CPE/EE 323 Introduction to Embedded Computer Systems
Homework III (MSP430 Assembly Language)

1(25)	2(25)	3(25)	4(25)	Total

Problem #1 (25 points) Assembly Language Directives

A. (15 points) Consider the assembly directives shown below. Show the content of relevant regions of the memory initialized by these directives. The memory is organized in words. The MSP430 ISA is a little-endian architecture. Fill in the table below. Assume that the assembler places the data segment in RAM memory starting from the address 0x1100. How many bytes is allocated and initialized by these directives?

```
                .data
lb1:            .byte 2, 4, -2, 'B'
lb2:            .byte 10011110b, 356q, 0x40
                .align 2
lw1:            .word -6, 4, 0x22AC
llw1:           .long 2
lf1:            .float 2.0
ls:             .cstring "ABC"
```

Label	Address [hex]	Memory[15:0] [hex]	
lb1	0x1100	0x0402	✓ 04=4, 02=2
	0x1102	0x42FE	✓ FE=-2, 42='B'
lb2	0x1104	0xEE9E	✓ EE=356, 9E 1001 1110
	0x1106	0x0040	
lw1	0x1108	0xFFFA	0xFFFA=-6
	0x110A	0x0004	0x0004=4
	0x110C	0x22AC	22AC=22AC
llw1	0x110E	0x0002	0x0002=2
	0x1110	0x0000	0x0002=2.0
lf1	0x1112	0x0000	
	0x1114	0x4000	
ls	0x1116	0x4241	42='B' 41='A'
	0x1118	0x0043	43='C'
	0x111A		
	0x111C		

B. (10 points) Consider the following sequence of instructions.

```
mov.b   lb1, R4
mov.b   lb1+2, R5
mov.w   lb2+2, R6
mov.w   llw1, R7
mov.w   lf1, R8
mov.w   ls, R9
mov.w   #ls, R10
mov.w   &lf1, R11
mov.b   &lb1+1, R12
mov.w   lf1+2, R13
```

What is the content of register R4-R9 after execution of this code snippet?

Register	Content [HEX]
R4	0X0002
R5	0X00FE
R6	0X0040
R7	0X0002
R8	0X0000
R9	0X4241
R10	0X1116
R11	0X1112
R12	0X0001
R13	0X4000

2. (25 points) Analyze assembly program src, dst dst ← src + not.dst +1

Consider the following code segment.

```
1          mov.w   #mya, R14      move the byte's into R14
2          mov.w   #myaa, R13     move empty array into R13
3          sub.w   R14, R13       R13 ← R13 + not R14      R13 ← 0x0007
4          mov.b   #0x80, R7      move 0x80 (128) into R7
5 MyLoop:  mov.b   @R14+, R15     Increment to next number, store in R15
6          cmp.b   R7, R15        Compare R7 to mya[R14]   R7  R15 ?
7          jl      lskip          If it is less than R7 , go to lskip   R7 < R15 ?
8          mov.b   R15, R7        If it is = or greater than, put R15->R7
9 lskip:   dec.w   R13            Decrement R13
10         jnz     MyLoop         jump not zero to MyLoop
11         mov.b   R7, P1OUT      Move R7 into P1OUT if is 0. End Program.
12
13 mya:    .byte   4, 5, 6, -10, 55, 64, -100      7, 8bit values: 7x8
14 myaa:           1  2  3   4   5   6    7
```

cmp 128, 4
→ lskip
dec R13
cmp

E

A. (3 points) How many bytes is allocated by the assembly directive in line 13? 4, 5, 6, -10, 55, 64, -100

7 bytes. 56 bits

B. (3 points) What is the content of register R13 after the instruction in line 3 is completed?

R13 ← R13 + not. R14 R13 -> 0x0007 2407 - 2400 = 0x0007

C. (10 points) What does this code segment do? Explain your answer. Hint: what does #0x80 represent?

0x80 = 128 d This code finds the highest 8bit value in a variable of bytes. [-128, 127]

