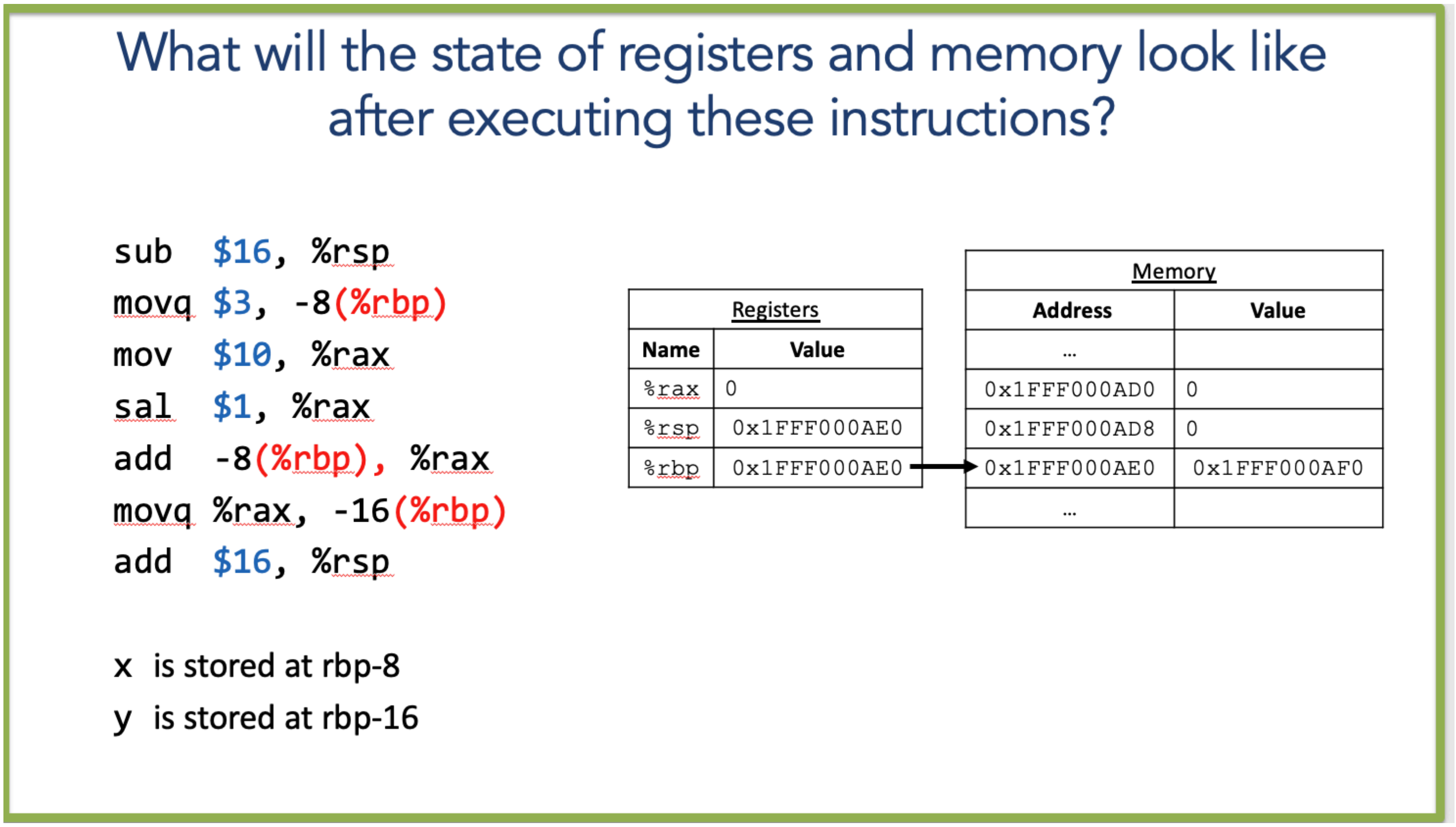
CS31 Worksheet: Week 4: Assembly

Q1. Explain the following assembly instructions in words:

```
mov %rax, %r15
add %r9,  %rdx

mov $10, %rax

add $0xF, %rdx

mov $20, (%rax)

movl %rax, -16(%rbp)
```

Q2. Let's try some more examples:



What will the state of registers and memory look like after executing these instructions?

