF format preferred, Word or Open Office formats also acceptable).