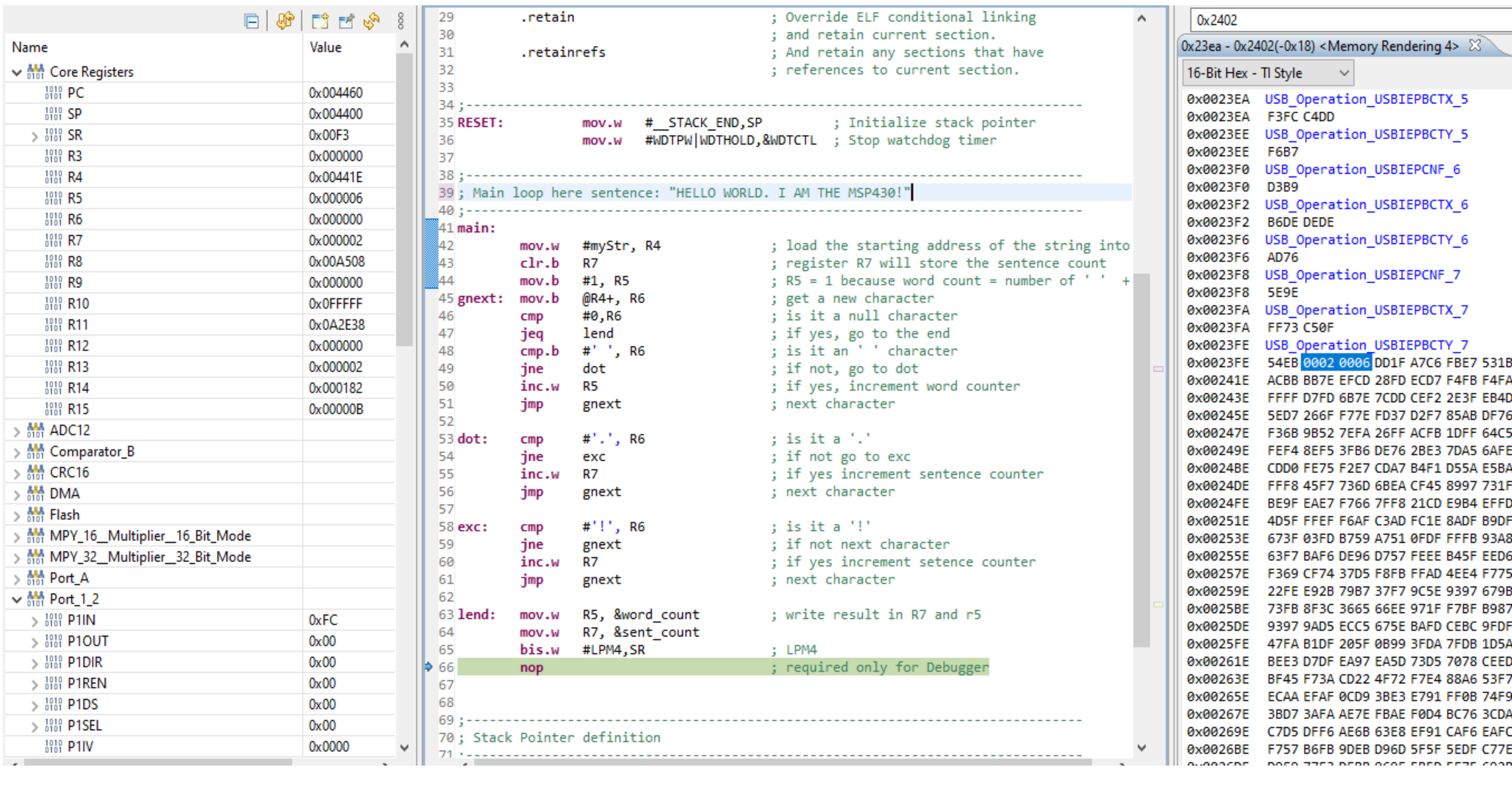
Juan Tarrat

LAB 4 REPORT

1. THEORY TOPICS

- **Assembler directives:** Assembler directives tell the assembler to se the data at particular addresses, allocate space in memory for constants and variables, define synonyms or include additional files. The main directives I have used in this lab are the .data directive, which places variables under this section in the RAM memory, and some constant initialization directives likes .int (integer) or .cstring (similar to .string but .cstring adds a null character at the end of the array of characters so it´s easier to know when the array ends.
  Where you place your variables is important depending on the task you need to do. If you place variables in the .text section these will be placed in flash memory and will only be available to read operations. However the .data section places information in the RAM memory which is read and write.
- **Addressing modes:** Addressing modes are used to tell the compiler how you are accessing a register or a memory location. There are a total of 7 addressing modes: *register, indexed, symbolic, absolute, indirect register, indirect autoincrement,* and *immediate*.
  Indirect autoincrement is an addressing mode only valid for source operands and with syntax @Rn+. The effective address is the content of the address pointed by the register. After this, the register is incremented by +1 if the operation is byte-size and by 2 if the operation is word-size. For example if R6 is pointing to address 0x2402 which holds the string *"oh"* and we have *mov.b @R6+, R7,* the instruction would look like this: R7🡨M[R6]; R6🡨R6 + 1. So R7 will end up having the character 'o' and R6 will now point to address 0x2403 where 'h' resides.

