

Your Name(s)/Lab Section(s):

**Question 1:** Convert the following C code fragment to equivalent IA32 assembly code in two steps.

(1) First, translate the loop to its equivalent C goto version

(2) Next, translate your C goto version to IA32, assuming that `dog` is at  $-4(r[\%ebp])$ , `cat` is at  $-8(r[\%ebp])$ , and `goat` is at  $-12(r[\%ebp])$ .

You must show both steps (1) and (2). To receive partial credit, annotate your IA32 code with comments describing which part of the C code you are implementing.

```
int dog, cat, goat;
dog = 12;
cat = 90;
goat = dog - cat;
while (dog < cat) {
    dog *= 2;
    goat += dog;
}
```

(1) C goto version

(2) IA32 Translation

-----

-----

## Question 2

Trace through the following IA32 code. Show the contents of the given memory and registers right before the instruction at point A is executed. Assume the `addl` instruction in `main` that is immediately after the `call` instruction is at memory address `0x1234`. Hints:

- remember to start execution in `main`.
- `%esp` points to the item on the top of the stack, so a `push` will grow the top of the stack and then move in the pushed value. A `pop` will move the value on top of the stack and then shrink the stack.
- The sequence of instructions `leave; ret` is equivalent to the sequence `movl %ebp, %esp; popl %ebp; popl %eip`.

```
func:
    pushl %ebp
    movl  %esp, %ebp
    subl  $16, %esp
    movl  8(%ebp), %eax
    addl  %eax, %eax
    movl  %eax, -4(%ebp)
    movl  -4(%ebp), %eax
    leave # point "A"
    ret

main:
    pushl %ebp
    movl  %esp, %ebp
    subl  $16, %esp
    movl  $6, -4(%ebp)
    pushl -4(%ebp)
    call  func
    addl  $4, %esp # at 0x1234
    movl  %eax, -4(%ebp)
    movl  $0, %eax
    leave
    ret
```

Memory Address	value at "A"
0x8880	
0x8884	
0x8888	
0x888c	
0x8890	
0x8894	
0x8898	
0x889c	
0x88a0	
0x88a4	
0x88a8	
0x88ac	
0x88b0	
0x88b4	
0x88b8	
0x88bc	
0x88c0	

Register	initial value	value at point "A"
<code>%eax</code>	2	
<code>%edx</code>	3	
<code>%esp</code>	0x88b0	
<code>%ebp</code>	0x88c0	