CPE 325: Intro to Embedded Computer System

Lab04

Assembly, MSP430 ISA, .asm files

Submitted by: Nolan Anderson

Date of Experiment: 09/20/2020

Report Deadline: 09/22/2020

Introduction

This lab is an introduction to using the MSP430 instruction set architecture in Code Composer Studio by using .asm files and writing in ARM Assembly. It covers how to use counters, loops, and use data from strings. It also covers how to locate your registers and variables in memory to make sure that the values calculated in the program match a would-be sample solution.

Theory

Assembler Directeives: Assembler directives supply data to the program and control the assembly process. There are several things they allow you to do (source is from Texas instruments):

- 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

For number 1, I used two assembler directives: .data (assembles data) and .cstring (initializes one or more text strings). Assembler directives are not only useful, but necessary for getting your code to work. There are several assembler directives that are auto generated when you make a new .asm project as well such as .text and .retain.

Addressing modes: Addressing modes are very important to the MSP430 and allow you to perform different 1 line operations on registers, making code more readable and shorter. Here are the addressing modes from Texas Instruments for the MSP430:

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

a. An example for indirect addressing with auto increment can be seen in my solution for #1. I auto increment the register that contains the .cstring I declared in the .data section of my directives. Incrementing this register gives me the next value in the string so that I can do my comparisons and operations on them.

   Here is my example:

```
38 next    mov.b       @R4+, R6            ; R6 gets the next character in R4
39         cmp.b       #0, R6              ; compare R6 value to NULL character
```

Results & Observation

Copy the question from the assignment here:

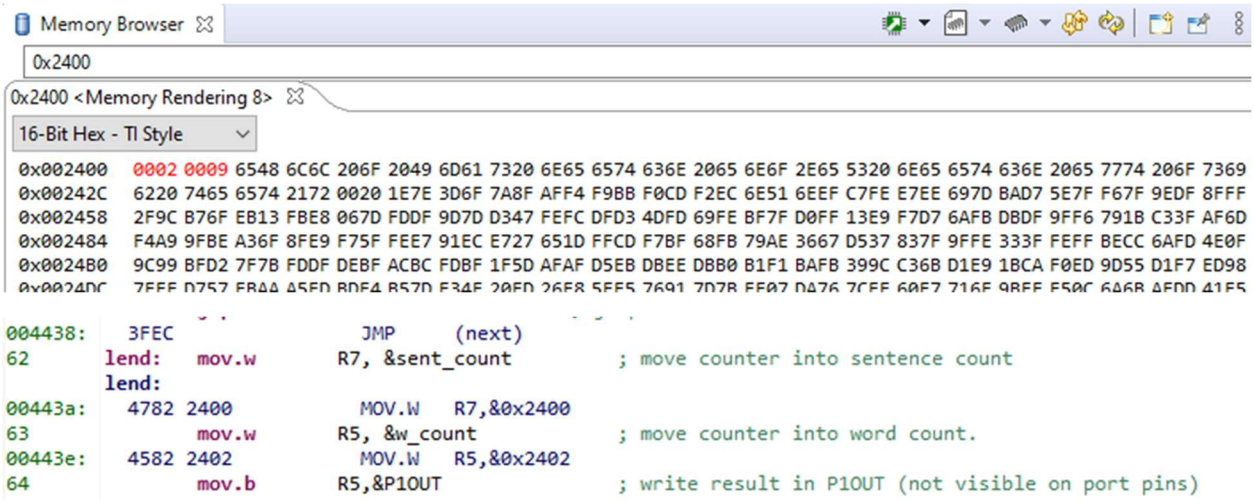
Please make sure that you have addressed following questions in your demonstration:

1. **Describe briefly how you solve Q1.**

First, you need to clear your counter registers for the counters and declare 2 final locations for the counters to reside, w_count and sent_count under .data, and your string variable. Then, in an incrementing for loop that goes through the string, check for a space, or either a '!', '.', or '?'. If it is a space, go to subroutine to add to counter. If it is a punctuation, go to sentence subroutine and add to the counter. If it is a NULL character, go to the end subroutine and store the counters: mov.w [register], &w_count and mov.w [register], &sent_count

