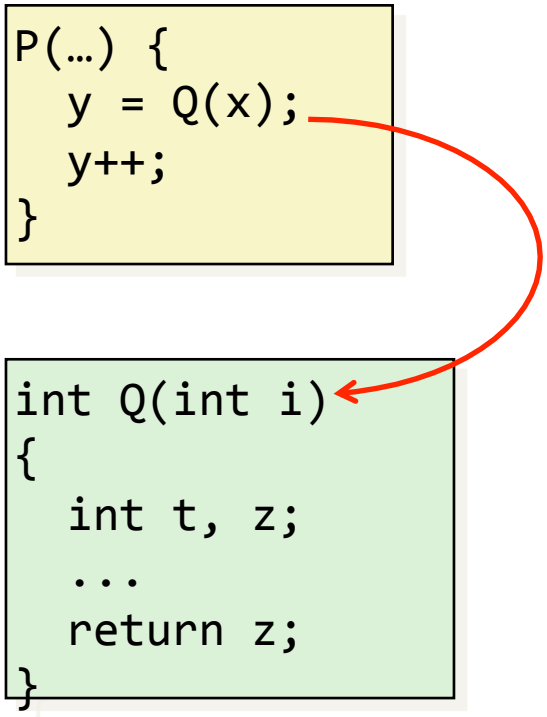
Machine Program: Procedure

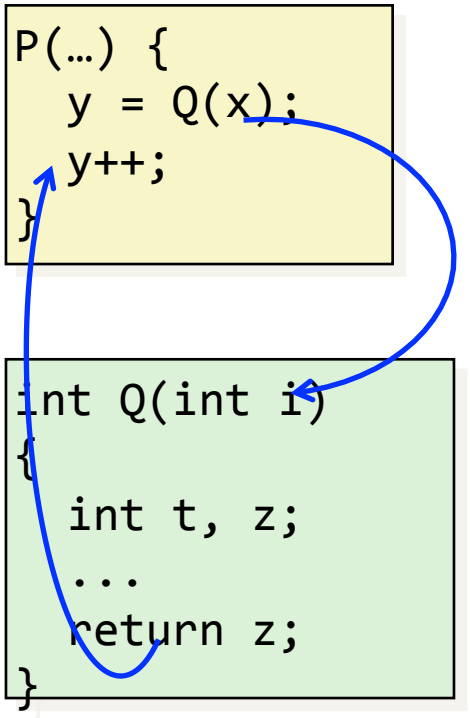
Zhaoguo Wang

Requirements of procedure calls?



1. Passing control

Requirements of procedure calls?



1. Passing control
2. Passing Arguments & return value

Requirements of procedure calls?

```
P(…) {
   y = Q(x);
   y++;
}
```

```
int Q(int i)
{
   int t, z;
   ...
   return z;
}
```

1. Passing control
2. Passing Arguments & return value
3. Allocate / deallocate local variables

How to transfer control for procedure calls?

```
void main(){
    ..
    f(..)
L1: ..
}
```

```
void f(){
    ..
    g(..)
L2: ..
}
```

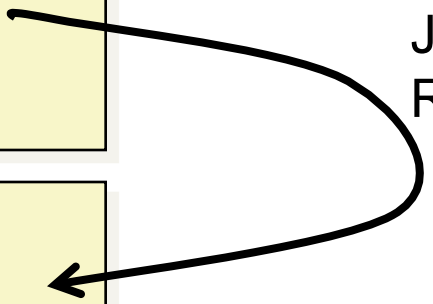
```
void g(){
    ..
    h(..)
L3: ..
}
```

How to transfer control for procedure calls?

```
void main(){
    ..
    f(..)
L1: ..
}
```

```
void f(){
    ..
    g(..)
L2: ..
}
```

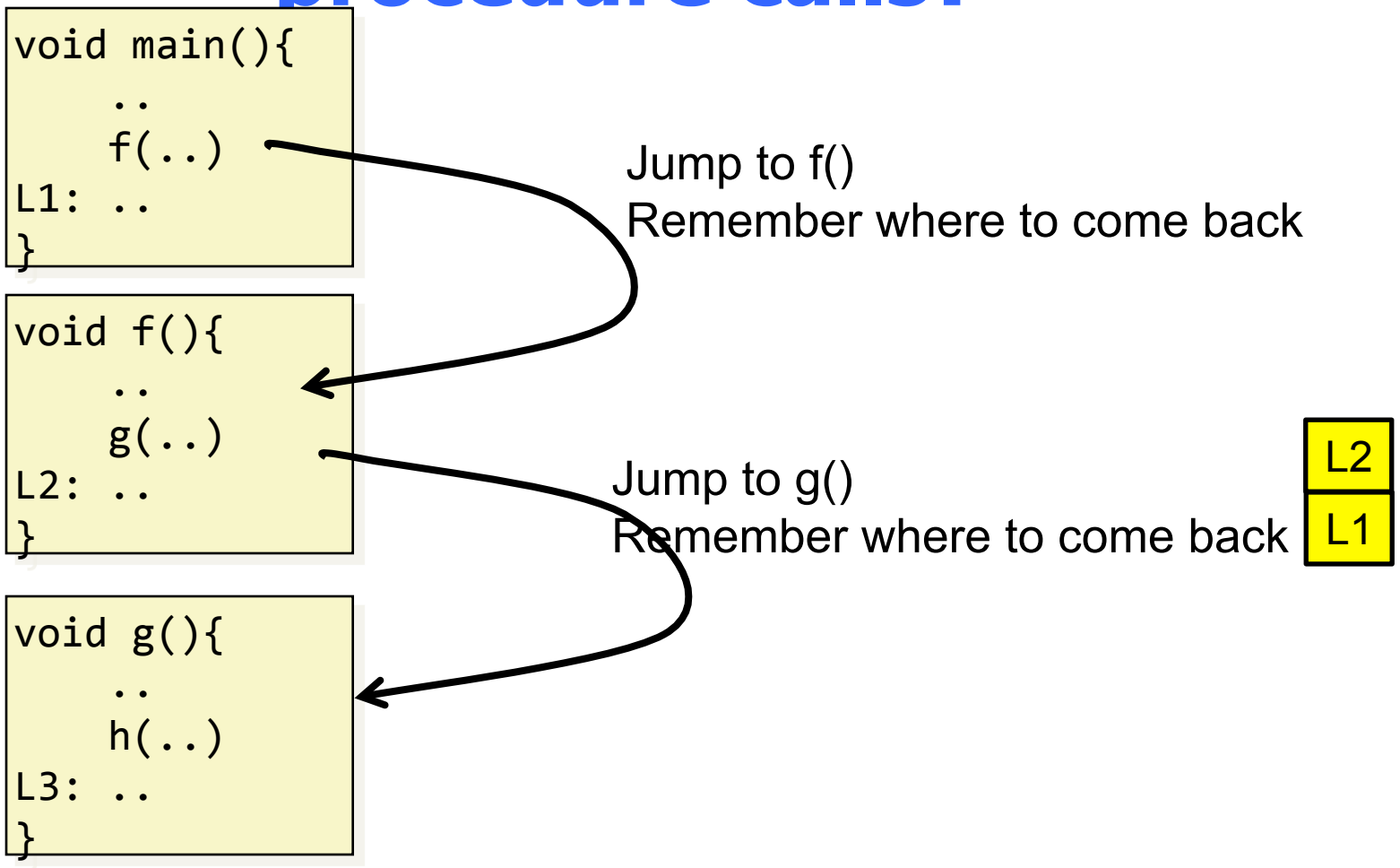
```
void g(){
    ..
    h(..)
L3: ..
}
```



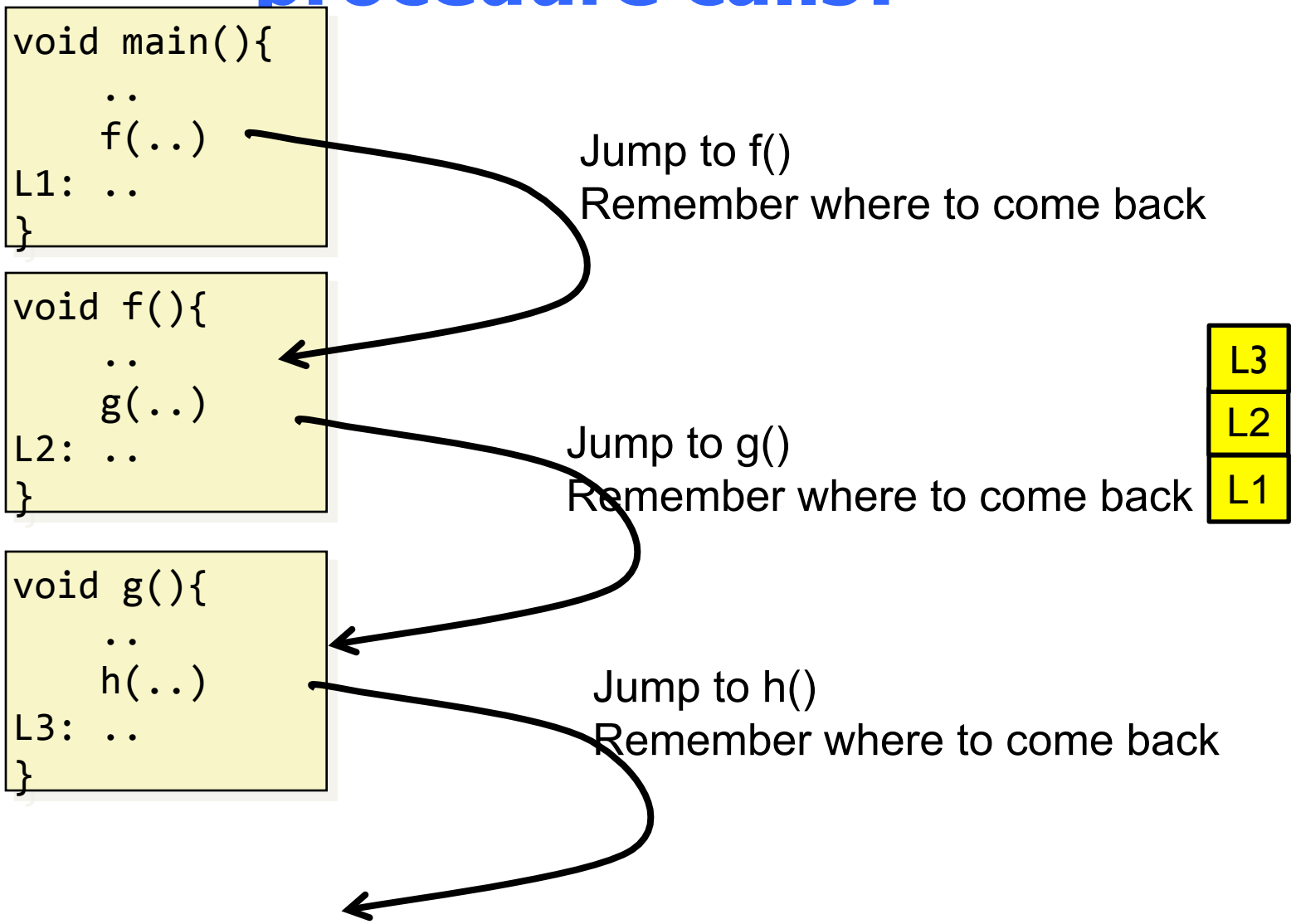
Jump to f()
Remember where to come back

L1

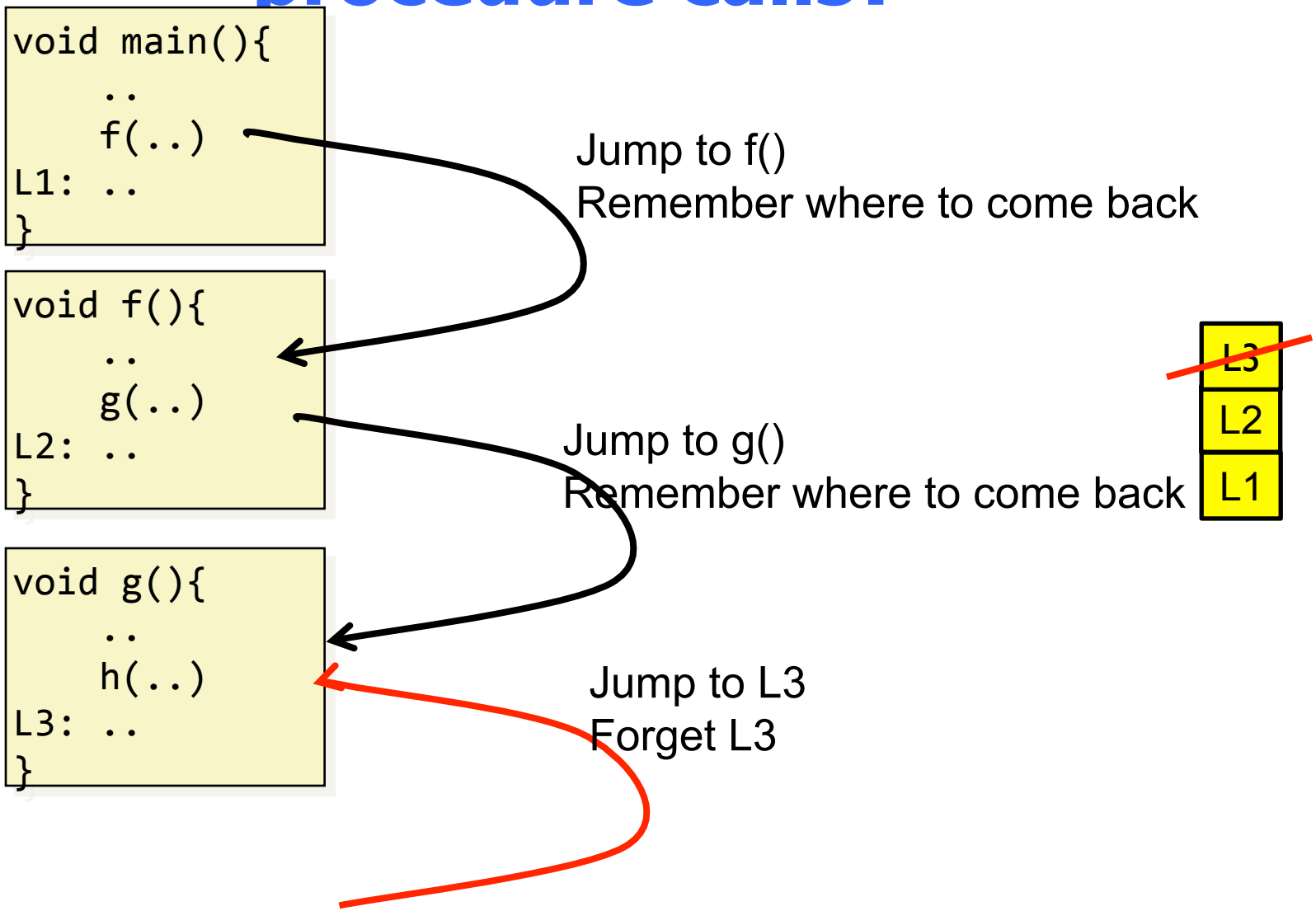
How to transfer control for procedure calls?



