CPE 325: Intro to Embedded Computer System

Lab04

Introduction to MSP430 Assembly Language Programming

Submitted by: Esther Shore

Date of Experiment: June 7, 2023

Report Deadline: June 13, 2023

Demonstration Deadline: June 19, 2023

**Introduction**

This lab introduces MSP430 assembly programming using assembler directives and different addressing modes. It also utilizes the register and memory browser windows in Code Composer Studio. The programs involve iterating through an input string, reading and writing to and from memory.

**Theory Topics**

1. Assembler Directives

Assembler directives are instructions for the assembler that commonly defines memory allocation or sets entry points for the program. In MSP430 assembly, assembler directives begin with a period followed by a keyword. For example, .data indicates the start of the data section where variables are initialized in memory, allowing read/write capabilities.

1. Different Addressing Modes - assembly format and description of:
   1. Register

Example format: R1

Directly accesses contents of specified register.

* 1. Indexed

Example format: 0(R2)

Accesses the value in memory at the address with a byte offset indicated by the integer prefix.

* 1. Symbolic

Example format: label

Accesses the value in memory at the address specified by symbolic label defined.

* 1. Absolute

Example format: &label

Uses the absolute address specified by some symbolic label defined.

* 1. Indirect

Example format: @R3

Accesses value in memory at the address contained in the specified register.

* 1. Immediate

Example format: #0x02

Uses literal value specified after the # symbol.

* 1. Indirect with autoincrement - also provide a case where you would use this mode

Example format: @R4+

Accesses value in memory at the address contained in the specified register and then increments the value in the register by the operand size.

This address mode is often used to iterate through each character in a sequence of elements. For instance, it was used in Program 1 to increment through the characters in the input string.

**Program 1 (count words & sentences)**

***Program Description:***

This program is intended to count the number of words and sentences in a given string and write the counts to memory. The design is to iterate through the characters in the string and increment the appropriate counter based on the character detected. If it is a space, then the word counter is incremented. If it is a period, exclamation point, or question mark, then the sentence counter is incremented. Finally, once the end of the string is reached, the word counter is incremented one final time to account for the last word of the string.

***Program Output:***

Input:



Input in Memory:



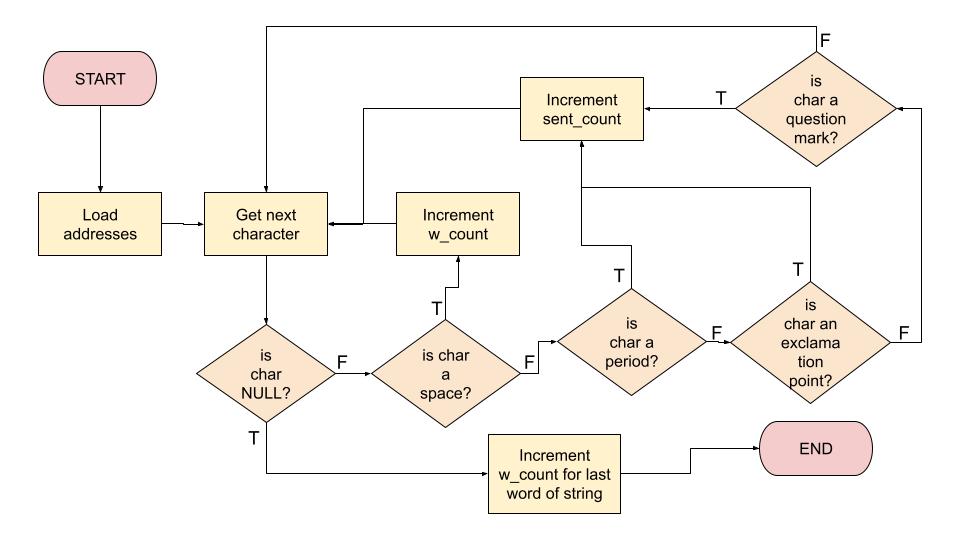
The string is terminated by the NULL character indicated by 00.

