CPE 325: Intro to Embedded Computer System

Lab04

MSP430 ASSEMBLY PROGRAMMING.

Submitted by: Dan Otieno.

Date of Experiment: 02/03/22.

Report Deadline: 02/07/22.

Demonstration Deadline: 02/10/22.

Introduction

This lab introduces assembly programming in MSP430. The report contains three different sections, each involving hard-coded strings that I later use assembly language to define various instructions for. There will be a total of 3 programs in this lab, each explained in more detail below.

Theory

Topic 1: **ASSEMBLER DIRECTIVES:**

a) In Assembly programming, directives tell the assembler to set the data and program at specific addresses, allocate space in memory for variables, allocate space in memory and initialize constants, define synonyms, or include additional files. The MSP430 assembler is referred to as ASM430 and in it, uninitialized data is assembled into the .bss section, initialized data into the .data section, and executable code into the .text section. We shall see .data used in the program codes on this lab. The table below shows some of the directives we find in the MSP430 family.

Table 1. Sections and section directives in ASM430 and A430.

Description	ASM430 (CCS)	A430 (IAR)
Reserve size bytes in the uninitialized sect.	.bss	-
Assemble into the initialized data section	.data	RSEG const
Assemble into a named initialized data section	.sect	RSEG
Assemble into the executable code	.text	RSEG code
Reserve space in a named (uninitialized) section	.usect	-
Align on byte boundary	.align 1	-
Align on word boundary	.align 2	EVEN

Figure 1: ASM430 Assembly directives.

Topic 2: **ADDRESSING MODES:**

The MSP430 comprises 7 addressing modes to specify a source operand in any location in memory. the first 4 of the seven can be used to specify the source/destination operand. Addressing modes are encoded using As (2-bit long) and Ad (1-bit long) address specifiers in the instruction word. Some addressing modes are discussed in detail below:

a) REGISTER MODE: This is the fastest and shortest mode to specify operands. Address specifiers are A(src) = 00 and A(dst) = 0. The register number, which is 4 bits, is specified in the address field.

b) INDEXED MODE: The operand for this mode is determined by the sum of specified address register and displacement X, where X can be represented by a decimal or hex value specified in the next instruction word. Effective operand address is ea, i.e. ea= Rn+X.

c) SYMBOLIC MODE: This is similar to indexed addressing mode, except the for the fact that the address register is PC, such that ea = PC+Offset. So the current operand address is specified relative to the current PC.
d) ABSOLUTE MODE: In this addressing mode, the instruction defines the absolute address of the operand in memory, and it includes a word that specifies that address. A(src) =01, A(dst) =1.
e) INDIRECT REGISTER MODE: Only used with source operands, the instruction specifies the address register, i.e. ea = Rn.
f) INDIRECT AUTOINCREMENT: The effective address of the operand is the content of register Rn, which is incremented by +1 for byte operations and +2 for word operations.
g) IMMEDIATE: The syntax for this is #N, where N is stored in the next word, or in combination of the preceding extension word and the next word. We use indirect autoincrement mode @PC+ in this form of addressing.

As, Ad	Addressing Mode	Syntax	Description
00, 0	Register	Rn	Register contents are operand.
01, 1	Indexed	X(Rn)	(Rn + X) points to the operand. X is stored in the next word, or stored in combination of the preceding extension word and the next word.
01, 1	Symbolic	ADDR	(PC + X) points to the operand. X is stored in the next word, or stored in combination of the preceding extension word and the next word. Indexed mode X(PC) is used.
01, 1	Absolute	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word, or stored in combination of the preceding extension word and the next word. Indexed mode X(SR) is used.
10, –	Indirect Register	@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 by 1 for .B instructions, by 2 for .W instructions, and by 4 for .A instructions.
11, –	Immediate	#N	N is stored in the next word, or stored in combination of the preceding extension word and the next word. Indirect autoincrement mode @PC+ is used.

Figure 2: Register Addressing Modes.

