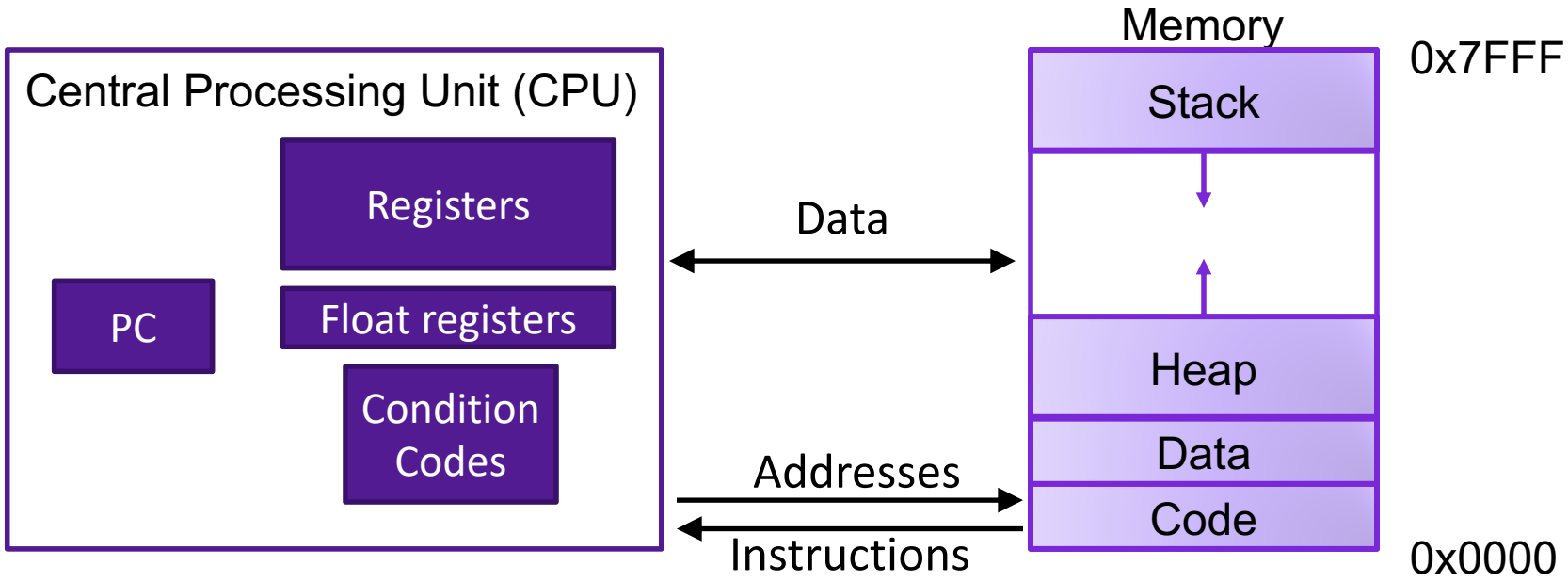
Lecture 6: Control Flow in Assembly

CS 105

Review: Assembly/Machine Code View



Programmer-Visible State

- PC: Program counter (%rip)
- Register file: 16 Registers
- Float registers
- Condition codes

Memory

- Byte addressable array
- Code and user data
- Stack to support procedures

Review: X86-64 Integer Registers

`%rax` (function result)	`%r8` (fifth argument)
`%rbx`	`%r9` (sixth argument)
`%rcx` (fourth argument)	`%r10`
`%rdx` (third argument)	`%r11`
`%rsi` (second argument)	`%r12`
`%rdi` (first argument)	`%r13`
`%rsp` (stack pointer)	`%r14`
`%rbp`	`%r15`

Review: Assembly Operations

- Transfer data between memory and register
  - Load data from memory into register
  - Store register data into memory
- Perform arithmetic function on register or memory data
- Transfer control
  - Conditional branches
  - Unconditional jumps to/from procedures

Review: Data Movement Instructions

- MOV source, dest

Moves data source->dest

```
dest = source
```

Suffixes

char	b	1
short	w	2
int	l	4
long	q	8
pointer	q	8

Review: Operand Forms

- Immediate:
  - Syntax: $Imm Value: Imm Example: $47
- Register:
  - Syntax: r Value: R[r] Example: %rbp
- Memory (Absolute):
  - Syntax: Imm Value: M[Imm] Example: 0x4050
- Memory (Indirect):
  - Syntax: (r) Value: M[R[r]] Example: (%rsp)
- Memory (Base+displacement):
  - Syntax: Imm(r) Value: M[Imm+R[r]] Example: 12(%rsp)
- Memory (Scaled indexed):
  - Syntax: Imm(r1, r2, s) Value: M[Imm+R[r1]+R[r2]*s] Example: 7(%rdx, %rdx, 4)

