Embedded Systems Lab

CPE 325-02

Introduction to Assembly on the MSP430

By: David Thornton

Lab Date: September 10, 2020

Lab Due: September 22, 2020

Demonstration Due: September 22, 2020

Introduction

This lab introduces assembly programming with the MSP430.

Theory

Topic 1: Assembler Directives

- “Assembler directives supply data to the program and control the assembly process. Assembler directives enable you to do the following: Assemble code and data into specified sections, reserve space in memory for uninitialized variables, control the appearance of listings, initialize memory, assemble conditional blocks, define global variables, specify libraries from which the assembler can obtain macros, examine symbolic debugging information.”
    - Source: MSP430 Assembly Language Tools, Page 67

Topic 2: Different Addressing Modes

a. Give an example of indirect addressing with auto increment.
    i. From Lab_4_Q1.asm, `mov.b @R4+, R6`

- “The MSP430 architecture has seven possibilities to address its operands. Four of them are implemented in the CPU, two of them result from the use of the program counter (PC) as a register, and a further one is claimed by indexing a register that always contains a zero (status register).”
    - Source: Architecture and Instruction Set, Page 8-4

As	Ad	Addressing Mode	Syntax	Description
00	0	Register Mode	Rn	Register contents are operand
01	1	Indexed Mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word
01	1	Symbolic Mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed Mode X(PC) is used
01	1	Absolute Mode	&ADDR	The word following the instruction contains the absolute address.
10	-	Indirect Register Mode	@Rn	Rn is used as a pointer to the operand
11	-	Indirect Autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards
11	-	Immediate Mode	#N	The word following the instruction contains the immediate constant N. Indirect Autoincrement Mode @PC+ is used

Source: Instruction Set Summary, Page 5-5, Table 5.2

Lab Assignment

1. For this lab assignment, please implement an assembly program that counts the number of words and sentences in a string variable. A sentence always ends with either '.', '!' or '?'. The string can be hard-coded in the program as a *cstring* type. Please store the count values in variables (defined using .data in your code). You should display this value using the memory browser.

   Hint: Please create an assembly project as mentioned in the tutorial for Lab4. You do not need to print the result in the console window.

   Hint: You need to declare your variable in data segment (.data) of the code to make them readable and writable as shown in the example below:

   .data

   sent_count: .int 0

   w_count: .int 0

2. Write an assembly program where you would define a variable which is a string. This character array should indicate a mathematical expression. For example: your mathematical expression can be as follows (which can be evaluated to an integer).

   "4-3+5"

   You are required to evaluate the string and send the value to P2OUT. You should demonstrate the value using the register window.

   Hint: The mathematical expression can be formed with all single digit numbers only. The mathematical operators can be restricted to "+" and "-".

