CS 261
Fall 2016

Mike Lam, Professor

x86-64 Procedures

Topics

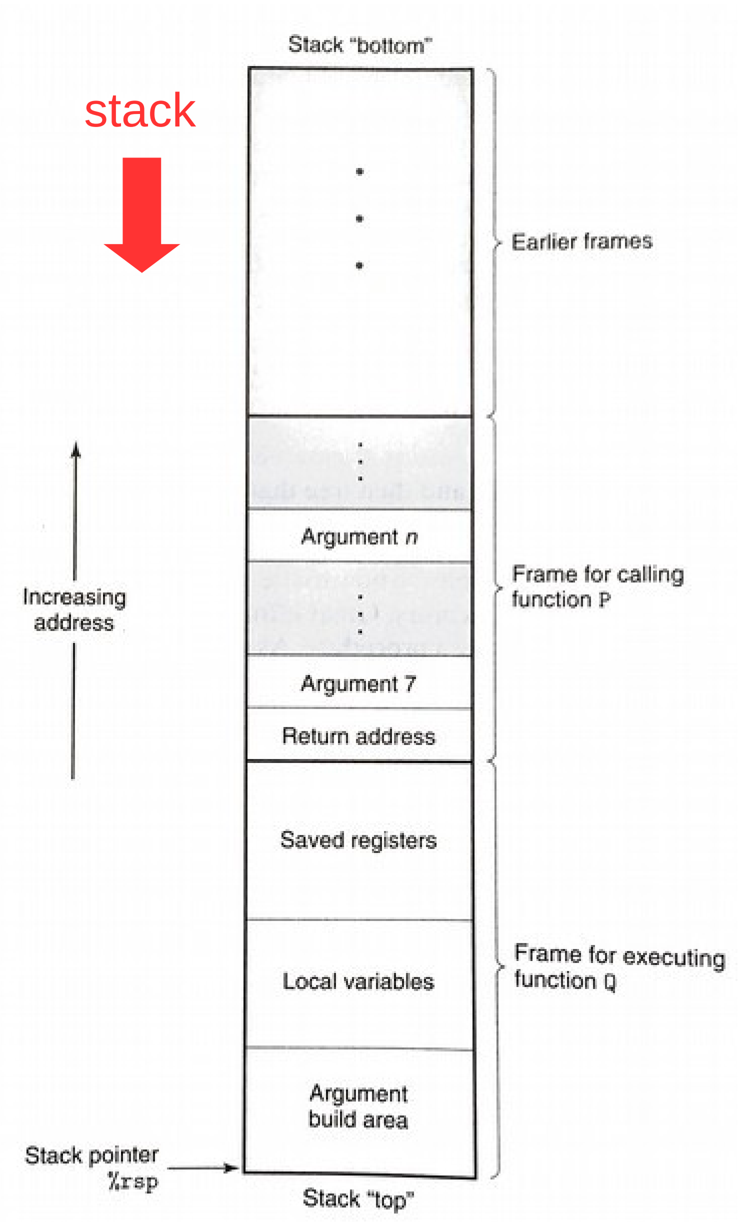
- Procedure calls
  - Runtime stack
  - Control transfer
  - Data transfer
  - Local storage
  - Recursive procedures

Procedure calls

- Procedures are a key abstraction in software
  - Provide modularity and encapsulation
  - Many alternative names: functions, methods, subroutines, handlers
- Well-designed procedures have:
  - Well-documented, strongly-typed input arguments
  - Well-documented return value(s)
  - Clear impact on program state (or no impact)
- Application Binary Interface (ABI)
  - Interface between program & system components at the binary level
  - Includes rules about how procedure calls are implemented
  - These rules are referred to as calling conventions
  - We will study the standard x86-64 calling conventions

Runtime stack

- Basic idea: keep a stack frame on the system stack for each function call
  - All active functions have a frame
  - Each frame stores information about a single active call
    - Arguments, local variables, return address
  - GDB's "backtrace" command follows the chain up
  - Recursion just works!
  - Caution: security can be compromised if a procedure writes past the end of its stack frame

