Procedures and the **Call Stack**

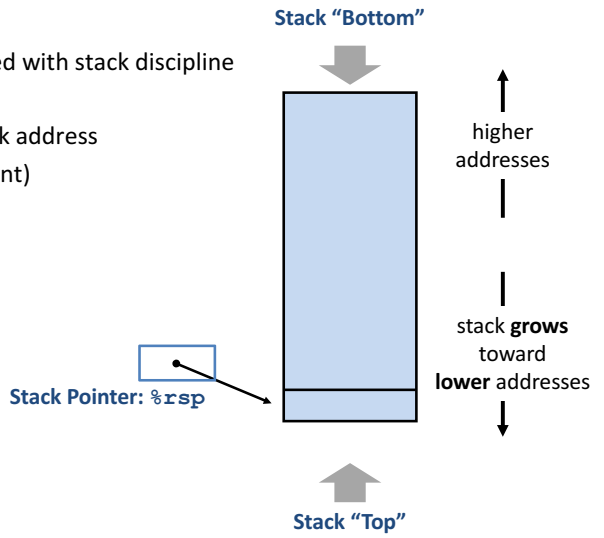
Topics

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

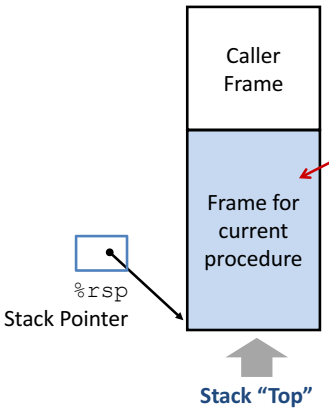
Call Stack

Memory region managed with stack discipline

`%rsp` holds lowest stack address
(address of "top" element)



Stack frames support procedure calls.



Contents

Local variables
Function arguments (after first 6)
Return information
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 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)
```

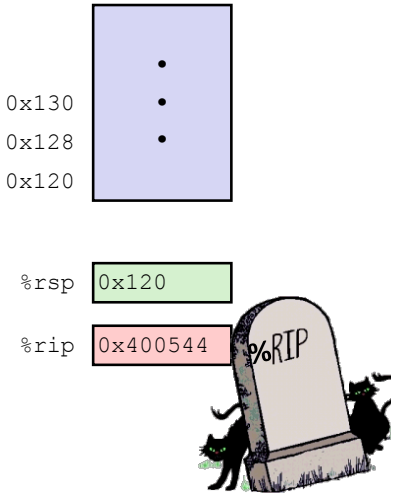
Procedure return: `retq`

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

Call Example (step 1)

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

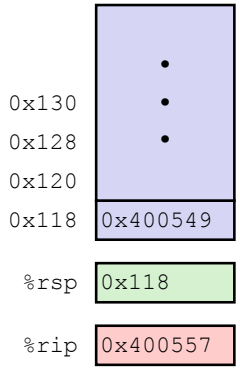
```
0000000000400550 <mult2>:
  400550:  mov    %rdi,%rax
  •
  •
  400557:  retq
```



Return Example (step 1)

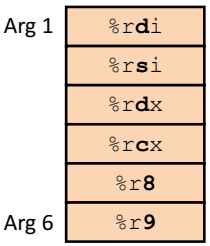
```
0000000000400540 <multstore>:
  •
  •
  400544: callq  400550 <mult2>
  400549: mov    %rax,(%rbx)
  •
  •
```

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



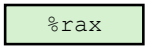
Procedure Data Flow

First 6 arguments passed in registers

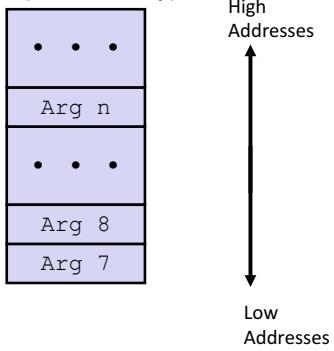


Diane's Silk Dress Costs $8 9

Return value

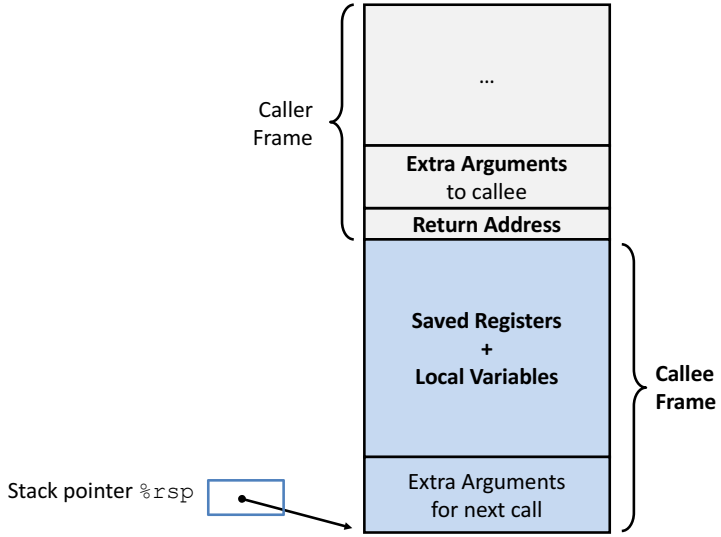


Remaining arguments passed on stack (in memory)



Only allocate stack space when needed

Stack Frame



Example: increment

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

