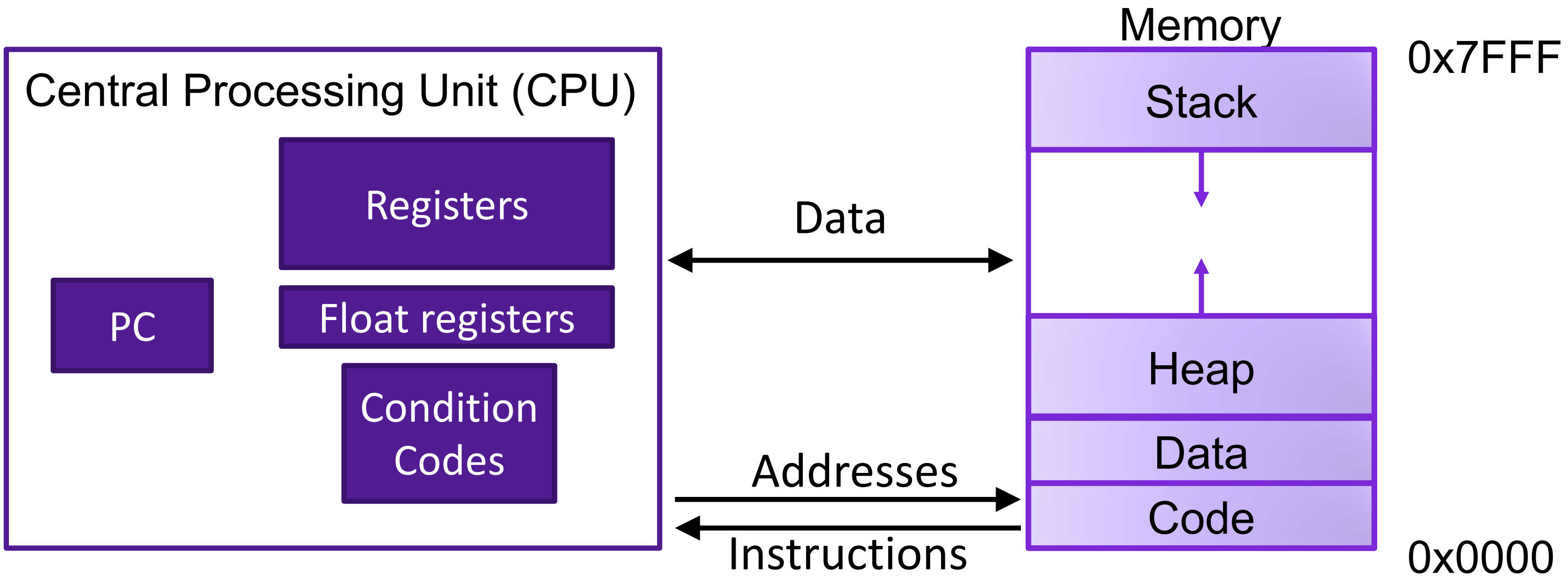
Lecture 6: Control Flow in Assembly

CS 105

Spring 2021

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

Jumps

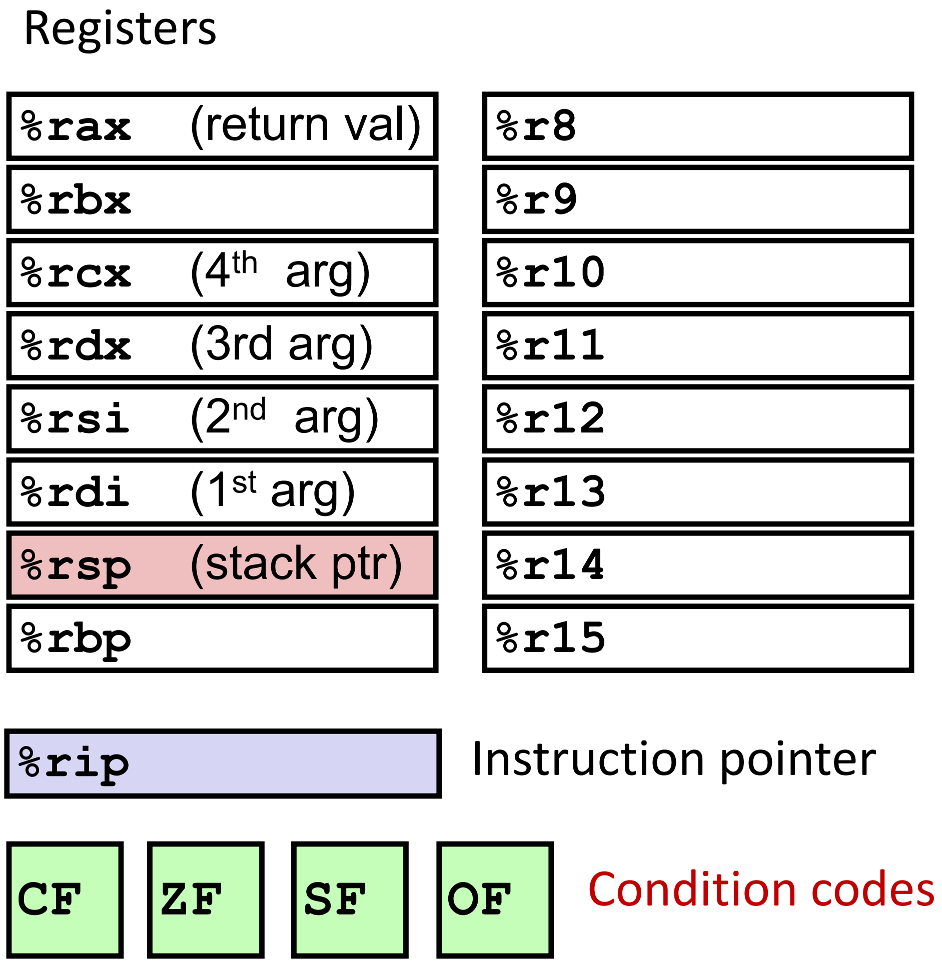
- A jump instruction can cause the execution to switch to a completely new position in the program (updates the program counter)
  - jmp Label
  - jmp *Operand

```
.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)



Registers

`%rax` (return val)	`%r8`
`%rbx`	`%r9`
`%rcx` (4th arg)	`%r10`
`%rdx` (3rd arg)	`%r11`
`%rsi` (2nd arg)	`%r12`
`%rdi` (1st arg)	`%r13`
`%rsp` (stack ptr)	`%r14`
`%rbp`	`%r15`

`%rip` Instruction pointer

CF ZF SF OF Condition codes

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
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
   ```

