CpE390 Lab 3

(Chapter 4 of Textbook)

The CML-12S-DP256 is an updated development board with 256KB of memory, a somewhat more advanced development board than the Axiom CME12BC32 described in your textbook. The memory map for the textbook's CME-12BC is shown and described on page 105 of your textbook. As noted there, "you need to understand the memory map of the ... demo board in order to be able to download your program to the proper memory space."

The Axiom CML 12S-DP256 is an HC12 evaluation board developed by Axiom Manufacturing (www.axman.com), providing 256 kB of memory. Memory maps differ among evaluation boards with different amounts and distributions of memory. **When programming for the CML 12S-DP2256, you should be applying the following memory map.** I recommend that you use locations $4000-$7FFF for your program and data.

Axiom CML 12S-DP2256 Memory Map

ADDRESS	TYPE MEMORY	MEMORY APPLICATION
$C000 -$FFFF	FLASH	MON12, NOICE, and Utility firmware located in internal flash, Page $3F.
$8000 -$BFFF	External Ram	**User Paged Program Memory space**, Pages $20 - $2E. Note: Pages $30 - $3F reside in the internal flash.
$4000 -$7FFF	External Ram	**User Program Memory.** Emulate fixed page $3E.
$3F8C -$3FFD	Internal Ram	Ram Interrupt Vector Table
$3E00 -$3F8B	Internal Ram	Monitor reserved ram memory. Stacks and variables
$1000 -$3DFF	Internal Ram	**User Internal Ram memory**
$0400 -$0FEB	Internal EEprom	User EEprom memory, Monitor reserves $FEC - $FEF for Autostart, user should avoid $FF0 - $FFF memory use.
$0000 -$03FF	HCS12 Registers	**Monitor or user access to control registers.**

Should you have questions regarding the use of this memory map, consult with your TAs. The memory range from $8000 to $BFFF contains 16 kB of memory. Similarly, the memory range from $4000 to $7FFF contains 16 kB of memory.

2. Laboratory Project

As noted above, when using programs given in the textbook **you will need to reallocate the memory locations declared in those programs.**

2.1: Textbook Example 4.2 (page 113):

This example describes a program that searches an array to determine if it contains a specific value, the *key*. The program for this search is provided. The value of "key" is specified as 190 at the start of the program. Looking at the end of the program where "vec_x" is defined, you will see the values stored - and the value 190.

Step 1: Change the program on page 113 of your textbook as needed to use the memory map of the development board. Your report should show your modified program (with changes highlighted).

Step 2: Assemble and download your program.

Step 3: Run the program and determine whether the key was found. Explain how you determined whether or not a key was found based on the variables used in your program (i.e., where is the needed variable stored, how did you read that variable after running the program, and what was the value found).

Step 4: Replace the value 190 in "vec_x" with a different number (using D-Bug12) and rerun your program (without changing your program). What is the value of the key that your program is trying to find? Did it find that value? How do you know?

2.2: Textbook Example 4.7 (Page 122):

In this case, the program counts characters and words in a document, a standard function in programming languages.

Step 1: Change the program on page 122 of your textbook as needed to use the memory map of the development board. Your report should show your modified program (with changes highlighted).

Step 2: Assemble and download your program.

Step 3: Run the program and determine whether the correct number of characters and words in the example's entry (string_x) is found. In what memory

location does the program store the number of characters. Similarly, where is the number of words stored? What are the values in those memory locations after running the program? Do white spaces count as characters?

Step 4: Change sentence in "string_x" and rerun the program. What is the purpose of the "fcb 0" statement following the "string_x" sentence? After changing "string_x", rerun the program and verify that it ran correctly.

2.3: Subroutines, Keyboards, and Terminal Display (Based on Textbook Example 4.7 on Page 122:

In this problem, you will be using the same basic program as used above. Rather than being a "main" program, the part of the program following the line "org $1000" on page 122 will be changed into a subroutine, with its own name (call it "wdcnt") and starting address, You will need to add the return from subroutine command at the end of your subroutine. Complete each of the following steps

Step 1: Starting with the program you used in Problem 2.2 above, change the counting portion of the program into a subroutine named "wdcnt" preceded by an "org xxxx" specifying the starting address xxxx for the subroutine. Include the rts command and end command at the end of the subroutine.

Step 2: Write a main program (in the program on page 122 of your textbook, it would be entered following the "org $1000" command, though you may have modified that address to match the memory map of the development board) that simply calls the subroutine "wdcnt".

Step 3: Assemble and download your program.

Step 4: Run the program and determine whether the correct number of characters and words in the example's entry (string_x) is found. In what memory location does the program store the number of characters. Similarly, where is the number of words stored? What are the values in those memory locations after running the program? Do white spaces count as characters?