



x86: Procedures and the Call Stack

The call stack discipline
x86 procedure call and return instructions
x86 calling conventions
x86 register-saving conventions

x86: Procedures and the Call Stack

Outline

1. Motivation
 - a. What we have seen so far
 - b. Why we can't implement procedure calls with jumps alone
2. High-level call stack example
3. Procedure control flow instructions: call and ret
4. Procedure call example (in depth!)
5. *(Finish with video)* Caller vs/callee example
6. *(Covered in lab, video)* Recursion example

Why procedures?

Why functions? Why methods?

```
int contains_char(char* haystack, char needle) {
    while (*haystack != '\0') {
        if (*haystack == needle) return 1;
        haystack++;
    }
    return 0;
}
```

Answer: procedural abstraction

Implementing procedures

Have we already seen how this is done?

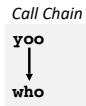
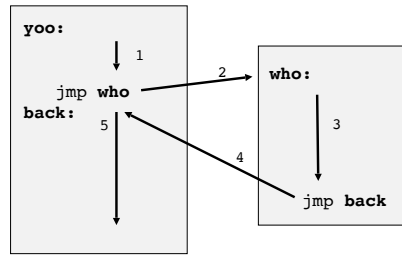
1. How does a caller pass arguments to a procedure? ✓
2. How does a caller receive a return value from a procedure? ✓
3. How does a procedure know where to return (what code to execute next when done)? ??
4. Where does a procedure store local variables? ✓?
5. How do procedures share limited registers and memory? ??

Procedure call/return: Jump?

```
yoo(...) {
  . . .
  who();
  . . .
}
```

```
who(...) {
  . . .
  . . .
  . . .
}
```

```
ru(...) {
  . . .
}
```



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

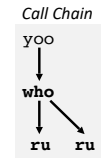
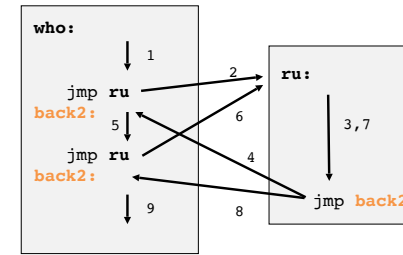
5

Procedure call/return: Jump? Broken!

```
yoo(...) {
  . . .
  who();
  . . .
}
```

```
who(...) {
  . . .
  ru();
  . . .
}
```

```
ru(...) {
  . . .
}
```



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

Broken: needs to track context.

6

Implementing procedures

requires **separate storage *per call!***
(not just per procedure)

Have we already seen
how this is done?

- How does a caller pass arguments to a procedure? ✓
 - How does a caller receive a return value from a procedure? ✓
 - How does a procedure know where to return (what code to execute next when done)? ??
 - Where does a procedure store local variables? ✓?
- How do procedures share limited registers and memory? ??

7

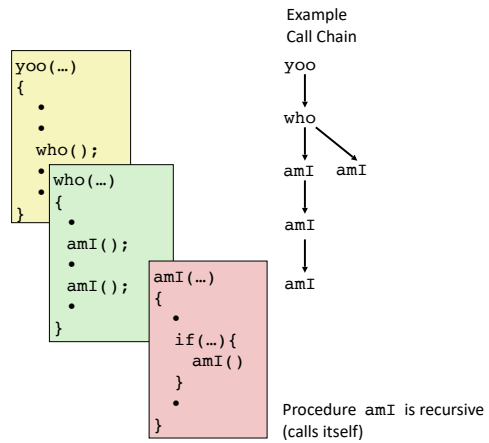
Memory Layout

reminder

Addr	Perm	Contents	Managed by	Initialized
2^N-1	RW	Procedure context	Compiler	Run-time
	RW	Dynamic data structures	Programmer, malloc/free, new/GC	Run-time
	RW	Global variables/static data structures	Compiler/Assembler/Linker	Startup
	R	String literals	Compiler/Assembler/Linker	Startup
	X	Instructions	Compiler/Assembler/Linker	Startup
0				

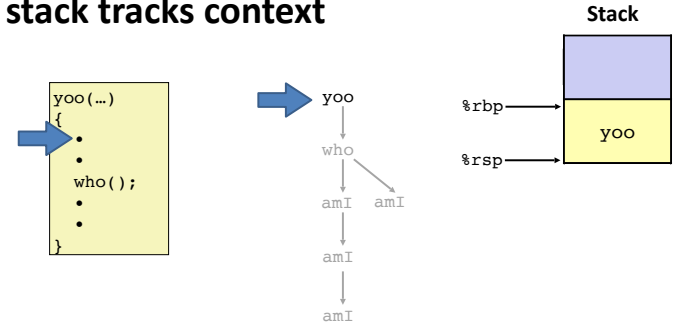
8

Call stack tracks context



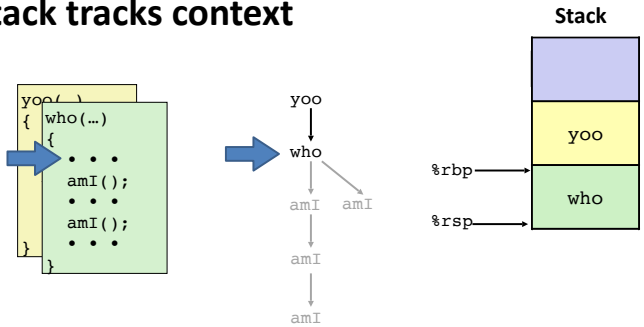
9

Call stack tracks context



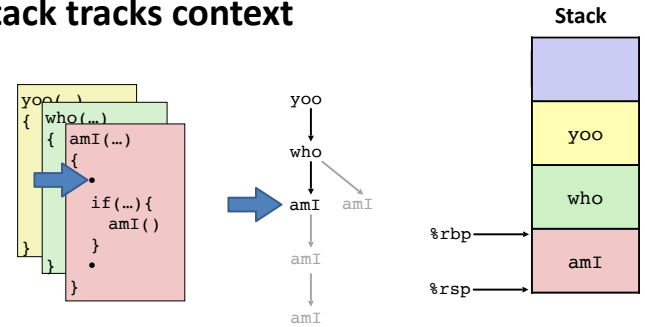
10

Call stack tracks context



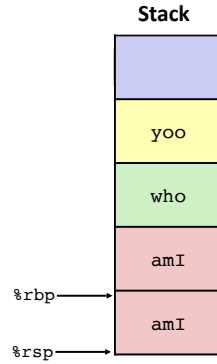
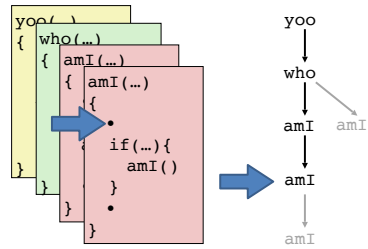
11

