CS 354: Intro to Computer Systems (Spring 2018)

Assembly - Worksheet 2

1. Stack Pointer

```
Instruction: pushl S
Description: Push double word
Effect:
R[%esp]← R[%esp] - 4;
M[R[%esp]]← S
```

```
Instruction: popl D
Description: Pop double word
Effect:
D ← M[R[%esp]];
R[%esp]← R[%esp] + 4
```

Assume the values in %eax, %edx, and %esp are 0x123, 0, and 0x108 respectively. What are the values of the registers after the following instructions.

	%eax	%edx	%esp
pushl %eax			
popl %eax			

2. Load Effective Address

```
Instruction: leal S, D
Description: Load effective address
Effect: D ← &S
```

Suppose register %eax holds value x and %ecx holds value y. Fill in the table below with formulas indicating the value that will be stored in register %edx for each of the given assembly code instructions: [From CSAPP: 3.6]

Instruction	Result
leal 6(%eax), %edx	
leal (%eax,%ecx), %edx	
leal (%eax,%ecx,4), %edx	
leal 7(%eax,%eax,8), %edx	
leal 0xA(,%ecx,4), %edx	
leal 9(%eax,%ecx,2), %edx	

call-3.S

```
func:
# put return value into stack location


ret


main:
# grow stack: for return value

call func
# move return value into eax


# reduce stack
```

call-4.S

```
# func should add two args together
# put return value into eax
func:



# grow stack; put 2 args into top
# call func
# reduce stack now; ret val in eax
main:
```

call-5.S

```
# make room for local int
# get two params and add together
# reduce stack (free local int)
func:



# grow stack; put 2 args into top
# call func
# reduce stack
main:
```

call-6.S

```
# save old ebp, set new one
# alloc local int + init to 1
# add params, local var into  eax
# free local vars, restore ebp
func:



# grow stack; put 2 args into top
# call func
# reduce stack
main:
```

call-7.S

```
# save old ebp, set new one
# alloc local int + init to 1
# add params, local var into  eax
# free local vars, restore ebp
func:



# set eax to some value
# save eax (caller save)
# grow stack; put 2 args into top
# call func
# reduce stack
# save retval into ebx
# restore eax
main:
```

assembly-function-1.c

```
int increment(int x) {
    return x + 1;
}

// implement this in assembly
int my_increment(int);

int main() {
    int f = 10;
    printf("value within main %d\n", f);
    f = my_increment(f);
    printf("last value in main %d\n", f);
    return 0;
}
```

assembly-function-2.c

```
void increment(int* x) {
    *x = *x + 1;
}

// implement this in assembly
void my_increment(int*);

int main() {
    int f = 10;
    printf("value within main %d\n", f);
    my_increment(&f);
    printf("last value in main %d\n", f);
    return 0;
}
```