Review: Some Arithmetic Operations

- Two Operand Instructions:

Format		Computation	
`andq`	Src,Dest	Dest = Dest & Src	
`orq`	Src,Dest	Dest = Dest | Src	
`xorq`	Src,Dest	Dest = Dest ^ Src	
`shlq`	Src,Dest	Dest = Dest << Src	Also called salq
`shrq`	Src,Dest	Dest = Dest >> Src	Logical
`sarq`	Src,Dest	Dest = Dest >> Src	Arithmetic
`addq`	Src,Dest	Dest = Dest + Src	
`subq`	Src,Dest	Dest = Dest – Src	
`imulq`	Src,Dest	Dest = Dest * Src	

Suffixes

`char`	**`b`**	**`1`**
`short`	**`w`**	**`2`**
`int`	**`l`**	**`4`**
`long`	**`q`**	**`8`**
`pointer`	**`q`**	**`8`**

Exercise: Translating Assembly

```
arith:
  movq     %rdi, %rax
  addq     %rsi, %rax
  addq     %rdx, %rax
  movq     %rsi, %rdx
  salq     $3,   %rdx
  movq     $47,  %rcx
  addq     %rdx, %rcx
  imulq    %rcx, %rax
  ret
```

```
long arith(long x, long y,
           long z){




}
```

Interesting Instructions

- **`salq`**: shift
- **`imulq`**: multiplication
  - But, only used once

Register	Use(s)
`%rdi`	Argument **`x`**
`%rsi`	Argument **`y`**
`%rdx`	Argument **`z`**
`%rax`	return value

lea Instruction

Scaled Memory Operands

`movq (%rdi,%rsi,8), %rax`

```
void ex(long* xp, long* yp){

  long* p = xp + 8*yp;
  long ret = *p;

}
```

`leaq` Source, Dest

`leaq (%rdi,%rsi,8), %rax`

```
void ex(long xp, long yp){
  long ret = xp + 8*yp;
}
```

- pointer arithmetic
  - E.g., p = x + i;
- arithmetic
  - expressions x + k*y (k=1, 2, 4, 8)

Converted to ASM by compiler:

```
long m12(long x){

  return x*12;

}
```

```
leaq (%rdi,%rdi,2), %rax # ret <- x+x*2
salq $2, %rax            # return ret<<2
```

CONTROL FLOW

Jumps

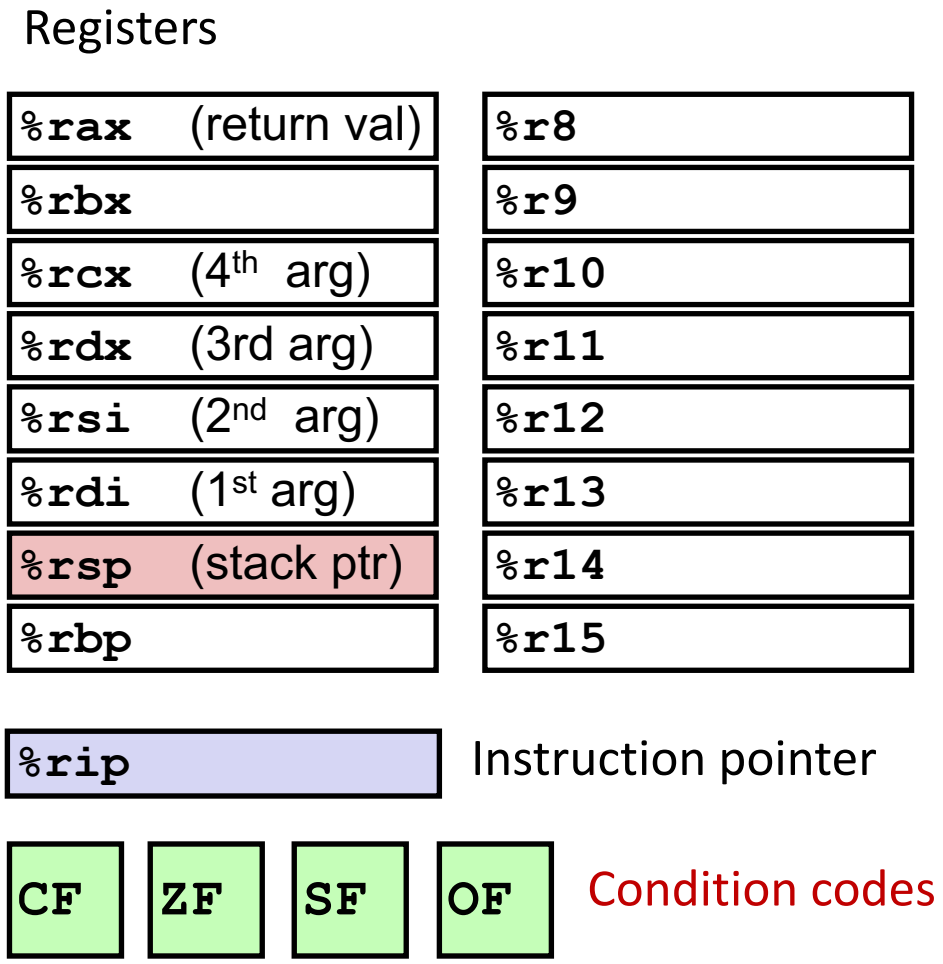
- A jump instruction can cause the execution to switch to a completely new position in the program (updates the program counter)
  - jmp Label
  - jmp *Operand

```
.L0:
  movq   $0, %rax
  jmp    .L1
  movq   (%rax), %rdx
.L1:
  movq   %rcx, %rax
```

```
jmp *%rax
```

