Recall: RFLAGS register is used for control flow

- RFLAGS contain different status flags
  - `ZF, SF, CF, OF`
- Certain instructions set status flags
  - Regular arithmetic instructions
  - Special flag-setting instructions instructions
- Instructions that read RFLAGS to…
  - set register values
  - determine value of %rip

Today's lesson plan

- Special instructions that set RFLAGS
  - Cmp, test
- Instructions that read RFLAGS to set register values
  - Set
- Instructions that (read RFLAGs to) set %rip
  - jmp

Status flags summary

flag	status
ZF (Zero Flag)	set if the result is zero.
SF (Sign Flag)	set if the result is negative.
CF (Carry Flag)	Overflow for unsigned-integer arithmetic
OF (Overflow Flag)	Overflow for signed-integer arithmetic

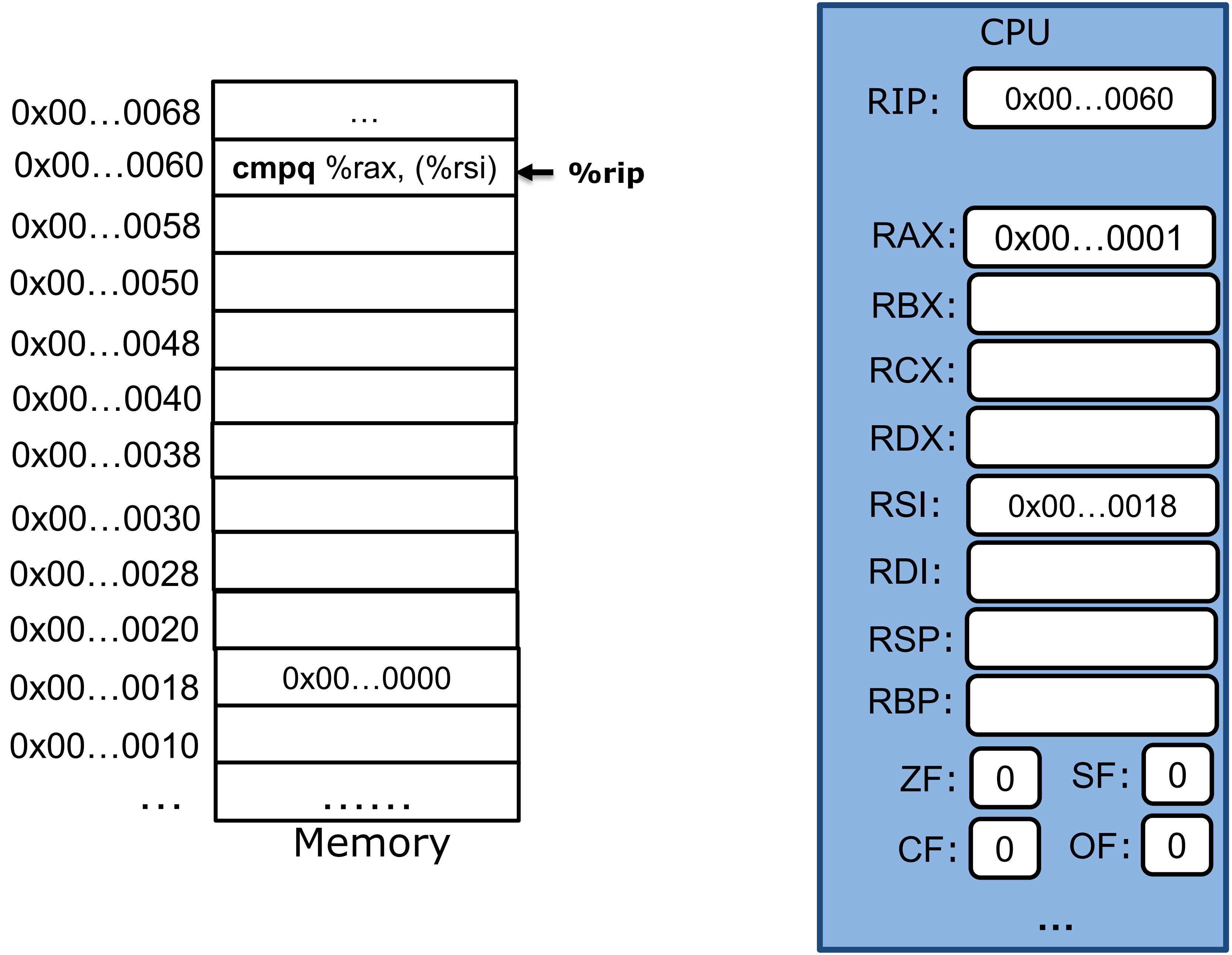
Arithmetic instructions set RFLAGS, e.g. add, inc, and, sal

`lea, mov do not set RFLAGS`

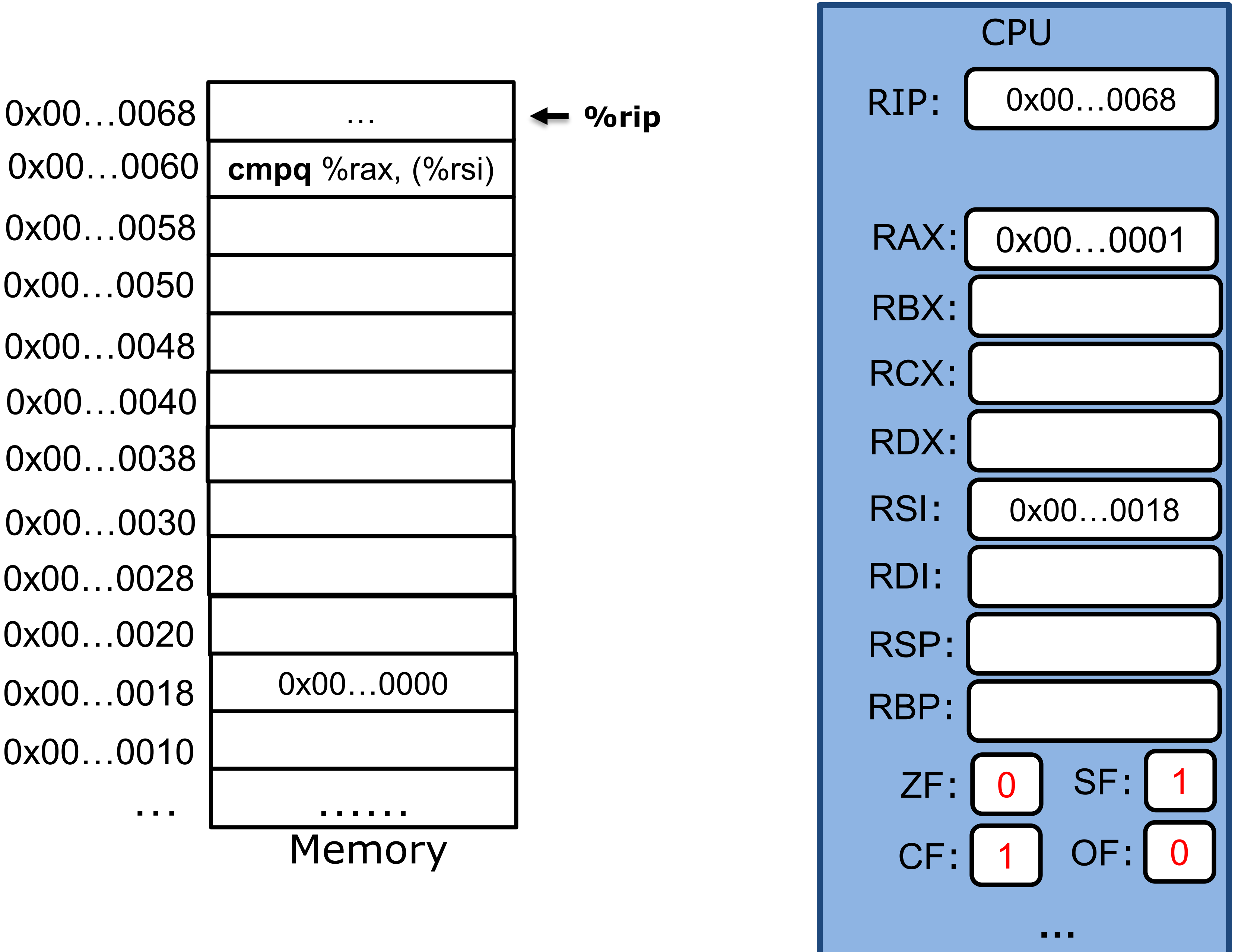
Instructions that set RFLAGS: cmp

`cmpq` `src, dst`

- Like **`subq`** `src, dst` except `dst` is unchanged
- Set CF, ZF, SF and OF appropriately



