Procedures and the **Call Stack**

Topics

- Procedures
- Call stack
- Procedure/stack instructions
- Calling conventions
- Register-saving conventions

Implementing Procedures

How does a caller pass **arguments** to a procedure?

How does a caller receive a **return value** from a procedure?

Where does a procedure store **local variables**?

How does a procedure know **where to return**
(what code to execute next when done)?

How do procedures **share limited registers** and **memory**?

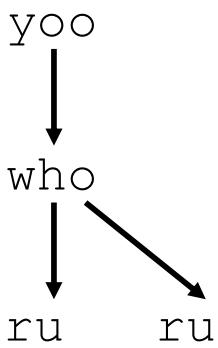
Call Chain

```
yoo(…)
{
    •
    •
    who();
    •
    •
}
```

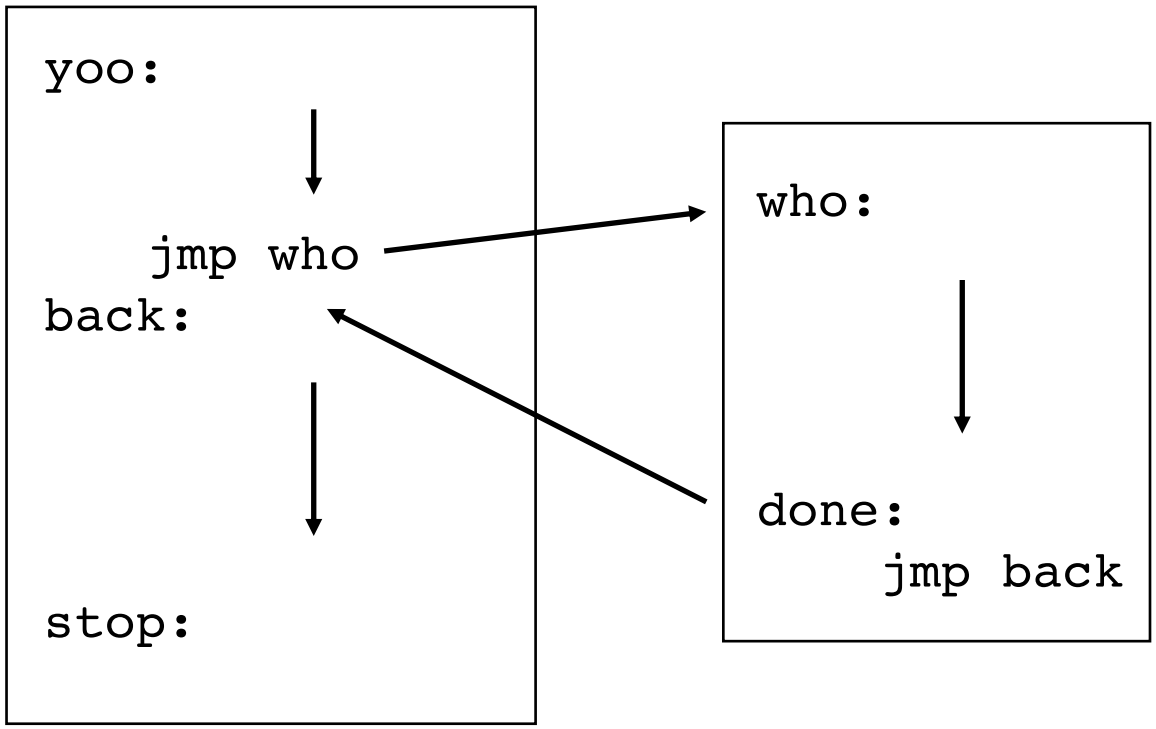
```
who(…)
{
    • • •
    ru();
    • • •
    ru();
    • • •
}
```

```
ru(…)
{
    •
    •
    •
}
```

Example
Call Chain

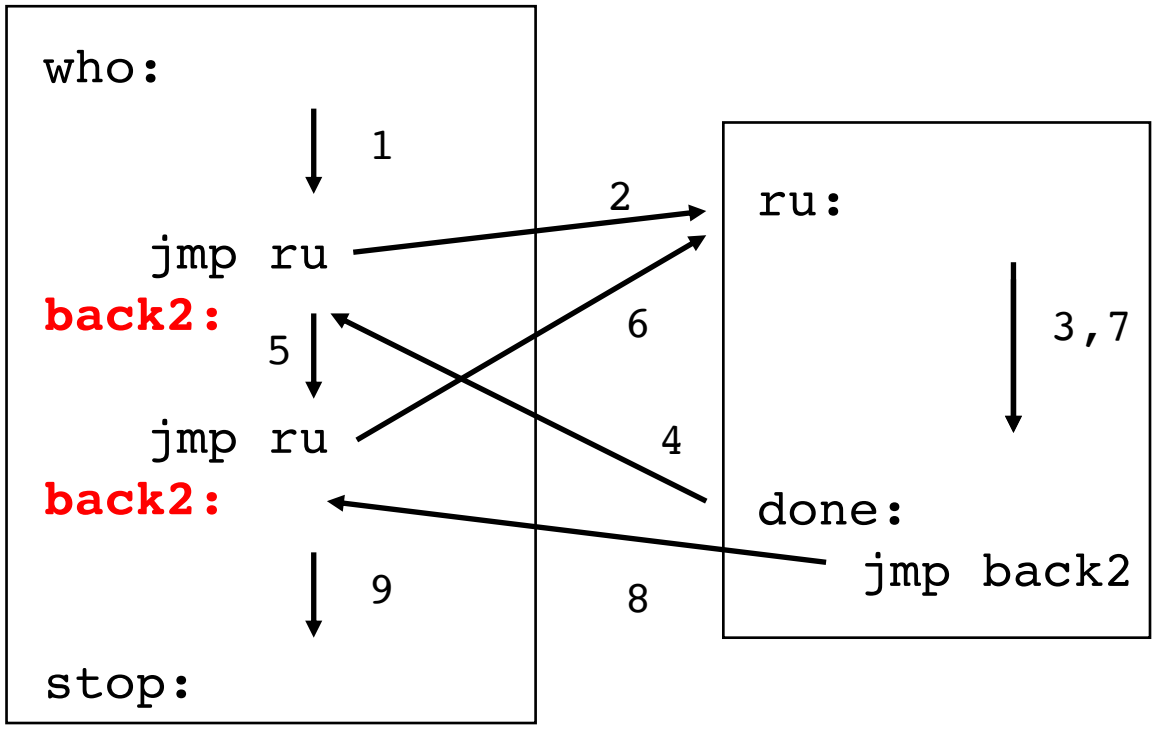


First Try…



But what if we want to call a function from multiple places in the code?

First Try… broken!



Implementing Procedures

How does a caller pass **arguments** to a procedure?

How does a caller receive a **return value** from a procedure?

Where does a procedure store **local variables**?

How does a procedure know **where to return**
(what code to execute next when done)?

How do procedures **share limited registers** and **memory**?

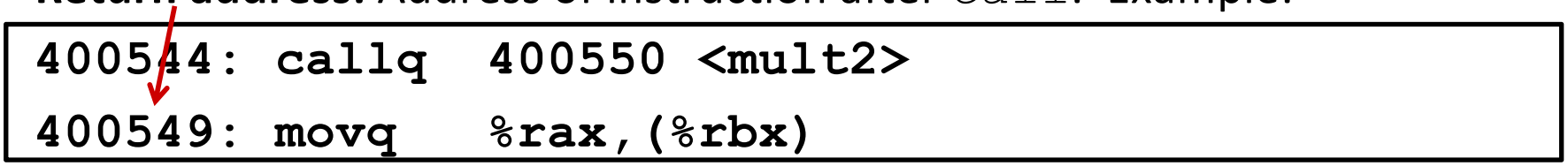
All these need **separate storage *per call!***
(not just per procedure)

Procedure Control Flow Instructions

Procedure call: `callq` *`label`*

1. Push return address on stack
2. Jump to *`label`*

Return address: Address of instruction after `call`. Example:



Procedure return: `retq`

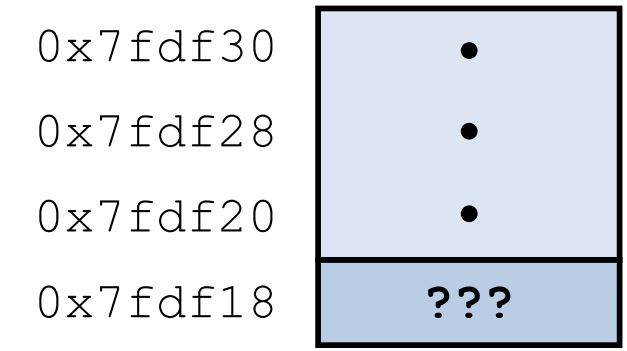
1. Pop return address from stack
2. Jump to return address

Call Example

```
0000000000400540 <multstore>:
  •
  •
  400544: callq  400550 <mult2>
  400549: mov    %rax,(%rbx)
  •
```

```
0000000000400550 <mult2>:
  400550:  mov    %rdi,%rax
  •
  •
  400557:  retq
```

Before `callq`



%rsp **0x7fdf20**

%rip **0x400544**

After `callq`

%rsp **0x7fdf18**

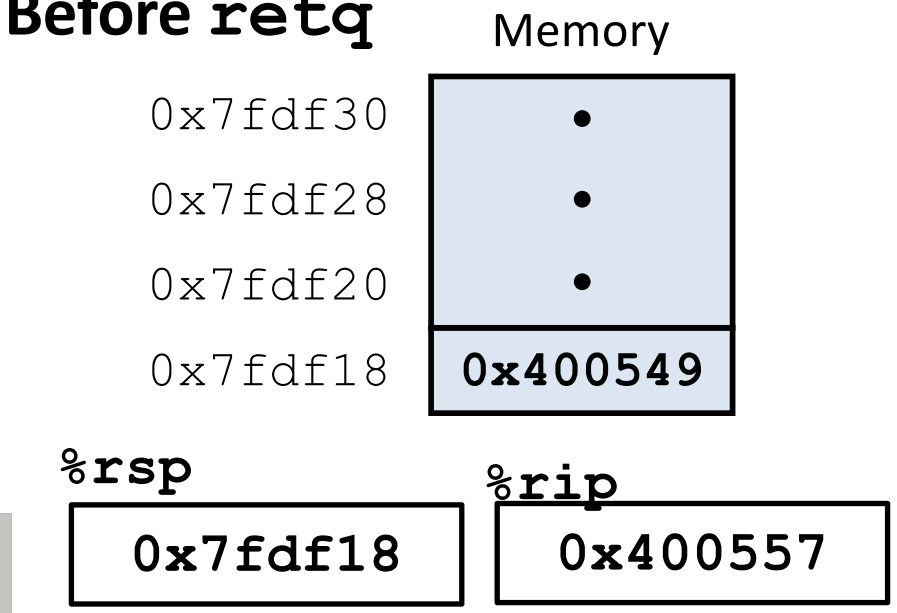
%rip **0x400550**

Return Example

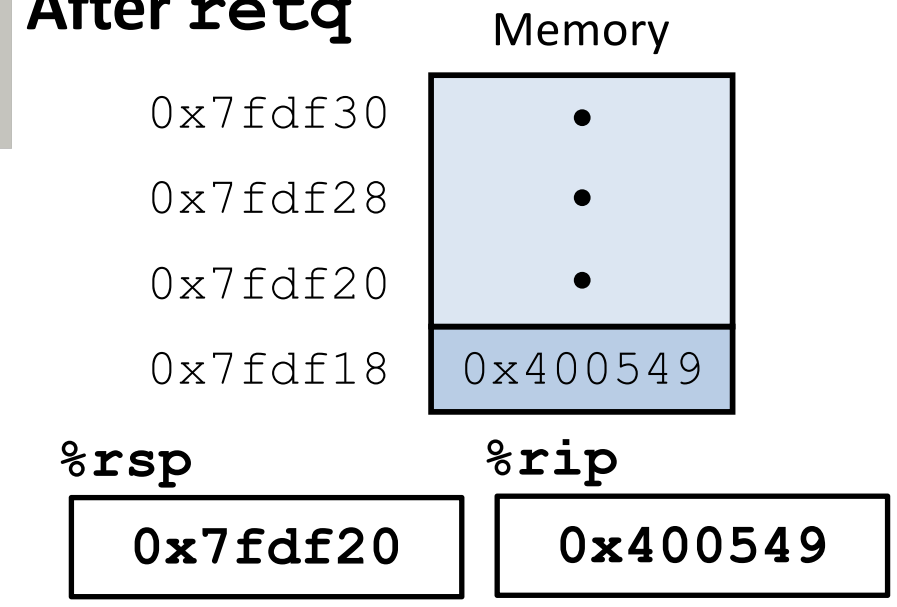
```
0000000000400540 <multstore>:
  •
  •
  400544: callq  400550 <mult2>
  400549: mov    %rax,(%rbx)
  •
```

```
0000000000400550 <mult2>:
  400550:  mov    %rdi,%rax
  •
  •
  400557:  retq
```

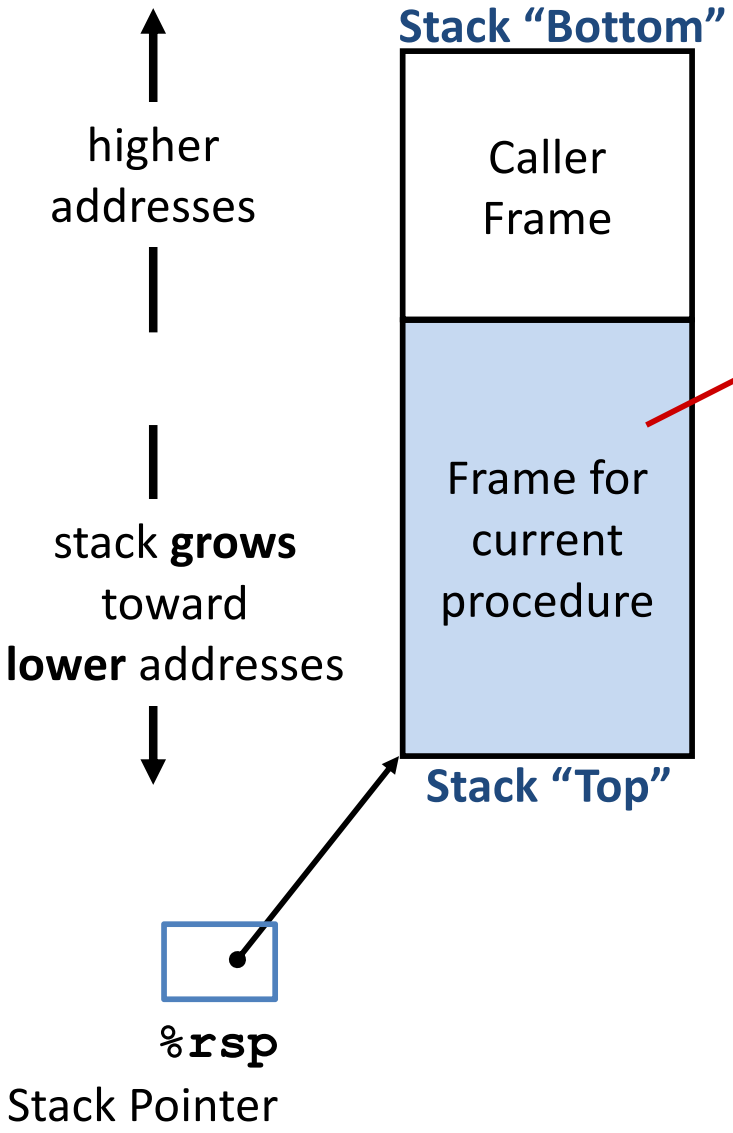
Before `retq`



After `retq`



Stack frames support procedure calls.