Results & Observation

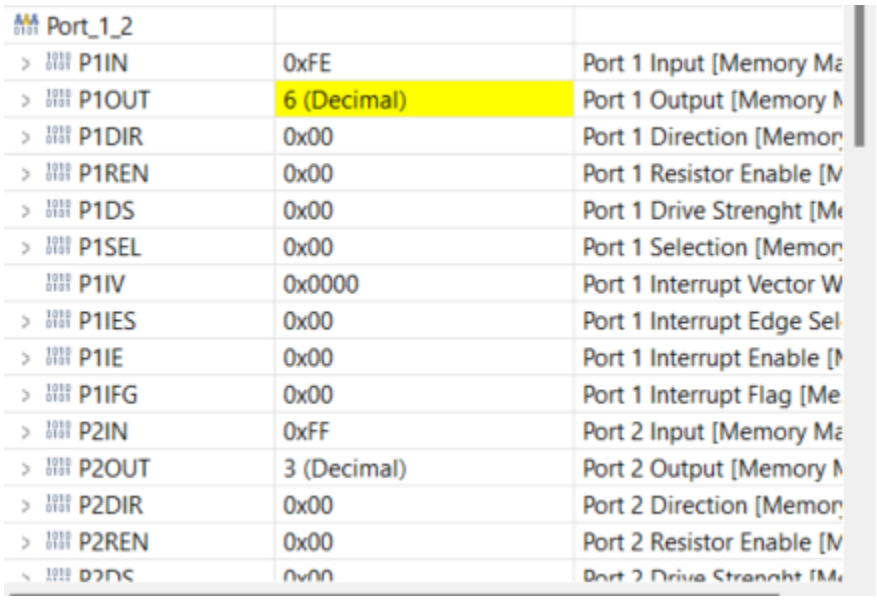
Program 1:

Program Description:

Explain your approach in solving the problem.

This problem required counting the number of words and sentences in a hard-coded string. My approach was to read each character and then determine whether they were a '.', '!' or '?', indicating a full sentence, if the program encountered these characters, the sentence counter would be incremented. I also declared a counter for the words, and that increments when a word is encountered. Finally, the results for both totals are sent into P1OUT and P2OUT.

Program Output:



Port_1_2		
P1IN	0xFE	Port 1 Input [Memory Ma
P1OUT	6 (Decimal)	Port 1 Output [Memory M
P1DIR	0x00	Port 1 Direction [Memor
P1REN	0x00	Port 1 Resistor Enable [M
P1DS	0x00	Port 1 Drive Strenght [Me
P1SEL	0x00	Port 1 Selection [Memor
P1IV	0x0000	Port 1 Interrupt Vector W
P1IES	0x00	Port 1 Interrupt Edge Sel
P1IE	0x00	Port 1 Interrupt Enable [M
P1IFG	0x00	Port 1 Interrupt Flag [Me
P2IN	0xFF	Port 2 Input [Memory Ma
P2OUT	3 (Decimal)	Port 2 Output [Memory M
P2DIR	0x00	Port 2 Direction [Memor
P2REN	0x00	Port 2 Resistor Enable [M
P2DS	0x00	Port 2 Drive Strenght [Me

Figure 3: Output for program 1

Program Flowchart:

