Gabe Mulene Lab 4

Do these without using a calculator for DIRECT translations. There are all sorts of online floating point conversion apps. (but DO use a calculator to check your work)

Review the presentation as a guide for these problems

For each of the following binary floating-point numbers, supply the equivalent value as a base 10 fraction, and then as a base 10 decimal.

Binary Floating Point	Base 10 Fraction	Base 10 Decimal
1.1	1 1/2	1.5
11.11	3 + 1/2 + 1/4	3.75
1.101	1 + 1/2 + 1/8	1.625
101.1	5 + 1/2	5.5
1101.01	13 + 1/4	13.25
1.000011	1 + 1/32 + 1/64	1.046875
10000.1	16 + 1/2	16.5
0.111	1/2 + 1/4 + 1/8	0.875
0.110011	1/2 + 1/4 + 1/32 + 1/64	0.796875

For each of the following exponent values, shown here in decimal, supply the actual binary bits that would be used for an unsigned 8-bit **exponent** in the IEEE 754 Short (32 bit) Real format.

Exponent (E)	Binary Representation
2	10000001
5	127 + 5 = 132 = 1000_0100
0	127 + 0 = 0111_1111
-10	127 - 10 = 117 = 0111_0101
128	127 + 128 = 255 1111_1111
-1	127 - 1 = 126 0111_1110

For each of the following floating-point binary numbers, supply the normalized mantissa and the resulting exponent.

Binary Value	**Normalized As**	**Exponent**
10000.11	1.000011	4
1101.101	1.101101	3
.00101	1.01	-3
1.0001	1.0001	0
10000011.0	1.0000011	7
.0000011001	1.1001	-6

90
10
112
64
32 16
0111 _

For each of the following floating-point binary examples, supply the complete binary representation of the number in IEEE 754 Short (32 bit) Real format.

Binary Value	**Sign, Exponent, Mantissa**
-1.11	1 01111111 11000000000000000000000
+1101.1	0 (130) 1000_0010 1011 - 0 →
-.00101	1 (124) 0111_1100 010 → 0 →
+100111.0	0 (132) 1000_0100 00111 0 →
+.0000001101011	0 (120) 0111_1000 101011 0 →

For each of the following decimal fractions, supply the complete binary representation of the number in IEEE Short Real format. Include dashes between the relevant fields in the binary representation

Decimal Fraction	**Binary Fraction**	**IEEE Short Real format**
13.6875	1101.101	0 1000_0010 10110100000000000000000
11.96	1011. repeat	0 1000_0010 011 → repeating
0.67	.101	0 0111_1110 0 + repeat
22.9375	1.0110.	0 1000_0011 01101110000000000000000
3.333	11.0101	0 1000_0000 10101010101010101010101

repeating: 1111 0101_11000 010 1000

repeating: 101_0111_ (xxx) 1010_1011_110

Exercise: Write a C program that parses an ***IEEE 754 single precision floating point number*** and creates a struct containing the following members.

- char: sign // ‘+’ or ‘-’
- unsigned char or other 8 bit type: exponent
- int mantissa
- float decimal_value

Write a function called **parse_fraction** that accepts a 32 bit int and returns a struct. The fraction should be specified in 32 bit hexadecimal notation

Example: Converting to IEEE 754 Form

Put 0.085 in single-precision format

1. **The first step is to look at the sign of the number.**
   Because 0.085 is positive, the sign bit =0.

2. **Write 0.085 in base-2 scientific notation.**
   This means that we must factor it into a number in the range [1 <= n < 2] and a power of 2.

   $0.085 = (-1)^{0} * (1+\text{fraction}) * 2^{\text{power}}$, or:

   $0.085 / 2^{\text{power}} = (1+\text{fraction})$.

   So we can divide 0.085 by a power of 2 to get the (1 + fraction).

   $0.085 / 2^{-1} = 0.17$

   $0.085 / 2^{-2} = 0.34$

   $0.085 / 2^{-3} = 0.68$

   $0.085 / 2^{-4} = 1.36$ <= we now have the fraction in “+ 1” format

   Therefore, $0.085 = 1.36 * 2^{-4}$

3. **Find the exponent.**
   The power of 2 is -4, and the bias for the single-precision format is 127. This means that the exponent = 123_{10}, or 01111011_{2}

4. **Write the fraction in binary form**
   The fraction = 0.36 . Unfortunately, this is not a "pretty" number. The best we can do is to approximate the value. Single-precision format allows 23 bits for the fraction.

   Binary fractions look like this:

   $0.1 = (1/2) = 2^{-1}$

   $0.01 = (1/4) = 2^{-2}$

   $0.001 = (1/8) = 2^{-3}$

   To approximate 0.36, we can say:

   0.36 = (0/2) + **(1/4)** + (0/8) + **(1/16)** + **(1/32)** +...

   $0.36 = 2^{-2} + 2^{-4} + 2^{-5} + ...$

   $0.36_{10} \sim 0.01011100001010001111011_2$

   The binary string we need is: 01011100001010001111011.

   It's important to notice that you will not get 0.36 exactly. This is why floating-point numbers have error when you put them in IEEE 754 format.

5. **Now put the binary strings in the correct order -**
   1 bit for the sign, followed by 8 for the exponent, and 23 for the fraction. The answer is:

	Sign	Exponent	Fraction
Decimal	0	123	0.36
Binary	**0**	**01111011**	**01011100001010001111011**

Example: Converting to Float

Convert the following single-precision IEEE 754 number into a floating-point decimal value.

1 10000001 10110011001100110011010

1. **First, put the bits in three groups.**
   Bit **31** (the leftmost bit) show the sign of the number.
   Bits **23-30** (the next 8 bits) are the exponent.
   Bits **0-22** (on the right) give the fraction

2. **Now, look at the sign bit.**
   If this bit is a 1, the number is negative.
   If it is 0, the number is positive.

   This bit is 1, so the number is negative.

3. **Get the exponent and the correct bias.**
   The exponent is simply a positive binary number.
   $10000001_2 = 129_{10}$

   Remember that we will have to subtract a bias from this exponent to find the power of 2. Since this is a single-precision number, the bias is 127.

4. **Convert the fraction string into base ten.**
   This is the trickiest step. The binary string represents a fraction, so conversion is a little different.

   Binary fractions look like this:

   $0.1 = (1/2) = 2^{-1}$
   $0.01 = (1/4) = 2^{-2}$
   $0.001 = (1/8) = 2^{-3}$

   So, for this example, we multiply each digit by the corresponding power of 2:

   $0.10110011001100110011010_2 = \mathbf{1*2^{-1}} + 0*2^{-2} + \mathbf{1*2^{-3}} + \mathbf{1*2^{-4}} + 0*2^{-5} + 0 * 2^{-6} + ...$
   $0.10110011001100110011010_2 = 1/2 + 1/8 + 1/16 + ...$

   Note that this number is just an approximation on some decimal number. There will most likely be some error. In this case, the fraction is about 0.7000000476837158.

5. **This is all the information we need. We can put these numbers in the expression:**

   $(-1)^{\text{sign bit}} * (1+\text{fraction}) * 2^{\text{exponent - bias}}$

   $= (-1)^1 * (1.7000000476837158) * 2^{129-127}$
   = -6.8 The answer is approximately -6.8.