2. PROGRAM 1

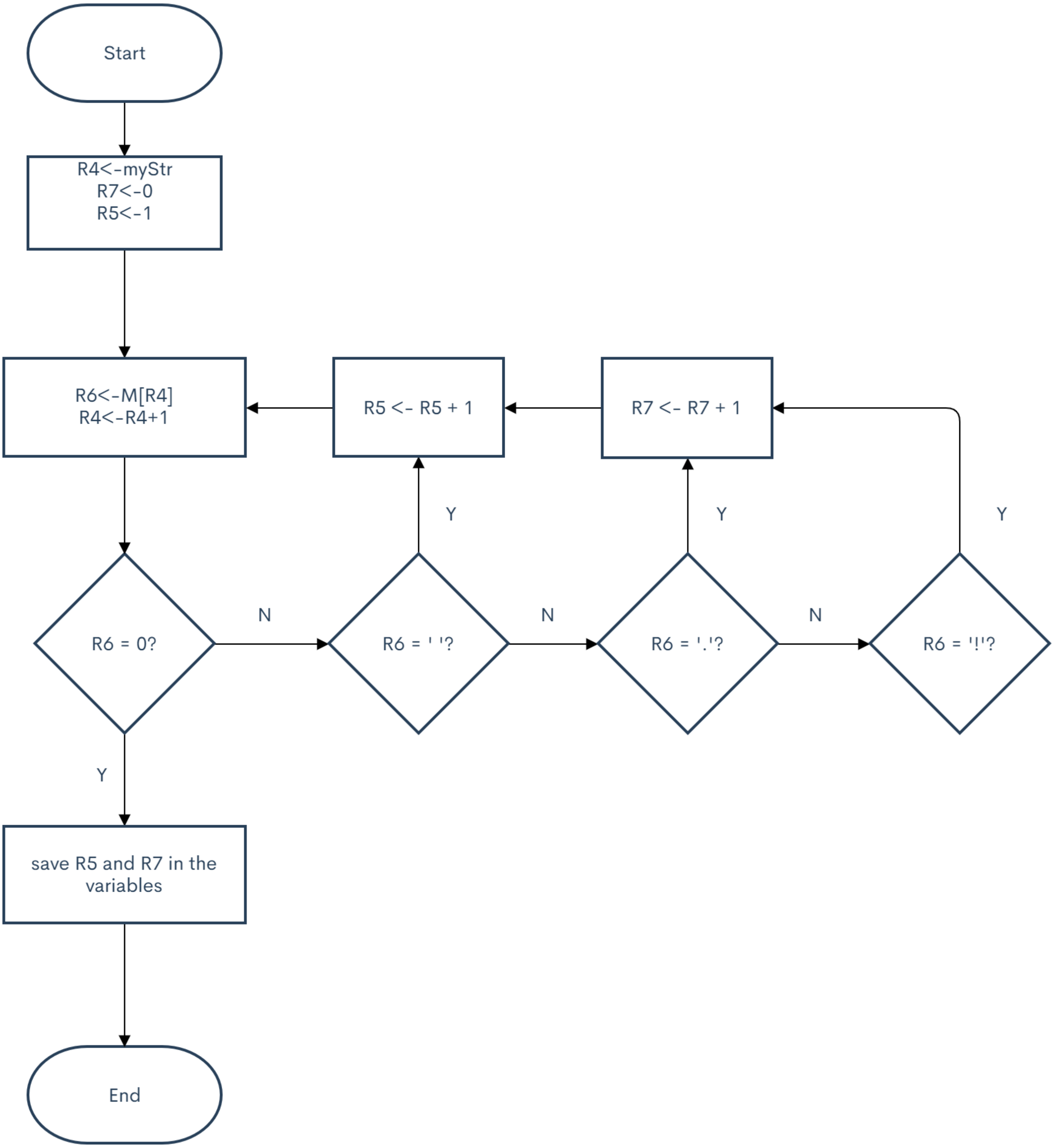


Name	Value
Core Registers	
PC	0x004460
SP	0x004400
SR	0x00F3
R3	0x000000
R4	0x00441E
R5	0x000006
R6	0x000000
R7	0x000002
R8	0x00A508
R9	0x000000
R10	0x0FFFFF
R11	0x0A2E38
R12	0x000000
R13	0x000002
R14	0x000182
R15	0x00000B
ADC12	
Comparator_B	
CRC16	
DMA	
Flash	
MPY_16__Multiplier__16_Bit_Mode	
MPY_32__Multiplier__32_Bit_Mode	
Port_A	
Port_1_2	
P1IN	0xFC
P1OUT	0x00
P1DIR	0x00
P1REN	0x00
P1DS	0x00
P1SEL	0x00
P1IV	0x0000

```
            .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 sentence: "HELLO WORLD. I AM THE MSP430!"
;-------------------------------------------------------------------------------
main:
            mov.w   #myStr, R4              ; load the starting address of the string into
            clr.b   R7                      ; register R7 will store the sentence count
            mov.b   #1, R5                  ; R5 = 1 because word count = number of ' '  +
gnext:      mov.b   @R4+, R6                ; get a new character
            cmp     #0,R6                   ; is it a null character
            jeq     lend                    ; if yes, go to the end
            cmp.b   #' ', R6                ; is it an ' ' character
            jne     dot                     ; if not, go to dot
            inc.w   R5                      ; if yes, increment word counter
            jmp     gnext                   ; next character

dot:        cmp     #'.', R6                ; is it a '.'
            jne     exc                     ; if not go to exc
            inc.w   R7                      ; if yes increment sentence counter
            jmp     gnext                   ; next character

exc:        cmp     #'!', R6                ; is it a '!'
            jne     gnext                   ; if not next character
            inc.w   R7                      ; if yes increment setence counter