0x00…0068
…
0x00…0060
cmpq %rax, (%rsi)
%rip
0x00…0058
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0000
0x00…0010
…
……
Memory
CPU
RIP:
0x00…0060
RAX:
0x00…0001
RBX:
RCX:
RDX:
RSI:
0x00…0018
RDI:
RSP:
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…



0x00…0068
…
%rip
0x00…0060
cmpq %rax, (%rsi)
0x00…0058
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0000
0x00…0010
…
……
Memory
CPU
RIP:
0x00…0068
RAX:
0x00…0001
RBX:
RCX:
RDX:
RSI:
0x00…0018
RDI:
RSP:
RBP:
ZF:
0
SF:
1
CF:
1
OF:
0
…

Instructions that set RFLAGS: test

`testq` `src, dst`

- Like **`andq`** `src, dst` except `dst` is unchanged
- Set ZF, SF appropriately

Questions

`testq` `%rax, %rax`

- When is ZF set?
- When is SF set?

Instructions that read RFLAGS: set

setX dst

- Set dst to 1 (or 0) if condition is true (or false).
- Suffix (X) indicates which condition to test for
  - Truthfulness of condition depends on status flags in RFLAGS.
- dst is a 1-byte register or a byte in memory.

setX dst

```
cmpq a, b
setX dst
```

SF:true, OF:true ← cmpq a=0xfff…ff, b=0x7f..ff
SF:false, OF:false

setX	Condition	Description
sete	ZF	Equal / Zero
setne	~ZF	Not Equal / Not Zero
sets	SF	Negative
setns	~SF	Nonnegative
setg	~(SF^OF)&~ZF	Greater (Signed)
setge	~(SF^OF)	Greater or Equal (Signed)
setl	(SF^OF)	Less (Signed)
setle	(SF^OF)|ZF	Less or Equal (Signed)
seta	~CF&~ZF	Above (unsigned)
setb	CF	Below (unsigned)

b >= a

1 byte register

%rax	%al	%r8	%r8b
%rbx	%bl	%r9	%r9b
%rcx	%cl	%r10	%r10b
%rdx	%dl	%r11	%r11b
%rsi	%eil	%r12	%r12b
%rdi	%dil	%r13	%r13b
%rsp	%spl	%r14	%r14b
%rbp	%bpl	%r15	%r15b

1 byte

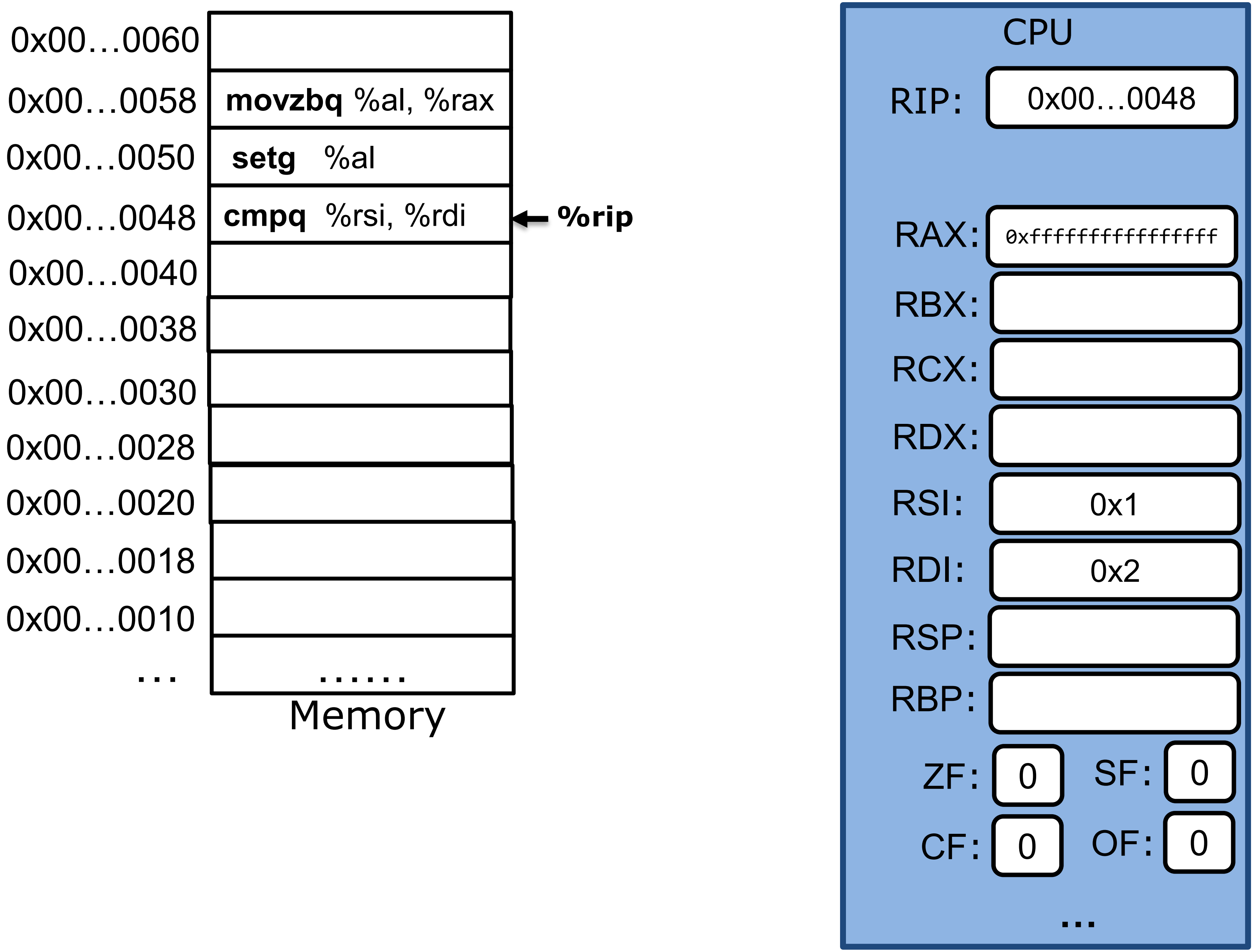
1 byte

Example

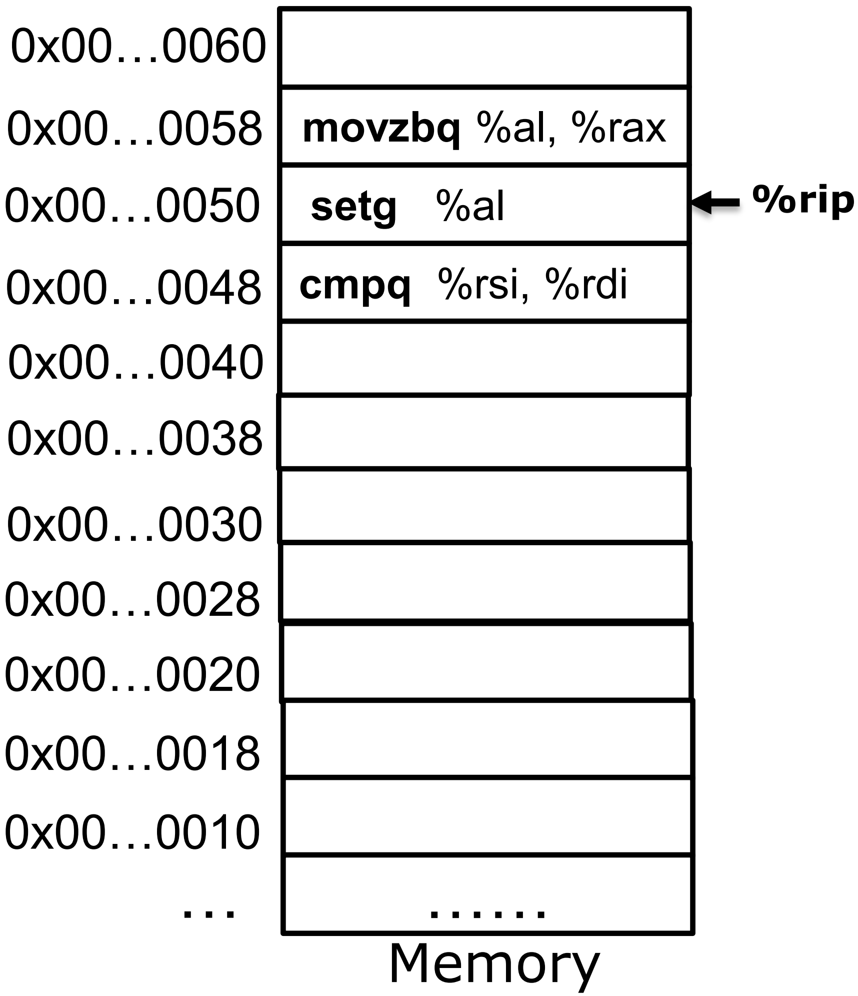
```
long gt (long x, long y)
{
  return x > y;
}
```

