Y86 ALU Operations

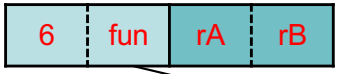
- Topic
  - Arithmetic and logical operations of the y86
- Learning Objectives
  - Define ALU
  - Describe all the y86 ALU operations
  - Describe how each ALU operation affects each condition code
  - Read/Write simple y86 programs using MOV and ALU operations.
- Reading
  - 4.1.3 (plus blue boxes)

Arithmetic and Logical Instructions

%rax	%rsp	%r8	%r12
%rcx	%rbp	%r9	%r13
%rdx	%rsi	%r10	%r14
%rbx	%rdi	%r11	

```
R[%rB] <- R[%rB] <fun> R[%rA]
```

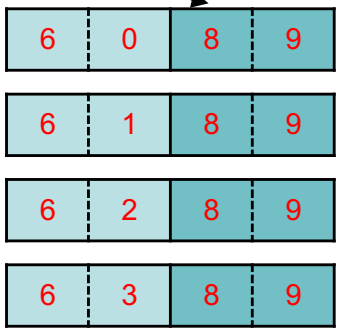
2-byte instruction




```
OPQ   %rA, %rB

ADDQ %r8, %r9
SUBQ %r8, %r9
ANDQ %r8, %r9
XORQ %r8, %r9
```

In-Memory



Bonus ALU Operations

- While the y86 in the text supports only ADDQ, SUBQ, ANDQ, and XORQ, CPSC 313 has three bonus ALU operations:
  - MULQ (fun = 4) :
    - MULQ %r8, %r9 # R[%r9] <- R[%r9] * R[%r8]
  - DIVQ (fun = 5)
    - DIVQ %r8, %r9 # R[%r9] <- R[%r9] / R[%r8]
  - MODQ (fun = 6)
    - MODQ %r8, %r9 # R[%r9] <- R[%r9] % R[%r8]

Condition Codes

- Recall: There are three condition codes
  - ZF: Zero – set if last operation produced 0
  - SF: Sign – set if last operation produced a number < 0
  - OF: Overflow – set if last arithmetic operation produced a 2's complement overflow (the operands had the same sign, but the result has a different sign)
- These are set on every ALU operation (and are then used to control condition instructions).

CC Examples

- Let's figure out how each of these operations would set the condition codes.

```
irmovq  0x1, %rax
irmovq  0x2, %rbx
addq    %rax, %rbx      # rbx = 1 + 2 = 3
subq    %rax, %rbx      # rbx = 3 – 1 = 2
subq    %rbx, %rax      # rax = 1 – 2 = 0xFF…F
xorq    %rax, %rax      # rax = 0
xorq    %rax, %rax      # Still 0
subq    %rbx, %rax      # rax = 0 – 2 = 0xFFFF...E
andq    %rax, %rax      # rax = unchanged

addq    %rax, %rax      # rax = -2 + -2 = 0xFFF...C
addq    %rax, %rbx      # rbx = 2 + -4 = 0xFFF…E
addq    %rbx, %rax      # rax = -2 + -4 = 0xFFF..A

irmovq  0x1, %rax       # rax = 1
addq    %rbx, %rax      # rax = 0xFFFF…F
irmovq  0x3, %rax       # rax = 3
addq    %rax, %rbx      # rbx = 0
# Generate Overflow
irmovq 0x7FFFFFFFFFFFFFFF, %rax$
addq %rax, %rax$
```