Floating point representation

CPU uses binary number system for computation.

We can represent whole numbers in binary number system without any complication.

Example:

**111111111111111111111111111111112**

**1\*231 +1\*230+1\*229+1\*228+1\*227+1\*226+1\*225+1\*224+1\*223+1\*222+1\*221+1\*220+1\*219+1\*218+1\*217+1\*216+1\*215+1\*214+1\*213+1\*212+1\*211+1\*210+1\*29+1\*28+1\*27+1\*26+1\*25+1\*24+1\*23+1\*22+1\*21+1\*20**

2147483648+1073741824+536870912+268435456+134217728+67108864+33554432+16777216+8388608+4194304+2097152+1048576+524288+262144+131072+65536+32768+16384+8192+4096+2048+1024+512+256+128+64+32+16+8+4+2+1 = **429496729510**

**111111111111111111111111111111112 = 429496729510**

What about fraction numbers ?

**How do we can represent fraction in binary number system ?**

**FRACTION IN DECIMAL:**

**12345.73343 = ( 1\*104 ) + ( 2\*103 ) + ( 3\*102 ) + ( 4\*101 ) + ( 5\*100 )+ ( 7 \* 10-1 ) + ( 3 \* 10-2) + ( 3 \* 10-3) + ( 4 \* 10-4 ) + ( 3 \* 10-5 )**

**10000 - 1000 - 100 - 10 - 1 - 1/10 - 1/100 - 1/1000 - 1/10000**

**FRACTION IN BINARY – FIXED POINT:**

**1110111111.1111 = ( 1 \* 29 ) + ( 1 \* 28 ) + ( 1 \* 27 ) + ( 0 \* 26 ) + ( 1 \* 25 ) + ( 1 \* 24 ) + ( 1 \* 23 ) + ( 1 \* 22 ) + ( 1 \* 21 ) + ( 1 \* 20 ) + ( 1 \* 2-1 ) + ( 1 \* 2-2 ) + ( 1 \* 2-3 )**

**+ ( 1 \* 2-4 )**

**= 959 + 0.5+0.25+0.125+0.0625**

**In fixed point binary representation there are fixed number of bits for whole part and fractional part.**

**For example if we have 16 bits for whole number and another 16 bits for fractional numbers we can only represent 216 whole numbers and 2-16 fractional numbers. The range is too small.**

**In floating point representation fraction is represented in scientific notation.so we can represent very small values as well as very big values.**

**In scientific notation we have base , exponent , mantissa.**

**Using the same bit in fixed point representation we can represent more values in scientific notation.**

**+----+ +----+----+----+----+----+----+----+----+ +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+**

**| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |**

**| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |**

**+----+ +----+----+----+----+----+----+----+----+ +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+**

**s e e e e e e e e f f f f f f f f f f f f f f f f f f f f f f f**

**Sign:**

**floating point number uses sign magnitude instead of two’s complement.**

**The first bit tells whether the number is positive or negative.**

**Exponent:**

**Exponent can be positive or negative. So we have to do something to represent negative numbers.**

**In floating point representation exponents are represented in offset binary or Excess-K notation.**

**You may ask why we do we use Excess-K notation instead of two’s complement.**

**In Excess-k notation numbers are stored as unsigned integers. As we store signed numbers as unsigned numbers we can**

**Easily compare two just by seeing the ON bits. We can’t do it in two’s complement.**

**So Excess-k notation is used in floating point representation.**

**In this notation negative numbers are converted to positive numbers.**

**To convert normal exponent to offset binary notation we have to add bias value.**

**Bias K = 2n-1-1**

**N – number of bits we have to represent EXPONENT.**

**Lest assume we have 4 bit exponent**

**K = 24-1-1 = 7**

**We have to add K with the actual exponent ( exponent is assumed to be in two’s complement if it is negative ).**

**How Biased exponent is getting stored in floating point representation.**

**0000 – reserved**

**0001- -6**

**0010 - -5**

**0011- -4**

**0100 - -3**

**0101 - -2**

**0110 - -1**

**0111 - 0**

**1000 - 1**

**1001 - 2**

**1010 - 3**

**1011 - 4**

**1100 - 5**

**1101 - 6**

**1110 - 7**

**1111 – reserved**

**Mantissa:**

**We have 23 bits to represent to represent Mantissa. An implicit 1 is assumed in Mantissa part. So we don’t need to represent it in 23 bits. Mantissa can be represented in two forms.**

**Normalized - normalized numbers have implicit 1 in mantissa**

**1.xxxx xxxx xxxx xxxx xxxx xxx**

**smallest number we can represent:**

**1.0 \* 2-126**

**biggest number we can represent:**

**1.9999998807909 \* 2127**

**Denormalized – denormalized numbers have implicit 0 in mantissa**

**0.xxxx xxxx xxxx xxxx xxxx xxx**

**smallest number we can represent:**

**0.000000119209289 \* 2-126**

**biggest number we can represent:**

**0.99999988079071 \* 2-126**

**Arithmetic on floating point**

**We may have negative floating point number. So we have to follow scheme for representing negative numbers. There are four schemes for**

**representing negative numbers.**

1. **One’s complement**
2. **Two’s complement**
3. **Sign magnitude**
4. **Biased K notation**