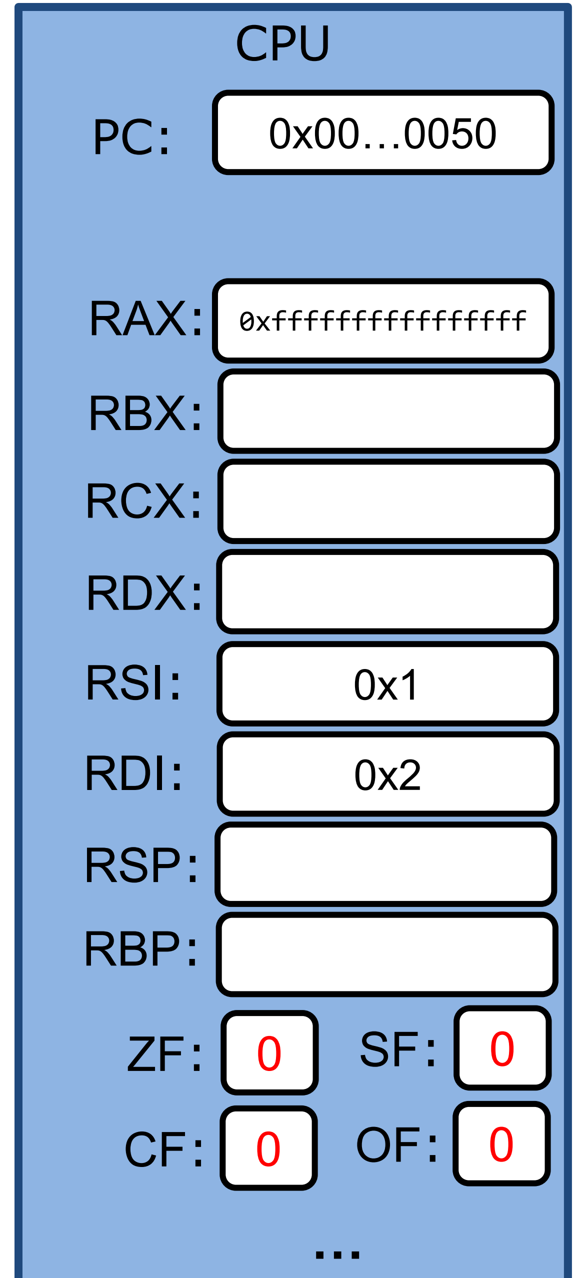
Register	Use(s)
`%rdi`	Argument **`x`**
`%rsi`	Argument **`y`**
`%rax`	Return value

```
cmpq   %rsi, %rdi    # cmpq y x
setg   %al           # set when >
movzbq %al, %rax     # zero extend %rax
```

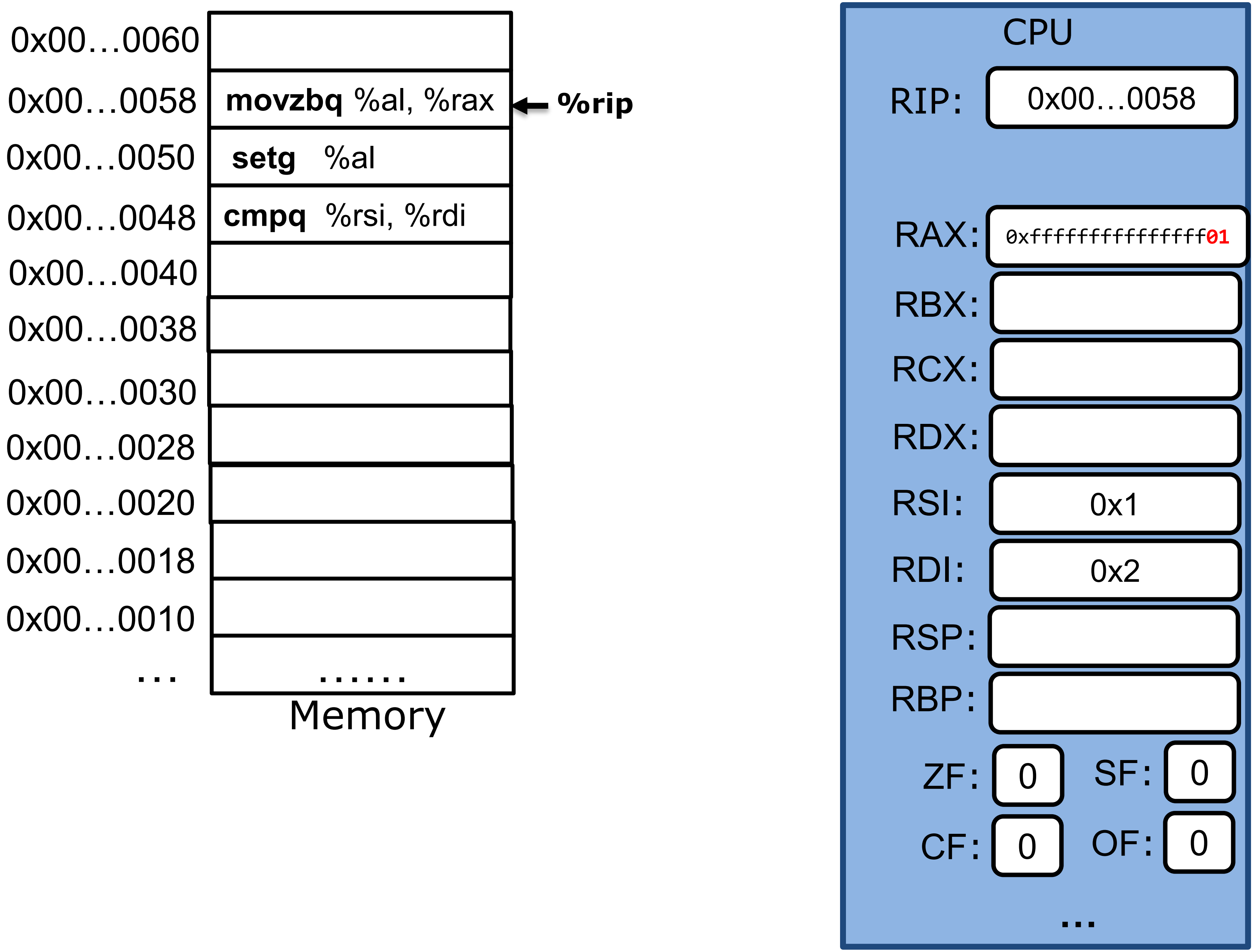


0x00…0060
0x00…0058
movzbq %al, %rax
0x00…0050
setg %al
0x00…0048
cmpq %rsi, %rdi
%rip
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
…
……
Memory
CPU
RIP:
0x00…0048
RAX:
0xffffffffffffffff
RBX:
RCX:
RDX:
RSI:
0x1
RDI:
0x2
RSP:
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…