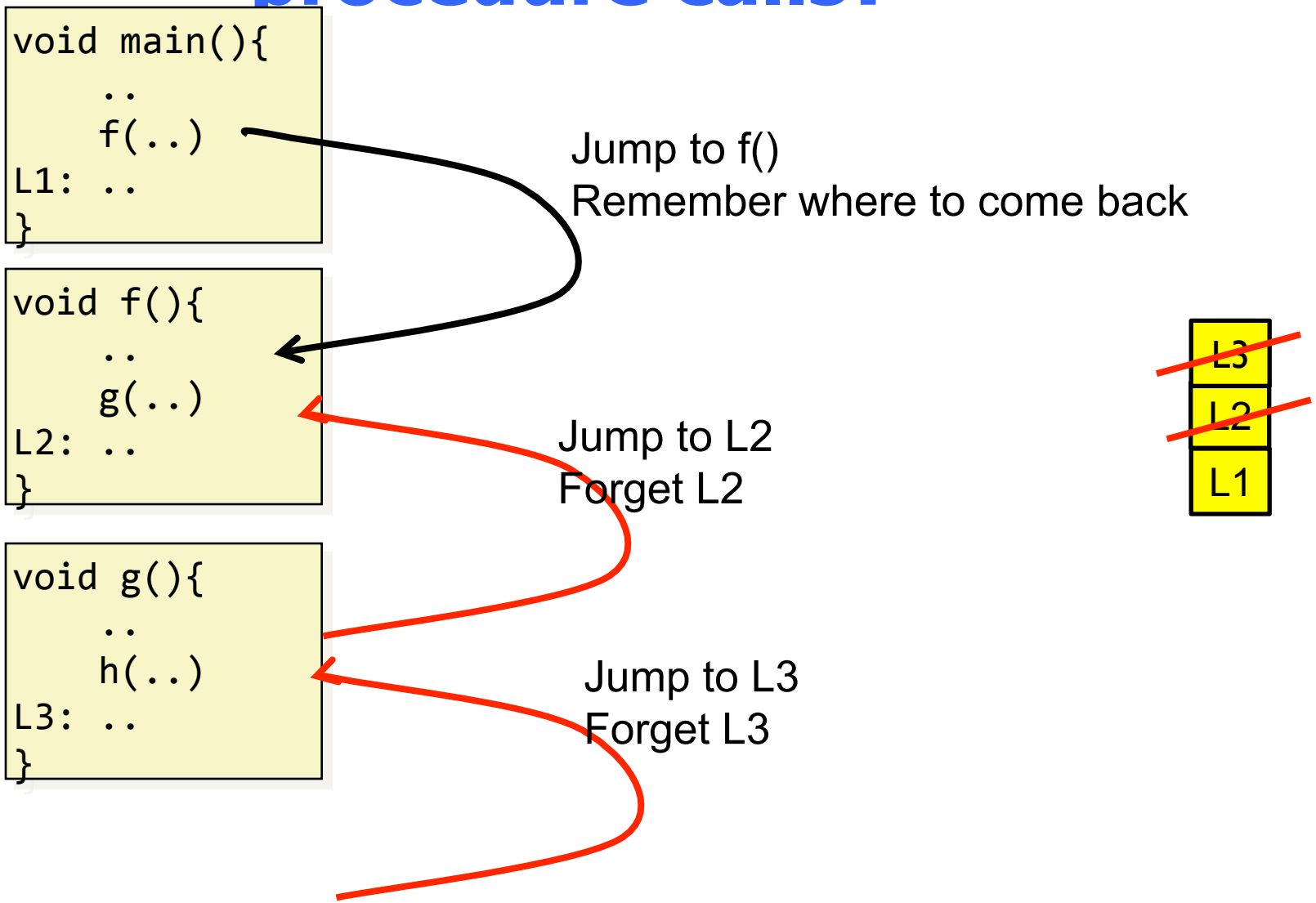
How to transfer control for procedure calls?



How to transfer control for procedure calls?



How to transfer control for procedure calls?



How to transfer control for procedure calls?

```
void main(){
    ..
    f(..)
L1: ..
}
```

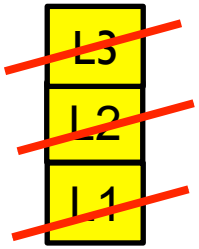
```
void f(){
    ..
    g(..)
L2: ..
}
```

```
void g(){
    ..
    h(..)
L3: ..
}
```

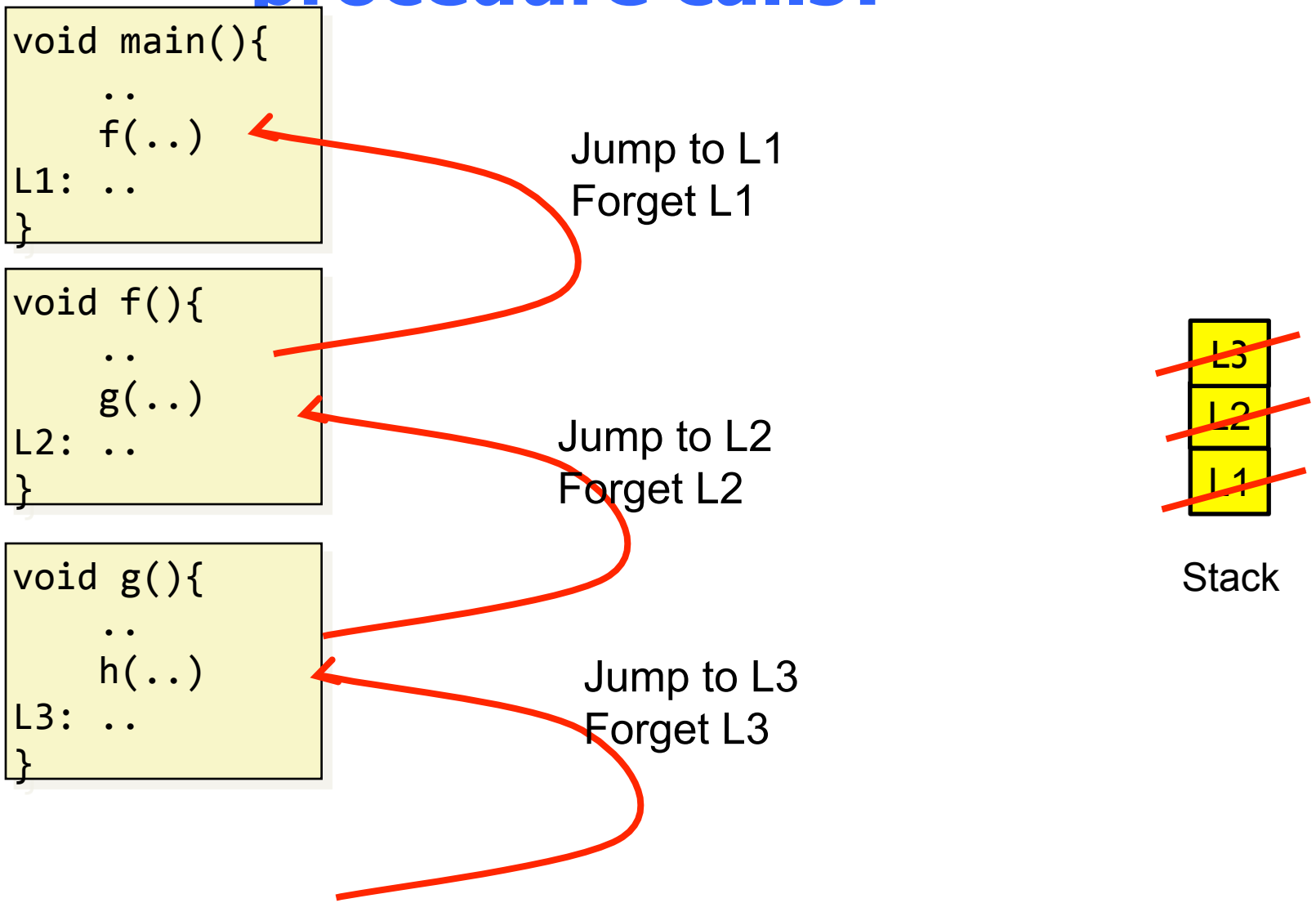
Jump to L1
Forget L1

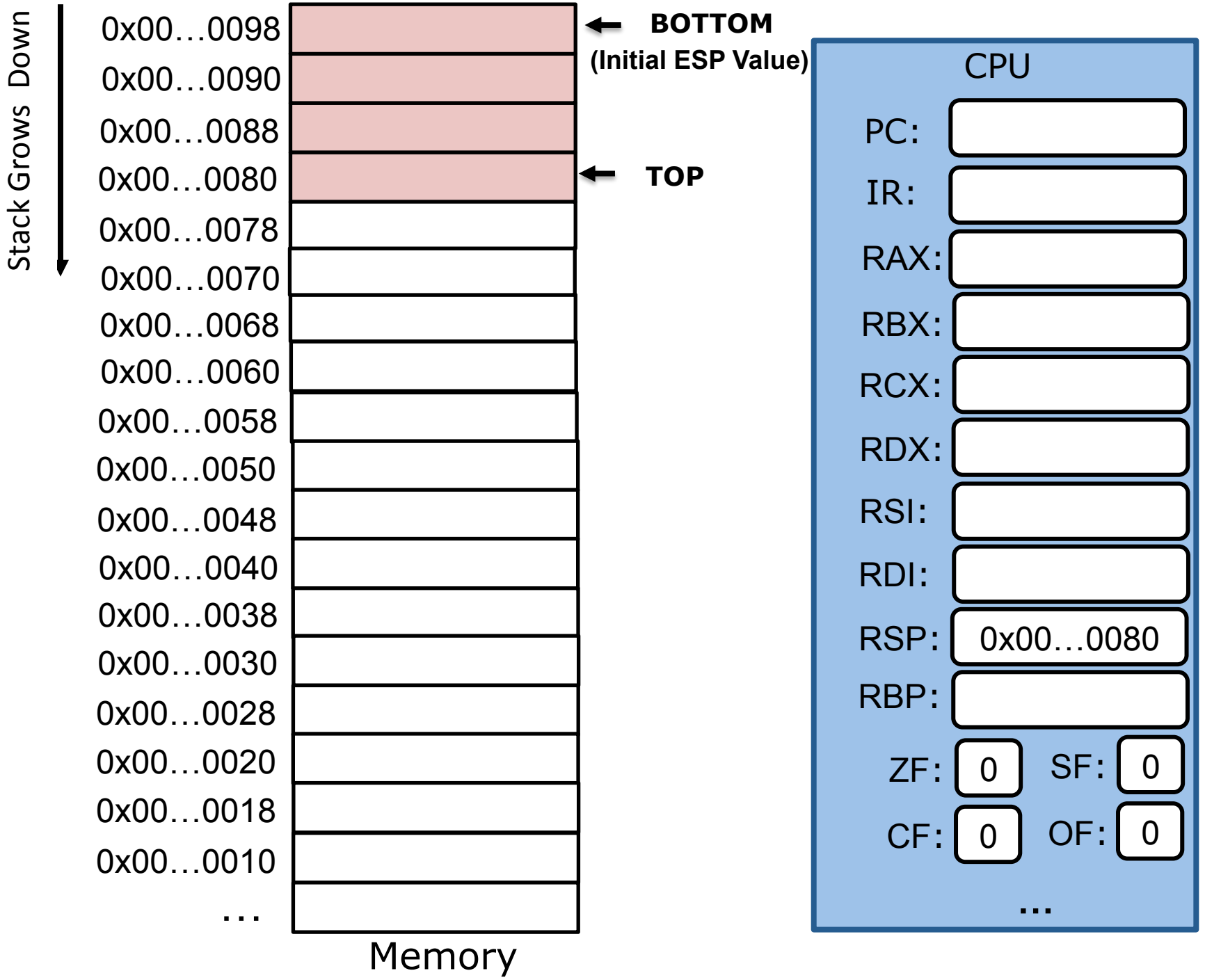
Jump to L2
Forget L2

Jump to L3
Forget L3



How to transfer control for procedure calls?

