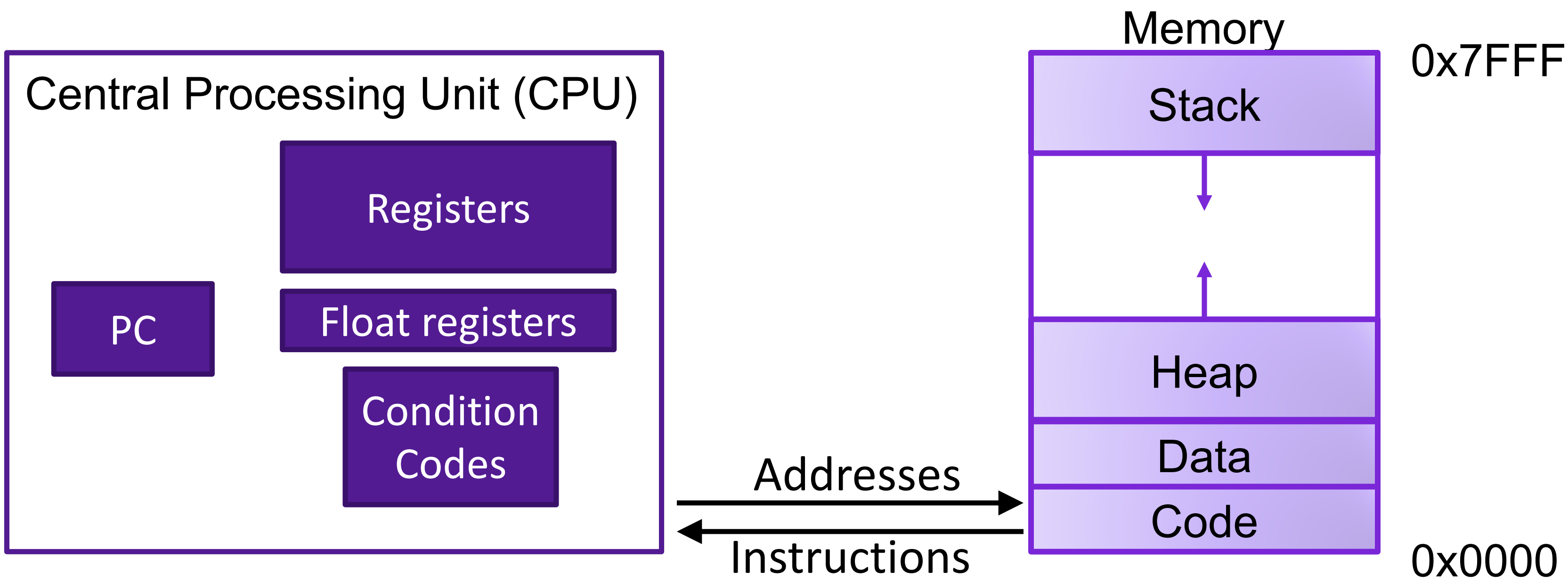
Lecture 6: Operations and Conditions in Assembly

CS 105 Fall 2023

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

ARITHMETIC IN ASSEMBLY

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

Some Arithmetic Operations

- One Operand Instructions

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

Suffixes

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

Exercise 1: Assembly Operations

Register	Value
`%rax`	`0x100`
`%rbx`	`0x110`
`%rdi`	`0x01`

Address	Value
`0x100`	`0x012`
`0x108`	`0x99a`
`0x110`	`0x809`

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

Sum	Location

Example: Translating Assembly

```
arith:
  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

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

Exercise 2: 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

- **`leaq`**: address computation
- **`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;

}
```

```
long m12(long x){

  return x*12;
}
```

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:

```
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
```

Conditional Jumps

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

jX	Description
jmp	Unconditional
je	Equal / Zero
jne	Not Equal / Not Zero
jl	Less (Signed)
jle	Less or Equal (Signed)
jg	Greater (Signed)
jge	Greater or Equal (Signed)

What condition are we evaluating?