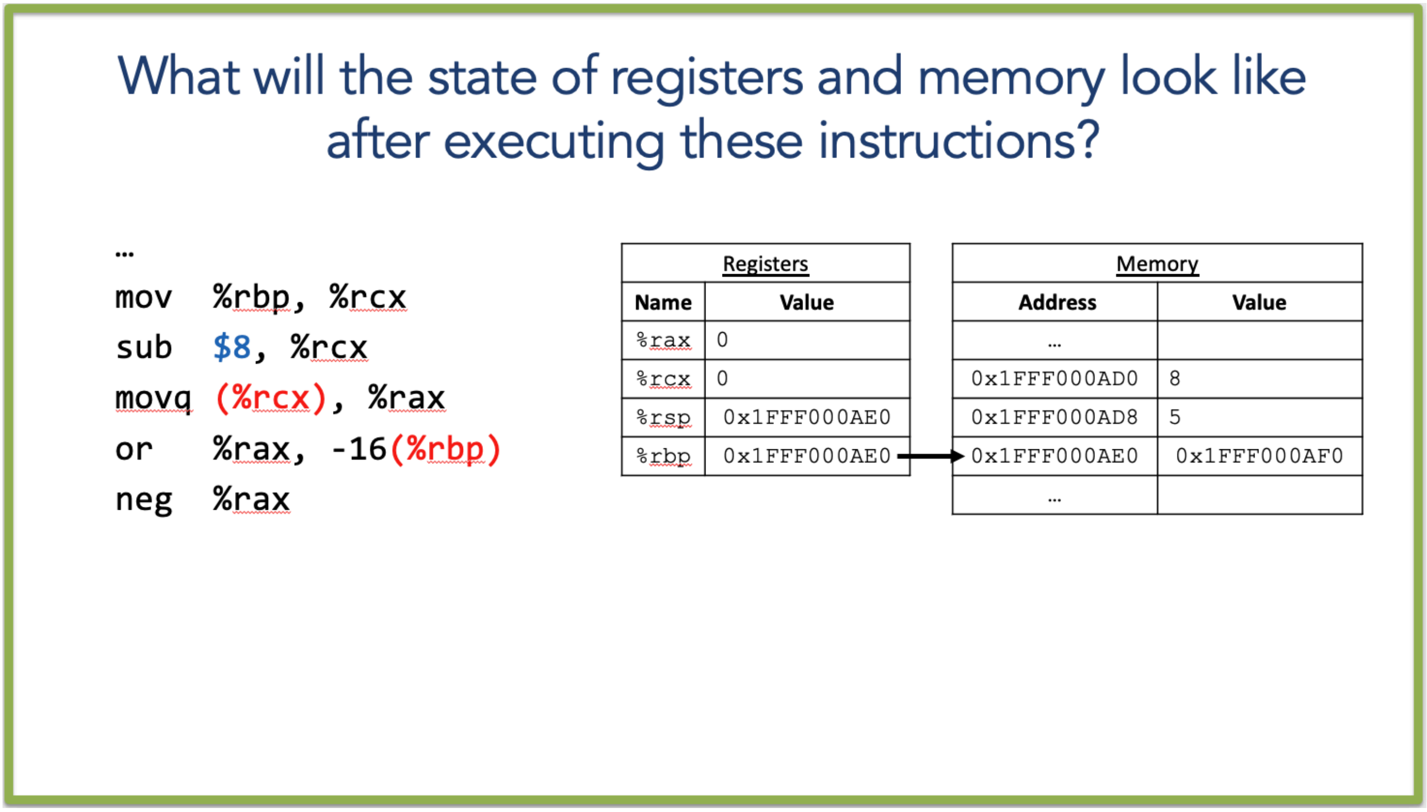
```
sub   $16, %rsp
movq  $3, -8(%rbp)
mov   $10, %rax
sal   $1, %rax
add   -8(%rbp), %rax
movq  %rax, -16(%rbp)
add   $16, %rsp
```

Registers	
Name	**Value**
%rax	0
%rsp	0x1FFF000AE0
%rbp	0x1FFF000AE0

Memory	
Address	**Value**
…	
0x1FFF000AD0	0
0x1FFF000AD8	0
0x1FFF000AE0	0x1FFF000AF0
…	