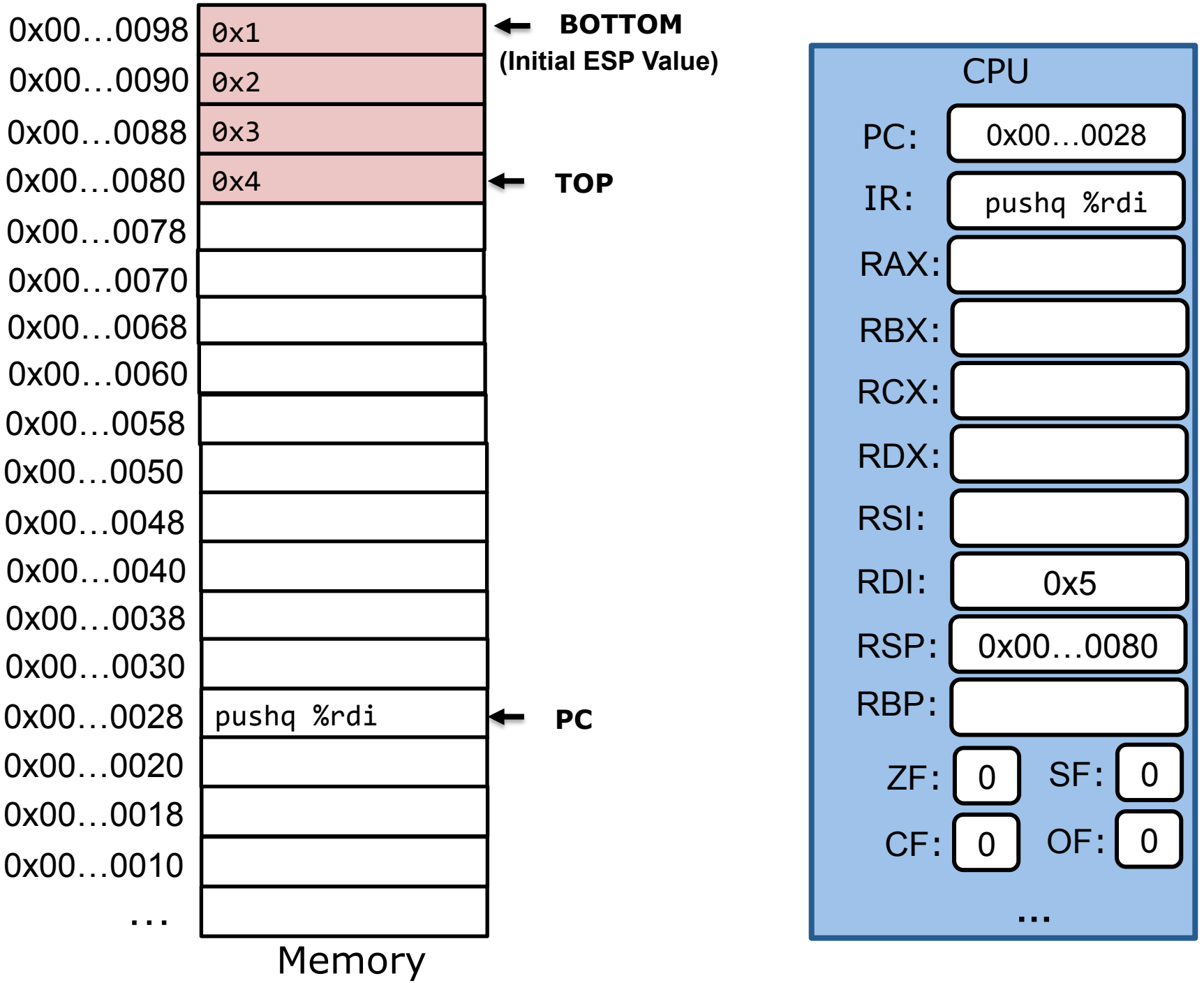


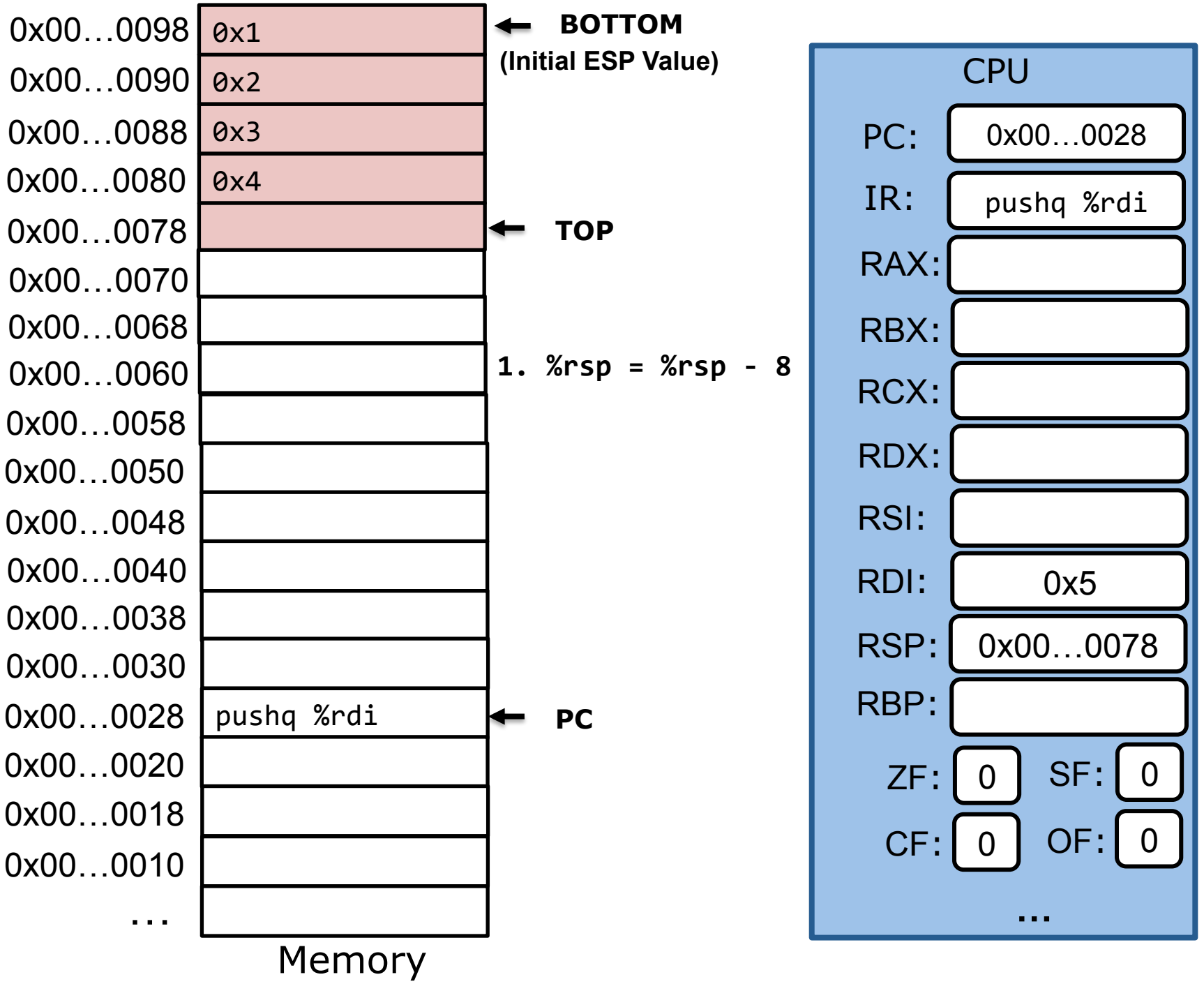

Stack Grows Down
0x00…0098
0x00…0090
0x00…0088
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x00…0010
…
BOTTOM
(Initial ESP Value)
TOP
Memory
CPU
PC:
IR:
RAX:
RBX:
RCX:
RDX:
RSI:
RDI:
RSP:
0x00…0080
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…

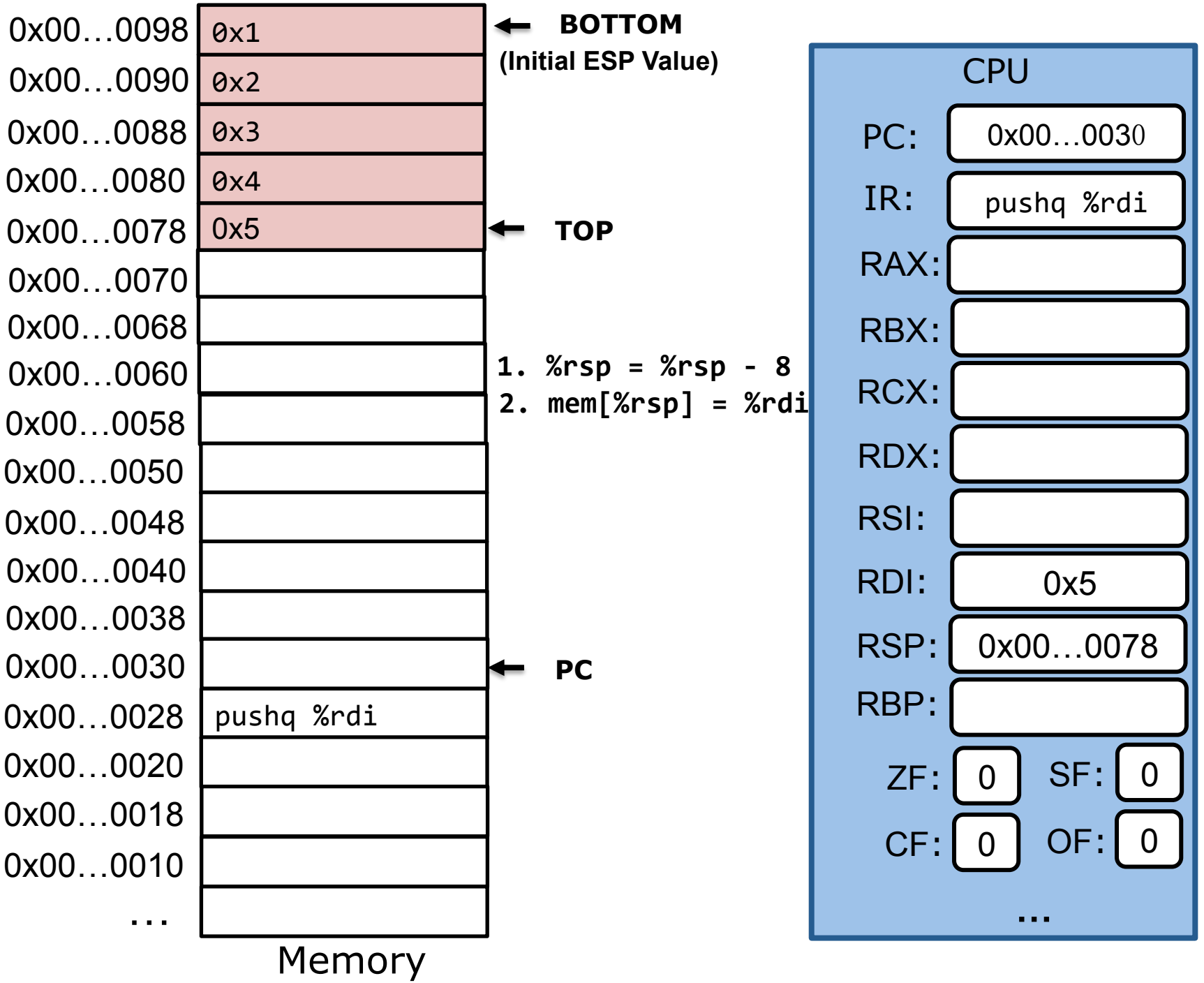
Stack – push Instruction

`pushq` `src`

- Decrement %rsp by 8
- Write operand at address given by `%rsp`


0x00…0098
0x1
BOTTOM
(Initial ESP Value)
0x00…0090
0x2
0x00…0088
0x3
0x00…0080
0x4
TOP
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
0x00…0028
pushq %rdi
PC
0x00…0020
0x00…0018
0x00…0010
…
Memory
CPU
PC:
0x00…0028
IR:
pushq %rdi
RAX:
RBX:
RCX:
RDX:
RSI:
RDI:
0x5
RSP:
0x00…0080
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…

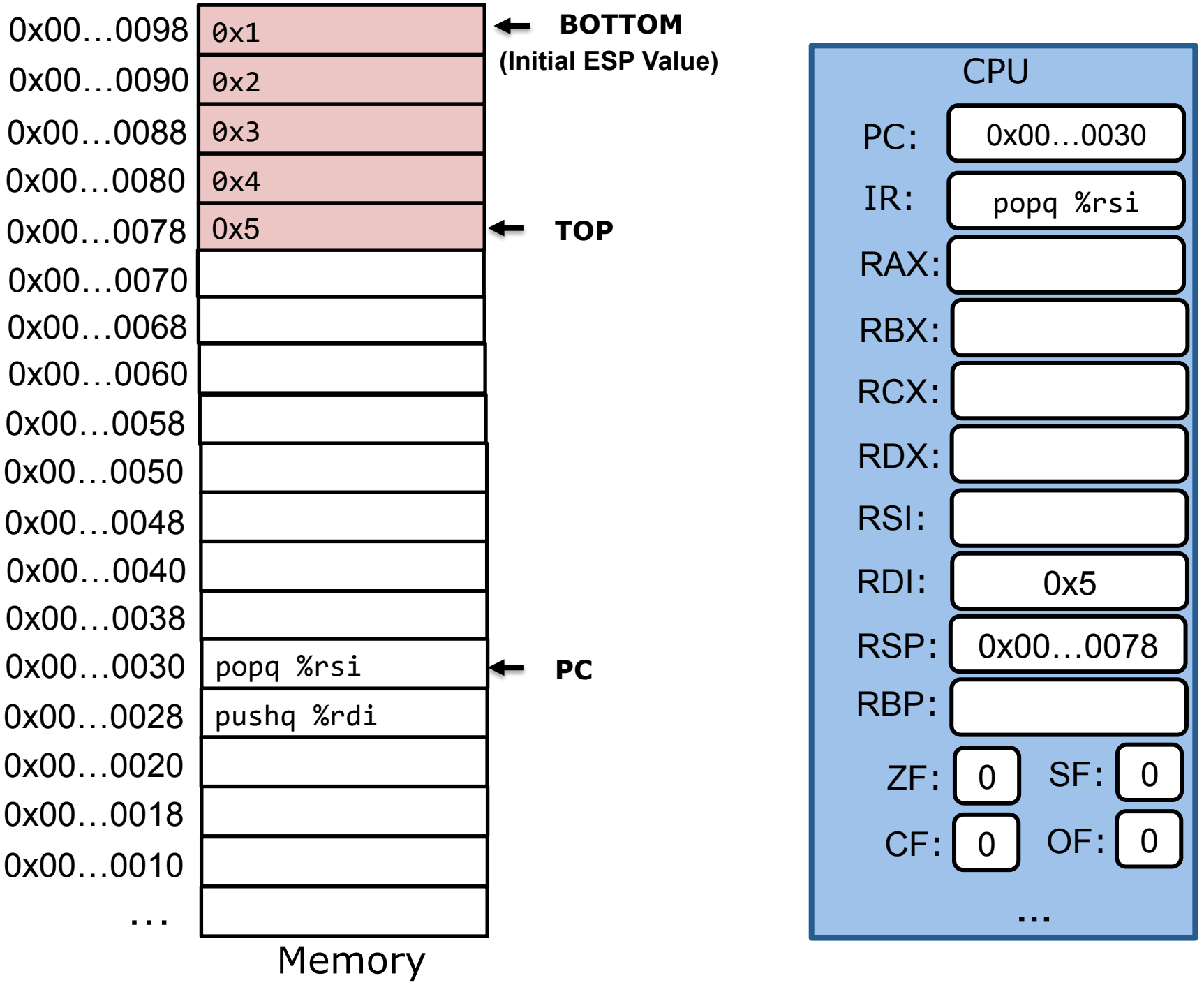

0x00…0098
0x1
BOTTOM
(Initial ESP Value)
0x00…0090
0x2
0x00…0088
0x3
0x00…0080
0x4
0x00…0078
TOP
0x00…0070
0x00…0068
0x00…0060
1. %rsp = %rsp - 8
0x00…0058
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
0x00…0028
pushq %rdi
PC
0x00…0020
0x00…0018
0x00…0010
…
Memory
CPU
PC:
0x00…0028
IR:
pushq %rdi
RAX:
RBX:
RCX:
RDX:
RSI:
RDI:
0x5
RSP:
0x00…0078
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…

