CS101 Introduction to computing

Floating Point Numbers

A. Sahu and S. V. Rao
Dept of Comp. Sc. & Engg.
Indian Institute of Technology Guwahati
Outline

• Need to floating point number
• Number representation : IEEE 754
• Floating point range
• Floating point density
  – Accuracy
• Arithmetic and Logical Operation on FP
• Conversions and type casting in C
Need to go beyond integers

- integer 7
- rational 5/8
- real $\sqrt{3}$
- complex 2 - 3i

Extremely large and small values:
- distance Pluto - sun = $5.9 \times 10^{12}$ m
- mass of electron = $9.1 \times 10^{-28}$ gm
Representing fractions

- Integer pairs (for rational numbers)
  
  \[
  \frac{5}{8} = 5/8
  \]

- Strings with explicit decimal point
  
  \[-247.009\]

- Implicit point at a fixed position
  
  \[010011010110001011\]

- Floating point
  
  \[\text{fraction} \times \text{base}^{\text{power}}\]
Numbers with binary point

\[ 101.11 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + . + 1 \times 2^{-1} + 1 \times 2^{-2} \]

\[ = 4 + 1 + . + 0.5 + 0.25 = 5.75_{10} \]

0.6 = \(0.1001100110011001.....\)

\[ .6 \times 2 = 1 + .2 \]

\[ .2 \times 2 = 0 + .4 \]

\[ .4 \times 2 = 0 + .8 \]

\[ .8 \times 2 = 1 + .6 \]
Numeric Data Type

• char, short, int, long int
  - char : 8 bit number (1 byte = 1B)
  - short: 16 bit number (2 byte)
  - int : 32 bit number (4B)
  - long int : 64 bit number (8B)

• float, double, long double
  - float : 32 bit number (4B)
  - double : 64 bit number (8B)
  - long double : 128 bit number (16B)
Numeric Data Type

- unsigned char
- char
- unsigned short
- short

Unsigned int

int
Numeric Data Type

• char, short, int, long int
  – We have: Signed and unsigned version
  – char (8 bit)
    • char: -128 to 127, we have +0 and -0 😊 😊 Fun
    • unsigned char: 0 to 255
  – int: \(-2^{31}\) to \(2^{31}-1\)
  – unsigned int: 0 to \(2^{32}-1\)

• float, double, long double
  – For fractional, real number data
  – All these numbered are signed and get stored in different format
Numeric Data Type

**Float**
- Sign bit
- Exponent
- Mantissa

**Double**
- Sign bit
- Exponent
- Mantissa-1
- Mantissa-2
FP numbers with base = 10

\((-1)^S \times F \times 10^E\)

S = Sign
F = Fraction (fixed point number)
    usually called **Mantissa** or **Significand**
E = Exponent (positive or negative integer)

- Example  \(5.9 \times 10^{12}, -2.6 \times 10^3, 9.1 \times 10^{-28}\)
- Only one non-zero digit left to the point
FP numbers with base = 2

\((-1)^S \times F \times 2^E\)

S = Sign
F = Fraction (fixed point number)
    usually called Mantissa or Significand
E = Exponent (positive or negative integer)

• How to divide a word into S, F and E?
• How to represent S, F and E?

Example  1.0101x2^{12},  -1.11012x10^3  1.101 \times 2^{-18}

Only one non-zero digit left to the point: default it will be 1 incase of binary
• So no need to store this
IEEE 754 standard

- **Single precision numbers**

  1 8 23
  0 1011 0101 1101 0110 1011 0001 0110 110

  S E F

- **Double precision numbers**

  1 11 20+32
  0 1011 0101 111 1101 0110 1011 0001 0110

  S E F

  1011 0001 0110 1100 1011 0101 1101 0110
Representing F in IEEE 754

- Single precision numbers
  
  23
  
  1. 110101101011000101101101
  
  F

- Double precision numbers
  
  20+32
  
  1. 101101011000101101101
  
  F

  101100010110110010110101110101101

Only one non-zero digit left to the point: default it will be 1 incase of binary. So no need to store this bit
Value Range for F

- Single precision numbers
  \[ 1 \leq F \leq 2 - 2^{-23} \quad \text{or} \quad 1 \leq F < 2 \]

- Double precision numbers
  \[ 1 \leq F \leq 2 - 2^{-52} \quad \text{or} \quad 1 \leq F < 2 \]

These are “normalized”.
Representing E in IEEE 754

- Single precision numbers
  - 8
  - 10110101
  - E bias 127

- Double precision numbers
  - 11
  - 10110101110
  - E bias 1023
Floating point values

• $E = E' - 127$, $V = (-1)^s \times 1.M \times 2^{E' - 127}$

• $V = 1.1101... \times 2^{(40 - 127)} = 1.1101.. \times 2^{-87}$

- Single precision numbers

<table>
<thead>
<tr>
<th>S</th>
<th>0</th>
<th>E'</th>
<th>0010 1000</th>
<th>1101 0110 1011 0001 0110 110</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Floating point values