Output in Memory:



The updated words in memory are displayed in red above, where the first represents the number of sentences and the second represents the number of words.

***Program Flowchart:***



**Figure 2:** Program 1 Flowchart

**Program 2 (string arithmetic)**

***Program Description:***

This program takes an input string with single digit numbers as well as addition and/or subtraction operators, computes the results and outputs it in P2OUT of the register window. To accomplish this, a cumulative result register is set and the input string is incremented through. For each operation set, there is a function for add and subtract and then the result is updated with the new value after the operation.

***Program Output:***

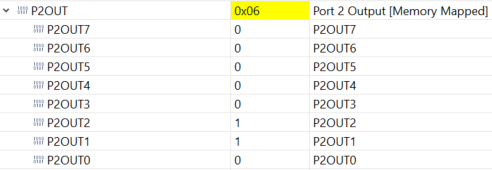
Input:



Input in Memory:



Output in Port:



**Bonus**

***Program Description:***

This program takes an input string in memory and updates the characters in memory to an uppercase letter if the existing character was a lowercase letter. This is accomplished by incrementing through the string and comparing the ASCII hex values to determine if it is a lowercase letter and then write to the corresponding byte with the appropriate uppercase letter. The little endian representation of bytes in a word had to be noted in order to write to the proper byte.

***Program Output:***

Input:



Input in Memory:



The string begins at address 0x2400. The end of the string is indicated by the NULL character 00.

Output in Memory:



The string begins at address 0x2400. The changed words are indicated in red, the end of the string is indicated by the NULL character 00.

Output Converted from Hex to ASCII:

"I ENJOY LEARNING MSP430"

**Conclusion**

No major issues for the first two programs. For the bonus program, the issue that arose was writing to the incorrect byte when updating the string. The solution to this problem was adding a -1 byte offset when writing the updated character to account for the little endian representation in memory. Learned a lot about MSP430 assembly programming and using different addressing modes in this lab, notably reading and writing from memory and writing to output registers.

***Appendix:***

**Table 1:** Program 1 Source Code

| ; ------------------------------------------------------------------------------  ; File: Lab04\_P1.asm  ; Function: Counts the number of words and sentences in a string variable  ; Description: Program traverses an input array of characters  ; to detect specified characters that indicate end of word or  ; end of sentence; exits when a NULL is detected  ; Input: The input string specified in myStr  ; Output: sent\_count and w\_count written in memory  ; Author(s): Esther Shore  ; Date: June 13, 2023  ; ------------------------------------------------------------------------------  .cdecls C, LIST, "msp430.h" ; Include device header file  ;-------------------------------------------------------------------------------  .def RESET ; Export program entry-point to  ; make it known to linker.  ;-------------------------------------------------------------------------------  myStr: .cstring "Sentence one. Sentence two? Sentence three!"  ;-------------------------------------------------------------------------------  ; Data R/W Variables  ;-------------------------------------------------------------------------------  .data  sent\_count: .int 0  w\_count: .int 0  ;-------------------------------------------------------------------------------  .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 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  mov.w #sent\_count, R5 ; Load address of sent\_count into R5  mov.w #w\_count, R6 ; Load address of w\_count into R6  loop: mov.b @R4+, R7 ; Get a new character  cmp.b #0, R7 ; Is it a null character  jeq endl ; If yes, jump to endl function  cmp.b #' ', R7 ; Is it a ' ' character  jeq word ; If yes, jump to word function  cmp.b #'.', R7 ; Is it a '.' character  jeq sent ; If yes, jump to sent function  cmp.b #'!', R7 ; Is it an '!' character  jeq sent ; If yes, jump to sent function  cmp.b #'?', R7 ; Is it a '?' character  jeq sent ; If yes, jump to sent function  jmp loop ; Go to the next character  word: inc.w 0(R6) ; w\_count++ w/ zero offset indirect addressing  jmp loop ; Go to the next character  sent: inc.w 0(R5) ; Increment sentence counter  jmp loop ; Go to the next character  endl: inc.w 0(R6) ; Increment word counter to account for last word  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  .end |
| --- |