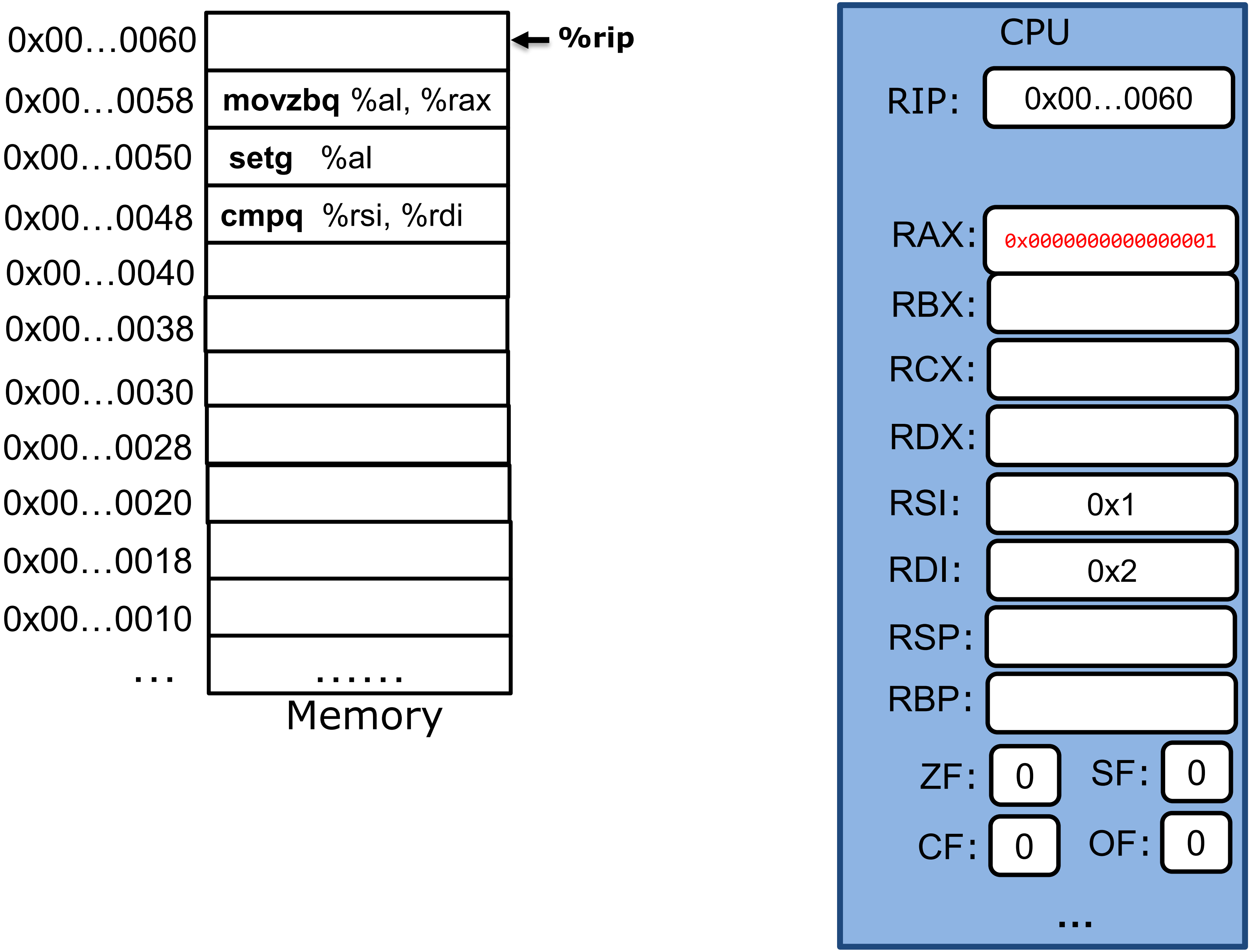

0x00…0060
0x00…0058
movzbq %al, %rax
0x00…0050
setg %al
%rip
0x00…0048
cmpq %rsi, %rdi
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
…
……
Memory


CPU
PC:
0x00…0050
RAX:
0xffffffffffffffff
RBX:
RCX:
RDX:
RSI:
0x1
RDI:
0x2
RSP:
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…

`setg`	`~(SF^OF)&~ZF`



0x00…0060
0x00…0058
movzbq %al, %rax
%rip
0x00…0050
setg %al
0x00…0048
cmpq %rsi, %rdi
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
…
……
Memory
CPU
RIP:
0x00…0058
RAX:
0xffffffffffffff01
RBX:
RCX:
RDX:
RSI:
0x1
RDI:
0x2
RSP:
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…



0x00…0060
0x00…0058
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
…
movzbq %al, %rax
setg %al
cmpq %rsi, %rdi
……
%rip
Memory
CPU
RIP:
0x00…0060
RAX:
0x0000000000000001
RBX:
RCX:
RDX:
RSI:
0x1
RDI:
0x2
RSP:
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…

Today’s lesson plan

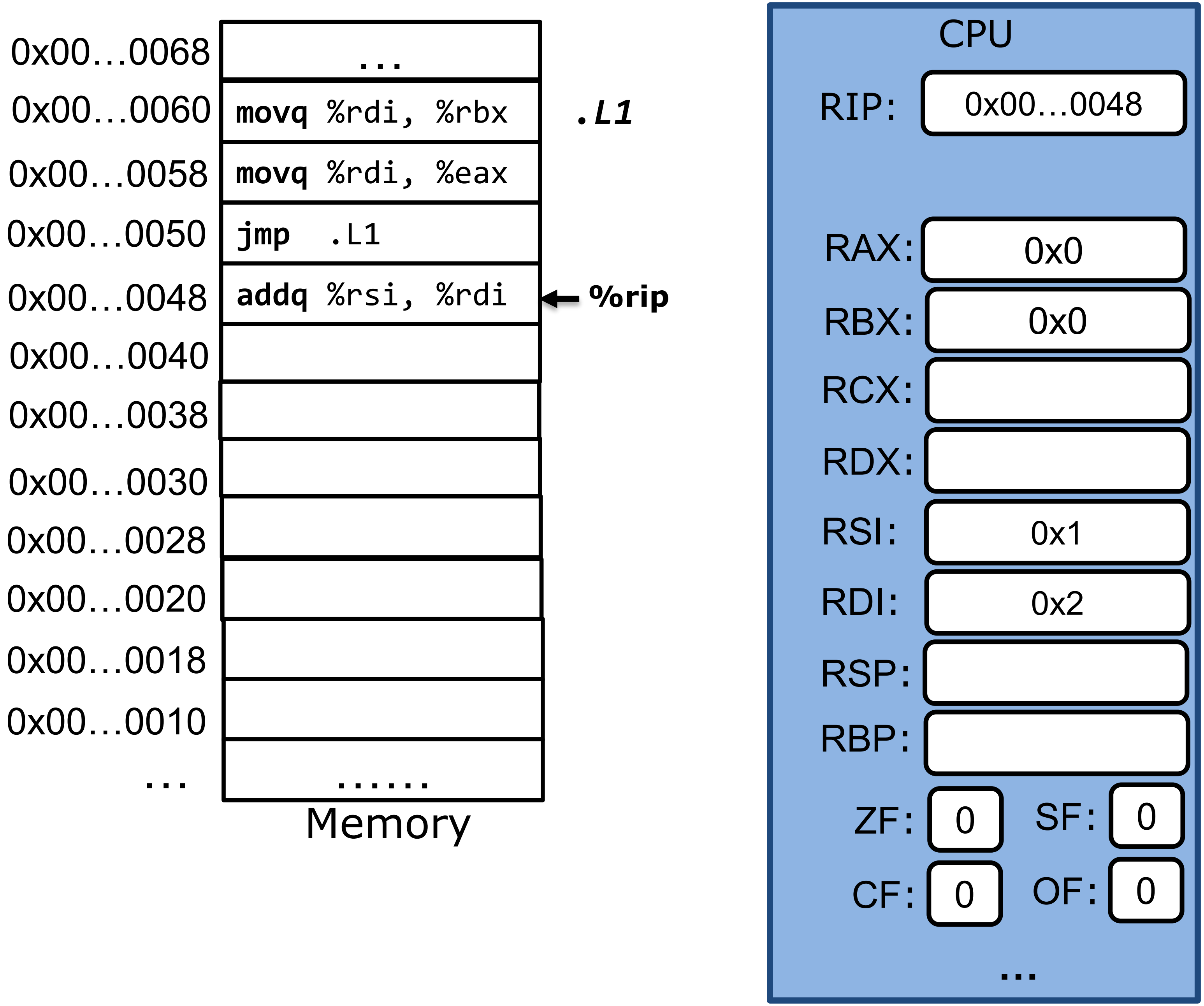
- Special instructions that set RFLAGS
  - Cmp, test
- Instructions that read RFLAGS to set register values
  - Set
- Instructions that (read RFLAGs to) set %rip
  - jmp