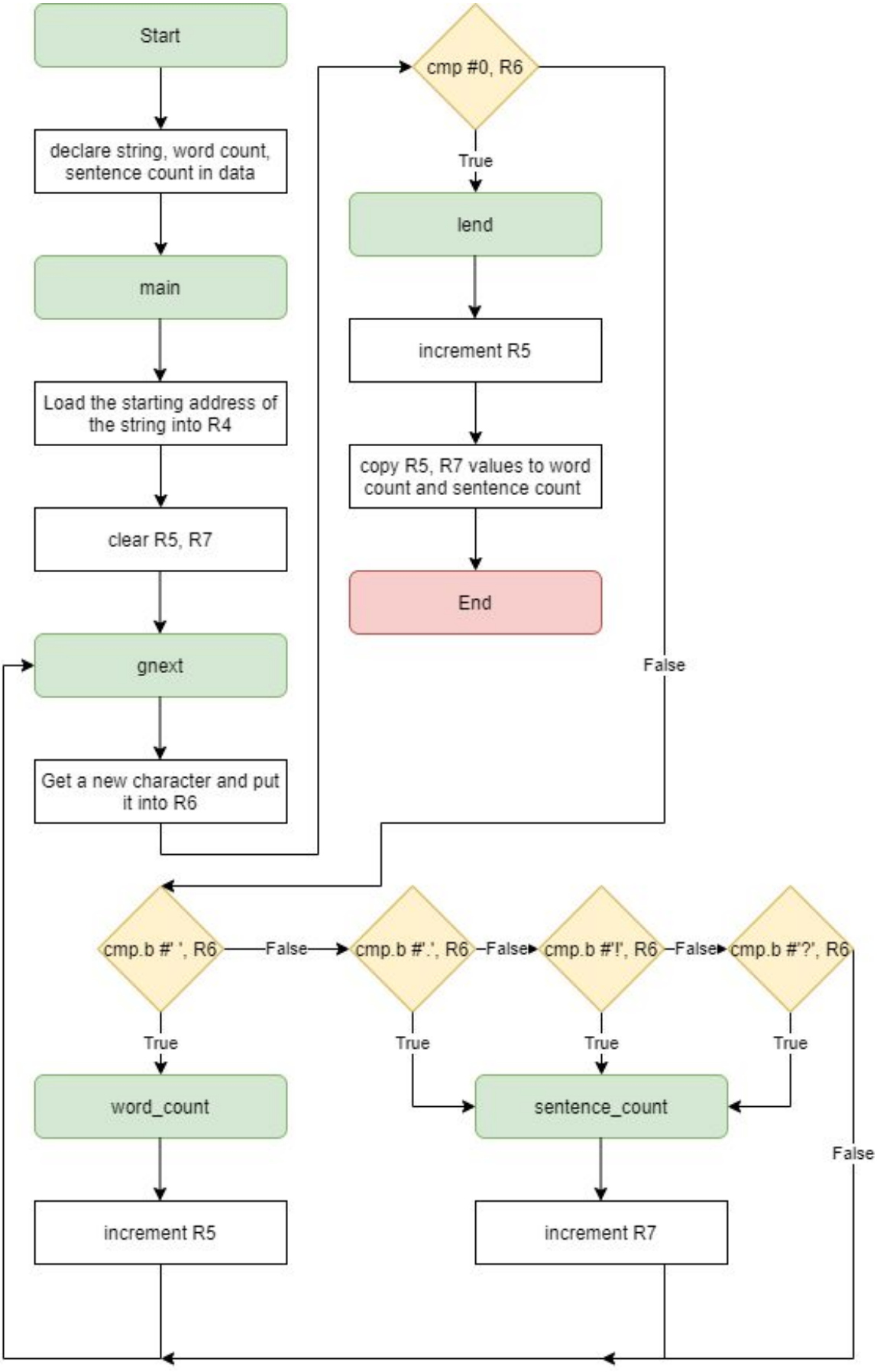
3. Write an assembly program where you would declare a string as shown below. You are required to update the contents of the strings to uppercase letters if they are lowercase letters. This means you need to change the value in their original location. For demonstration, you must present the updated values in the memory browser.

   Hint: You need to declare your variable in the data segment (.data) of the code to make them readable and writable as shown below.

   .data

   myString: .cstring "I enjoy learning msp430"

Flow Chart

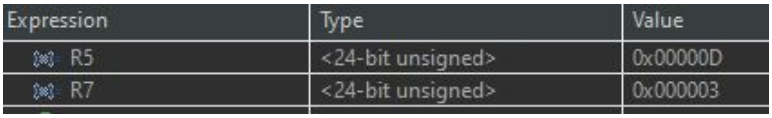

Start
declare string, word count, sentence count in data
main
Load the starting address of the string into R4
clear R5, R7
gnext
Get a new character and put it into R6
cmp #0, R6
True
lend
increment R5
copy R5, R7 values to word count and sentence count
End
False
cmp.b #' ', R6
False
cmp.b #'.', R6
False
cmp.b #'!', R6
False
cmp.b #'?', R6
True
True
True
True
word_count
sentence_count
increment R5
increment R7
False

Observations

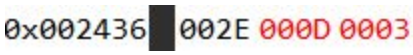
The programs satisfy the assignment requirements.

Question 1 Results:

Register Window:



Memory Window:



Question 2 Results

Register Window:

Question 3 Results:

Memory Window:

Conclusion

This lab expanded my knowledge of assembly instructions, Code Composer Studio, and how to run and debug a program on the MSP-EXP430F5529LP. The most significant issues I faced during this lab were with question 2. I had difficulty with keeping the running sum in ASCII.

