CmpE213 – Digital Systems Design

Homework 6
Machine instructions for the 8051

1. Enter and compile the following program in uVision2 and then simulate and debug it.

```
   mystuff segment data
   rseg mystuff
     blah: DS 1
   mycode segement code
   rseg mycode
1          MOV 80H, #0FFH      ; initialize P0 to 0FFH
2          MOV A,#5
3          MOV R0,#20H
4  LOOP:   JNB P0.2,NEXT
5          JMP LOOP
6  NEXT:   MOV @R0, #A7H
7          CLR 07H
8  TIME:   DEC A
9          JNZ TIME
10         MOV mystuff,#42H
11         NOP
12         NOP
13         END
```

   (a) set the watch window to watch the value in the accumulator, in R0, and in variable mystuff.

   (b) Single step through lines 1-5. Will the entire code segment be executed with the current values in p0? Why/why not? Alter the value of p0 to enable the program to continue execution. b) Single step through lines 6 and 7. Display and record the contents of data memory location 20h, using the memory window.

   (c) Find the opcode for instructions 1 and 2 by displaying the contents of code memory starting at c:0000 or by displaying the disassembly window. Note that the first instruction begins at c:0000 in code memory. Record the appropriate values for the opcode.

   (d) What are the memory addresses of LOOP, NEXT, TIME, and mystuff?

   (e) Find the number of cycles and seconds it takes to execute the loop at instructions 8 and 9 in the following manner: Single step down to line 8 (you should already be there). Set a breakpoint on line 10. Record the cycles and seconds found on the register display. Execute (i.e. GO!) the loop at lines 8 and 9 until the breakpoint is reached. The time it took to execute the loop is the difference between the recorded cycles/times and the values now shown in the register display. Record this difference.

2. Write an assembly language program (complete with segment and variable declarations) to perform the following tasks (in the order given). Be sure to set aside memory for the stack and to initialize SP.

   (a) Load external memory location 5280H with the value 40H.

   (b) Set the value of R4 to 42H in register bank 3 and set the register R6 to 2AH in register bank 1. Use the PSW and register addressing (instead of direct addressing). Verify that the correct values have been set by observing either the register window or the memory window in uVision2

   (c) Set the value of R3 to 13D in register bank 4 and set the register R0 to 01110101B in register bank 0 using direct addressing.

   (d) Use the PUSH and POP operations to place several bytes onto an internal stack. Record the values in data memory and the locations effected, noting the final value of the stack pointer. What problems occur if you do not initialize the stack pointer? How might you avoid these problems?

   (e) Place 42H into the accumulator if bit 2 of P1 is set (equals 1). Otherwise, place the value at external memory location 5280H into the accumulator. Perform this operation twice, once using byte-based instructions and once using bit-based instructions. When debugging this instruction sequence, be sure to test both the case where P1.2 is set and where P1.2 is cleared.

   Simulate in uVision2 to prove that your program works. Turn in a) a printout of your code and b) a screen dump of uVision2 after your program completes. The screen dump should be obtained in the following manner. Using the 'd' command, display the appropriate data memory locations to show the results of the operations performed in steps a-c. Make sure that the register contents are also displayed. Hit the shift key and "print screen" key at the same time. Open a word processor such as Microsoft Word and create a new document. Hit edit and paste to place the screen dump into the document. Now print the document.

3. 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 which 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 (which is connected to P1), 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 an 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 is not changing 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. disp.
                       // 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.