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

(a) **T** **F** At most one operand of an x86 assembly instruction can be an memory address.
(b) **T** **F** At most one operand of an x86 assembly instruction can be a register.
(c) **T** **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** **F** In C, a formal parameter that is an array of strings could have the type `char **`.
(e) **T** **F** In C, given the declaration “`int num[10]`”, the use of the name “`num`” in the program corresponds to an address.
(f) **T** **F** C has no built-in boolean type, rather the value 0 is used to represent false and all other values represent true.
(g) **T** **F** ebp is a caller-saved register.
(h) **T** **F** In x86 assembly, local variables in a procedure are generally declared in the `.data` section.
(i) **T** **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** **F** 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:** __________

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

```
# Intel Syntax     # AT&T Syntax
mov eax, 0xff    mov $0xff, %eax
xor eax, 0xf0    xor $0xf0, %eax
shr eax, 2      shr $2, %eax
shl eax, 4      shl $4, %eax
```

xor is the bitwise exclusive-or operator (i.e. it produces a 1 bit in the result if either corresponding bit in the two operands is 1, but not both).

**Answer:** __________
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
.intel_syntax #AT&T syntax
.globl _foo

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

Pop ebp
call bar
add esp, 4
```

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.


(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
}
```

(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.

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 a string called name and an integer called age.

(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.

(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.


(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).

(b) Given a value \(y\) in the EAX register, write some assembly code that would place \([\log_2 y]\) in the ECX register (Hint: Consider the relationship between \(x\) and the result placed in the EAX register in the previous question).
7. Put your answers in the blue book. Given the following C procedure

```c
int sum(int *a, int num)
{
    int i;
    int sum = 0;
    for(i=0; i<num; i++)
        sum = sum + a[i];
    return sum;
}
```

using the appropriate x86 addressing modes, fill in the missing code in the assembly procedure below that corresponds to the procedure `sum()`, above. Choose either Intel or AT&T syntax.

```
_sum:                     _sum:
    push ebp              push %ebp
    mov ebp,esp           mov %esp,%ebp

    mov ecx,[ebp+8]       mov 8(%ebp),%ecx  #ecx holds a
    mov edx,[ebp+12]      mov 12(%ebp),%edx #edx holds num
    mov eax,0             mov $0,%eax      #eax holds sum
    mov ebx,0             mov $0,%ebx      #ebx holds i

    TOP:                   TOP:
        #MISSING CODE HERE

        jge OUT          jge OUT  #jump out if i >= num
        #MISSING CODE HERE

        inc ebx          incl %ebx  #increment i
        jmp TOP         jmp TOP    #jump to top of loop

    OUT:                   OUT:
        #MISSING CODE HERE

    pop ebp               pop %ebp
    ret                   ret
```

4