**Table 2:** Program 2 Source Code

| ; ------------------------------------------------------------------------------  ; File: Lab04\_P2.asm  ; Function: Evaluates simple mathematical expression string.  ; Description: Program traverses an input array of characters to detect the  ; numbers and the operations; exits when a NULL is detected  ; Input: The input string specified in myStr  ; Output: The port P2OUT displays the result of the expression  ; Author(s): Esther Shore  ; Date: June 13, 2023  ; ------------------------------------------------------------------------------  .cdecls C, LIST, "msp430.h" ; Include device header file  ;-------------------------------------------------------------------------------  .def RESET ; Export program entry-point to  ; make it known to linker.  ;-------------------------------------------------------------------------------  myStr: .cstring "4-3+5"  ;-------------------------------------------------------------------------------  .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 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.w R5 ; Clear R5 to represent result  loop: mov.b @R4+, R7 ; Get a new character  cmp.b #0, R7 ; Is it a null character  jeq endl ; If yes, jump to endl function  cmp.b #'+', R7 ; Is it a '.' character  jeq addf ; If yes, jump to sent function  cmp.b #'-', R7 ; Is it an '!' character  jeq subf ; If yes, jump to sent function  sub.w #'0', R7 ; Convert ASCII to numeric value  add.w R7, R5 ; Otherwise, add R7 to R5 result  jmp loop ; Go to the next character  addf: mov.b @R4+, R7 ; Get next character  sub.w #'0', R7 ; Convert ASCII to num  add.w R7, R5 ; R5 = R5 + R7  jmp loop ; Go to the next character  subf: mov.b @R4+, R7 ; Get next character  sub.w #'0', R7 ; Convert ASCII to num  sub.w R7, R5 ; R5 = R5 - R7  jmp loop ; Go to the next character  endl: mov.b R5, &P2OUT ; Write result in P2OUT  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  .end |
| --- |

**Table 3:** Bonus Source Code

| ; ------------------------------------------------------------------------------  ; File: Lab04\_P3.asm  ; Function: Converts lowercase to uppercase characters in a string.  ; Description: Program traverses an input array of characters  ; to detect lowercase characters and update them in memory to their  ; uppercase counterparts; exits when a NULL is detected  ; Input: The input string specified in myString  ; Output: Updated myString written in memory  ; Author(s): Esther Shore  ; Date: June 13, 2023  ; ------------------------------------------------------------------------------  .cdecls C, LIST, "msp430.h" ; Include device header file  ;-------------------------------------------------------------------------------  .def RESET ; Export program entry-point to  ; make it known to linker.  ;-------------------------------------------------------------------------------  ; Data R/W Variables  ;-------------------------------------------------------------------------------  .data  myString: .cstring "I enjoy learning msp430"  ;-------------------------------------------------------------------------------  .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 stack pointer  mov.w #WDTPW|WDTHOLD,&WDTCTL ; Stop watchdog timer  ;-------------------------------------------------------------------------------  ; Main loop here  ;-------------------------------------------------------------------------------  main: mov.w #myString, R4 ; Load the starting address of the string into R4  loop: mov.b @R4+, R5 ; Get a new character  cmp.b #0, R5 ; Is it a null character  jeq endl ; If yes, jump to endl function  cmp.b #'a', R5 ; Compare character to 'a'  jl loop ; If less than, get next character  cmp.b #'z'+1, R5 ; Compare character to char after 'z' in ASCII  jge loop ; If greater than or equal to, go to next character  sub.b #'a'-'A', R5 ; Convert character from lower to uppercase ASCII  mov.b R5, -1(R4) ; Write new character to address w/ -1 byte offset  jmp loop ; Go to the next character  endl: 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  .end |
| --- |