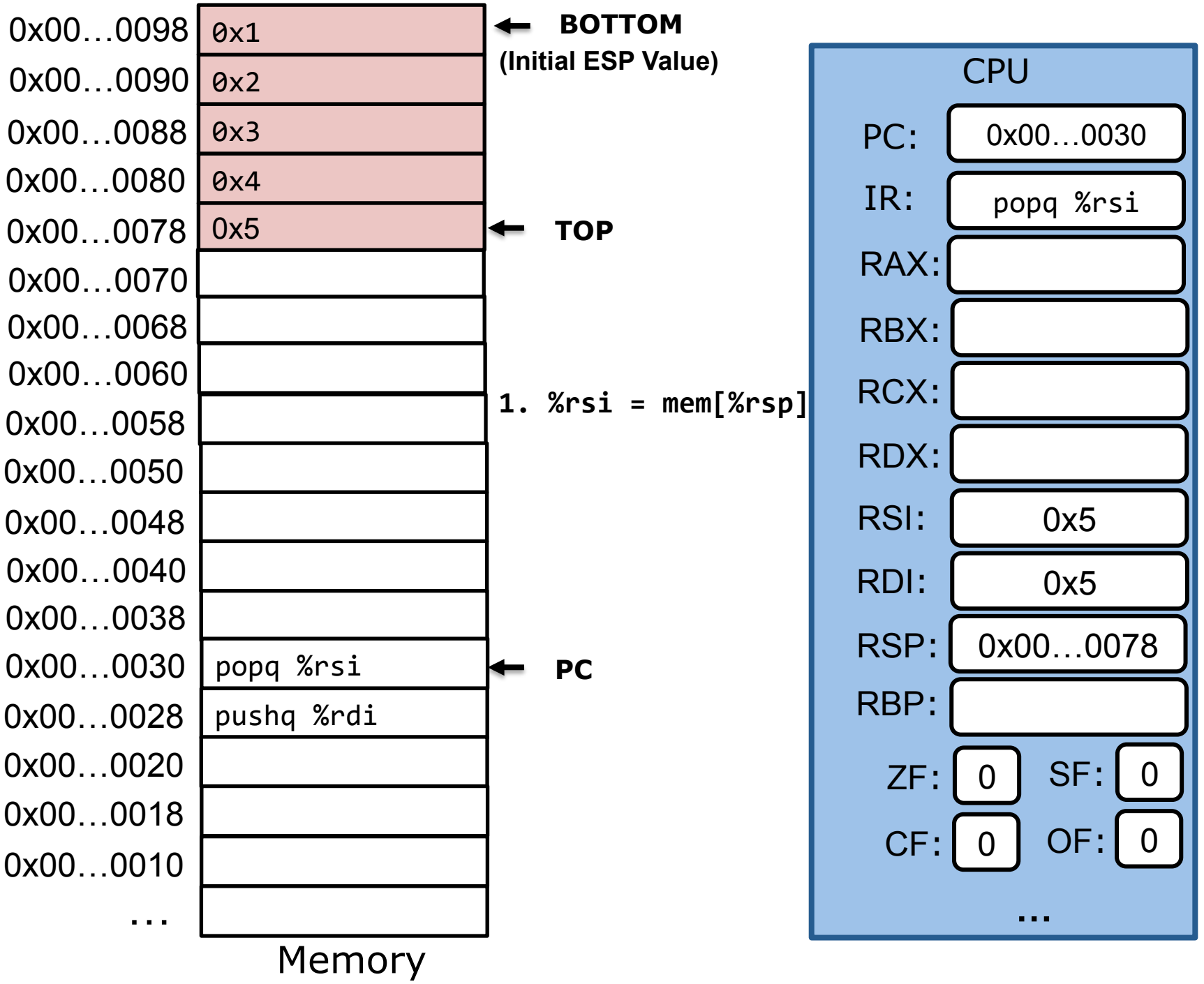

0x00…0098
0x1
BOTTOM
(Initial ESP Value)
0x00…0090
0x2
0x00…0088
0x3
0x00…0080
0x4
0x00…0078
0x5
TOP
0x00…0070
0x00…0068
0x00…0060
1. %rsp = %rsp - 8
2. mem[%rsp] = %rdi
0x00…0058
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
PC
0x00…0028
pushq %rdi
0x00…0020
0x00…0018
0x00…0010
…
Memory
CPU
PC:
0x00…0030
IR:
pushq %rdi
RAX:
RBX:
RCX:
RDX:
RSI:
RDI:
0x5
RSP:
0x00…0078
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…

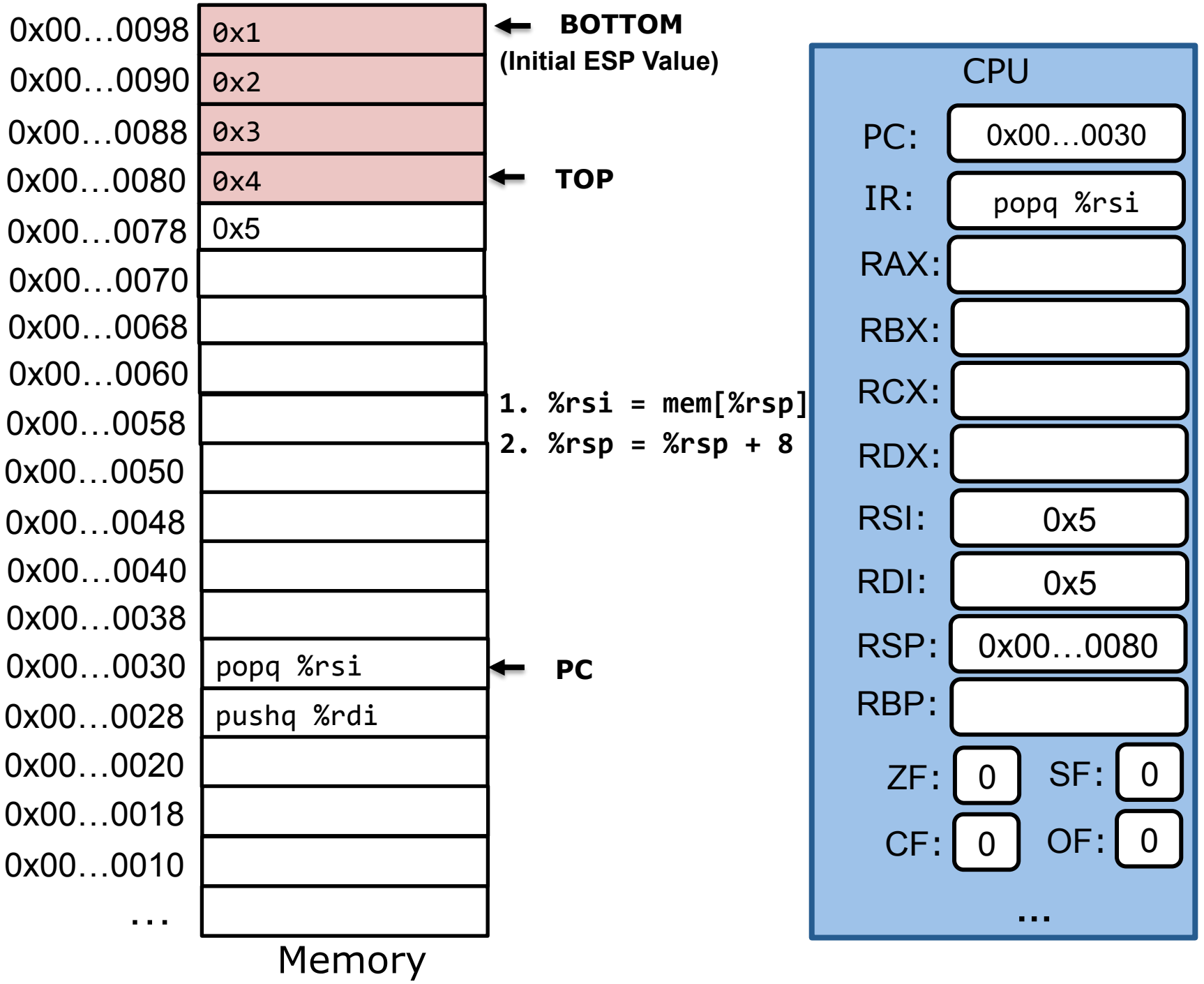
Stack – pop Instruction

popq `dest`

- Store the value at address `%rsp` to dest
- Increment %rsp by 8


0x00…0098
0x1
BOTTOM
(Initial ESP Value)
0x00…0090
0x2
0x00…0088
0x3
0x00…0080
0x4
0x00…0078
0x5
TOP
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
popq %rsi
PC
0x00…0028
pushq %rdi
0x00…0020
0x00…0018
0x00…0010
…
Memory
CPU
PC:
0x00…0030
IR:
popq %rsi
RAX:
RBX:
RCX:
RDX:
RSI:
RDI:
0x5
RSP:
0x00…0078
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…


0x00…0098
0x1
BOTTOM
(Initial ESP Value)
0x00…0090
0x2
0x00…0088
0x3
0x00…0080
0x4
0x00…0078
0x5
TOP
0x00…0070
0x00…0068
0x00…0060
1. %rsi = mem[%rsp]
0x00…0058
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
popq %rsi
PC
0x00…0028
pushq %rdi
0x00…0020
0x00…0018
0x00…0010
…
Memory
CPU
PC:
0x00…0030
IR:
popq %rsi
RAX:
RBX:
RCX:
RDX:
RSI:
0x5
RDI:
0x5
RSP:
0x00…0078
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…


0x00…0098
0x1
BOTTOM
(Initial ESP Value)
0x00…0090
0x2
0x00…0088
0x3
0x00…0080
0x4
TOP
0x00…0078
0x5
0x00…0070
0x00…0068
0x00…0060
0x00…0058
1. %rsi = mem[%rsp]
2. %rsp = %rsp + 8
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
popq %rsi
PC
0x00…0028
pushq %rdi
0x00…0020
0x00…0018
0x00…0010
…
Memory
CPU
PC:
0x00…0030
IR:
popq %rsi
RAX:
RBX:
RCX:
RDX:
RSI:
0x5
RDI:
0x5
RSP:
0x00…0080
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
…

Control transfer – call Instruction

call label(func name)

- Push return address on stack
  - Current pc + 8
- Jump label
  - Change the pc to the address of the label

```
int add(int a, int b) {
  int c = a + b;
  return c;
}
```

```
int main() {

    int a = 0;
    int b = 2;
    int c = add(a, b);
    printf("%d\b", c);
    return 0;
}
```

Control transfer – call Instruction

`call` `label(func name)`

- Push return address on stack
  - Current pc + 8
- Jump label
  - Change the pc to the address of the label

```
add:
    leal (%rdi,%rsi), %eax
    ret
```

```
main:
    movl    $2, %esi
    movl    $0, %edi
    call    add
    movl    %eax, %edx
    ...
```