Contents

- Local variables
- Function arguments (after first 6)
- Return address
- Temporary space

Management

Space allocated when procedure is entered

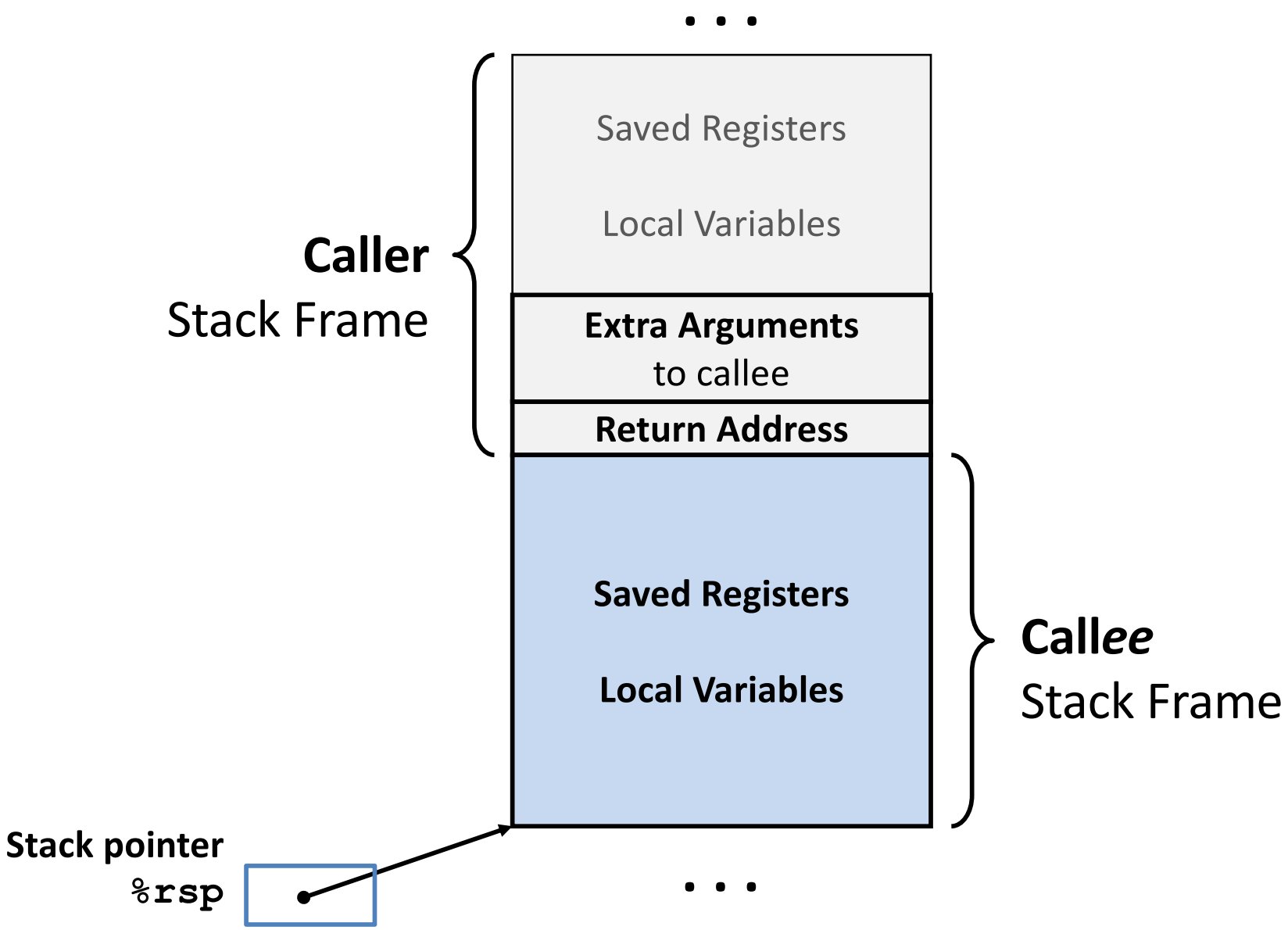
"Setup" code

Space deallocated before return

"Finish" code

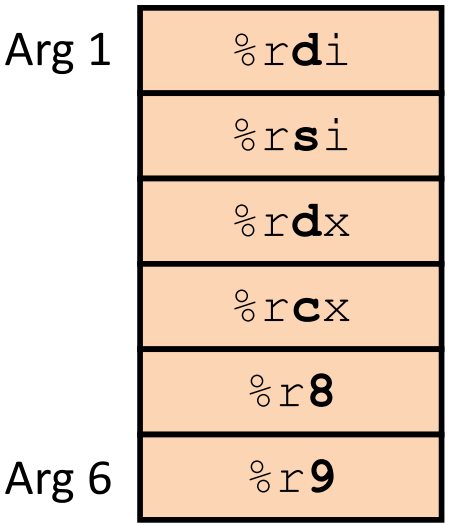
Why not just give every *procedure* a permanent chunk of memory to hold its local variables, etc?

Stack Frames



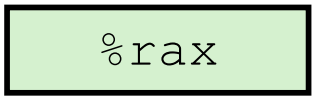
Procedure Data Flow Conventions

First 6 arguments passed in registers

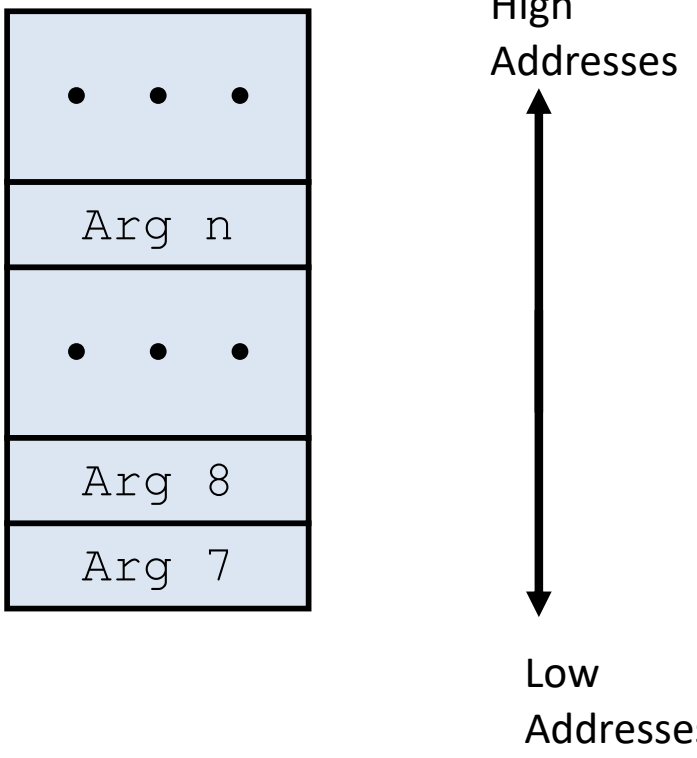


Diane's Silk Dress Costs $8 9

Return value



Remaining arguments passed on stack (in memory)



Allocate stack space only when needed.

A Puzzle

C function body:

```
*p = d;
return x - c;
```

assembly:

```
movsbl %dl,%edx
movl   %edx,(%rsi)
movswl %di,%edi
subl   %edi,%ecx
movl   %ecx,%eax
```

Write the C function header, types, and order of parameters.

movsbl = move sign-extending a byte to a long (4-byte)
movswl = move sign-extending a word (2-byte) to a long (4-byte)

Procedure Call / Stack Frame Example

```
long call_incr() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

Passes address of local variable (in stack).

```
call_incr:
400509:  subq  $8, %rsp
40050d:  movq  $240, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  (%rsp), %rax
400526:  addq  $8, %rsp
40052a:  retq
```

Uses memory through pointer.

```
long increment(long* p, long val) {
    long x = *p;
    long y = x + val;
    *p = y;
    return x;
}
```

```
increment:
4004cd:  movq  (%rdi), %rax
4004d0:  addq  %rax, %rsi
4004d3:  movq  %rsi, (%rdi)
4004d6:  retq
```

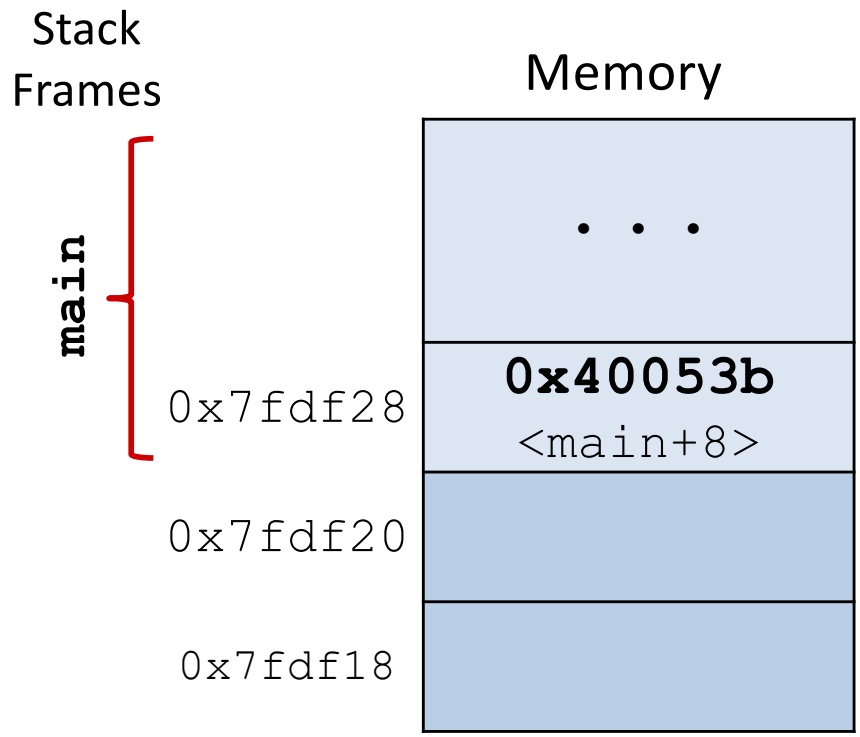
Procedure Call Example (step 0)

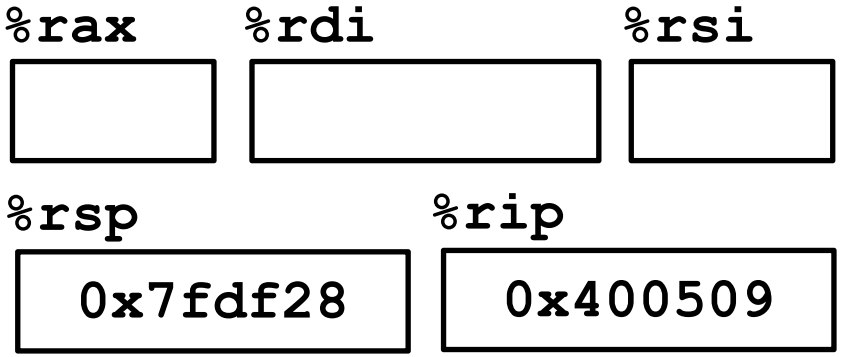
`main` called `call_incr`

```
long call_incr() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

```
call_incr:
400509:  subq  $8, %rsp
40050d:  movq  $240, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  (%rsp), %rax
400526:  addq  $8, %rsp
40052a:  retq
```

```
increment:
4004cd:  movq  (%rdi), %rax
4004d0:  addq  %rax, %rsi
4004d3:  movq  %rsi, (%rdi)
4004d6:  retq
```





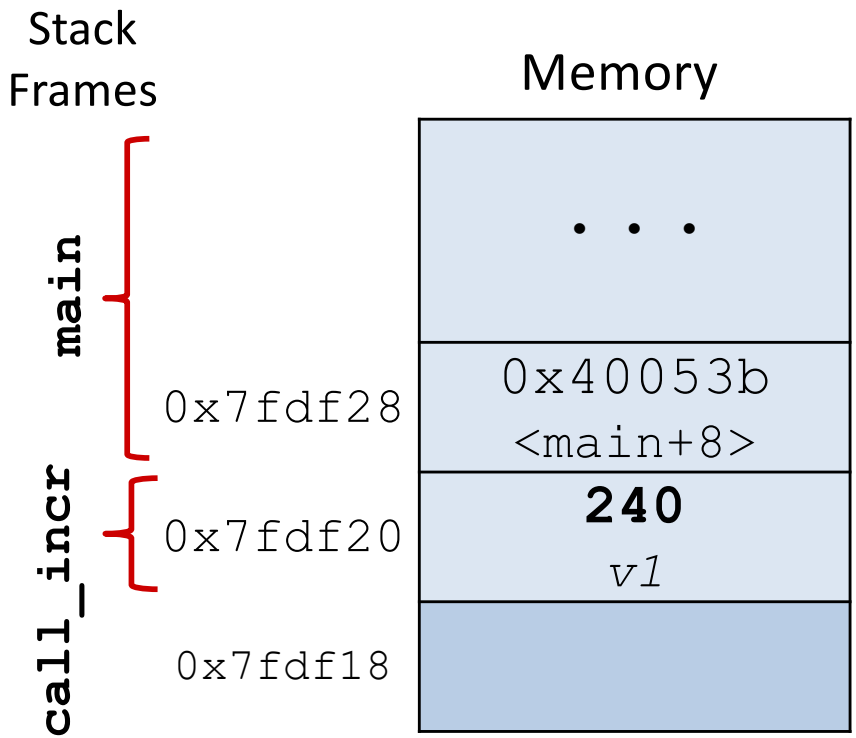
Procedure Call Example (step 1)

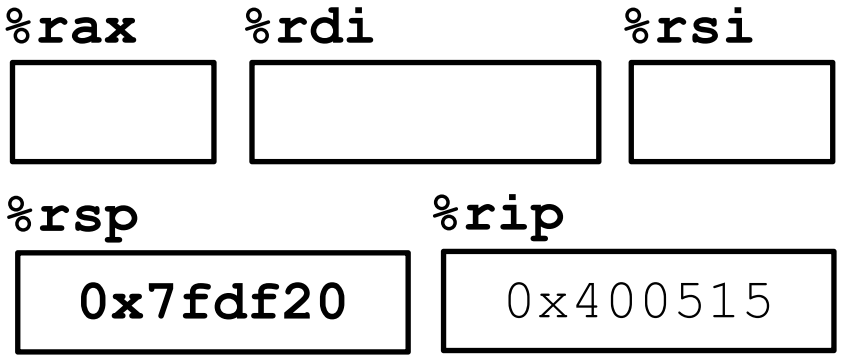
Allocate space for local vars

```
long call_incr() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

```
call_incr:
400509:  subq  $8, %rsp
40050d:  movq  $240, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  (%rsp), %rax
400526:  addq  $8, %rsp
40052a:  retq
```

```
increment:
4004cd:  movq  (%rdi), %rax
4004d0:  addq  %rax, %rsi
4004d3:  movq  %rsi, (%rdi)
4004d6:  retq
```





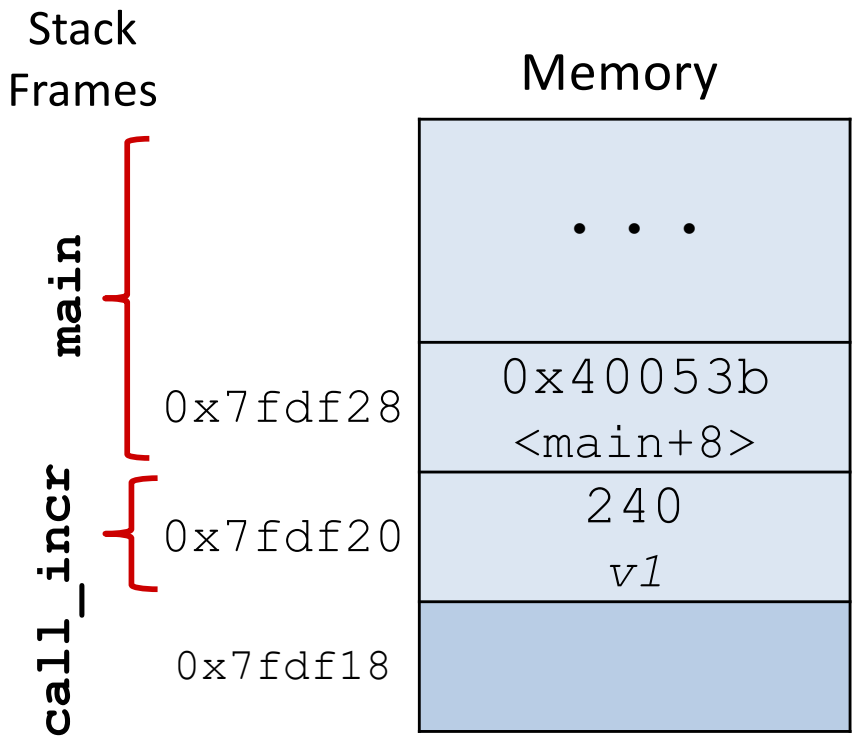
Procedure Call Example (step 2)

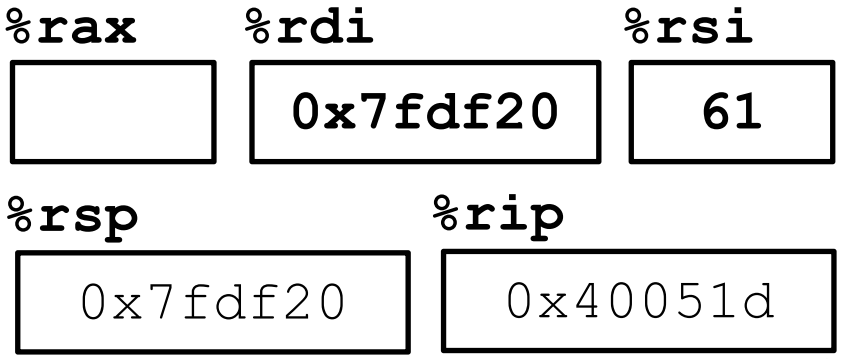
Set up args for call to `increment`

```
long call_incr() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

```
call_incr:
400509:  subq  $8, %rsp
40050d:  movq  $240, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  (%rsp), %rax
400526:  addq  $8, %rsp
40052a:  retq
```

```
increment:
4004cd:  movq  (%rdi), %rax
4004d0:  addq  %rax, %rsi
4004d3:  movq  %rsi, (%rdi)
4004d6:  retq
```





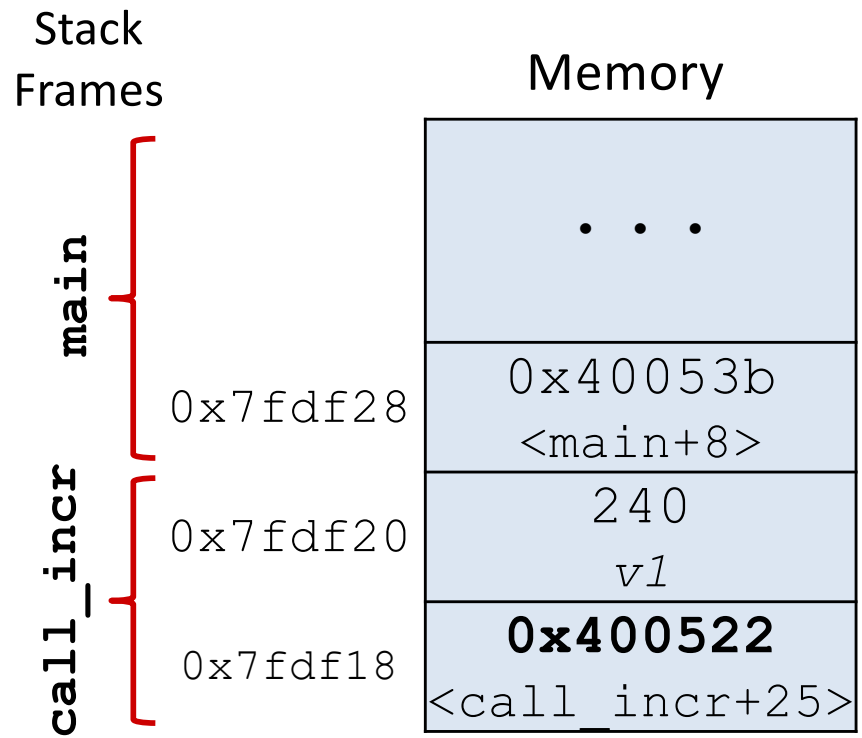
Procedure Call Example (step 3)

