Name: ________________

Directions: **Work only on this sheet** (on both sides, if needed); do not turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing.

1. (10) Give a single assembly language instruction equivalent to

```
popl %ecx
popl %ecx
popl %ecx
```

assuming that we do not care what the popped values actually are.

2. (10) When the call `scanf("%d%d%d",&u,&v,&w)` is compiled, how many push operations will appear before the CALL instruction?

3. (10) Say you are running some program on CSIF that makes use of a special library in your own home directory, say `/home/thisisme/`. What command should you run to enable the OS to find that library when you execute the program?

4. (10) List the Intel-specific registers (using their official Intel names) whose values are affected when a RET instruction executes.

5. This problem concerns the code in pp.137-139. Suppose we were to change things so that `addone()` would have (as viewed as a function callable from C) the signature

```c
int addone(int x)
```

as opposed to what was in the version in the book,

```c
void addone(int *x)
```

The function will now return the value of its argument plus 1.

(a) (30) Fill in the gap in the revised version of `addone()`:

```c
.text
.globl addone
addone:
    # insert at most 4 instructions here
ret
```

(b) (30) Suppose the call in `TryAddOne.c` is now wrapped inside a print call:

```c
printf("%d
",addone(x));
```

I ran the new code through `gcc -S`, an excerpt of which appears below. Fill in the gaps.

```
movl $7, x
movl x, %eax
movl %eax, (%esp)  # gap 1
call addone
movl $LCO, (%esp)  # gap 2
call printf
```

Solutions:

1. addl $12, %esp
2. 4
3. setenv LD_LIBRARY_PATH /home/thisisme
4. ESP, EIP
5.a

```
.text
.globl addone
addone:
    movl 4(%esp), %eax
    incl %eax
    ret
```

5.b

```
movl $7, x
movl x, %eax
movl %eax, 4(%esp)
call addone
movl $.LC0, (%esp)
call printf
```