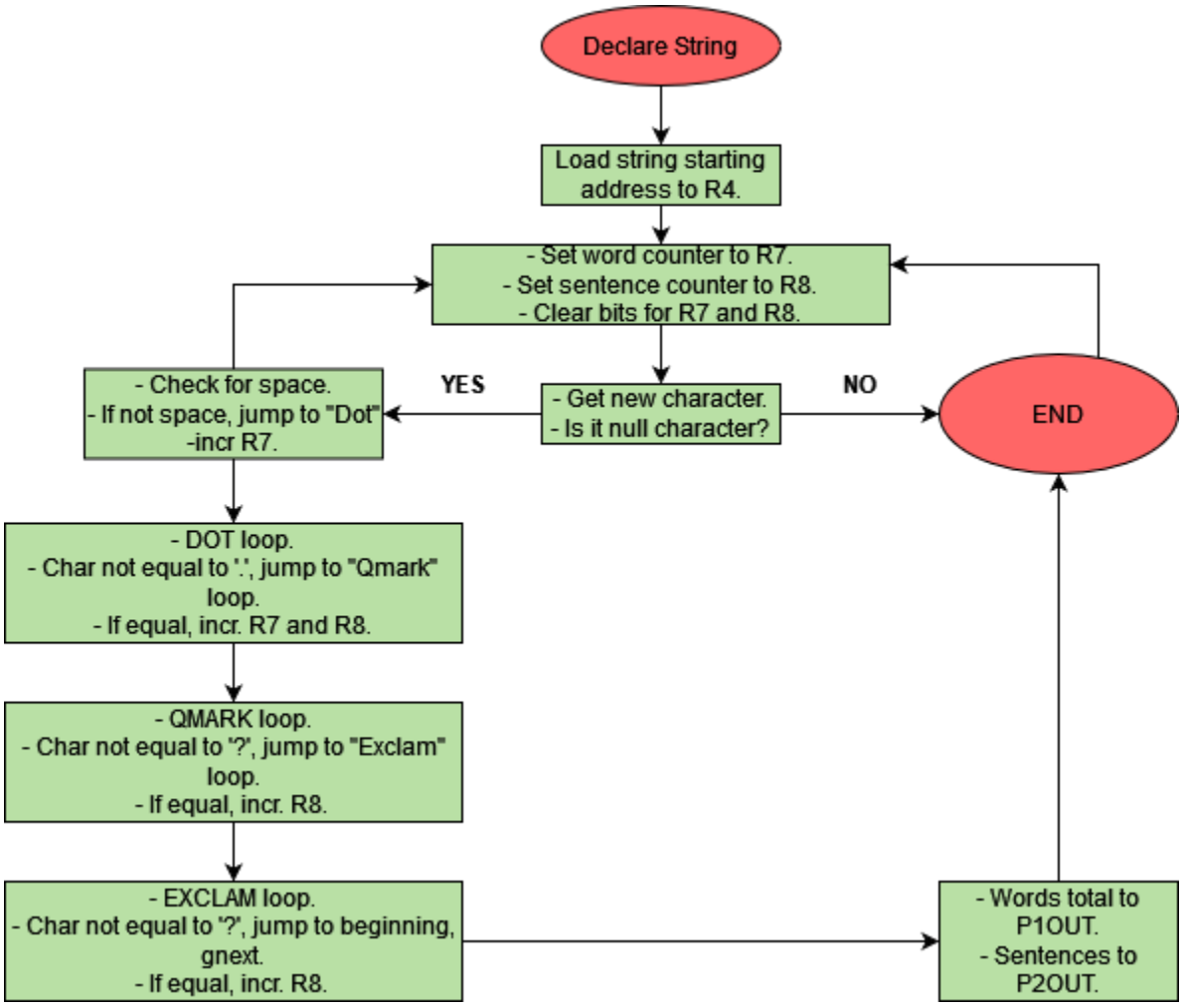


Figure 4: Flowchart for program 1

Program 2:

Program Description:

The second program required evaluating a string as a mathematical expression. Similar to the first one, the string is hard-coded, after which I assigned R5 to store the addition and subtraction results once those operations were performed. Next, we read the string, and then check for the null character first, if we encounter a null character, we jump to the end of the program and get no result, otherwise, we check every character and then, depending on whether we run into '+' or '-', we jump into loops to either add or subtract. The result is then stored in R5 and we get the output in the P2OUT register window. Because this is a string, it is also important to store the values as int datatypes. So I initialized R5 and R7 to int by using sub.b command with 0.

Program Output:

P1IFG	0x00
P2IN	0xFF
P2OUT	6 (Decimal)
P2DIR	0x00
P2REN	0x00

Figure 4: Output for Program 2.