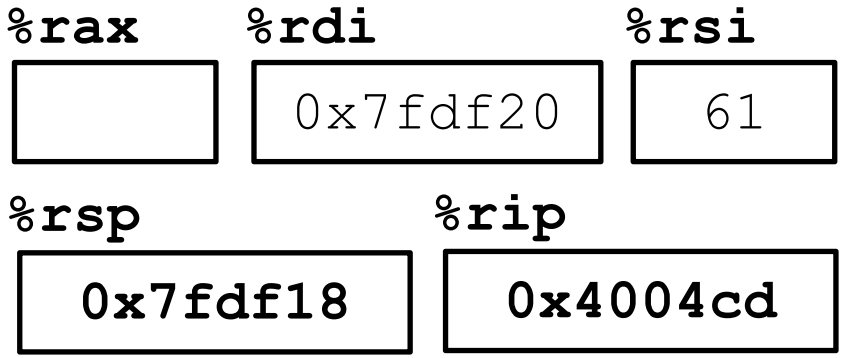
Call `increment`

```
long call_incr() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

```
call_incr:
400509:  subq   $8, %rsp
40050d:  movq   $240, (%rsp)
400515:  movq   %rsp, %rdi
400518:  movl   $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq   (%rsp), %rax
400526:  addq   $8, %rsp
40052a:  retq
```

```
increment:
4004cd:  movq   (%rdi), %rax
4004d0:  addq   %rax, %rsi
4004d3:  movq   %rsi, (%rdi)
4004d6:  retq
```





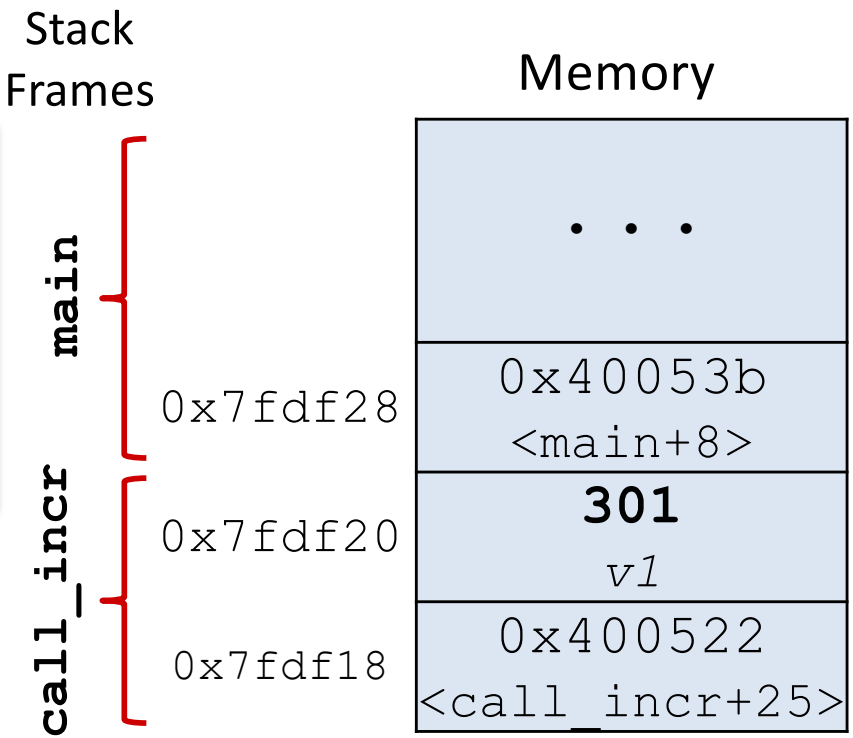
Procedure Call Example (step 4)

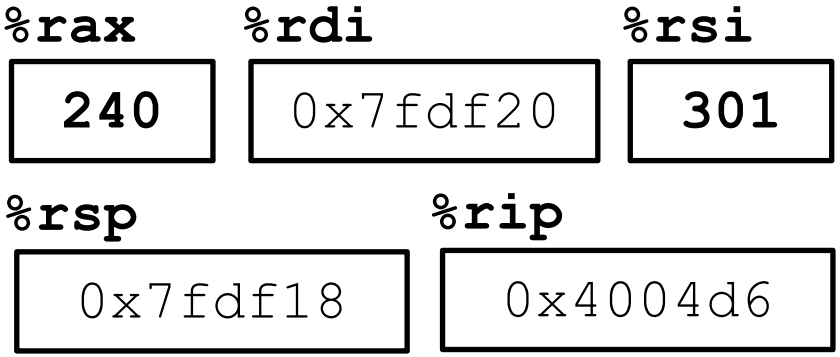
Run `increment`

```
long call_incr() {
    long increment(long* p, long val) {
        long x = *p;
        long y = x + val;
        *p = y;
        return x;
    }
}
```

```
ca
40050d:   movq   $240, (%rsp)
400515:   movq   %rsp, %rdi
400518:   movl   $61, %esi
40051d:   callq  4004cd <increment>
400522:   addq   (%rsp), %rax
400526:   addq   $8, %rsp
40052a:   retq
```

```
increment:
4004cd:   movq   (%rdi), %rax
4004d0:   addq   %rax, %rsi
4004d3:   movq   %rsi, (%rdi)
4004d6:   retq
```





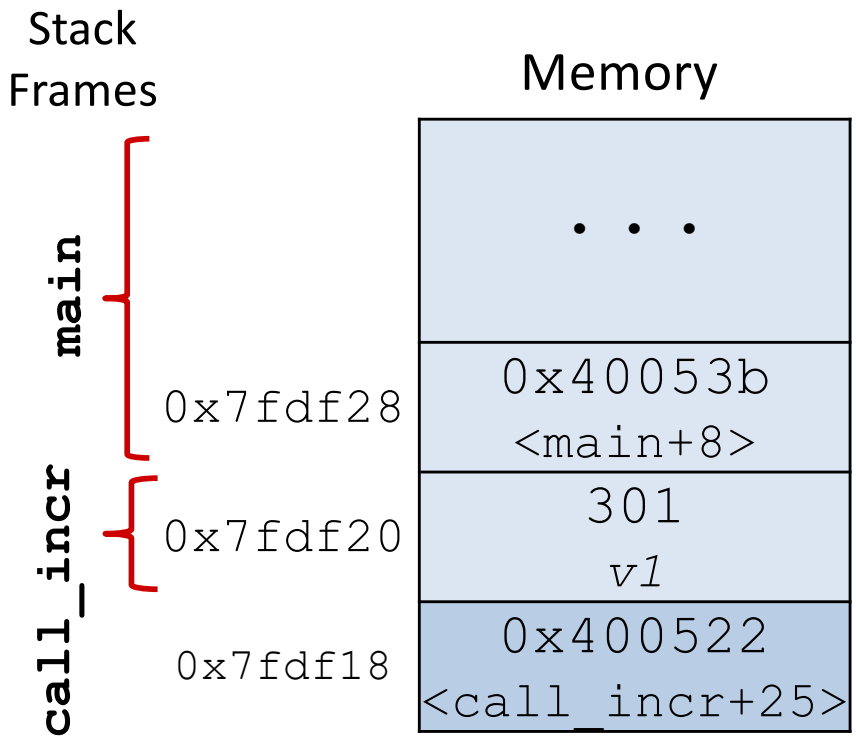
Procedure Call Example (step 5)

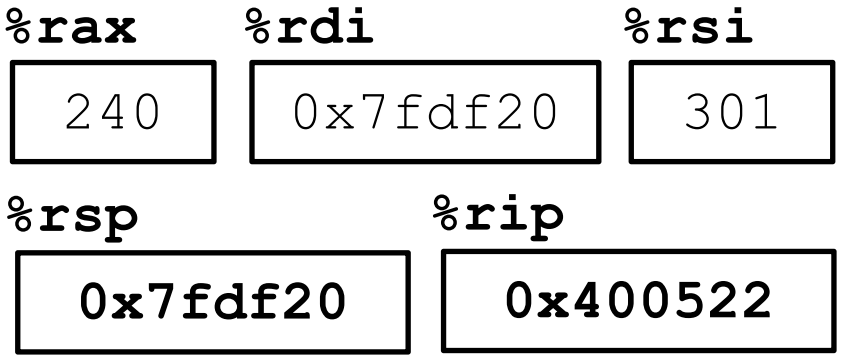
Return from `increment` to `call_incr`

```
long call_incr() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

```
call_incr:
400509:  subq  $8, %rsp
40050d:  movq  $240, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  (%rsp), %rax
400526:  addq  $8, %rsp
40052a:  retq
```

```
increment:
4004cd:  movq  (%rdi), %rax
4004d0:  addq  %rax, %rsi
4004d3:  movq  %rsi, (%rdi)
4004d6:  retq
```





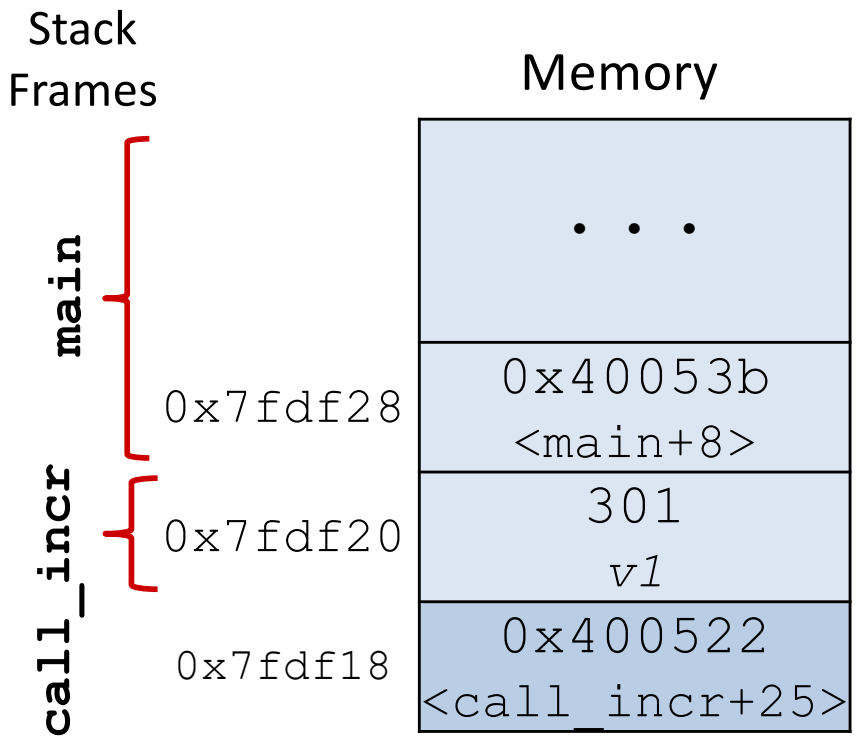
Procedure Call Example (step 6)

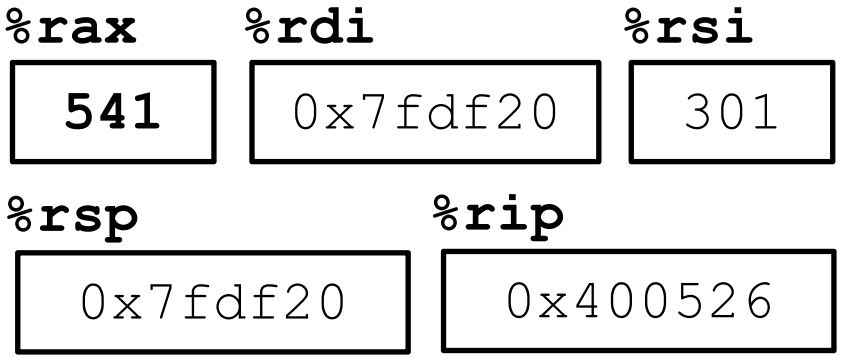
Prepare `call_incr` result

```
long call_incr() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

```
call_incr:
400509:  subq  $8, %rsp
40050d:  movq  $240, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  (%rsp), %rax
400526:  addq  $8, %rsp
40052a:  retq
```

```
increment:
4004cd:  movq  (%rdi), %rax
4004d0:  addq  %rax, %rsi
4004d3:  movq  %rsi, (%rdi)
4004d6:  retq
```





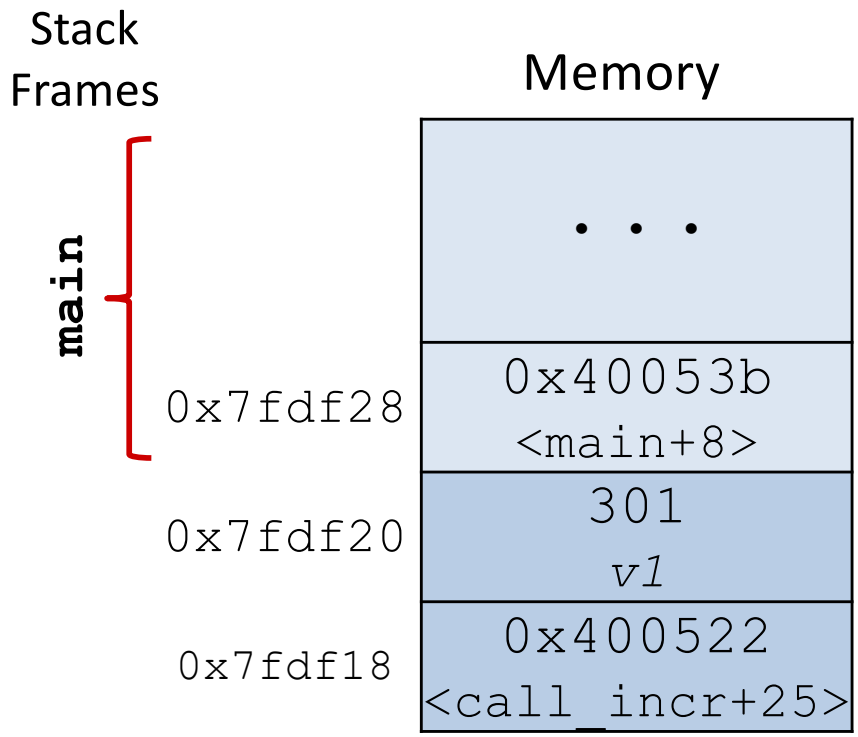
Procedure Call Example (step 7)

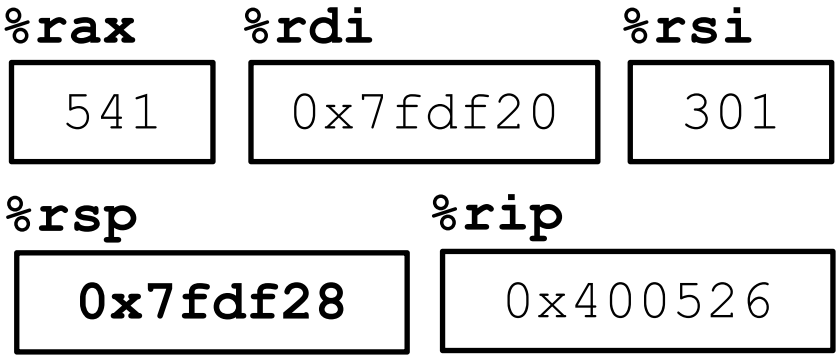
Deallocate space for local vars

```
long call_incr() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

```
call_incr:
400509:  subq  $8, %rsp
40050d:  movq  $240, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  (%rsp), %rax
400526:  addq  $8, %rsp
40052a:  retq
```

```
increment:
4004cd:  movq  (%rdi), %rax
4004d0:  addq  %rax, %rsi
4004d3:  movq  %rsi, (%rdi)
4004d6:  retq
```





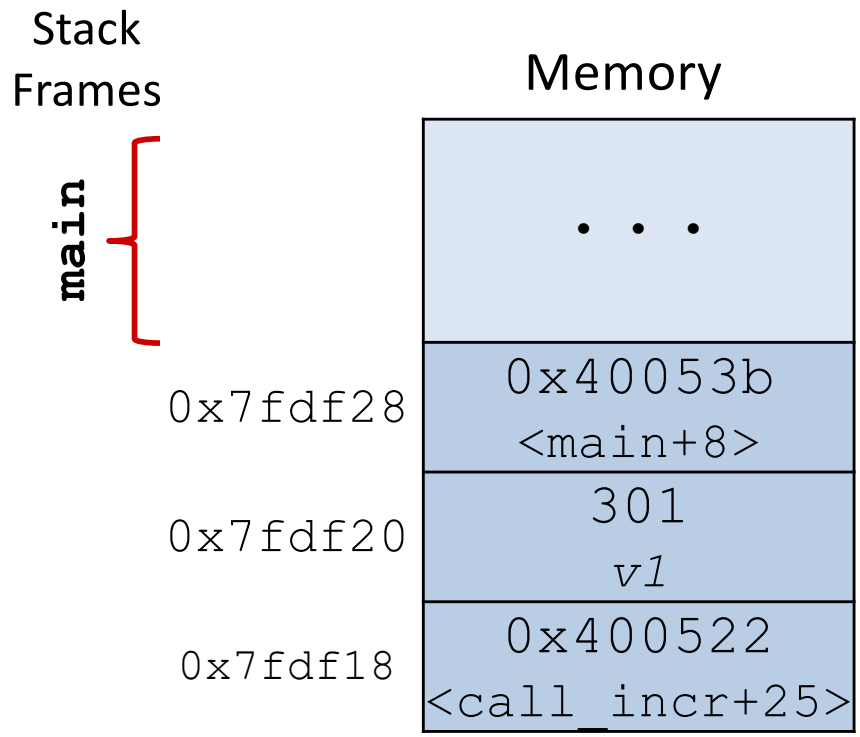
Procedure Call Example (step 8)

Return from `call_incr` to `main`

```
long call_incr() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

```
call_incr:
400509:  subq  $8, %rsp
40050d:  movq  $240, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  (%rsp), %rax
400526:  addq  $8, %rsp
40052a:  retq
```

```
increment:
4004cd:  movq  (%rdi), %rax
4004d0:  addq  %rax, %rsi
4004d3:  movq  %rsi, (%rdi)
4004d6:  retq
```



Register Saving Conventions

yoo calls who:

Caller ***Callee***

Will register contents still be there after a procedure call?

