Convert 77 to 32 Bit Single Precision IEEE 754 Binary Floating Point Standard, From a Base 10 Decimal Number

$77_{(10)}$ to 32 bit single precision IEEE 754 binary floating point (1 bit for sign, 8 bits for exponent, 23 bits for mantissa) = ?

1. Divide the number repeatedly by 2.

Keep track of each remainder.

We stop when we get a quotient that is equal to zero.

division = quotient + remainder;
77 ÷ 2 = 38 + 1;
38 ÷ 2 = 19 + 0;
19 ÷ 2 = 9 + 1;
9 ÷ 2 = 4 + 1;
4 ÷ 2 = 2 + 0;
2 ÷ 2 = 1 + 0;
1 ÷ 2 = 0 + 1;

2. Construct the base 2 representation of the positive number.

Take all the remainders starting from the bottom of the list constructed above.

$$77_{(10)} =$$

$$100\ 1101_{(2)}$$

3. Normalize the binary representation of the number.

Shift the decimal mark 6 positions to the left so that only one non zero digit remains to the left of it:

$$77_{(10)} =$$

$$100\ 1101_{(2)} =$$

$$100\ 1101_{(2)} \times 2^{0} =$$

$$1.0011\ 01_{(2)} \times 2^{6}$$

4. Up to this moment, there are the following elements that would feed into the 32 bit single precision IEEE 754 binary floating point representation:

Sign: 0 (a positive number)

Exponent (unadjusted): 6

Mantissa (not normalized):
1.0011 01

5. Adjust the exponent.

Use the 8 bit excess/bias notation:

$$\text{Exponent (adjusted)} =$$

$$\text{Exponent (unadjusted)} + 2^{(8-1)} - 1 =$$

$$6 + 2^{(8-1)} - 1 =$$

$$(6 + 127)_{(10)} =$$

$$133_{(10)}$$

6. Convert the adjusted exponent from the decimal (base 10) to 8 bit binary.

Use the same technique of repeatedly dividing by 2:

division = quotient + remainder;

133 ÷ 2 = 66 + 1;

66 ÷ 2 = 33 + 0;

33 ÷ 2 = 16 + 1;

16 ÷ 2 = 8 + 0;

8 ÷ 2 = 4 + 0;

4 ÷ 2 = 2 + 0;

2 ÷ 2 = 1 + 0;

1 ÷ 2 = 0 + 1;

7. Construct the base 2 representation of the adjusted exponent.

Take all the remainders starting from the bottom of the list constructed above:

Exponent (adjusted) =

$133_{(10)}$ **=**

$1000\ 0101_{(2)}$

8. Normalize the mantissa.

a) Remove the leading (the leftmost) bit, since it's allways 1, and the decimal point, if the case.

b) Adjust its length to 23 bits, by adding the necessary number of zeros to the right.

Mantissa (normalized) =

~~1.~~ 00 1101 **0 0000 0000 0000 0000 =**

001 1010 0000 0000 0000 0000

9. The three elements that make up the number's 32 bit single precision IEEE 754 binary floating point representation:

Sign (1 bit) =
0 (a positive number)

Exponent (8 bits) =
1000 0101

Mantissa (23 bits) =
001 1010 0000 0000 0000 0000

Number 77 converted from decimal system (base 10) to 32 bit single precision IEEE 754 binary floating point:
0 - 1000 0101 - 001 1010 0000 0000 0000 0000

$77_{(10)}$ =

0-10000101-00110100000000000000000

(32 bits IEEE 754)

Sign (1 bit):

0

31

Exponent (8 bits):

1	0	0	0	0	1	0	1
30	29	28	27	26	25	24	23

Mantissa (23 bits):

0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

More operations of this kind:

76 = ? ... 78 = ?

Convert to 32 bit single precision IEEE 754 binary floating point standard

Decimal number: 77

Convert to 32 bit single precision IEEE 754 binary floating point standard

convert

A number in 32 bit single precision IEEE 754 binary floating point standard representation requires three building elements: sign (it takes 1 bit and it's either 0 for positive or 1 for negative numbers), exponent (8 bits) and mantissa (23 bits)

Latest decimal numbers converted from base ten to 32 bit single precision IEEE 754 floating point binary standard representation

77 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
0.200 000 000 000 000 011 102 230 246 251 565 404 26 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
955.74 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
-53 843 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
0.000 000 000 000 000 000 000 000 000 000 003 126 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
1 011 110.09 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
-42.312 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
10.158 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
45 376 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
11 000 011 011 110 100 000 000 009 972 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
3 758.16 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
-697.852 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*
83 579.18 to 32 bit single precision IEEE 754 binary floating point = ?	*Aug 19 09:50 UTC (GMT)*

All base ten decimal numbers converted to 32 bit single precision IEEE 754 binary floating point

How to convert decimal numbers from base ten to 32 bit single precision IEEE 754 binary floating point standard

Follow the steps below to convert a base 10 decimal number to 32 bit single precision IEEE 754 binary floating point:

1. If the number to be converted is negative, start with its the positive version.

2. First convert the integer part. Divide repeatedly by 2 the base ten positive representation of the integer number that is to be converted to binary, until we get a quotient that is equal to zero, keeping track of each remainder.

3. Construct the base 2 representation of the positive integer part of the number, by taking all the remainders of the previous dividing operations, starting from the bottom of the list constructed above. Thus, the last remainder of the divisions becomes the first symbol (the leftmost) of the base two number, while the first remainder becomes the last symbol (the rightmost).

4. Then convert the fractional part. Multiply the number repeatedly by 2, until we get a fractional part that is equal to zero, keeping track of each integer part of the results.

5. Construct the base 2 representation of the fractional part of the number by taking all the integer parts of the previous multiplying operations, starting from the top of the constructed list above (they should appear in the binary representation, from left to right, in the order they have been calculated).

