**Floating Point** For + infinity, the bits of exp are 1, all else is 0, for –infinity the sign bit and all exp are 1, rest are 0. V = (-1)^s x M x 2^E

|  |  |  |  |
| --- | --- | --- | --- |
| CASE | DESC | M | E |
| 1 | Exp is neither all 0’s nor all 1’s | 1 + f (fraction bits unsigned) | e-bias |
| 2 | When exp is all 0’s | f | 1 – bias |
| 3 | Exp is all 1’s | If frac = 0, val is +- infinity, when frac is non 0, value is NaN | |

Bias = 2^(k-1) -1 (k is the # of bits of exp field); e = unsigned number represented by exp field. f goes like ½ + 1/8 + 1/16 …

Smallest denormalized > = 1/(2^(# of fraction bits). Largest normalized > 0 = 1 + (2^f – 1 / 2^5) i.e. 1 + 255/256

loopy:

pushl %ebp

movl $2, %eax

movl %esp, %ebp

movl 8(%ebp), %edx

movl 12(%ebp), %ecx

.L2:

subl $1, %edx

sall $2, %eax

cmpl %ecx, %edx

jb .L2

popl %ebp

ret

unsigned loopy (unsigned x, unsigned y)

{

int result = 2;

do {

result = result\*4;

x = x-1;

} while (x < y);

return result;

}

Type promotion (goes up towards more general)

Long double > double > float > unsigned long > long > unsigned int > int > unsigned short > short > char

Representing numbers:

-1 in signed binary is 11111111… flip the bits & add 1 to convert to and from 2’s comp For n bit 2’s complement representation:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TMin | Tmax | Tmax – Tmin | Tnin – Tmax | -2 is | -TMin |
| -2^n | 2^n -1 | -1 | 1 | -64 + 32 + 16 + 8 + 4 +2 | TMin |

|  |  |
| --- | --- |
| Instruction | Does |
| Compl scrc2, src1 | Sets cc to srcs1 – src2 |
| Testl src2, src1 | Sets ccs to src1 & src2 |
| movl Src, Dest | Dest = Src |
| leal Src, Dest | Dest = address of Src |
| jb label | jump below (unsigned) |
| jge label | jump greater or equal (signed) |
| push Src | %esp = %esp – 4, Mem[%esp] = Src |
| pop Dest | Dest = Mem[%esp], %esp = %esp + 4 |
| call label | push address of next instruction, jmp label |

Addressing modes

• Immediate

$val Val

val: constant integer value

movl $17, %eax

• Normal

(R) Mem[Reg[R]]

R: register R specifies memory address

movl (%ecx), %eax

• Displacement

D(R) Mem[Reg[R]+D]

R: register specifies start of memory region

D: constant displacement D specifies offset

movl 8(%ebp), %edx

• Indexed

D(Rb,Ri,S) Mem[Reg[Rb]+S\*Reg[Ri]+D]

D: constant displacement 1, 2, or 4 bytes

Rb: base register: any of 8 integer registers

Ri: index register: any, except %esp

S: scale: 1, 2, 4, or 8

movl 0x100(%ecx,%eax,4), %edx

Struct Alignment

struct s1{

char c1;

int i1, i2;

char c2; };

size of (s1): 16 bytes

Offset c1: 0, c2, 12, i1: 4, i2:8

struct s2{

double d1, d1;

float f1, f2;

short s2;

char c3;

};

Sizeof(s2): 28

Offset: d1:0, d2:8, f1:16, f2:20, s2:24, c3:26

Loop Unrolling.

double inner\_prod (double a1[ ], double a2 [ ], int N) {

int i;

double sum = 0.0;

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

sum+= a1[i ] + \* a2 [ i ];

}

return sum; }

double inner\_prod\_unroll (double a1[ ], double a2 [ ], int N) {

int i;

double sum = 0, sum1 = 0;

int limit = N – N%2;

for (i = 0; i < limit ; i+=2) {

sum0 += a1[i ] \* a2 [ i ];

sum1 += a1[i + 1 ] \* a2[i +1];

}

for ( ; i <N; i++)

{

sum0+= a1[i ] \* a2 [ i ];

}

return sum0 + sum1; }

**Caches:**

* Cache size is B\* E \* S

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S = 2^s | # of sets |  | m = log2(M) | # of physical addr bits |
| E | # of lines per set |  | s = log2(S) | # of set index bits |
| B = 2^b | Block Size (bytes) |  | b = log2(B) | # of block offset bits |
| M = 2^m | Max # of unique mem addrss |  | t = m – (s + b | # of tag bits |

Memory address of m bits is divided into 3 parts. tag, cache set index and offset. Cache set index has 2 bits and offset has b bits. [t0 t1 t2 t3 t4 s0 s21 o0 o1]