

Chapter 2

Floating Point Numbers

Computer Application

Flaxer Eli - ComputerAppl

Ch 2 - 1

Floating Point Numbers

- Computer representation inspired by scientific notation
 - Examples
 - 3.15576×10^9
 - 1.001101×2^4
 - Floating point representation encodes
 - Sign
 - Exponent
 - Significand

Flaxer Eli - ComputerAppl

Ch 2 - 2

IEEE-754 representation

Single Precision - 32 bits



1 bit 8 bits 25 bits

Double Precision - 64 bits



1 bit 11 bits 32 bits

Flaxer Eli - ComputerAppl

Ch 2 - 3

IEEE-754 fields

- **Sign** bit: 1 if -ve
 - **Exponent:** The value of exponent is 8 bit two's compliment offset by 127 (-126 to +127 for single precision). Special exponent values 0 and 255 used to code for 0, Nan and Infinity.
 - **Significand:** Leading bit is implicit - significand field contains the bits *after* the binary point.

Flaxer Eli - ComputerAppl

Ch 2 - 4

Examples

- $0.75 = -0.11 = -1.1 \times 2^{-1}$
 - IEEE-754 single precision representation
 - 1011111101000000000000000000000000000000
 - $1.00 = 1.0 \times 2^0$
 - 00111111000000000000000000000000
 - $52.0 = -110100 = -1.101 \times 2^5$
 - 11000010010100000000000000000000

Flaxer Eli - ComputerAppl

Ch 2 - 5

Elaboration (float)

<u>Exponent</u>	<u>Significand</u>	<u>Object represented</u>
0	0	0
1-254	anything	floating-point number
255	0	infinity
255	nonzero	Nan
0	nonzero	denormalized number

Flaxer Eli - ComputerAppl

Ch 2 - 6

Why Offset

- To keep the order in integer form.
- If $(x > y) \Leftrightarrow (\text{int})x > (\text{int})y$.
- Comparing integer is faster and simpler.
- Example:

With Offset 2.0 = 01000000000000000000000000000000
 0.5 = 00111110000000000000000000000000

No Offset 2.0 = 00000001000000000000000000000000
 0.5 = 01111110000000000000000000000000

Flaxer Eli - ComputerAppl

Ch 2 - 7

C Examples

```
#include <conio.h>
#include <stdio.h>
union dami {
    float f;
    long l;
} tt[4];
void main()
{
    FILE *fp;
    fp=fopen ("xxx.txt", "w");
    clrscr();
    tt[0].f = 1.0;
    tt[1].f = -1.0;
    tt[2].f = 0.0;
    tt[3].f = 1.25;
    fprintf(fp,"%08lx\n", tt[0].l);
    fprintf(fp,"%08lx\n", tt[1].l);
    fprintf(fp,"%08lx\n", tt[2].l);
    fprintf(fp,"%08lx\n", tt[3].l);
}
```

Result:
 3f800000
 bf800000
 00000000
 3fa00000

Ch 2 - 8

Floating Point Addition

Start

1. Compare the exponents of the two numbers.
 Shift the smaller number to the right until its exponent would match the larger exponent

2. Add the significands

3. Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent

Overflow or underflow?

Exception

4. Round the significand to the appropriate number of bits

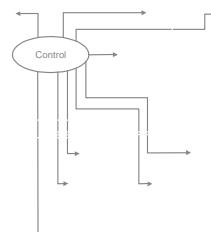
Still normalized?

Done

Ch 2 - 9

Flaxer Eli - ComputerAppl

Floating Point Adder



Flaxer Eli - ComputerAppl

Ch 2 - 10

Floating Point Multiplication

Start

1. Add the biased exponents of the two numbers, incrementing the sum to get the new biased exponent

2. Multiply the significands

3. Normalize the product if necessary, shifting it right and incrementing the exponent

Overflow or underflow?

Exception

4. Round the significand to the appropriate number of bits

Still normalized?

5. Set the sign of the product to positive if the signs of the original operands are the same; if they differ make the sign negative

Done

Ch 2 - 11

Flaxer Eli - ComputerAppl

IEEE-754 Rounding

- Two extra bits, guard and round, are maintained in intermediate results to do rounding properly.
- Rounding options
 - Truncation
 - Round up
 - Round Down
 - Round to nearest

Flaxer Eli - ComputerAppl

Ch 2 - 12

Finite Precision Arithmetic

- IEEE-754 only captures a finite number of members of the infinite set of reals
- Floating point operations produce approximate, not exact results this can have profound consequences when you want to perform a long sequence of arithmetic operations. Egs weather prediction, nuclear weapons simulations.