GCC -O1 *.c

Control transfer – call Instruction

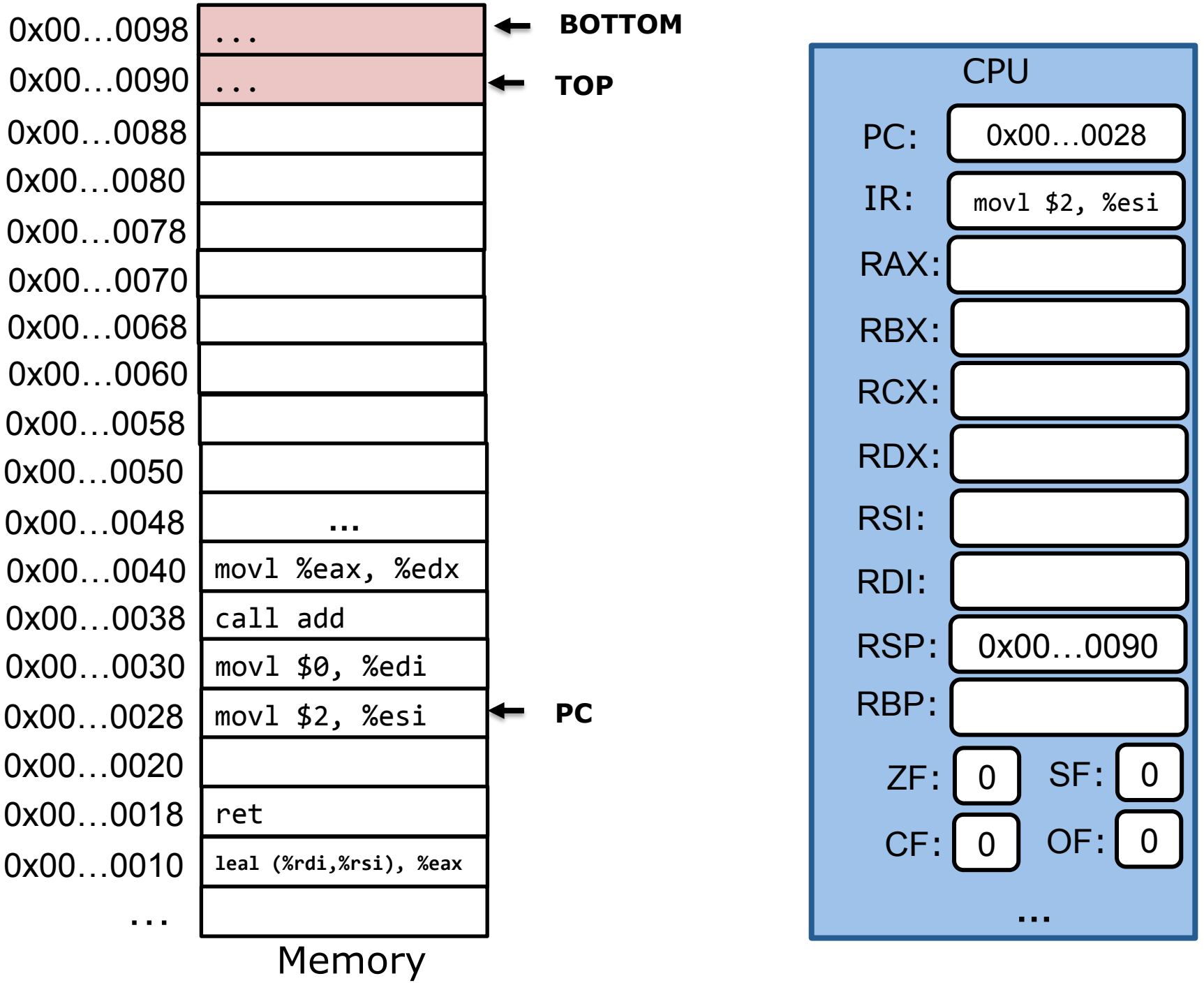
ret

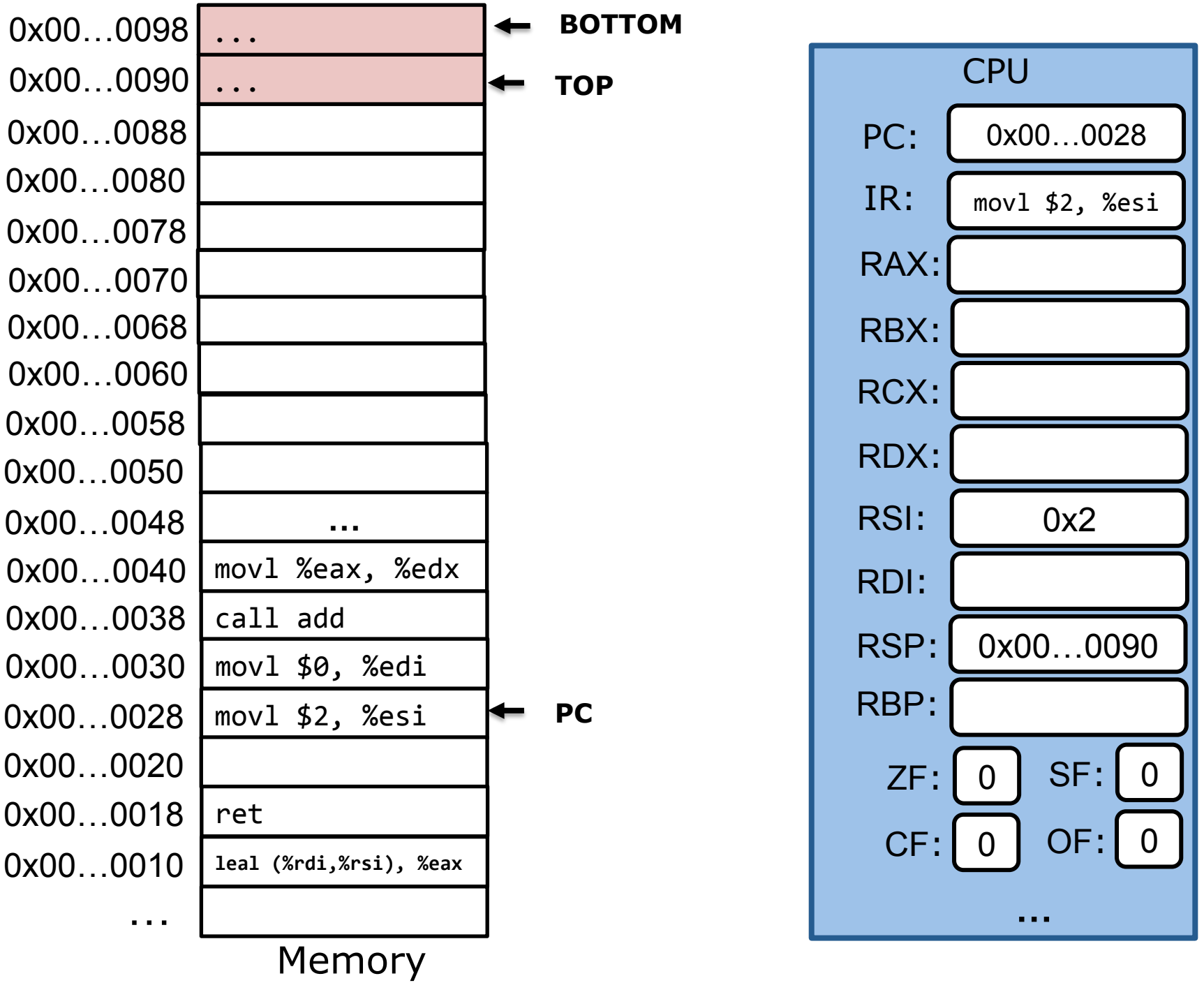
- Pop 8 bytes from the stack to PC
  - pc = mem[%rsp]

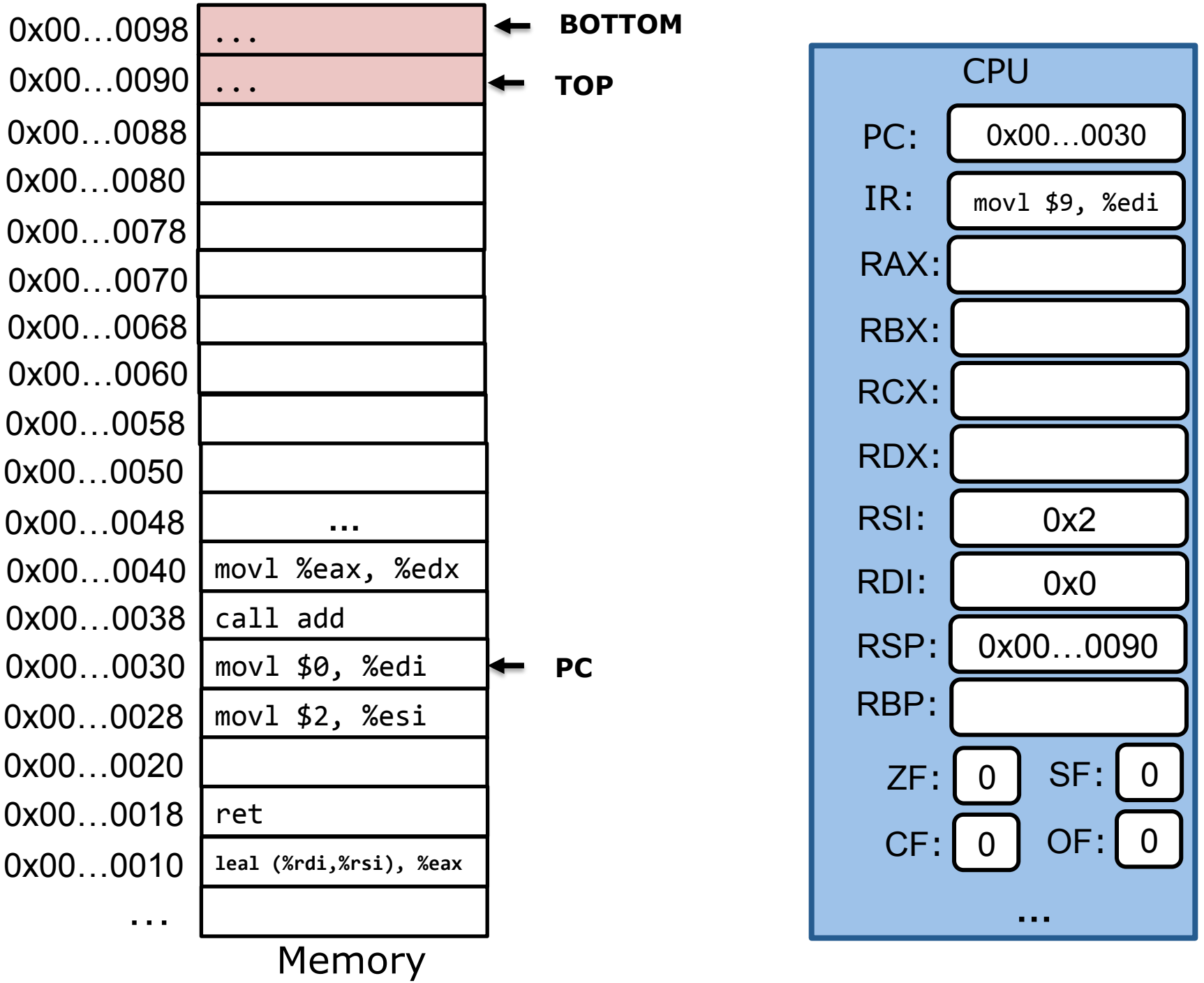
```
add:
    leal (%rdi,%rsi), %eax
    ret
```

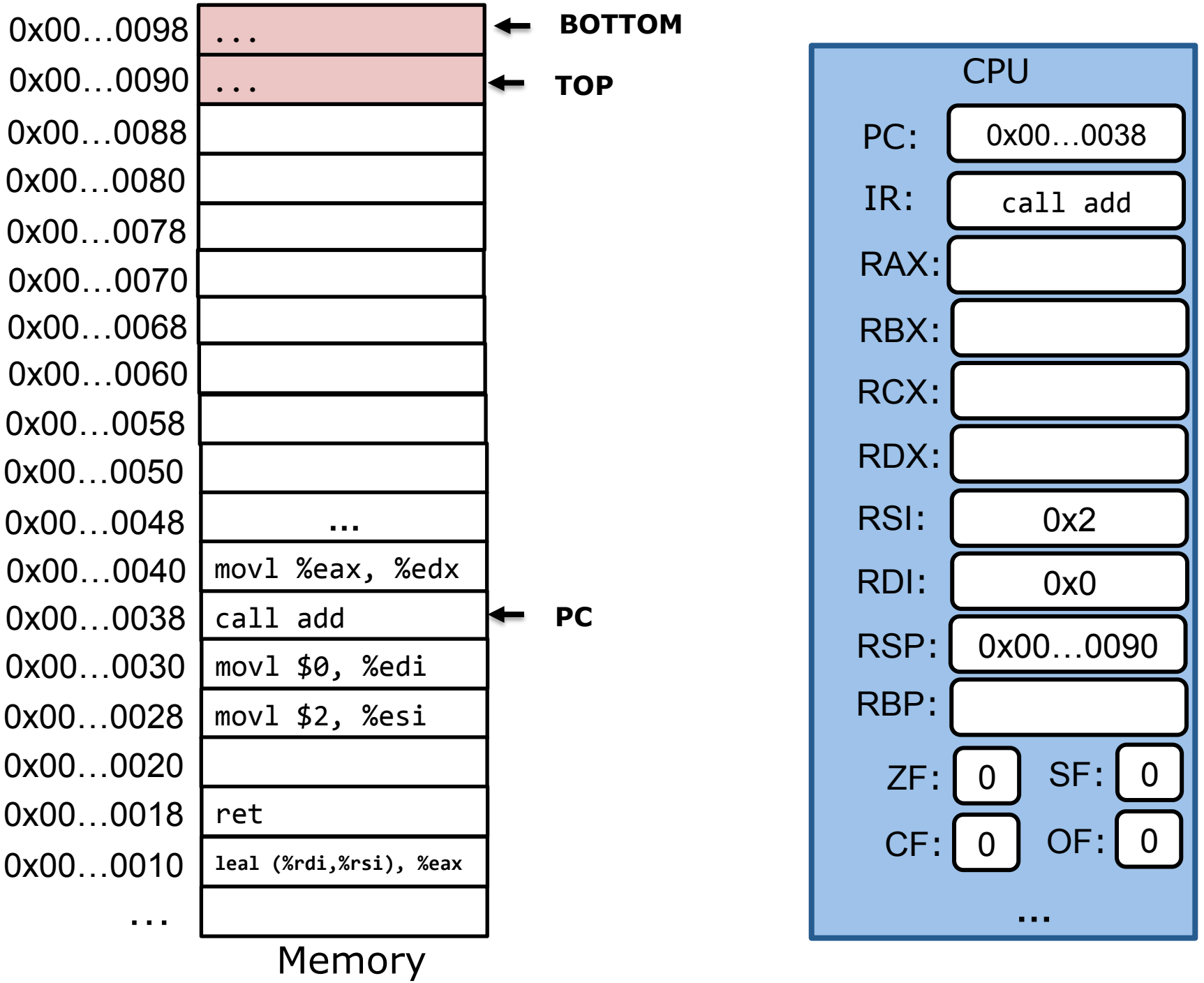
```
main:
    movl    $2, %esi
    movl    $0, %edi
    call    add
    movl    %eax, %edx
     ...
```

