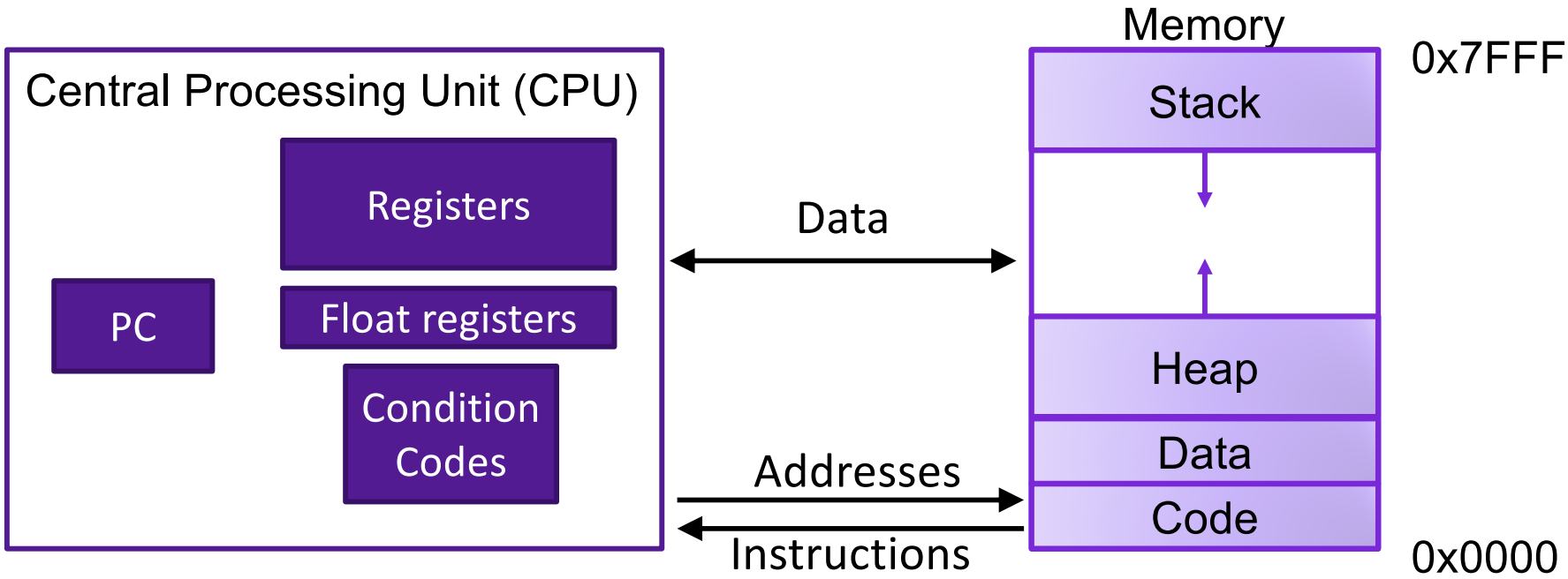
Lecture 6: Assembly Arithmetic and Control

CS 105

February 10, 2020

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

Assembly Characteristics: 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

ARITHMETIC IN ASSEMBLY

Some Arithmetic Operations

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

Suffixes

- 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	

Some Arithmetic Operations

- One Operand Instructions

notq	Dest = ~Dest
incq	Dest = Dest + 1
decq	Dest = Dest – 1
negq	Dest = – Dest

- See text for more instructions

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

Suffixes

Example: Assembly Operations

1. addq $0x47, %rax
2. addq %rbx, %rax
3. addq (%rbx), %rax
4. addq %rbx, (%rax)
5. addq 8(%rax,%rdi,8), %rax

- Also movq, subq, andq, …

Register	Value
`%rax`	`0x100`
`%rbx`	`0x108`
`%rdi`	`0x01`

Address	Value
`0x100`	`0x01234567`
`0x108`	`0x89abdef`
`0x110`	`0x00112233`

Arithmetic Exercise

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

```
arith2:
  orq     %rsi, %rdi
  sarq    $3, %rdi
  notq    %rdi
  movq    %rdx, %rax
  subq    %rdi, %rax
  ret
```

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




}
```

Interesting Instructions

- **sarq**: arithmetic right shift

Arithmetic Exercise

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

```
arith2:
  orq     %rsi, %rdi
  sarq    $3, %rdi
  notq    %rdi
  movq    %rdx, %rax
  subq    %rdi, %rax
  ret
```

```
long arith(long x, long y,
           long z){
  x = x | y;
  x = x >> 3;
  x = ~x;

  long ret = z - x;
  return ret

}
```

Interesting Instructions

- **sarq**: arithmetic right shift

Address Computation Instruction

- **`leaq`** Source, Dest
  - Source is address mode expression
  - Set Dest to address denoted by expression

- Uses
  - Computing addresses without a memory reference
    - E.g., translation of **`p = &x[i];`**
  - Computing arithmetic expressions of the form x + k*y
    - k = 1, 2, 4, or 8

- Example