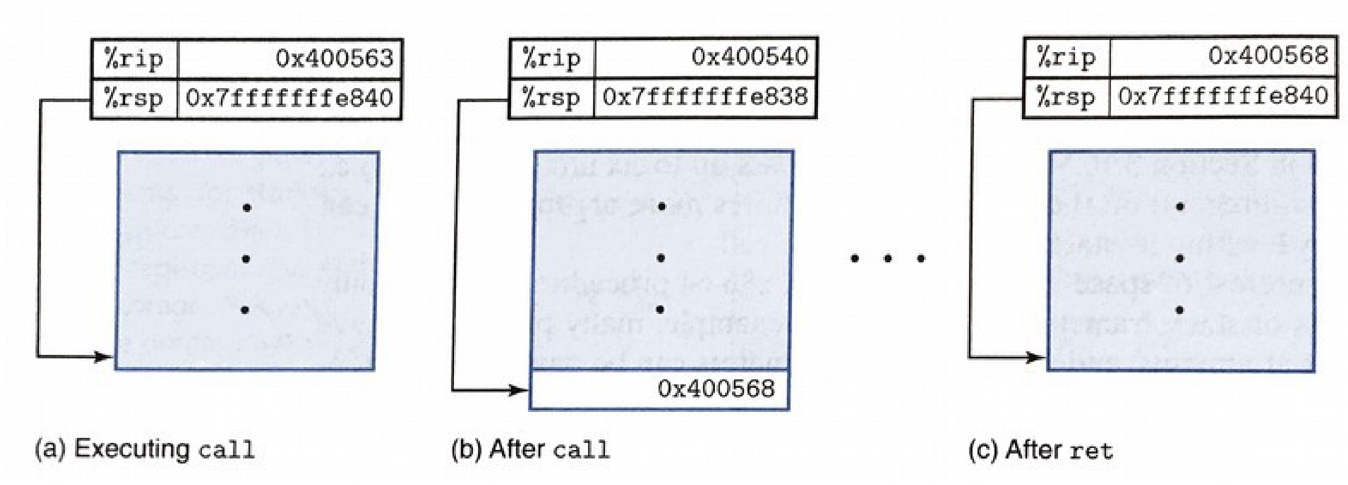


Control transfer

- Use stack to store return addresses
  - Return address: the instruction AFTER the `call`
  - `call` pushes return address onto stack
  - `ret` pops the return address and sets `%rip`

```
400550 <main>:
  ...
  400563   callq 400540 <foo>
  400568   mov 0x8(%rsp), %rdx
  ...
```

```
400550 <foo>:
  400540   push %rbx
  ...
  40054d   retq
```

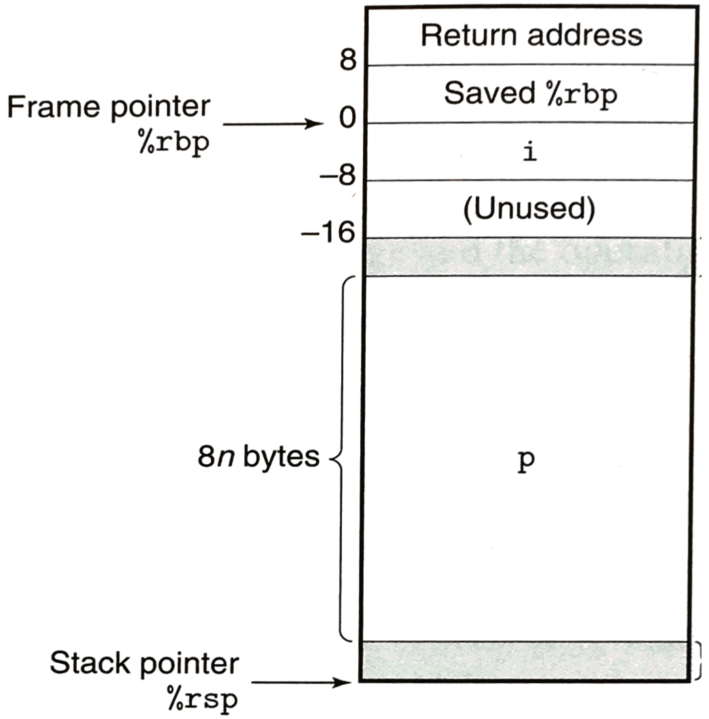


Data transfer

- Up to six integral (integer or pointer) arguments are passed via registers in x86-64:
  - `%rdi, %rsi, %rdx, %rcx, %r8, %r9`
  - Other arguments are passed on the stack
- A single return value is passed back via `%rax`
- Some registers are designated callee-saved
  - In x86-64: `%rbx, %rbp, %r12, %r13, %r14, %r15`
  - A procedure must save/restore these registers (often using push/pop) if they are used during the procedure
  - Other registers except `%rsp` are caller-saved (caller must save them if they need to be preserved)

Local storage

- Procedures can allocate space on the stack for local variables
  - Subtract # of bytes needed from `%rsp`
- Variable-sized allocations require special handling
  - Use base pointer (`%rbp`) to track "anchor" for current frame
  - Save previous base pointer on stack at beginning of function
  - Section 3.10.5 in textbook



