1. CPE 325: Laboratory Assignment #3
MSP430 IAR Assembly Language Programming

Objectives: This tutorial will help you get started with the MSP30 IAR Assembly program development. You will learn the following topics:

- Assembly language programming
- Creating an application project using assembly language programs
- Debugging using the IAR C-SPY® Debugger (Simulator)

Note:
It is required that students have completed the tutorial titled "The MSP430 IAR Embedded Workbench," before starting with this one.

1.1. Count Number of Characters 'E' in a String: Problem Statement

This section defines the problem that will be solved by the Count Characters program using the MSP430 assembly language. Our task is to develop an assembly program that will scan a given string of characters, for example, "HELLO WORLD, I AM THE MSP430!", and find the number of appearances of the character 'E' in the string. A counter that records the number of characters 'E' is then written to the parallel port P1. The port should be configured as an output port, and the binary value of the port will correspond to the counter value.

Step 1: Analyze the assignment.
To solve this assignment, let us first analyze the problem statement.
Your task is to write an assembly program that will count the number of characters 'E' in a string. First, the problem implies that we need to allocate space in memory that will keep the string "HELLO WORLD, I AM THE MSP430!". The string has 29 characters and they are encoded using the ASCII table. To allocate and initialize a string in memory we can use an assembly language directive, for example DB (Define Byte).

```
DB "HELLO WORLD, I AM THE MSP430!"
```

We can also put a label to mark the beginning of this string in memory, for example, mystr:

```
mystr DB "HELLO WORLD, I AM THE MSP430!"
```

When assembler sees the DB directive, it will allocate the space in memory required for the string that follows and initialize the allocated space with corresponding ASCII characters. The assembler will also automatically add an ASCII NULL character at the end of each string (ascii(NULL)=0x00). So, the total number of bytes occupied by this string is 30.

Step 2: Develop a plan.

Our task is now to write a small program that will scan the string, character by character, check whether the current character is equal to the character 'E', and if yes, increment a counter. The string scan is done in a program loop. The program ends when we reach the end of the string, which is detected when the current character is equal to the NULL character (0x00).
To scan the string we will use a register to point to the current character in the string. This pointer register is initialized at the beginning of the program to point to the first character in the string. The pointer will be incremented in each iteration of the program loop. Another register, initialized to zero at the beginning, will serve as the counter, and it is incremented every time the current character is 'E'.
After we exit the program loop, the current value of the counter will be written to the port P1, which should be initialized as an output port.
Note: It is required that you are familiar with the MSP430 instruction set and addressing modes to be able to solve this problem. Also, we will assume that the string is no longer than 255 characters, so the result can be displayed on an 8-bit port.

1.1.1. Count Characters

1.1.2. Assembly Code

Figure 1 shows the assembly code for this program. Here is a short description of the assembly code.
The comments in a single line start with a column character (;). Multi-line comments can use C-style /* comment */ notation.
Line 10, `#include <msp430xG46x.h>`, is a C-style pre-processor directive that specifies a header file to be included in the source code. The header file includes all macro definitions, for example, special function register addresses (WDTCTL), and control bits (WDTPW+WDTHOLD).
The ORG assembly directive in line 11 is used to set the program location counter of the current segment to the value of an expression that follows. Thus, `ORG 0FF00h` sets the location counter at the absolute address 0FF00h. This means that the location counter is moved to this address.
Next, in line 13 we allocate the string myStr that starts at the location 0FF00h using `DB` directive: `myStr DB "HELLO WORLD, I AM THE MSP430!"`. As explained, this directive will allocate 30 bytes in memory starting at the address 0FF00h and initialize it with the string content, placing the ASCII codes for the string characters in the memory. The hexadecimal content in memory will be as follows: 48 45 4c 4c 4f 20 57 4f 52 4c 44 2c 20 49 20 41 4d 20 54 48 45 20 4d 53 50 34 33 30 21 00 (ascii('H')=0x48, ascii('H')=0x45, … ascii('!')=0x21, ascii(NULL)=x00).
How does my program execute on an MSP430? Upon powering-up the MSP430 control logic always generates a RESET interrupt request (it is the highest priority interrupt request). The value stored at the address 0xFFFE (the last word in the 64KB address space) is reserved to keep the starting address of the reset handler (interrupt service routine), and the first thing that the microcontroller does is to fetch the content from this address and put it in the program counter (PC, register R0). Thus, the starting address of our program should be stored at location 0xFFFE. Here in line 20, we move location counter to 0xFFFE and allocate 2 bytes (DC16 allocates 16 bits or two bytes) that will be initialized with the starting address of the main program. The starting

