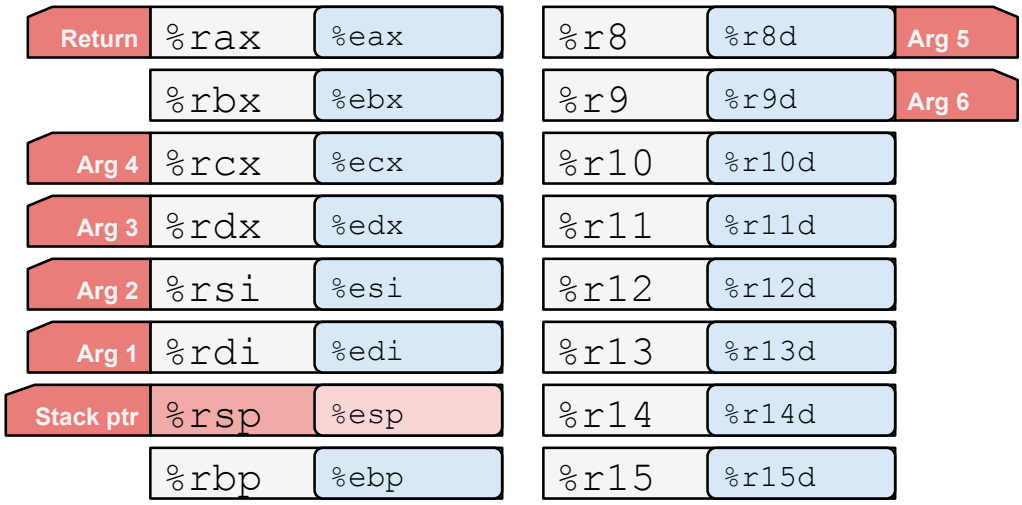
CSCI 2330 – Reverse Engineering Exercises

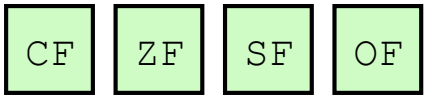
1. Translate the x86-64 instructions below into a C function, which you can assume is non-void and takes one argument. Remember to specify the appropriate types of the argument and return value. **Do not use any goto statements in your function.**

```
compute:
     movq      $0, %rax
     movq      $1, %rbx
     jmp       .L2
.L1: addq      $1, %rax
     shlq      $1, %rbx        # left shift
.L2: cmpq      %rdi, %rbx
     jl        .L1             # jump if less
     ret
```

2. Translate the x86-64 instructions below into a C function, which again is a non-void function with one argument. As before, do not use any goto statements.

```
prod10:
        movq    %rdi, %rax
        movl    $1, %ecx
        movl    $0, %esi
        jmp     .L2
.L3:    movl    (%rax), %edx
        imull   %edx, %ecx
        addq    $4, %rax
        addl    $1, %esi
.L2:    movl    %esi, %edi
        cmpl    $9, %edi
        jle     .L3             # jump if less or equal to
        movl    %ecx, %eax
        ret
```





Condition codes

CF: **Carry flag** (set if carry-out bit = 1)

ZF: **Zero flag** (set if result = 0)

SF: **Sign flag** (set if result high bit = 1)

OF: **Overflow flag** (set if signed overflow)

Setting CCs

(1) `cmpq a,b` (set based on `b - a`)

(2) `testq a,b` (set based on `a & b`)

(3) arithmetic insts (implicit, all except `leaq`)

JX	Condition	Description
`jmp`	`1`	Unconditional
`je/jz`	`ZF`	Equal / Zero
`jne/jnz`	`~ZF`	Not Equal / Not Zero
`js`	`SF`	Negative
`jns`	`~SF`	Nonnegative
`jg`	`~(SF^OF)&~ZF`	Greater (Signed)
`jge`	`~(SF^OF)`	Greater or Equal (Signed)
`jl`	`(SF^OF)`	Less (Signed)
`jle`	`(SF^OF)|ZF`	Less or Equal (Signed)
`ja`	`~CF&~ZF`	Above (unsigned)
`jb`	`CF`	Below (unsigned)