CPE 325: Intro to Embedded Computer System

Lab04

MSP430 Assembly Language Programming

Submitted by: Hannah Kelley

Date of Experiment: September 17, 2025

Report Deadline: September 24, 2025

Demonstration Deadline: September 24, 2025

Theory

Topic 1: Assembler Directives

a) In MSP430 assembly programming, assembler directives are essential for organizing and controlling how code and data are handled during assembly. They do not produce executable instructions but instead instruct the assembler in tasks such as memory allocation, symbol definition, program structuring, and linking with other modules. Some examples of assembler directives include .bss, .space, .text, and .sect.

Topic 2: Different Addressing Modes

a) Format and Description of Each Mode
    a. Register
        i. MOV Rx, Ry
        ii. Register direct addressing uses values located directly in the CPU registers. This is the fastest mode since no memory access is needed.
    b. Indexed
        i. MOV 20(Rx), Ry
        ii. The effective address of the indexed operand is obtained by adding the constant value specified outside of the parentheses to the contents of the register inside the parentheses, also called the base address. This mode can be used for accessing arrays where the register holds the base address and the constant is the offset.
    c. Symbolic
        i. MOV LABEL, R6
        ii. Symbolic is a type of indexing address mode where the PC is used as the base address. This mode allows the use of variables and labels in code.
    d. Absolute
        i. MOV &0x0200, R6
        ii. Absolute addressing forces the assembler to treat the value as an absolute address. This mode is useful when working with pointers and arrays. It is also a special type of indexed addressing where the full 16-bit address is specified.
    e. Indirect
        i. MOV @Rx, Ry
        ii. Indirect addressing uses the contents of the specified register as a pointer to a location in memory. This mode is useful for dereferencing pointers.
    f. Immediate
        i. MOV #20, Ry
        ii. Immediate addressing uses the given value as a constant instead of a memory address. The value is embedded in the instruction.
b) Example of Indirect Addressing with Autoincrement
    a. MOV @R10+, R6
    b. Indirect Addressing with Autoincrement uses the value in the source register as a pointer to memory. The data in memory is moved to the destination register and the source register is automatically incremented (by 1 if using .b or by 2 if using .w) to point

to the next location in memory. This mode is extremely useful for stepping through arrays or strings in memory.

Results & Observation

Program 1:

Program Description:

This program processes an input string by removing all special characters while preserving letters, numbers, and spaces. The cleaned string is stored as a new output string. During execution, the program counts the number of deleted special characters and stores the result in the P3OUT register, while the number of spaces are counted and stored in the P4OUT register.

Program Output:

0x002400	W	e	l	c	o	m	e	.	T	o	.	M	S	P	4	3
0x002410	0	.	A	s	s	e	m	b	l	y	.	L	a	n	g	u
0x002420	a	g	e	.	.	.	.	.	.	.	^	C	.	:	.	.
0x002430	=	.	.	D	.	.	.	.	.	.	.	.	.	.	{	.
0x002440	.	.	.	.	.	.	r	.	;	.	.	!	#	.	.	.
0x002450	.	W	w	.	s	.	.	m	.	.	.	.	.	.	V	.
0x002460	/	]	J	.	|	.	.	.	W	e	l	c	o	m	e	.
0x002470	T	o	.	M	S	P	/	4	3	0	.	A	s	s	*	e
0x002480	m	b	l	y	.	L	a	n	g	u	a	g	e	:	)	!
0x002490	.	.	.	.	.	.	.	7	.	.	.	.	.	.	.	.

Figure 01: Program 1 Memory Output

P3OUT	0x05
P3DIR	0xFF
P3REN	0x00
P3DS	0x00
P3SEL	0x00
P4IN	0x24
P4OUT	0x04
P4DIR	0xFF
P4REN	0x00
P4DS	0x00
P4SEL	0x00

Figure 02: Program 1 Register Output

Program Flowchart:

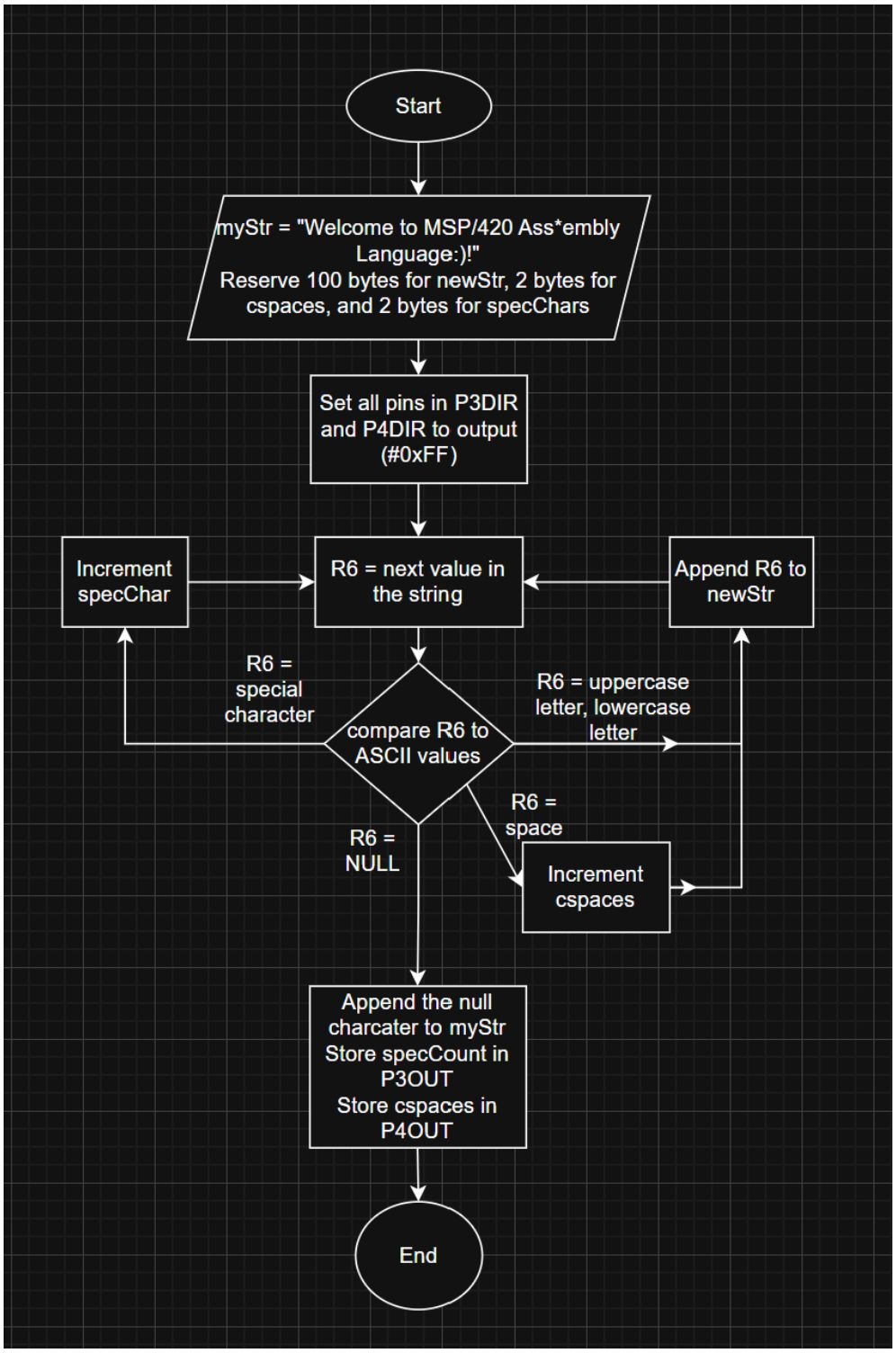

Figure 03: Program 1 Flowchart

Program 2:

Program Description:

This program updates an input string by converting each uppercase letter to its corresponding lowercase letter and each lowercase letter to its corresponding uppercase letter. Non-alphabetic characters remain unchanged. The program also counts the number of case changes: conversions from lowercase to uppercase are stored in the P3OUT register, and conversions from uppercase to lowercase are stored in the P4OUT register.