D. (4 points) What is the value of P1OUT at the end of the program.

0x40 , or 64

E. (5 points) Calculate the total execution time in seconds for the code sequence from above (line 1 – line 11). We know the following: the average CPI is 1.8 clocks per instruction. Assume the clock frequency is 1 MHz. What is MIPS rate for this code?

```
1        1  1  mov.w   #mya, R14    ; 2 CC  1
2        2  1  mov.w   #myaa, R13   ; 2 CC  2
3        3  1  sub.w   R14, R13     ; 1 CC  3
4        4  1  mov.b   #0x80, R7    ; 1 CC  4
5 MyLoop:1x7   mov.b   @R14+, R15   ;          1x7
6        1x7   cmp.b   R7, R15      ;          1x7
7        1x7   jl      lskip        ; 2 CC
8        1x5   mov.b   R15, R7      ; 1 CC
9 lskip: 1x7   dec.w   R13          ; 1 CC     1x7
10       1x7   jnz     MyLoop       ;          1x7
11       5  1x1  mov.b   R7, P1OUT  ; 1 CC     1x1
12
```

CCF = 1 MHz 5 + 35 = 40 instructions

CPI = 1.8 clocks

$$ET = \frac{40 \times 1.8}{1 \times 10^{6}}$$

$$ET = 7.2 \times 10^{-5} \text{ s}$$

$$MIPS = \frac{CCF}{(10^{6})(CPI \times 10^{6}}$$

$$MIPS = \frac{10^{6}}{1.8 \times 10^{6}} = 0.5556$$

3. (25 points) Analyze assembly program

Consider the following code segment.

```
 1 RESET:     mov.w   #__STACK_END,SP          ; Initialize stack pointer
 2 StopWDT:   mov.w   #WDTPW|WDTHOLD,&WDTCTL   ; Stop watchdog timer
 3            sub.w   #12, SP        ; sub 12 from stack pointer
 4            mov.w   SP, R6         ; mov stack pointer to R6
 5            mov.w   #myinput, R4   ; R4 <- # my input
 6 gnext:     mov.b   @R4+, R5       ; increment to next character, store in R5
 7            cmp.b   #0, R5         ; compare R5 to 0, NULL
 8            jz      lend           ; jump to end if R5=0
 9            cmp.b   #'A', R5       ; Compare R5 to 'A'
10            jl      lcopy          ; R5 < 'A' (not a letter)
11            cmp.b   #'Z'+1, R5     ; Compare R5 to 'Z'
12            jl      lconv          ; R5 < 97 Ascii, conv
13            jmp     lcopy          ; unconditionally jump to lcopy
14 lconv:    sub.b   #'A', R5       ; R5 <- R5-'A'
15            add.b   #'a', R5       ; R5 <- R5+'a'
16 lcopy:    mov.b   R5, 0(R6)      ; R6 to <- R5
17            inc.w   R6             ; Increment R6
18            jmp     gnext          ; unconditionally jump to gnext
19 lend:     mov.b   R5, 0(R6)
20            jmp     $                ; jump to current location '$' (endless loop)
21 myinput:  .cstring "CPE325-lab"
```

A. (12 points) What does this program do? Add code comments (lines 1-19).

This program converts letters to lower case.

B. (10 points) Sketch the content of the stack at the moment when the program executes the instruction at line 20. Assume that the original value of R1=0x4400 (initialized in line 1). ascii('A')=0x41, ascii('Z')=0x5A, ascii('0')=0x30.

Address	M[15..8]	M[7..0]
0X43F4	0X70	0X63
0X43F6	0X33	0X65
0X43F8	0X35	0X32
0X43FA	0X6C	0X2D
0X43FC	0X62	0X61
0X43FE	0X00	0X00

C. (3 points) Calculate the total execution time in seconds (time it takes for the program to reach statement in line 20). Assume the clock frequency is 8 MHz. How many instructions are executed in this program (before reaching the statement at line 20)?

