CPE 325: Intro to Embedded Computer System

Lab04

MSP430 Assembly Language Programming

Submitted by: Anshika Sinha

Date of Experiment:__01/29/2025_______

Report Deadline:____02/04/2025_________

Demonstration Deadline: __02/10/2025____

Theory

Topic 1: Assembler Directives

a) Assembler directives in MSP430 are special instructions that guide the assembler but do not translate into machine code.

b) They help define data, allocate memory, and control the assembly process.

c) Some common directives include:

   a. .data – Defines the beginning of a data section.

   b. .text – Marks the start of the code (text) section.

   c. .global – Declares symbols to be accessible across files.

   d. .byte, .word – Allocate memory for storing byte- or word-sized data.

   e. .string – used to define and store strings with with a null character automatically appended to the end

Topic 2: Addressing Modes: Addressing modes define how operands are accessed in instructions.

a) Register: In Register Addressing, the operand is stored in a register, making it the fastest mode since it operates directly on CPU registers.
b) Indexed: determines the address of an operand by adding a base register and an offset. This is useful for accessing arrays and structured data.
c) Symbolic: uses labels to refer to memory locations, making code more readable. The assembler replaces the label with an actual memory address.
d) Absolute: A fixed memory address is used as the operand. This mode is typically used for accessing special function registers
e) Indirect: accesses an operand stored at the memory location held in a register. This is commonly used in pointer-based operations.
f) Immediate: directly specifies the operand value within the instruction itself. This mode is useful for loading constants.
g) Indirect with autoincrement: variation of indirect addressing where the register automatically increments after accessing the memory location. This is useful for iterating through arrays or loops.

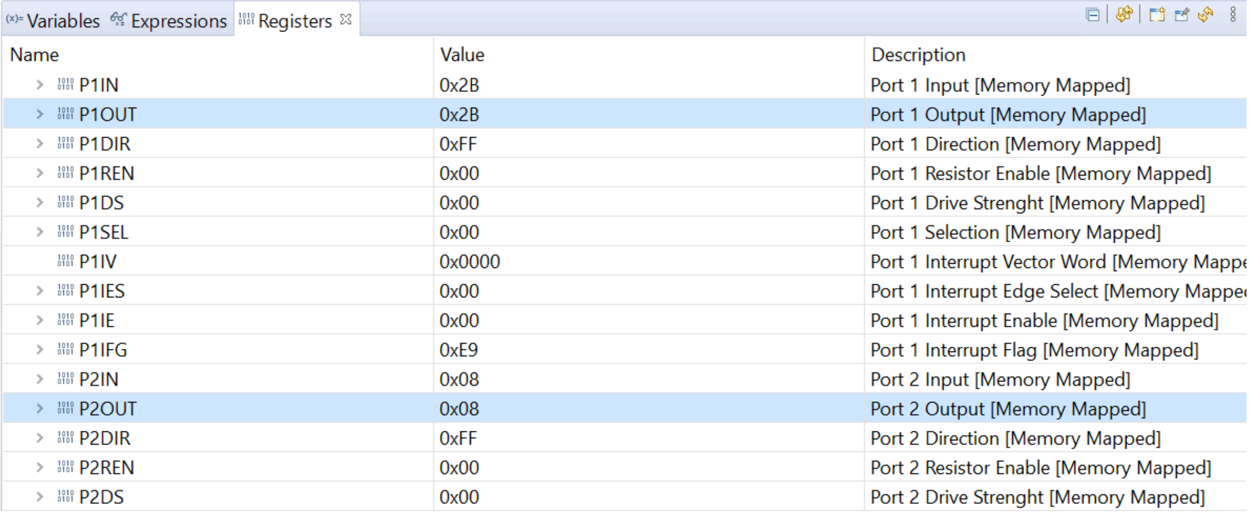
Results & Observation

Program 1:

Program Description:

This MSP430 assembly program counts the number of digits and number of characters in the given string. The string is hard-coded as “Welcome 2 the MSP430 Assembly, Spring 2025!” and is not more than one line. The number of digits and length of string are outputted on ports P2 and P1 respectively.

