ECE 471 Lab Schedule

Lab1: Week #1: Practice using the software tools. Write a small 8051 assembly program. Wire wrap practice

Lab2: Week #2: Build the core of your system. Week #3: Get it working, write a short demo program.

Lab3: Week #4: Add peripheral hardware, install a debugger Week #5: Write a video display program (X–Y ’scope).

Lab4: Week #6: Add more peripheral hardware. Week #7: Write a sound sampling program.

Project: Week #8–10: Independent Project.

APPENDIX:
A. How to get better grade? B. Serial I/O C. Datasheets for DAC and ADC D. Data sheet for SRAM E. Data sheet for EPROM

LAB Grading:

Working demonstrations of your hardware and software are the basis for the grading of the ECE 471 labs. The demonstrations are typically due on Fridays, but may be done early. Late demonstrations will receive only partial credit, and will generally not be accepted three days past the established due date. Refer to the course syllabus for details. The lab reports are due on the next school day after the demonstration is made. Though the lab reports are worth only a small portion of the grade, they must be handed in to receive credit, since the grade for each lab is written on the report by your TA, then sent to the professor to be recorded. The first three/four labs are worth 250 points. The final project is worth 250 points. Refer to the grading summaries at the end of each lab for details.
Lab #1: Getting Started

This first lab assignment is designed to familiarize you with the 8051 assembly language, software tools used to write 8051 assembly programs and wire wrap techniques. You will first practice using the software by entering, assembling, and simulating a short program, provided below. You will then write a short program, and demonstrate it using the simulator. No hardware construction will be required to complete this first lab exercise.

Part 1: Getting Started with the 8051 (Optional)

The following code segment was designed to be used in a low–cost consumer electronic device, in which the user may edit parameters by using a set of pushbuttons and seven segment displays located on the front panel. When the power to the system is shut off, external circuitry and an interrupt routine (which you need not be concerned about) switches the processor into a power–down mode, where the internal ram is maintained by a battery. However, the battery may go dead, become loose, or be stolen from the unit for use in some other device. This routine is called immediately after any parameters are changed and when the power is restored. If the memory was corrupted, the routine sets the carry bit, to indicate that the parameters must be initialized to the ”factory default” values, thereby preventing the device from operating with random parameters.

There are 15 user parameters, whose values are stored in the internal memory locations 30h to 3Eh. A checksum byte is stored in location 3Fh. The routine first computes a new checksum. It is then compared to the old value, stored in location 3Fh. If they are equal, then the carry bit is cleared, otherwise it is set. Finally, the value in location 3Fh is replaced with the new checksum. This one routine can serve the dual purpose of verifying the user data’s integrity at start up as well as updating the checksum byte immediately after the data is modified by the user. When used at start up, the returned value in the carry bit is important, but when used to update the checksum in memory, the returned carry bit is not used.

Check:  
    PUSH ACC  ;save the accumulator  
    MOV A, R0  ;and save R0 also  
    PUSH ACC  
    CLR A  ;initialize ACC to zero  
    MOV R0, #30h  ;initialize R0 with 30h  
    Check2:  
        ADD A, @R0  ;add in each byte  
        INC RO  ;increment the pointer  
        CJNE R0, #3Fh, Check2  ;loop if we’re not done yet  
        MOV R0, A  ;store ACC in R0 for now  
        CLR C  ;clear Carry for subtraction  
        SUBB A, 3Fh  ;subtract old checksum  
        MOV 3Fh, R0  ;store new checksum  
        CLR C  
        JZ Check3  ;check was ok if ACC=0  
        SETB C  ;set carry if check failed  
    Check3: 
        POP ACC  ;restore R0 and ACC  
        MOV R0, A  ; Carry and 3Fh are the only  
        POP ACC  ; things left modified  
        RET  ;and we’re done!

TASK: Enter the code above using a text editor, such as PC Write or vi. You may use a word processor, but the file must be saved as standard ASCII text. Assemble the code. You may use any of the assemblers available. Finally, use the 8051 simulator provided to verify that the code works.
**Part 2: Writing a Short Routine (optional)**

The 8051 processor supports only 8 bit arithmetic, however by performing multiple 8 bit operations, larger quantities can be handled. Recently a design group has been developing software using unsigned 16 bit integers. They need you to write a routine to compute the product of two 16 bit (unsigned) numbers for use by their program.