- $E = E' - 127$, $V = (-1)^s \times 1.M \times 2^{E' - 127}$

- $V = -1.1 \times 2^{(126 - 127)} = -1.1 \times 2^{-1} = -0.11 \times 2^0$
  
  $$= -0.11 = -\frac{11}{2^2} = -\frac{3}{4} = -0.75_{10}$$

- **Single precision numbers**

```plaintext
1 0111 1110 1000 0000 0000 0000 0000 000
```

- $S$ = 1
- $E'$ = 0111 1110
- $F$ = 1000 0000 0000 0000 0000 000
Value Range for E

■ Single precision numbers
  -126 ≤ E ≤ 127
  (all 0’s and all 1’s have special meanings)

■ Double precision numbers
  -1022 ≤ E ≤ 1023
  (all 0’s and all 1’s have special meanings)
Floating point demo applet on the web

- [https://www.h-schmidt.net/FloatConverter/IEEE754.html](https://www.h-schmidt.net/FloatConverter/IEEE754.html)

- Google “Float applet” to get the above link
Overflow and underflow

largest positive/negative number (SP) = \[\pm (2 - 2^{-23}) \times 2^{127} \approx \pm 2 \times 10^{38}\]

smallest positive/negative number (SP) = \[\pm 1 \times 2^{-126} \approx \pm 2 \times 10^{-38}\]

Largest positive/negative number (DP) = \[\pm (2 - 2^{-52}) \times 2^{1023} \approx \pm 2 \times 10^{308}\]

Smallest positive/negative number (DP) = \[\pm 1 \times 2^{-1022} \approx \pm 2 \times 10^{-308}\]
Density of int vs float

Int : 32 bit

Float : 32 bit

- Exponent
- Mantissa

- Number of number can be represented
  - Both the cases (float, int) : $2^{32}$
- Range
  - int (-$2^{31}$ to $2^{31}$-1)
  - float Large $\pm (2 - 2^{-23}) \times 2^{127}$ Small $\pm 1 \times 2^{-126}$
- 50% of float numbers are Small (less then $\pm 1$ )
Density of Floating Points

• 256 Persons in Room of Capacity 256  (Range)
  8 bit integer : 256/256 = 1

• 256 person in Room of Capacity  200000 (Range)
  – 1<sup>st</sup> Row should be filled with 128 person
  – 50% number with negative power are -1 < N > +1

• Density of Floating point number is
  – Dense towards 0
Expressible Numbers (int and float)

Expressible integers

- overflow 0 + overflow

-2^{31} 0 2^{31}-1

Expressible Float

- overflow 0 + overflow

(1-2^{-24})x2^{128} -0.5x2^{-127} 0.5x2^{-127} (1-2^{-24})x2^{128}

- underflow + underflow

- overflow + overflow
Distribution of Values

- 6-bit IEEE-like format
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is 3

![Distribution of Values Diagram]
Distribution of Values (close-up view)

- 6-bit IEEE-like format
  - $e = 3$ exponent bits
  - $f = 2$ fraction bits
  - Bias is 3
Density of 32 bit float SP

- Fraction/mantissa is 23 bit
- Number of different number can be stored for particular value of exponent
  - Assume for \( \text{exp}=1 \), \( 2^{23}=8\times1024\times1024 \approx 8\times10^6 \)
  - Between 1-2 we can store \( 8\times10^6 \) numbers
- Similarly
  - for \( \text{exp}=2 \), between 2-4, \( 8\times10^6 \) number of number can be stored
  - for \( \text{exp}=3 \), between 4-8, \( 8\times10^6 \) number of number can be stored
  - for \( \text{exp}=4 \), between 8-16, \( 8\times10^6 \) number of number can be stored
Density of 32 bit float SP

• Similarly
  – for exp=23, between $2^{22}-2^{23}$, $8 \times 10^6$ number of number can be stored
  – for exp=24, between $2^{23}-2^{24}$, $8 \times 10^6$ number of number can be stored
  – for exp=25, between $2^{24}-2^{25}$, $8 \times 10^6$ number of number can be stored
    • $2^{24}-2^{25} > 8 \times 10^6$
  – ...
  – for exp=127, between $2^{126}-2^{127}$, $8 \times 10^6$ number of number can be stored
Density of 32 bit float SP

- $2^{23} = 8 \times 10^{24} \times 10^{24} \approx 8 \times 10^6$
Numbers in float format

• largest positive/negative number (SP) =
  \( \pm (2 - 2^{-23}) \times 2^{127} \approx \pm 2 \times 10^{38} \)

Second largest number:
  \( \pm (2 - 2^{-22}) \times 2^{127} \)

Difference Largest FP - 2\text{nd} largest FP
  \( = (2^{-23} - 2^{-22}) \times 2^{127} = 2 \times 2^{105} = 2 \times 10^{32} \)

