Laboratory #2
PIC24 Assembly & C Programming

Objective: To learn the Assembly and C programming of the PIC24 MCU.

Introduction:

In this lab exercise you will write and simulate assembly and C programs for the PIC24 MCU. Follow the "New project set-up" procedure from lab 1 to create your projects for this assignment. You will also apply the debugging techniques studied in the previous lab for testing and verifying your programs.

Refer to the lecture notes for the general syntax of Assembly and C programs for the PIC24 MCU target. Note that assembly program source codes should have the file extension ".s". This will enable you to easily use the C compiler driver without having to specify the option to tell the driver that the file should be treated as an assembly file. See the *MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User's Guide (DS51284)* for more details on the C compiler driver.

Program documentation and code efficiency:

Proper documentation and program efficiency are important things to keep in mind when designing and implementing your lab projects.

How much comment is good enough? The answer to this question and the way comments are formatted is sort of subjective. I personally prefer providing good documentation at the beginning of the source code and inserting comment lines as necessary in the body of the program to explain what the different blocks of code are for. Therefore, I expect your program documentation to include good description about the purpose of the program, its parameters, side effects, algorithm description (when appropriate), and comments inserted in the body of the source code as necessary. Also do not forget to include, somewhere at the beginning, the authors' names and the last modification date of the program.

How efficient is your code? Obviously, if there is a "better" way (both in terms of program code length and execution time) of doing the same task, you should do it using the better way, while keeping elegance and readability in mind. Thus, when designing and implementing algorithms putting the extra effort in the initial stage to come up with a well thought out design will certainly pay off by achieving a more efficient solution and better debugging experience. For example, in Assembly programming, you could find "better" ways of doing things with "smart" choices of instructions and addressing modes.

In all the lab assignments in this course both program efficiency and proper documentation are gradable features.

Tasks:

1) Assembly program to copy an array of data from source to destination locations.

Your program will have three input parameters, namely, source array address (*src*), destination array address (*dst*), and array length (*len*). All the parameters are 16-bit wide each. Assume the elements of the array are 16-bit numbers, thus the length of the array (*len*) specifies the number of 16-bit numbers in the source array to be copied to the destination. If the array length is zero no action should be taken. Assume the locations of the arrays to be non-overlapping.

Write a well-commented and efficient Assembly program (.s) to implement this task. Refer to the lecture2 slides for information about the format of a PIC24 assembly program. To avoid linker error the entry point for your program should be defined as:

```
.global __reset
```

For debugging purpose, create some sample data in the data memory area (i.e. starting at address 0x0800 in the PIC24FJ128GA010). Observe the variables and affected Special Function Registers (SFRs) in a watch window and verify the data copy operation by examining the data memory display ("View -> File Registers").

2) C program to implement string search.

Design an algorithm and implement a C program for the PIC24 MCU target that performs a string search operation as follows. It is NOT allowed to use the C "string library" functions to do the work. The declarations for the input string and the search pattern, as well as the output of the search operation are to be defined as shown below:

```
char text[] = "The quick brown fox jumps over the lazy dog.";
char look[] = "fox";
int   found, pos;
```

Where, *text[]* is the source string, defined as an array of characters with a given initialization string. *look[]* is the search string that we want to locate in the source string, if it is present. The results of the search process are returned in *found* and *pos*. *found* is an integer quantity that should be assigned 1 if the search string is found in the source string, otherwise it should be assigned 0. If the search string is found in the source string, then *pos* should return its location.

<u>Checkout and report</u>:

Demonstrate your working programs to the lab instructor and hand in a lab report that includes the following:

- Title page: course #, course title, Lab# & title, date, professor's name, names of group members
- This lab handout.
- Printout of the source codes for the two tasks.
- Printout of the disassembly listing files for both tasks.