CPE 325: Embedded Systems Laboratory
Laboratory #4 Tutorial
Introduction to MSP430 Assembly Language Programming

Aleksandar Milenković
Email: milenka@uah.edu
Web: http://www.ece.uah.edu/~milenka

Objective:

This tutorial will help you get started with MSP30 assembly program development. In this lab, you will learn the following topics:

Assembly language programming
Creating an application project using assembly language programming
Debugging the application using CCS Debug

Notes:

All previous tutorials introducing the TI Experimenter's Board and the Code Composer Studio software development environment are required for successful completion of this lab. Class materials required are: MSP430 ISA and MSP430 Assembly Language Programming.

Contents:

1 Problem Statement: Count Number of Characters ‘E’ in a String

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. Two single quotes at the end allocate a NULL character (0x00). To allocate and initialize a string in memory we can use an assembly language directive `.string` (or `.byte`).

```
myStr   .string "HELLO WORLD, I AM THE MSP430!", ''
```

A label myStr marks the beginning of this string in memory, and the value of this symbol corresponds to the starting address of the string. When assembler sees the `.string` directive, it will allocate the space in memory required for the string that follows and initialize the allocated space with corresponding ASCII characters. Including the ASCII NULL character at the end of the string (`ascii(NULL)=0x00`), the total number of bytes occupied by this directive 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.

2 Counting Characters

This section will go more in depth into how to write the assembly code to solve this problem in addition to how to build and link the code for the MSP430.

2.1 Assembly Program

The assembly code for this program can be seen on the next page. However, here is a brief description of how the code works.

The comments in a single line start with a column character (;). Multi-line comments are not supported. Note the proceeding (-) characters are used to indicate different sections of the assembly file and are good practice for clean readable code. In line 11, you will notice an assembly 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). Next, in line 17 we allocate the string that is going to be counted. As explained, this directive will allocate 30 bytes in memory and initialize it with the string content, placing the ASCII codes for the string characters in the memory.

So how does the 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 corresponding to the entry point called RESET should be stored at location 0xFFFE. Line 57 declares the reset vector that is then initialized with the value that corresponds to the symbol RESET, which is the starting address of our program.

The first instruction on line 26 initializes the stack pointer register. However, our program does not use the stack, so we can ignore this instruction for now. The instruction on line 27 will set certain control bits of the watchdog timer control register (WDTCTL) to disable it. The watchdog timer by default is active upon reset, generating periodic system resets if left unattended. As this functionality is not needed in our program, we 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 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 bit of the direction register 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 loop starts at the next label called gnext. We use the autoincrement addressing mode to read a new character (one byte) from the string as seen in line 36. The current character is kept in register R6. We then compare the current character with the NULL character on line 37. 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 on line 44.

```
1   ;-------------------------------------------------------------------------------
2   ; File       : Lab4_D1.asm (CPE 325 Lab4 Demo code)
3   ; Function   : Counts the number of characters E in a given string
4   ; Description: Program traverses an input array of characters
5   ;              to detect a character 'E'; exits when a NULL is detected
6   ; Input      : The input string is specified in myStr
7   ; Output     : The port P1OUT displays the number of E's in the string
8   ; Author     : A. Milenkovic, milenkovic@computer.org
9   ; Date        : August 14, 2008
10  ;-------------------------------------------------------------------------------
11          .cdecls C,LIST,"msp430.h"       ; Include device header file
12
13  ;-------------------------------------------------------------------------------
14          .def    RESET                   ; Export program entry-point to
15                                          ; make it known to linker.
16
17  myStr:  .string "HELLO WORLD, I AM THE MSP430!", ''
18  ;-------------------------------------------------------------------------------
19          .text                           ; Assemble into program memory.
20          .retain                         ; Override ELF conditional linking
21                                          ; and retain current section.
22          .retainrefs                     ; And retain any sections that have
23                                          ; references to current section.
24
25  ;-------------------------------------------------------------------------------
26  RESET:  mov.w   #__STACK_END,SP         ; Initialize stack pointer
27          mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
28
29  ;-------------------------------------------------------------------------------
30  ; Main loop here
31  ;-------------------------------------------------------------------------------
32  main:   bis.b   #0FFh,&P1DIR            ; configure P1.x output
33          mov.w   #myStr, R4              ; load the starting address of the string
34  into R4
35          clr.b   R5                      ; register R5 will serve as a counter
36  gnext:  mov.b   @R4+, R6                ; get a new character
37          cmp     #0,R6                   ; is it a null character
38          jeq     lend                    ; if yes, go to the end
39          cmp.b   #'E',R6                 ; is it an 'E' character
40          jne     gnext                   ; if not, go to the next
41          inc.w   R5                      ; if yes, increment counter
42          jmp     gnext                   ; go to the next character
43
44  lend:   mov.b   R5,&P1OUT               ; set all P1 pins (output)
45          bis.w   #LPM4,SR                ; LPM4
46          nop                             ; required only for Debugger
47
48
49  ;-------------------------------------------------------------------------------
50  ; Stack Pointer definition
51  ;-------------------------------------------------------------------------------
```

```
52          .global __STACK_END
53          .sect   .stack
54
55  ;-------------------------------------------------------------------------------
56  ; Interrupt Vectors
57  ;-------------------------------------------------------------------------------
58          .sect   ".reset"                ; MSP430 RESET Vector
59          .short  RESET
60          .end
61
```

Figure 1. MSP430 Assembly Code for Count Character Program (Lab4_D1.asm)

2.2 Creating an Application Project

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

Step 1. Choose Project>Create New CCS Project. This time you need to select Empty Assembly-only Project and save it as Lab4_D1.