Program Output:

0x0023FF	w	w	e	l	c	o	m	e	.	T	o	.	M	s	P
0x00240E	4	3	0	.	a	s	s	e	m	b	l	y	.	l	A
0x00241D	n	G	u	A	g	E	!	.	.	.	.	.	.	^	C
0x00242C	.	:	.	.	=	.	.	D	.	.	.	.	.	.	.
0x00243B	.	.	.	{	.	.	.	.	.	.	.	r	.	;	.
0x00244A	.	!	#	.	.	.	.	W	w	.	s	.	.	m	.
0x002459	.	.	.	.	.	V	.	/	]	J	.	|	.	.	.
0x002468	W	E	L	C	O	M	E	.	t	O	.	m	S	p	4
0x002477	3	0	.	A	S	S	E	M	B	L	Y	.	L	a	N
0x002486	g	U	a	G	e	!	.	:	)	!	.	.	.	.	.

Figure 04: Program 2 Memory Output

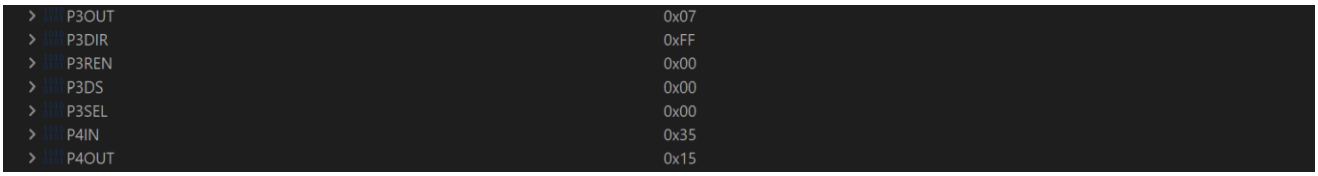

Figure 05: Program 2 Register Output

Program Flowchart:

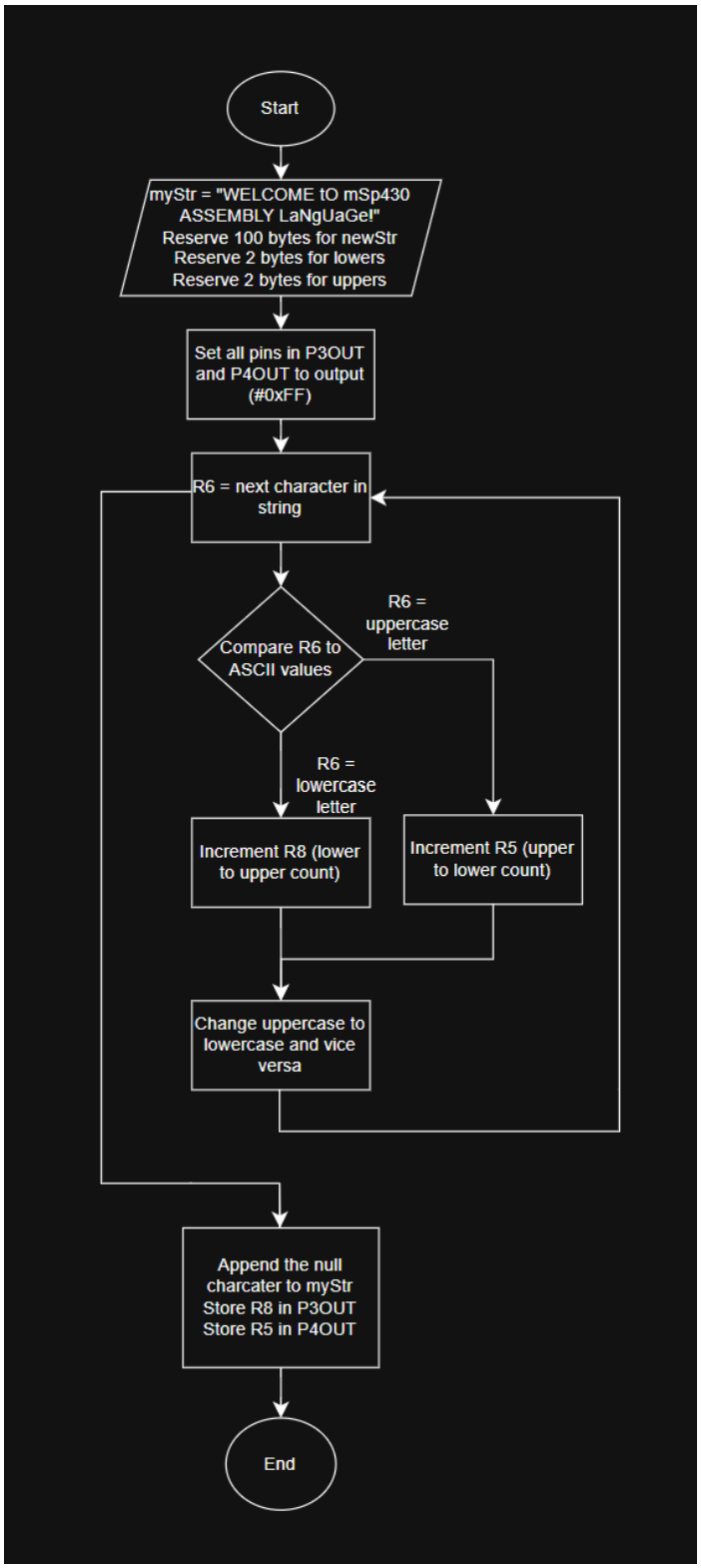

Figure 06: Program 2 Flowchart

Appendix

Table 01: Program 1 Source Code

```
;-------------------------------------------------------------------------------
; File:        main.asm
; Description: Create a new string by deleting special characters from an
original string.
;              Counts the number of spaces and special characters.
; Board:       MSP430
; Input:       myStr
; Output:      newStr, with counts output to P3OUT and P4OUT.
; Author:      Hannah Kelley
; Date:        September 17, 2025
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file
;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
            .data
myStr:      .cstring "Welcome To MSP/430 Ass*embly Language:)!"
             .bss newStr, 100, 1
             .bss cspaces, 2, 1
             .bss specChars, 2, 1
            .text
;-------------------------------------------------------------------------------
            .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

; Configure P3 and P4 as output ports
            bis.b   #0xFF, &P3DIR           ; Set all P3 pins to output
direction
            bis.b   #0xFF, &P4DIR           ; Set all P4 pins to output
direction
; Main loop here-------------------------------------------------------------
main:
            mov.w #myStr, R4   ; R4 = pointer to starting address of myStr
            mov.w #newStr, R7  ; R7 = pointer to starting address of newStr
            clr.w R5           ; R5 = counter for special characters removed
            clr.w R8            ; R8 = counter for spaces

gnext:      mov.b @R4+, R6      ; R6 = next value in the string
            cmp #0, R6          ; Check if it's the null terminator
            jeq end             ; if equal, jump to end

            ; Check for spaces (and keep them)
            cmp #' ', R6        ; Compare R6 with ASCII space
            jeq spaces          ; If it's a space, jump to spaces handling

            ; Check for numbers
            cmp #'0', R6        ; Compare R6 with ASCII '0'
```

```
            jge numcheck        ; Jump if it's a number or greater

            ; If not a space or number, it's a special character (less than
'0')
            jmp delete

spaces:     inc.w R8          ; Increment spaces counter
            jmp keep

numcheck:   cmp #':', R6  ; Check if between numbers and
uppercase letters
            jl keep          ; If less, it's a number, so keep it

            cmp #'A', R6     ; Check if it's an uppercase letter
            jge uppercheck   ; If greater or equal, check uppercase range
            jl delete    ; If less, it's a special character between 9 and A

uppercheck: cmp #'[', R6     ; Check for end of uppercase range
            jl keep          ; If less, it's an uppercase letter, so keep it

            cmp #'a', R6     ; Check for lowercase letters
            jge lowercheck   ; If greater or equal, check lowercase range
            jl delete ; It's a special character between upper and lowercase

lowercheck: cmp #'{', R6     ; Check for end of lowercase range
            jl keep          ; If less, it's a lowercase letter, so keep it
            jge delete       ; If greater or equal, it's a special character

keep:       mov.b R6, 0(R7)   ; Write character at the write pointer
            inc.w R7          ; Increment write pointer
            jmp gnext         ; Continue the process

delete:     inc.w R5          ; Increment special character counter
            jmp gnext         ; Go to the next character

end:        mov.b #0, 0(R7)   ; Append the null character to the new string
            mov.b R5, &P3OUT  ; Store special char count in P3OUT
            mov.b R8, &P4OUT  ; Store space count in P4OUT
done:       jmp done          ; Halt the program

            nop
; Stack Pointer definition-------------------------------------------------
            .global __STACK_END
            .sect   .stack

; Interrupt Vectors--------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET
```