x is stored at rbp-8

y is stored at rbp-16

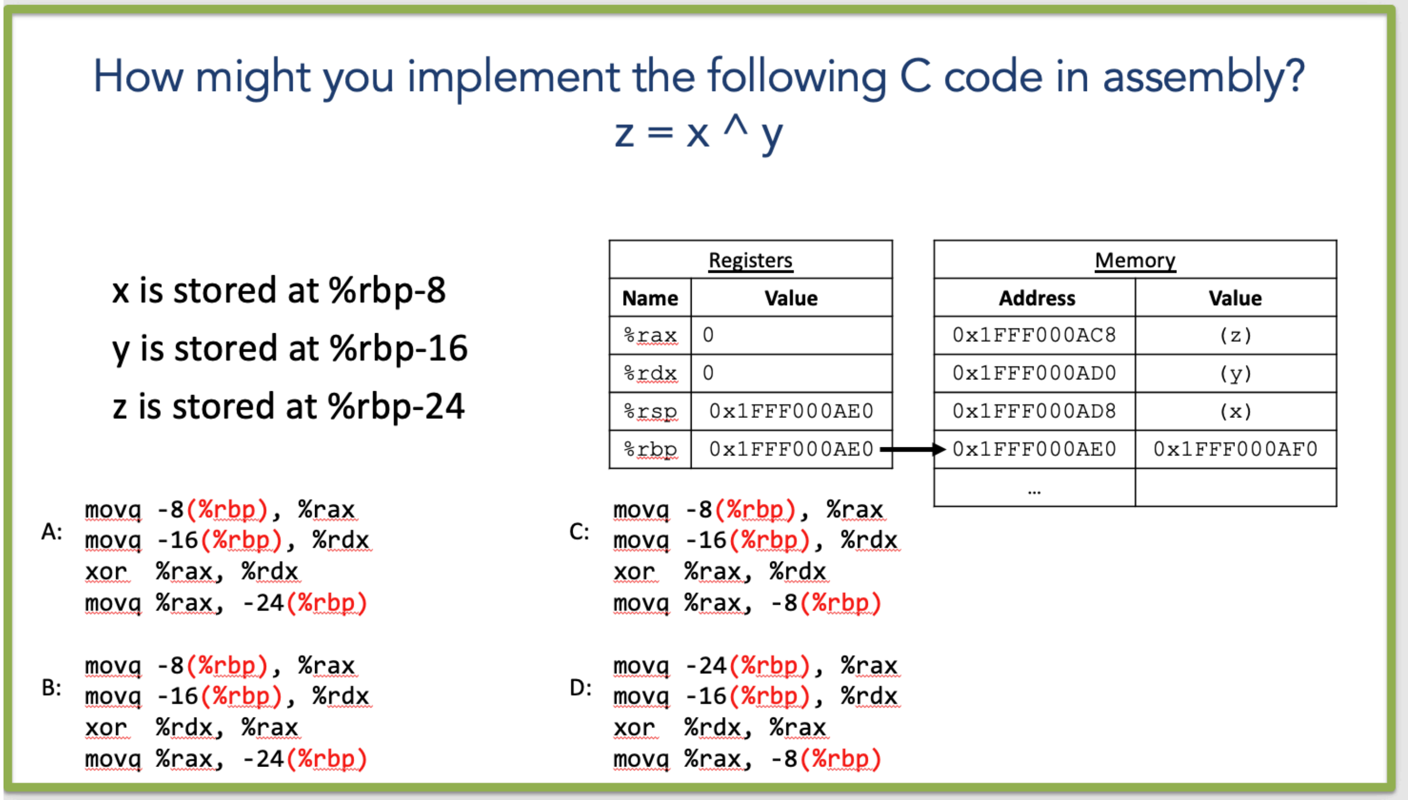


What will the state of registers and memory look like after executing these instructions?

```
…
mov   %rbp, %rcx
sub   $8, %rcx
movq  (%rcx), %rax
or    %rax, -16(%rbp)
neg   %rax
```

Registers	
Name	**Value**
%rax	0
%rcx	0
%rsp	0x1FFF000AE0
%rbp	0x1FFF000AE0 →

Memory	
Address	**Value**
…	
0x1FFF000AD0	8
0x1FFF000AD8	5
0x1FFF000AE0	0x1FFF000AF0
…	



How might you implement the following C code in assembly?
z = x ^ y

x is stored at %rbp-8

y is stored at %rbp-16

z is stored at %rbp-24

Registers	
Name	**Value**
%rax	0
%rdx	0
%rsp	0x1FFF000AE0
%rbp	0x1FFF000AE0 →

Memory	
Address	**Value**
0x1FFF000AC8	(z)
0x1FFF000AD0	(y)
0x1FFF000AD8	(x)
0x1FFF000AE0	0x1FFF000AF0
…	

A:
```
movq -8(%rbp), %rax
movq -16(%rbp), %rdx
xor  %rax, %rdx
movq %rax, -24(%rbp)
```

B:
```
movq -8(%rbp), %rax
movq -16(%rbp), %rdx
xor  %rdx, %rax
movq %rax, -24(%rbp)
```