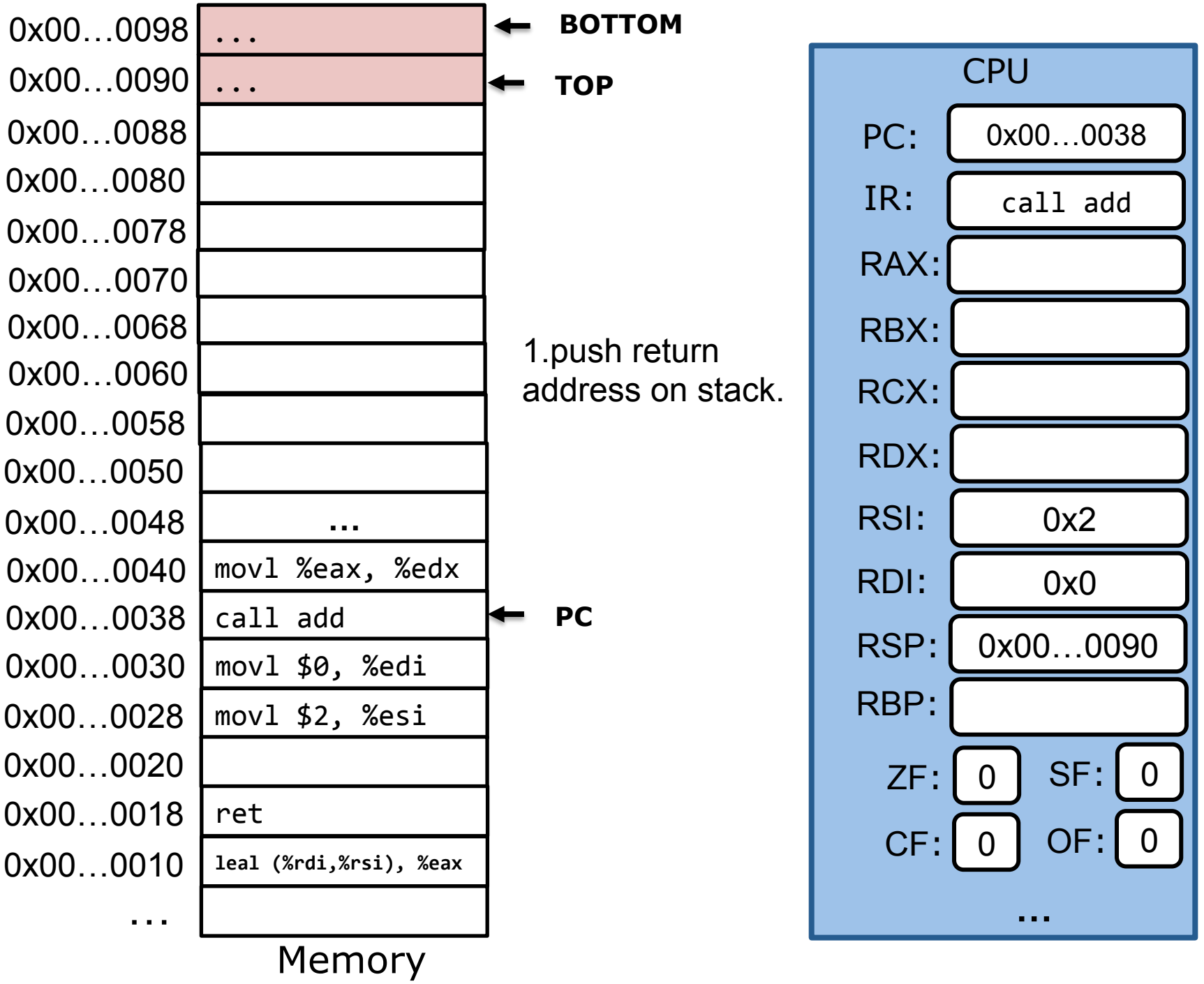
GCC -O1 *.c

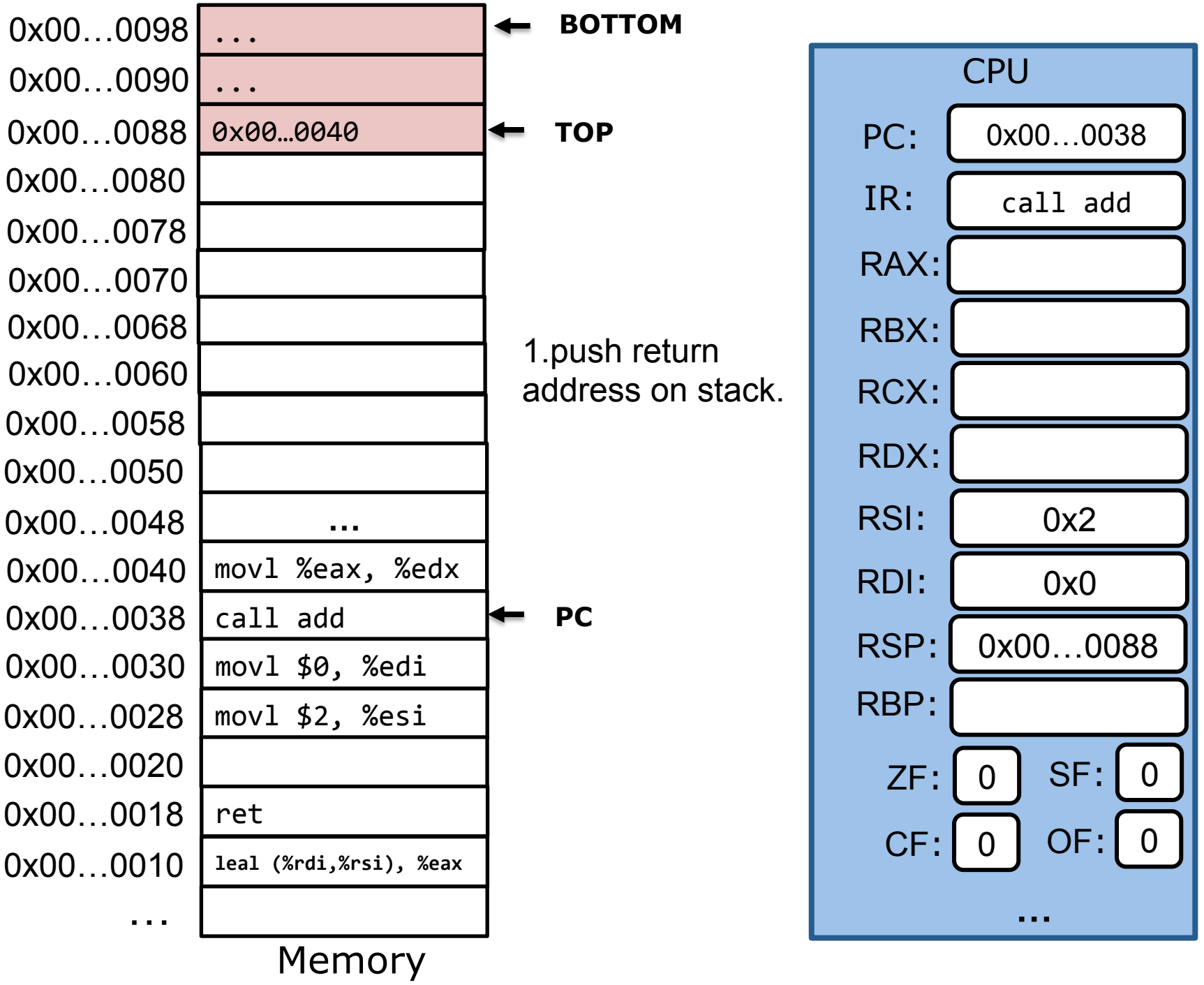

0x00…0098
...
BOTTOM
0x00…0090
...
TOP
0x00…0088
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
0x00…0038
call add
0x00…0030
movl $0, %edi
0x00…0028
movl $2, %esi
PC
0x00…0020
0x00…0018
ret
0x00…0010
leal (%rdi,%rsi), %eax
...
Memory
CPU
PC:
0x00…0028
IR:
movl $2, %esi
RAX:
RBX:
RCX:
RDX:
RSI:
RDI:
RSP:
0x00…0090
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...

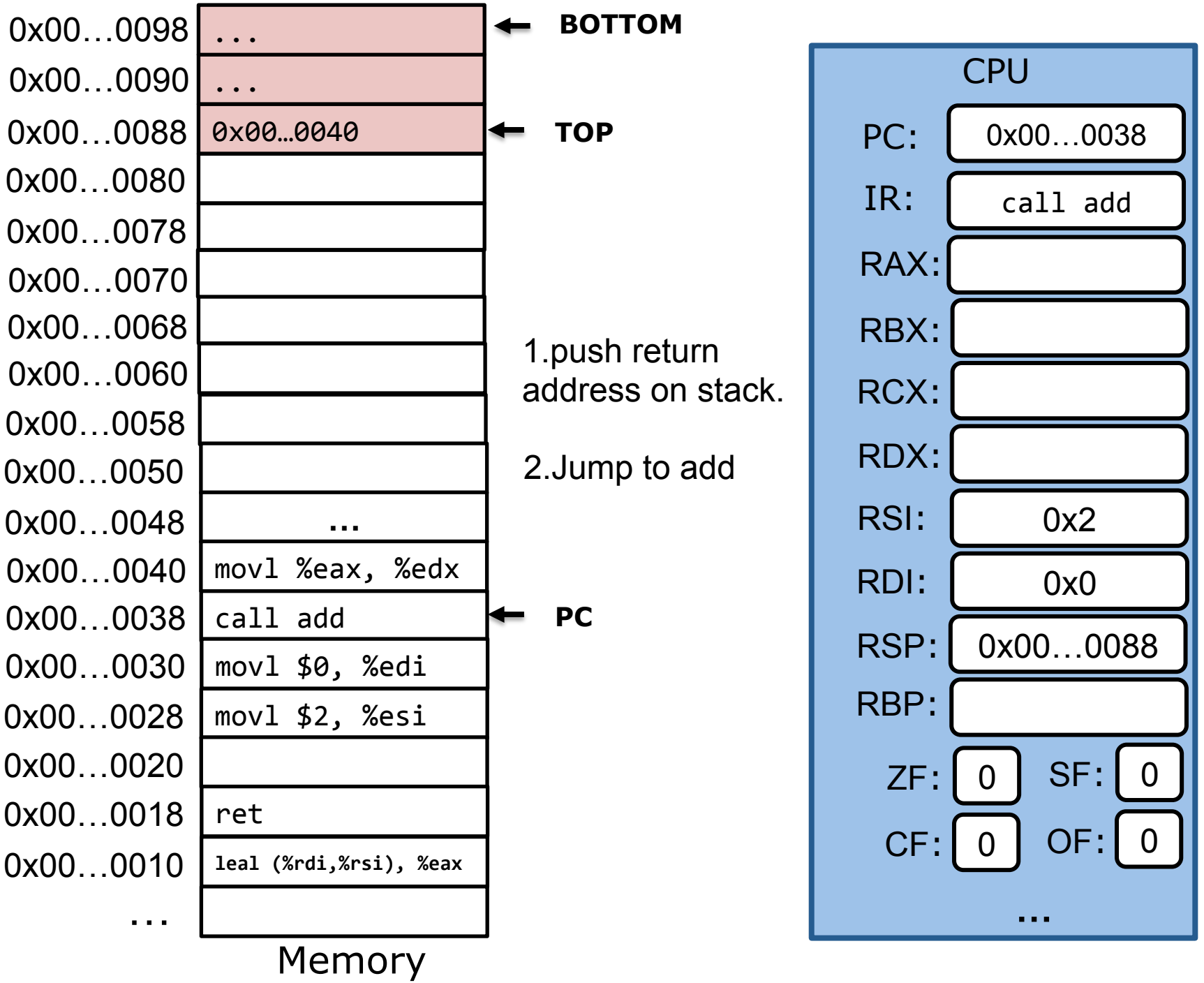

0x00…0098
...
BOTTOM
0x00…0090
...
TOP
0x00…0088
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
0x00…0038
call add
0x00…0030
movl $0, %edi
0x00…0028
movl $2, %esi
PC
0x00…0020
0x00…0018
ret
0x00…0010
leal (%rdi,%rsi), %eax
...
Memory
CPU
PC:
0x00…0028
IR:
movl $2, %esi
RAX:
RBX:
RCX:
RDX:
RSI:
0x2
RDI:
RSP:
0x00…0090
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...

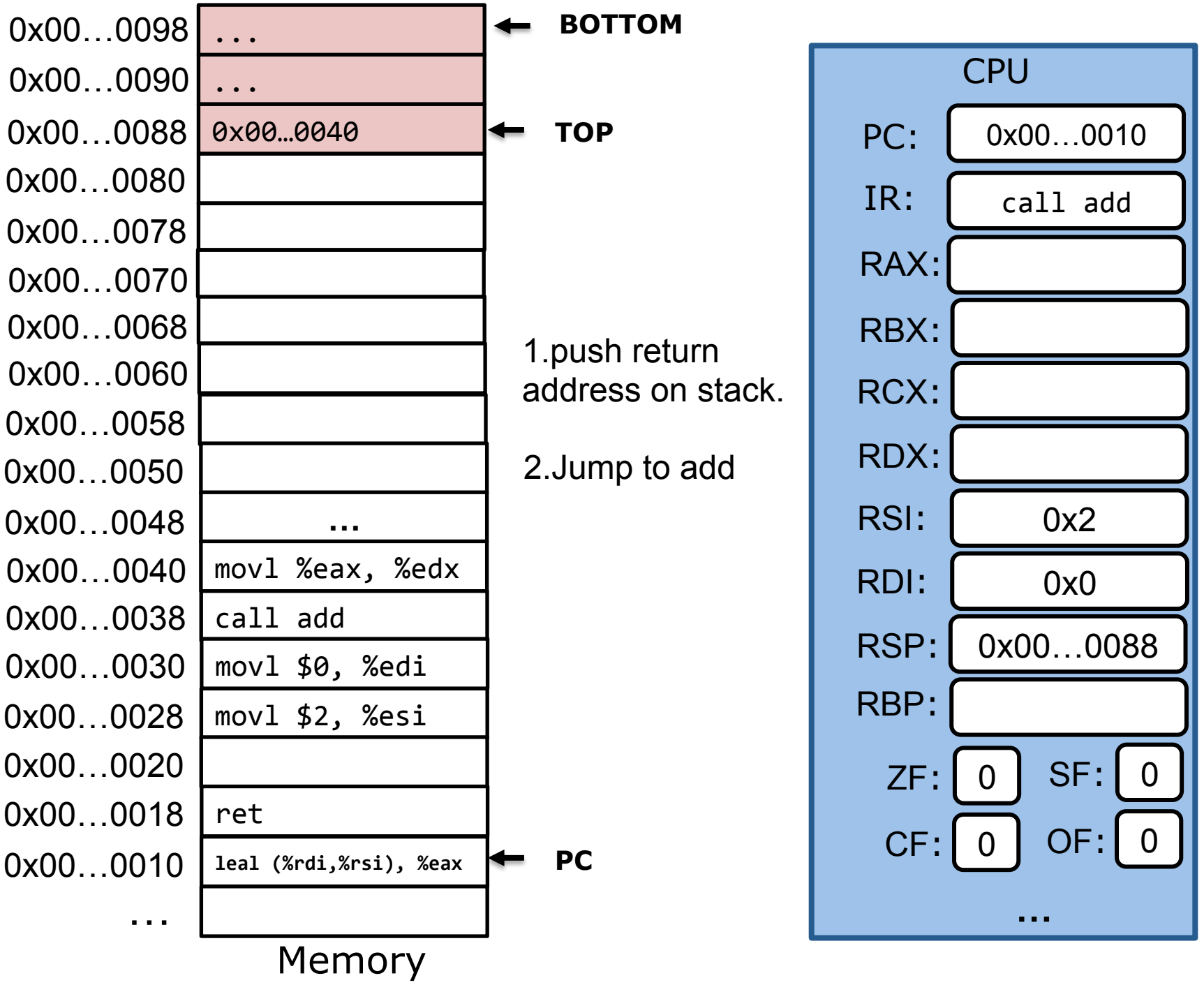

0x00…0098
...
BOTTOM
0x00…0090
...
TOP
0x00…0088
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
0x00…0038
call add
0x00…0030
movl $0, %edi
PC
0x00…0028
movl $2, %esi
0x00…0020
0x00…0018
ret
0x00…0010
leal (%rdi,%rsi), %eax
...
Memory
CPU
PC:
0x00…0030
IR:
movl $9, %edi
RAX:
RBX:
RCX:
RDX:
RSI:
0x2
RDI:
0x0
RSP:
0x00…0090
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...