Call stack tracks context

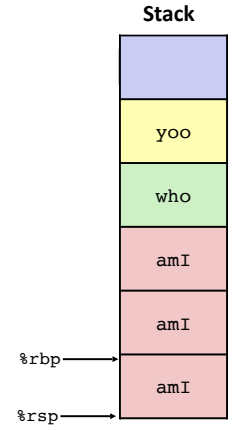
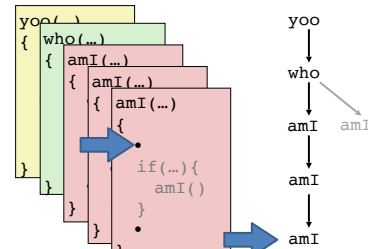


12

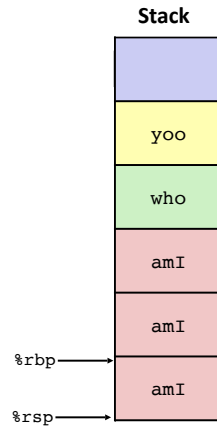
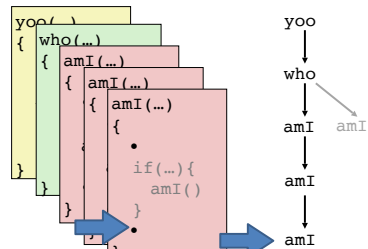
Call stack tracks context



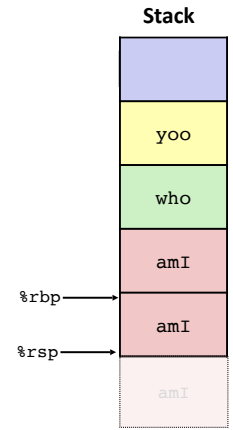
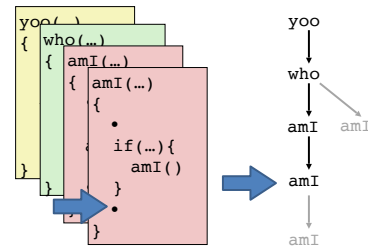
Call stack tracks context



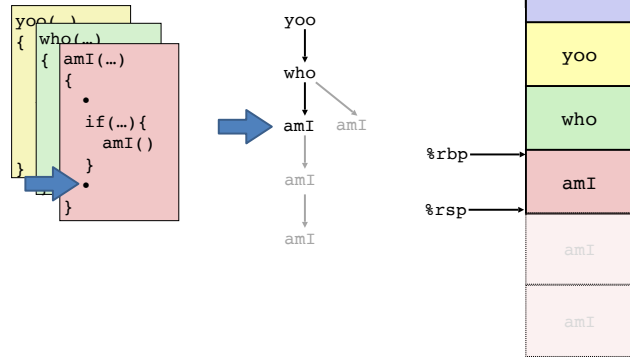
Call stack tracks context



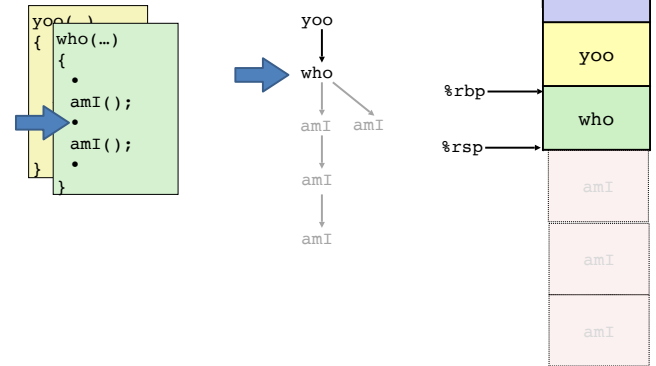
Call stack tracks context



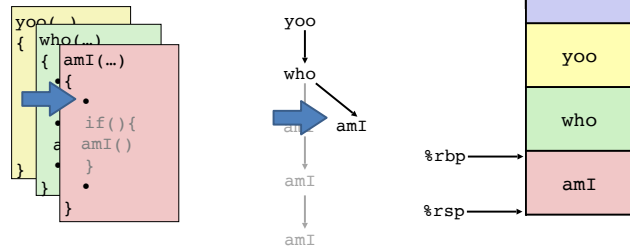
Call stack tracks context



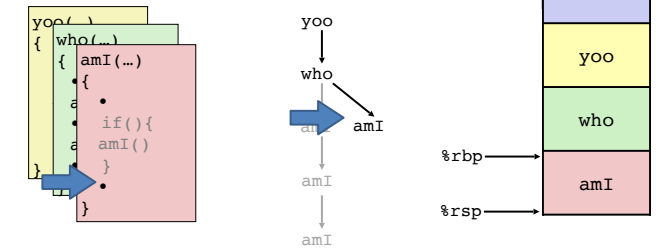
Call stack tracks context



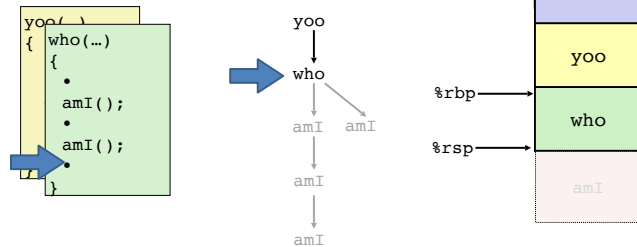
Call stack tracks context



Call stack tracks context

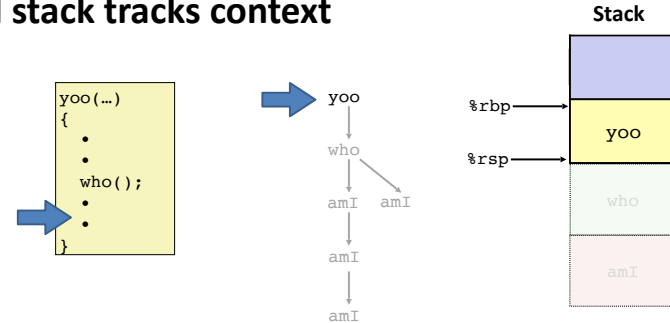


Call stack tracks context



21

Call stack tracks context



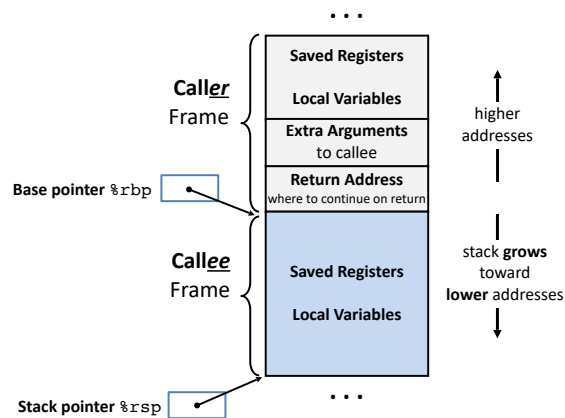
22

The call stack supports procedures

Stack frame: section of stack used by one procedure *call* to store context while running.