Branches and Jumps

- Processor state (partial)
  - Temporary data (**%rax**, …)
  - Location of runtime stack (**%rsp**)
  - Location of current code control point (**%rip**, …)
  - Status of recent tests (CF, ZF, SF, OF)



Condition Codes

- Single bit registers
  - SF Sign Flag (for signed)
  - ZF Zero Flag
  - OF Overflow Flag (for signed)
- Implicitly set (as a side effect) by arithmetic operations and comparison operations
- Not set by **`leaq`** instruction

Condition Codes: `compare`

- Instruction **`cmp`** explicitly sets condition codes

- **`cmpq a,b`** like computing **`b-a`** without setting destination
  - ZF set if **`(b-a) == 0`**
  - SF set if **`(b-a) < 0`** (as signed)
  - CF set if carry out from most significant bit (used for unsigned comparisons)
  - OF set if two's-complement (signed) overflow

Condition Codes: `test`

- Instruction **`test`** explicitly sets condition codes
- **`testq a,b`** like computing **`a&b`** without setting destination
  - ZF set when **`a&b == 0`**
  - SF set when **`a&b < 0`**
- Test for zero: **`testq %rax, %rax`**

Jumping

- jX instructions
  - Jump to different part of code if condition is true

jX	Condition	Description
jmp	1	Unconditional
je	ZF	Equal / Zero
jne	~ZF	Not Equal / Not Zero
jl	(SF ^ OF)	Less (Signed)
jle	(SF ^ OF) | ZF	Less or Equal (Signed)
jg	~(SF ^ OF) & ~ZF	Greater (Signed)
jge	~(SF ^ OF)	Greater or Equal (Signed)

`cmpq a,b` like computing `b-a` without setting destination

Exercise 1: Conditional Jumps

- Consider each of the following segments of assembly code, and indicate whether or not the jump will occur. In all cases, assume that %rdi contains the value 47 and %rsi contains the value 13

1.
```
addq %rdi, %rsi
je .L0
```
2.
```
subq %rdi, %rsi
jge .L0
```
3.
```
cmpq %rdi, %rsi
jl .L0
```
4.
```
testq %rdi, %rdi
jne .L0
```

Conditional Branching

```
long absdiff(long x, long y){
  long result;

  if (x > y){
    result = x-y;
  } else {
    result = y-x;
  }

  return result;
}
```

```
absdiff:
    cmpq    %rsi, %rdi
    jle     .L4
    movq    %rdi, %rax
    subq    %rsi, %rax
    ret
.L4                 # x-y <= 0
    movq    %rsi, %rax
    subq    %rdi, %rax
    ret
```

Register	Use
%rdi	x
%rsi	y
%rax	result

Exercise 2: Conditionals

```
test:
  leaq (%rdi, %rsi), %rax
  addq %rdx, %rax
  cmpq $-3, %rdi
  jge .L2
  cmpq %rdx, %rsi
  jge .L3
  movq %rdi, %rax
  imulq %rsi, %rax
  ret
.L3:
  movq %rsi, %rax
  imulq %rdx, %rax
  ret
.L2
  cmpq $2, %rdi
  jle .L4
  movq %rdi, %rax
  imulq %rdx, %rax
.L4:
  rep; ret
```

```
long test(long x, long y, long z){
  long val = ___________;

  if(_______){

    if(_______){

      val = ______;


    } else {
      val = ______;

    }

 } else if (______){

    val = _______;

  }
  return val;
}
```

Reg	Use
%rdi	x
%rsi	y
%rdx	z
%rax	result