CmpE213 – Digital Systems Design

Learning Objectives
ASM and C for the 8051

At the end of the following sections, you should be able to:

Chapter 3,7:

1. Given an ASM instruction, a) find the opcode, b) show the changes that take place in memory during execution, c) identify the basic operations that take place (e.g. Add A,#42H : (A) <– (A) + 42H). Use an opcode map to find the opcode of an instruction, the number of bytes to store the instruction, and the number of cycles to execute the instruction.

2. List, define, and give an example of the addressing modes used by the 8051. Use addressing modes interchangeably in an ASM program to read or write locations with similar results (do for both instructions like MOV and JMP). Given an 8051 instruction, identify the addressing mode(s) used by that instruction.

3. Given an ASM instruction, find it's opcode (e.g. Opcode for ADD A,#42H is 2442H). Given an opcode, find the equivalent instruction. Given contents of code memory, find the instructions that are contained there (i.e. "disassemble" code memory).

4. Given a short segment of code: a) calculate the number of bytes it takes to store the code b) calculate the number of instruction/machine/clock cycles or microseconds the code takes to execute c) show the changes that take place in memory (internal, external, SFRs) as the code executes d) explain, briefly, what the code does.

5. Given the address of an intended jump instruction and the desired destination, calculate the rel/absolute/long address to be used. Explain the limitations of rel/absolute addressing (where you can and cannot jump). Explain the advantage of using rel/absolute addressing over long addressing (saves code space).

6. Write a 3 or 4 instruction code segment for the 8051 to accomplish a simple task (e.g. set bit X in blah, read bit Y in blah, Move X to Y using addressing mode blah, etc). Analyze/debug the same code segment, showing changes in registers and memory if needed.

7. Explain the operation of a stack. Given a set of stack instructions (PUSH, POP, CALL, RET), show the changes that may occur in memory and to SP, PC. Given a "picture" of memory, showing memory contents and the value of SP, show what happens on a PUSH or a POP.

8. Write ASM programs to perform moderately simple tasks (20-30 instruction in length, plus directives), using properly defined segments and variables. Create, assemble, simulate, and debug ASM program using Keil uVision and dScope. Debug (analyze and correct) ASM programs by hand, charting changes that take place in memory.

9. Properly use absolute and relocatable segments. Define the location and contents of ASM segments. Define variables and labels within a segment. Use variables and labels to make code more readable.

10. Define and use arrays with ASM.

C-Programming for 8051:

1. Define the primary operation performed by each C instruction (for, while, do, if, switch, return, define, include). Write/analyze mathematical expressions in C, with proper application of operator precedence (including operators like (char), >>, >=, ++, etc).

2. Properly define variable types and explain the consequences of using one type over another (for example, of using a float over an unsigned char). Identify the number of bytes associated with each type.

3. Explain the operations of pointers and write/analyze/debug code using them (in C or ASM). Define and use 1- and 2-dimensional arrays. Access array locations using standard techniques as well as pointers. Explain limitations of using an array with a microcontroller (memory) and apply understanding of these limitations when writing code. Calculate the amount of memory used by an array.

4. Write a 3 or 4 instruction segment of C-code to perform a simple task for the 8051. Analyze/debug the same code, showing changes in registers and memory if needed.

5. Set/clear/examine bits in C or ASM using either bit or byte instructions, without corrupting other bits (e.g. Set bit 5 of mem location 42H, without changing any other bits. Clear the ACC if bit 4 of xdata mem location 5280 is clear). Explain why you cannot always use bit instructions.

6. Write/analyze/debug functions in C and ASM. Properly return results from more than 1 variable. Identify local and global variables and explain the scope of each. Explain and apply over-writable and static function variables. Explain why you must set aside space for a stack in ASM. Write a function that creates a XmS "delay" and explain how you would ensure the delay was truly XmS.

7. Read and write C-code which uses instructions/directives/types unique to C for the 8051. For example: define the type of memory where a variable will be stored (code, data, xdata, etc); define a variable as a bit, sfr, or sbit; access a specific memory location using sfr, sbit, _at_, and XBYTE[]; define the memory model, register usage, interrupt number, or re-entrant qualities of a function. Explain *where* a variable of type const, data, xdata, etc, is stored. Write/debug/analyze/explain programs using any of the above.

8. Write C programs to perform moderately simple tasks (20-30 instructions in length, plus directives), using properly defined variables, functions, etc. Create, compile, simulate, and debug C programs using Keil uVision and dScope. Debug (analyze and correct) C programs for the 8051 by hand, charting changes that take place in memory.

9. Translate between segments of C and ASM code (i.e. given the following C-code, create ASM code which does the same, and vice-versa). Given C-code and the assembly code generated by a compiler, identify which ASM instructions are associated with which C instructions. Identify primary advantages/disadvantages to writing a code-segment in C compared to ASM and vice-versa.

Applications (some Ch 10):

1. Write/debug/analyze/explain C and ASM code to control/use a: keypad, unipolar stepper motor, lookup table, external or internal A/D converter, and LCD module. Explain the basic mechanisms behind a keypad, unipolar stepper motor, and A/D converter (e.g. Explain how and *why* we are able to read the state of 16 external keypad pushbutton switches using only a single 8-bit I/O port. Explain *how* to make a stepper motor spin and *why* our method does, in fact, make it spin. Explain how to read an A/D converter. Explain what an A/D converter does. Explain how one might read a resistor value if they do not have an A/D converter). Given the state of a keypad and outputs to a keypad, identify what values would be read at the inputs.

2. Explain the advantages of using lookup tables and limited-precision arithmetic with microcontrollers. Write/debug/analyze C or ASM code which uses lookup tables.

3. Explain how and why PWM may be used to create an analog output. Write code to do so.