CF = 8 MHz

$ET = IC \times CPI \times CCT = \frac{IC \times CPI}{CF}$ 91 instructions

$ET = \frac{91 \times 1}{8 \times 10^6} = 11.375 \times 10^{-6}$

4. (25 points) Write a subroutine

Design and write an MSP430 assembly language subroutine i2a_s(char *a, int myI) that converts a 16-bit integer, myI, into a character array with elements corresponding to the hexadecimal representation of the integer. For example, an integer myI=13,486=0x34AE is converted into an array with 4 elements as follows: a[0]='E', a[1]='A', a[2]='4', a[3]='3'. The main program that calls the subroutine is shown below. Ascii('A')=0x41, ascii('0')=0x30.

```
;-------------------------------------------------------------------------------
RESET:      mov.w   #__STACK_END,SP         ; Initialize stack pointerStopWDT:
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
;-------------------------------------------------------------------------------
; Main code here
;-------------------------------------------------------------------------------
                sub.w   #4, SP                  ; allocate space for ascii chars
                mov.w   SP, R14                 ; R14 points to the allocated area
                mov.w   myI, R4                 ; integer is passed through R4
                mov.w    array, R11                             ; making space for the new character array.
                push.w  R14                     ; push the starting address on the stack
                call    #i2a_s                  ; call subroutine
                add.w   #2, SP                  ; free space on the stacklend:
                jmp     $
myI:    .word     0x34AE
array:
;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
                .global __STACK_END
                .sect   .stack
;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
                .sect   ".reset"                ; MSP430 RESET Vector
                .short  RESET

i2a_s:
                        mov.w   R4, R5          ; Copy value of R4 into R5
                        and.b   #0x0F, R5       ; R5 wil; have 0x0003
                        jmp     rotate          ; jump to rotate subroutine, Rotate the R4 value [0xE34A]

                        mov.w   R4, R6          ; Copy value of R4 into R6
                        and.b   #0x0F, R6       ; R5 wil; have 0x0004
                        jmp     rotate          ; jump to rotate subroutine, Rotate the R4 value [0xAE34]

                        mov.w   R4, R7          ; Copy value of R4 into R6
                        and.b   #0x0F, R7       ; R7 will have 0x000A
                        jmp     rotate          ; jump to rotate subroutine, Rotate the R4 value  [0x4AE3]

                        mov.w   R4, R8          ; Copy value of R4 into R6
                        and.b   #0x0F, R8       ; R8 will have 0x000E
                        jmp     rotate          ; jump to rotate subroutine, Rotate the R4 value [0x34AE]

                        mov.b   R5, R10                 ; copy R5 into R10
                        jmp     compare                 ; Jump To Compare
                        mov.b   R10, R5                 ; Update R5

                        mov.b   R6, R10                 ; copy R6 into 10
                        jmp     compare                 ; Jump To Compare
                        mov.b   R10, R6                 ; Update R6

                        mov.b   R7, R10                 ; copy R7 into R10
                        jmp             compare         ; Jump To Compare
                        mov.b   R10, R7                 ; Update R7

                        mov.b   R8, R10                 ; copy R8 into R10
                        jmp             compare         ; Jump To Compare
                        mov.b   R10, R8                 ; Update R8

                        mov.b   R8, 0(R11)      ; Move the value into the first part of R11
```

```
                    mov.b   R7, 1(R11)      ; Move the value into the second part of R11
                    mov.b   R6, 2(R11)      ; Move the value into the third part of R11
                    mov.b   R5, 3(R11)      ; Move the value into the fourth part of R11
                    ret

rotate:             rra     R4      ; Rotate R4 for next bit
                    rra     R4      ; Rotate R4 for next bit
                    rra     R4      ; Rotate R4 for next bit
                    rra     R4      ; Rotate R4 for next bit

compare:            cmp.b   #0x0A, R10      ; compare A to R7
                    jl      number          ; If it is less than, then this is a number.
                    jge     letter          ; If it is greater than, then we know that it is a letter.

letter:
                    add     #0x37, R10              ; converting to hex
                    ret

number:             add             #0x30, R10      ; converting to hex
                    ret
```