Program Output:



Name	Value	Description
P1IN	0x2B	Port 1 Input [Memory Mapped]
P1OUT	0x2B	Port 1 Output [Memory Mapped]
P1DIR	0xFF	Port 1 Direction [Memory Mapped]
P1REN	0x00	Port 1 Resistor Enable [Memory Mapped]
P1DS	0x00	Port 1 Drive Strenght [Memory Mapped]
P1SEL	0x00	Port 1 Selection [Memory Mapped]
P1IV	0x0000	Port 1 Interrupt Vector Word [Memory Mappe
P1IES	0x00	Port 1 Interrupt Edge Select [Memory Mappe
P1IE	0x00	Port 1 Interrupt Enable [Memory Mapped]
P1IFG	0xE9	Port 1 Interrupt Flag [Memory Mapped]
P2IN	0x08	Port 2 Input [Memory Mapped]
P2OUT	0x08	Port 2 Output [Memory Mapped]
P2DIR	0xFF	Port 2 Direction [Memory Mapped]
P2REN	0x00	Port 2 Resistor Enable [Memory Mapped]
P2DS	0x00	Port 2 Drive Strenght [Memory Mapped]

Figure 01: Program 1 Output

Program Flowchart:

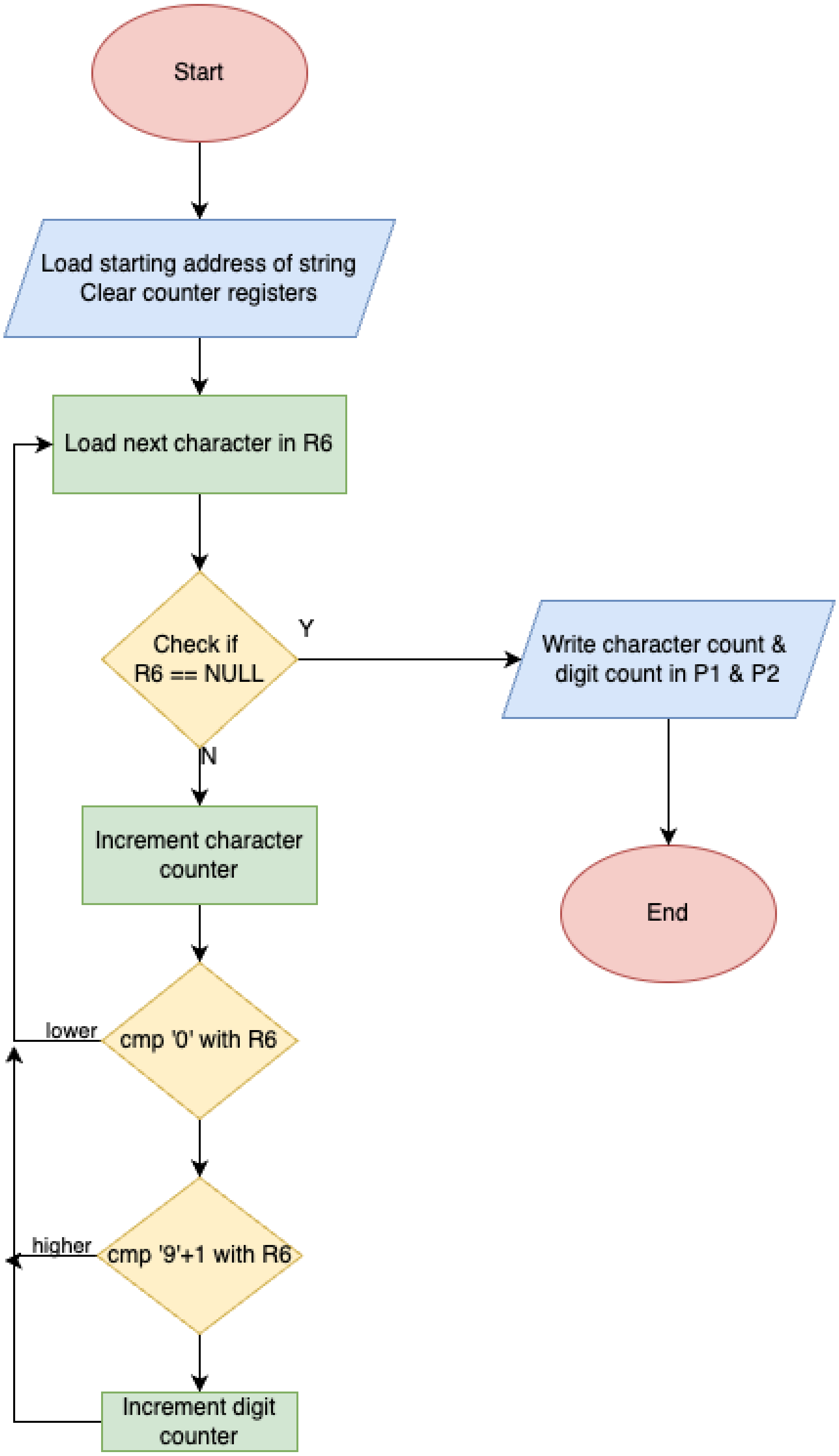


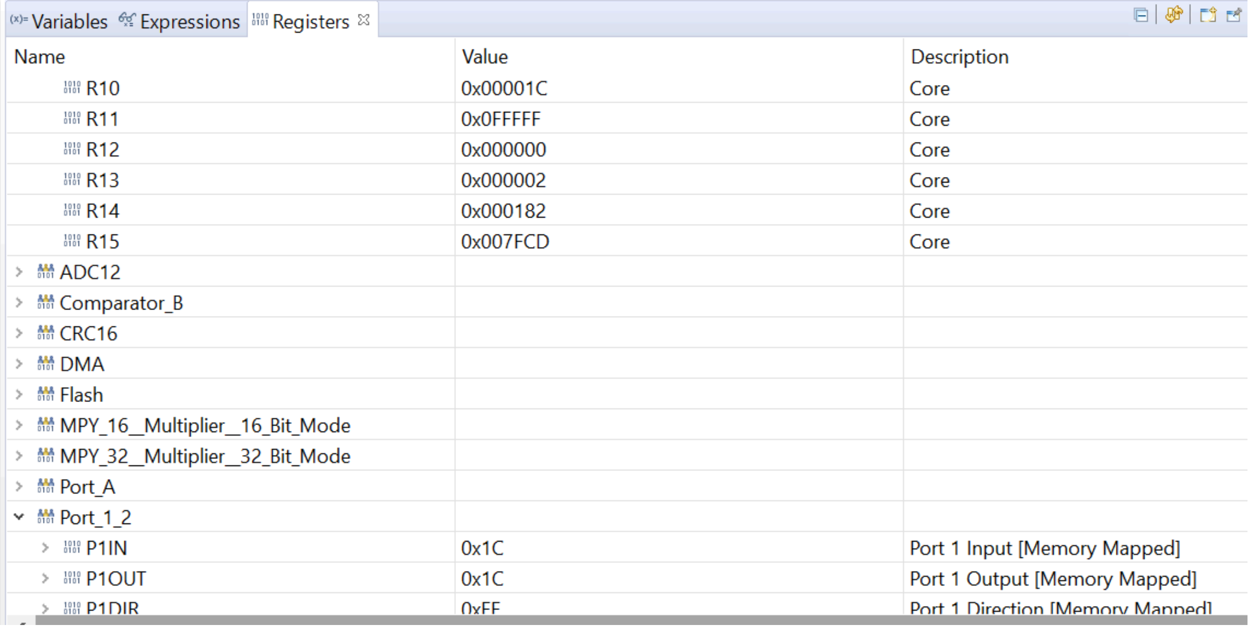
Figure 02: Program 1 Flowchart

Program 2:

Program Description:

This MSP430 assembly program performs multiplication by addition from a given string. The string must be in the format “x*y”. If it is not, 0xFF is written to the output port P1OUT. The program iterates based on the value of the smaller operand and adds the larger integer to itself in a loop. The final value is an integer value written to P1OUT.

Program Output:



Name	Value	Description
R10	0x00001C	Core
R11	0x0FFFFF	Core
R12	0x000000	Core
R13	0x000002	Core
R14	0x000182	Core
R15	0x007FCD	Core
ADC12		
Comparator_B		
CRC16		
DMA		
Flash		
MPY_16__Multiplier__16_Bit_Mode		
MPY_32__Multiplier__32_Bit_Mode		
Port_A		
Port_1_2		
P1IN	0x1C	Port 1 Input [Memory Mapped]
P1OUT	0x1C	Port 1 Output [Memory Mapped]
P1DIR	0xFF	Port 1 Direction [Memory Mapped]

Figure 03: Program 2 Output

Program Flowchart:

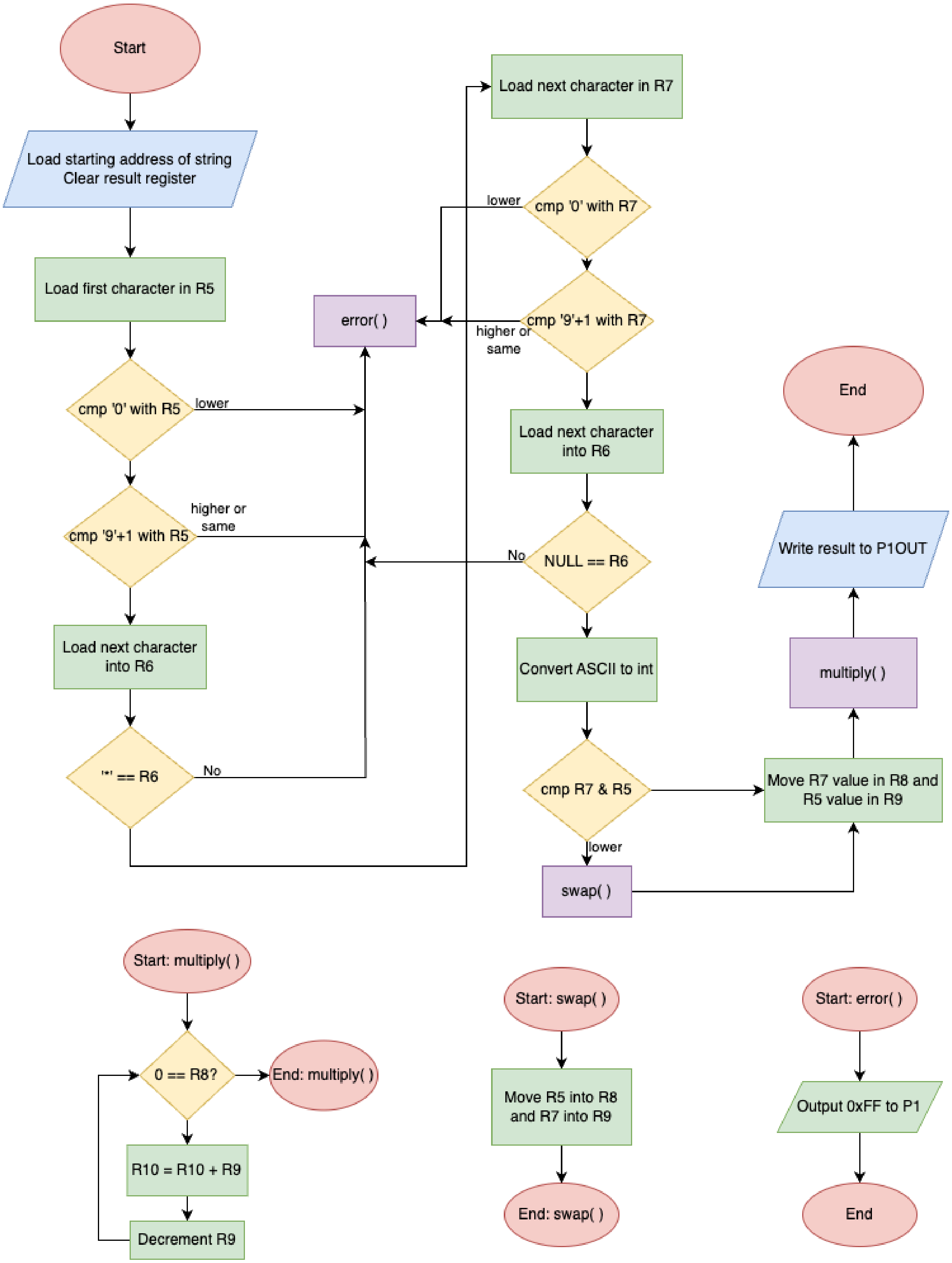


Figure 04: Program 2 Flowchart

Appendix

Table 01: Program 1 source code

```
;-------------------------------------------------------------------------------
; File:         Lab04_P1.asm
; Function:     Count digits and characters in string
; Description: This MSP430 assembly program counts the number of
;  digits and number of characters in the given string. The string
is
;  hard-coded as “Welcome 2 the MSP430 Assembly, Spring 2025!” and
is
;  not more than one line. The number of digits and length of
string
;  are outputted on ports P2 and P1 respectively.
; Input:        The input string defined in myStr
; Output:       Length of given string and Total number of digits
; Author(s):    Anshika Sinha
; Date:         01/29/2025
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.

myStr: .string "Welcome 2 the MSP430 Assembly, Spring 2025!", ''
                               ; '' ensures NULL character follows string
;-------------------------------------------------------------------------------
            .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 ;Output the the length of string on port pin P1
        bis.b   #0FFh, &P2DIR   ; Output the number of digits on port pin P2
        mov.w   #myStr, R4 ; Load the starting address of the string into R4
        clr.b   R5           ; Register R5 will serve as a character counter
        clr.b   R7             ; Register R7 will serve as a digit counter

count:  mov.b   @R4+, R6                ; Load next character
        cmp.b   #0, R6                  ; Check for null character
        jeq     write                   ; If yes, go to the end
        inc.w   R5                      ; If not, increment counter
        cmp.b   #'0', R6               ; Check if character is >= '0'
        jlo            count           ; Jump if lower
        cmp.b   #'9'+1, R6             ; Check if character is <= '9'+1
        jhs            count          ; Jump if higher
        inc.w   R7                    ; If digit, increment counter
        jmp     count                  ; Go to the next character

write:  mov.b   R5,&P1OUT              ; Write character count in P1OUT
       mov.b    R7,&P2OUT               ; Write digit count 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 02: Program 2 source code

```
;----------------------------------------------------------------------------
; File:        Lab04_P2.asm
; Function:    Multiplication by addition
; Description: This MSP430 assembly program performs multiplication
by
;  addition from a given string. The string must be in the format
;  “x*y”. If it is not, 0xFF is written to the output port P1OUT.
The
;  program iterates based on the value of the smaller operand and
;  adds the larger integer to itself in a loop. The final value is
an
;  integer value written to P1OUT.
; Input:       Input string of format "x*y" defined in myStr
; Output:      Integer product of two numbers
; Author(s):   Anshika Sinha
; Date:        02/02/2025
;----------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file