2. **In your memory browser window, show where the values are stored for Q1.**

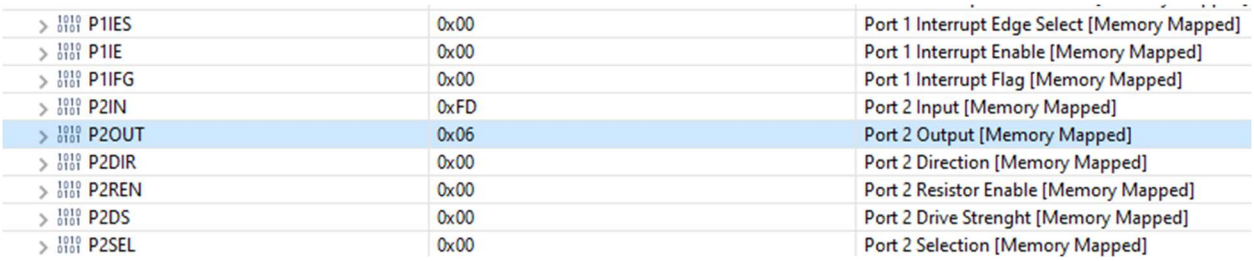
Using string: `"Hello I am sentence one. Sentence two is better! "`




```
004438:   3FEC              JMP     (next)
62      lend:   mov.w       R7, &sent_count         ; move counter into sentence count
        lend:
00443a:   4782 2400           MOV.W   R7,&0x2400
63              mov.w       R5, &w_count            ; move counter into word count.
00443e:   4582 2402           MOV.W   R5,&0x2402
64              mov.b       R5,&P1OUT               ; write result in P1OUT (not visible on port pins)
```

3. **In the registers window, show the value of P2OUT at the end of Q2.**

Using string: "4-3+5" which is equal to 6. To get 6 instead of the hex value, I just subtracted 48 to get it from ASCII to decimal.



