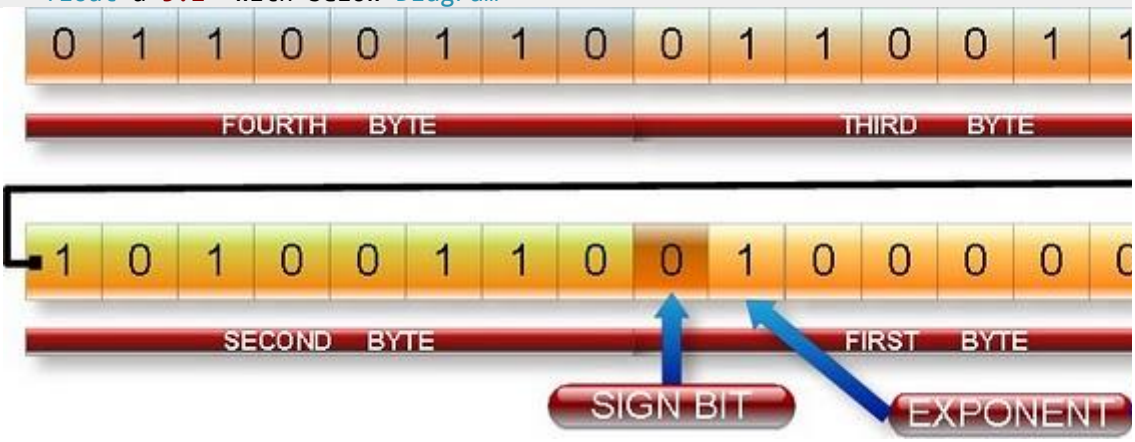
How to represent FLOAT number in memory in C

up vote22 down vote

While reading a tutorial I came across how to represent Float number in memory. The tutorial had an example with a floating point number.

```
float a=5.2  with below Diagram
```



Can anyone please tell how this 5.2 is converted in to binary and how it is represented in memory in above the above diagram?

c floating-point

shareimprove this question

edited Oct 21 '14 at 0:59

Community♦
1

asked Aug 2 '1

Amit Singh To
1,80884095

3 This tutorial seems to be developing a bit more on the subject : kipirvine.com/asm/workbook/floating_tut.htm– Clement Bellot Aug 2 '11 at 9:52

1 Is the diagram surely correct? h-schmidt.net/FloatApplet/IEEE754.html says something else. – phimuemue Aug 2 at 10:18

I have taken this diagram from question number 60 from this link cquestionbank.blogspot.com/2010/01/… – Ami Singh Tomar Aug 2 '11 at 10:36

add a comment

9 Answers

activeoldestvotes

up vote27 down vote

As was said, 5.2 is represented as a sign bit, an exponent and a mantissa. How do you encode 5.2?

5 is easy:

```
101.
```

The rest, 0.2 is 1/5, so divide `0.FFFFFFF...` (hex) by 5 (or `1.00000...` (hex)) and you get `0.3333333...` (hex).

That should give you

```
0.00110011001100110011001100110011...
```

Add 5, and you get

```
101.00110011001100110011...                    (5.2 * 2^0)
```

Now shift it and adjust the exponent

```
1.01001100110011001100110011001100110011... exp +2   (1.3 * 2^2)
```

Now you only have to add the bias of 127 (i.e. `129 = 0b10000001`) to the exponent and store it:

```
0 10000001 1010 0110 0110 0110 0110 0110
```

Forget the top 1 of the mantissa (which is always supposed to be 1, except for some special values, so it is not stored), and you get:

```
01000000 10100110 01100110 01100110
```

Now you only have to decide little or big endian.

This is not exactly how it works, but that is more or less what happens when a number like 5.2 is converted to binary.

shareimprove this answer — edited Aug 26 '14 at 16:33 — answered Aug

Rudy Velthuis
12.5k1740

Thanks @Rudy this is what I'am looking for. – Amit Singh Tomar Aug 2 '11 at 11:42

One thing i could not able to get how shift and adjust the exponent part works here – Amit Singh TomarAug 2 '11 11:47

2 The exponent counts powers of 2. The number should not change its value, so if you shift the mantissa right one b (which is equivalent to a division by 2), the exponent must be incremented by 1 (which is equivalent to a multiplic by 2). For 5.2, you do this twice. The top bit must always be 1, and the "decimal/binary point" is right behind it. S go from 5.2 * 2^0 to 2.6 * 2^1 to 1.3 * 2^2. The value must always be between 1 and 2 (except for special values or **very** small values, or NaNs, etc.). This is called normalization. Since the top bit is always 1, it is not stored. – Velthuis Aug 2 '11 at 11:53

The 'shifting' of exponents is done so you can store exponents smaller than zero. E.g. 0.125 is 1/8. That's stored as Mantissa (1)00..., sign +, Exp 124 (127-3) – MSalters Aug 2 '11 at 12:30

1 Can you please explain why did you choose `0.FFFFFFF...` to divide and also why add 127 ? – g4ur4v Feb 10 ' 13:47

show **4** more comments

up vote 32 down vote

I think the diagram is not one hundret percent correct.

Floats are stored in memory as follows:

They are decomposed into:

- sign `s` (denoting whether it's positive or negative) - 1 bit
- mantissa `m` (essentially the digits of your number - 24 bits
- exponent `e` - 7 bits

Then, you can write any number `x` as `s * m * 2^e` where `^` denotes exponentiation.