The program will place two 16 bit quantities in memory locations 20h and 21h, as well as locations 22h and 23h with the lower memory holding the least significant byte respectively. Your routine should place the result, which will of course be a 32 bit value, in memory locations 24h through 27h, as indicated in the table below:

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>20h</td>
<td>Input value, Least Significant Byte</td>
</tr>
<tr>
<td>21h</td>
<td>Input value, Most Significant Byte</td>
</tr>
<tr>
<td>22h</td>
<td>Input value, Least Significant Byte</td>
</tr>
<tr>
<td>23h</td>
<td>Input value, Most Significant Byte</td>
</tr>
<tr>
<td>24h</td>
<td>Output value, Least Significant Byte</td>
</tr>
<tr>
<td>25h</td>
<td>Output value, Less Significant Byte</td>
</tr>
<tr>
<td>26h</td>
<td>Output value, More Significant Byte</td>
</tr>
<tr>
<td>27h</td>
<td>Output value, Most Significant Byte</td>
</tr>
</tbody>
</table>

Your code may use whatever registers and memory locations are needed. Remember, a full multiplication routine is not required, only the square of a single 16 bit value. Your routine does not need to be optimized for speed, but it is crucial to the operation of the final product that your routine returns the correct 32 bit square of the input value.

**Suggested Algorithm:** Consider the task of multiplying two ordinary numbers, such as 2382 and 5347. Simply plugging them into a calculator will yield a product of 12,736,554. If your calculator could only accept two digit numbers as inputs, then an algorithm analogous to the 16 bit square is encountered. The numbers must be considered two digits at a time:

\[
2382 \times 5347 = ((23 \times 100) + 82) \times ((53 \times 100) + 47) = ?
\]

By factoring, the following expression is obtained.

\[
(23 \times 53 \times 100 \times 100) + (23 \times 47 \times 100) + (82 \times 53 \times 100) + (82 \times 47) = ?
\]

Now the problem consists of four multiplications which can be performed, followed by formidable addition.

\[
(1219 \times 10000) + (1081 \times 100) + (4346 \times 100) + 3854 = 12,736,554
\]

Though other techniques exist for multiplying numbers, this algorithm is well suited for multiplying 16 bit unsigned integers on the 8051 processor, since it makes use of the 8 bit multiplication which returns a 16 bit value. It is recommended that you implement an algorithm similar to this one.

**TASK:** Again, enter your code with a text editor, assemble and simulate it. Create a short list of numbers (several of them should be larger than 255, please) and run the program with each of them to verify that it accurately computes the square of each number. Make sure the program works to your satisfaction, demonstrate it to your TA if you wish. You automatically get 10 pts for this lab.
Part 3: Wire wrap construction practice (Optional)

You will be using write wrap construction techniques to build most of your system. Wire wrap construction uses special chip sockets with long, wide pins. The sockets are inserted into a piece of perforated board. It is often helpful to glue the sockets to the board, preventing strain on the wire wrap connections, which would otherwise be responsible for holding the sockets in place. ID tags are available to label the pin numbers on the bottom of the board. It is highly recommended that you use ID tags, since you will be viewing the chips from the bottom side, thereby reversing the configuration of the pin number from the diagrams shown in data books. (If you make your own ID tags, check them carefully to avoid destroying $30 worth of chips to save $3 on ID tags!)

By reviewing the pin assignments for all of the chips to be installed on the board (including those for lab #3 and #4), a reasonable layout scheme should be apparent. Avoid leaving too much space between the sockets since a considerable amount of circuitry will be added in later labs. Good layout decisions made at this point will prevent a considerable amount of frustration in the near future. Because a significant number of chips will be added in lab #3, it is a good idea to install these sockets in the board before wrapping wires. The wires are added using a wire wrap tool. A typical wire wrap tool is shown below, with instructions. Wire wrap wire (30 gauge) is available in a variety of colors. It is strongly recommended that a color coding scheme is used when wiring the board.

**TASK:** Practice wiring these three chips in the following schematic together. Double check your connection with a multi-meter.

**Points:** You get 10 pts free with or without doing this lab