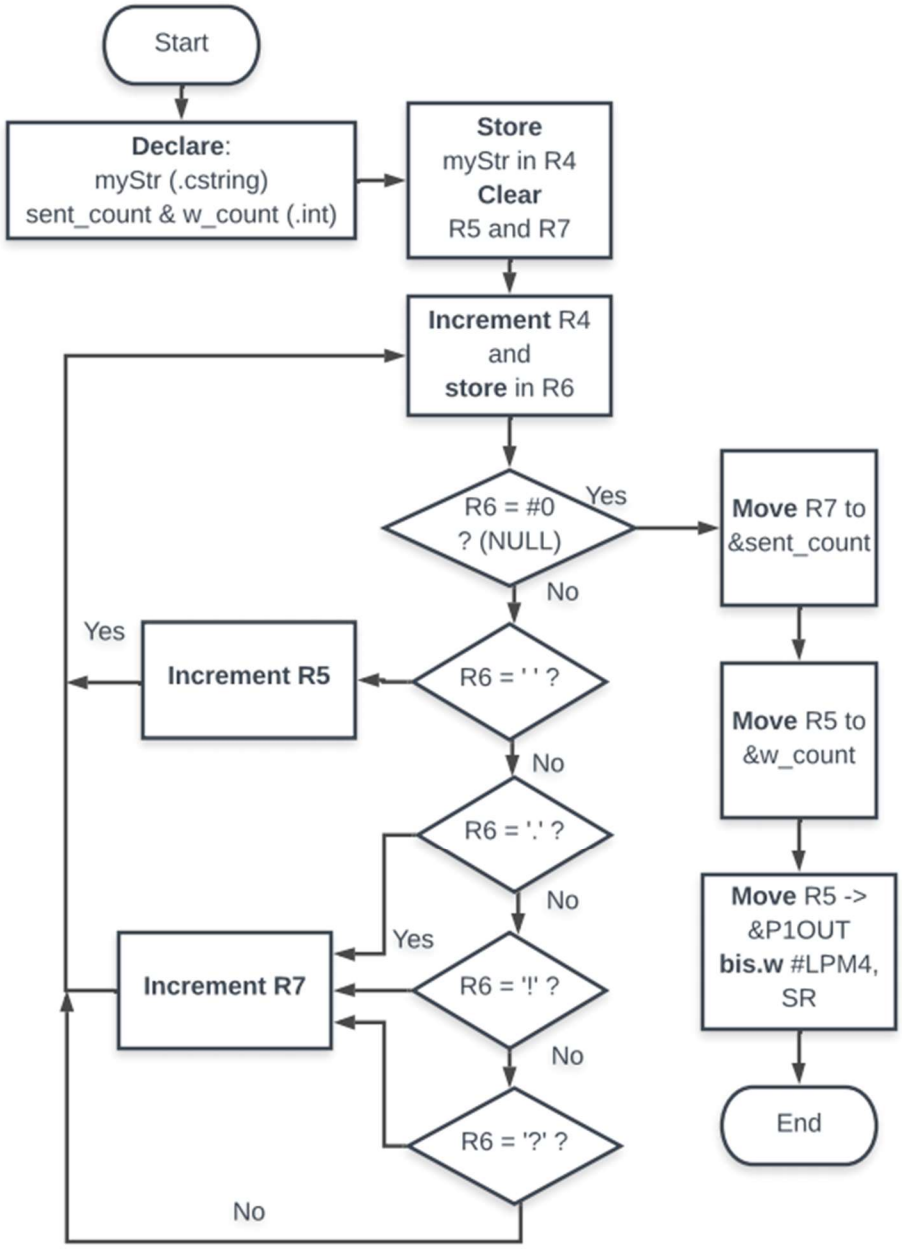
P1IES	0x00	Port 1 Interrupt Edge Select [Memory Mapped]
P1IE	0x00	Port 1 Interrupt Enable [Memory Mapped]
P1IFG	0x00	Port 1 Interrupt Flag [Memory Mapped]
P2IN	0xFD	Port 2 Input [Memory Mapped]
P2OUT	0x06	Port 2 Output [Memory Mapped]
P2DIR	0x00	Port 2 Direction [Memory Mapped]
P2REN	0x00	Port 2 Resistor Enable [Memory Mapped]
P2DS	0x00	Port 2 Drive Strenght [Memory Mapped]
P2SEL	0x00	Port 2 Selection [Memory Mapped]

4. What is register indirect addressing with auto increment? Do you use it anywhere in your code? How and Where?

This is a special case of indirect register mode in which you increment the register every time you pass through this operation, and it contains the address of the operand. I use it in #1 by auto incrementing the address of the current string character that I am on. I then store this value into another Register so I can compare and do my operations.

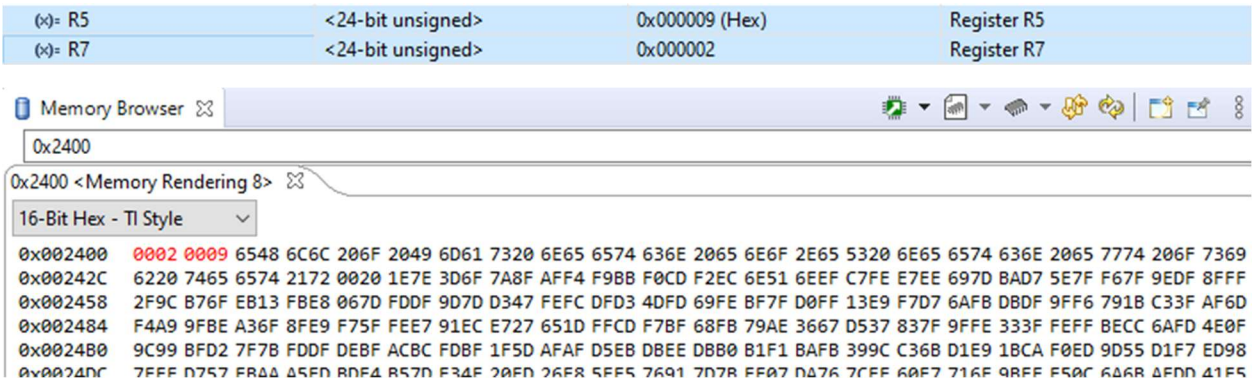
Flow Charts:

#1



Results Screenshots/Pictures:

#1



(x)= R5	<24-bit unsigned>	0x000009 (Hex)	Register R5
(x)= R7	<24-bit unsigned>	0x000002	Register R7

Memory Browser

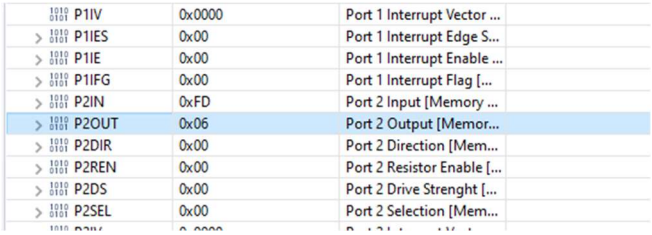
0x2400

0x2400 <Memory Rendering 8>

16-Bit Hex - TI Style

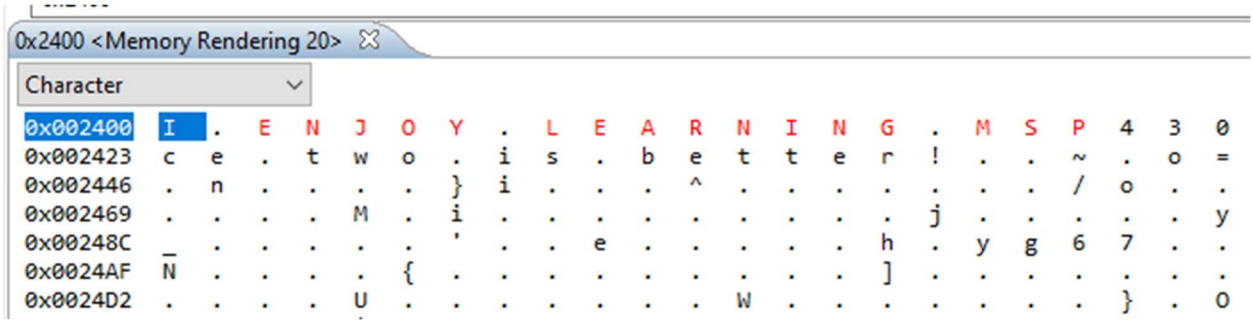
```
0x002400  0002 0009 6548 6C6C 206F 2049 6D61 7320 6E65 6574 636E 2065 6E6F 2E65 5320 6E65 6574 636E 2065 7774 206F 7369
0x00242C  6220 7465 6574 2172 0020 1E7E 3D6F 7A8F AFF4 F9BB F0CD F2EC 6E51 6EEF C7FE E7EE 697D BAD7 5E7F F67F 9EDF 8FFF
0x002458  2F9C B76F EB13 FBE8 067D FDDF 9D7D D347 FEFC DFD3 4DFD 69FE BF7F D0FF 13E9 F7D7 6AFB DBDF 9FF6 791B C33F AF6D
0x002484  F4A9 9FBE A36F 8FE9 F75F FEE7 91EC E727 651D FFCD F7BF 68FB 79AE 3667 D537 837F 9FFE 333F FEFF BECC 6AFD 4E0F
0x0024B0  9C99 BFD2 7F7B FDDF DEBF ACBC FDBF 1F5D AFAF D5EB DBEE DBB0 B1F1 BAFB 399C C36B D1E9 1BCA F0ED 9D55 D1F7 ED98
0x0024DC  7EEE D757 EBAA A5ED BDE4 B57D E34E 20ED 26E8 5EE5 7691 7D7B EE07 DA76 7CEE 60E7 716E 9BEE E50C 6A6B AED0 41E5
```

#2



P1IV	0x0000	Port 1 Interrupt Vector ...
P1IES	0x00	Port 1 Interrupt Edge S...
P1IE	0x00	Port 1 Interrupt Enable ...
P1IFG	0x00	Port 1 Interrupt Flag [...
P2IN	0xFD	Port 2 Input [Memory ...
P2OUT	0x06	Port 2 Output [Memor...
P2DIR	0x00	Port 2 Direction [Mem...
P2REN	0x00	Port 2 Resistor Enable [...
P2DS	0x00	Port 2 Drive Strenght [...
P2SEL	0x00	Port 2 Selection [Mem...

