a | b = ~(~a & ~b)

a ^ b = (a & ~b) | (~a & b)

**float and double**:

Shape, rectangle

Description automatically generatedShape, rectangle

Description automatically generated

**Normalized value** (exp ≠ 000…0 and exp ≠ 111…1)

E = Exp – Bias

Exp: unsigned value of exp field

Bias = , k = # of exponent bits. Single precision: 127 (Exp: 1…254, E: -126…127); Double precision: 1023 (Exp: 1…2046, E: -1022…1023)

Frac = 1.xxx…x

**Denormalized Value** (exp = 000…0)

Exponent value: E = 1 – Bias (instead of E = 0 – Bias because smallest normalized value has Exp = 1, **equispaced**)

Frac = 0.xxx…x2

exp = 000…0, frac = 000…0 represents zero. There are also –0

**Infinity**: exp = 111…1, frac = 000…0

**NaN**: exp = 111…1, frac ≠ 000…0

**Round To Even**

**Text

Description automatically generated with low confidence**

**Table

Description automatically generated with medium confidence**

**x86-64 linux calling convention:**

Integer parameters:

%rdi, %rsi, %rdx, %rcx, %r8 and %r9

Others are stored in stack, pushed in reversed (right-to-left) order

long myfunc(long a, long b, long c, long d, long e, long f, long g, long h)

{

long xx;

long yy;

long zz;

// ...

}

Chart

Description automatically generated

b = 1 byte, w = 2 bytes, l = 4 bytes, q = 8 bytes

**CF** Carry Flag (for unsigned) **SF** Sign Flag (for signed) **ZF** Zero Flag **OF** Overflow Flag (for signed)

Implicitly set by arithmetic operations (**but not set by leaq instruction)**

addq Src DestDest (t = a + b)

**CF** set if carry out from most significant bit (unsigned overflow)

**ZF** set if t == 0

**SF** set if t < 0 (as signed)

**OF** set if two’s complement (signed) overflow

**Rules for turning on the carry flag**

1. The carry flag is set if the addition of two numbers causes a carry out of the most significant bits added.

1111 + 0001 = 0000 (carry flag is turned on)

2. The carry (borrow) flag is also set if the subtraction of two numbers requires a borrow into the most significant (leftmost) bits subtracted

0000 - 0001 = 1111 (carry flag is turned on)

**Rules for turning on the overflow flag**

1. If the sum of two numbers with the sign bits off yields a result number with the sign bit on

0100 + 0100 = 1000 (overflow flag is turned on)

2. If the sum of two numbers with the sign bits on yields a result number with the sign bit off

1000 + 1000 = 0000 (overflow flag is turned on)

**Note that different from above (1111 + 0001 = 0000), the result is correct even though CF is set**

In unsigned arithmetic, use the carry flag

In signed arithmetic, use the overflow flag

**cmp Instruction**

cmp b, a

Computes *b - a* (just like sub). Sets condition codes based on result, but **does not change *b***

**test instruction**

test a, b

Computes 𝑏 ∧ 𝑎 just like and. Sets condition codes (only SF and ZF) based on result, but **does not change 𝒃**

Most common use: test x, x

to compare x to zero

Table

Description automatically generated

**movzbl**: zero-extend, byte -> long. **movslq**: sign-extend, long -> quad. Etc.

A picture containing diagram

Description automatically generated

**Buffer overflow attacks**

Stack Smashing Attacks: overwrite normal return address A with address of some other code S. When Q executes ret, will jump to other code

Code Injection Attacks: input string contains byte representation of executable code, overwrite return address A with address of buffer B, when Q executes ret, will jump to exploit code

**Measures**

Avoid overflow vulnerabilities: strcpy -> strncpy

Employ system-level protections: Randomized stack offsets, Nonexecutable code

segments

Have compiler use stack canaries

**Return-Oriented Programming Attacks**

Work around stack randomization and marking stack nonexecutable

Does not overcome stack canaries

Diagram

Description automatically generatedGraphical user interface, text, application

Description automatically generated Graphical user interface, text, application, email

Description automatically generated