1. For the given Y86 code:
   (a) explain what the code does
   (b) comment each statement
   (c) convert the code to a C function

**len2:**
```
pushl %ebp                        # ________________________________
rrmovl %esp, %ebp                 # ________________________________
pushl %esi                        # ________________________________
irmovl $4, %esi                   # ________________________________
pushl %edi                        # ________________________________
irmovl $1, %edi                   # ________________________________
mrmovl 8(%ebp), %edx              # ________________________________
irmovl $0, %ecx                   # ________________________________
mrmovl (%edx), %eax              # ________________________________
addl %esi, %edx                   # ________________________________
andl %eax, %eax                   # ________________________________
je Done                           # ________________________________
```

**Loop:**
```
addl %edi, %ecx                   # ________________________________
mrmovl (%edx), %eax              # ________________________________
addl %esi, %edx                   # ________________________________
andl %eax, %eax                   # ________________________________
jne Loop                          # ________________________________
```

**Done:**
```
rrmovl %ecx, %eax                 # ________________________________
popl %edi                         # ________________________________
popl %esi                         # ________________________________
rrmovl %ebp, %esp                 # ________________________________
popl %ebp                         # ________________________________
ret                               # ________________________________
```
2. Consider the following Y86 Assembly-language program:
   .pos 0x12345676
   xorl %edx, %edx
   here: mrmovl here(%edx), %esi
   halt
What value will be in register %esi once the program terminates? Justify your answer.

3. The following Y86 machine codes start at memory address 0x300, which is also the address of the label “loop”. Write corresponding Y86 assembly instructions.

   0x300: loop:
   0x300: 40 50 1c 04 00 00
   0x306: 61 30
   0x308: 73 00 03 00 00

4. The sequential Y86 implementation discussed in class has Fetch, Decode, Execute, Memory, Write-back and PC-update stages, in this order. List one Y86 instruction that could not be executed if the Write-back stage came before the Memory stage, and explain why.

5. Consider the following fragment of IA32 code from the C standard library:
   0x400446e3 <malloc+7>: call 0x400446e8 <malloc+12>
   0x400446e8 <malloc+12>: popl %eax
After the popl instruction completes, what hex value does register %eax contain? ________________

6. The following is a silly recursive procedure:
   int silly(int n, int *p)
   { int val, val2;
     if (n > 0)
       val2 = silly(n << 1, &val);
     else
       val = val2 = 0;
     *p = val + val2 + n;
     return val + val2; }
that yields the following IA32 code:

```assembly
silly:  pushl %ebp
        movl %esp,%ebp
        subl $20,%esp
        pushl %ebx
        movl 8(%ebp),%ebx
        testl %ebx,%ebx
        jle .L3
        addl $-8,%esp
        leal -4(%ebp),%eax
        pushl %eax
        leal (%ebx,%ebx),%eax
        pushl %eax
        call silly
        jmp .L4
.L3:   xorl %eax,%eax
        movl %eax,-4(%ebp)
.L4:   movl -4(%ebp),%edx
        addl %eax,%edx
        movl 12(%ebp),%eax
        addl %edx,%ebx
        movl %ebx(%eax)
        movl %eax,ebx
        movl %ebp,%esp
        popl %ebp
        ret
```

A. Is the variable val stored on the stack? If so, at what byte offset (relative to %ebp) is it stored, and why is it necessary to store it on the stack?

B. Is the variable val2 stored on the stack? If so, at what byte offset (relative to %ebp) is it stored, and why is it necessary to store it on the stack?

C. What (if anything) is stored at -24(%ebp)? If something is stored there, why is it necessary to store it?

D. What (if anything) is stored at -8(%ebp)? If something is stored there, why is it necessary to store it?
7. Based on the problem given in lab 5, what does the stack look like if you call ADD from MAIN, SUBTRACT from ADD, and MULTIPLY from ADD, based on the following information:

Assume that two values are being sent to each function: `funcname(int x, int y)`
Assume that you could possibly need all of the “caller” registers for each function call
Be sure to designate the increasing/decreasing direction of the stack address

8. Consider the following C declaration:

```c
struct More {
    char a;
    short b;
    char c;
};
struct Node {
    char c;
    double value;
    struct Node* left;
    struct Node* right;
    struct More abc;
};
```

Using the template below, indicate the allocation of data for the above struct. Mark off and label the areas for each individual element. Put an X in the parts that are allocated, but not used (to satisfy alignment). Assume the Linux alignment rules and sizes as discussed in class examples. Clearly indicate the right hand boundary of the data structure with a vertical line.
Rewrite the structure, keeping each structure intact, to save as much memory as possible; then indicate the allocation of data for this new struct.