```
long m12(long x)
{
  return x*12;
}
```

Converted to ASM by compiler:

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

Arithmetic Example

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

```
arith:
  leaq    (%rdi,%rsi), %rax
  addq    %rdx, %rax
  leaq    (%rsi,%rsi,2), %rdx
  salq    $4, %rdx
  leaq    4(%rdi,%rdx), %rcx
  imulq   %rcx, %rax
  ret
```

```
long arith(long x, long y,
           long z){
  long ret = x+y;
  ret = ret+z;

  z = y * 48;
  long temp = x + z + 4;
  ret = ret * temp;
  return ret;
}
```

Interesting Instructions

- **leaq**: address computation
- **salq**: shift
- **imulq**: multiplication
  - But, only used once

CONTROL FLOW

Jumps

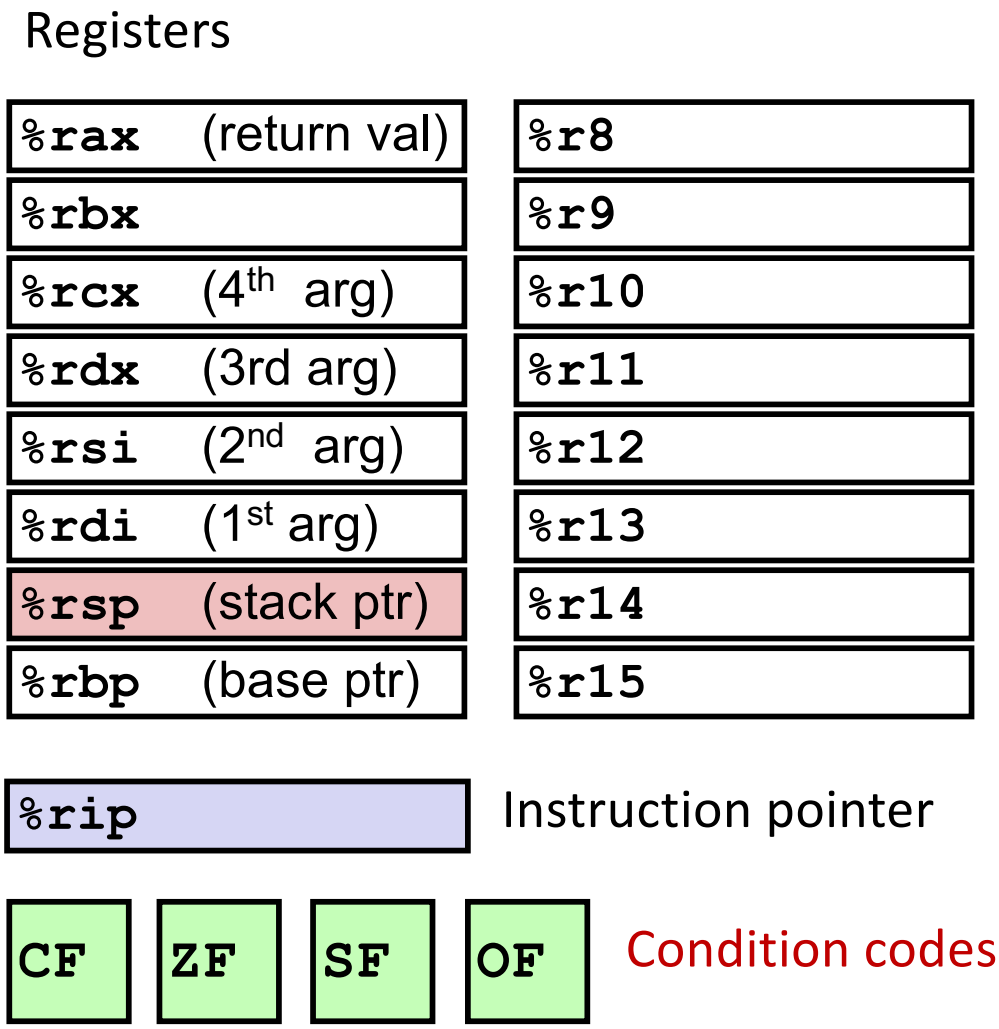
- A jump instruction can cause the execution to switch to a completely new position in the program (updates the program counter)
  - jmp Label
  - jmp *Operand

```
.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
  - CF Carry Flag (for unsigned)
  - 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
js	SF	Negative
jns	~SF	Nonnegative
jg	~(SF ^ OF) & ~ZF	Greater (Signed)
jge	~(SF ^ OF)	Greater or Equal (Signed)
jl	(SF ^ OF)	Less (Signed)
jle	(SF ^ OF) | ZF	Less or Equal (Signed)

Conditional Branching

Register	Use
%rdi	x
%rsi	y
%rax	return value

```
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
    movq    %rsi, %rax
    subq    %rdi, %rax
    ret
```

Exercise

```
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;
}
```

Exercise

```
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 = x + y + z;

  if(x < -3){

    if(y < z){

      val = x*y;


    } else {
      val = y*z;

    }

 } else if (x > 2){

    val = x*z;

  }
  return val;
}
```