            jmp     gnext                   ; next character

lend:       mov.w   R5, &word_count         ; write result in R7 and r5
            mov.w   R7, &sent_count
            bis.w   #LPM4,SR                ; LPM4
            nop                             ; required only for Debugger


;-------------------------------------------------------------------------------
; Stack Pointer definition
```

```
0x2402
0x23ea - 0x2402(-0x18) <Memory Rendering 4>
16-Bit Hex - TI Style
0x0023EA  USB_Operation_USBIEPBCTX_5
0x0023EA  F3FC C4DD
0x0023EE  USB_Operation_USBIEPBCTY_5
0x0023EE  F6B7
0x0023F0  USB_Operation_USBIEPCNF_6
0x0023F0  D3B9
0x0023F2  USB_Operation_USBIEPBCTX_6
0x0023F2  B6DE DEDE
0x0023F6  USB_Operation_USBIEPBCTY_6
0x0023F6  AD76
0x0023F8  USB_Operation_USBIEPCNF_7
0x0023F8  5E9E
0x0023FA  USB_Operation_USBIEPBCTX_7
0x0023FA  FF73 C50F
0x0023FE  USB_Operation_USBIEPBCTY_7
0x0023FE  54EB 0002 0006 DD1F A7C6 FBE7 531B
0x00241E  ACBB BB7E EFCD 28FD ECD7 F4FB F4FA
0x00243E  FFFF D7FD 6B7E 7CDD CEF2 2E3F EB4D
0x00245E  5ED7 266F F77E FD37 D2F7 85AB DF76
0x00247E  F36B 9B52 7EFA 26FF ACFB 1DFF 64C5
0x00249E  FEF4 8EF5 3FB6 DE76 2BE3 7DA5 6AFE
0x0024BE  CDD0 FE75 F2E7 CDA7 B4F1 D55A E5BA
0x0024DE  FFF8 45F7 736D 6BEA CF45 8997 731F
0x0024FE  BE9F EAE7 F766 7FF8 21CD E9B4 EFFD
0x00251E  4D5F FFEF F6AF C3AD FC1E 8ADF B9DF
0x00253E  673F 03FD B759 A751 0FDF FFFB 93A8
0x00255E  63F7 BAF6 DE96 D757 FEEE B45F EED6
0x00257E  F369 CF74 37D5 F8FB FFAD 4EE4 F775
0x00259E  22FE E92B 79B7 37F7 9C5E 9397 679B
0x0025BE  73FB 8F3C 3665 66EE 971F F7BF B987
0x0025DE  9397 9AD5 ECC5 675E BAFD CEBC 9FDF
0x0025FE  47FA B1DF 205F 0B99 3FDA 7FDB 1D5A
0x00261E  BEE3 D7DF EA97 EA5D 73D5 7078 CEED
0x00263E  BF45 F73A CD22 4F72 F7E4 88A6 53F7
0x00265E  ECAA EFAF 0CD9 3BE3 E791 FF0B 74F9
0x00267E  3BD7 3AFA AE7E FBAE F0D4 BC76 3CDA
0x00269E  C7D5 DFF6 AE6B 63E8 EF91 CAF6 EAFC
0x0026BE  F757 B6FB 9DEB D96D 5F5F 5EDF C77E
```

