1) From mycourses download this pdf, dxp_Lab4_ArrayFill_c1.c, and dxp_Lab4_ArrayFill_a1.asm.
2) The purpose of this lab is to continue your understanding of the use of instructions, directives and addressing modes, as well as the allocation of data into both volatile and non-volatile memory. You will also gain further insight into programming in C and assembly for MSP430.

3) **Part1: Modify fmlxxxx_ArrayFill_a1.asm to fill down.**

4) Create a new assembly only project in your ee420 workspace named fmlxxxx_Lab4_a1. Create a copy of the dxp_Lab3_ArrayFill_a1.asm code from last lab and rename it fmlxxxx_lab4_FillDown_a1.asm. Then add it to your fmlxxxx_Lab4_a1 project. Modify fmlxxxx_lab4_FillDown_a1.asm so that it stores a decreasing index value in each location starting with 0x0f at the first location, and stores the number of array elements accessed in the process (NOT the sum of the actual elements) in the location **SUM.** Build and debug the project. Proper operation of your program would result in **Array** being filled with decreasing values from 0x0f down to 0x00, and **SUM** will contain the number of times that the elements in the array were accessed in the process.

5) **Part2: Modify fmlxxxx_ArrayFill_a1.asm to fill in different rows.**

6) Create a new assembly only project in your ee420 workspace named fmlxxxx_Lab4_a2. Create a copy of the dxp_Lab3_ArrayFill_a1.asm and rename it fmlxxxx_lab4_FillRows_a2.asm and add it to your fmlxxxx_Lab4_a2 project. Modify fmlxxxx_lab4_FillRows_a2.asm so that it stores the constant labeled **Zeroes** in the array elements referred to by the label **ROW0,** the constant labeled **Ones** in the array elements referred to by the label **ROW1,** the constant labeled **Odds** in the array elements referred to by the label **ROW2** and the constant labeled **Evens** in the array elements referred to by the label **ROW3.** Store the sum of all array elements contents in the location **SUM** (this time it really IS the sum)**.** Build and debug the project. Proper operation of your program would result in **ROW0** elements containing the value 0x00, **ROW1** elements will contain the value 0xff, **ROW2** elements will contain the value 0x55, **ROW3** elements will contain the value 0xaa, and **SUM** will contain the sum of the values in all four rows of Array.

7) **Part3: Modify fmlxxxx_ArrayFill_c1.c to fill down and fill in different rows.**

8) Create two new projects in Code Composer named fmlxxxx_Lab4_c1 and fmlxxxx_Lab4_c2. Add fmlxxxx_ArrayFill_c1.c to each of these projects. Rename the file in fmlxxxx_Lab4_c1 to fmlxxxx_FillDown_c1.c, and in fmlxxxx_Lab4_c2 to fmlxxxx_FillRows_c2.c. Modify each so that the effects of processing the arrays are identical to those in parts 1 and 2. **Take note on the data types used in C** for your array, and your fill values for "ones", "zeroes", "evens" and "odds". The resulting

memory values have to be identical to those in the assembly parts. In C, this compiler will use 16 bits for an integer data type, and 8 bits for a char data type. Build and debug the projects.

9) Determine, compare, and include in your report the machine code sizes for fmlxxxx_FillDown_a1.asm, fmlxxxx_FillDown_c1.c, fmlxxxx_FillRows_a2.asm, and fmlxxxx_FillRows_c2.c.

10) **<u>Part4</u>: Create a program that multiplies two arbitrary 8-bit numbers.**

11) Create an assembly program, and separately a C program, that will multiply two arbitrary, unsigned, 8-bit numbers. The source operands and final result in the assembly program are stored in two registers. Recall binary multiplication from EE240 Introduction to Digital Systems, or equivalent course, or from a lecture preceding this lab. You should be using roll/shift functions, and adding the partial products by keeping a running sum. The multiplication will always go through the loop (8) times, regardless of the operands. Write down the algorithm steps. Come up with a program flow chart. Include these in your report. Create an assembly only project for fmlxxxx_Lab4_a3 and a separate C project for fmlxxxx_Lab4_c3. Build and debug these programs. Determine, compare, and include in your report the machine code sizes for these two programs.

12) **<u>Note</u>:** You are not allowed to use the built-in hardware multiplier. To avoid this in the C program, implement the same algorithm as in assembly, i.e. using shifts and additions. If you do not implement the algorithm seen in class, which uses roll/shifts instructions, and adds the partial products, you will receive ZERO credit for this part! If you are not sure you are implementing the correct algorithm, ASK your TA or instructor BEFORE the lab is due.

13) Make sure you write the report and upload it along with your project archives on mycourses. As time permits, demo the intermediate steps to the TA.

14) Grading:

   a. Part 1 = 10 points

   b. Part 2 = 10 points

   c. Part 3 = 10 points

   d. Part 4 = 10 points.