Demo link

Appendix

Appendix 1: Lab_4_Q1.asm

```
;-------------------------------------------------------------------------------------------------------
; File: Lab_4_Q1.asm
; Description: Counts the number of words and sentences in a string
; Input: Hard coded string
; Output: R5 and R7 contain word and sentence count respectively
; Author: David Thornton
; Lab Section: 2
; Date: September 22, 2020
;-------------------------------------------------------------------------------------------------------
                .cdecls C,LIST,"msp430.h"       ; Include device header file
;-------------------------------------------------------------------------------------------------------
                .def    RESET                           ; Export program entry-point to
make it known to linker
                .data                                   ; Declare variable in a data
segment
myStr:  .cstring "This is a test! Is CPE 325 fun? Depends on who you ask."
w_count:        .int    0                               ; Given word count variable
sent_count:     .int    0                               ; Given sentence count variable
                                                                ; Expected: words =
13, sentences = 3
;-------------------------------------------------------------------------------------------------------
                .text                                   ; Assemble into program
memory
                .retain                                 ; Override ELF conditional
linking and retain current section
                .retainrefs                             ; Retain any sections that
have references to current section
;-------------------------------------------------------------------------------------------------------
RESET:  mov.w   #__STACK_END, SP                ; Initialize stack pointer
                mov.w   #WDTPW|WDTHOLD, &WDTCTL   ; Stop watchdog timer
;-------------------------------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------------------------------
main:   mov.w   #myStr, R4                      ; Load the starting address of the string
into R4
                clr.b   R5                              ; R5 is the counter for
words
                clr.b   R7                              ; R7 is the counter for
sentences

gnext:  mov.b   @R4+, R6                        ; Get a new character and put it into R6
```

```
                cmp     #0, R6                          ; Check to see if it is NULL
                jeq     lend                            ; If yes, go to the "lend" subroutine
                cmp.b   #' ', R6                        ; Check to see if R6 is the ' '
character
                jeq     word_count                      ; If yes, go to the "word_count"
subroutine
                cmp.b   #'.', R6                        ; Check to see if R6 is the '.'
character
                jeq     sentence_count          ; If yes, go to the "sentence_count"
subroutine
                cmp.b   #'!', R6                        ; Check to see if R6 is the '!'
character
                jeq     sentence_count          ; If yes, go to the "sentence_count"
subroutine
                cmp.b   #'?', R6                        ; Check to see if R6 is the '?'
character
                jeq     sentence_count          ; If yes, go to the "sentence_count"
subroutine
                jmp     gnext                           ; Go to the next character

word_count
                inc.w R5                                        ; Increment word counter
                jmp     gnext                           ; Go to the next character

sentence_count
                inc.w R7                                        ; Increment sentence
counter
                jmp     gnext                           ; Go to the next character

lend:   inc.w R5                                        ; Since the end of the sentence
does not end with a space,
                                                                ; you have to add
one to the word count.
                mov.w   R5, &w_count                    ; The memory address of w_count
gets the contents of R5
                mov.w   R7, &sent_count                 ; The memory address of
sent_count gets the contents of R7
                nop                                             ; Required only for
debugger

;-------------------------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------------------------
                .global __STACK_END
                .sect   .stack
;-------------------------------------------------------------------------------------------------
; Interrupt Vectors
```

```
;-------------------------------------------------------------------------------------------------
                .sect   ".reset"                        ; MSP430 RESET Vector
                .short  RESET
                .end
```

