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**?

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:

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

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`

	Memory
0x7fdf30	•
0x7fdf28	•
0x7fdf20	•
0x7fdf18	???

`%rsp` 0x7fdf20 `%rip` 0x400544

After `callq`

	Memory
0x7fdf30	•
0x7fdf28	•
0x7fdf20	•
0x7fdf18	0x400549

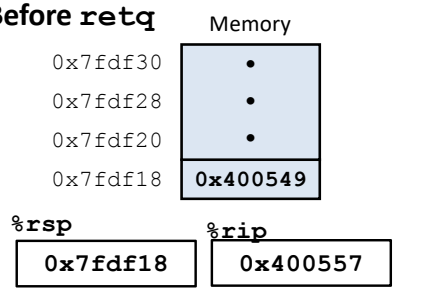
`%rsp` 0x7fdf18 `%rip` 0x400550

Return Example

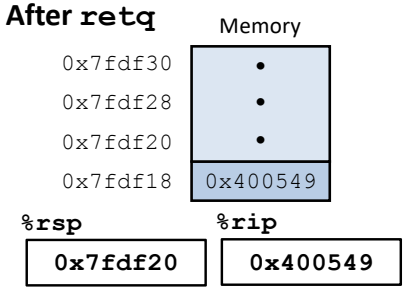
```
0000000000400540 <multstore>:
  •
  •
  400544: callq  400550 <mult2>
  400549: mov    %rax,(%rbx)
  •
```

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

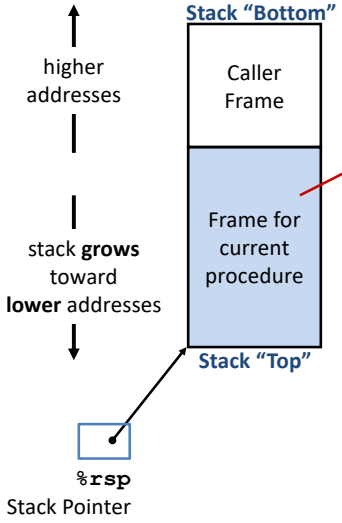
Before `retq`



After `retq`



Stack frames support procedure calls.



Contents

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

Management

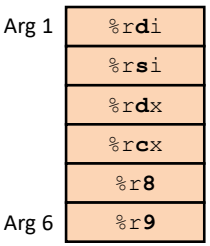
Space allocated when procedure is entered
"Setup" code

Space deallocated before return
"Finish" code

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

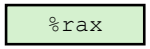
Procedure Data Flow Conventions

First 6 arguments passed in registers

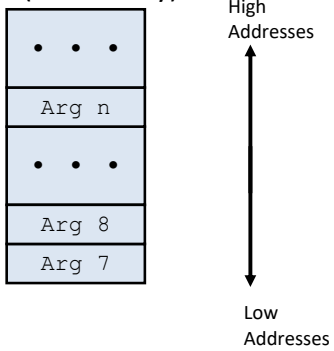


Diane's Silk Dress Costs $8 9

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;
}
```

```
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
```

Passes address of local variable (in stack).

Uses memory through pointer.

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

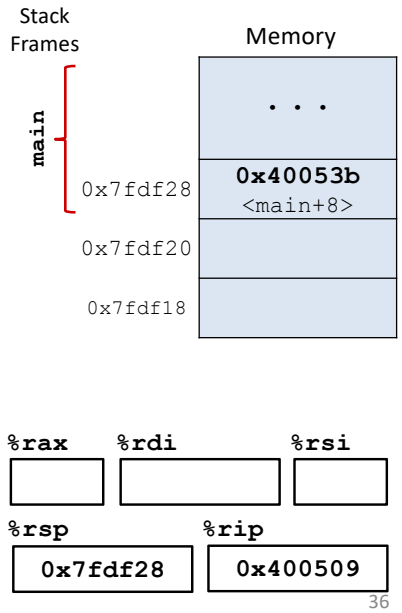
```
increment:
4004cd:  movq  (%rdi), %rax
4004d0:  addq  %rax, %rsi
4004d3:  movq  %rsi, (%rdi)
4004d6:  retq
```


Procedure Call Example

```
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
```

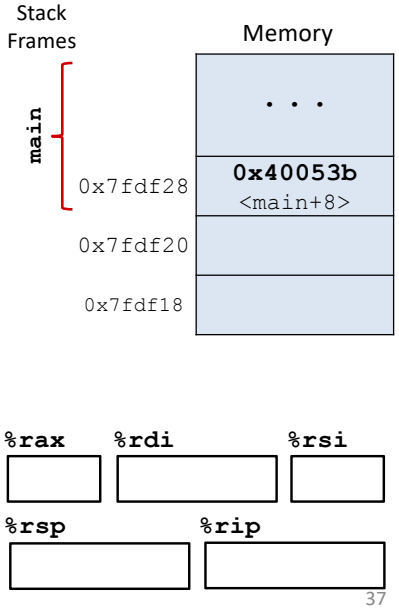


Procedure Call Example

