CS 24: Introduction to Computing Systems

x86-64 Reference

`mov` and `lea` Instructions

x86-64		C Pseudocode
`mov %src, %dst`	↔	`%dst = %src`
`mov %src, c(%dst)`	↔	`*(%dst + c) = %src`
`mov c(%src), %dst`	↔	`%dst = *(%src + c)`
`mov $src, c(%dst)`	↔	`*(%dst + c) = $src`
`mov c(%b, %i, $s), %dst`	↔	`%dst = *(%b + %i*$s + c)`
`lea c(%b, %i, $s), %dst`	↔	`%dst = %b + %i*$s + c`

Arithmetic Instructions

x86-64		C Pseudocode
`add %src, %dst`	↔	`%dst += %src`
`sub %src, %dst`	↔	`%dst -= %src`
`imul %src, %dst`	↔	`%dst *= %src`
`idiv %denom`	↔	`%rax = (%rdx:%rax) / (%denom)`
		`%rdx = (%rdx:%rax) % (%denom)`
`xor %src, %dst`	↔	`%dst ^= %src`
`and %src, %dst`	↔	`%dst &= %src`
`or %src, %dst`	↔	`%dst |= %src`
`shl $by, %dst`	↔	`%dst <<= $by`
`shr $by, %dst`	↔	`%dst = ((unsigned) %dst) >> $by`
`sar $by, %dst`	↔	`%dst = ((signed) %dst) >> $by`
`sar %dst`	↔	`%dst = ((signed) %dst) >> 1`

"Cast" Instructions

x86-64		C Pseudocode
`movzbl (%src),%dst`	↔	`%dst = (uint32_t)*(uint8_t*)%src`
`movzwl (%src),%dst`	↔	`%dst = (uint32_t)*(uint16_t*)%src`
`movsbq (%src),%dst`	↔	`%dst = (int64_t)*(int8_t*)%src`
`movslq (%src),%dst`	↔	`%dst = (int64_t)*(int32_t*)%src`
`cltq`	↔	`%rax = (int64_t)%eax`
`cqto`	↔	`%rdx:%rax = (int128_t)%rax`

Stack Instructions

x86-64		C Pseudocode
`push %src`	↔	`%rsp -= sizeof(%src); *(%rsp) = %src`
`pop %dst`	↔	`%dst = *(%rsp); %rsp += sizeof(%dst)`
`ret`	↔	`%rip = *(%rsp); %rsp += 8`
`repz ret`	↔	`%rip = *(%rsp); %rsp += 8`
`call addr`	↔	`%rsp -= 8; *(%rsp) = %rip + c; %rip = addr`

Control Flow

The processor has a special register that contains "flags" which **test** sets.

```
test %r1, %r2
```

- ZF set to result of `(%r1 & %r2) == 0`
- SF set to result of `(%r1 & %r2) < 0`

The processor has a special register that contains "flags" which **cmp** sets.

```
cmp %r1, %r2
```

- ZF set to result of `(%r2 - %r1) == 0`
- SF set to result of `(%r2 - %r1) < 0`
- CF set to result of "there is an *unsigned* carry out when computing `%r2 - %r1`"
- OF set to result of "there is a *signed* overflow when computing `%r2 - %r1`"

The processor has a special register that contains "flags" which *arithmetic instructions* implicitly set.

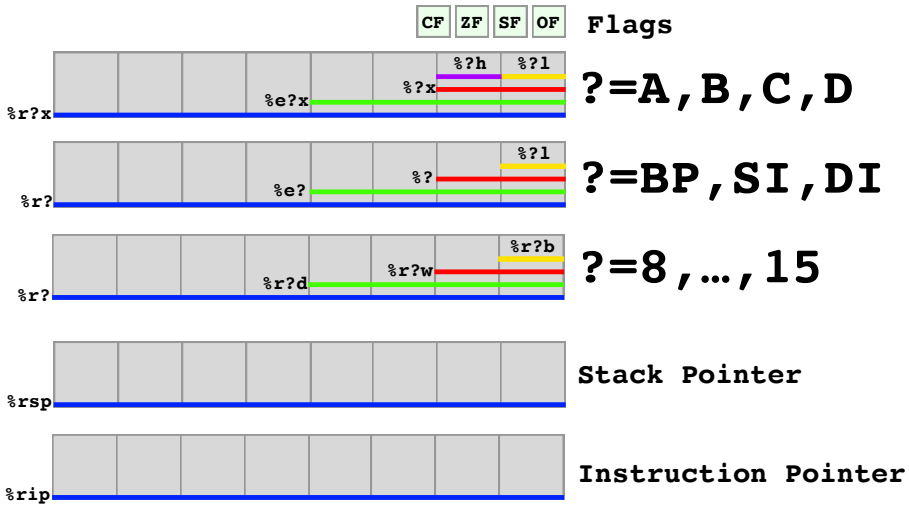
```
add %r1, %r2
```

- ZF set to result of `(%r2 + %r1) == 0`
- SF set to result of `(%r2 + %r1) < 0`
- CF set to result of "there is an *unsigned* carry out when computing `%r2 + %r1`"
- OF set to result of "there is a *signed* overflow when computing `%r2 + %r1`"

Suffix		Flag To Check		What It Means
___e	↔	ZF	↔	Zero
___ne	↔	∼ZF	↔	Not Zero
___s	↔	SF	↔	Negative
___ns	↔	∼SF	↔	Non-negative
___g	↔	∼(SF ^ OF) & ∼ZF	↔	Greater (signed)
___ge	↔	∼(SF ^ OF)	↔	Greater or Equal (signed)
___l	↔	(SF ^ OF)	↔	Less (signed)
___le	↔	(SF ^ OF) | ZF	↔	Less or Equal (signed)
___a	↔	∼CF & ∼ZF	↔	Above (unsigned)
___b	↔	CF	↔	Below (unsigned)

x86-64		C Pseudocode
set__ %r	↔	%r = FLAG
cmov__ %src, %dst	↔	%dst = %src **if** FLAG
j__ *addr*	↔	%rip = *addr* **if** FLAG
jmp *addr*	↔	%rip = *addr*

Registers and x86-64 System V ABI




```
Return:       %rax
Arguments:    %rdi, %rsi, %rdx, %rcx, %r8, %r9
Caller-saved: Arguments, %r10, %r11
Callee-saved: %rbx, %rbp, %r12, %r13, %r14, %r15
```

Stack and x86-64 System V ABI