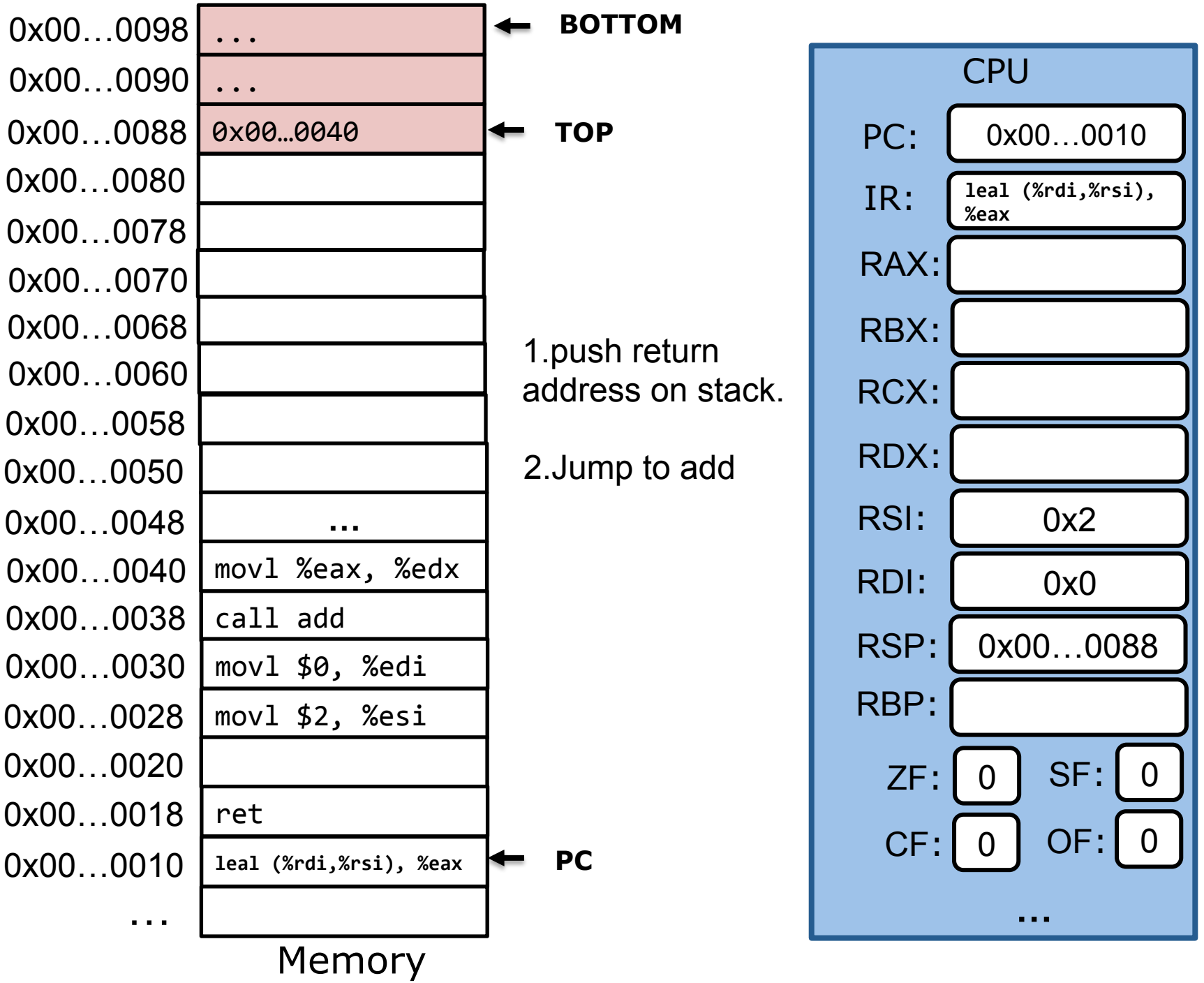

0x00…0098
...
BOTTOM
0x00…0090
...
TOP
0x00…0088
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
0x00…0038
call add
PC
0x00…0030
movl $0, %edi
0x00…0028
movl $2, %esi
0x00…0020
0x00…0018
ret
0x00…0010
leal (%rdi,%rsi), %eax
...
Memory
CPU
PC:
0x00…0038
IR:
call add
RAX:
RBX:
RCX:
RDX:
RSI:
0x2
RDI:
0x0
RSP:
0x00…0090
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...

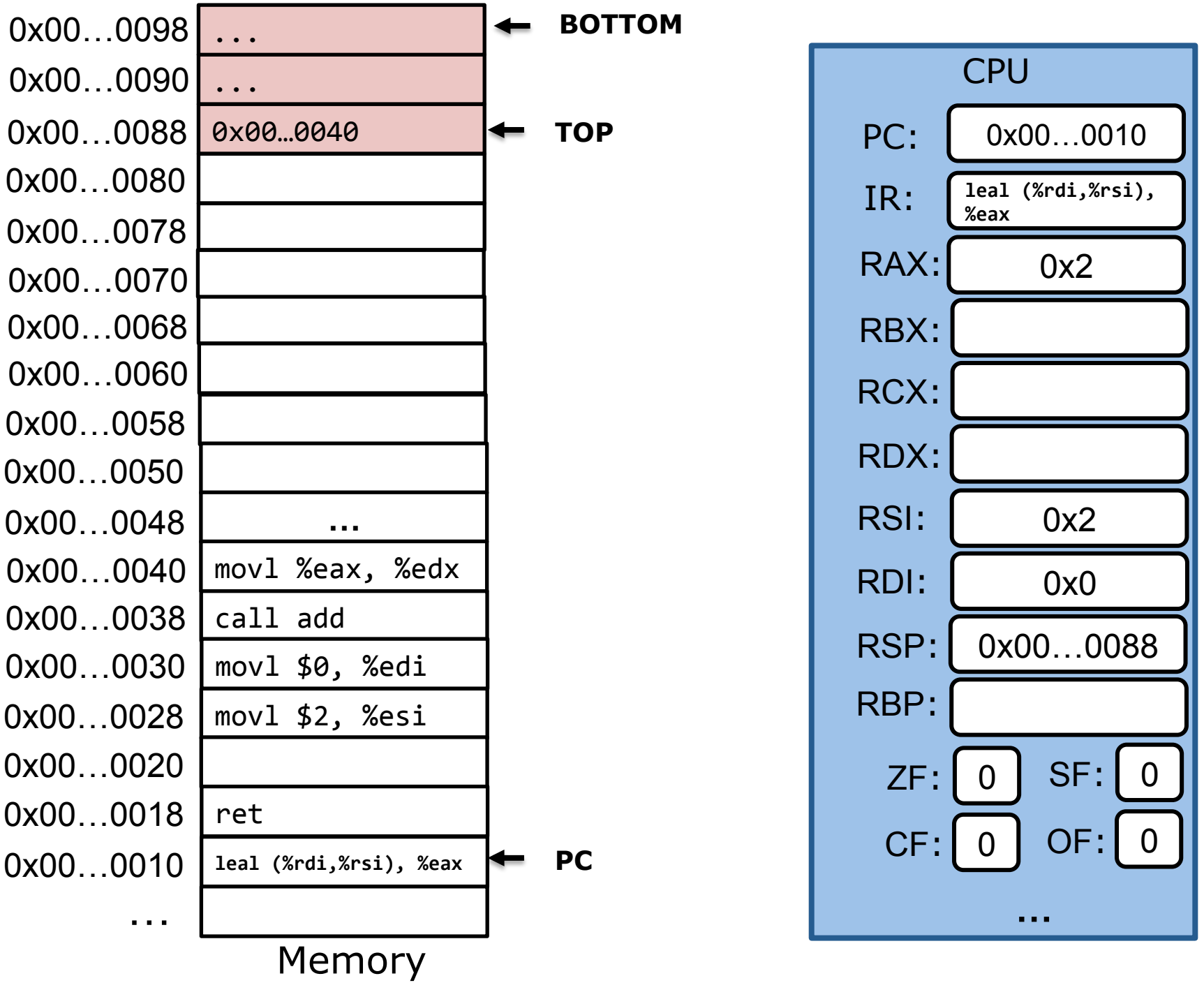

0x00…0098
...
BOTTOM
0x00…0090
...
TOP
0x00…0088
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
0x00…0038
call add
PC
0x00…0030
movl $0, %edi
0x00…0028
movl $2, %esi
0x00…0020
0x00…0018
ret
0x00…0010
leal (%rdi,%rsi), %eax
...
Memory
1.push return address on stack.
CPU
PC:
0x00…0038
IR:
call add
RAX:
RBX:
RCX:
RDX:
RSI:
0x2
RDI:
0x0
RSP:
0x00…0090
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...

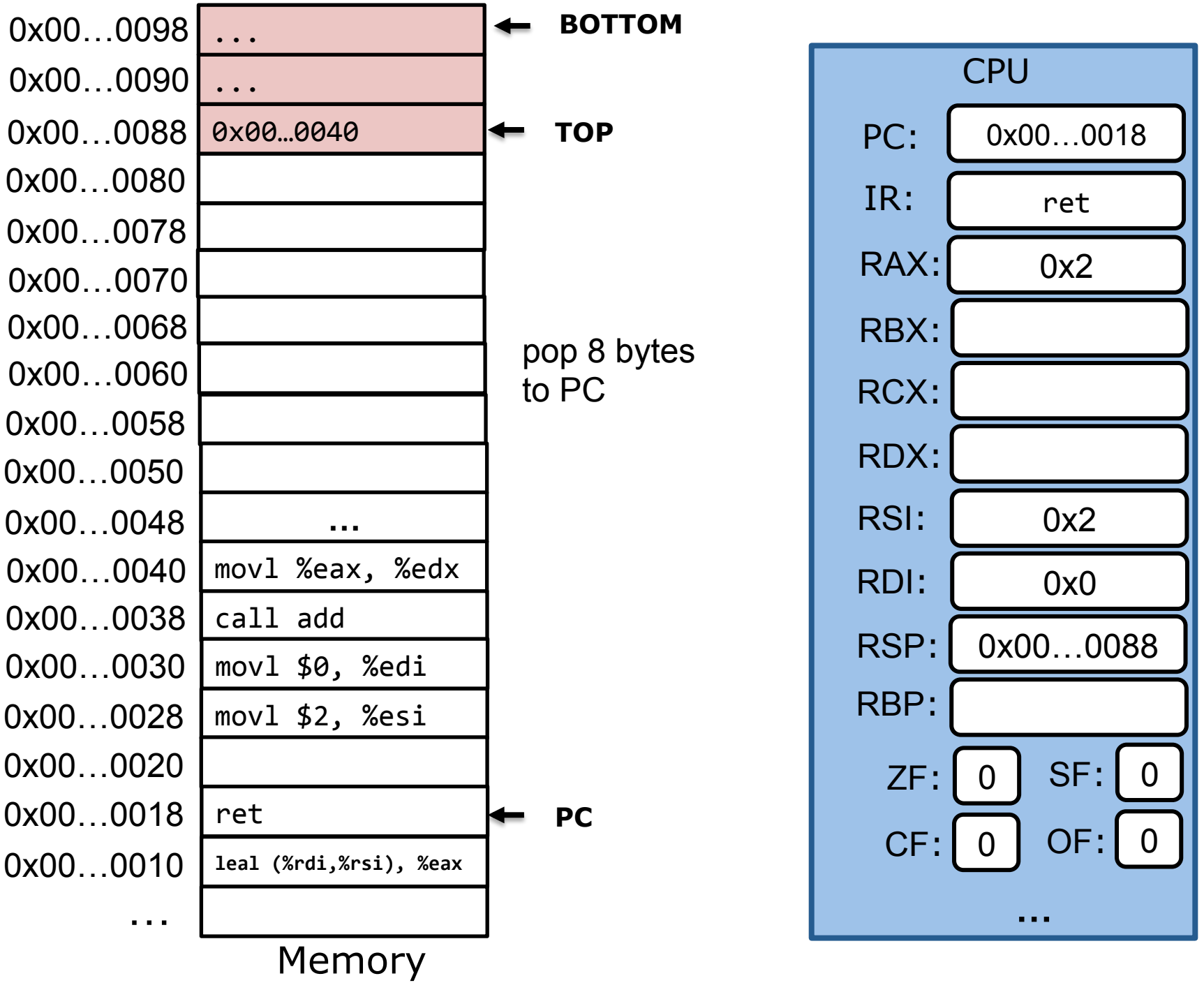

0x00…0098
...
BOTTOM
0x00…0090
...
0x00…0088
0x00…0040
TOP
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
0x00…0038
call add
PC
0x00…0030
movl $0, %edi
0x00…0028
movl $2, %esi
0x00…0020
0x00…0018
ret
0x00…0010
leal (%rdi,%rsi), %eax
...
Memory
1.push return address on stack.
CPU
PC:
0x00…0038
IR:
call add
RAX:
RBX:
RCX:
RDX:
RSI:
0x2
RDI:
0x0
RSP:
0x00…0088
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...