address of the main program is marked by the label `main`, and the memory location 0xFFFE is initialized with the value of the label main (which is the starting address of the main program). RSEG is a segment control assembler directive that controls how code and data are located in memory. `RSEG` is used to mark the beginning of a relocatable code or data segment. `CODE` and `DATA` are recognized segment types that are resolved by the linker. The IAR XLINK linker can recognize any other type of segment (e.g., `CSTACK` for code stack).
First instruction initializes the stack pointer register (`MOV #SFE(CSTACK), SP`). Our program does not use the stack, so we could have omitted RSEG CSTACK and this instruction.
The instruction `MOV.W #WDTPW+WDTHOLD,&WDTCTL` sets certain control bits of the watchdog timer control register (`WDTCTL`) to disable it. The watchdog timer by default is active upon reset, generating interrupt requests periodically. As this functionality is not needed in our program, we simply need to disable it.
Parallel ports in the MSP430 microcontroller can be configured as either input or output ports. A control register PxDIR determines whether the port x is an input or an output port (we can configure each individual port pin). Our program drives all eight pins of the port P1, so it should be configured as an output port by setting each individual pin to 1 (`P1DIR=0xFF`). Register `R4` is loaded to point to the first character in the string. Register R5, the counter, is cleared before starting the main program loop.
The main loop starts at the next label. We use the autoincrement addressing mode to read a new character (one byte) from the string (`MOV.B @R4+, R6`). The current character is kept in register R6. We then compare the current character with the NULL character (`CMP #0, R6`). If it is the NULL character,the end of the string has been reached and we exit the loop (JEQ lend). Pay attention that we used JEQ instruction? Why?
If it is not the end of the string, we compare the current character with 'E'. If there is no match we go back to the first instruction in the loop. Otherwise, we increase the value of the counter (register R5).
Finally, once the end of the string has been reached, we move the lower byte from R5 to the parallel port 1, `P1OUT=R5[7:0]`.

```
1.  /*-------------------------------------------------------------------
2.  * Program     : Counts the number of characters E in a string
3.  * Input       : The input string is the myStr
4.  * Output      : The port one displays the number of E's in the string
5.  * Written by : A. Milenkovic
6.  * Date        : August 14, 2008
7.  * Description: MSP430 IAR EW; Demonstration of the MSP430 assembler
8.  *-------------------------------------------------------------------*/
9.
10. #include "msp430.h"                    ; #define controlled include file
11.
12.         ORG 0FF00h
13. myStr   DB "HELLO WORLD, I AM THE MSP430!" ; the string is placed on the stack
14. ; the null character is automatically added    ; after the '!'
15.
16.         NAME    main                  ; module name
17.
18.         PUBLIC  main                  ; make the main label visible
19.                                       ; outside this module
20.         ORG     0FFFEh
21.         DC16    main                  ; set reset vector to 'main' label
```

```
22.
23.         RSEG    CSTACK                  ; pre-declaration of segment
24.         RSEG    CODE                    ; place program in 'CODE' segment
25.
26. main:   MOV     #SFE(CSTACK), SP        ; set up stack
27.         MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
28.         BIS.B   #0FFh,&P1DIR            ; configure P1.x output
29.         MOV.W   #myStr, R4              ; load the starting address of the string
30.                                         ; into the register R4
31.         CLR.B   R5                      ; register R5 will serve as a counter
32. gnext:  MOV.B   @R4+, R6                ; get a new character
33.         CMP     #0,R6                   ; is it a null character
34.         JEQ     lend                    ; if yes, go to the end
35.         CMP.B   #'E',R6                 ; is it an 'E' character
36.         JNE     gnext                   ; if not, go to the next
37.         INC     R5                      ; if yes, increment counter
38.         JMP     gnext                   ; go to the next character
39.
40. lend:   MOV.B   R5,&P1OUT                ; Set all P1 pins
41.         BIS.W   #LPM4,SR                ; LPM4
42.         NOP                             ; Required only for debugger
43.
44.         END
```

Figure 1. MSP430 Assembly Code for Count Character Program (Lab2_D1.s43).

1.1.3. Creating an Application Project

Please consult the tutorial on how to create a new workspace and project.

Step 1. Choose Project>Create New Project. Select Empty project and save it as Lab2_D1.

Step 2. Add the source file Lab2_D1.s43 to the project. Choose Project>Add Files.

Step 3. Choose Project>Options and set all categories as explained in the Getting Starting With MSP430 IAR Embedded Workbench tutorial.

Step 4. Select the source file Lab2_D1.s43 and compile it. (Choose Project>Compile). Click on the list file Lab2_D1.lst to see report generated by the assembler.

Step 5. Link the program. Choose Project>Make. The application is ready for debugging.

1.1.4. Program Simulation and Debugging

In this section we will discuss how to simulate program execution.

Step 1: Go to Project>Options and select Debugger category. In the Driver box, choose Simulator (Figure 2). Click OK. Here we use absolute assembly code (our program defines directly the location of code and data). Consequently, we need to let linker know to use absolute assembly code (set settings as shown in Figure 2).

Step 2: Choose Project>Debug. You will see the EW display as shown in Figure 3.

Step 3: Step through the program step by step (use F11). Observe the Disassembly, Register View, and Memory View windows. Answer the following questions.
What is the starting address of the program?
How many clock cycles does each instruction take to execute?
Observe the contents of memory location and registers as you step through the program. What is the content of the memory location at the address 0xFFFE?

What are the addresses of the special-purpose registers P1DIR and P1OUT? Monitor the contents of these locations as you walk through your program. Set breakpoints to move easier through your program.

