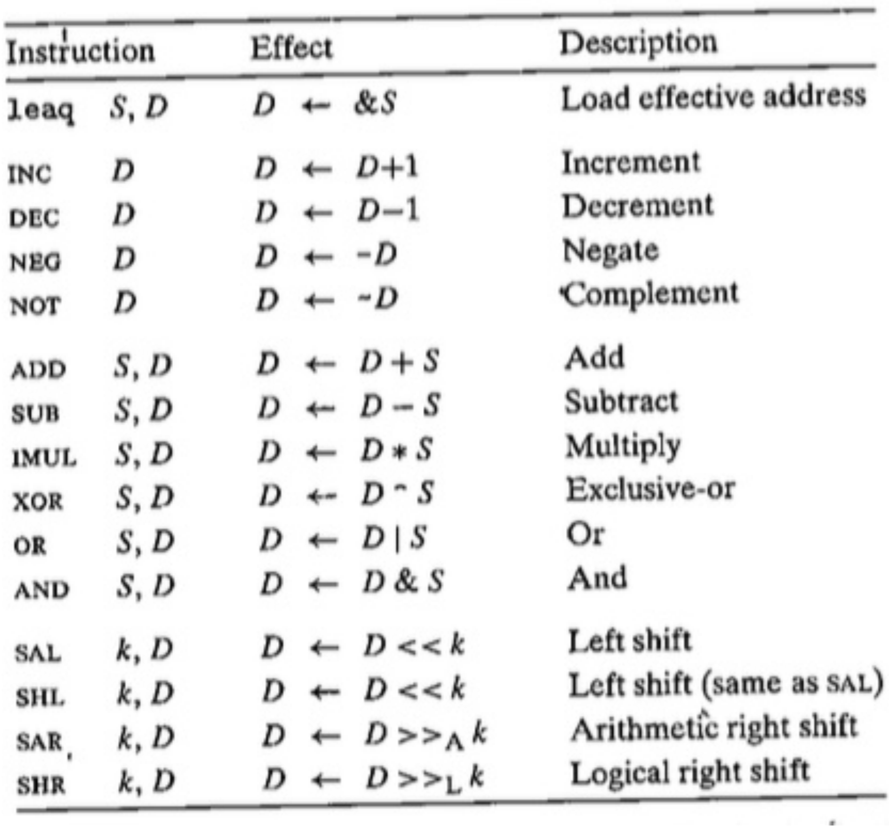
Chapter 3.5: Arithmetic & Logical Operations



Instruction		Effect	Description
`leaq`	S, D	$D \leftarrow \&S$	Load effective address
INC	D	$D \leftarrow D+1$	Increment
DEC	D	$D \leftarrow D-1$	Decrement
NEG	D	$D \leftarrow -D$	Negate
NOT	D	$D \leftarrow \sim D$	Complement
ADD	S, D	$D \leftarrow D + S$	Add
SUB	S, D	$D \leftarrow D - S$	Subtract
IMUL	S, D	$D \leftarrow D * S$	Multiply
XOR	S, D	$D \leftarrow D \wedge S$	Exclusive-or
OR	S, D	$D \leftarrow D \mid S$	Or
AND	S, D	$D \leftarrow D \,\&\, S$	And
SAL	k, D	$D \leftarrow D << k$	Left shift
SHL	k, D	$D \leftarrow D << k$	Left shift (same as SAL)
SAR	k, D	$D \leftarrow D >>_A k$	Arithmetic right shift
SHR	k, D	$D \leftarrow D >>_L k$	Logical right shift

- each instruction class shown has instructions for operating on each of the four different sizes of data (byte, word, double word, quad)
- operations are divided into four groups
    - load effective address
    - unary
        - have one operand
    - binary
        - have two operands
    - shifts

3.5.1: Load Effective Address (`leaq`)

- actually a variant of the `movq` instruction
    - copies the address of the source operand into the destination
    - `D = &S`
- can be used to generate pointers for later memory references
- can be used to compactly describe common arithmetic operations
- if `%rdx` contains the value x

    - `leaq 7(%rdx,%rdx,4),%rax` will set `%rax` to 5x + 7
- destination operand **must be** a register

Practice Problem 3.6

Suppose register `%rax` holds value x and `%rcx` holds value y. Fill in the table below with formulas indicating the value that will be stored in register `%rdx` for each of the given assembly-code instructions:

Instruction	Result
`leaq 6(%rax), %rdx`	6 + x
`leaq (%rax,%rcx), %rdx`	x + y
`leaq (%rax,%rcx,4), %rdx`	x + 4y
`leaq 7(%rax,%rax,8), %rdx`	7 + 9x
`leaq 0xA(,%rcx,4), %rdx`	10 + 4y
`leaq 9(%rax,%rcx,2), %rdx`	9 + x + 2y

Clever `leaq` example

```c
long scale(long x, long y, long z) {
    long t = x + 4 * y + 12 * z;
    return t;
}
```

```
scale:
    leaq    (%rdi,%rsi,4), %rax     ; x + 4y
    leaq    (%rdx,%rdx,2), %rdx     ; z + 2z = 3z
    leaq    (%rax,%rdx,4), %rax     ; (x+4y) + 4(3z) = x + 4y + 12z
    ret
```

3.5.2: Unary and Binary Operations

- operations in the second group are unary operations, with a single operand as both the **source** and the **destination**
- Ex: `incq (%rsp)` causes the 8-byte element on top of the stack to be incremented
    - similar to C's increment (++) and decrement (—) operators

- the third group consists of binary operations, where the second operand is both a **source** and a **destination**
    - similar to C's assignment operators, such as `x -= y`
- **CAUTION:** Source argument is given first
    - subtraction (`subq(%rax,%rdx)`), decrements 2nd arg (`%rdx`) by 1st arg (`%rax`)

    - Read as “Subtract `%rax` from `%rdx`“
- first argument may be **immediate**, **register**, or **memory location**
- second argument may be **register** or **memory location**

Practice Problem 3.8

Assume the following values are stored at the indicated memory addresses and registers:

Address	Value	Register	Value
0x100	0xFF	%rax	0x100
0x108	0xAB	%rcx	0x1
0x110	0x13	%rdx	0x3
0x118	0x11		

Fill in the following table showing the effects of the following instructions, in terms of both the register or memory location that will be updated and the resulting value.

Instruction	Destination	Value
`addq %rcx,(%rax)`	0x100	0x100
`subq %rdx,8(%rax)`	0x108	0xA8
`imulq $16,(%rax,%rdx,8)`	0x118	0x110
`incq 16(%rax)`	0x110	0x14
`decq %rcx`	%rcx	0x0
`subq %rdx,%rax`	%rax	0xFD

3.5.3: Shift Operations

- final group is shift operations
- shift amount given first, value to shift given second
    - shift amount given as an immediate or with the single-byte register `%cl`
- both arithmetic and logical right shifts are possible
- `SAL` and `SHL` do the same thing, shift left logically
- `SAR` shifts arithmetically to the right, `SHR` performs a logical shift
- if only one (register) operand is given, assume it to be shifted by 1