Procedure code manages stack frames explicitly.

- **Setup:** allocate space at start of procedure.
- **Cleanup:** deallocate space before return.



23

Procedure control flow instructions

Procedure call: `callq target`

1. Push return address on stack
2. Jump to *target*

Return address: Address of instruction after `call`.

```
400544: callq 400550 <mult2>
400549: movq %rax, (%rbx)
```

Procedure return: `retq`

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

24

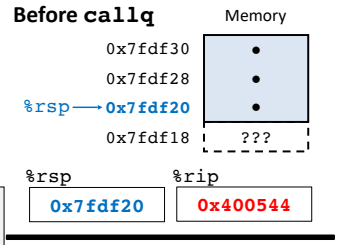
Call example

```

0000000000400540 <multstore>:
.
.
.
400544: callq 400550 <mult2>
400549: mov  %rax, (%rbx)
.
.
0000000000400550 <mult2>:
400550: mov  %rdi,%rax
.
.
.
400557: retq
    
```



- callq target**
1. Push return address on stack
 2. Jump to *target*



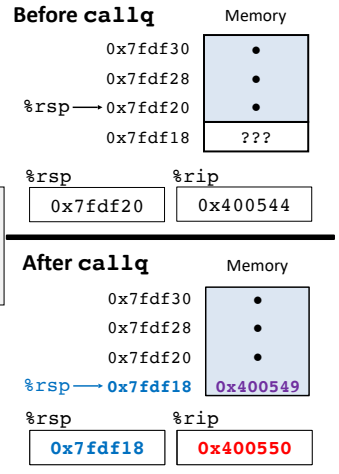
Call example

```

0000000000400540 <multstore>:
.
.
.
400544: callq 400550 <mult2>
400549: mov  %rax, (%rbx)
.
.
0000000000400550 <mult2>:
400550: mov  %rdi,%rax
.
.
.
400557: retq
    
```



- callq target**
1. Push return address on stack
 2. Jump to *target*



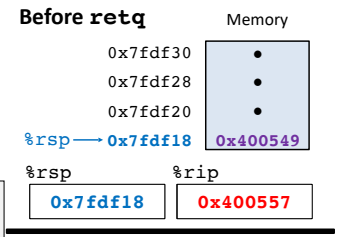
Return example

```

0000000000400540 <multstore>:
.
.
.
400544: callq 400550 <mult2>
400549: mov  %rax, (%rbx)
.
.
0000000000400550 <mult2>:
400550: mov  %rdi,%rax
.
.
.
400557: retq
    
```



- retq**
1. Pop return address from stack
 2. Jump to return address

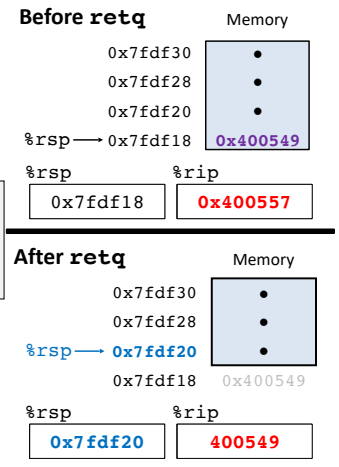


Return example

```

0000000000400540 <multstore>:
.
.
.
400544: callq 400550 <mult2>
400549: mov  %rax, (%rbx)
.
.
0000000000400550 <mult2>:
400550: mov  %rdi,%rax
.
.
.
400557: retq
    
```

- retq**
1. Pop return address from stack
 2. Jump to return address



Procedure data flow conventions

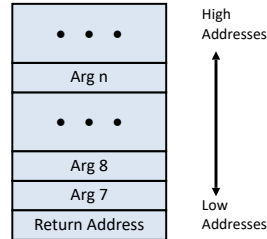
First 6 arguments:
passed in registers

Arg 1	%rdi	<i>Diane's</i>
Arg 2	%rsi	<i>Silk</i>
Arg 3	%rdx	<i>Dress</i>
Arg 4	%rcx	<i>Costs</i>
Arg 5	%r8	<i>\$89</i>
Arg 6	%r9	

Return value:
passed in %rax

%rax

Remaining arguments:
passed on stack (in memory)



29

Procedure call / stack frame example

```

step_up:
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
    
```

```

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

Passes address of local variable (in stack).

Uses memory through pointer.

```

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

```

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

30

Procedure call example (step 0)

main called step_up

```

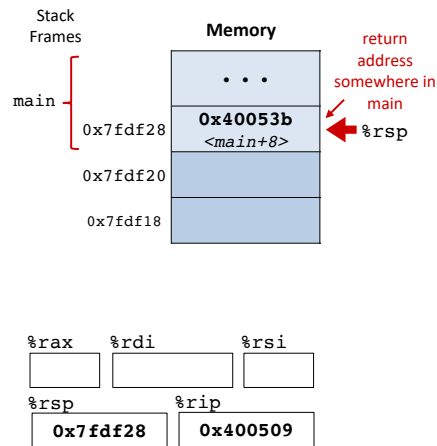
long step_up() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
    
```

```

step_up:
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
    
```



31

Procedure call example (step 1)

Allocate space for local vars

```

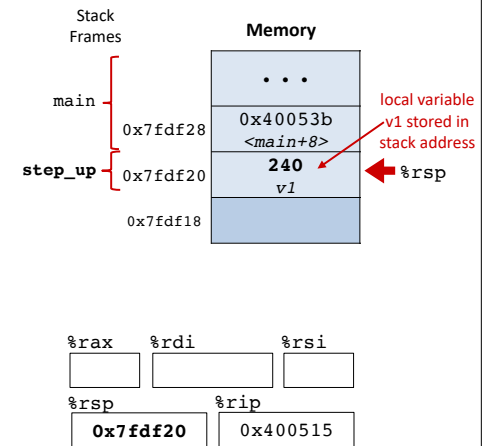
long step_up() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
    
