CmpE213 - Digital Systems Design

Due: Tuesday Nov. 4, 2003

Show your work for full credit.

Homework 6

ASM and C for the 8051

1. Write an ASM program to a) add two unsigned char variables; x and y, b) add two unsigned hit variables, v and w, and c) write results to P1. This ASM program should be complete with segments, variables, labels, etc. In particular, give your program the following characteristics:
    - Declare x as a 1-byte variable in a relocatable segment of directly-addressable internal memory
    - Declare y as a 1-byte variable in an absolute segment of external memory beginning at 0000H.
    - Declare v as a 2-byte variable in an absolute segment of internal memory beginning at 70H.
    - Declare w as a 2-byte variable in a relocatable segment of internal indirectly-addressable memory (Hint: by defining w in indirectly addressable memory, you must access it using @R0 or @R1).
    - Write your instructions in an absolute code segment beginning at 0000H. In this code segment, perform the following operations:
        (a) Add x and y, putting the result in x.
        (b) Add v and w, putting the result in v.
        (c) Write x out to port 1.

    Write and assemble your code in uVision and debug (simulate) in uVision2 (i.e. step through your code, put values into x, y, v, and w within the debugger, make sure that the values are what you expect. *Try* values you think might cause your program to "break". Check that output to P1 is correct. Note that getting your code to compile doesn't mean your code works! In a real project you are *maybe* 30% of the way done when you get your code to compile. The next 70% of time is spent debugging your code and getting it to work right).
    Turn in a) a print-out of your code, b) a screen-dump of the debugger after your program completes. Windows that should be displayed include: the module window, memory window showing internal direct memory, and the register window. **Please put your name at the top of your program (and anyone who may have helped you) and comment your code appropriately.**

2. Microcontrollers are used to read values from many different external devices. Sometimes, the values those devices give the microcontroller are not easily understood. For example, consider a thermometer that returns a 2-bit number to the 8051. Most thermometers are not linear. A binary 0 may mean the thermometer reads 32 degrees F, a binary 1 mean 65 degrees, a binary 2 mean 80 degrees, and a binary 3 mean 105 degrees. The following C code reads a number from this thermometer (that is connected to Pl), finds how many degrees this number represents from a lookup table, and then writes the number of degrees it is reading out to P3. P3 might be hooked to something like a seven-segment display to show a user the temperature reading. A do nothing loop is inserted between each reading, so that the value on the seven segment display does not change too quickly.

```
unsigned char lookup []   = {32, 65, 80, 105};
unsigned char i;
void main(void){
   while(1){
       P3 = lookup[P1];  //Read value from P1 and write to seven seg.
                         // Assume P1 is between 0 and 3.
       for (i=0; i<0x42; i++){;} // Do nothing - create a delay
   }
}
```

Re-write this program in ASM, properly declaring segments and the variables "lookup" and "i." Simulate your code in uVision2. Verify to yourself that the correct value is being written to P3 for a given value on P1 (i.e. if P1=3, the value 105 is written to P3, etc). Turn in a printout of your code.

3. Write a complete ASM program that performs the same function as the following C code. Assume the array, x, is in EXTERNAL DATA space, while all other variables are in internal data space.

```
unsigned int xdata x[10]; // a 2-byte unsgn int in external data space
unsigned char i,j;
main(){
      // read tank levels
      for (i=0;i<10; i++){
            P1 = i; // select the tank to read
            j = 0;  // start a loop and do nothing (wait)
            do{
                  j++;
            }while(j<0x2a);
            x[i]  = P3; // read the tank level and store for later use
      }
      while(1){;}
}
```

Compile your code and simulate it to be certain it works (i.e. step through your code and do things like change the value of P1, make sure the correct value of P3 gets written to x, check that the loop really executed 0x2a iterations, etc.). Send your ASM code (only) to tsuiter@umr.edu as an attachment to an email (with an appropriate subject line). Turn in a print-out of your code.

For those of you who might be interested in a purpose to this code, we might imagine that we have a 10-channel A/D (analog-to-digital) converter connected to port 1. This A/D converter is reading analog voltages from some sensor - say a set of 10 level sensors, which return an analog voltage proportional to the amount of water in 10 different tanks. The channel (tank from which a reading is taken) is selected using P1. If P1 is set to a binary 2, we're reading from tank 2 and so on. The A/D value is read from P3. The loop simply a) selects a channel to read from, b) waits a while for the A/D converter to produce a digital reading of the analog voltage, and c) writes that digital reading to the array x. This code represents just part of a larger program - a “real” program would have gone on to do something with these tank levels.