Table 02: Program 2 Source Code

```
;-------------------------------------------------------------------------------
; File:        main.asm
; Description: Create a new string by updating the case of letters from an
;              original string. Converts uppercase to lowercase and vice
versa.
;              Counts the number of each type of change.
; Board:       MSP430
; Input:       myStr
; Output:      newStr, with counts output to P3OUT and P4OUT.
; Author:      Hannah Kelley
; Date:        September 17, 2025
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"         ; Include device header file

;-------------------------------------------------------------------------------
            .def    RESET                 ; Export program entry-point to
                                          ; make it known to linker.
            .data
myStr:      .cstring "WELCOME tO mSp430 ASSEMBLY LaNgUaGe!"
            .bss newStr, 100, 1
            .bss lowers, 2, 1
            .bss uppers, 2, 1
            .text
;-------------------------------------------------------------------------------
            .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

; Configure P3 and P4 as output ports
            bis.b  #0xFF, &P3DIR    ; Set all P3 pins to output direction
            bis.b  #0xFF, &P4DIR    ; Set all P4 pins to output direction

; Main loop here-------------------------------------------------------------
main:
            mov.w  #myStr, R4   ; R4 = pointer, starting addr of myStr (r)
            mov.w  #newStr, R7  ; R7 = pointer, starting addr of newStr (w)
            clr.b  R5     ; R5 = counter for Upper to Lower changes (P4OUT)
            clr.b  R8     ; R8 = counter for Lower to Upper changes (P3OUT)

gnext:      mov.b  @R4+, R6     ; R6 = next value in the string
            cmp.b  #0, R6       ; Check if it's the null terminator
            jeq    end          ; if equal, jump to end

            ; Check for uppercase letters ('A' through 'Z')
            cmp.b  #'A', R6     ; Is character >= 'A'?
            jl     check_lower  ; No, check if it's a lowercase letter
            cmp.b  #'[', R6     ; Is character >= '['?
            jge    check_lower  ; Yes, so it's not a letter, check lowercase
```

```
            ; It's an uppercase letter, so convert to lowercase
            xor.b  #0x20, R6 ; Flip the 5th bit (bit 5) to toggle case
            add.b  #1,R5   ; Increment the Upper to Lower counter
            jmp    keep    ; Now copy the converted character to newStr

check_lower:
            ; Check for lowercase letters ('a' through 'z')
            cmp.b  #'a', R6  ; Is character >= 'a'?
            jl     keep      ; No, it's a non-letter character, just keep it
            cmp.b  #'{', R6  ; Is character >= '{'?
            jge    keep  ; Yes, so it's a non-letter character, just keep it

            ; It's a lowercase letter, so convert to uppercase
            xor.b  #0x20, R6   ; Flip the 5th bit to toggle case
            add.b  #1, R8      ; Increment the Lower to Upper counter
            jmp keep

keep:       mov.b  R6, 0(R7)   ; Write the character (original or modified)
            add.b  #1, R7      ; Increment write pointer
            jmp    gnext       ; Continue the process

end:        mov.b  #0, 0(R7)   ; Append the null character to the new string
            mov.b  R8, &P3OUT         ; Store Lower to Upper count in P3OUT
            mov.b  R5, &P4OUT         ; Store Upper to Lower count in P4OUT
done:       jmp    done               ; Halt the program

            nop
; Stack Pointer definition-------------------------------------------------
            .global __STACK_END
            .sect   .stack

; Interrupt Vectors--------------------------------------------------------
            .sect   ".reset"                  ; MSP430 RESET Vector
            .short  RESET
```