Program Flowchart:

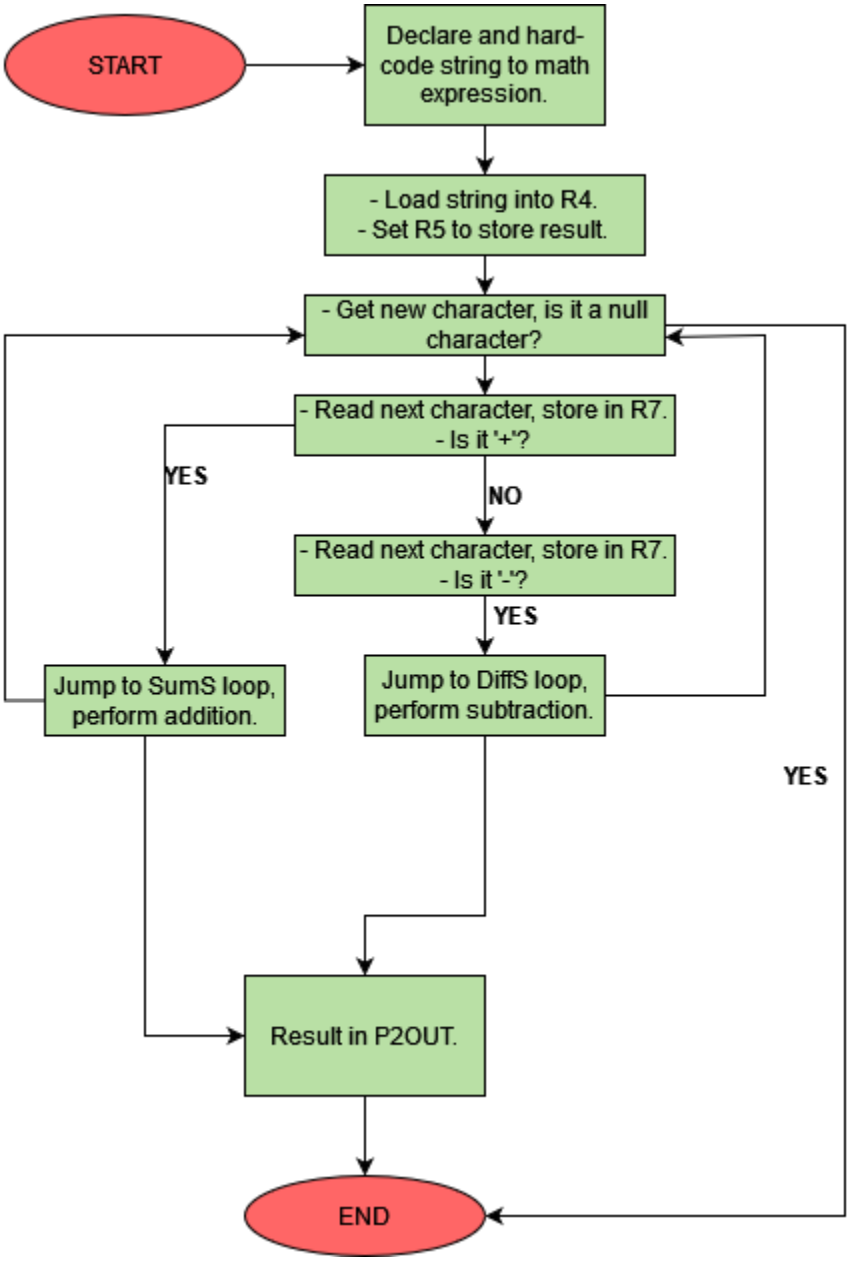


Figure 5: Program 2 Flowchart.

Program 3 (BONUS):

Program Description:

This is a bonus lab question; no flowchart is requested so one is not included with the report. Program requires reading a hard-coded string and converting lowercase letters in the string to uppercase. My approach to this was almost the same as the past two, string is read and stored in R4, then I checked for null character, and if there was no null, I used ASCII value codes to check for uppercase, i.e. checking if the character ASCII value is less than #91, which is the next character value after uppercase 'Z', if yes, then we skip it and check the next character, if the next character is lowercase, the program jumps to the loop, where we use the sub.b instruction to convert it to uppercase and store the result back into R4.