```
yoo:
    • • •
    movq $12345, %rbx
    call who
    addq %rbx, %rax
    • • •
    ret
```

?

```
who:
    • • •
    addq %rdi, %rbx
    • • •
    ret
```

Conventions:

Caller Save

Callee Save

x86-64 64-bit Register Conventions

%rax	Return value – Caller saved
%rbx	**Callee** saved
%rcx	Argument #4 – Caller saved
%rdx	Argument #3 – Caller saved
%rsi	Argument #2 – Caller saved
%rdi	Argument #1 – Caller saved
%rsp	Stack pointer
%rbp	**Callee** saved

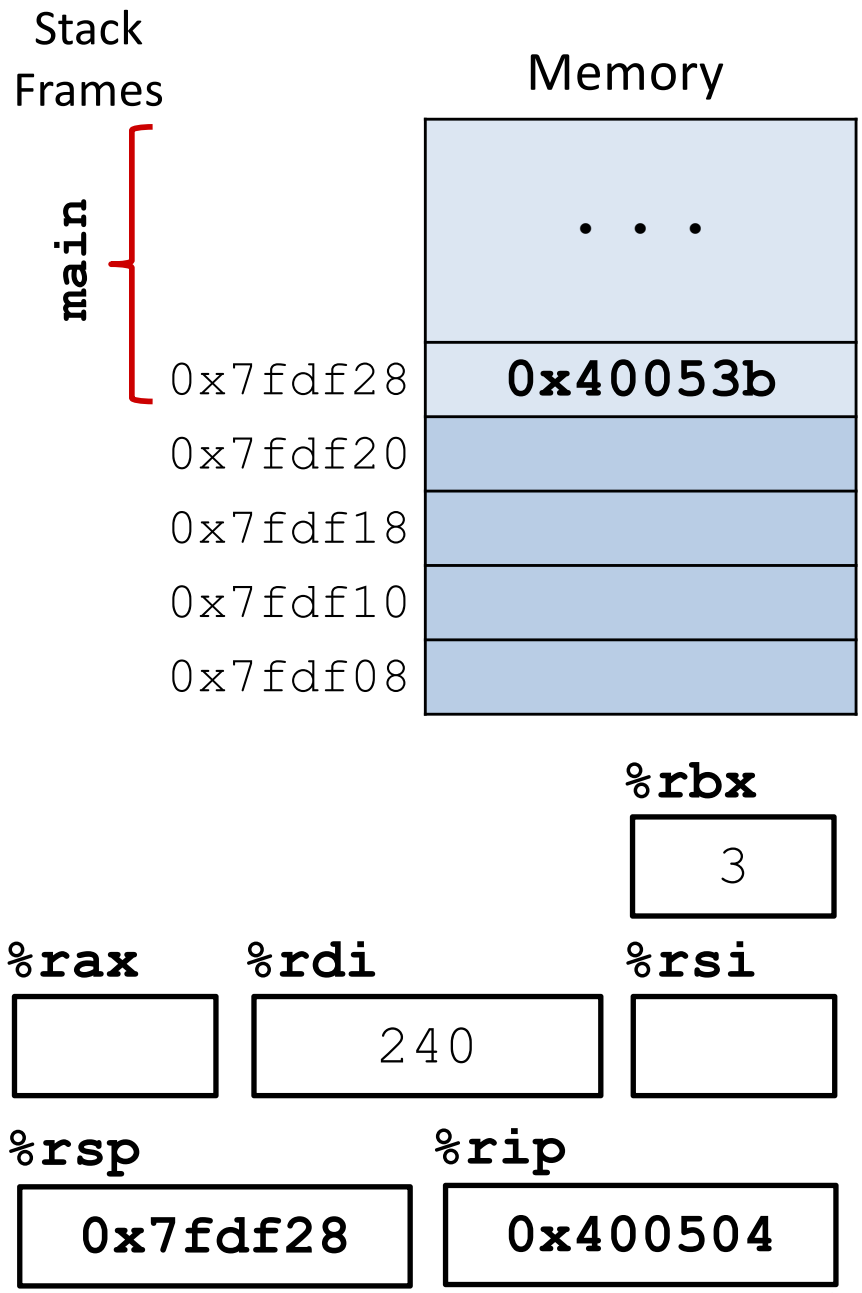
%r8	Argument #5 – Caller saved
%r9	Argument #6 – Caller saved
%r10	Caller saved
%r11	Caller Saved
%r12	**Callee** saved
%r13	**Callee** saved
%r14	**Callee** saved
%r15	**Callee** saved

Callee-Save Example (step 0)

`main` called `call_incr2(240)`

```
long call_incr2(long x) {
    long v1 = x;
    long v2 = increment(&v1, 61);
    return x + v2;
}
```

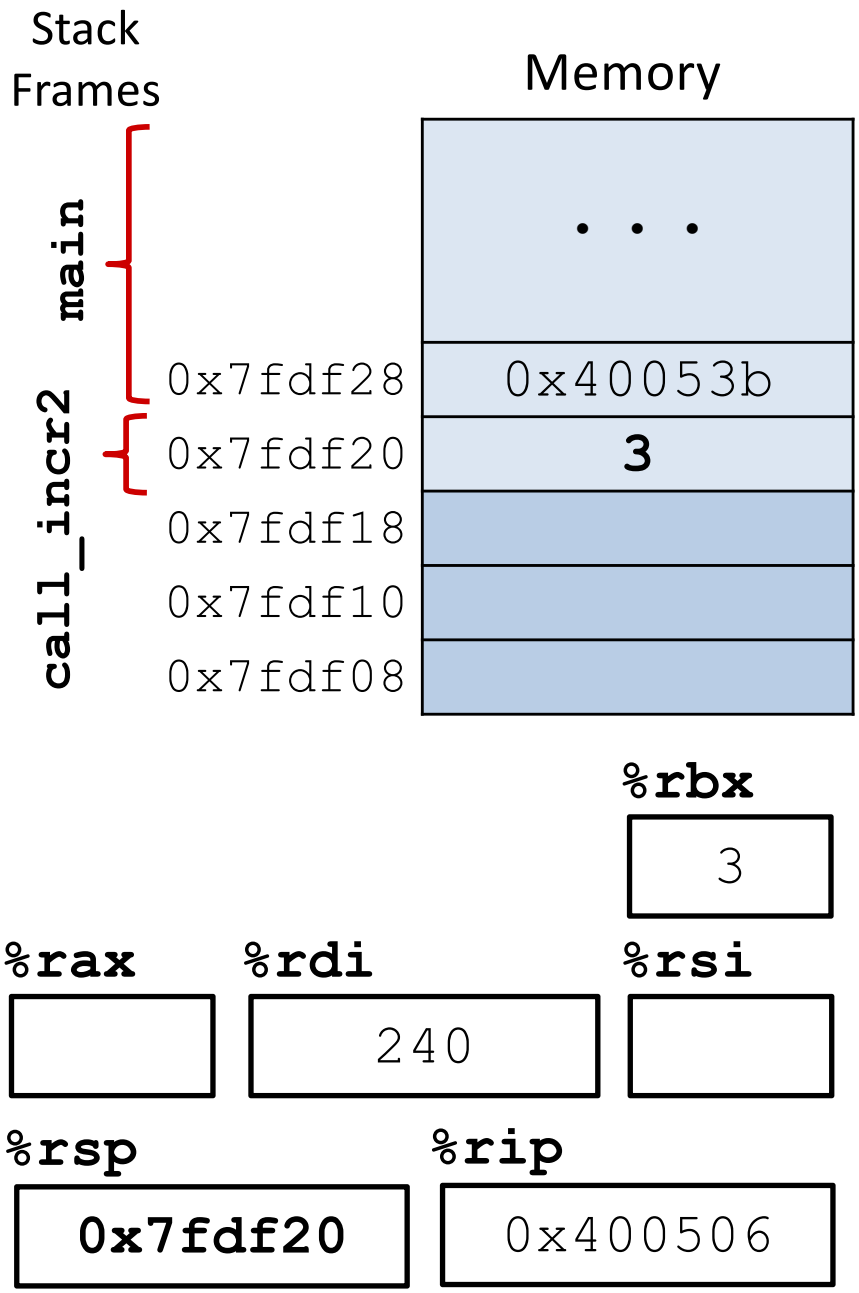
```
call_incr2:
400504:  pushq %rbx
400506:  movq  %rdi, %rbx
400509:  subq  $16, %rsp
40050d:  movq  %rdi, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  %rbx, %rax
400525:  addq  $16, %rsp
400529:  popq  %rbx
40052b:  retq
```



Callee-Save Example (step 1)

```
long call_incr2(long x) {
    long v1 = x;
    long v2 = increment(&v1, 61);
    return x + v2;
}
```

```
call_incr2:
400504:  pushq %rbx
400506:  movq  %rdi, %rbx
400509:  subq  $16, %rsp
40050d:  movq  %rdi, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  %rbx, %rax
400525:  addq  $16, %rsp
400529:  popq  %rbx
40052b:  retq
```

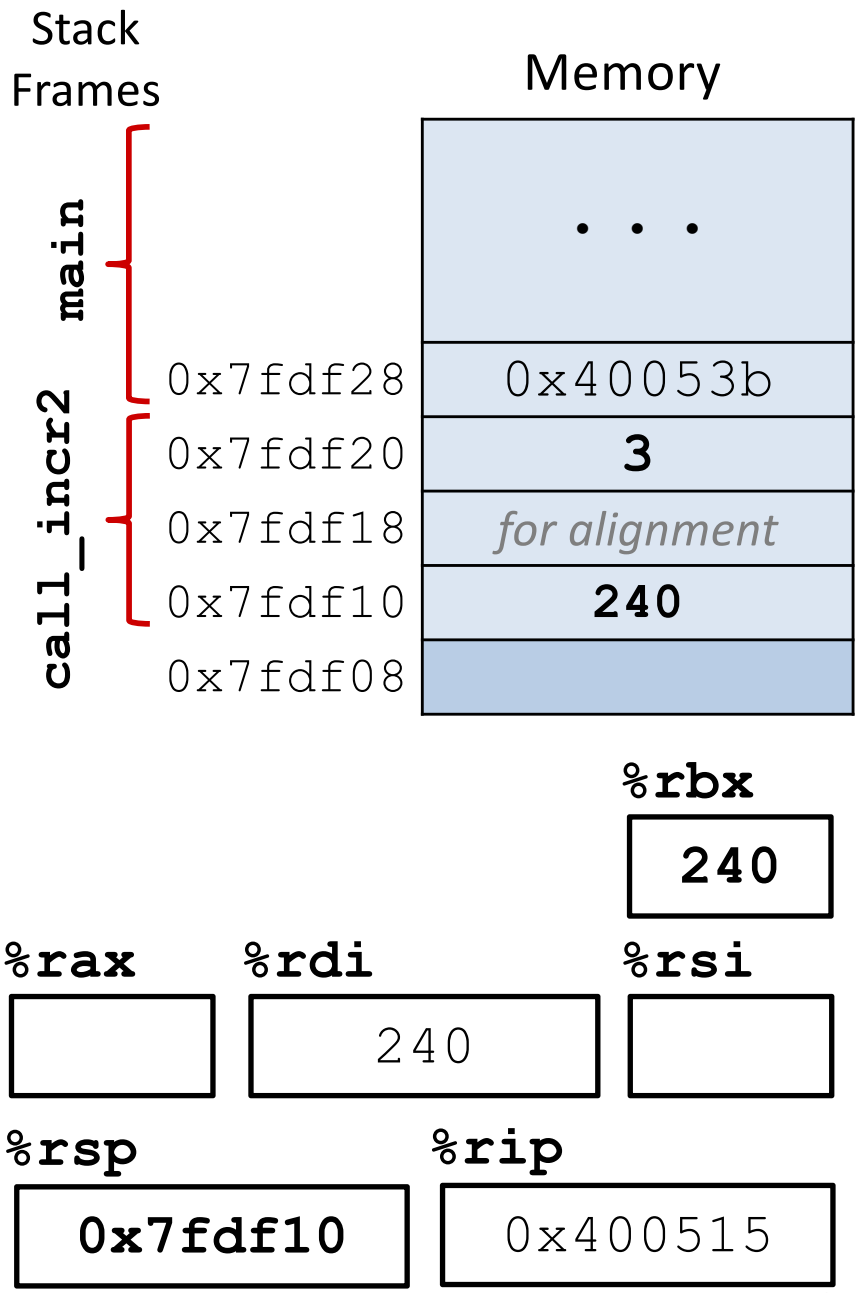


Callee-Save Example (step 2)

```
long call_incr2(long x) {
    long v1 = x;
    long v2 = increment(&v1, 61);
    return x + v2;
}
```

```
call_incr2:
400504:  pushq %rbx
400506:  movq  %rdi, %rbx
400509:  subq  $16, %rsp
40050d:  movq  %rdi, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  %rbx, %rax
400525:  addq  $16, %rsp
400529:  popq  %rbx
40052b:  retq
```

Stack frame setup
(extra slot for alignment)

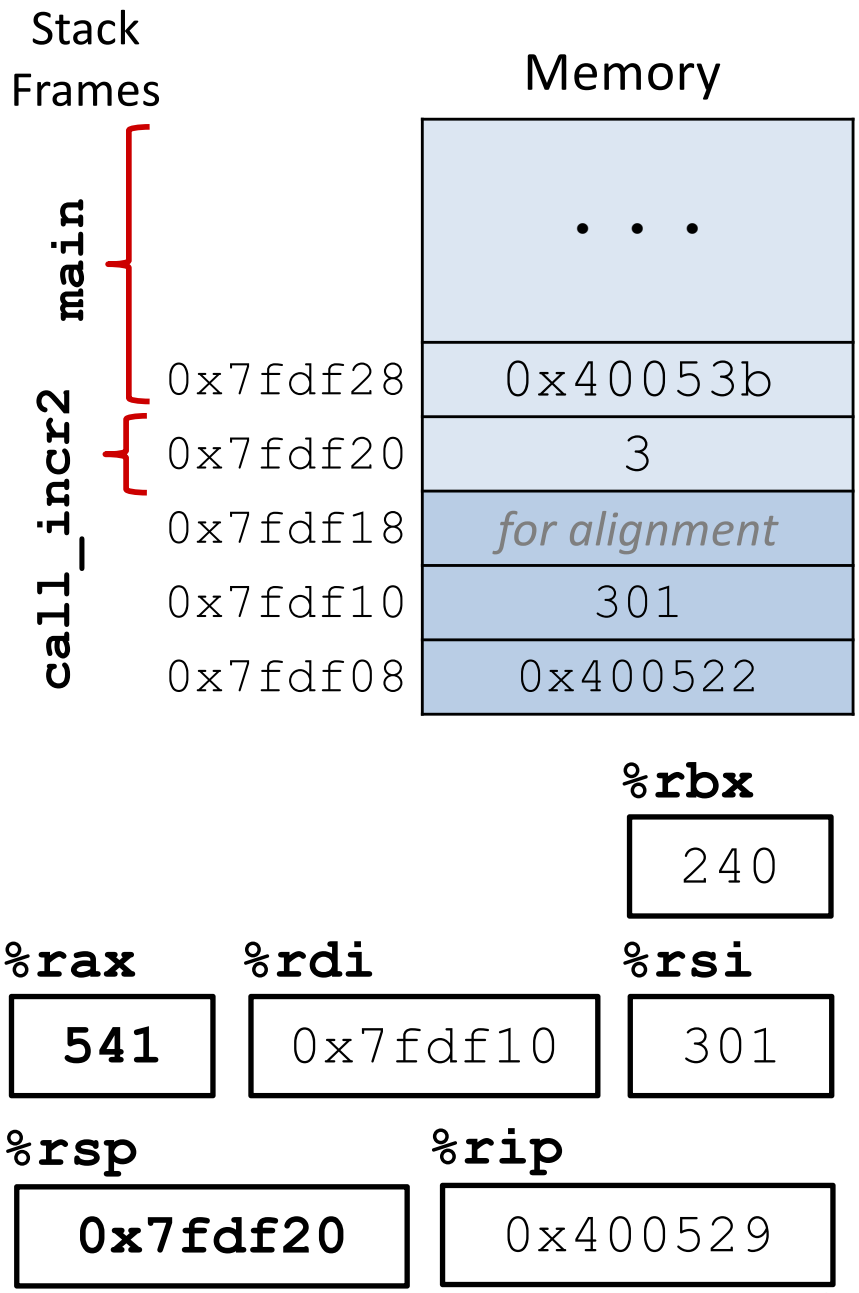


Callee-Save Example (step 3)

Prepare return value
and tear down stack frame