```
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
```

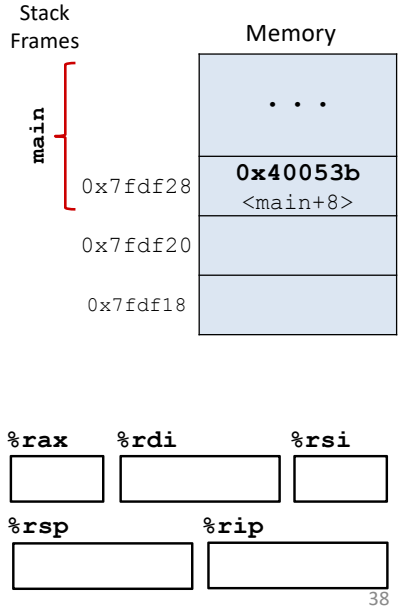


Procedure Call Example

```
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
```

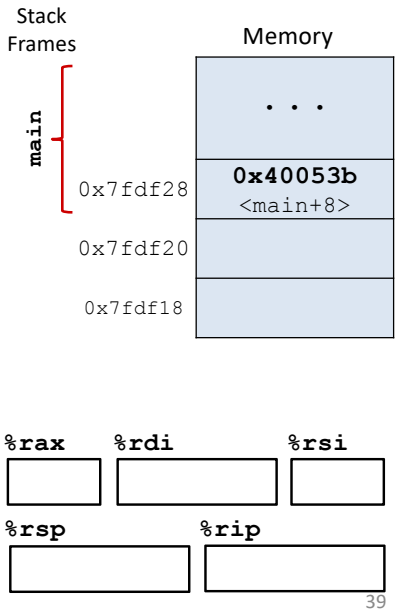


Procedure Call Example

```
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

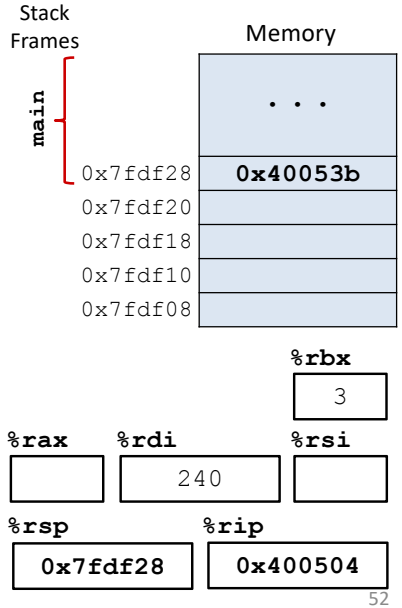
x86-64 64-bit Register Conventions

%rax	Return value – Caller saved	%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

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

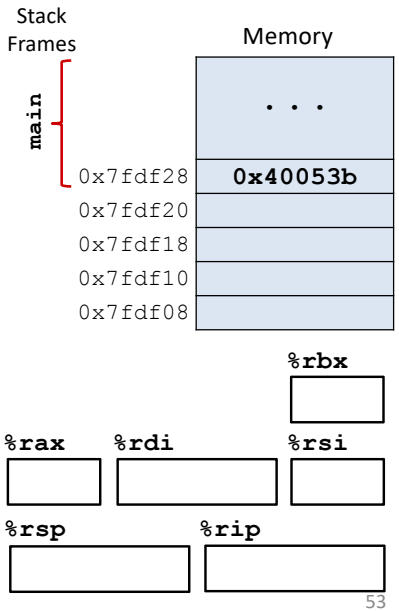
```
call_incr2:
400504:  pushq %rbx
400506:  movq  %rdi, %rbx
400509:  subq  $16, %rsp
40050d:  movq  %rdi, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  %rbx, %rax
400525:  addq  $16, %rsp
400529:  popq  %rbx
40052b:  retq
```



Callee-Save Example

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

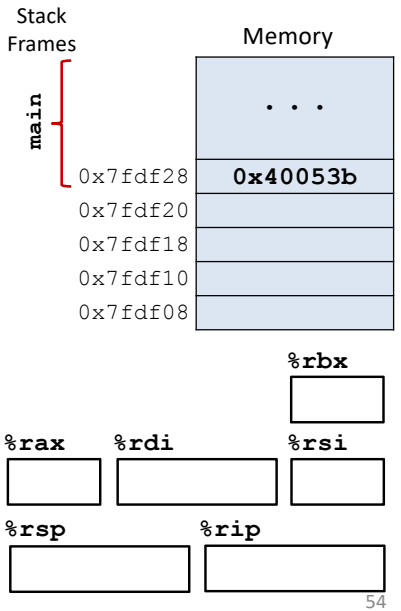
```
call_incr2:
400504:  pushq %rbx
400506:  movq  %rdi, %rbx
400509:  subq  $16, %rsp
40050d:  movq  %rdi, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  %rbx, %rax
400525:  addq  $16, %rsp
400529:  popq  %rbx
40052b:  retq
```