(Unconditional) jump instruction

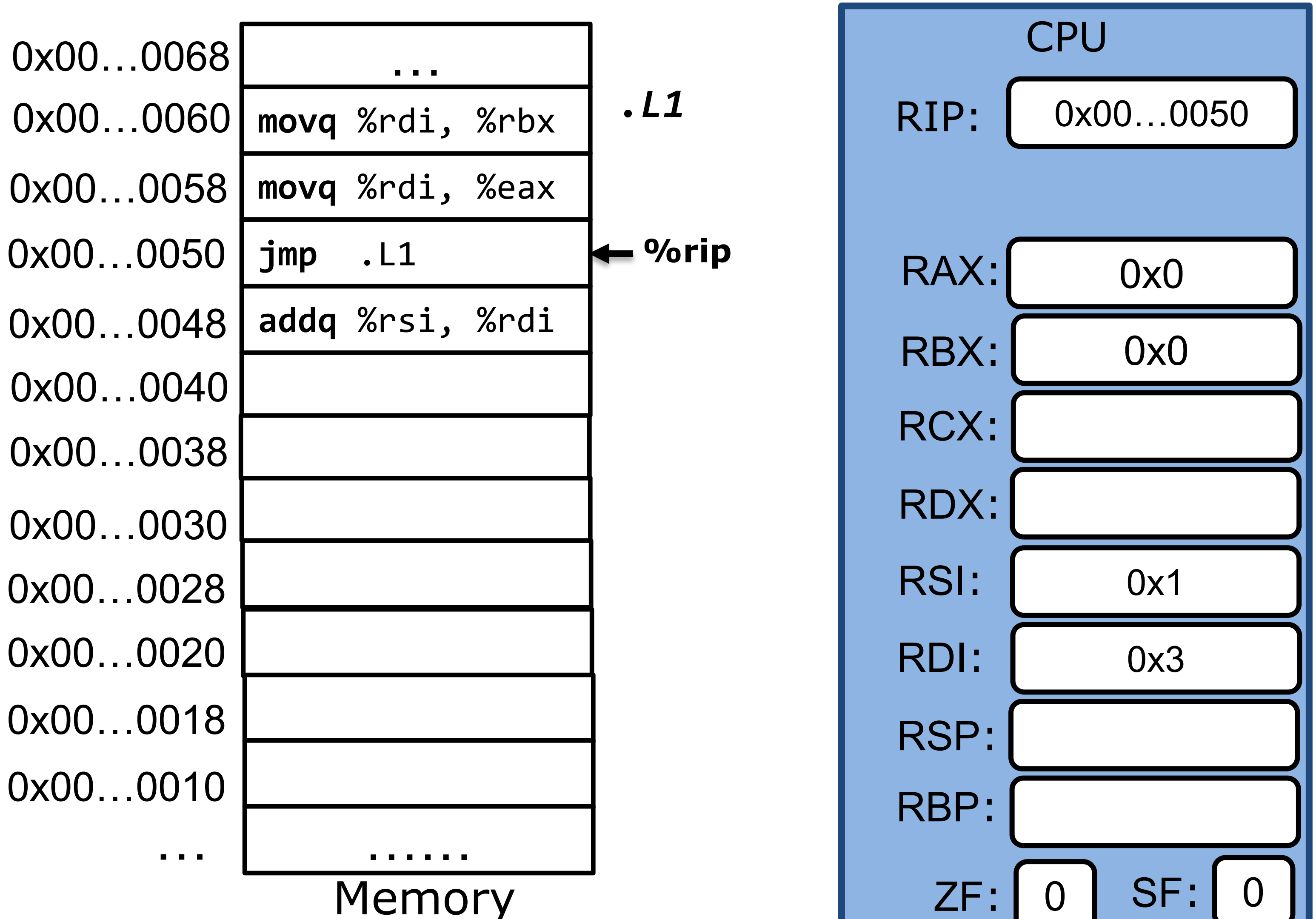
jmp label

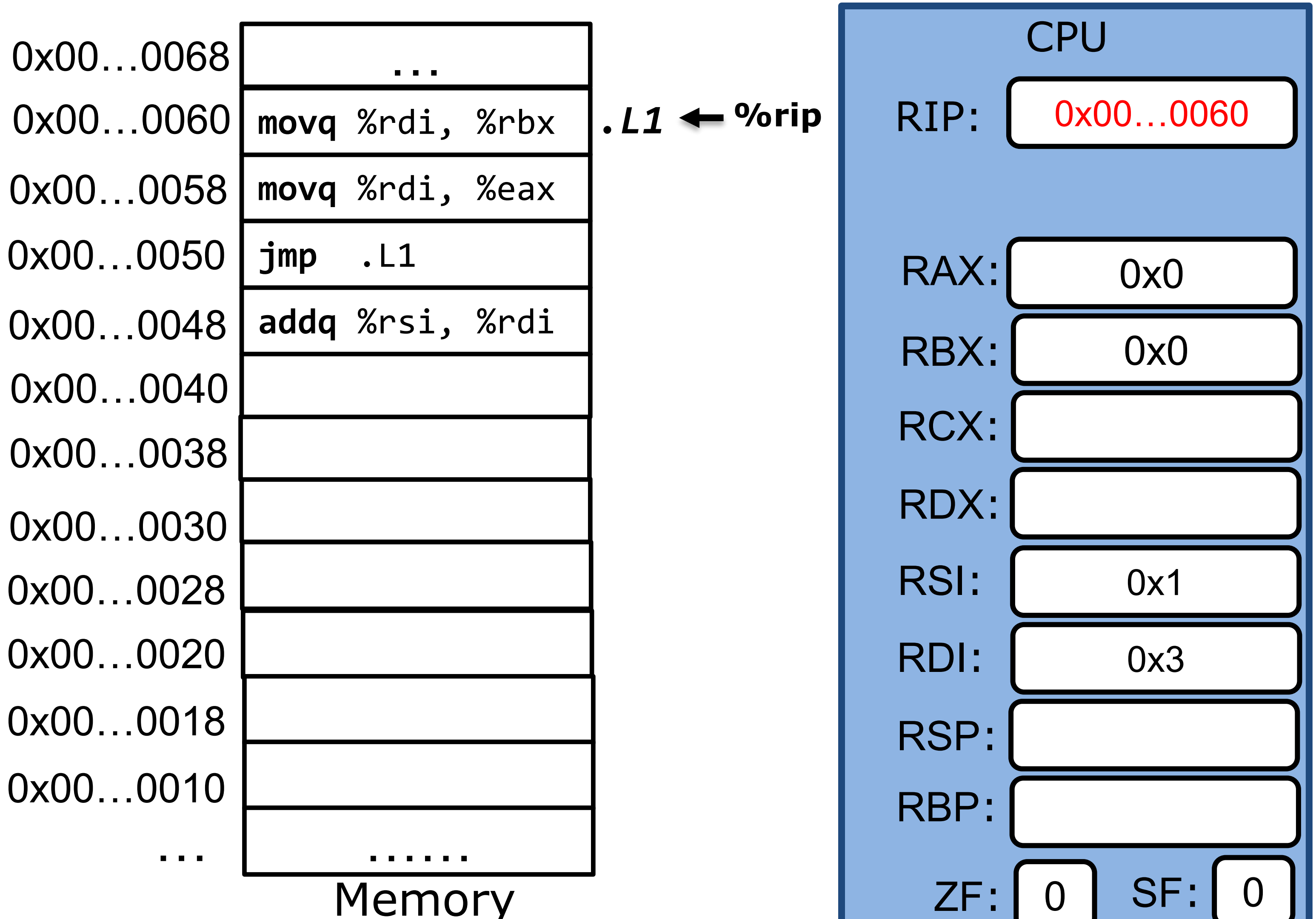
- Transfer control to a different point in the instruction stream by changing %rip
- Label specifies the address to jump to
- jmp is like ***goto***