```

```

step_up:
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
    
```



32

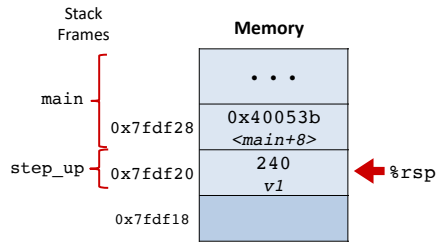
Procedure call example (step 2)

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

```
step_up:
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
```

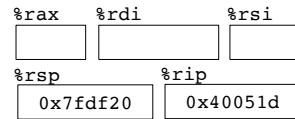
```
increment:
4004cd: movq (%rdi), %rax
4004d0: addq %rax, %rsi
4004d3: movq %rsi, (%rdi)
4004d6: retq
```

Set up args for call to increment



Place args in registers before call

★ Move the two args to the right registers



33

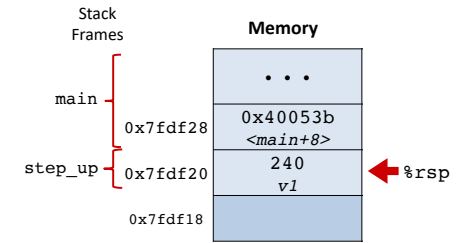
Procedure call example (step 2)

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

```
step_up:
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
```

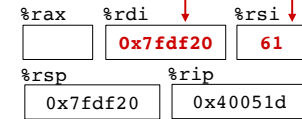
```
increment:
4004cd: movq (%rdi), %rax
4004d0: addq %rax, %rsi
4004d3: movq %rsi, (%rdi)
4004d6: retq
```

Set up args for call to increment



Place args in registers before call

1st arg for increment 2nd arg for increment



34

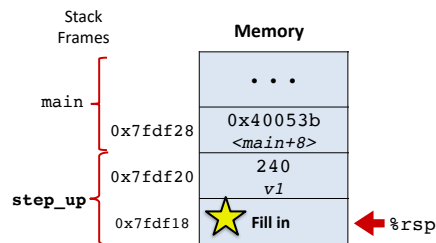
Procedure call example (step 3)

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

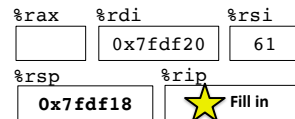
```
step_up:
400509: subq $8, %rsp
40050d: movq $240, (%rsp)
400515: movq %rsp, %rdi
400518: movl $61, %esi
40051d: callq 4004cd <increment>
400522: addq (%rsp), %rax
400526: addq $8, %rsp
40052a: retq
```

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

Call increment



call has two steps
(1) Push return address on stack
(2) Jump to target



35

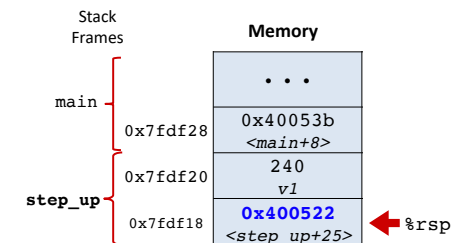
Procedure call example (step 3)

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

```
step_up:
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
```

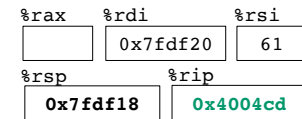
```
increment:
4004cd: movq (%rdi), %rax
4004d0: addq %rax, %rsi
4004d3: movq %rsi, (%rdi)
4004d6: retq
```

Call increment



call has two steps

(1) Push return address on stack



36

Procedure call example (step 4)

Run increment

```

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

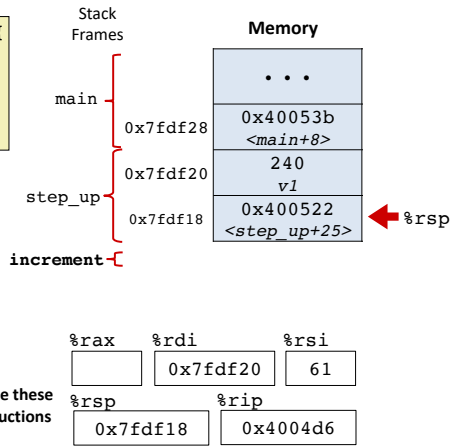
```

step_up:
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
    
```

★ Execute these 3 instructions



37

Procedure call example (step 4)

Run increment

```

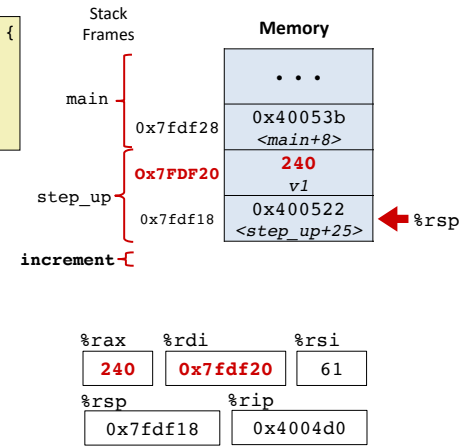
10 long increment(long* p, long val) {
    long x = *p;
    long y = x + val;
    *p = y;
    return x;
}
    
```

```

step_up:
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
    
```



38

Procedure call example (step 4)

Run increment

```

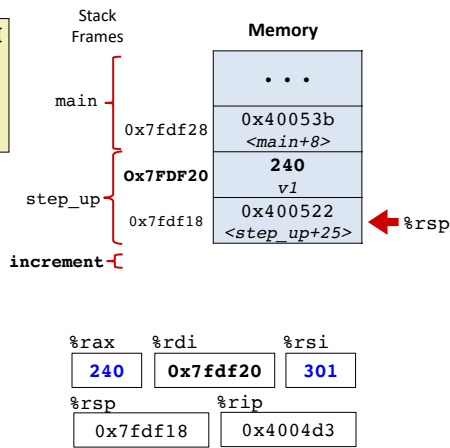
10 long increment(long* p, long val) {
    long x = *p;
    long y = x + val;
    *p = y;
    return x;
}
    
```

```

step_up:
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
    
```



39

Procedure call example (step 4)