```
long call_incr2(long x) {
    long v1 = x;
    long v2 = increment(&v1, 61);
    return x + v2;
}
```

```
call_incr2:
400504:  pushq %rbx
400506:  movq  %rdi, %rbx
400509:  subq  $16, %rsp
40050d:  movq  %rdi, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  %rbx, %rax
400525:  addq  $16, %rsp
400529:  popq  %rbx
40052b:  retq
```

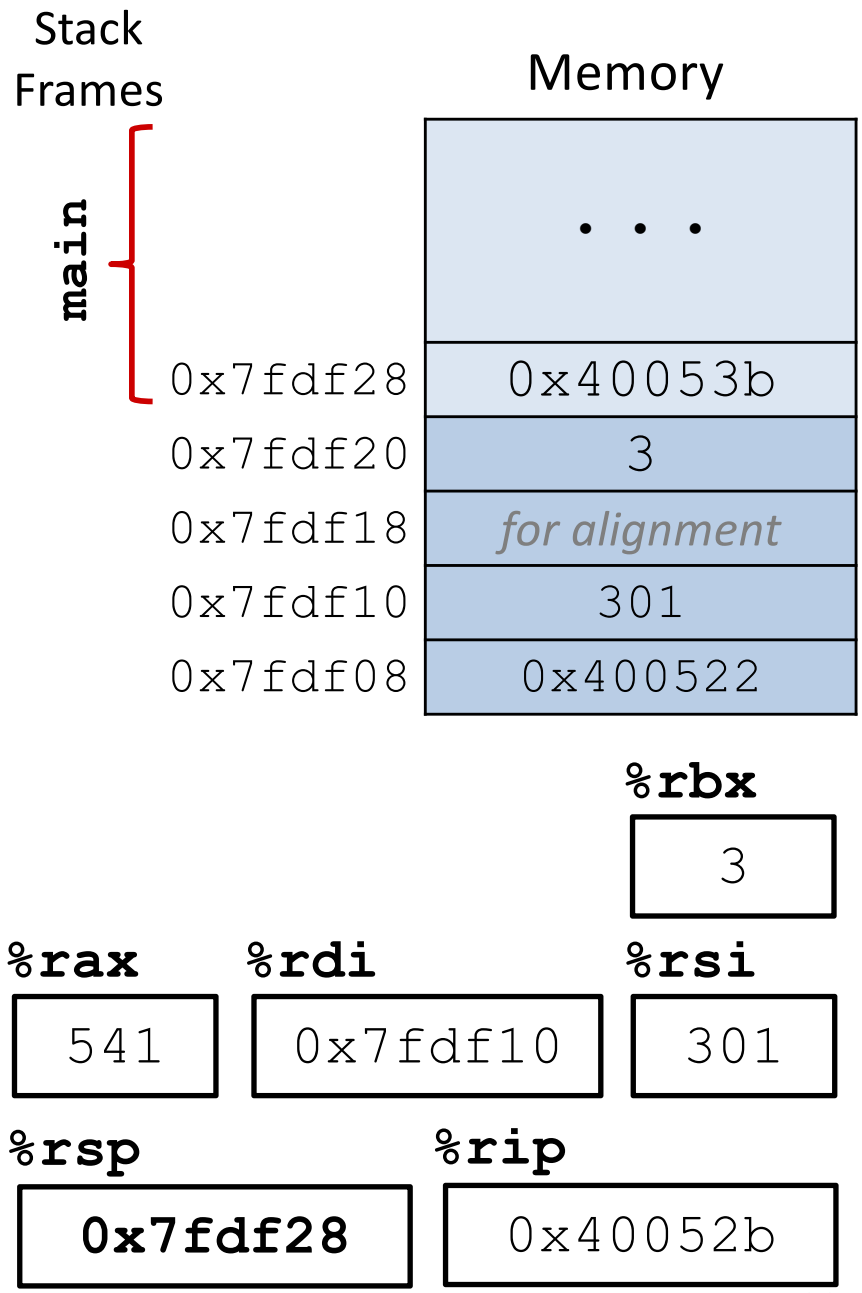


Callee-Save Example (step 4)

Register restore