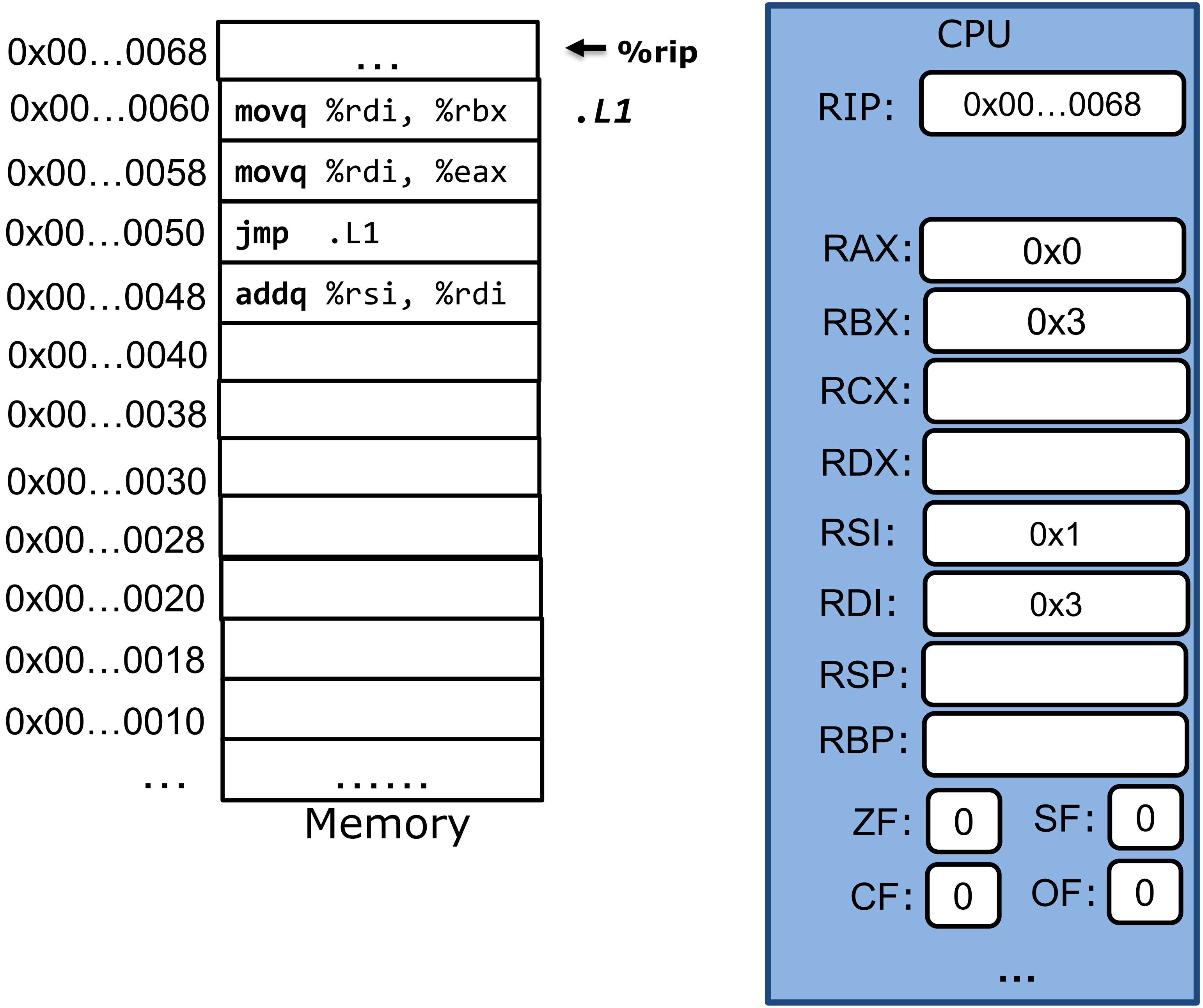
```
  addq %rsi, %rdi
  jmp  .L1
  movq %rdi, %eax
.L1
  movq %rdi, %rbx
```



0x00…0068
...
0x00…0060
movq %rdi, %rbx
.L1
0x00…0058
movq %rdi, %eax
0x00…0050
jmp .L1
0x00…0048
addq %rsi, %rdi
%rip
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
...
......
Memory
CPU
RIP:
0x00…0048
RAX:
0x0
RBX:
0x0
RCX:
RDX:
RSI:
0x1
RDI:
0x2
RSP:
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...


0x00…0068
...
0x00…0060
movq %rdi, %rbx
.L1
0x00…0058
movq %rdi, %eax
0x00…0050
jmp .L1
%rip
0x00…0048
addq %rsi, %rdi
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
...
......
Memory
CPU
RIP: 0x00…0050
RAX: 0x0
RBX: 0x0
RCX:
RDX:
RSI: 0x1
RDI: 0x3
RSP:
RBP:
ZF: 0
SF: 0
CF: 0
OF: 0
...


0x00…0068
...
0x00…0060
movq %rdi, %rbx
.L1
%rip
0x00…0058
movq %rdi, %eax
0x00…0050
jmp .L1
0x00…0048
addq %rsi, %rdi
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
...
......
Memory
CPU
RIP:
0x00…0060
RAX:
0x0
RBX:
0x0
RCX:
RDX:
RSI:
0x1
RDI:
0x3
RSP:
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...

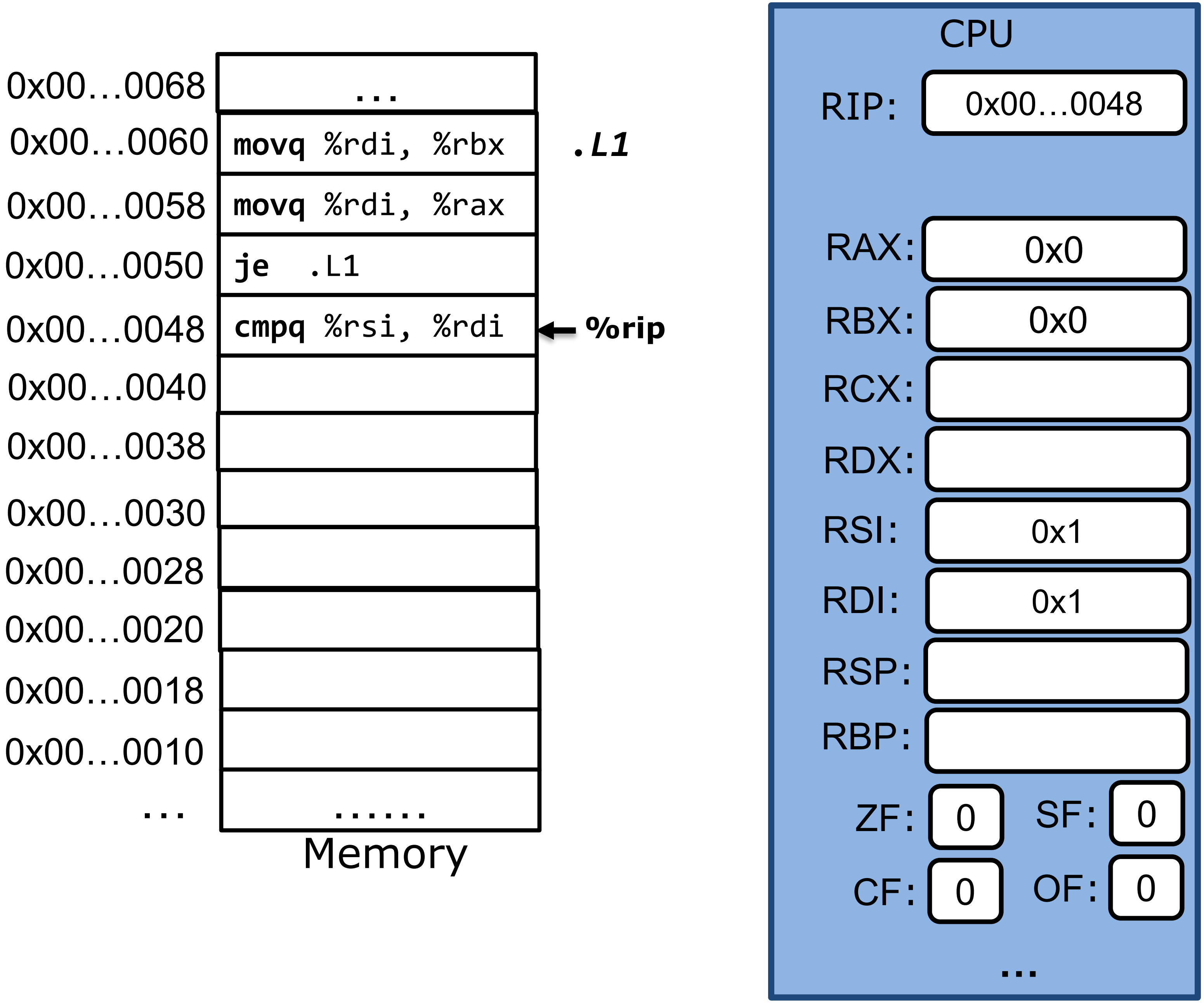

0x00…0068
...
%rip
0x00…0060
movq %rdi, %rbx
.L1
0x00…0058
movq %rdi, %eax
0x00…0050
jmp .L1
0x00…0048
addq %rsi, %rdi
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
...
......
Memory
CPU
RIP:
0x00…0068
RAX:
0x0
RBX:
0x3
RCX:
RDX:
RSI:
0x1
RDI:
0x3
RSP:
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...