Base pointers

- Use base pointer (`%rbp`) to track the beginning of current frame
  - Parameters at positive offsets
  - Local values at negative offsets
  - Chain of base pointers up the stack
  - Push/pop BP like return address

```
foo():               bar(x):            baz(x, y):
    int a,b              int c              int d
    bar(a)               bax(x,c)           return
    return               return
```

CALLER

Pre-call:

```
pushq <param2>
pushq <param1>
callq <func>
```

CALLEE

Prologue:

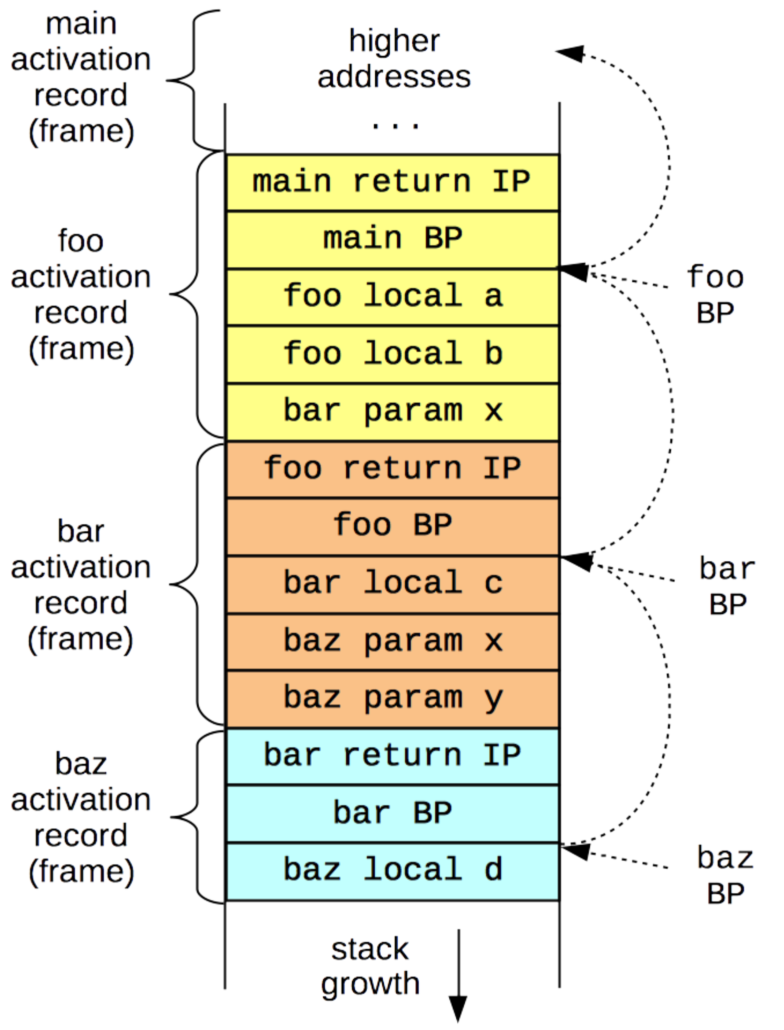
```
pushq %rbp
movq %rsp, %rbp
subq $n, %rsp
...
```