Appendix 2: Lab_4_Q2.asm

```
;-------------------------------------------------------------------------------------------------
; File: Lab_4_Q2.asm
; Description: Performs addition and subtraction of single digit base 10 numbers
; Constraints: This program can only accept the integers [0-9] as inputs, as well as
;                         only perform the operations addition and subtraction.
;                         The scope could be easily expanded to include multiplication and
division.
; Input: Hard coded string containing a mathematical expression. The input must not contain
spaces.
; Output: P2OUT contians the hex result of the mathematical expression
; Author: David Thornton
; Lab Section: 2
; Date: September 22, 2020
;-------------------------------------------------------------------------------------------------
                .cdecls C,LIST,"msp430.h"               ; Include device header file
;-------------------------------------------------------------------------------------------------
                .def    RESET                                   ; Export program entry-point to
make it known to linker
                .data                                           ; Declare variable in a data
segment
myStr:  .cstring "9+9-6"
;-------------------------------------------------------------------------------------------------
                .text                                           ; Assemble into program
memory
                .retain                                         ; Override ELF conditional
linking and retain current section
                .retainrefs                                     ; Retain any sections that
have references to current section
;-------------------------------------------------------------------------------------------------
RESET:  mov.w   #__STACK_END, SP                ; Initialize stack pointer
                mov.w   #WDTPW|WDTHOLD, &WDTCTL   ; Stop watchdog timer
;-------------------------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------------------------
main:   mov.w   #myStr, R4                      ; Load the starting address of the string
into R4
                clr.b   R5                                      ; R5 is the running total
                clr.b   R6                                      ; R6 is a temp register
```

```
gnext:  mov.b   @R4+, R6                ; Get the next character, put it into R6,
and increment R4
                cmp.b   #0, R6                  ; Check to see if R6 is the NULL
character
                jeq     lend                    ; If yes, go to the "lend" subroutine
                cmp.b   #'+', R6                ; Check to see if R6 is the '+'
character
                jeq     add_only                ; If yes, go to the "add_only"
subroutine
                cmp.b   #'-', R6                ; Check to see if R6 is the '-'
character
                jeq     sub_only                ; If yes, go to the "sub_only"
subroutine
                mov.b   R6, R5                          ; Puts the first character of
the string into R5,
                                                        ; a positive integer
is expected.
                                                        ; The program only
reaches this statement once,
                                                        ; because
otherwise a subroutine is called.
                jmp     gnext                   ; Go to the next character

add_only
                mov.b   @R4+, R6                ; Get a new character, put it into
R6, and increment R4
                add     R6, R5                  ; Perform addition and store result
in R5
                sub     #48, R5                 ; This is necessary to keep the
"running total" correct
                jmp     gnext                   ; Go to the next character

sub_only
                mov.b   @R4+, R6                ; Get a new character, put it into
R6, and increment R4
                sub     R6, R5                  ; Perform subtraction and store
result in R5
                add     #48, R5                         ; This is necessary to keep
the "running total" correct
                jmp     gnext                   ; Go to the next character

lend:   sub     #48, R5                 ; Convert from ASCII to decimal value
                mov.b   R5, &P2OUT              ; Write result to P2OUT (not visible
on port pins)
                nop                                     ; Required only for
debugger
```

```
;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack
;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET
            .end
```

Appendix 3: Lab_4_Q3.asm

```
;-------------------------------------------------------------------------------
; File: Lab_4_Q3.asm
; Description: Capitalizes all letters in a string
; Input: Hard coded string
; Output: Upper case string in the same memory location as input
; Author: David Thornton
; Lab Section: 2
; Date: September 22, 2020
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file
;-------------------------------------------------------------------------------
            .def    RESET                           ; Export program entry-point to
make it known to linker
            .data                                   ; Declare variable in a data
segment
myStr:  .cstring "I enjoy learning msp430"
;-------------------------------------------------------------------------------
            .text                                   ; Assemble into program
memory
            .retain                                 ; Override ELF conditional
linking and retain current section
            .retainrefs                             ; Retain any sections that
have references to current section
;-------------------------------------------------------------------------------
RESET:  mov.w   #__STACK_END, SP                ; Initialize stack pointer
            mov.w   #WDTPW|WDTHOLD, &WDTCTL   ; Stop watchdog timer
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
main:   mov.w   #myStr, R4                      ; Load the starting address of the string
```

```
into R4

gnext:  mov.b   @R4+, R6                ; Get the next character, put it into R6,
and increment R4
                cmp.b   #0, R6                  ; Check to see if R6 is the NULL
character
                jeq     lend                    ; If yes, go to the "lend" subroutine
                cmp.b   #97, R6                         ; Check to see if R6 is
upper case
                jc      to_upper        ; If yes, go to the "to_upper" subroutine
                jmp     gnext                   ; Go to the next character

to_upper
                cmp.b   #123, R6                ; Check if the character is within
the lower case values
                jc      gnext                   ; If R6 is outside the range, go to
the "gnext" subroutine
                sub     #32, R6                 ; If R6 is inside the range [97-122],
subtract 32
                mov.b   R6, -1(R4)              ; Move the new (upper case)
character (R6) to the
                                                        ; previous position
in R4 (original string).
                                                        ; We use -1 as the
index since R4 was already incremented.
                jmp     gnext                   ; Go to the next character

lend:   nop                                             ; Required only for
debugger

;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
                .global __STACK_END
                .sect   .stack
;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
                .sect   ".reset"                ; MSP430 RESET Vector
                .short  RESET
                .end
```