C:
```
movq -8(%rbp), %rax
movq -16(%rbp), %rdx
xor  %rax, %rdx
movq %rax, -8(%rbp)
```

D:
```
movq -24(%rbp), %rax
movq -16(%rbp), %rdx
xor  %rdx, %rax
movq %rax, -8(%rbp)
```

How might you implement the following C code in assembly?
x = y >> 3 | x * 8

x is stored at %rbp-8

y is stored at %rbp-16

z is stored at %rbp-24

Registers	
Name	**Value**
%rax	0
%rdx	0
%rsp	0x1FFF000AE0
%rbp	0x1FFF000AE0

Memory	
Address	**Value**
0x1FFF000AC8	(z)
0x1FFF000AD0	(y)
0x1FFF000AD8	(x)
0x1FFF000AE0	0x1FFF000AF0
…	

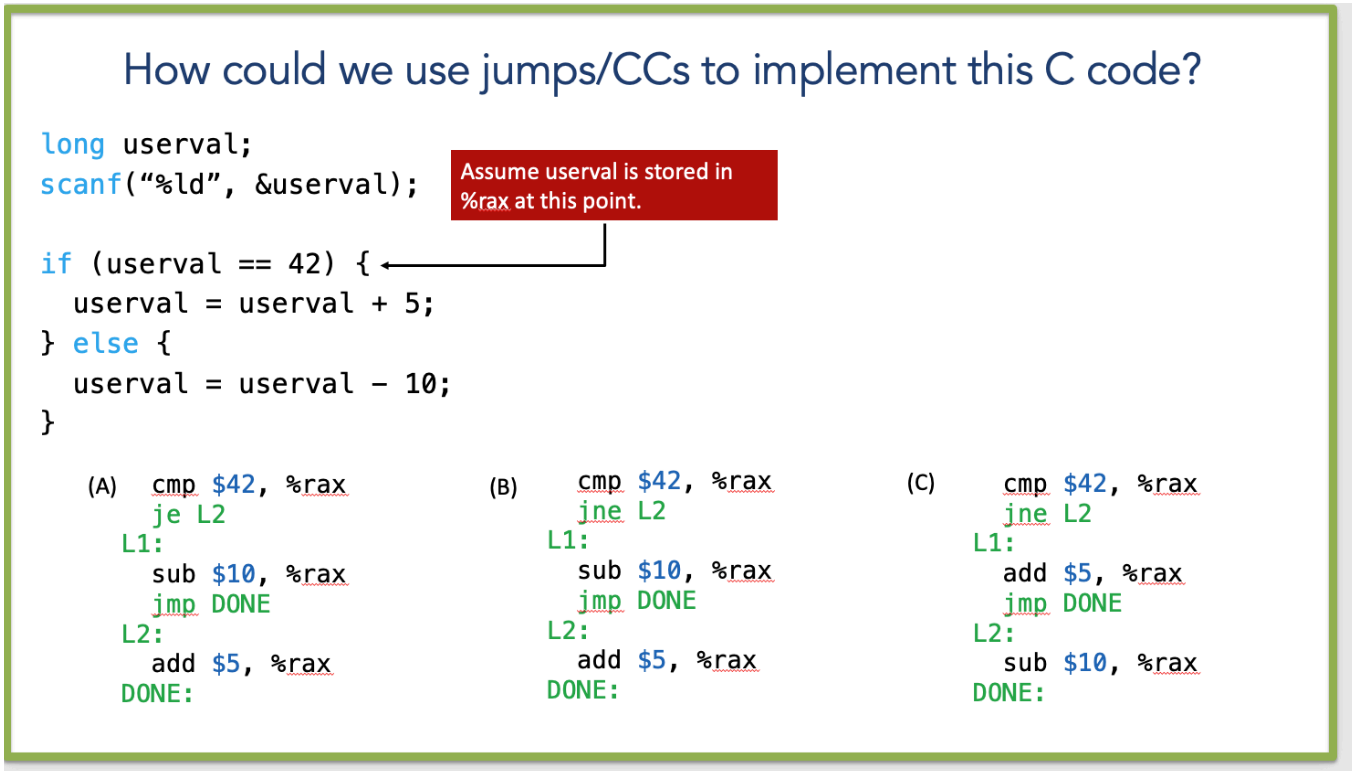
Which flags would this cmp set?

Suppose %rax holds 5, %rcx holds 7

```
cmp %rcx, %rax
```

If the result is zero (ZF)
If the result's first bit is set (negative if signed) (SF)
If the result overflowed (assuming unsigned) (CF)
If the result overflowed (assuming signed) (OF)

A. ZF
B. SF
C. CF and ZF
D. CF and SF
E. CF, SF, and CF



How could we use jumps/CCs to implement this C code?