**IEEE floating point standard uses sign magnitude for representing negative numbers.**

**So while doing arithmetic on floating point numbers we have to follow sign and magnitude arithmetic algorithm.**

**Sign magnitude representation of negative numbers.**

**0000 – 0 1000 – (-0)**

**0001 – 1 1001 – (-1)**

**0010 – 2 1010 – (-2)**

**0011 – 3 1011 – (-3)**

**0100 – 4 1100 – (-4)**

**0101 – 5 1101 – (-5)**

**0110 – 6 1110 – (-6)**

**0111 – 7 1111 – (-7)**

**operation add-magnitude a>b a<b a=b**

**----------------------------------------------------------------------------------------------------**

**(+a)+(+b) + (a + b)**

**-----------------------------------------------------------------------------------------------------**

**(+a)+(-b) +(a-b) -(b-a) +(a-b)**

**----------------------------------------------------------------------------------------------------**

**(-a)+(+b) -(a-b) +(b-a) +(a-b)**

**----------------------------------------------------------------------------------------------------**

**(-a)+(-b) - (a + b)**

**----------------------------------------------------------------------------------------------------**

**(+a)-(+b) +(a-b) -(b-a) +(a-b)**

**----------------------------------------------------------------------------------------------------**

**(+a)-(-b) + (a + b)**

**----------------------------------------------------------------------------------------------------**

**(-a)-(+b) - (a + b)**

**----------------------------------------------------------------------------------------------------**

**(-a)-(-b) -(a-b) +(b-a) +(a-b)**

**-----------------------------------------------------------------------------------------------------**

**Sign 0ne’s**

**Magnitude Complement unsigned**

**0000 – 0 0000 – 0 0000 - 0**

**0001 – 1 0001 – 1 0001 – 1**

**0010 – 2 0010 – 2 0010 – 2**

**0011 – 3 0011 – 3 0011 – 3**

**0100 – 4 0100 – 4 0100 – 4**

**0101 – 5 0101 – 5 0101 – 5**

**0110 – 6 0110 – 6 0110 – 6**

**0111 – 7 0111 – 7 0111 – 7**

**1000 – (-0) 1000 – (-8 1000 – 8**

**1001 – (-1) 1001 – (-7 ) 1001 – 9**

**1010 - (-2) 1010 – (-6) 1010 – 10**

**1011 - (-3) 1011 – (-5) 1011 – 11**

**1100 – (-4) 1100 – (-4) 1100 – 12**

**1101 – (-5) 1101 – (-3) 1101 – 13**

**1110 – (-6) 1110 – (-2) 1110 – 14**

**1111 – (-7) 1111 – (-1) 1111 – 15**

**(+1) + (-1) computation using two’s complement (+1)+(-1) computation using sign magnitude**

**0001 - 1 (+1) 0001 - 1 (+1)**

**1111 - 15 (-1) 1001 - 9 (-1)**

**--------------------- ----------------------**

**10000 - 16 (0) 1010 - 10 (-2)**

**(-2) + (+3) computation using two’s complement (-2)+(+3) computation using sign magnitude**

**1110 – 14 (-2) 1010 – 10 (-2)**

**0011 – 3 (+3) 0011 – 3 (+3)**

**----------------------- ------------------------**

**10001 – 17 (1) 1101 - 13 (-3)**

**As we can see above normal binary arithmetic won’t work in sign and magnitude. So we need to follow some algorithms to get the correct results from arithmetic that uses sign and magnitude representation for negative numbers.**

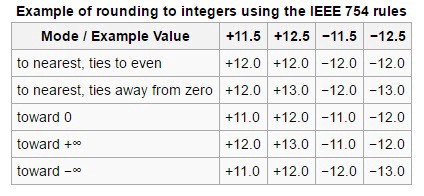
**Floating point addition and subtraction:**

**To add floating point numbers , exponents of the two numbers must be same.**

**So before adding two numbers , we must shift some bits to make the exponents same.**

**We must adjust the smaller exponent to the higher exponent.**

**ROUNDING:**

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **S** | **E** | **E** | **E** | **F** | **F** | **F** | **F** |

**S = 0 POSITIVE**

**S = 1 NEGATIVE**

**E – EXPONENT**

**BIAS K = 2n-1-1**

**= 23-1-1**

**= 22-1**

**K = 3**

**000 – 0 - reserved**

**001 – 1 - -2**

**010 – 2 - -1**

**011 – 3 - 0**

**100 – 4 – 1**

**101 – 5 - 2**

**110 – 6 - 3**

**111 – 7 - reserved**

**1.0000**

**1.0001**

**1.0010**

**1.0011**

**1.0100**

**1.0101**

**1.0110**

**1.0111**

**1.1000**

**1.1001**

**1.1010**

**1.1011**

**1.1100**

**1.1101**

**1.1110**

**1.1111**

**NORMALIZED:**

**Smallest number we can represent: 1.0 \* 2-2 = 0.001**

**Biggest number we can represent: 1.0 \* 2+3 = 1000**

**DENORMALIZED:**

**Smallest number we can represent: 0.0001 \* 2-2 = 0.000001**

**Biggest number we can represent: 0.1111 \* 2-2 = 0.001111**