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
```
$13 + 47 \overset{?}{==} 0$ **no jump**

2.
```
subq %rdi, %rsi
jge .L0
```
$13 - 47 \overset{?}{\geq} 0$ **no jump**

3.
```
cmpq %rdi, %rsi
jl .L0
```
$13 - 47 \overset{?}{<} 0$ **jump**

4.
```
testq %rdi, %rdi
jne .L0
```
$13 \,\&\, 13 \,! \overset{?}{=} 0$ **jump**

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;
}
```

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
```

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

```
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;
}
```

Loops

- All use conditions and jumps
  - do-while
  - while
  - for

Do-while Loops

```
long bitcount(unsigned long x){
  long result = 0;
  do {
    result += x & 0x1;
    x >>= 1;
  } while (x);
  return result;
}
```

```
long bitcount(unsigned long x){
  long result = 0;
loop:
  result += x & 0x1;
  x >>= 1;
  if(x) goto loop;
  return result;
}
```

```
   movq    $0, %rax    #  result = 0
.L2:                   #  loop:
   movq    %rdi, %rdx
   andq    $1, %rdx    #  t = x & 0x1
   addq    %rdx, %rax  #  result += t
   shrq    %rdi, $1    #  x >>= 1
   jne     .L2         #  if (x) goto loop
   rep; ret
```

Register	Use(s)
%rdi	x
%rax	result

While Loops

```
while (Condition) {
    Body
}
```

```
if (Condition){
    do {
      Body
    } while (Condition)
}
```

```
long bitcount(unsigned long x) {
  long result = 0;
  while (x) {
    result += x & 0x1;
    x >>= 1;
  }
  return result;
}
```

```
       movq   $0, %rax
       jmp    .L2
.L3:
       movq   %rdi, %rdx
       andq   $1, %rdx
       addq   %rdx, %rax
       shrq   %rdi, $1
.L2:
       testq  %rdi, %rdi
       jne    .L3
       rep ret
```

Register	Use(s)
%rdi	x
%rax	result

For loops

Register	Use(s)
%rdi	Argument **x**
%rax	result

```
for (Init; Cond; Incr){
      Body
}
```

```
Init;
while (Cond) {
      Body;
      Incr;
}
```

Initial test can often be optimized away:

```
   for (j = 0; j < 99; j++)
```

```
long bitcount(unsigned long x) {
  long result;
  for (result = 0; x; x >>= 1)
    result += x & 0x1;
  return result;
}
```

```
        movq   $0, %rax
        jmp    .L2
.L3:
        movq   %rdi, %rdx
        andq   $1, %rdx
        addq   %rdx, %rax
        shrq   %rdi, $1
.L2:
        testq  %rdi, %rdi
        jne    .L3
        rep ret
```

Register	Use(s)
%rdi	x
%rax	result

Exercise 3: Loops

```
loop:
  movq $0, %rax
  movq $0, %rdx
  jmp L1
L0:
  addq %rdx, %rax
  incq %rdx
L1:
  cmp %rdi, %rdx
  jl L0
  ret
```

```
long loop(long val){
  long ret = ______;
  long i;

  for(i = ___; ________; ____){

    ret = _______;


  }

  return ret;
}
```

Exercise 3: Loops

```
loop:
  movq $0, %rax
  movq $0, %rdx       # init
  jmp L1
L0:
  addq %rdx, %rax
  incq %rdx           # update
L1:
  cmp %rdi, %rdx
  jl L0               # condition
  ret
```

Register	Use(s)
`%rdi`	Argument `val`
`%rdx`	Local `i`
`%rax`	Local `ret`

```
long loop(long val){
  long ret =   0   ;
  long i;

  for(i =  0 ; i < val ; i++ ){

    ret = ret + i;

  }

  return ret;
}
```

Exercise 4: Feedback

1. Rate how well you think this recorded lecture worked
    1. Better than an in-person class
    2. About as well as an in-person class
    3. Less well than an in-person class, but you still learned something
    4. Total waste of time, you didn't learn anything

2. How much time did you spend on this video lecture (including time spent on exercises)?

3. Do you have any comments or feedback?