Problem #1 (40 points) Consider the following C program. Assume all variables are allocated on the stack in the order as they appear in the program (e.g., the last variable will be on the top of the stack).

```c
#include "io430.h"
#include "stdio.h"

int main(void) {
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
    int x = 5; // an integer x
    int *p_x; // a pointer to int
    int y1; // an integer y1 (uninitialized)
    long int y2, y3; // long integers y2, y3
    long int *p_y2; // a pointer to long integer
    char mya[10] = "Hello!"; // character array
    char *p_mya; // pointer to character

    p_x = &x; // p_x points to x
    *p_x = 7;
    *p_x = *p_x + 2;
    y1 = 10 + *x; // new value to y1
    y2 = -1;
    p_y2 = &y2; // pointer p_y2 points to y2
    *p_y2 = y2 + 3;
    y3 = 10 + *p_y2;
    p_mya = mya; // p_mya points to array mya
    p_mya = p_mya + 3;
    *p_mya = 'L';

    // display addresses and variables in terminal i/o
    printf("a.x=%x, x=%x\n", &x, x);
    printf("a.p_x=%x, p_x=%x\n", &p_x, p_x);
    printf("a.y1=%x, y1=%x\n", &y1, y1);
    printf("a.y2=%x, y2=%lx\n", &y2, y2);
    printf("a.y3=%x, y3=%lx\n", &y3, y3);
    printf("a.p_y2=%x, p_y2=%x\n", &p_y2, p_y2);
    printf("a.mya=%x, mya=%s\n", &mya, mya);
    printf("a.p_mya=%x, p_mya=%s\n", &p_mya, p_mya);
    return 0;
}
```
A. **(20 points)**. Illustrate the content of the stack at the moment (i) before the statement in line 15 is executed and (ii) before the statement in line 27 is executed. Use the comments fields to indicate the individual variables.

<table>
<thead>
<tr>
<th></th>
<th>Orig. TOS</th>
<th>Memory[15:0] hex</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong></td>
<td>0x01FE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Orig. TOS</th>
<th>Memory[15:0] hex</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ii</strong></td>
<td>0x01FE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B. (20 points). For each statement from line 15 to line 25 show its assembly language implementation.

<table>
<thead>
<tr>
<th>Assembly code</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_x = &amp;x;</code></td>
<td></td>
</tr>
<tr>
<td><code>*p_x = 7;</code></td>
<td></td>
</tr>
<tr>
<td><code>*p_x = *p_x + 2;</code></td>
<td></td>
</tr>
<tr>
<td><code>y1 = 10 + x;</code></td>
<td></td>
</tr>
<tr>
<td><code>*p_y2 = y2 + 3;</code></td>
<td></td>
</tr>
</tbody>
</table>
2. (30 points) Consider the following assembly subroutine \textit{mysub} shown below. Two 8-bit input parameters are passed to the subroutine through registers R12 and R13. The result is returned back to a caller in register R13.

\begin{verbatim}
 CC
 1 #include "msp430.h" ; #define controlled include file
 2 3 PUBLIC mysub
 4 5 RSEG CODE
 6
 7 mysub:  PUSH    R10 ; 3
 8         PUSH    R11 ; 3
 9         CLR     R11 1
10         SXT     R13 ; 1
11         MOV     #7, R10 2
12
13 lnext:  BIT.B   #0x01, R12 ; 2
14         JZ      lskip 2
15         ADD     R13, R11 ; 1
16 lskip:  RRA.B   R12 ; 1
17         RLA     R13 ; 1
18         DEC     R10 ; 1
19         JNZ     lnext ; 2
20         BIT.B   #0x01, R12 ; 2
21         JZ      lend ; 2
22         SUB     R13, R11 ; 1
23 lend:   MOV     R11, R13 ; 1
24         POP     R11 ; 2
25         POP     R10 ; 2
26         RET                    ; 3
27 END
\end{verbatim}

A. (15 points) What does this program do? Add code comments (lines 7-26).

B. (10 points) Calculate the total number of clock cycles needed to execute \textit{mysub}. The last column CC gives the execution time in clock cycles for each instruction. Assume that instructions at lines 15 and 22 are executed with probability of 50%.

C. (5 points) Assuming that the CPU runs at 16 MHz clock frequency, what is subroutine execution time? What is CPI (cycles per instruction)? What is MIPS rate for this program?
3. **(30 points)** 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=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`.

How the parameters are passed? Explain your answer.

```assembly
#include "msp430.h"                     ; #define controlled include file
NAME    main                    ; module name
EXTERN  i2a_s
PUBLIC  main                    ; make the main label visible
    ; outside this module
ORG     0FFFEh
DC16    init                    ; set reset vector to 'init' label
RSEG    CSTACK                  ; pre-declaration of segment
RSEG    CODE                    ; place program in 'CODE' segment

init:   MOV     #SFE(CSTACK), SP        ; set up stack
main:   NOP                             ; main program
        MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
        SUB.W   #4, SP                  ; allocate space for ascii chars
        MOV     SP, R14                 ; R14 points to the allocated area
        MOV     myI, R4                 ; integer is passed through R4
        PUSH    R14                     ; push the starting address on the stack
        CALL    #i2a_s                  ; call subroutine
        ADD     #2, SP
skip:   JMP $

myI     DC16     0x34AE
END
```

```assembly
#include "msp430.h"                     ; #define controlled include file

PUBLIC i2a_s
RSEG CODE

i2a_s:
```

```