```
increment:
  movq    (%rdi), %rax
  addq    %rax, %rsi
  movq    %rsi, (%rdi)
  ret
```

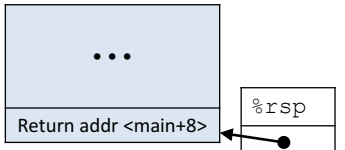
Register	Use(s)
%rdi	Argument **p**
%rsi	Argument **val, y**
%rax	**x**, Return value

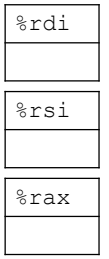
Procedure Call Example (initial state)

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

```
call_incr:
  subq    $16, %rsp
  movq    $240, 8(%rsp)
  movl    $61, %esi
  leaq    8(%rsp), %rdi
  call    increment
  addq    8(%rsp), %rax
  addq    $16, %rsp
  ret
```

Initial Stack Structure





Procedure Call Example (step 4)

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

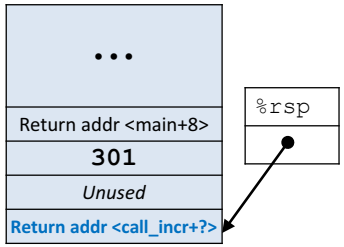
```
call_incr:
  subq    $16, %rsp
  movq    $240, 8(%rsp)
  movl    $61, %esi
  leaq    8(%rsp), %rdi
  call    increment
  addq    8(%rsp), %rax
  addq    $16, %rsp
  ret
```

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

```
increment:
  movq    (%rdi), %rax # x = *p
  addq    %rax, %rsi   # y = x+61
  movq    %rsi, (%rdi) # *p = y
  ret
```

v1 in call_incr →

Stack Structure



Procedure Call Example (step 6)

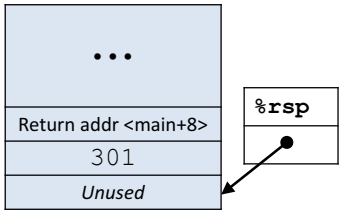
```
long call_incr() {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return v1+v2;
}
```

```
call_incr:
  subq    $16, %rsp
  movq    $240, 8(%rsp)
  movl    $61, %esi
  leaq    8(%rsp), %rdi
  call    increment
  addq    8(%rsp), %rax
  addq    $16, %rsp
  ret
```

Update %rax: v1+v2

v1 in call_incr →

Stack Structure



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)

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

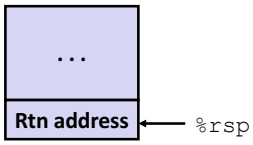
%rax	Return value – Caller saved	%r8	Argument #5 – Caller saved
%rbx	**Callee** saved	%r9	Argument #6 – Caller saved
%rcx	Argument #4 – Caller saved	%r10	Caller saved
%rdx	Argument #3 – Caller saved	%r11	Caller Saved
%rsi	Argument #2 – Caller saved	%r12	**Callee** saved
%rdi	Argument #1 – Caller saved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

Callee-Saved Example

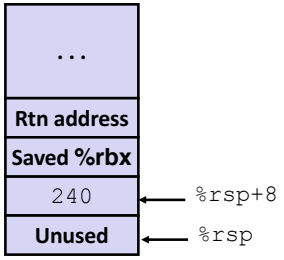
```
long call_incr2(long x) {
    long v1 = 240;
    long v2 = increment(&v1, 61);
    return x+v2;
}
```

```
call_incr2:
  pushq   %rbx
  subq    $16, %rsp
  movq    %rdi, %rbx
  movq    $240, 8(%rsp)
  movl    $61, %esi
  leaq    8(%rsp), %rdi
  call    increment
  addq    %rbx, %rax
  addq    $16, %rsp
  popq    %rbx
  ret
```

Begin/End Stack Structure



Stack Structure in call_incr2



Recursive Function

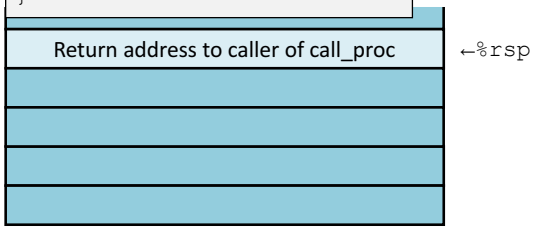
```
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0) {
    return 0;
  } else {
    return (x & 1)
           + pcount_r(x >> 1);
  }
}
```

```
pcount_r:
  movl    $0, %eax
  testq   %rdi, %rdi
  je      .L6
  pushq   %rbx
  movq    %rdi, %rbx
  andl    $1, %ebx
  shrq    %rdi
  call    pcount_r
  addq    %rbx, %rax
  popq    %rbx
.L6:
  rep; ret
```


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
  • • •
```



Procedure Summary

call, ret, push, pop

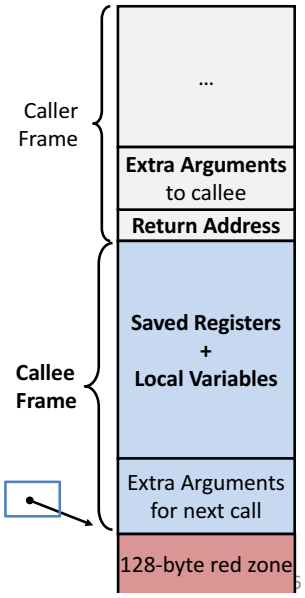
Stack discipline fits procedure call / return.*

- 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



*Take 251 to learn about languages where it doesn't.