Program Output:

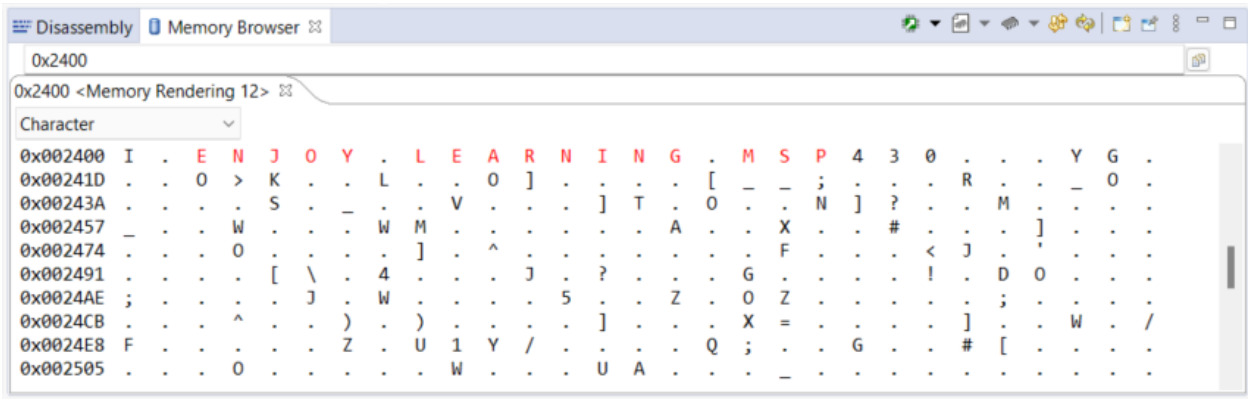


Figure 6: Program 3 output.

Conclusion

One of the tricky steps in program 1 was to determine how to avoid counting spaces when looking for full words. I managed to resolve that by adding another word counter in the first loop to check for the '.' Character. But my takeaway from this was I learned how to display output through the memory browser window and register window, as opposed to the console.

Appendix

Table 01: Program 1 source code

```
/*-------------------------------------------------------------------------
 * File:         Lab4Asm1.asm
 * Function:     This assembly code will count the words and sentences in a given string.
 * Description: This program counts and returns the number of words and sentences in a
given string
 *               which is hard-coded.
 * Input:        None.
 * Output:       Number of words, in P1OUT, number of sentences, in P2OUT.
 * Author(s):    Dan Otieno, dpo0002@uah.edu
 * Date:         February 5th, 2022.
 * ------------------------------------------------------------------------*/
;-------------------------------------------------------------------------
; MSP430 Assembler Code Template for use with TI Code Composer Studio
;
;
;-------------------------------------------------------------------------
            .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!"
;-------------------------------------------------------------------------
            .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.
                .data
w_count:    .int        0
s_count:    .int        0

;-------------------------------------------------------------------------
RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer


;-------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------
main:      ; bis.b         #OFFh, &P1DIR         ; Do not output result on port pins.
                mov.w #myStr, R4            ; Load starting address of the string into R4.
```

```
                mov.w #w_count, R7
                mov.w #s_count, R8
                clr.b R7                        ; Clear bit for Words counter.
                clr.b R8                        ; Clear bit for Sentence counter.

gnext:          mov.b @R4+, R6              ; Get a new character.
                cmp       #0,R6                 ; Check if null character.
                jeq       lend                  ; If yes, jump to the end.
                cmp.b #' ',R6                   ; Check if character is a space.
                jne       Dot                       ; If it is not, jump to loop to
check for '.'.
                inc.w R7                        ; If it is increment word count.
                jmp       gnext                 ; Go to the next character.

Dot:       cmp.b #'.',R6                    ; Check for '.' character.
                jne       Qmark                 ; If is not, check next character.
                inc.w R7                        ; Count as full word, incerement word
count.
                inc.w R8                        ; If it is, increment sentence count.
                jmp       gnext                 ; Go to the next character.

Qmark:          cmp.b #'?',R6                   ; Check for '?' character.
                jne       Exclam                    ; If not, check next character.
                inc.w R8                        ; If it is, increment sentence count.
                jmp       gnext                 ; Go to the next character.


Exclam:         cmp.b #'!',R6                   ; Check for '!' character.
                jne       gnext                 ; If not, check next character.
                inc.w R8                        ; If it is, increment sentence count.
                jmp       gnext                 ; Go to the next character.

lend:      mov.b R7,&P1OUT             ; Write word count result in P1OUT.
                mov.b R8,&P2OUT             ; Write sentence count result in P2OUT.
                bis.w #LPM4,SR              ; LPM4.
                nop                                 ; For debugger.

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

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
```

