ECE-3120
Fall 2008

LAB 5 – Messing with Bits

The purpose of this lab is to reinforce your basic programming skills with the 68HCS12 using program loops, arrays, indirect addressing, bit shifting, bit testing, and bit manipulation instructions.

PRE-LAB:

Prepare pseudocode and the first draft of the program and calculate the expected results by hand (both decimal and hex). This must be completed before coming to the lab and shown to the lab instructor at the start of the lab session. Note: The Pre-Lab must be typed into a proper *.ASM source file, following our standard Program Format requirements.

Approved: Lab TA _____ *Date* _____

PROGRAMMING ASSIGNMENT:

Write a fully-commented program for the Dragon12 board, including appropriate directives and labels for memory operands and constants, called **Bit_mess.asm**. The program should do the following:

- In large, realistic programs, data is often distributed in various memory locations and these locations may actually change during program execution. One way to handle "migrating" data like this is to have a table, at a fixed address, containing the variable addresses of the data that may wander around in memory. Whenever a data item moves to a new location, its address in the table is updated accordingly. Indirect addressing can always be used to access the data through the table.
- Use directives to create a table (array) of memory addresses, each of which points to one item in a set of unsigned data bytes that are spread around at various locations in memory.
- Be able to process any set of 1 to 16 unsigned data bytes. This means that the program should be able to handle any set size in this range, with any set of values, by changing ONLY the directive defining the set size, without modifying any instructions.
- In the following steps, access the data set ONLY via the table of memory addresses, without using the individual data locations directly.

- For each byte in the data set, do the following steps in sequence, according to the order of their location in the memory address table. Do NOT modify the original data set bytes.
    1. Clear the Carry flag and then rotate a copy of the original byte to the left by 3 bits.
    2. Do these bit manipulations to the modified byte from step 1:
        a. If bits 6 and 5 of the byte are ones, then clear bits 3 thru 0;
        b. Else if bits 4 and 1 of the byte are zeros, then set bits 1 and 0;
        c. Else toggle bits 7 thru 4.
    3. Do these logical operations to the modified byte from step2:
        a. If this is NOT the last byte of the set, perform the logical OR of this byte and the next one in the set;
        b. But if this is the last byte in the set, perform the logical AND of this byte and the first one in the set.
    4. Count the number of ones in the modified byte from step 3 and store this count into the next consecutive location of the OnesCount array.
    5. Store the modified data byte from step 3 into the next consecutive location of the ResultByte array.
- The code must start at $1000.

The program directives should initialize the following set of **unsigned data bytes** in the data space at the indicated locations, where x:y indicates address x contains the data value y:
$1220:$15, $1227:$27, $1234:$EF, $1242:$90.

Program directives should initialize a table of memory addresses, beginning at address $1200, that points to each item in the data set in the order given above.

The program should store these results in memory:

- The ResultByte array of bytes starting in memory location $1300
- The OnesCount array of bytes starting in memory location $1320

Procedure: First, use D-Bug12 to fill memory locations $1000 through $14FF with zeros. Then assemble, download, and debug/execute the program as follows.

a. Set a breakpoint to stop execution at the end of each loop and run through the program to the end, pausing at each breakpoint to display the important values. Verify that each value is correct at the end of each loop and that all final results are correct at the end of the program. Use the listing file to determine the breakpoint address.
b. Then reset the processor (which also removes the breakpoint), download the program again, run it at full speed until it stops, and verify that the final values are still correct.
c. When finished debugging and executing, copy the entire terminal window output and paste it into a Notepad or Word document for inclusion in the report. You should edit out mistakes and unnecessary repetitions before submission.

Approved: Lab TA _____ *Date* _____

Things to turn in as your Lab Report, attached in this order:

1. This assignment sheet, with your name at the top, signed by the TA where shown.

2. Your uncorrected pre-lab document (commented source code).

3. A printout of the final **Bit_mess.asm and Bit_mess.lst** files. You'll need to print the listing file in landscape mode to make it fit. Use Notepad to print them.

4. A printout from the terminal screen (method: highlight text, type ctrl+c, and paste into a Notepad or Word document, using Courier New as the font) that includes everything done. Add brief hand-written comments and highlight important values at each step in the procedure, showing the relevant memory contents and register contents.

5. Answer the following questions, in your document:

    1) What are the final values, in both hex and decimal? Label each value clearly.
    2) Based on the given addresses and program requirements, is the number of bytes reserved adequate to handle any set of up to 16 data bytes for the 3 arrays: MemoryAddresses, ResultByte, and OnesCount? Explain.
    3) Write an alternative code sequences to accomplish steps 2a, 2b, and 2c. The code sequence should be different, perhaps longer and less efficient. There is always more than one way to accomplish things!