Step 2. Download and copy Lab4_D1.asm into the project and delete main.asm by right clicking it and selecting delete or by clicking the delete key.

Step 3. Right-Click the project and select Properties then set all options as you did in Lab 1.

Step 4. Select the source file Lab4_D1.asm and compile it. (Right-Click->Build). Click on the list file Lab4_D1.lst to see report generated by the assembler.

Step 5. Link the program  . The application is ready for debugging.

2.3 Program Simulation and Debugging

In this section we will discuss how to debug the program.

Step 1. Click the  icon.

Step 2. Step through the program step by step (use F6). 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? How do the clock cycles taken by each instruction depend on addressing mode?

Hint: http://processors.wiki.ti.com/index.php/Profile_clock_in_CCS

Step 3. 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.

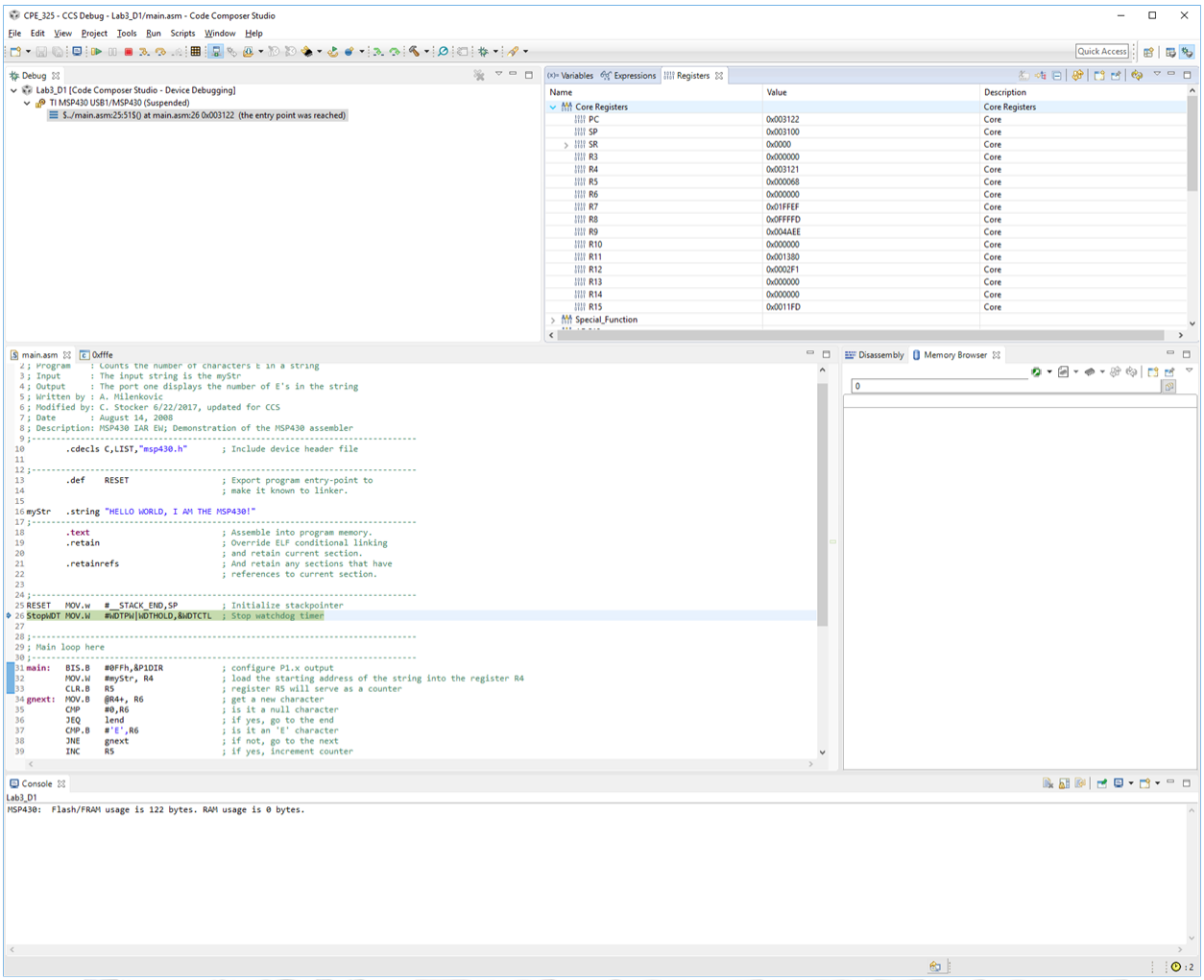

Figure 2. Code Composer Studio Debugging Window

3 References

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

- MSP430 ISA
- MSP430 Assembly Language Programming
- Chapter 5, pages 119 - 146 in John H. Davies' MSP430 Microcontroller Basics
- Sections 4.4 and 4.5 in the MSP430FG4618 User's Guide on Addressing modes and the instruction set (pages 131 - 173)