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

Table 02: Program 2 source code

```
/*-------------------------------------------------------------------------
 * File:         Lab4Asm2.asm
 * Function:     This assembly code will read a string and evaluate it as a math
expression.
 * Description: This program reads a string, determines add or subtract characters and
 *               performs the corresponding operation (addition or subtraction).
 * Input:        None.
 * Output:       Caluclation result in P2OUT.
 * Author(s):    Dan Otieno, dpo0002@uah.edu
 * Date:         February 5th, 2022.
 * ------------------------------------------------------------------------*/
;-------------------------------------------------------------------------
; MSP430 Assembler Code Template for use with TI Code Composer Studio
;
;
;-------------------------------------------------------------------------
            .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 stackpointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer

;-------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------
main:      ; bis.b    #0FFh,&P2DIR
                mov.w      #myStr, R4                   ; Load starting address of the
string into R4.
                mov.b @R4+, R5                      ; Result will be stored in R5.
```

```
                sub.b #48, R5                           ; Read R5 as int data type, set to
0.


gnext:          mov.b @R4+, R6                  ; Get a new character.
                cmp         #0, R6                          ; Check if null character.
                jeq         lend                    ; If it is, go to the end.
                mov.b @R4+, R7                  ; Get next character and store in R7.
                sub.b #48, R7                           ; Read character as integer data
type, set to 0.
                cmp         #'+', R6                    ; Check if Character is '+'.
                jeq         SumS                        ; If it is, jump to SumS function
(for addition).
                cmp         #'-', R6                    ; Check if character is '-'.
                jeq         DiffS                       ; If it is, jump to DiffS function
(for subtraction).

SumS:       add.b R7, R5                        ; Perform addition, store result in R5.
                jmp         gnext

DiffS:          sub.b R7, R5                            ; Perform subtraction, store
result in R5.
                jmp         gnext

lend:       mov.b       R5,&P2OUT               ; Output in P2OUT;
                bis.w #LPM4, SR                 ; LPM4
                nop                                     ; required for debugger

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

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

Table 3: Program 3 source code

```
/*-------------------------------------------------------------------------
 * File:         Lab04_Bonus.asm
 * Function:     This assembly code will read a string and convert lowercase charcters to
uppercase.
 * Description: This program reads and converts the lowercase characters in a hard-coded
string to
 *               uppercase. Result is displayed in memory browser window.
 * Input:        None.
 * Output:       Demo conversion from lower to upper in memory browser.
 * Author(s):    Dan Otieno, dpo0002@uah.edu
 * Date:         February 5th, 2022.
 * -------------------------------------------------------------------------*/
;-------------------------------------------------------------------------
; MSP430 Assembler Code Template for use with TI Code Composer Studio
;
;
;-------------------------------------------------------------------------
            .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"
;-------------------------------------------------------------------------
            .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:      ; bis.b   #0FFh,&P1DIR          ; do not output the result on port pins.
                mov        #myStr, R4                ; Load starting address of the
string into R4.

gnext:          mov.b @R4+, R6                  ; Get a new character.
                cmp        #0, R6                       ; Check if null character.
                jeq        lend                     ; If yes, go to the end.
```

```
                cmp.b #91, R6                           ; Use ASCII value to check if
character is uppercase.
                jl          gnext                       ; If value is less than last
uppercase character, check next character.
                jmp         low2up                          ; Jump to loop to check for
lowercase.

low2up:         cmp.b #97, R6                           ; Check if character is lowercase.
                jl          gnext                       ; If not, check next character.
                sub.b #32, R6                           ; If yes, convert lowercase to
uppercase.
                mov.b R6, -1(R4)                    ; Store result back to R4.
                jmp         gnext



lend:       mov.b       R4, &P2OUT                  ; Send output to P2OUT but display
result in memory browser.
                bis.w #LPM4, SR                     ; LPM4
                nop                                             ; required for debugger

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

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