;----------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
;----------------------------------------------------------------------------
myStr:          .string "4*7", ''
;----------------------------------------------------------------------------
            .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                  ; Set P1 as output
                mov.w   #myStr, R4; Load starting address of string into R4
                clr.b   R10        ; Clear result

; Check validity of string
                mov.b   @R4+, R5               ; Load first character into R5
                cmp.b   #'0', R5               ; Check if char is >= '0'
                jlo             error          ; Jump to error if lower
                cmp.b   #'9'+1, R5             ; Check if  char is <= '9'+1
                jhs             error   ; Jump to error if higher or same

                mov.b   @R4+, R6               ; Load next character
                cmp.b   #'*', R6               ; Check if '*'
                jne     error                  ; If not, jump to error

                mov.b   @R4+, R7               ; Load second operand
                cmp.b   #'0', R7               ; Check if char is >= '0'
                jlo     error                   ; Jump to error if lower
                cmp.b   #'9'+1, R7             ; Check if  char is <= '9'+1
                jge     error             ; Jump to error if higher or same

                mov.b   @R4+, R6        ; Load next character in string
                cmp.b   #0, R6          ; Check if null character
                jne     error           ; If not, jump to error

                ; Convert ASCII to int
                sub.b   #'0', R5        ; Convert first operand to int
                sub.b   #'0', R7        ; Convert second operand to int

                ; Find smaller int
                cmp.b   R7, R5                  ; Check R5 > R7
                jlo             swap           ; swap if first is smaller
                mov.b   R7, R8                 ; Smaller number in R8
                mov.b   R5, R9                 ; Larger number in R9
                jmp     multiply

swap:
                mov.b   R5, R8                ; move smaller number in R8
                mov.b   R7, R9                ; move larger number in R9

multiply:
                cmp     #0, R8
                jeq     done                   ; Exit if counter < 0
                add.b   R9, R10                ; Add larger number to result
                dec     R8                     ; Decrease loop counter
                jmp     multiply

done:   mov.b   R10,&P1OUT               ; Write result in P1OUT
```

```
lend:   bis.w   #LPM4, SR               ; LPM4
        nop                             ; Required only for debugger

error:  mov.b   #0FFh, &P1OUT                  ; Output 0xFF if input is
invalid
                jmp     lend
                nop

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

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

Make sure to submit a single pdf file only