1. **True/False.** Circle the appropriate choice.

   (a) **T** At most one operand of an x86 assembly instruction can be a memory address
   (b) **F** At most one operand of an x86 assembly instruction can be a register.
   (c) **F** A program written directly in machine language (binary) would run faster than the same program written in assembly code and then assembled.
   (d) **T** The jne operation depends on the zero flag.
   (e) **T** In C, given the declaration ‘`int num[10]`’, the use of the name ‘`num`’ in the program corresponds to an address.
   (f) **T** C has no built-in boolean type, rather the value 0 is used to represent false and all other values represent true.
   (g) **F** EBP is a caller-saved register.
   (h) **T** In two’s complement representation of binary numbers, a negative number will always have a 1 in the leftmost bit.
   (i) **F** In x86 assembly using the calling convention discussed in class, the first parameter to a procedure can be found at the address computed by adding the value of the ebp register and 12.
   (j) **T** The instruction ‘`pop ebp`’ (or ‘`pop %ebp`’) is equivalent to ‘`mov ebp,[esp]`’ followed by ‘`add esp,4`’ (or ‘`mov (%esp),%ebp`’ followed by ‘`add $4,%esp`’)

2. **Short Answer.** Write the answers in the space provided.

   (a) What does the following C code print?

   ```c
   int main()
   {
       int *p = (int *) 100;
       char *c = (char *) p;
       p++;
       c++;
       printf("p = %d, c = %d\n", p, c);
   }
   ```

   **Answer:** p = 104, c = 101

   (b) What value ends up in EAX after the following code is executed?

   ```assembly
   mov eax,0xff
   xor eax,0xf0
   shr eax,2
   shl eax,4
   ```

   xor is the bitwise exclusive-or operator.

   **Answer:** 48 (decimal) or 110000 (binary) or 0x30 (hex)
3. Write your answer on this page.

Suppose you are writing x86 code that conforms to the calling convention discussed in class. Suppose also that you have already written the following code.

```assembly
.globl _foo
_faces:
push %ebp
mov %esp,%ebp
mov ebx,10
mov ecx,5
mov edx,0
top:
cmp ecx,0
jle done
imul ebx, [ebp+8]
call _bar
add esp, 4
add edx, eax
dec ecx
jmp top
done:
mov eax, edx
pop ebp
ret
```

Your program doesn’t work however, and you suspect that it has to do with your not saving registers. Mark the above code, indicating what instructions you would add to the code, and where, to save registers in a manner consistent with the calling convention.
Answer: The code, with the appropriate instructions inserted (shown underlined) would be the following:

```
.globl _foo
foo:
push ebp
mov ebp,esp
push ebx
mov ebx,10
mov ecx,5
mov edx,0
top:
cmp ecx,0
jle done
push ecx
push edx
imul ebx,[ebp+8]
push ebx
call _bar
add esp,4
pop edx
pop ecx
add edx,eax
dec ecx
jmp top
done:
mov eax,edx
pop ebx
pop ebp
ret
```


(a) Translate the following C code into x86 assembly code. Try to make it fairly efficient (e.g. avoiding imul and idiv).

```c
int foo(int x, int y)
{
    int i;  // a local variable, not a global variable
    i = (x*8) + y;  // computing
    return i / 4;  // integer division
}
```
Answer:

.globl _foo
_foo:
push ebp
mov ebp,esp
sub esp,4
mov eax,[ebp+8]
shl eax,3
add eax,[ebp+12]
mov [ebp-4],eax
mov eax,[ebp-4]
shr eax,2
add esp,4
pop ebp
ret

.globl foo
_foo:
push %ebp
mov %ebp,%esp
sub $4,%esp #make room for i
mov 8(%ebp),%eax #load x into eax
shl $3,%eax #multiply by 8 (shift)
add 12(%ebp),%eax #add y
mov %eax, -4(%ebp) #store into i
mov -4(%ebp),%eax #load i (not necessary)
shr $2,%eax #divide by 4, leave in eax
add $4,%esp #remove i
pop %ebp
ret

(b) Draw the state of the stack right after the assembly code corresponding to the statement that has the comment “computing” is executed. Label each item on the stack in a way that makes it clear what that item is and be sure to show the stack pointer and the base pointer.

Answer:

5. Put your answers in the blue book.

As you know from 102, a binary tree is a structure that looks like this:
(a) Define in C a type NODE that can serve as the node of a binary tree, where each node contains string called name and an integer called age.

**Answer:**

```c
typedef struct node {
    char name[80];
    int  age;
    struct node *left, *right;
} NODE;
```

(b) Write a C procedure with the signature

```c
NODE *create_node(char *name, int age);
```

that creates a node that has the specified name and age. This node will have no children yet.

**Answer:**

```c
NODE *create_node(char *name, int age)
{
    NODE *n = (NODE *) malloc(sizeof(NODE));
    int i;
    for(i=0;name[i]!=0;i++)
        n->name[i] = name[i];
    n->name[i] = 0;
    n->age = age;
    n->left = NULL;
    n->right = NULL;
    return n;
}
```

(c) Write a C procedure with the signature

```c
void insert_left_child(NODE *n, NODE *root);
```

that inserts the node n into the tree as the left child of the root node. The existing left child of the root should become the left child of n. In other words, the tree should change as follows.

![Diagram of tree modification](image-url)

(a) Write in x86 assembly a two instruction sequence that would place the value of $2^x$ in the EAX register, for any integer $x$ between 0 and 31 (note: assume $x$ is a constant, not a variable).

Answer:

```
    mov eax,1
    mov $1,%eax
    shl eax,x
    shl $x,%eax
```

(b) Given a value $y$ in the EAX register, write some assembly code that would place $\lfloor \log_2 y \rfloor$ in the ECX register (Hint: Consider the relationship between $x$ and the result placed in the EAX register in the previous question).

Answer:

```
    mov ecx,0
    mov $0,%ecx
    top:
    cmp eax,1
    cmp $1,%eax  # eax cannot be 0
    je done
    shr eax,1
    shr $1,%eax
    inc ecx
    inc %ecx
    jmp top
    done:
    done:
```