Conditional Jumps

- Whether or not we jump depends on how the output of the last arithmetic operation compares to zero

```
movq $47, %rax
subq $13, %rax
jg .L2
```

jump

```
movq $47, %rax
subq $13, %rax
je .L2
```

no jump

- Not set by `lea` instruction

- Unless there's an explicit conditional evaluation more recently

Condition Evaluations

- **`cmp a,b`** like computing **`b-a`** without setting destination
- **`test a,b`** like computing **`a&b`** without setting destination
- Test for zero: **`test %rax, %rax`**

Exercise 3: 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 4: Conditionals

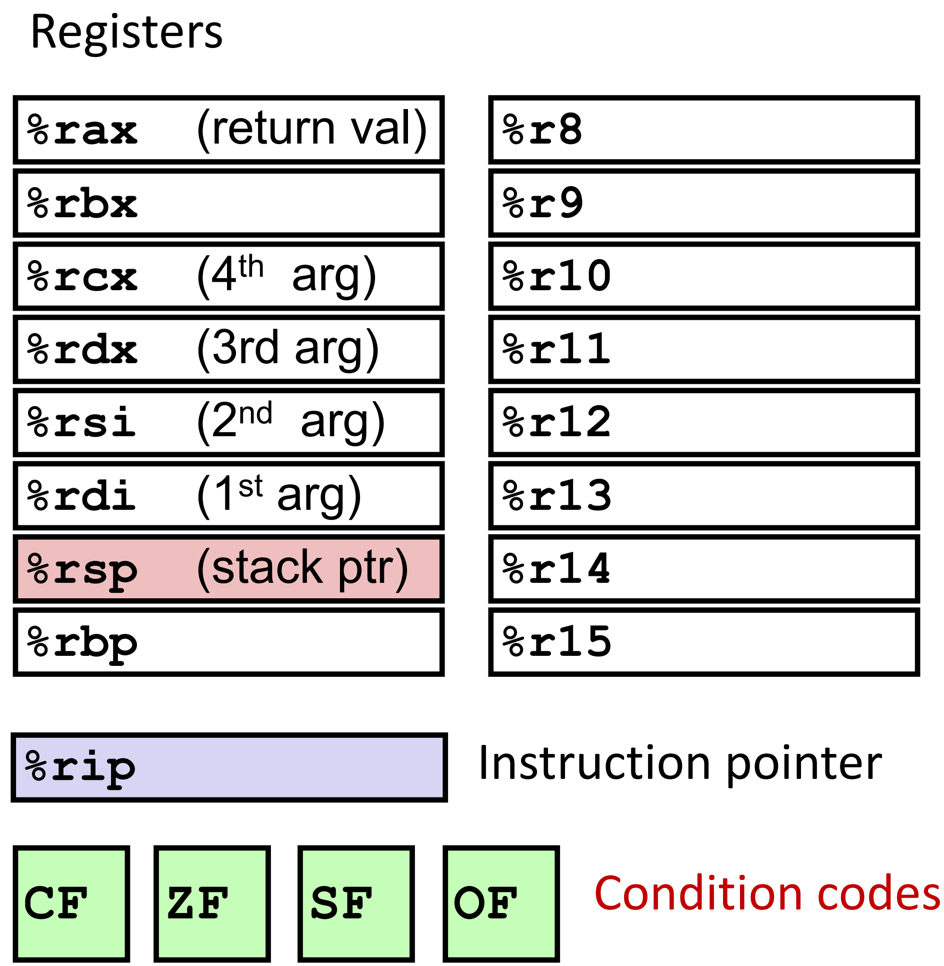
```
test:
  leaq (%rdi, %rsi), %rax
  addq %rdx, %rax
  cmpq $-3, %rdi
  jge .L2
  cmpq %rdx, %rsi
  jge .L3
  movq %rdi, %rax
  imulq %rsi, %rax
  ret
.L3:
  movq %rsi, %rax
  imulq %rdx, %rax
  ret
.L2
  cmpq $2, %rdi
  jle .L4
  movq %rdi, %rax
  imulq %rdx, %rax
.L4:
  rep; ret
```

```
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

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)
  - CF Carry Flag (for unsigned)
- Implicitly set (as a side effect) by arithmetic operations
- Explicitly set by **`cmp`** and **`test`**
- Not set by **`leaq`** instruction

Example 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

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)