Run increment

```

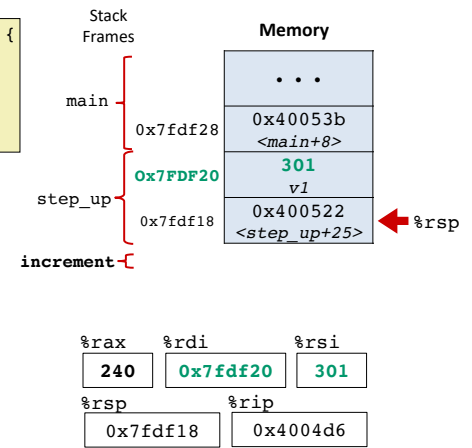
10 long increment(long* p, long val) {
    long x = *p;
    long y = x + val;
    *p = y;
    return x;
}
    
```

```

step_up:
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
    
```



40

Procedure call example (step 5a)

```

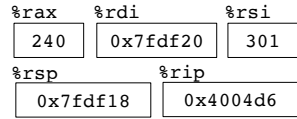
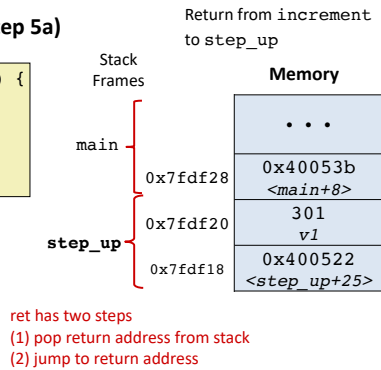
long increment(long* p, long val) {
    long x = *p;
    long y = x + val;
    *p = y;
    return x;
}
    
```

```

step_up:
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 5a)

```

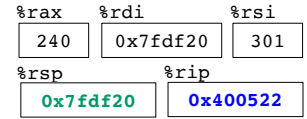
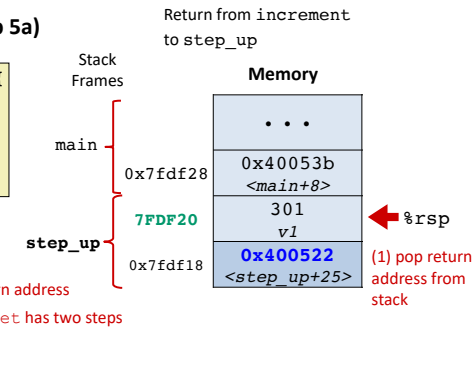
long increment(long* p, long val) {
    long x = *p;
    long y = x + val;
    *p = y;
    return x;
}
    
```

```

step_up:
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)

```

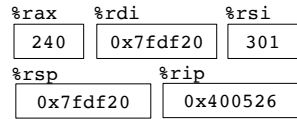
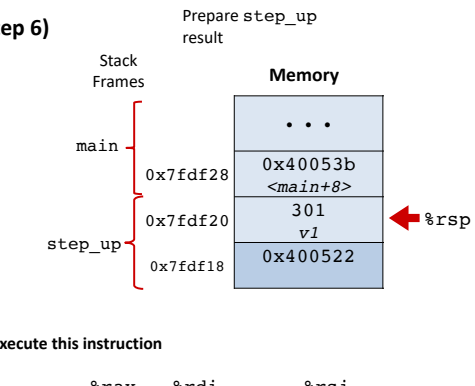
long step_up() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
    
```

```

step_up:
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)

```

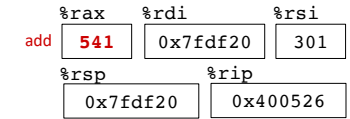
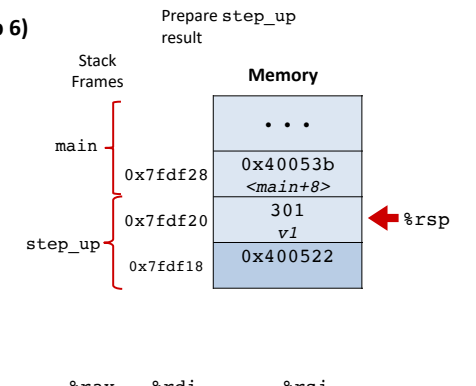
long step_up() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
    
```

```

step_up:
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
    
```

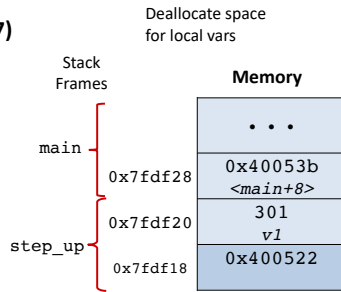


Procedure call example (step 7)

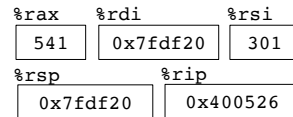
```
long step_up() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

```
step_up:
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
```



★ Execute this instruction



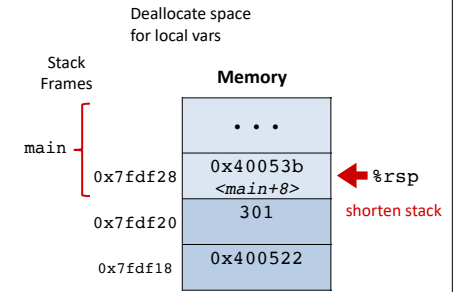
45

Procedure call example (step 7)

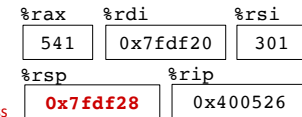
```
long step_up() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

```
step_up:
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
```



higher address



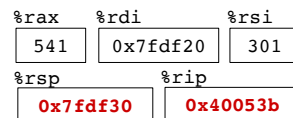
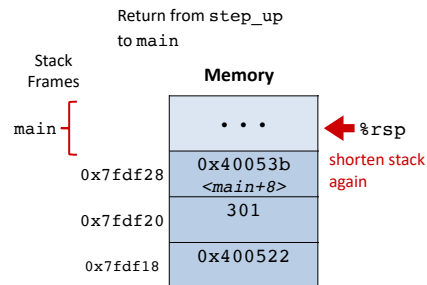
46

Procedure call example (step 8)

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

```
step_up:
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
```



47

Implementing procedures

Have we now seen how this is done?

1. How does a caller pass arguments to a procedure? ✓
2. How does a caller receive a return value from a procedure? ✓
3. How does a procedure know where to return (what code to execute next when done)? ✓
4. Where does a procedure store local variables? ✓
5. How do procedures share limited registers and memory? ??

48

Register saving conventions

yoo calls who:
Caller **Callee**