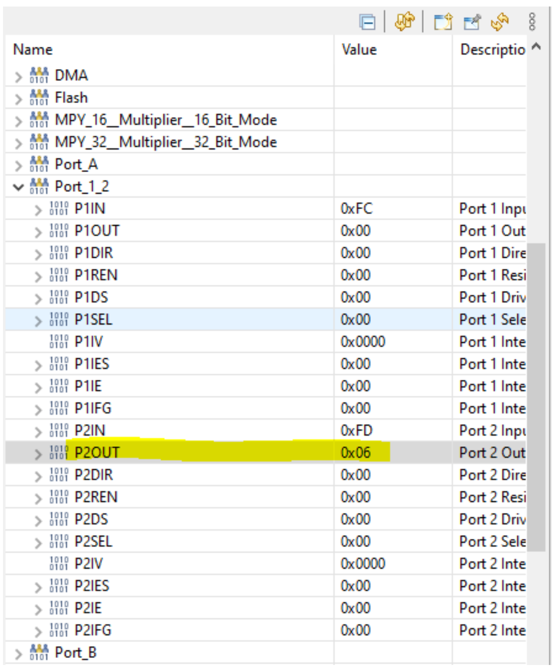
Juan Tarrat



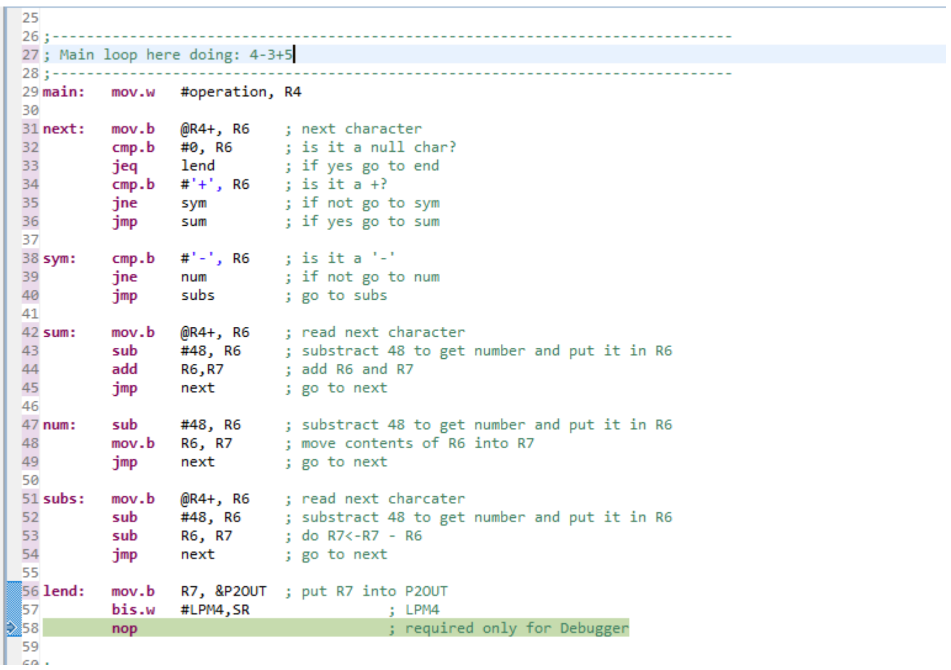
Start
R4<-myStr
R7<-0
R5<-1
R6<-M[R4]
R4<-R4+1
R5 <- R5 + 1
R7 <- R7 + 1
R6 = 0?
N
Y
R6 = ' '?
N
Y
R6 = '.'?
N
Y
R6 = '!'?
Y
save R5 and R7 in the variables
End

