1. **True/False.** Circle the appropriate choice on this sheet (there are no trick questions).

(a) **T** **F** The x86 registers (eax, ebx, etc.) are part of main memory (RAM).

(b) **T** **F** In C, a formal parameter that is an array of strings could have the type `char **`.

(c) **T** **F** In x86 assembly, `call f` pushes the return address on the stack and jumps to the code at the label `f`.

(d) **T** **F** In x86 assembly, global variables are allocated on the stack and local variables reside in the data segment.

(e) **T** **F** CPU and main memory are connected by a bus.

(f) **T** **F** A procedure is free to overwrite a caller-saved register without saving it first.

(g) **T** **F** In C, the expression `p->x` is equivalent to `(*p).x`.

(h) **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 (assuming a 32-bit machine).

(i) **T** **F** In x86 assembly, if `f` is a recursive procedure that takes two parameters, then when it calls itself, it doesn’t need to push any more parameters on the stack.

(j) **T** **F** In x86 assembly, `pop eax` (in AT&T syntax) is equivalent to `mov %eax,-4(%esp)` followed by `add %esp,4` (in AT&T syntax, `mov (%esp),%eax` followed by `add $4,%esp`).

2. The assembly code that I wrote corresponding to the C procedure

```c
void foo(int x, int y) { int z = x + y; z = z - 7; return z; }
```

is

<table>
<thead>
<tr>
<th>#Intel Syntax</th>
<th>#AT&amp;T Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>.foo:</td>
<td>.foo:</td>
</tr>
<tr>
<td>push ebp</td>
<td>push %ebp</td>
</tr>
<tr>
<td>mov ebp,esp</td>
<td>mov %esp,%ebp #DRAW THIS</td>
</tr>
<tr>
<td>sub esp,4</td>
<td>sub $4,%esp</td>
</tr>
<tr>
<td>mov eax,[ebp+8]</td>
<td>mov 8(%ebp),%eax</td>
</tr>
<tr>
<td>add eax,[ebp+12]</td>
<td>add 12(%ebp),%eax</td>
</tr>
<tr>
<td>mov [ebp-4],eax</td>
<td>mov %eax,-4(%ebp) #DRAW THIS</td>
</tr>
<tr>
<td>sub DWORD PTR [ebp-4],7</td>
<td>subl $7,-4(%ebp)</td>
</tr>
<tr>
<td>mov eax,[ebp-4]</td>
<td>mov -4(%ebp),%eax</td>
</tr>
<tr>
<td>pop ebp</td>
<td>pop %ebp #DRAW THIS</td>
</tr>
<tr>
<td>ret</td>
<td>ret</td>
</tr>
</tbody>
</table>

(a) Assume `foo(13,14)` is called from `main()`. Draw the state of the stack, showing the values stored in the stack frame for `foo` and where ESP and EBP point, **after** the execution of each instruction that has the comment “DRAW THIS”. Since there are three such instructions, you should draw the stack three times.
(b) My program crashes during the execution of foo. Why is that? How should I fix it?

3. In C, suppose you have a binary tree whose node type is defined by the following:

```c
typedef struct node {
    int value;
    struct node *left;
    struct node *right;
} NODE;
```

(a) Below is my version of the procedure that creates a new node and returns a pointer to it, so that the new node can be inserted into the binary tree. This code compiles without errors, but doesn’t execute correctly.

```c
NODE *new_node(int val)
{
    NODE n;
    n.left = NULL;
    n.right = NULL;
    n.value = val;
    return &n;
}
```

Explain what I am doing wrong.

(b) Write the correct code for `new_node`.

(c) Write a C procedure

```c
void insert_left(NODE *p);
```

that inserts the node pointed to by `p` as the leftmost leaf of the tree. Assume that the global variable `root` points to the root of the tree (but don’t assume that `root` is not NULL).

4. Write a C procedure with the signature

```c
int intlog(int x);
```

that computes \( \lfloor \log x \rfloor \) without using * (multiply) or / (divide). That is, it returns the largest integer no greater than \( \log x \) (i.e. the log, base 2, of \( x \)). For example, \( \text{intlog}(56) \) returns 5, since \( 5 \leq \log 56 < 6 \). You can assume that \( x > 0 \).

5. Write, in x86 assembly, a procedure which would have the C signature

```c
int max(int a[], int size);
```

and returns the value of the largest element of the array `a`, whose size is the second parameter to `max`. 