```
yoo(...) {
    . . .
    who();
    . . .
}
```

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 register conventions

%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

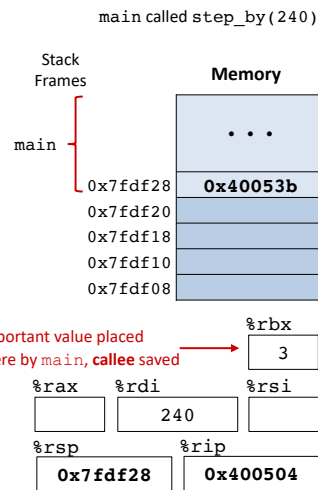
Callee-save example (step 0)

Similar function, but now takes an arg for the local variable

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

```
step_by:
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
```

caller saved: %rax, %rdi, %rsi
 callee saved: %rbx



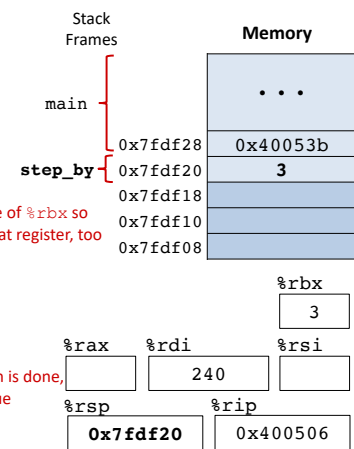
Callee-save example (step 1)

Save register %rbx

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

```
step_by:
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
```

caller saved: %rax, %rdi, %rsi
 callee saved: %rbx

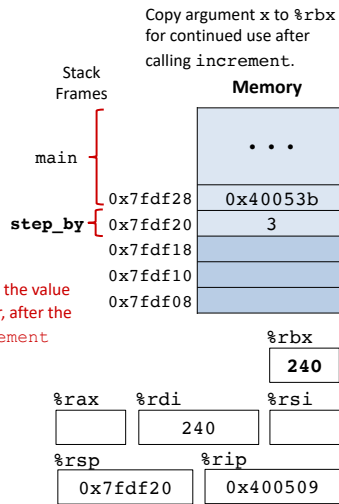


Callee-save example (step 2)

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

```
step_by:
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
```

caller saved: %rax, %rdi, %rsi
callee saved: %rbx



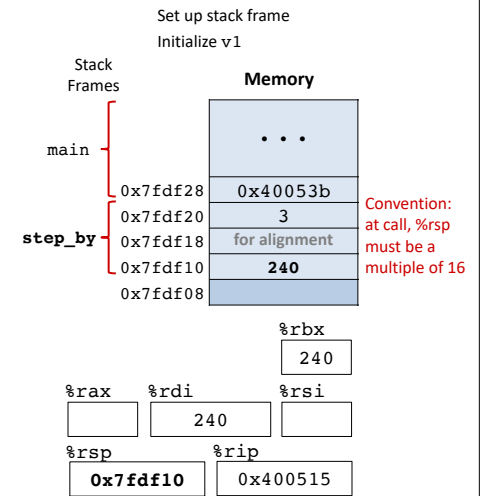
53

Callee-save example (step 3)

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

```
step_by:
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
```

caller saved: %rax, %rdi, %rsi
callee saved: %rbx



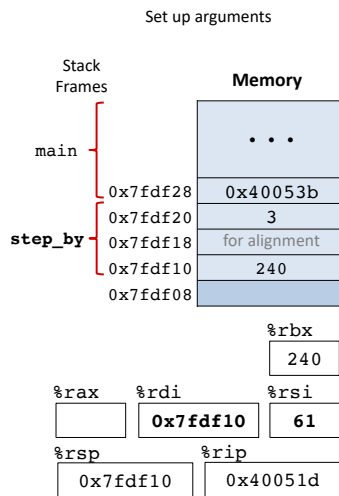
54

Callee-save example (step 4)

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

```
step_by:
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
```

caller saved: %rax, %rdi, %rsi
callee saved: %rbx



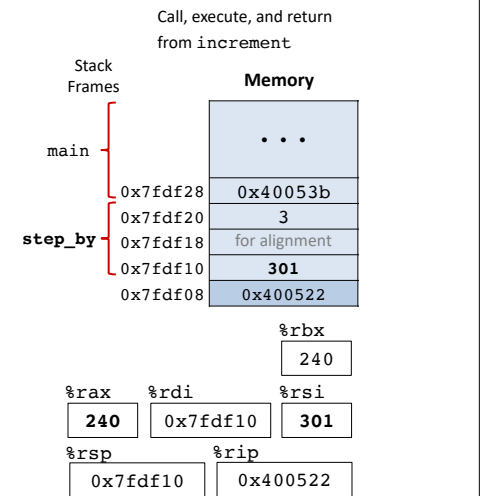
55

Callee-save example (step 5)

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

```
step_by:
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
```

caller saved: %rax, %rdi, %rsi
callee saved: %rbx



56

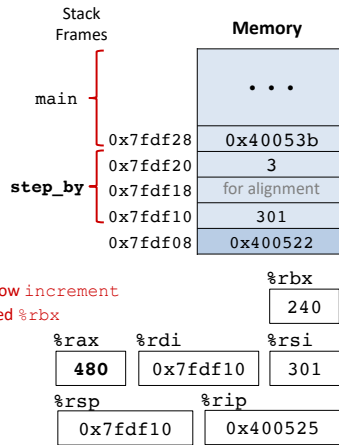
Callee-save example (step 6)

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

```
step_by:
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
```

caller saved: %rax, %rdi, %rsi
callee saved: %rbx

Prepare return value



57

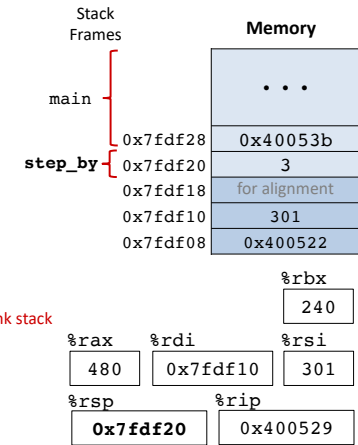
Callee-save example (step 7)

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

```
step_by:
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
```

caller saved: %rax, %rdi, %rsi
callee saved: %rbx

Clean up stack frame



58

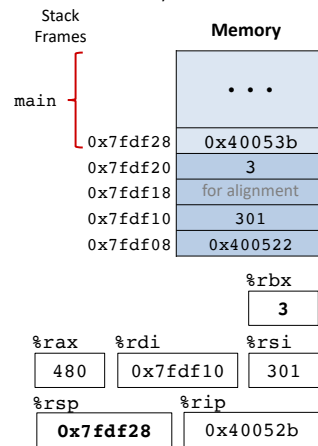
Callee-save example (step 8)

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