Juan Tarrat

3. Program 2

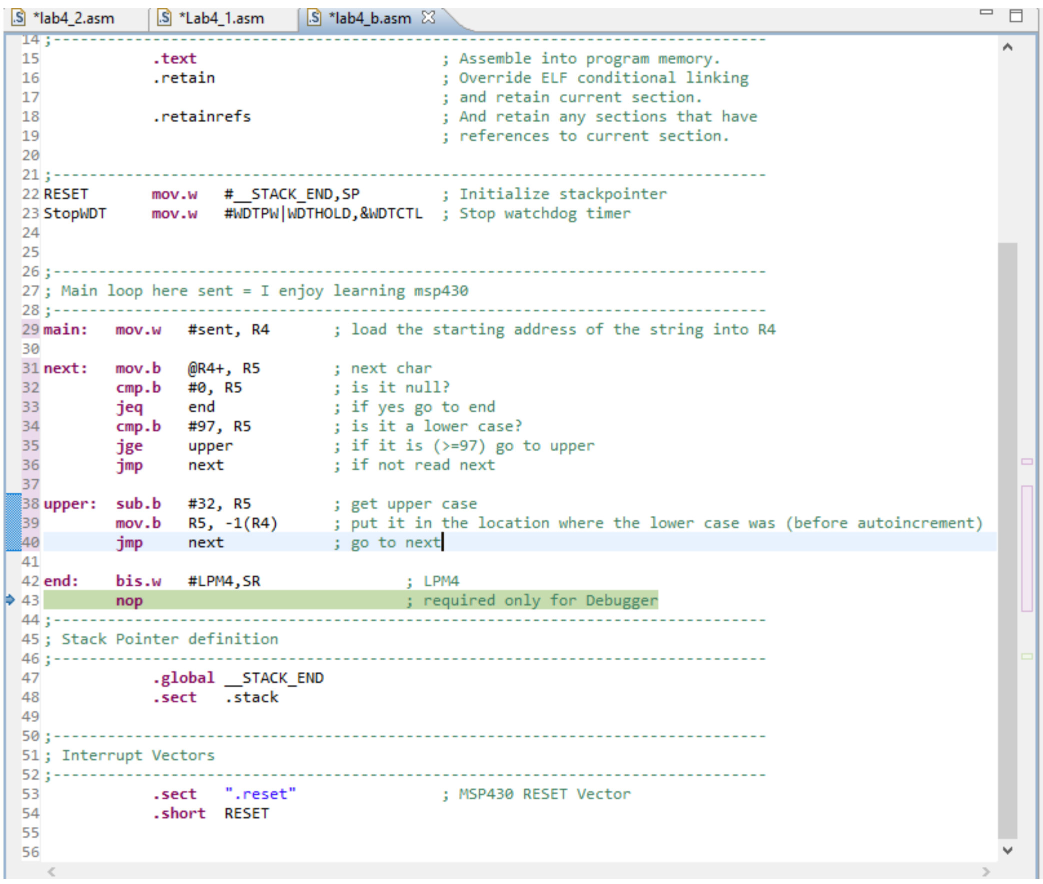


Name	Value	Descriptio
DMA		
Flash		
MPY_16__Multiplier__16_Bit_Mode		
MPY_32__Multiplier__32_Bit_Mode		
Port_A		
Port_1_2		
P1IN	0xFC	Port 1 Inpu
P1OUT	0x00	Port 1 Out
P1DIR	0x00	Port 1 Dire
P1REN	0x00	Port 1 Resi
P1DS	0x00	Port 1 Driv
P1SEL	0x00	Port 1 Sele
P1IV	0x0000	Port 1 Inte
P1IES	0x00	Port 1 Inte
P1IE	0x00	Port 1 Inte
P1IFG	0x00	Port 1 Inte
P2IN	0xFD	Port 2 Inpu
P2OUT	0x06	Port 2 Out
P2DIR	0x00	Port 2 Dire
P2REN	0x00	Port 2 Resi
P2DS	0x00	Port 2 Driv
P2SEL	0x00	Port 2 Sele
P2IV	0x0000	Port 2 Inte
P2IES	0x00	Port 2 Inte
P2IE	0x00	Port 2 Inte
P2IFG	0x00	Port 2 Inte
Port_B		



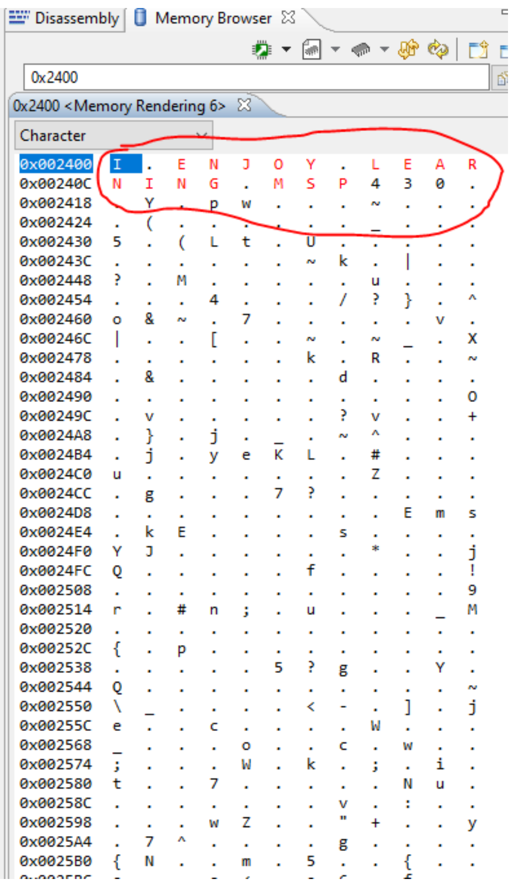
```
;-------------------------------------------------------------------------------
; Main loop here doing: 4-3+5
;-------------------------------------------------------------------------------
main:   mov.w   #operation, R4

next:   mov.b   @R4+, R6      ; next character
        cmp.b   #0, R6        ; is it a null char?
        jeq     lend          ; if yes go to end
        cmp.b   #'+', R6      ; is it a +?
        jne     sym           ; if not go to sym
        jmp     sum           ; if yes go to sum

sym:    cmp.b   #'-', R6      ; is it a '-'
        jne     num           ; if not go to num
        jmp     subs          ; go to subs

sum:    mov.b   @R4+, R6      ; read next character
        sub     #48, R6       ; substract 48 to get number and put it in R6
        add     R6,R7         ; add R6 and R7
        jmp     next          ; go to next

num:    sub     #48, R6       ; substract 48 to get number and put it in R6
        mov.b   R6, R7        ; move contents of R6 into R7
        jmp     next          ; go to next

subs:   mov.b   @R4+, R6      ; read next charcater
        sub     #48, R6       ; substract 48 to get number and put it in R6
        sub     R6, R7        ; do R7<-R7 - R6
        jmp     next          ; go to next

lend:   mov.b   R7, &P2OUT    ; put R7 into P2OUT
        bis.w   #LPM4,SR                        ; LPM4
        nop                                     ; required only for Debugger