6. Normalize the binary representation of the number, by shifting the decimal point (or if you prefer, the decimal mark) "n" positions either to the left or to the right, so that only one non zero digit remains to the left of the decimal point.

7. Adjust the exponent in 8 bit excess/bias notation and then convert it from decimal (base 10) to 8 bit binary, by using the same technique of repeatedly dividing by 2, as shown above:
Exponent (adjusted) = Exponent (unadjusted) + $2^{(8-1)}$ - 1

8. Normalize mantissa, remove the leading (leftmost) bit, since it's allways '1' (and the decimal sign if the case) and adjust its length to 23 bits, either by removing the excess bits from the right (losing precision...) or by adding extra '0' bits to the right.

9. Sign (it takes 1 bit) is either 1 for a negative or 0 for a positive number.

Example: convert the negative number -25.347 from decimal system (base ten) to 32 bit single precision IEEE 754 binary floating point:

1. Start with the positive version of the number:

|-25.347| = 25.347

2. First convert the integer part, 25. Divide it repeatedly by 2, keeping track of each remainder, until we get a quotient that is equal to zero:

division = quotient + **remainder**;

25 ÷ 2 = 12 + **1**;

12 ÷ 2 = 6 + **0**;

6 ÷ 2 = 3 + **0**;

3 ÷ 2 = 1 + **1**;

1 ÷ 2 = 0 + **1**;

We have encountered a quotient that is ZERO => FULL STOP

3. Construct the base 2 representation of the integer part of the number by taking all the remainders of the previous dividing operations, starting from the bottom of the list constructed above:

$25_{(10)} = 1\ 1001_{(2)}$

4. Then convert the fractional part, 0.347. Multiply repeatedly by 2, keeping track of each integer part of the results, until we get a fractional part that is equal to zero:

#) multiplying = **integer** + fractional part;

1) 0.347 × 2 = **0** + 0.694;

2) 0.694 × 2 = **1** + 0.388;

3) 0.388 × 2 = **0** + 0.776;

4) 0.776 × 2 = **1** + 0.552;

5) 0.552 × 2 = **1** + 0.104;

6) 0.104 × 2 = **0** + 0.208;

7) 0.208 × 2 = **0** + 0.416;

8) 0.416 × 2 = **0** + 0.832;

9) 0.832 × 2 = **1** + 0.664;

10) 0.664 × 2 = **1** + 0.328;

11) 0.328 × 2 = **0** + 0.656;

12) 0.656 × 2 = **1** + 0.312;

13) 0.312 × 2 = **0** + 0.624;

14) 0.624 × 2 = **1** + 0.248;

15) 0.248 × 2 = **0** + 0.496;

16) 0.496 × 2 = **0** + 0.992;

17) 0.992 × 2 = **1** + 0.984;

18) 0.984 × 2 = **1** + 0.968;

19) 0.968 × 2 = **1** + 0.936;

20) 0.936 × 2 = **1** + 0.872;

21) 0.872 × 2 = **1** + 0.744;

22) 0.744 × 2 = **1** + 0.488;

23) 0.488 × 2 = **0** + 0.976;

24) 0.976 × 2 = **1** + 0.952;

We didn't get any fractional part that was equal to zero. But we had enough iterations (over Mantissa limit = 23) and at least one integer part that was different from zero => FULL STOP (losing precision...).

5. Construct the base 2 representation of the fractional part of the number, by taking all the integer parts of the previous multiplying operations, starting from the top of the constructed list above:

$0.347_{(10)} = 0.0101\ 1000\ 1101\ 0100\ 1111\ 1101_{(2)}$

6. Summarizing - the positive number before normalization:

$25.347_{(10)} = 1\ 1001.0101\ 1000\ 1101\ 0100\ 1111\ 1101_{(2)}$

7. Normalize the binary representation of the number, shifting the decimal point 4 positions to the left so that only one non-zero digit stays to the left of the decimal point:

$25.347_{(10)} =$
$1\ 1001.0101\ 1000\ 1101\ 0100\ 1111\ 1101_{(2)} =$
$1\ 1001.0101\ 1000\ 1101\ 0100\ 1111\ 1101_{(2)} \times 2^0 =$
$1.1001\ 0101\ 1000\ 1101\ 0100\ 1111\ 1101_{(2)} \times 2^4$

8. Up to this moment, there are the following elements that would feed into the 32 bit single precision IEEE 754 binary floating point:

Sign: 1 (a negative number)

Exponent (unadjusted): 4

Mantissa (not-normalized): 1.1001 0101 1000 1101 0100 1111 1101

9. Adjust the exponent in 8 bit excess/bias notation and then convert it from decimal (base 10) to 8 bit binary (base 2), by using the same technique of repeatedly dividing it by 2, as already demonstrated above:

Exponent (adjusted) = Exponent (unadjusted) + $2^{(8-1)}$ - 1 = $(4 + 127)_{(10)}$ = $131_{(10)}$ = $1000\ 0011_{(2)}$

10. Normalize the mantissa, remove the leading (leftmost) bit, since it's allways '1' (and the decimal point) and adjust its length to 23 bits, by removing the excess bits from the right (losing precision...):

Mantissa (not-normalized): 1.1001 0101 1000 1101 0100 1111 1101

Mantissa (normalized): 100 1010 1100 0110 1010 0111

Conclusion:

Sign (1 bit) = 1 (a negative number)

Exponent (8 bits) = 1000 0011

Mantissa (23 bits) = 100 1010 1100 0110 1010 0111

Number -25.347, converted from the decimal system (base 10) to 32 bit single precision IEEE 754 binary floating point =

1 - 1000 0011 - 100 1010 1100 0110 1010 0111