```
step_by:
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
```

caller saved: %rax, %rdi, %rsi
callee saved: %rbx

Restore register %rbx
Ready to return



59

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
```

base case/
condition

recursive
case

x&1 in %rbx
across call

save/restore

%rbx (callee-save)

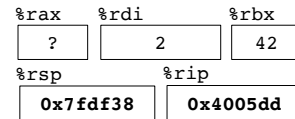
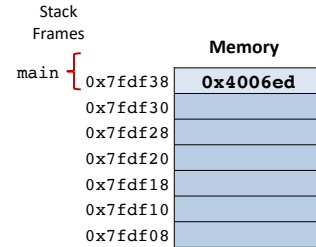
60

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
```



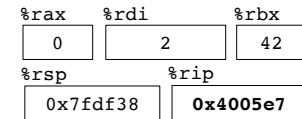
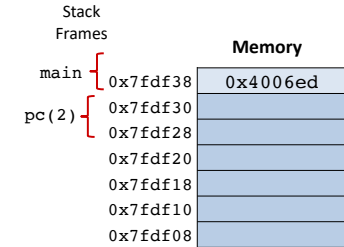
61

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
```



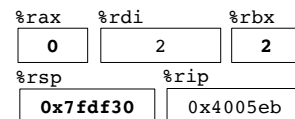
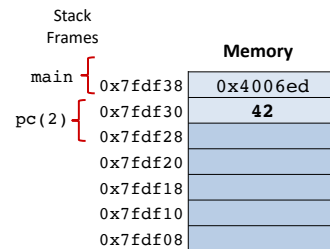
62

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
```



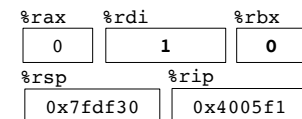
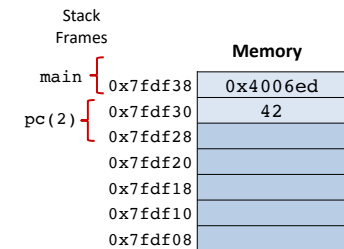
63

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
```

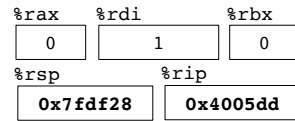
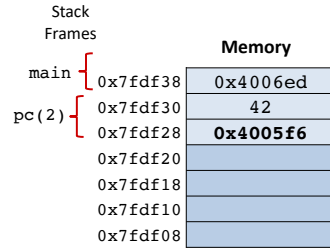


64

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

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

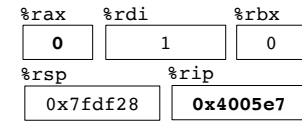
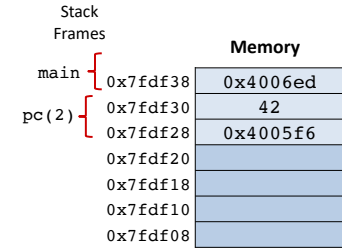
```
pcount:
4005dd: movl $0, %eax
4005e2: testq %rdi, %rdi
4005e5: je 4005fa <.L6>
4005e7: pushq %rbx
4005e8: movq %rdi, %rbx
4005eb: andl $1, %ebx
4005ee: shrq %rdi
4005f1: callq pcount
4005f6: addq %rbx, %rax
4005f9: popq %rbx
.L6:
4005fa: rep
4005fb: retq
```



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

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

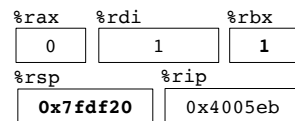
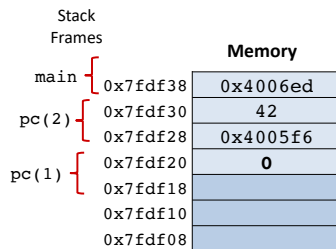
```
pcount:
4005dd: movl $0, %eax
4005e2: testq %rdi, %rdi
4005e5: je 4005fa <.L6>
4005e7: pushq %rbx
4005e8: movq %rdi, %rbx
4005eb: andl $1, %ebx
4005ee: shrq %rdi
4005f1: callq pcount
4005f6: addq %rbx, %rax
4005f9: popq %rbx
.L6:
4005fa: rep
4005fb: retq
```



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

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

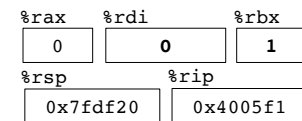
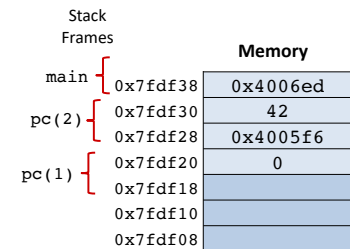
```
pcount:
4005dd: movl $0, %eax
4005e2: testq %rdi, %rdi
4005e5: je 4005fa <.L6>
4005e7: pushq %rbx
4005e8: movq %rdi, %rbx
4005eb: andl $1, %ebx
4005ee: shrq %rdi
4005f1: callq pcount
4005f6: addq %rbx, %rax
4005f9: popq %rbx
.L6:
4005fa: rep
4005fb: retq
```



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

```
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
```



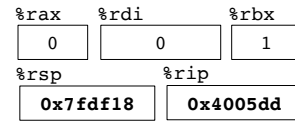
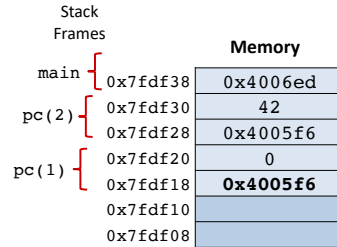
Recursion Example: pcount(2) → pcount(1) → **pcount(0)**

```

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
    
```



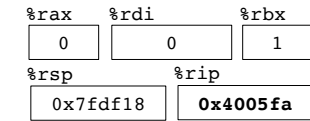
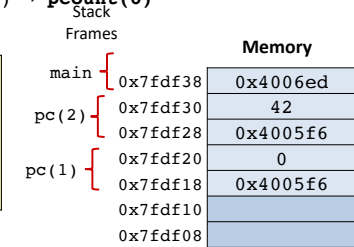
Recursion Example: pcount(2) → pcount(1) → **pcount(0)**

```

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

```

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
    
```



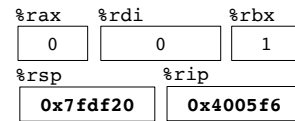
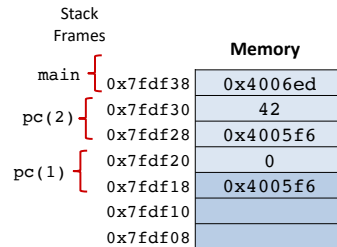
Recursion Example: pcount(2) → pcount(1) → **pcount(0)**