```

4. Bonus



*lab4_2.asm | *Lab4_1.asm | *lab4_b.asm

```
;-------------------------------------------------------------------------------
            .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 sent = I enjoy learning msp430
;-------------------------------------------------------------------------------
main:   mov.w   #sent, R4       ; load the starting address of the string into R4

next:   mov.b   @R4+, R5        ; next char
        cmp.b   #0, R5          ; is it null?
        jeq     end             ; if yes go to end
        cmp.b   #97, R5         ; is it a lower case?
        jge     upper           ; if it is (>=97) go to upper
        jmp     next            ; if not read next

upper:  sub.b   #32, R5         ; get upper case
        mov.b   R5, -1(R4)      ; put it in the location where the lower case was (before autoincrement)
        jmp     next            ; go to next

end:    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


```



Disassembly | Memory Browser

0x2400

0x2400 <Memory Rendering 6>

Character

```
0x002400  I  .  E  N  J  O  Y  .  L  E  A  R
0x00240C  N  I  N  G  .  M  S  P  4  3  0  .
0x002418  .  Y  .  p  w  .  .  .  ~  .  .  .
0x002424  .  (  .  .  .  .  .  .  _  .  .  .
0x002430  5  .  (  L  t  .  U  .  .  .  .  .
0x00243C  .  .  .  .  .  .  ~  k  .  |  .  .
0x002448  ?  .  M  .  .  .  .  .  u  .  .  .
0x002454  .  .  .  4  .  .  .  /  ?  }  .  ^
0x002460  o  &  ~  .  7  .  .  .  .  .  v  .
0x00246C  |  .  .  [  .  .  ~  .  ~  _  .  X
0x002478  .  .  .  .  .  .  k  .  R  .  .  ~
0x002484  .  &  .  .  .  .  .  d  .  .  .  .
0x002490  .  .  .  .  .  .  .  .  .  .  .  O
0x00249C  .  v  .  .  .  .  .  ?  v  .  .  +
0x0024A8  .  }  .  j  .  _  .  ~  ^  .  .  .
0x0024B4  .  j  .  y  e  K  L  .  #  .  .  .
0x0024C0  u  .  .  .  .  .  .  .  Z  .  .  .
0x0024CC  .  g  .  .  .  7  ?  .  .  .  .  .
0x0024D8  .  .  .  .  .  .  .  .  .  E  m  s
0x0024E4  .  k  E  .  .  .  .  s  .  .  .  .
0x0024F0  Y  J  .  .  .  .  .  .  *  .  .  j
0x0024FC  Q  .  .  .  .  .  f  .  .  .  .  !
0x002508  .  .  .  .  .  .  .  .  .  .  .  9
0x002514  r  .  #  n  ;  .  u  .  .  .  _  M
0x002520  .  .  .  .  .  .  .  .  .  .  .  .
0x00252C  {  .  p  .  .  .  .  .  .  .  .  .
0x002538  .  .  .  .  .  5  ?  g  .  .  Y  .
0x002544  Q  .  .  .  .  .  .  .  .  .  .  ~
0x002550  \  _  .  .  .  .  <  -  .  ]  .  j
0x00255C  e  .  .  c  .  .  .  .  W  .  .  .
0x002568  _  .  .  .  o  .  .  c  .  w  .  .
0x002574  ;  .  .  .  W  .  k  .  ;  .  i  .
0x002580  t  .  .  7  .  .  .  .  .  N  u  .
0x00258C  .  .  .  .  .  .  .  v  .  :  .  .
0x002598  .  .  .  w  Z  .  .  "  +  .  .  y
0x0025A4  .  7  ^  .  .  .  .  g  .  .  .  .
0x0025B0  {  N  .  .  m  .  5  .  .  {  .  .
```