CPE 325: Intro to Embedded Computer System

**Lab04**

**Assembly, MSP430 ISA, .asm files**

**Submitted by**: Nolan Anderson

**Date of Experiment**: 09/20/2020

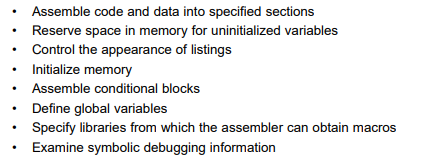
**Report Deadline**: 09/22/2020

# 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.

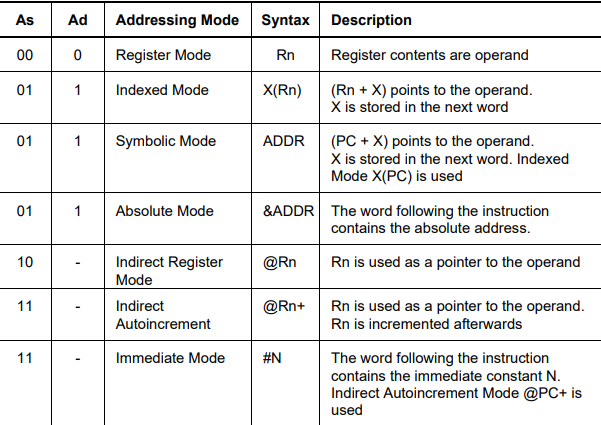
Theory

**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):



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:



1. 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: 

# Results & Observation

Copy the question from the assignment here:

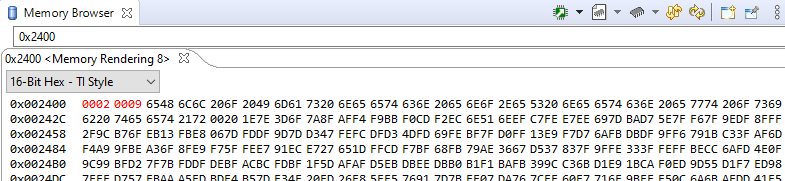
Please make sure that you have addressed following questions in your demonstration:

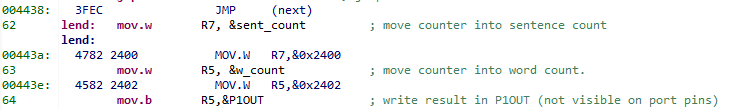
1. **Describe briefly how you solve Q1.**

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

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

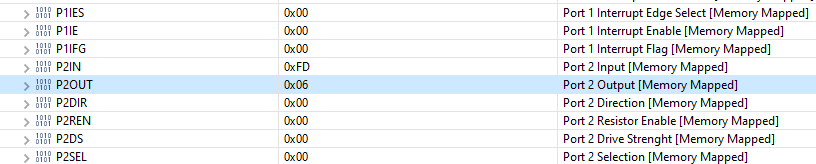
Using string: "Hello I am sentence one. Sentence two is better! "





1. **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.

****

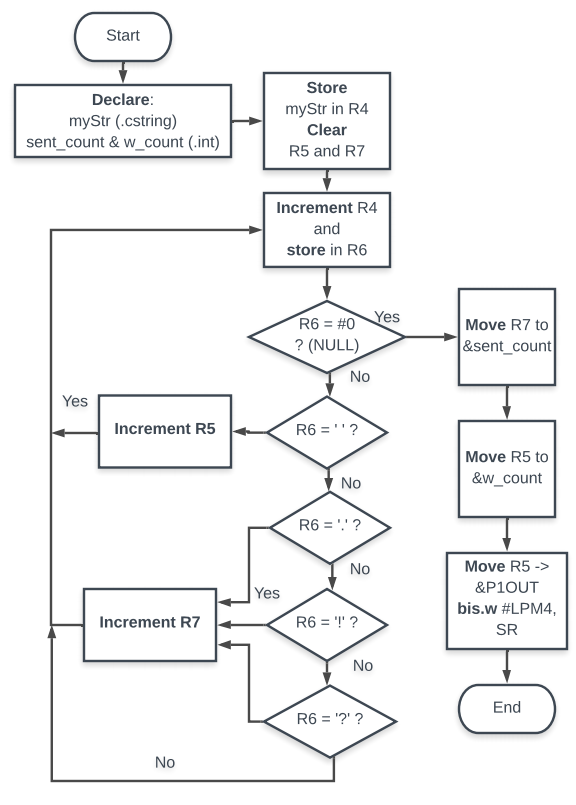
**4. What is register indirect addressing with auto increment? Do you use it anywhere in your**

**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:

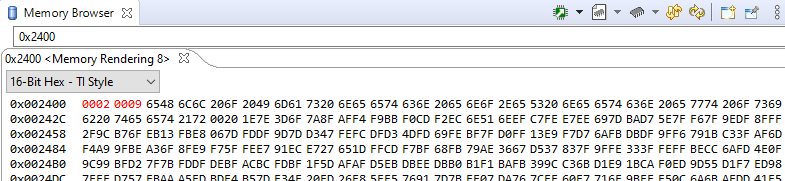
#1



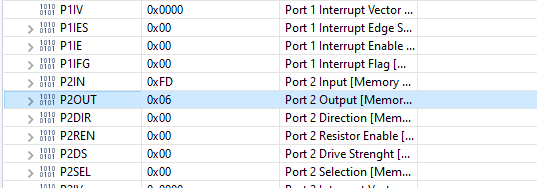
## Results Screenshots/Pictures:

#1

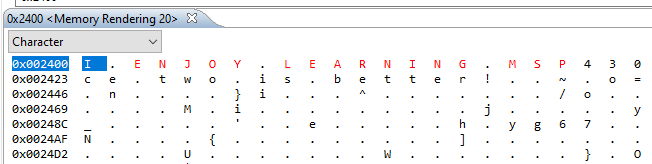




#2



#3



## 

## 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 |