EE 215 Homework #4

(6 problems, 90 pts total)

1. Loops (10 pts)
   Explain what the following assembly language does. Comment each line of code AND describe the overall effect of the program.

```
        mov.b #7,R4
        mov.w #0x2490,R5
loop:
        clr.w 0(R5)
        add.w #2,R5
        sub.b #1,R4
        jne loop
```

Answer:

```
    mov.b #7,R4   ; Move the value 7 into register R4 (8-bit move); Overall
effect: Initializes R4 with the value 7.
    mov.w #0x2490,R5   ; Move the value 0x2490 into register R5 (16-bit
move); Overall effect: Initializes R5 with the value 0x2490.
loop:
    clr.w 0(R5)   ; Clear the word (16-bit) value at the memory address
pointed to by R5
                  ; Overall effect: Clears the 16-bit value at the memory
address pointed to by R5.
    add.w #2,R5   ; Add 2 to the value in R5 (incrementing the memory
address)
                  ; Overall effect: Increments the memory address in R5 by 2.
    sub.b #1,R4   ; Subtract 1 from the value in R4 (8-bit subtraction)
                  ; Overall effect: Decrements the value in R4 by 1.
    jne loop      ; Jump to the 'loop' label if the zero flag is not set (if
R4 is not equal to 0)
                  ; Overall effect: Repeats the loop as long as R4 is not
equal to 0.
```

2. Branching (10 pts)
   Translate this high-level language into assembly language.

```
if  T is greater than 10
then X=3
else Y=6
end if
```

The labels ‘T’, ‘X’, and ‘Y’ have already been defined as references to specific memory addresses, complete the rest of the assembly language program.

```
        .data
T:      .word  0x000A      ; memory location
X:      .space 2           ; word size memory for variable X
Y:      .space 2           ; word size memory for variable Y

        .text
```

Answer：

```
        .data
T:      .word   0x000A      ; Memory location for T
X:      .space  2          ; Memory space for variable X (2 bytes)
Y:      .space  2          ; Memory space for variable Y (2 bytes)

        .text
        mov.w   &T, R4       ; Load the value of T into register R4
        cmp     #10, R4      ; Compare the value in R4 with 10
        jl      else         ; Jump to the else label if less than 10
        mov     #3, R5       ; Set R5 to 3
        mov.w   R5, &X        ; Store the value of R5 in the memory location
of X
        jmp     end_if       ; Jump to the end of the if statement
else:
        mov     #6, R6       ; Set R6 to 6
        mov.w   R6, &Y        ; Store the value of R6 in the memory location
of Y
          jmp $             ; infinite loop to end program
```

3. While loop (20 pts total)
   When programming, you may not know the length of a list of data, but the list can be marked at the end by a character like zero or a negative number. In this case, use a “while” loop to process the data.
   (a) Comment each line of code.
   (b) Draw the memory map diagram (with labels) illustrating the contents of all memory affected at the end of execution. Denote all labels.
   (c) Show the final value of the affected registers at the end of execution.
   (d) Describe the overall effect of the program.

```
        .data
List:   .byte 1,3,5,7,9,0
Sum:    .byte 0

        .text
        mov.w #List,R5
Loop:   mov.b @R5,R6
        cmp.b #0,R6
        jeq   Done
        add.b R6,Sum
        inc.w R5
        jmp   Loop
Done:   jmp   $
```

Answer：

(a)

```
    .data
List:  .byte 1,3,5,7,9,0     ; Define a list of bytes, terminated by 0
Sum:   .byte 0               ; Initialize Sum to 0

    .text
  mov.w  #List,R5            ; Load the address of List into R5

Loop:  mov.b  @R5,R6         ; Load the byte at the address in R5 into R6
  cmp.b  #0,R6               ; Compare the byte in R6 with 0
  jeq    Done                ; If equal (zero), jump to Done
  add.b  R6,Sum              ; Add the value in R6 to the Sum
  inc.w  R5                  ; Increment the address in R5
  jmp    Loop                ; Jump back to Loop

  Done:  jmp    $            ; End of program
```

(b)

0x2400	List	1
0x2401		3
0x2402		5
0x2403		7
0x2404		9
0x2405		0
0x2406	Sum	25

(c)

R5: It would contain the memory address of the byte immediately after the '0' in the List, in this case, 0x2406.

R6: It would contain the last value read from the List, which is 0.

(d) The overall effect of the program is to calculate the sum of the values in the List. It starts at the beginning of the List, adds each value to the Sum, and continues until it encounters a 0, which marks the end of the List. At the end of execution, Sum would contain the sum of the values in the List, and R5 would point to the memory address immediately after the '0' in the List.

4. Conditional Jump (10 pts).
   Write a sentence to explain each of the 5 test conditions jXX = jge, jz, jl, jn, jnz. For each one, tell what is in the value of R11 at the end of the program with each of the 5 test conditions. (see MSP430 x5xx Family Users Guide, section 6.6.2)

```
        .text
        clr.w   R11 ; counter
        mov.w   #3,R12    ; for loop?
again:
        inc.w   R11 ; increment the counter
        tst.w   R12
        jXX     done
        sub.w   #1, R12
        jmp     again

done:   jmp  $     ; infinite loop to end
```

Answer：

1. jge (Jump if Greater or Equal):

Checks if the zero flag is clear (R12 is greater than or equal to 0).

R11 at the end: R11 will contain the number of times the loop was executed before exiting.

2. jz (Jump if Zero):

Checks if the zero flag is set (R12 is equal to 0).

R11 at the end: R11 will contain the value 3, as the loop executes 3 times, and R12 reaches 0.

3. jl (Jump if Less):

Checks if the negative flag is set (R12 is less than 0).

R11 at the end: R11 will contain 0, as the loop never executes when R12 is initialized with 3.

4. jn (Jump if Negative):

Checks if the negative flag is set (R12 is negative).

R11 at the end: R11 will contain the number of times the loop was executed before exiting if R12 starts as a negative value.

5. jnz (Jump if Not Zero):

Checks if the zero flag is clear (R12 is not equal to 0).

R11 at the end: R11 will contain the value 3, as the loop executes 3 times, and R12 reaches 0.

5. Stacks (20 pts)
   Fill in the final value of the affected registers and the memory at the end of execution.

```
        .text
        mov #2, R4
        mov #10,R5 ;count
        mov #0x2700, SP
loop:
        push.w R4
        incd R4
        dec R5
        jz  end
        jmp loop
end:
        mov.w 2(SP),R6
        mov.b 3(SP),R7
        jmp $
```

Answer：

	Value	Memory
R4	12	0x2710
R5	0	0x2401
R6	4	0x2702

R7	5	0x2703

6. Subroutine (20 pts) [MSP430 Microcontroller Basics, section 4.6]

   (a) When a subroutine is called, what happens to the stack?

   (b) What would happen if a subroutine changes the top value of the stack?

   (c) What instruction enters a subroutine?

   (d) What instruction returns from a subroutine?

   (e) How do you pass parameters to a subroutine?

   Answer：

   (a) When a subroutine is called, the program's return address is pushed onto the stack.

   (b) If a subroutine changes the top value of the stack, it can lead to unexpected behavior and possibly crashes, as the return address would be altered, affecting the program flow.

   (c) The "CALL" or "CALLA" instruction is used to enter a subroutine.

   (d) The "RET" or "RETI" instruction is used to return from a subroutine.

   (e) Parameters are typically passed to a subroutine via registers, memory locations, or the stack, depending on the calling convention and specific requirements of the subroutine.