#3



0x2400 <Memory Rendering 20>

Character

```
0x002400  I  .  E  N  J  O  Y  .  L  E  A  R  N  I  N  G  .  M  S  P  4  3  0
0x002423  c  e  .  t  w  o  .  i  s  .  b  e  t  t  e  r  !  .  .  ~  .  o  =
0x002446  .  n  .  .  .  .  }  i  .  .  .  ^  .  .  .  .  .  .  .  /  o  .  .
0x002469  .  .  .  .  M  .  i  .  .  .  .  .  .  .  .  .  j  .  .  .  .  .  y
0x00248C  _  .  .  .  .  .  '  .  .  e  .  .  .  .  .  h  .  y  g  6  7  .  .
0x0024AF  N  .  .  .  .  {  .  .  .  .  .  .  .  .  .  ]  .  .  .  .  .  .  .
0x0024D2  .  .  .  .  U  .  .  .  .  .  .  .  W  .  .  .  .  .  .  .  }  .  0
```

Observations:

ARM is a lot less straight forward than typical code, but it is nice to see how a computer works on a fundamental level. I honestly really enjoyed figuring out how to work with addresses, ISA etc. It makes coding a lot more interesting, even if it takes a lot longer to do a simple task.

Conclusion

In this lab I learned how to use .asm files on the MSP430, got better with addressing modes, I understand the ISA more now, and learned what the different auto-generated lines mean on the .asm files. This was one thing I struggled with and was a little hard for me to understand what they all meant at first. Looking into the comments and documentation from TI a little more, it makes more sense now.

Link to Video:

https://drive.google.com/file/d/1Akuap4YaCvGSbyV3L1IoqhD_Z8eGywhA/view?usp=sharing

Link to Folder:

https://drive.google.com/drive/folders/1_Y3ABMDhCUxc9phtQ8JKHCM8LDOK7k7J?usp=sharing

Appendix 1

```
;-------------------------------------------------------------------------------
; File       : main.asm (CPE 325 Lab4 Q1 Code)
; Function   : Counts the number of words and sentences in a string
; Description: Program traverses an input array of characters
;              to detect number of words and sentences
; Input      : The input string is specified in myStr
; Output     : The port P1OUT displays the number of E's in the string
; Author     : N. Anderson  npa0002@uah.edu
; Date       : September 19, 2020
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
            .data
sent_count:          .int        0       ;
w_count:             .int        0       ;
myStr:      .cstring "Hello I am sentence one. Sentence two is better!"
            ; string variable shown above.
;-------------------------------------------------------------------------------
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.
;-------------------------------------------------------------------------------
RESET:       mov.w   #__STACK_END,SP        ; Initialize stackpointer
mov.w        #WDTPW|WDTHOLD,&WDTCTL         ; Stop watchdog timer
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
main:
        mov.w       #myStr, R4          ; move string into R4
        clr.b       R5                  ; clear R5 for counter
        clr.b       R7                  ; clear R7 for counter
next    mov.b       @R4+, R6             ; R6 gets the next character in R4
        cmp.b       #0, R6              ; compare R6 value to NULL character
        jeq         lend                ; if it is a NULL jump to end.
        cmp.b       #' ', R6            ; compare R6 to a space, this is not working correctly.
        jeq          word                ; incrememnt word counter
        cmp.b       #'.', R6            ; compare R6 to period
        jeq         sent                ; jump to count if yes
        cmp.b       #'?', R6            ; compare R6 to '?'
        jeq         sent                ; jump to count if yes
        cmp.b       #'!', R6            ; comapre R6 to '!'
        jeq         sent                ; jump to count if yes
        jne         next                ; if not go back to next.
word    inc.w       R5                   ; increment word counter
        jmp          next                ; jump to next.
sent    inc.w       R7                  ; incrememnt sentence character
        jmp         next                ; jump to next
lend:   inc.w       R5                   ; increment the word counter since some strings do not end in a space.
        mov.w       R7, &sent_count      ; move counter into sentence count
        mov.w       R5, &w_count         ; move counter into word count.
        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