Smallest positive/negative number (SP) =
  \( \pm 1 \times 2^{-126} \approx \pm 2 \times 10^{-38} \)
Addition/Sub of Floating Point

Step I: Align Exponents

Step 2: Add Mantissas

Step 3: Normalize
Floating point operations: ADD

• Add/subtract \( A = A_1 \pm A_2 \)
\[
[(-1)^{S_1} \times F_1 \times 2^{E_1}] \pm [(-1)^{S_2} \times F_2 \times 2^{E_2}]
\]
suppose \( E_1 > E_2 \), then we can write it as
\[
[(-1)^{S_1} \times F_1 \times 2^{E_1}] \pm [(-1)^{S_2} \times F_2' \times 2^{E_1}]
\]
where \( F_2' = F_2 / 2^{E_1-E_2} \),

The result is
\[
(-1)^{S_1} \times (F_1 \pm F_2') \times 2^{E_1}
\]
It may need to be normalized

\[
\text{Examples:}
\begin{align*}
3.2 \times 10^8 \pm 2.8 \times 10^6 \\
320 \times 10^6 \pm 2.8 \times 10^6 \\
322.8 \times 10^6 \\
3.228 \times 10^8
\end{align*}
\]
Testing Associatively with FP

• \( X = -1.5 \times 10^{38}, \ Y = 1.5 \times 10^{38}, \ z = 1000.0 \)

• \( X + (Y + Z) = -1.5 \times 10^{38} + (1.5 \times 10^{38} + 1000.0) \)
  \( = -1.5 \times 10^{38} + 1.5 \times 10^{38} \)
  \( = 0 \)

• \( (X+Y)+Z = (-1.5 \times 10^{38} + 1.5 \times 10^{38}) + 1000.0 \)
  \( = 0.0 + 1000.0 \)
  \( = 1000 \)
Multiply Floating Point

Step 1: Multiply Mantissas

Step 2: Add Exponents

Step 3: Normalize

For 32 bit SP Float: one 23 bit multiplication and 8 bit addition
For 32 bit int: one 32 bit multiplication

Above example: 3.2x5.8 is simpler, also 6+8 is also simpler as compared to 32 bit multiplication
Floating point operations

• Multiply

\[((-1)^{S_1} \times F_1 \times 2^{E_1}) \times ((-1)^{S_2} \times F_2 \times 2^{E_2})\]

\[= (-1)^{S_1 \oplus S_2} \times (F_1 \times F_2) \times 2^{E_1 + E_2}\]

Since \(1 \leq (F_1 \times F_2) < 4\),
the result may need to be normalized

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3.2 \times 10^8) X (5.8 \times 10^6)</td>
<td>(1.856 \times 10^{15})</td>
</tr>
<tr>
<td>(3.2 \times 5.8 \times 10^8 \times 10^6)</td>
<td>(18.56 \times 10^{14})</td>
</tr>
</tbody>
</table>
Floating point operations

• Divide

\[ (-1)^{S_1} \times F_1 \times 2^{E_1} \div (-1)^{S_2} \times F_2 \times 2^{E_2} \]

\[ = (-1)^{S_1 \oplus S_2} \times (F_1 \div F_2) \times 2^{E_1 - E_2} \]

Since \(.5 < (F_1 \div F_2) < 2\), the result may need to be normalized

(assume \(F_2 \neq 0\))
Float and double

• Float: single precision floating point
• Double: Double precision floating point
• Floating points operation are slower
  – But not in newer PC 😊😊
• Double operation are even slower
Floating point Comparison

• Three phases
• Phase I: Compare sign (give result)
• Phase II: If (sign of both numbers are same)
  – Compare exponents and give result
  – 90% of case it fall in this categories
  – Faster as compare to integer comparison:
    Require only 8 bit comparison for float and 11 bit for double (Example: sorting of float numbers)
• Phase III: If (both sign and exponents are same)
  – compare fraction/mantissa
Storing and Printing Floating Point

```c
float x=145.0, y;
y=sqrt(sqrt((x)));
x=(y*y) * (y*y);
printf("\nx=%f", x);
```

Many Round off cause loss of accuracy

```
x=145.0000015
```

Value stored in x is not exactly same as 1.0/3.0

```c
float x=1.0/3.0;
if ( x==1.0/3.0)  
    printf("YES");
else
    printf("NO");
```

One is before round of and other (stored x) is after round of

NO
Storing and Printing Floating Point

```c
float a = 34359243.5366233;
float b = 3.5366233;
float c = 0.00000212363;
printf("a=%8.6f, b=%8.6f, c=%8.12f\n", a, b, c);
```

Big number with small fraction cannot be combined

a = 34359243.000000
b = 3.5366233
C = 0.000002123630
Storing and Printing Floating Point

//15 S digits to store
float a=34359243.5366233;
//8 S digits to store
float b=3.5366233;
//6 S digits to store
float c=0.00000212363;

Thumb rule: 8 to 9 significant digits of a number can be stored in a 32 bit number
Thanks