5.2 should be represented as follows:

```
0 10000001 01001100110011001100110
S    E              M
```

`S=0` denotes that it is a positive number, i.e. `s=+1`

`E` is to be interpreted as unsigned number, thus representing `129`. Note that you must subtract 127 from `E` to obtain the original exponent `e = E - 127 = 2`

`M` must be interpreted the following way: It is interpreted as a number beginning with a `1` followed by a point (`.`) and then digits after that point. The digits after `.` are the ones that are actually coded in `m`. We introduce weights for each digit:

```
bits in M: 0   1    0     0      1       ...
weight:    0.5 0.25 0.125 0.0625 0.03125 ... (take the half of the previous
in each step)
```

Now you sum up the weights where the corresponding bits are set. After you've done this, you add `1`(due to normalization in the IEEE standard, you always add 1 for interpreting `M`) and obtain the original `m`.

Now, you compute `x = s * m * 2^e` and get your original number.

So, the only thing left is that in real memory, bytes might be in reverse order. That is why

the number may not be stored as follows:

```
0 10000001 01001100110011001100110
S    E               M
```

but more the other way around (simply take 8-bit blocks and mirror their order)

```
01100110 01100110 10100110 01000000
MMMMMMMM MMMMMMMM EMMMMMMM SEEEEEEE
```

shareimprove this answer

edited Feb 22 '14 at 18:12

answered Aug

Samir Talwar
9,69511745

phimuemue
12.6k23170

Thanks @Phimuemue for your explanation. – Amit Singh Tomar Aug 2 '11 at 10:43

2 Floats in IEEE-754 are represented with a 8 bit exponent and a 23 bit mantissa. That excludes the hidden bit, whic always 1 (except for denormals or NaNs or 0), so the mantissa (significand) is usually 24 bits, but only 23 are stor The diagram shown is almost correct for little-endian floats. M should always have a top bit of 1, and the exponen adjusted correspondingly. – Rudy Velthuis Aug 2 '11 at 11:23

add a comment

up vote 6 down vote

The value is represented in memory in reverse order, but the confusing point may be that 5.2f is really represented as 5.1999998 due to the acuracy loss of the floating point values.

shareimprove this answer

answered Aug 2

beren
1163

add a comment

up vote 3 down vote

Raw float 5.2:

```
01000000101001100110011001100110
^ sign bit
```

In memory, reverse byte order (as your diagram):

```
01100110011001101010011001000000
                          ^ sign bit
```

shareimprove this answer

answered Aug 2

Momotapa Lim
21.8k33976

Thanks @Cicada but i wanted to know what is the equivalent value of 5.2 in decimal and then it can easily be understood how it is represented in memory – Amit Singh Tomar Aug 2 '11 at 9:56

2 That's not what your question says... – Momotapa Limpopo Aug 2 '11 at 9:57

add a comment

up vote 2 down vote

Representing 5.2 is very simple in binary logic:

```
    8 4 2 1
5 -> 0 1 0 1
```

For a decimal number:

Take .2 and multiply by 2 (since it is represented in binary).

```
.2 X 2 = 0.4 -> take the value after the
                decimal point, don't take the value before
```

```
                the decimal point

.4 X 2 = 0.8
.8 X 2 = 1.6
.6 X 2 = 1.2
.2 X 2 = 0.4
```

and so on...

After this step, take the value before the decimal point from output of the above steps:

```
.2 X 2 = 0.4 -> take 0 from this for representing in binary form
```

So the final o/p of 5.2 is:

```
0101.00110...
```

shareimprove this answer

edited Apr 16 '13 at 6:58

MiJyn
7821835

answered Mar 3

abababa
140111

1 this is not a floating point representation, just a normal binary representation. – Bartlomiej Lewandowski May 19 ' at 9:40

@BartlomiejLewandowski the OP ask about how to convert the number to binary and **then** represented in memory course the first step is converting it to binary – Luu Vĩnh Phúc Feb 2 '14 at 5:43

add a comment

up vote 1 down vote

5.2

The number is stored in form of "Sign Bit,Exponent,Mantissa. in binary form of 5 is `8 4 2 1` so `0101`and .2 is

```
.2*2=.4   0
.4*2=.8   0
.8*2=1.6  1
```

and sign bit 0 Because Number is positive.

```
0 0101 001....
```

shareimprove this answer

edited Sep 11 '14 at 10:00

AlexVogel
5,37483457

answered Sep 1

shivaji kapale
111

add a comment

up vote 0 down vote

5.2 is represented as "01000000101001100110011001100110"

Check the Converter Applet

shareimprove this answer

answered Aug 2

stacker
37k972135

add a comment

up vote 0 down vote

The conversion technique posted originally on the other website is shown unnecessarily complex (although it takes us to right answer) . For memory representation of 5.2 in memory:

First convert it into simple binary system, which will give us

101.00110011001100110011001100110011

Now change it into scientific form : 1.01001100110011001100110011 x 10^2 .
Now our sign bit is 0 as the number is positive
For exponent we need (127 + 2) upto 8 bits which gives us 10000001
Fraction is 01001100110011001100110 . (23 bits) (Discarding the leading 1 of scientific form)
=> the representation is
0 10000001 0100 1100 1100 1100 1100 110

shareimprove this answer

answered Aug 1

Navkamal Rakr
586

add a comment

up vote0 down vote

Below two references really helped me understand the IEE 754 floating point number encoding in binary format,
http://www.pitt.edu/~juy9/142/slides/L3-FP_Representation.pdf
http://en.wikipedia.org/wiki/Single-precision_floating-point_format

shareimprove this answer

answered Feb 2

Harish
211

1 Thanks for participating, but this question is three years old and has already been answered. Also, note that link-onl answers are not well-considered on this site, because links go stale and links to large, complete documents leave the reader to work out how the document answer the questions. The best answers provide the link but offer an explanat in the particular context of the question. – Pascal Cuoq Feb 2 '14 at 5:47