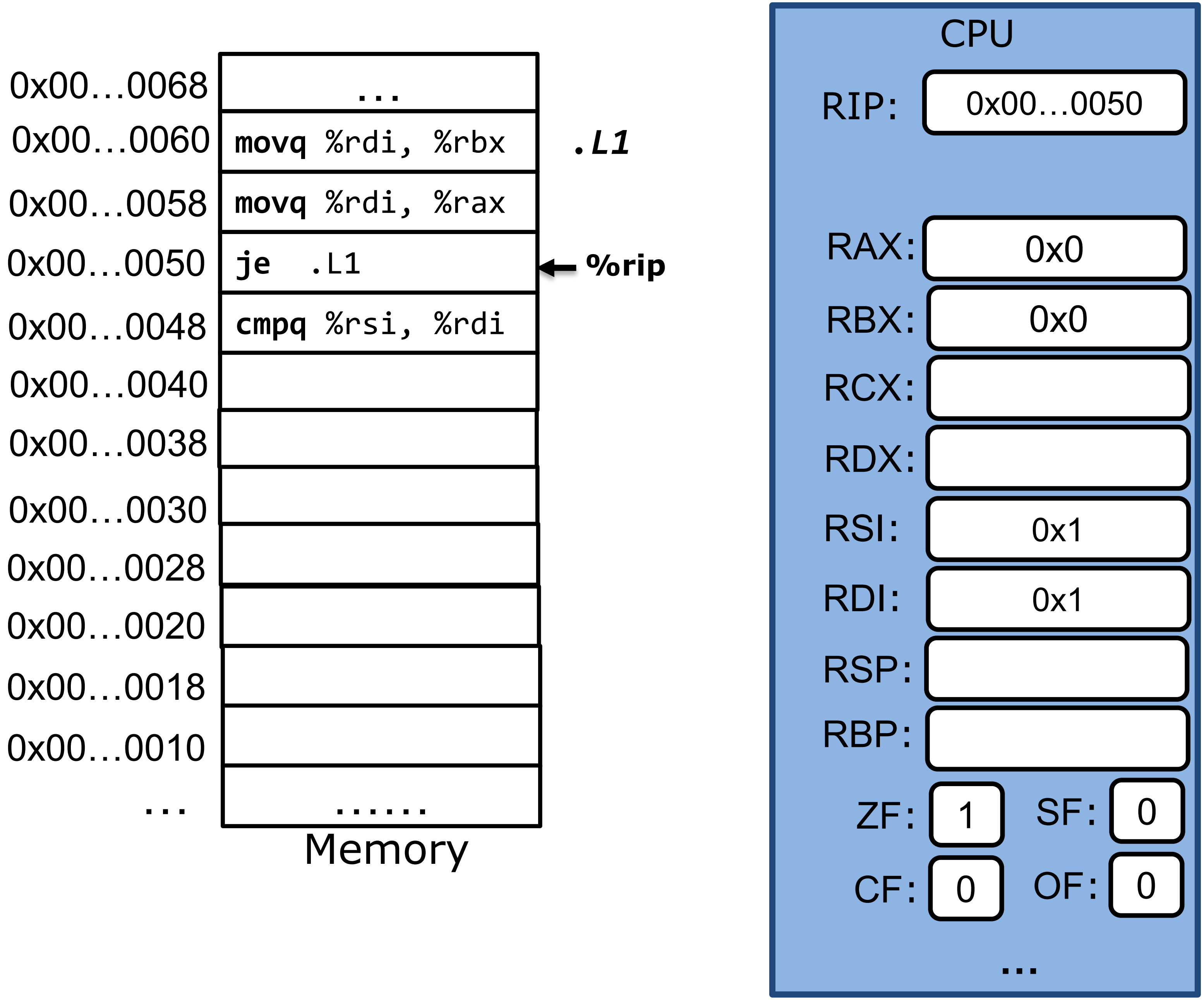
Conditional jump instruction

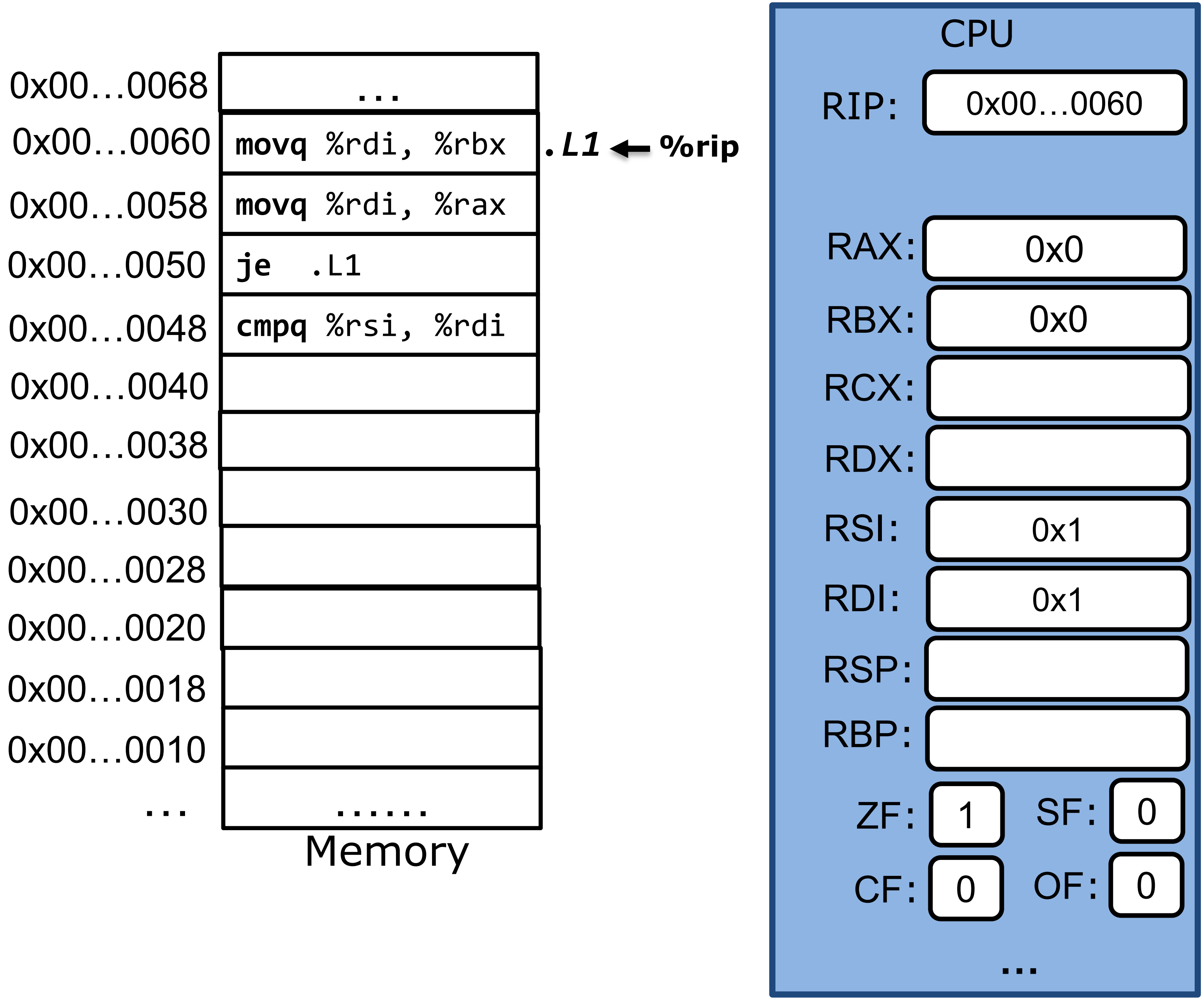
`jX label`

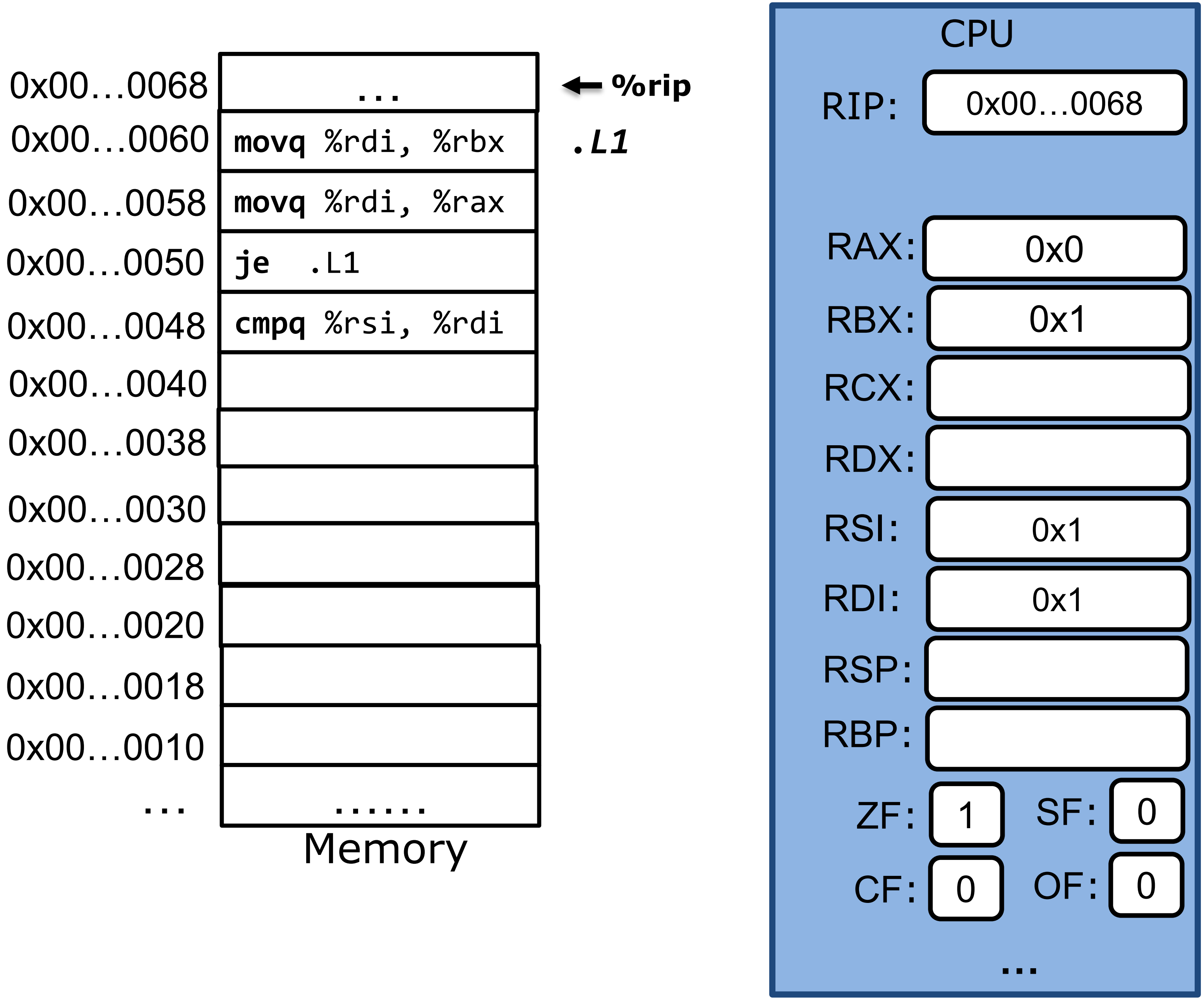
- If condition **X** is met, jump to the label

jX	Condition	Description
`je`	`ZF`	Equal / Zero
`jne`	`~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)


0x00…0068
...
0x00…0060
movq %rdi, %rbx
.L1
0x00…0058
movq %rdi, %rax
0x00…0050
je .L1
0x00…0048
cmpq %rsi, %rdi
%rip
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
...
......
Memory
CPU
RIP:
0x00…0048
RAX:
0x0
RBX:
0x0
RCX:
RDX:
RSI:
0x1
RDI:
0x1
RSP:
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...


0x00…0068
...
0x00…0060
movq %rdi, %rbx
.L1
0x00…0058
movq %rdi, %rax
0x00…0050
je .L1
%rip
0x00…0048
cmpq %rsi, %rdi
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
...
......
Memory
CPU
RIP:
0x00…0050
RAX:
0x0
RBX:
0x0
RCX:
RDX:
RSI:
0x1
RDI:
0x1
RSP:
RBP:
ZF:
1
SF:
0
CF:
0
OF:
0
...


0x00…0068
...
0x00…0060
movq %rdi, %rbx
.L1
%rip
0x00…0058
movq %rdi, %rax
0x00…0050
je .L1
0x00…0048
cmpq %rsi, %rdi
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
...
......
Memory
CPU
RIP:
0x00…0060
RAX:
0x0
RBX:
0x0
RCX:
RDX:
RSI:
0x1
RDI:
0x1
RSP:
RBP:
ZF:
1
SF:
0
CF:
0
OF:
0
...


0x00…0068
...
%rip
0x00…0060
movq %rdi, %rbx
.L1
0x00…0058
movq %rdi, %rax
0x00…0050
je .L1
0x00…0048
cmpq %rsi, %rdi
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
...
......
Memory
CPU
RIP:
0x00…0068
RAX:
0x0
RBX:
0x1
RCX:
RDX:
RSI:
0x1
RDI:
0x1
RSP:
RBP:
ZF:
1
SF:
0
CF:
0
OF:
0
...

How “if..else..” statement works

```
long compare(long x, long y)
{
  long result;
  if (x > y)
    result = 1;
  else
    result = 0;
  return result;
}