```
long call_incr2(long x) {
    long v1 = x;
    long v2 = increment(&v1, 61);
    return x + v2;
}
```

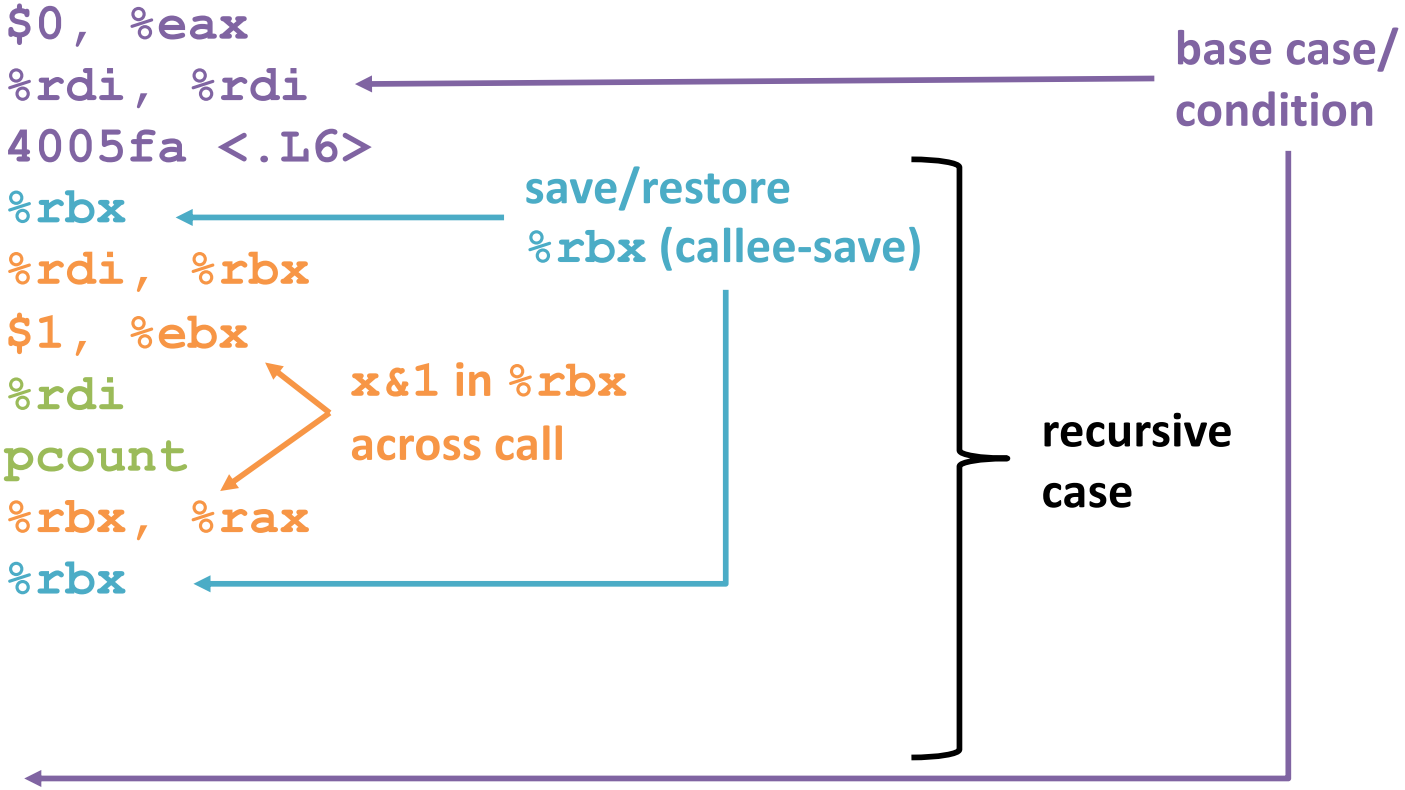
```
call_incr2:
400504:  pushq %rbx
400506:  movq  %rdi, %rbx
400509:  subq  $16, %rsp
40050d:  movq  %rdi, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  %rbx, %rax
400525:  addq  $16, %rsp
400529:  popq  %rbx
40052b:  retq
```



Recursion Example: code

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

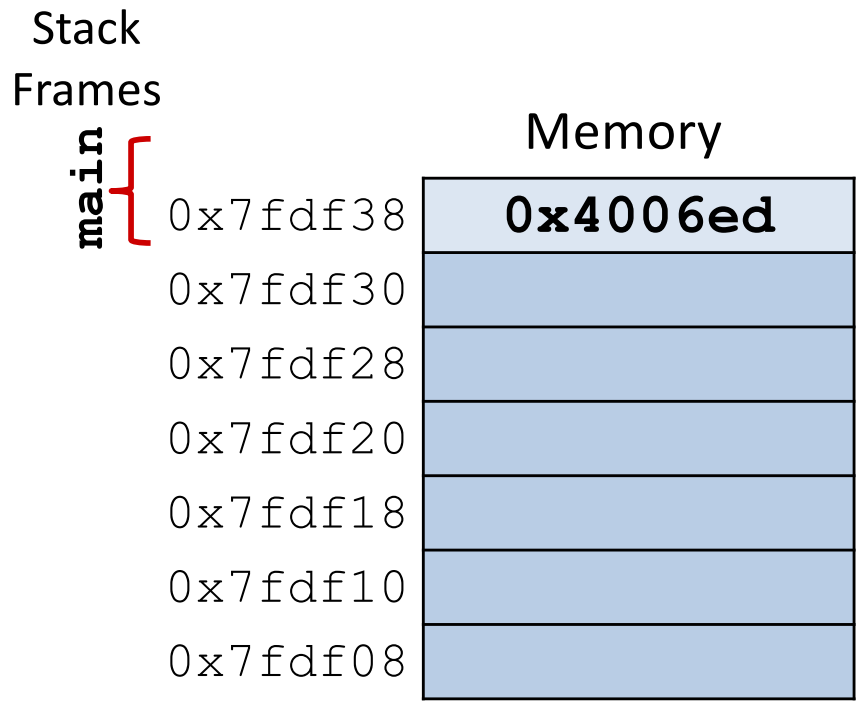
```
pcount:
4005dd:   movl   $0, %eax
4005e2:   testq  %rdi, %rdi
4005e5:   je     4005fa <.L6>
4005e7:   pushq  %rbx
4005e8:   movq   %rdi, %rbx
4005eb:   andl   $1, %ebx
4005ee:   shrq   %rdi
4005f1:   callq  pcount
4005f6:   addq   %rbx, %rax
4005f9:   popq   %rbx
.L6:
4005fa:   rep
4005fb:   retq
```

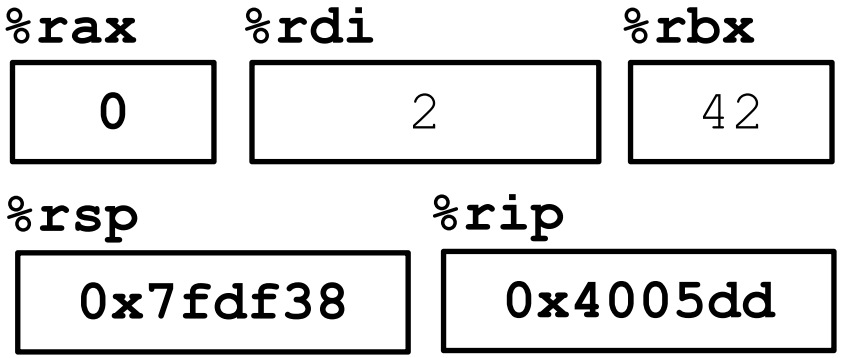


Recursion Example: `pcount(2)`

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl  $0, %eax
4005e2:  testq %rdi, %rdi
4005e5:  je    4005fa <.L6>
4005e7:  pushq %rbx
4005e8:  movq  %rdi, %rbx
4005eb:  andl  $1, %ebx
4005ee:  shrq  %rdi
4005f1:  callq pcount
4005f6:  addq  %rbx, %rax
4005f9:  popq  %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

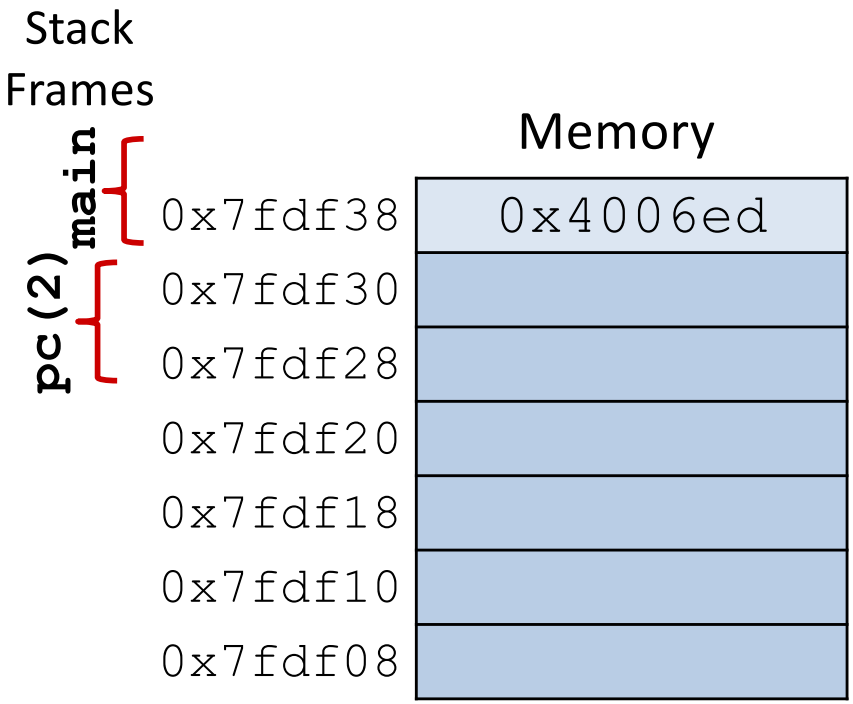


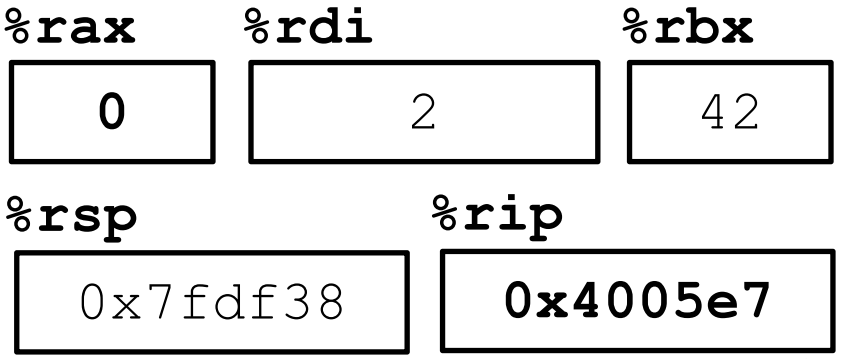


Recursion Example: `pcount(2)`

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

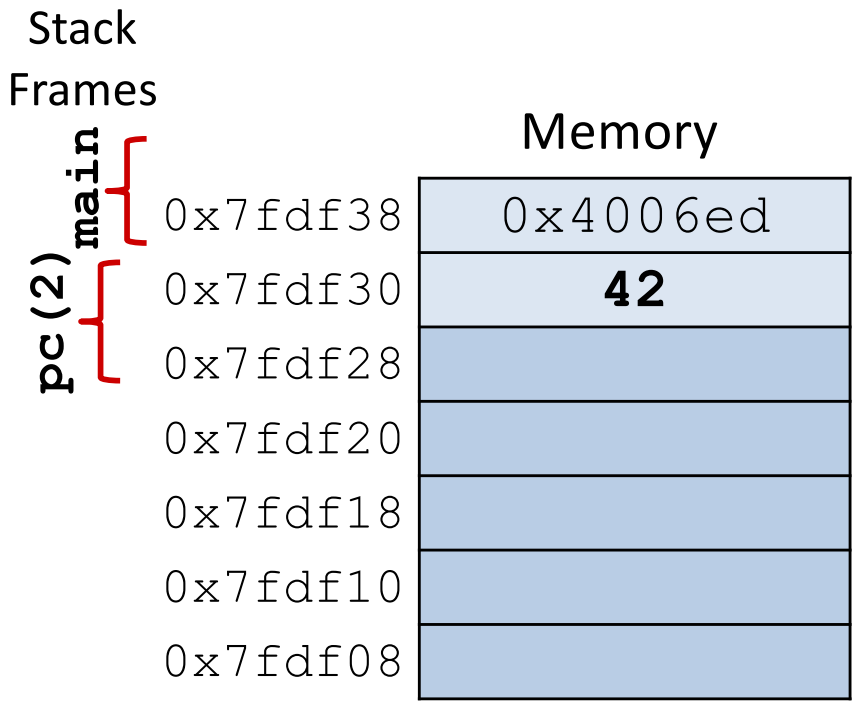


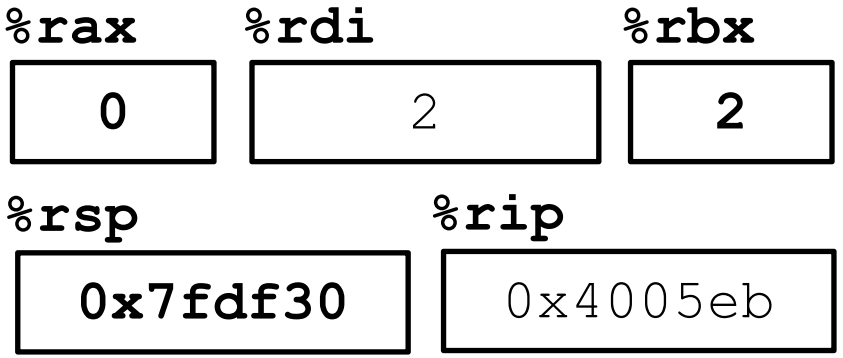


Recursion Example: `pcount(2)`

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

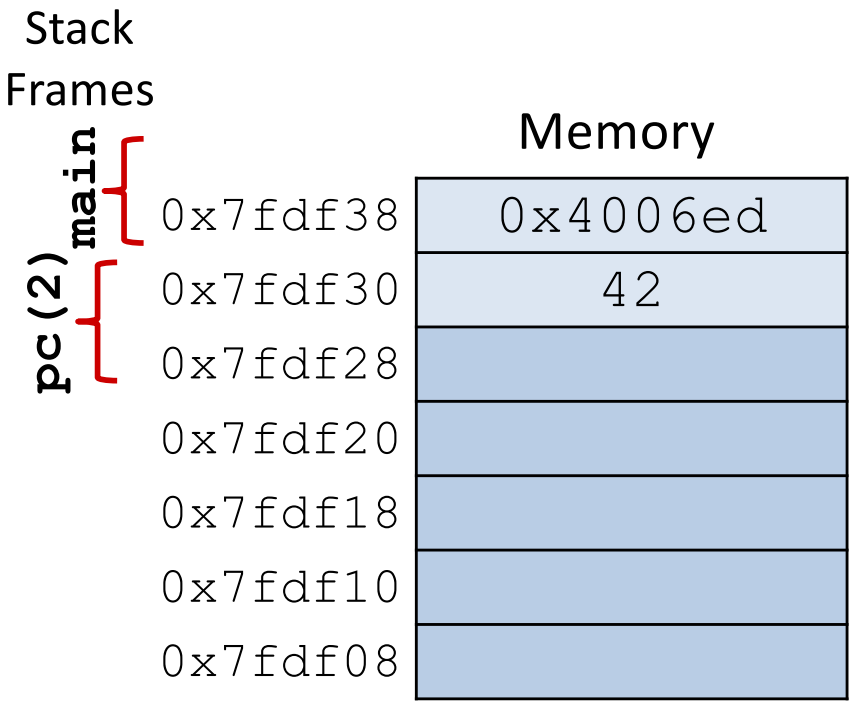


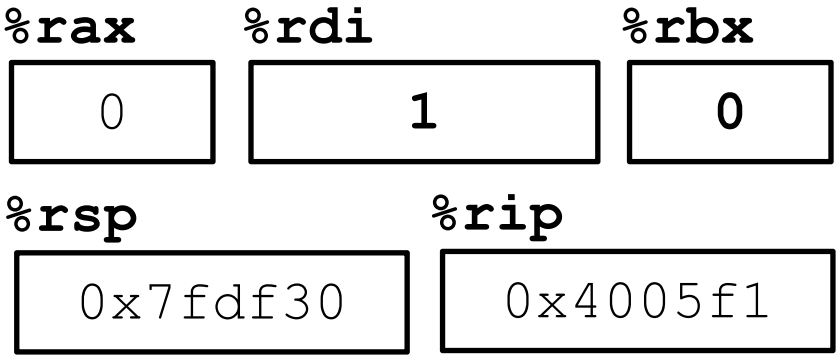


Recursion Example: `pcount(2)`

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl  $0, %eax
4005e2:  testq %rdi, %rdi
4005e5:  je    4005fa <.L6>
4005e7:  pushq %rbx
4005e8:  movq  %rdi, %rbx
4005eb:  andl  $1, %ebx
4005ee:  shrq  %rdi
4005f1:  callq pcount
4005f6:  addq  %rbx, %rax
4005f9:  popq  %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

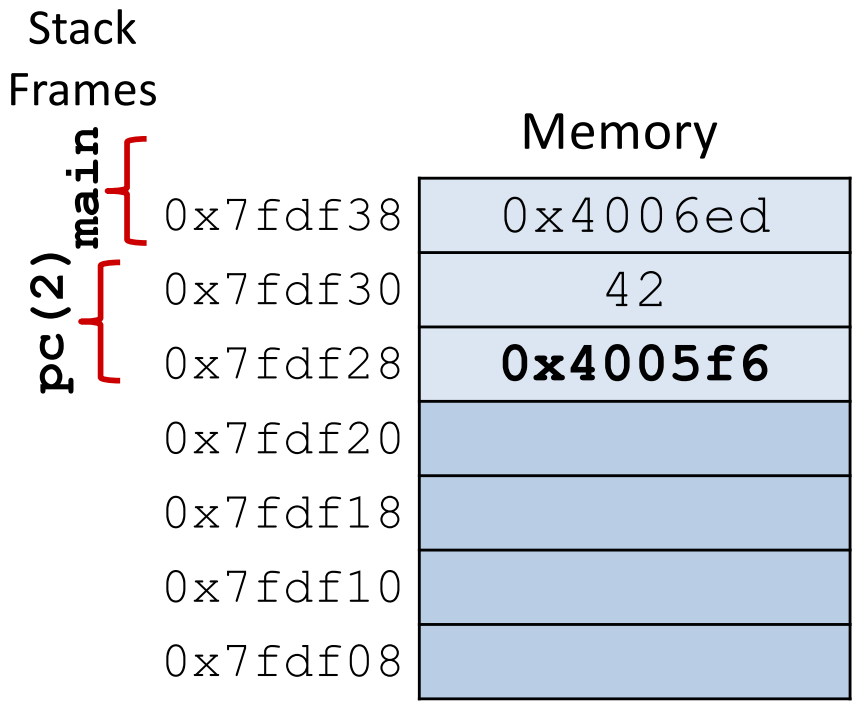


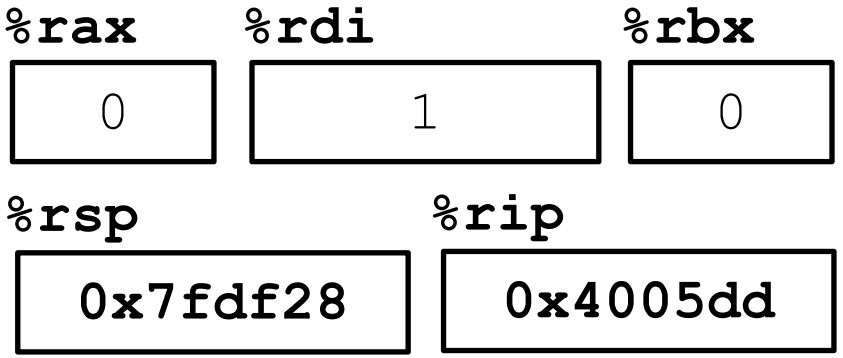


Recursion Example: `pcount(2)` → `pcount(1)`

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