Epilogue:

```
movq %rbp, %rsp
popq %rbp
retq
```

Post-return:

```
subq $16, %rsp
...
```



Exercise

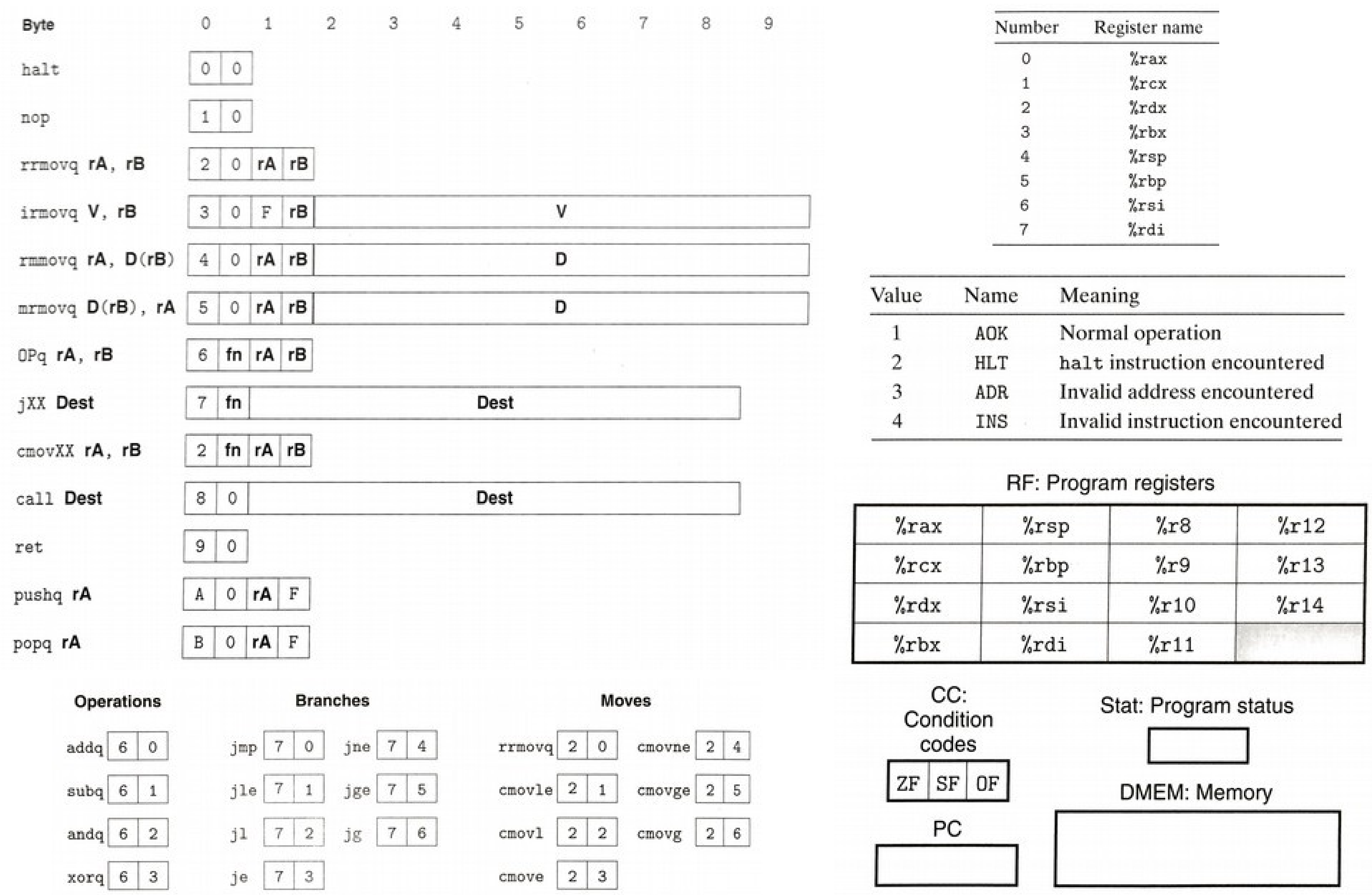
- Trace the following code--what is the value of `%rax` at the end?
  - Initial values: `%rdi = 100, %rsp = 0x7fffe820`

```
400540 <leaf>:
  400540  lea 0xf(%rdi), %rdi         # rdi = rdi + 15
  400544  retq

400545 <top>:
  400545  sub $0x5, %rdi              # rdi = rdi - 5
  400549  callq 400540 <leaf>
  40054e  add %rdi, %rdi              # rdi = rdi + rdi
  400551  retq

400550 <main>:
...
  40055b  callq 400545 <top>
  400560  mov %rdi, %rax              # rdx = rax
...
```

Aside: Y86-64 ISA



Byte	0	1	2	3	4	5	6	7	8	9
halt	0 0									
nop	1 0									
rrmovq rA, rB	2 0	rA rB								
irmovq V, rB	3 0	F rB	V							
rmmovq rA, D(rB)	4 0	rA rB	D							
mrmovq D(rB), rA	5 0	rA rB	D							
OPq rA, rB	6 fn	rA rB								
jXX Dest	7 fn	Dest								
cmovXX rA, rB	2 fn	rA rB								
call Dest	8 0	Dest								
ret	9 0									
pushq rA	A 0	rA F								
popq rA	B 0	rA F								

Operations

addq	6	0
subq	6	1
andq	6	2
xorq	6	3

Branches

jmp	7	0	jne	7	4
jle	7	1	jge	7	5
jl	7	2	jg	7	6
je	7	3			

Moves

rrmovq	2	0	cmovne	2	4
cmovle	2	1	cmovge	2	5
cmovl	2	2	cmovg	2	6
cmove	2	3			

Number	Register name
0	%rax
1	%rcx
2	%rdx
3	%rbx
4	%rsp
5	%rbp
6	%rsi
7	%rdi

Value	Name	Meaning
1	AOK	Normal operation
2	HLT	halt instruction encountered
3	ADR	Invalid address encountered
4	INS	Invalid instruction encountered

RF: Program registers

%rax	%rsp	%r8	%r12
%rcx	%rbp	%r9	%r13
%rdx	%rsi	%r10	%r14
%rbx	%rdi	%r11	

CC: Condition codes

ZF	SF	OF

Stat: Program status

PC

DMEM: Memory