```

gcc -Og -S compared.c

```
compare:
    cmpq   %rdi, %rsi
    jge    .L3
    movl   $1, %rax
    ret
.L3:
    movl   $0, %rax
    ret
```

Register	Use(s)
%rdi	Argument **x**
%rsi	Argument **y**
%rax	Return value

How “while” works

logical right shift

```
long count(unsigned long x)
{
  long cnt = 0;
  while (x != 0) {
    x = x >> 1;
    cnt++;
   }
   return cnt;
}
```

gcc -Og -S count.c

```
count:
    movq $0, %rax
    jmp .L2
.L3:
    shrq %rdi
    addq $1, %rax
.L2:
    testq %rdi, %rdi
    jne .L3
    ret
```

Sets RFLAGS based on result of logical AND

Register	Use(s)
%rdi	Argument **x**
%rax	Return value

“While” Translation example

```
long count(unsigned long x)
{
  long cnt = 0;
  while (x != 0) {
    x = x >> 1;
    cnt++;
   }
   return cnt;
}
```

```
count:
    movq $0, %rax        long cnt = 0;
    jmp .L2              goto .L2
.L3:                    .L3:
    shrq %rdi              x = x >> 1
    addq $1, %rax          cnt = cnt + 1
.L2:                    .L2:
    testq %rdi, %rdi       if x != 0
    jne .L3                  goto .L3
    ret
                           return cnt
```

Register	Use(s)
%rdi	Argument **x**
%rax	Return value

"For" Loop translation

For Version

```
for (Init; Test; Update )
      Body
```

While Version

```
Init ;
while (Test ) {
      Body
      Update ;
}
```

“Loop” Translation example

- `gcc –Og –S *.c`

```
long sum(long n)
{
  long s = 0;
  for (long i=0; i<n; i++){
    s += i;
  }
  return s;
}
```

```
sum:
    movq $0, %rdx
    movq $0, %rax
    jmp .L5
.L6:
    addq %rdx, %rax
    addq $1, %rdx
.L5:
    cmpq %rdi, %rdx
    jl  .L6
    ret
```

Register	Use(s)
`%rdi`	`n`
`%rax`	`s`
`%rdx`	`i`