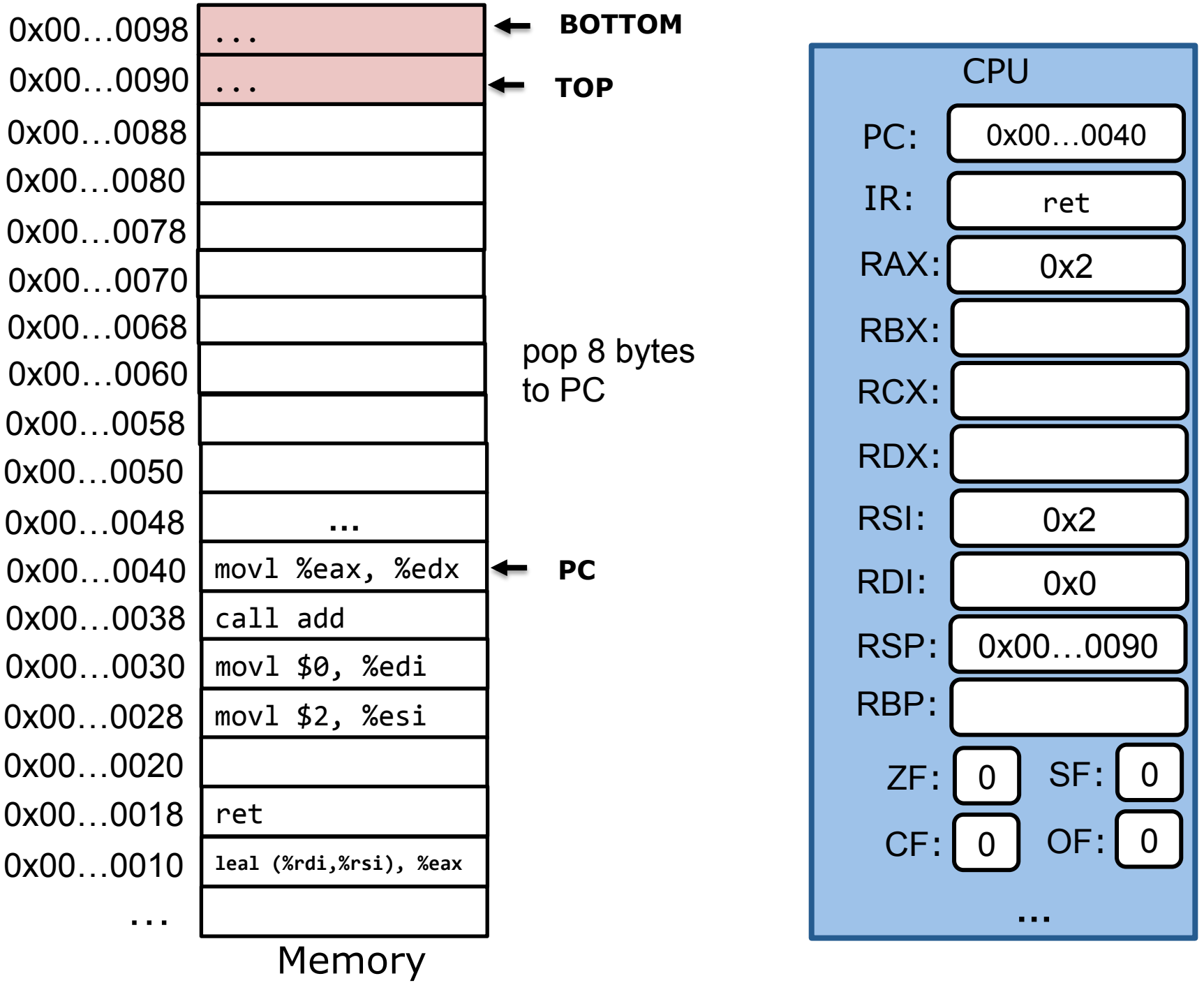

0x00…0098
...
BOTTOM
0x00…0090
...
0x00…0088
0x00…0040
TOP
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
0x00…0038
call add
PC
0x00…0030
movl $0, %edi
0x00…0028
movl $2, %esi
0x00…0020
0x00…0018
ret
0x00…0010
leal (%rdi,%rsi), %eax
...
Memory
1.push return address on stack.
2.Jump to add
CPU
PC:
0x00…0038
IR:
call add
RAX:
RBX:
RCX:
RDX:
RSI:
0x2
RDI:
0x0
RSP:
0x00…0088
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...


0x00…0098
...
BOTTOM
0x00…0090
...
0x00…0088
0x00…0040
TOP
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
0x00…0038
call add
0x00…0030
movl $0, %edi
0x00…0028
movl $2, %esi
0x00…0020
0x00…0018
ret
0x00…0010
leal (%rdi,%rsi), %eax
PC
...
Memory
1.push return address on stack.
2.Jump to add
CPU
PC:
0x00…0010
IR:
call add
RAX:
RBX:
RCX:
RDX:
RSI:
0x2
RDI:
0x0
RSP:
0x00…0088
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...


0x00…0098
...
BOTTOM
0x00…0090
...
0x00…0088
0x00…0040
TOP
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
0x00…0038
call add
0x00…0030
movl $0, %edi
0x00…0028
movl $2, %esi
0x00…0020
0x00…0018
ret
0x00…0010
leal (%rdi,%rsi), %eax
PC
...
Memory
1.push return address on stack.
2.Jump to add
CPU
PC:
0x00…0010
IR:
leal (%rdi,%rsi), %eax
RAX:
RBX:
RCX:
RDX:
RSI:
0x2
RDI:
0x0
RSP:
0x00…0088
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...


0x00…0098
...
BOTTOM
0x00…0090
...
0x00…0088
0x00…0040
TOP
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
0x00…0038
call add
0x00…0030
movl $0, %edi
0x00…0028
movl $2, %esi
0x00…0020
0x00…0018
ret
0x00…0010
leal (%rdi,%rsi), %eax
PC
...
Memory
CPU
PC:
0x00…0010
IR:
leal (%rdi,%rsi), %eax
RAX:
0x2
RBX:
RCX:
RDX:
RSI:
0x2
RDI:
0x0
RSP:
0x00…0088
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...

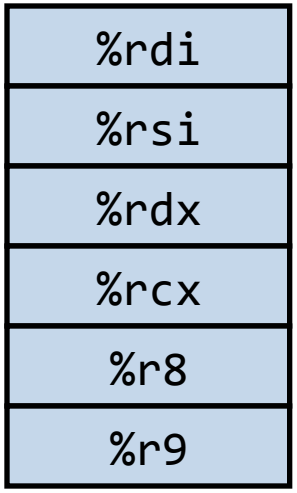

0x00…0098
...
BOTTOM
0x00…0090
...
0x00…0088
0x00…0040
TOP
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
pop 8 bytes
to PC
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
0x00…0038
call add
0x00…0030
movl $0, %edi
0x00…0028
movl $2, %esi
0x00…0020
0x00…0018
ret
PC
0x00…0010
leal (%rdi,%rsi), %eax
...
Memory
CPU
PC:
0x00…0018
IR:
ret
RAX:
0x2
RBX:
RCX:
RDX:
RSI:
0x2
RDI:
0x0
RSP:
0x00…0088
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...


0x00…0098
...
BOTTOM
0x00…0090
...
TOP
0x00…0088
0x00…0080
0x00…0078
0x00…0070
0x00…0068
0x00…0060
pop 8 bytes
to PC
0x00…0058
0x00…0050
0x00…0048
...
0x00…0040
movl %eax, %edx
PC
0x00…0038
call add
0x00…0030
movl $0, %edi
0x00…0028
movl $2, %esi
0x00…0020
0x00…0018
ret
0x00…0010
leal (%rdi,%rsi), %eax
...
Memory
CPU
PC:
0x00…0040
IR:
ret
RAX:
0x2
RBX:
RCX:
RDX:
RSI:
0x2
RDI:
0x0
RSP:
0x00…0090
RBP:
ZF:
0
SF:
0
CF:
0
OF:
0
...

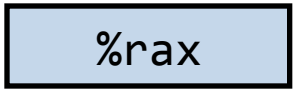
C's calling convention: args/return values

Registers

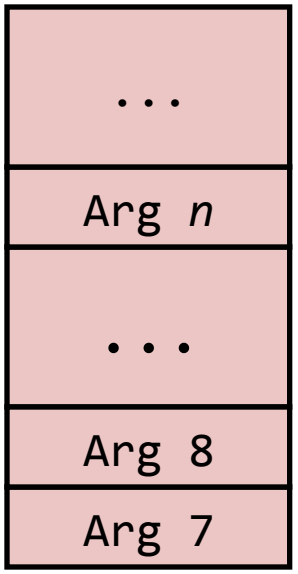
First 6 arguments



Return value



Stack



Only allocate stack space when needed

C's calling convention: args/return values

Registers

- First 6 Arguments: %rdi, %rsi, %rdx, %rcx, %r8, %9
- Return value: %rax

```
int add(int a, int b, int c, int d, int e, int f, int g, int h) {
  int r = a + b + c + d + e + f + g + h;
  return r;
}

int main() {

    int c = add(1, 2, 3, 4, 5, 6, 7, 8);
    printf("%d\b", c);
    return 0;
}
```

C's calling convention: args/return values

```
int add(int a, int b, int c, int d, int e, int f, int g, int h) {
  int r = a + b + c + d + e + f + g + h;
  return r;
}
```

```
main:
    subq    $8, %rsp
    pushq   $8
    pushq   $7
    movl    $6, %r9d
    movl    $5, %r8d
    movl    $4, %ecx
    movl    $3, %edx
    movl    $2, %esi
    movl    $1, %edi
    call    add
```

```
add:
    addl    %esi, %edi
    addl    %edi, %edx
    addl    %edx, %ecx
    addl    %r8d, %ecx
    addl    %r9d, %ecx
    movl    %ecx, %eax
    addl    8(%rsp), %eax
    addl    16(%rsp), %eax
    ret
```

How to allocate/deallocate local variables?

Use registers whenever possible

Allocate local variables on the stack

- `subq $0x8,%rsp` //allocate 8 bytes
- `movq $1, 8(%rsp)` //store 1 in the allocated 8 bytes

Calling convention: Register Saving

When procedure `f` calls `g`:

- `f` is the caller, `g` is the callee

Can caller assume register values do not change when callee returns?

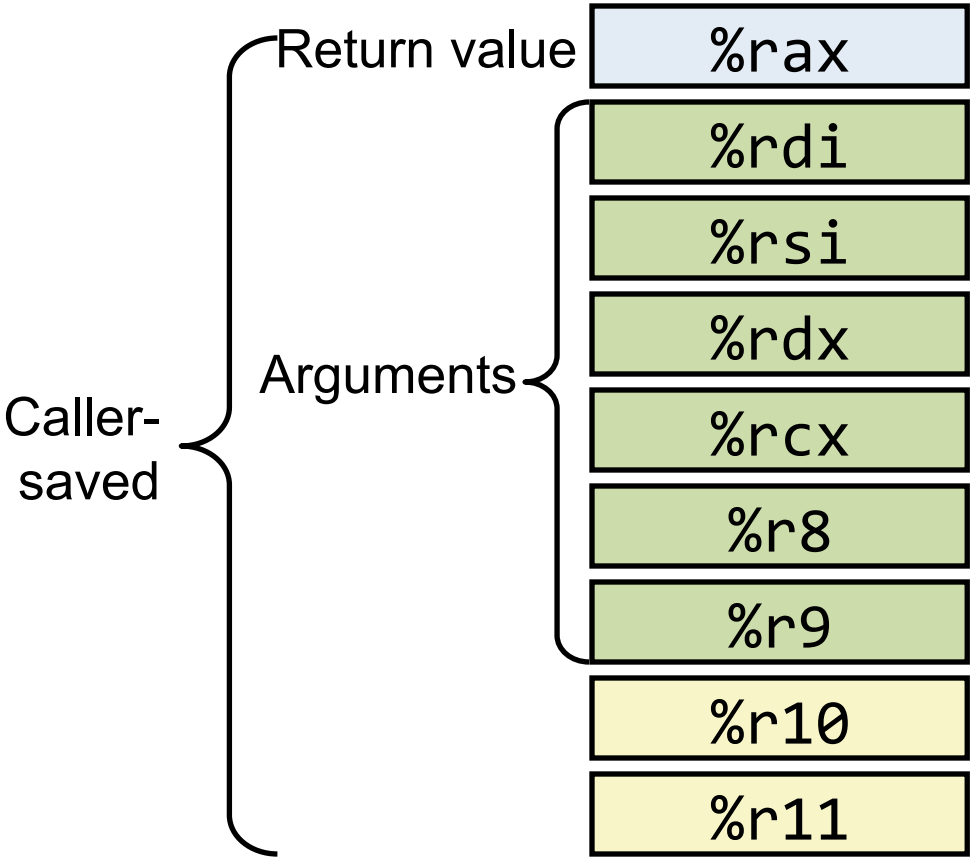
If not, caller must save register values (in memory) that it needs to use them later

Calling convention: register saving

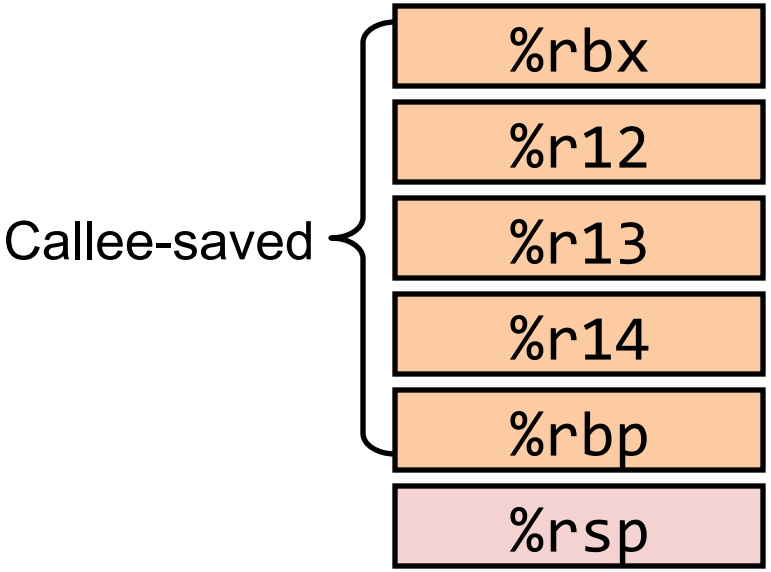
Some registers are “caller saved”, others are “callee saved”

- Caller saved
  - Caller saves “caller saved” registers on stack before the call
- Callee saved
  - Callee saves “callee saved” registers on stack before using
  - Callee restores them before returning to caller

C' calling convention: Register Usage



Callee can directly use these registers



Caller can assume these registers are unchanged.

Example

```
int add2(int a, int b)
{
  return a + b;
}
```

```
add2:
    leal    (%rdi,%rsi), %eax
    ret
```

```
int add3(int a, int b, int c)
{
  int r = add2(a, b);
  r = r + c;
  return r;
}
```

```
add3:
    pushq   %rbx
    movl    %edx, %ebx
    movl    $0, %eax
    call    add2
    addl    %ebx, %eax
    popq     %rbx
    ret
```

Registers

First 6 Arguments: %rdi, %rsi, %rdx, %rcx, %r8, %9

Return value: %rax

Example

```
int add2(int a, int b)
{
  return a + b;
}
```

```
int add3(int a, int b, int c)
{
  int r = add2(a, b);
  r = r + c;
  return r;
}
```

```
add2:
    leal    (%rdi,%rsi), %eax
    ret
```

```
add3:
    pushq   %rbx        # use %rbx to keep r
    movl    %edx, %ebx
    movl    $0, %eax
    call    add2
    addl    %ebx, %eax
    popq    %rbx   # restore %rbx before ret
    ret
```

Registers

First 6 Arguments: %rdi, %rsi, %rdx, %rcx, %r8, %9

Return value: %rax