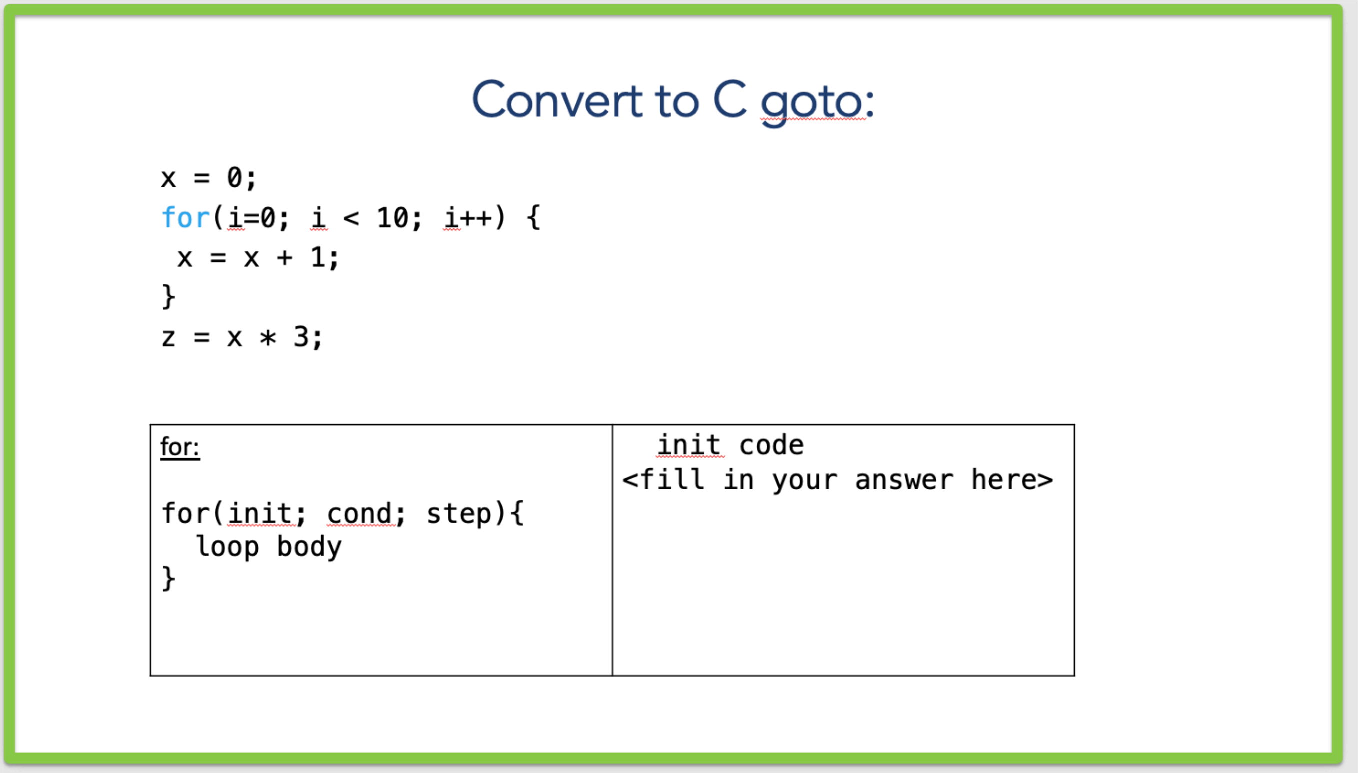
```
long userval;
scanf(“%ld”, &userval);

if (userval == 42) {
  userval = userval + 5;
} else {
  userval = userval – 10;
}
```

Assume userval is stored in %rax at this point.

(A)
```
    cmp $42, %rax
    je L2
  L1:
    sub $10, %rax
    jmp DONE
  L2:
    add $5, %rax
  DONE:
```

(B)
```
    cmp $42, %rax
    jne L2
  L1:
    sub $10, %rax
    jmp DONE
  L2:
    add $5, %rax
  DONE:
```

(C)
```
    cmp $42, %rax
    jne L2
  L1:
    add $5, %rax
    jmp DONE
  L2:
    sub $10, %rax
  DONE:
```



Convert to C goto:

```
x = 0;
for(i=0; i < 10; i++) {
 x = x + 1;
}
z = x * 3;
```

for:
for(init; cond; step){
loop body
}
	init code
	<fill in your answer here>