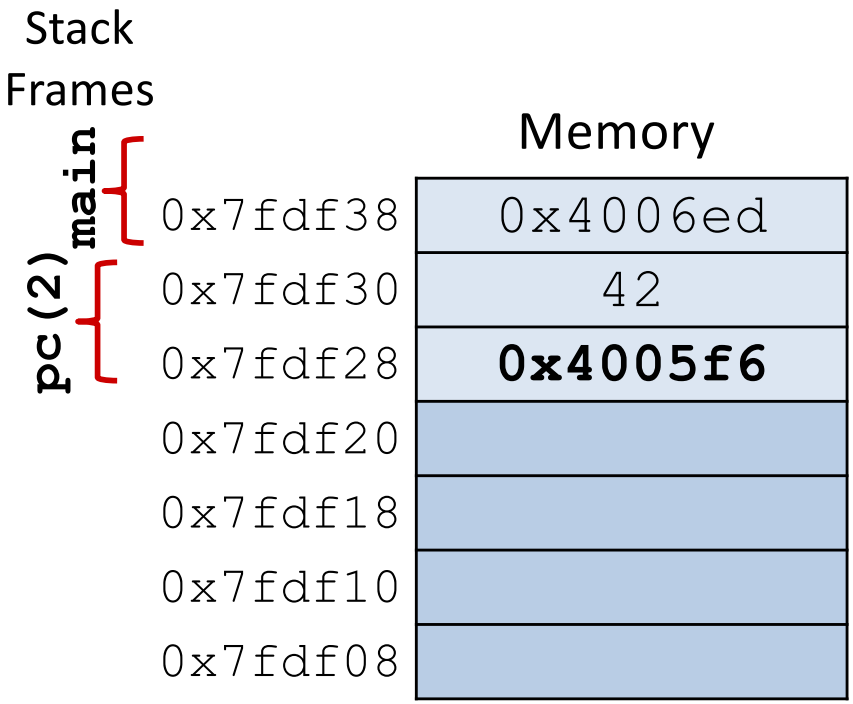


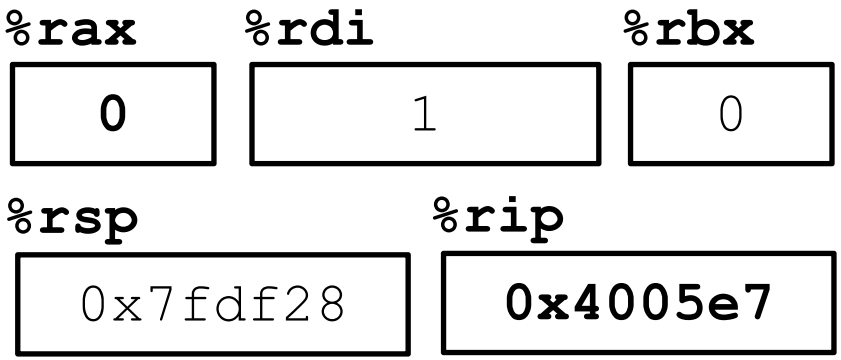


Recursion Example: `pcount(2)` → `pcount(1)`

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

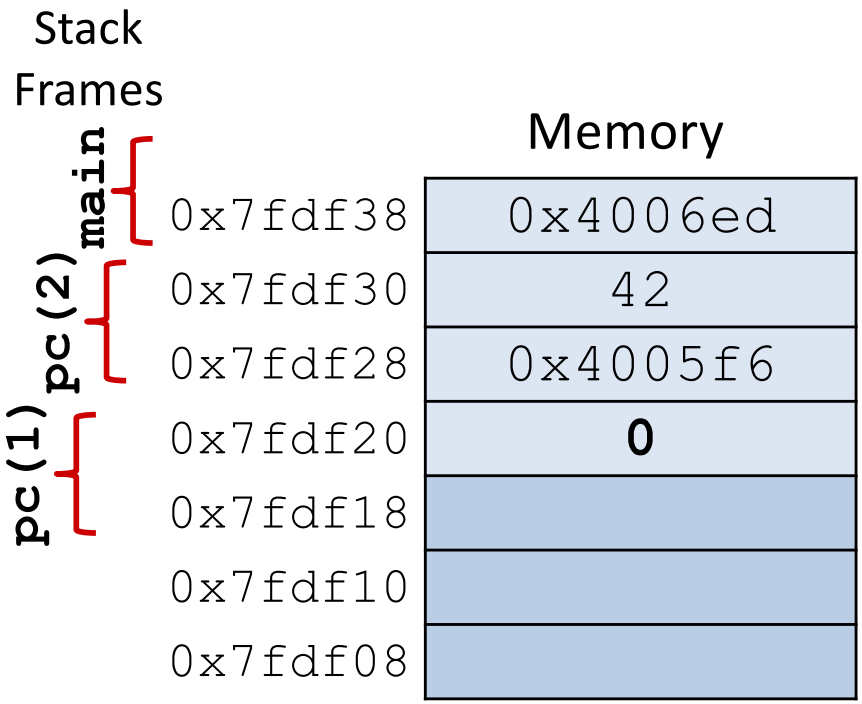


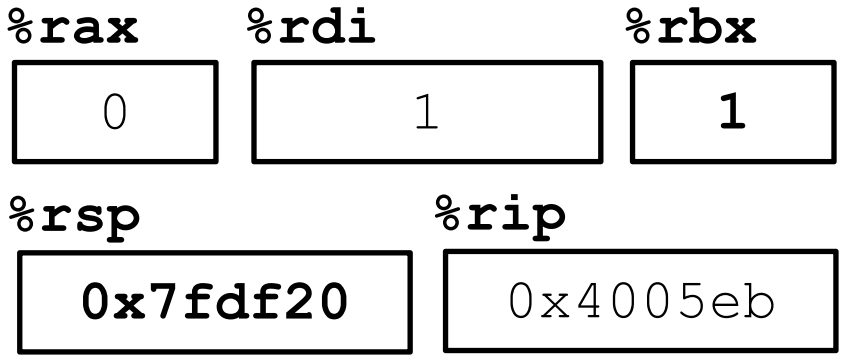


Recursion Example: `pcount(2)` → `pcount(1)`

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

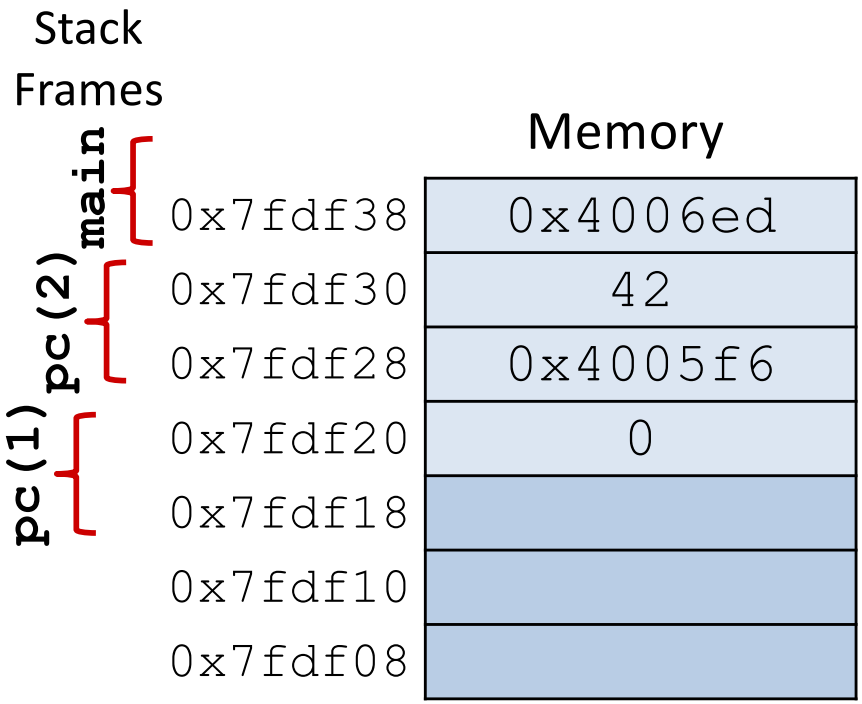


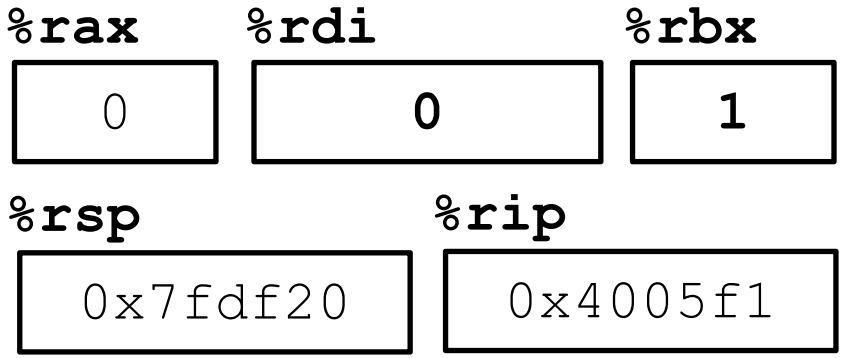


Recursion Example: `pcount(2)` → `pcount(1)`

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

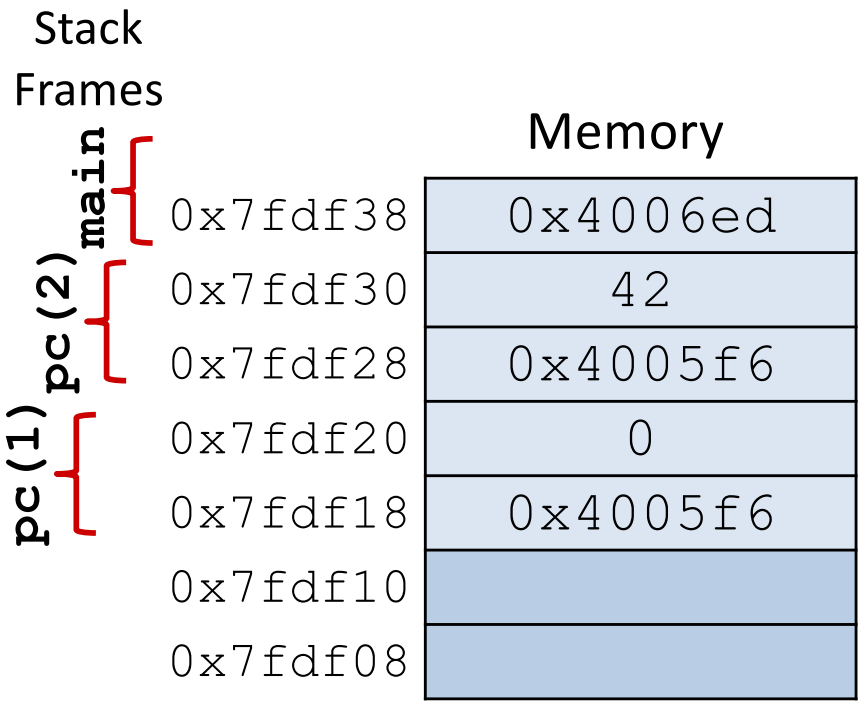


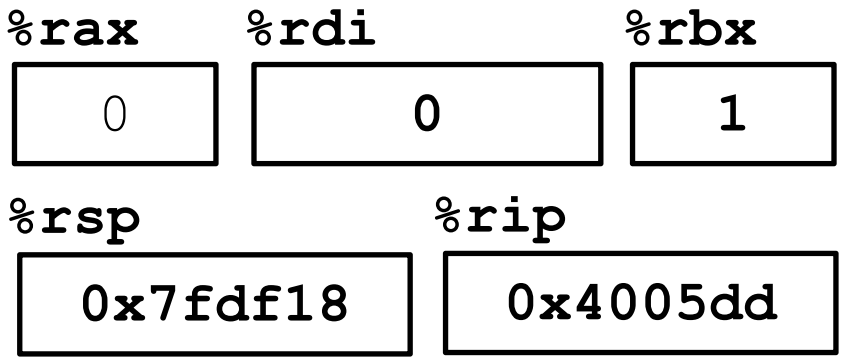


Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

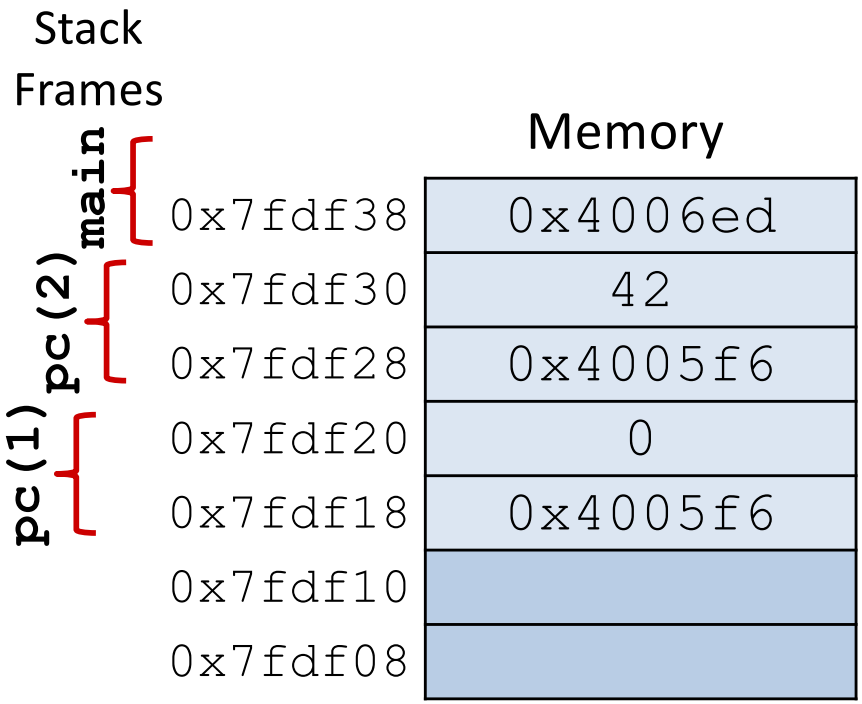


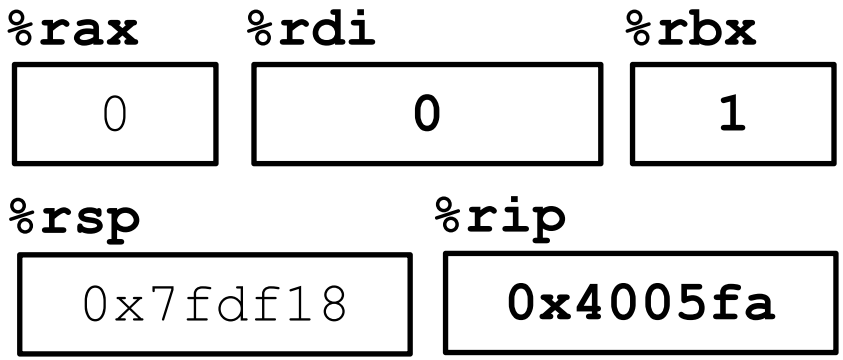


Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

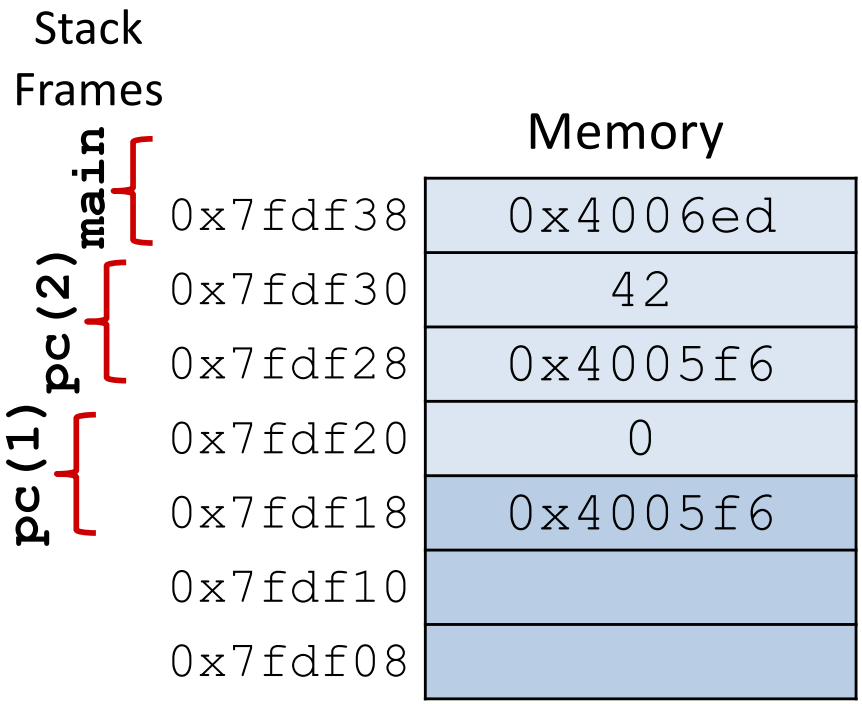


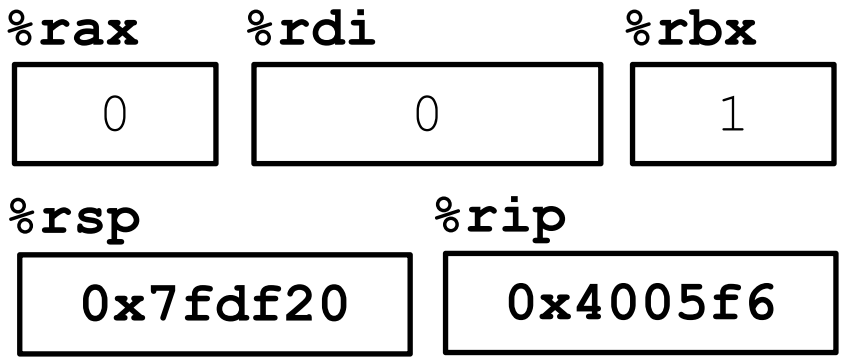


Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

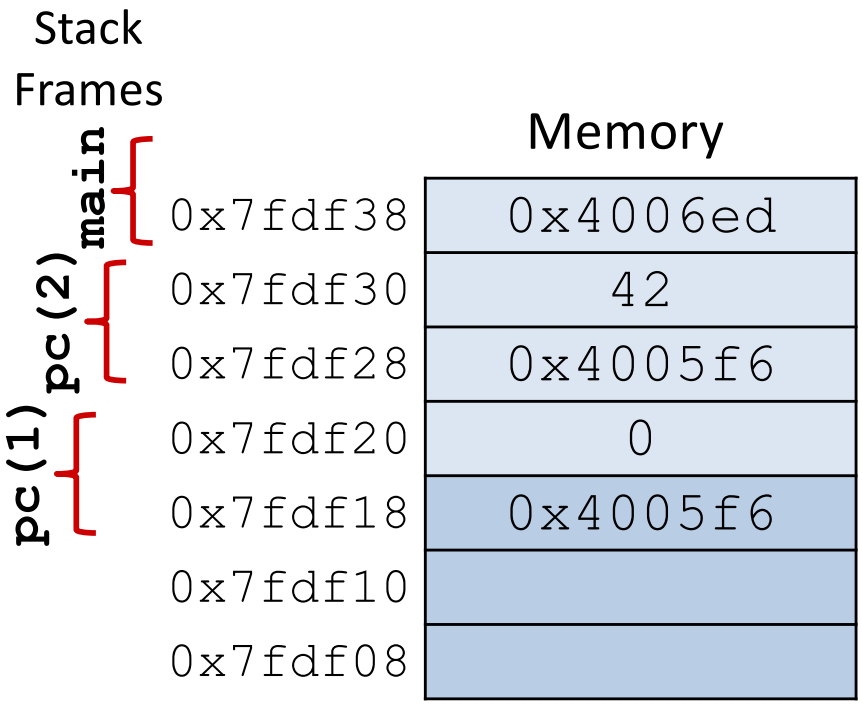


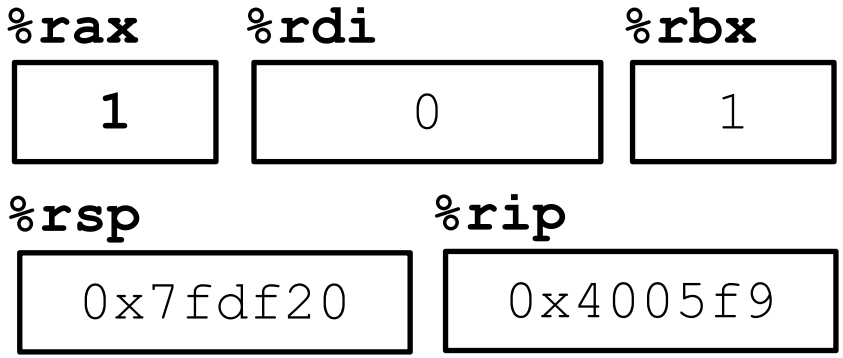


Recursion Example: pcount(2) → **pcount(1)** → pcount(0)

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

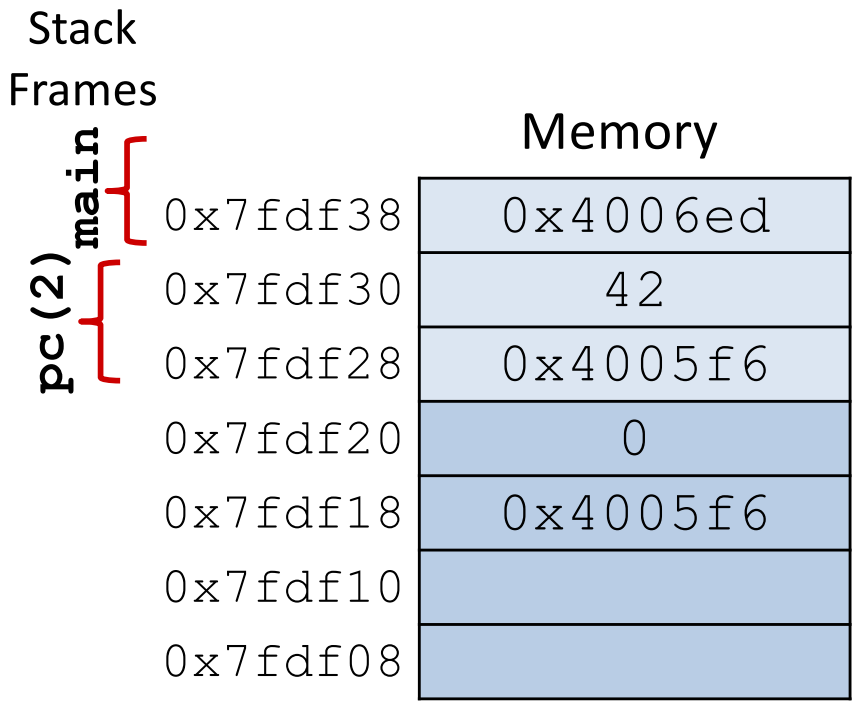


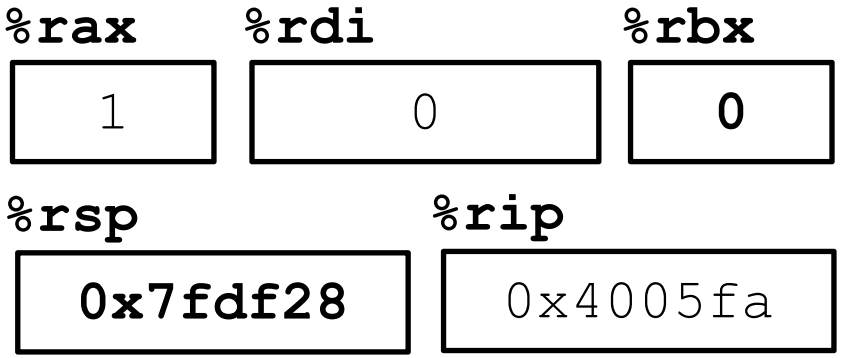


Recursion Example: pcount(2) → **pcount(1)** → pcount(0)

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

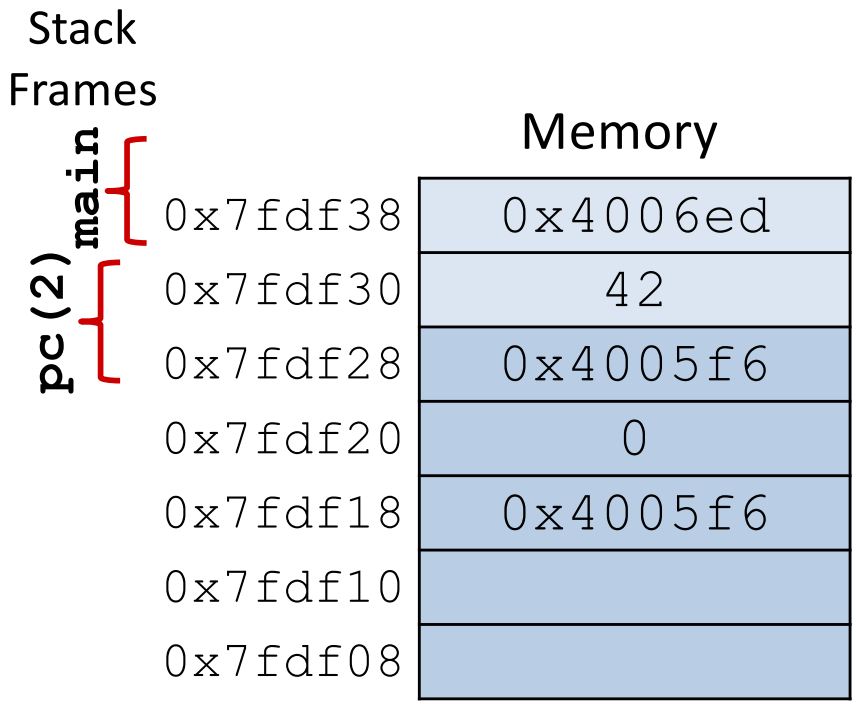


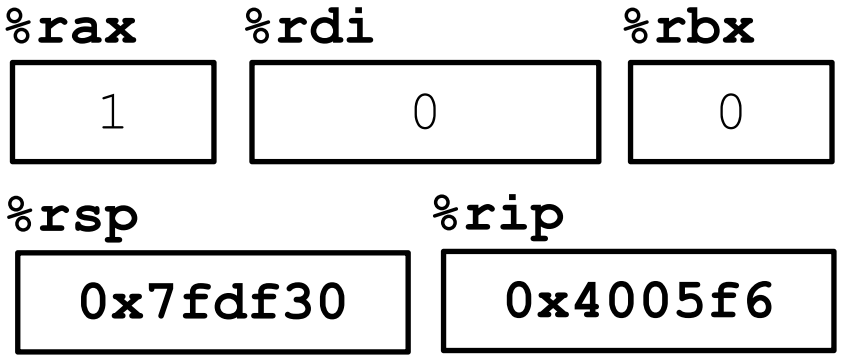


Recursion Example: pcount(2) → **pcount(1)** → pcount(0)

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

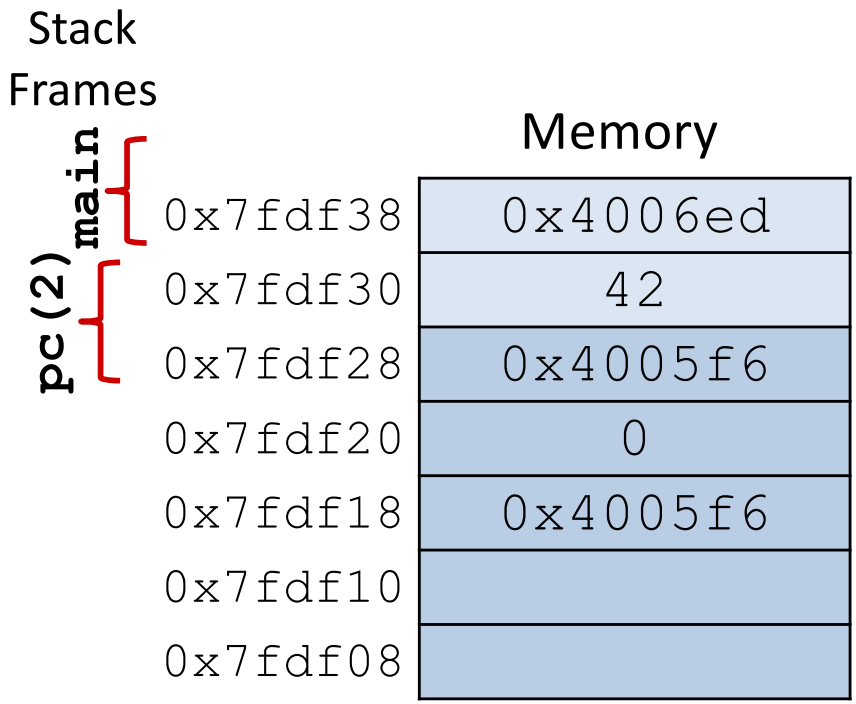


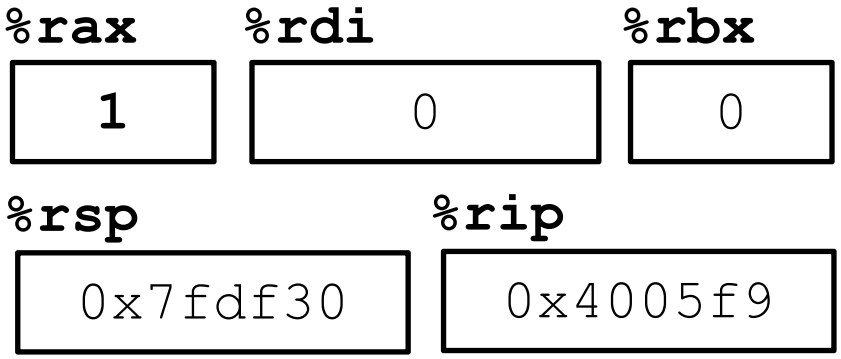


Recursion Example: `pcount(2)` → `pcount(1)` → `pcount(0)`

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```





Recursion Example: `pcount(2)` → `pcount(1)` → `pcount(0)`

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

Stack Frames

main

Memory

0x7fdf38	0x4006ed
0x7fdf30	42
0x7fdf28	0x4005f6
0x7fdf20	0
0x7fdf18	0x4005f6
0x7fdf10	
0x7fdf08	

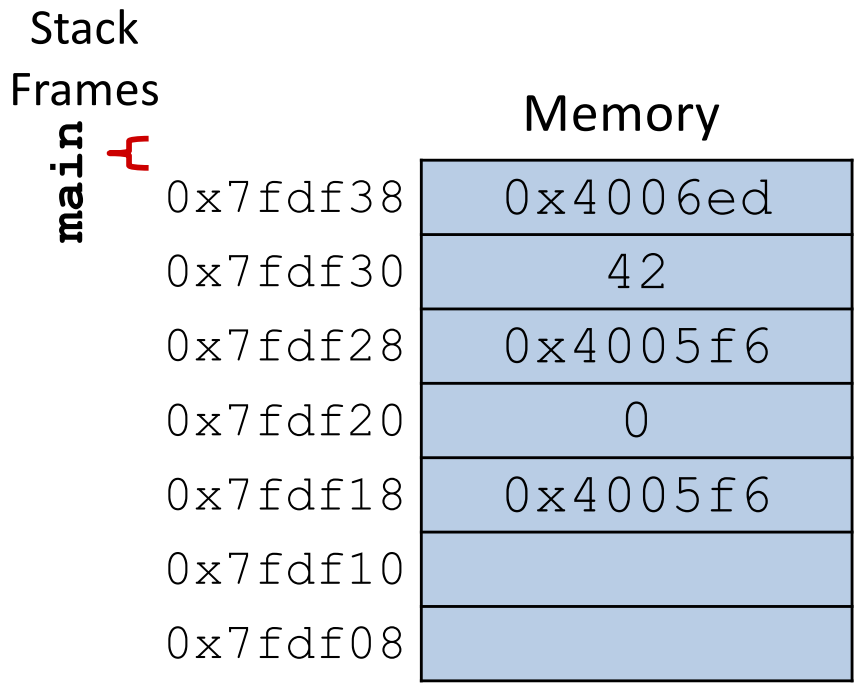
%rax	**%rdi**	**%rbx**
1	0	**42**

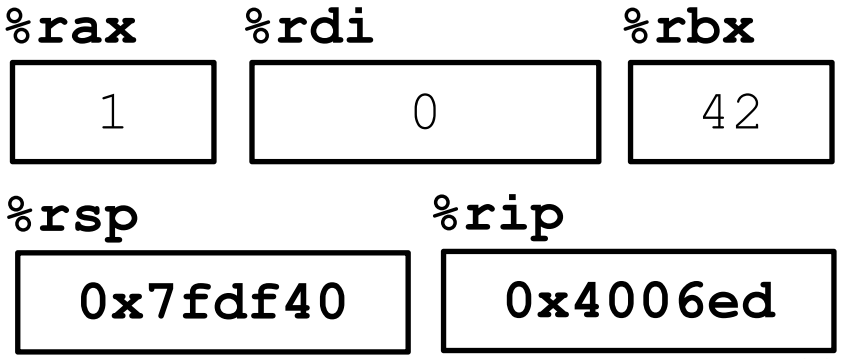
%rsp	**%rip**
0x7fdf38	0x4005f9

Recursion Example: `pcount(2)` → `pcount(1)` → `pcount(0)`