Step 4: Choose Stop Debugging.

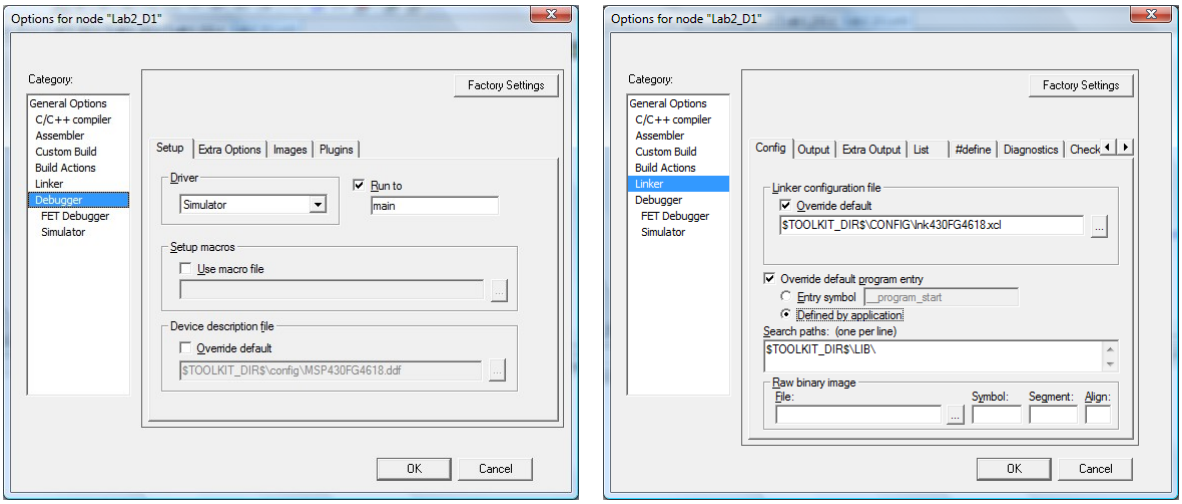

Figure 2. Debugger category.

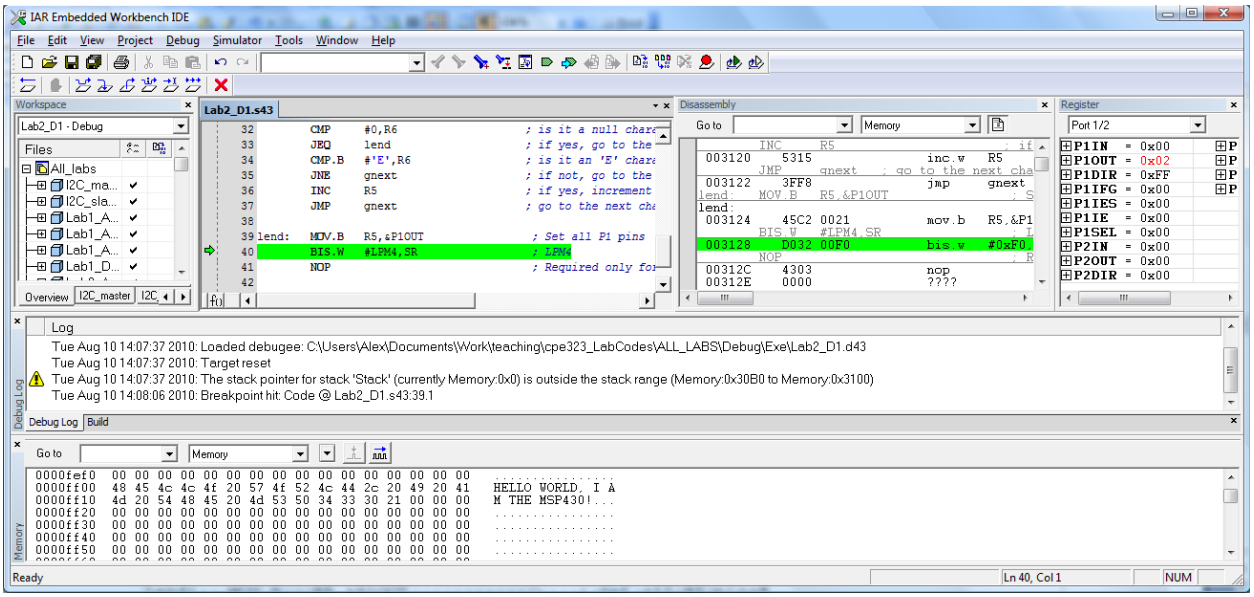

Figure 3. IAR Simultor.

1.2. References

It is important to be comfortable with the MSP430 instruction set and addressing modes. Your text and the MSP430 user guide contain invaluable information that you will need to understand in order to effectively program the MSP430. Please read the following:

- Chapter 5, pages 119 – 146 in the Davies text
- Sections 4.4 and 4.5 in the MSP430FG4618 User's Guide on Addressing modes and the instruction set (pages 131 – 173)

1.3. Assignment

Write an assembly program that creates a string of digits, counts the number of digits and calculates the sum of the digits. The sum is output to P1 and the number of digits is output to P2.