```
;-------------------------------------------------------------------------------
; File       : main.asm (CPE 325 Lab4 Q2 Code)
; Function   : Runs a mathematical expression from a string
; Description: This program reads a string of numbers and operators
;              and performs the operation.
; Input      : The input string is specified in myStr
; Output     : The port P2OUT displays the result of the operation
; Author     : N. Anderson  npa0002@uah.edu
; Date       : September 19, 2020
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file
;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
            .data
myStr:      .cstring "4-3+5"                            ; string character
;-------------------------------------------------------------------------------
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.
;-------------------------------------------------------------------------------
RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
main:
        mov.w           #myStr, R4          ; move string into R4
        clr.b           R5                  ; clear R5 for final number
qnext:
        mov.b           @R4+, R6            ; R6 gets the next character
        cmp.b           #0, R6              ; compare R6 value to NULL character
        jeq             lend                ; if it is a NULL jump to end.
        cmp.b           #'+', R6            ; compare R6 to +
        jeq             aop                 ; if yes, jump to aop to add
        cmp.b           #'-', R6            ; compare R6 to -
        jeq             sop                 ; if yes, jump to sop to sub
        mov.b           R6, R5              ; If no, store R6 in R5. This will essentially
                                            ; get the first number in the operation to start.
        jmp             qnext               ; jump to get next character (should be operation)
aop     mov.b           @R4+, R7            ; increment the counter and move it to R7
        add             R7, R5              ; R5 <- R5 + R7
        jmp             qnext               ; Get next character.
sop     mov.b           @R4+, R7            ; Increment the counter and move it to R6
        sub             R7, R5              ; R5 <- R5 + (not) R7 + 1
        jmp             qnext               ; Get the next character.
lend:   sub             #48, R5             ; R5 <- R5 + (not) 48 + 1
        mov.b           R5, &P2OUT          ; write result in P2OUT (not visible on port pins)
        bis.w           #LPM4,SR            ; LPM4
        nop                                 ; required only for Debugger
;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack
;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET
```

Appendix 3

```
;-------------------------------------------------------------------------------
; File       : main.asm (CPE 325 Lab4 Q2 Code)
; Function   : Updates the value of the lowercase letters to upper case
; Description: This program reads a string and converts
;                              the lower case to upper case
; Input      : The input string is specified in myStr
; Output     : None, just updated register values.
; Author     : N. Anderson  npa0002@uah.edu
; Date       : September 19, 2020
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file
;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
            .data
myStr:      .cstring "I enjoy learning msp430"                          ; string character
;-------------------------------------------------------------------------------
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.
;-------------------------------------------------------------------------------
RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
main:
        mov.w           #myStr, R4          ; move string into R4
        clr.b           R5                  ; Pointer to the R6 address
qnext:  mov             R4, R5              ; move R4 into R5 for the value.
        mov.b           @R4+, R6            ; R6 gets the next character
        cmp.b           #0, R6              ; compare R6 value to NULL character
        jeq             lend                ; if it is a NULL jump to end.
        cmp.b           #97, R6             ; compare 97 to R6
        jc              upper               ; switch to upper case if it is greater than 97
        jmp             qnext               ; jump back to qnext
upper
        cmp.b           #123, R6            ; compare to 123
        jc              qnext              ; if it is greater than jump to qnext
        sub             #32, R6            ; update the address' value
        mov.b           R6, 0(R5)          ; R5+0 <- R6
        jmp             qnext              ; jump to qnext for the next value.
lend:   nop                                 ; required only for Debugger
;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack
;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET
```