```
long pcount(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1) + pcount(x >> 1);
  }
}
```

```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq  %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq  %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq  pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

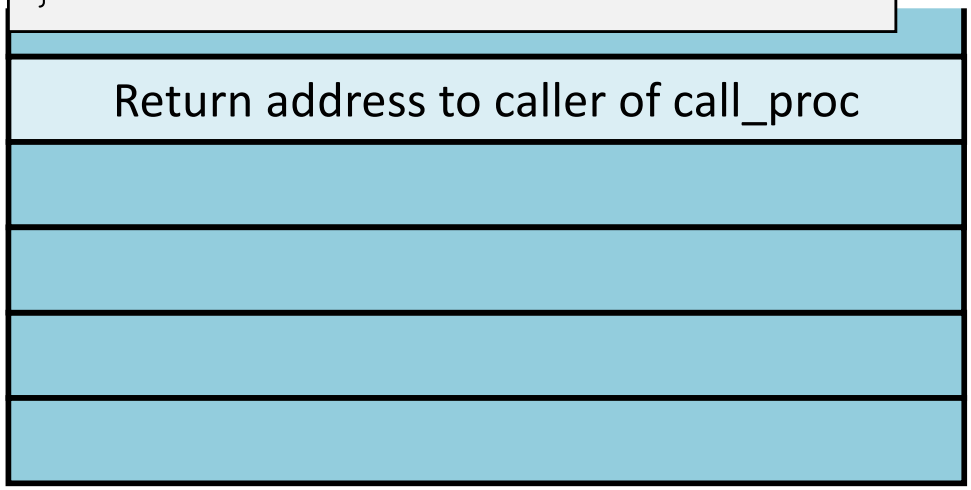




x86-64 stack storage example (1)

```
long int call_proc()
{
  long  x1 = 1;
  int   x2 = 2;
  short x3 = 3;
  char  x4 = 4;
  proc(x1, &x1, x2, &x2,
       x3, &x3, x4, &x4);
  return (x1+x2)*(x3-x4);
}
```

```
call_proc:
  subq  $32,%rsp
  movq  $1,16(%rsp) # x1
  movl  $2,24(%rsp) # x2
  movw  $3,28(%rsp) # x3
  movb  $4,31(%rsp) # x4
  • • •
```



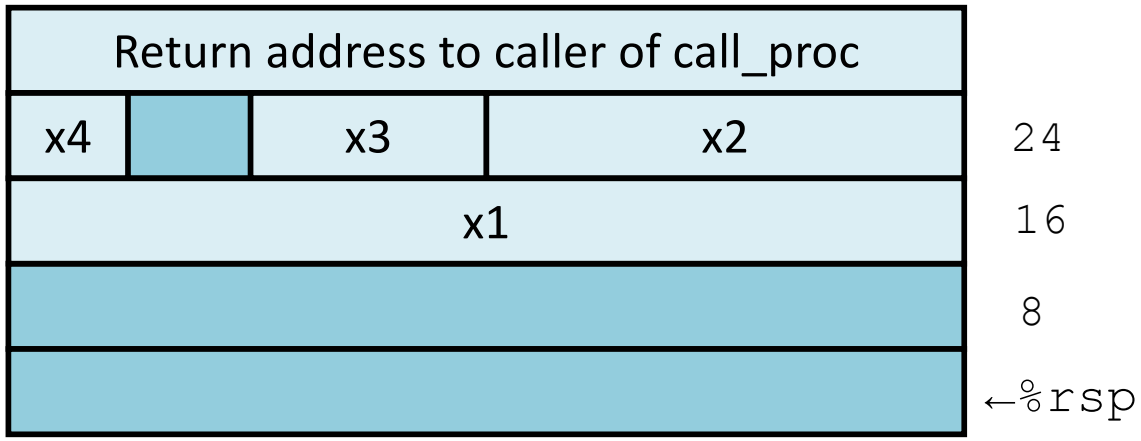
←%rsp

x86-64 stack storage example

(2) Allocate local vars

```
long int call_proc()
{
  long  x1 = 1;
  int   x2 = 2;
  short x3 = 3;
  char  x4 = 4;
  proc(x1, &x1, x2, &x2,
       x3, &x3, x4, &x4);
  return (x1+x2)*(x3-x4);
}
```

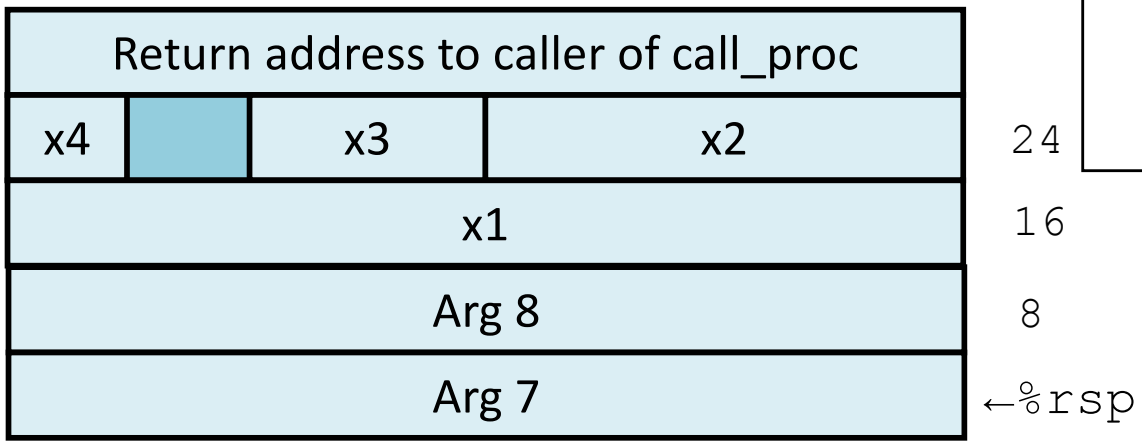
```
call_proc:
  subq  $32,%rsp
  movq  $1,16(%rsp)  # x1
  movl  $2,24(%rsp)  # x2
  movw  $3,28(%rsp)  # x3
  movb  $4,31(%rsp)  # x4
  • • •
```



x86-64 stack storage example

(3) setup args to `proc`

```
long int call_proc()
{
  long  x1 = 1;
  int   x2 = 2;
  short x3 = 3;
  char  x4 = 4;
  proc(x1, &x1, x2, &x2,
       x3, &x3, x4, &x4);
  return (x1+x2)*(x3-x4);
}
```




```
call_proc:
  . . .
  leaq  24(%rsp),%rcx # &x2
  leaq  16(%rsp),%rsi # &x1
  leaq  31(%rsp),%rax # &x4
  movq  %rax,8(%rsp)  # ...
  movl  $4,(%rsp)     # 4
  leaq  28(%rsp),%r9  # &x3
  movl  $3,%r8d       # 3
  movl  $2,%edx       # 2
  movq  $1,%rdi       # 1
  call  proc
  . . .
```

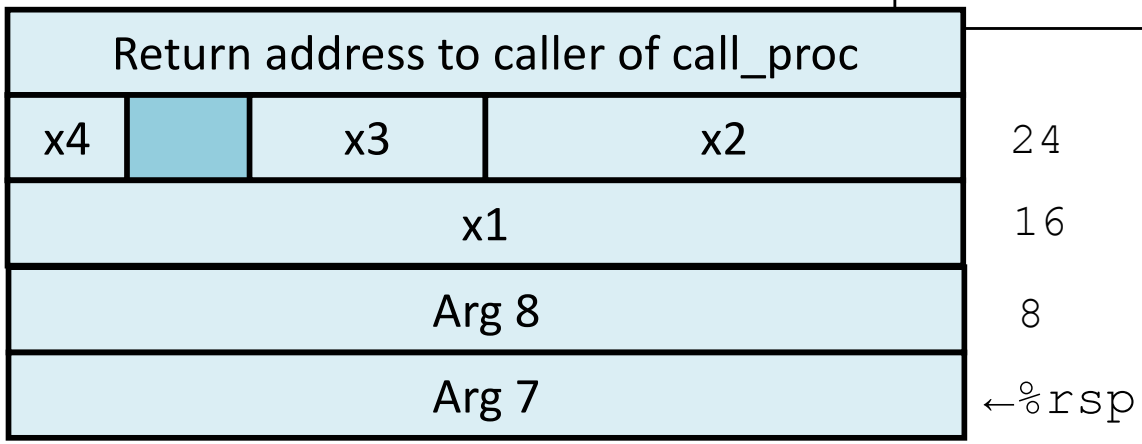
Arguments passed in (in order): rdi, rsi, rdx, rcx, r8, r9

x86-64 stack storage example

(4) after call to `proc`

```
long int call_proc()
{
  long  x1 = 1;
  int   x2 = 2;
  short x3 = 3;
  char  x4 = 4;
  proc(x1, &x1, x2, &x2,
       x3, &x3, x4, &x4);
  return (x1+x2)*(x3-x4);
}
```

```
call_proc:
  . . .
  movswl 28(%rsp),%eax # x3
  movsbl 31(%rsp),%edx # x4
  subl   %edx,%eax  # x3-x4
  cltq   # sign-extend %eax->rax
  movslq 24(%rsp),%rdx # x2
  addq   16(%rsp),%rdx # x1+x2
  imulq  %rdx,%rax     # *
  addq   $32,%rsp
  ret
```

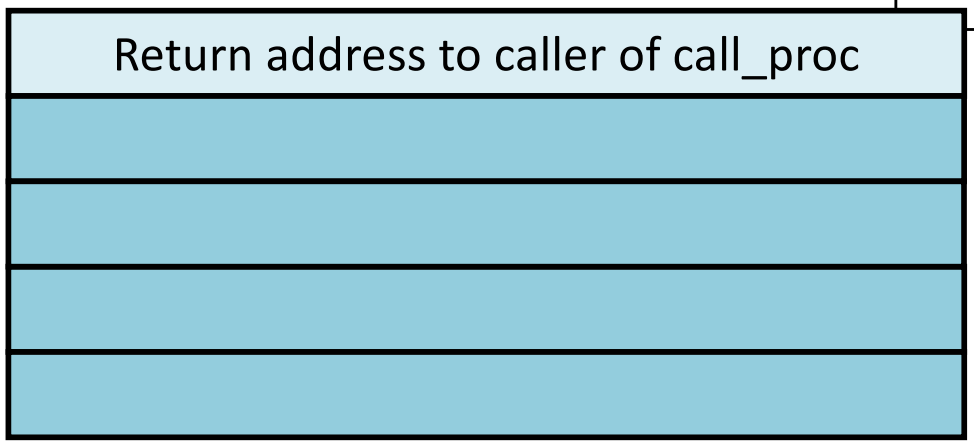


x86-64 stack storage example

(5) deallocate local vars

```
long int call_proc()
{
  long  x1 = 1;
  int   x2 = 2;
  short x3 = 3;
  char  x4 = 4;
  proc(x1, &x1, x2, &x2,
       x3, &x3, x4, &x4);
  return (x1+x2)*(x3-x4);
}
```

```
call_proc:
  • • •
  movswl 28(%rsp),%eax
  movsbl 31(%rsp),%edx
  subl   %edx,%eax
  cltq
  movslq 24(%rsp),%rdx
  addq   16(%rsp),%rdx
  imulq  %rdx,%rax
  addq   $32,%rsp
  ret
```



←%rsp

Procedure Summary

call, ret, push, pop

Stack discipline fits procedure call / return.

- If P calls Q: Q (and calls by Q) returns before P

Conventions support arbitrary function calls.

- Register-save conventions.
- Stack frame saves extra args or local variables.
- Result returned in **`%rax`**

`%rax` Return value – Caller saved	`%r8` Argument #5 – Caller saved
`%rbx` Callee saved	`%r9` Argument #6 – Caller saved
`%rcx` Argument #4 – Caller saved	`%r10` Caller saved
`%rdx` Argument #3 – Caller saved	`%r11` Caller Saved
`%rsi` Argument #2 – Caller saved	`%r12` Callee saved
`%rdi` Argument #1 – Caller saved	`%r13` Callee saved
`%rsp` Stack pointer	`%r14` Callee saved
`%rbp` Callee saved	`%r15` Callee saved