Callee-Save Example

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

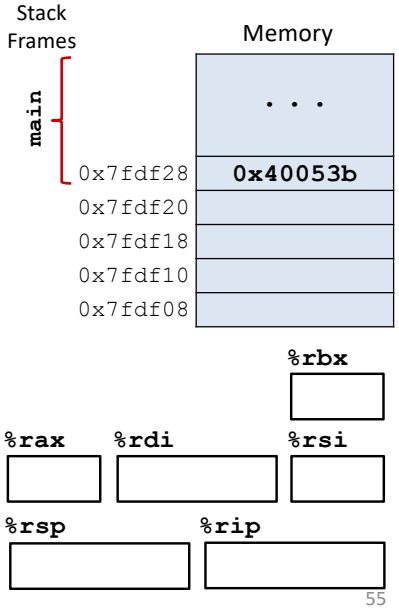
```
call_incr2:
400504:  pushq %rbx
400506:  movq  %rdi, %rbx
400509:  subq  $16, %rsp
40050d:  movq  %rdi, (%rsp)
400515:  movq  %rsp, %rdi
400518:  movl  $61, %esi
40051d:  callq 4004cd <increment>
400522:  addq  %rbx, %rax
400525:  addq  $16, %rsp
400529:  popq  %rbx
40052b:  retq
```



Callee-Save Example

```
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
```

base case/ condition

save/restore %rbx (callee-save)

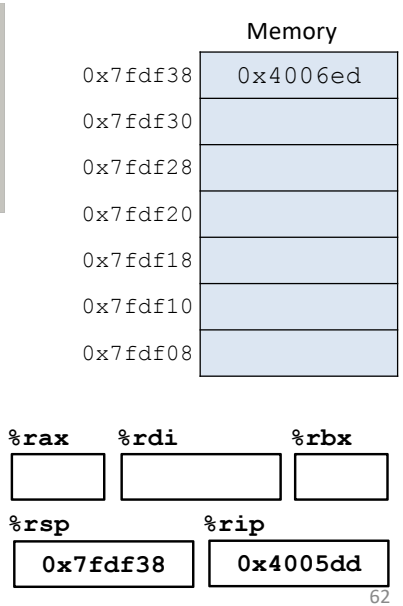
x&1 in %rbx across call

recursive case

Recursion Example Handout

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

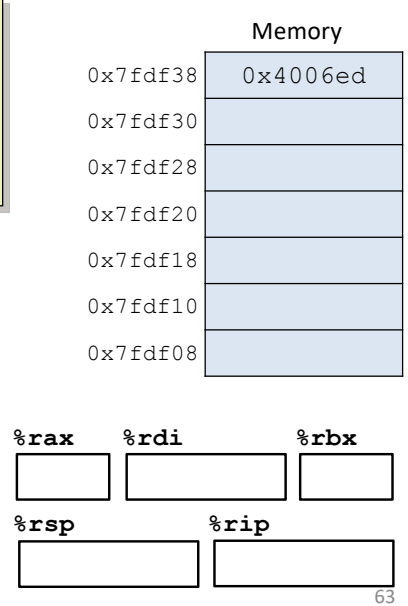
```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```



Recursion Example Handout

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

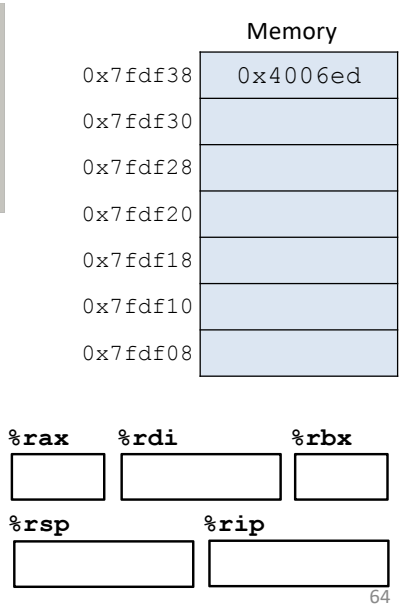
```
pcount:
4005dd:  movl   $0, %eax
4005e2:  testq %rdi, %rdi
4005e5:  je     4005fa <.L6>
4005e7:  pushq %rbx
4005e8:  movq   %rdi, %rbx
4005eb:  andl   $1, %ebx
4005ee:  shrq   %rdi
4005f1:  callq pcount
4005f6:  addq   %rbx, %rax
4005f9:  popq   %rbx
.L6:
4005fa:  rep
4005fb:  retq
```



Recursion Example Handout

```
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 Handout

```
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
```