```

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

```

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
    
```



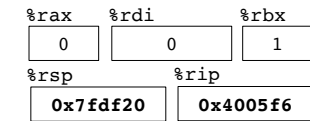
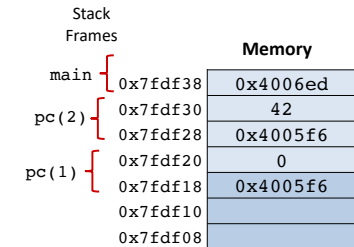
Recursion Example: pcount(2) → pcount(1) → **pcount(0)**

```

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)`

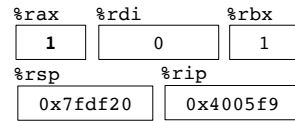
```

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
    
```

Stack Frames		Memory
main	{ 0x7fdf38	0x4006ed
	{ 0x7fdf30	42
pc(2)	{ 0x7fdf28	0x4005f6
	{ 0x7fdf20	0
pc(1)	{ 0x7fdf18	0x4005f6
	{ 0x7fdf10	
	{ 0x7fdf08	



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

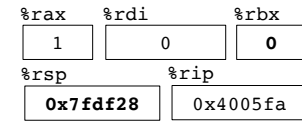
```

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
    
```

Stack Frames		Memory
main	{ 0x7fdf38	0x4006ed
	{ 0x7fdf30	42
pc(2)	{ 0x7fdf28	0x4005f6
	{ 0x7fdf20	0
	{ 0x7fdf18	0x4005f6
	{ 0x7fdf10	
	{ 0x7fdf08	



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

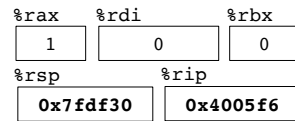
```

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
    
```

Stack Frames		Memory
main	{ 0x7fdf38	0x4006ed
	{ 0x7fdf30	42
pc(2)	{ 0x7fdf28	0x4005f6
	{ 0x7fdf20	0
	{ 0x7fdf18	0x4005f6
	{ 0x7fdf10	
	{ 0x7fdf08	



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

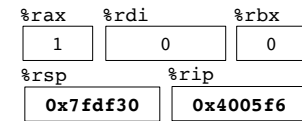
```

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
    
```

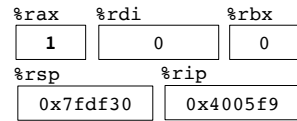
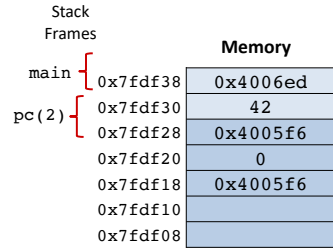
Stack Frames		Memory
main	{ 0x7fdf38	0x4006ed
	{ 0x7fdf30	42
pc(2)	{ 0x7fdf28	0x4005f6
	{ 0x7fdf20	0
	{ 0x7fdf18	0x4005f6
	{ 0x7fdf10	
	{ 0x7fdf08	



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);
    }
}
```

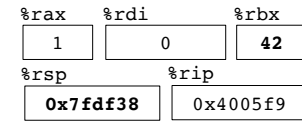
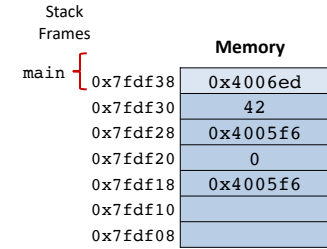
```
pcount:
4005dd: movl $0, %eax
4005e2: testq %rdi, %rdi
4005e5: je 4005fa <.L6>
4005e7: pushq %rbx
4005e8: movq %rdi, %rbx
4005eb: andl $1, %ebx
4005ee: shrq %rdi
4005f1: callq pcount
4005f6: addq %rbx, %rax
4005f9: popq %rbx
.L6:
4005fa: rep
4005fb: retq
```



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);
    }
}
```

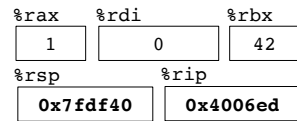
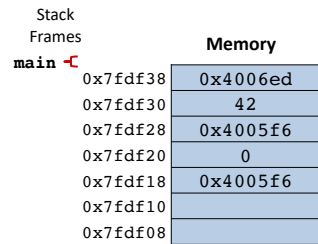
```
pcount:
4005dd: movl $0, %eax
4005e2: testq %rdi, %rdi
4005e5: je 4005fa <.L6>
4005e7: pushq %rbx
4005e8: movq %rdi, %rbx
4005eb: andl $1, %ebx
4005ee: shrq %rdi
4005f1: callq pcount
4005f6: addq %rbx, %rax
4005f9: popq %rbx
.L6:
4005fa: rep
4005fb: retq
```



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
```

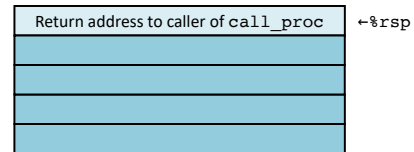


Stack storage example (1)

optional

```
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
...
```

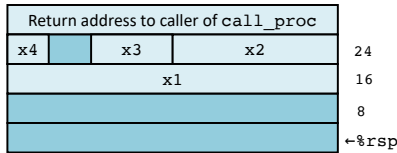


Stack storage example
(2) Allocate local vars

optional

```
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
    . . .
```

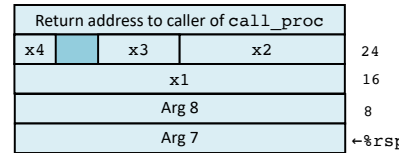


Stack storage example
(3) setup args to proc

optional

```
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
    . . .
```



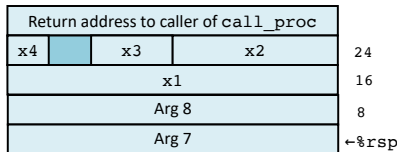
Arguments passed in (in order):
 %rdi, %rsi, %rdx, %rcx, %r8, %r9

Stack storage example
(4) after call to proc

optional

```
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
```

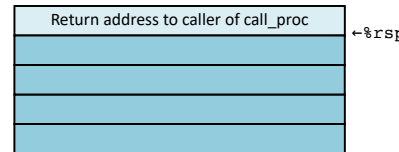


Stack storage example
(5) deallocate local vars

optional

```
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
```



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

