Floating-Point Representation

What are in representation

Floating-point representation consists of sign (1-bit), exponent (8-bits), and fraction bits (23-bits).

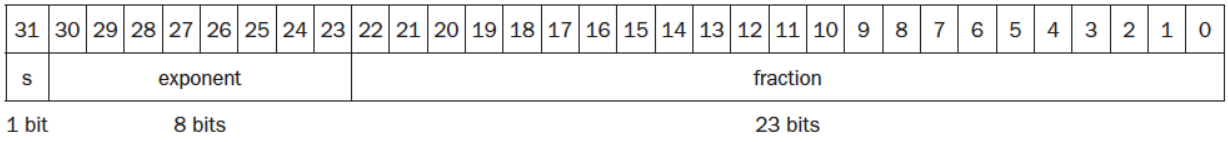

Figure 1. Single Precision Representation

Trade-off

As you know MIPS only deals with 32 bits at a time. This forces the size of each part in the floating-point representation fixed. As you see above, exponent can take 8-bits and fraction can take 23-bits. What if we had a bigger exponent field (that is, greater than just 8-bits)? We would have had a larger range of numbers that can be represented. However, this alludes that we have to give up on better accuracy. On the other hand, if we had a larger fraction field part in the representation (that is, greater than 23-bits), then we might represent a number with higher precision with the trade-off in the range of numbers that we can represent.

How to represent a floating-point number in general?

Floating-point numbers are of the form:

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

Out-of-range problem?

We know that fixed field size may generate overflow (+ values) or underflow (- values) problems. Since exponent field is fixed in size, any arithmetic computation that potentially causes a number that cannot be expressed with 8-bits of exponent could be problematic. One simple way to resolve this is increasing the size of exponent field although it doesn't solve the fundamental problem it has. This is the point where the double precision representation has to be introduced. In short, double precision has 11-bits of exponent field which is greater than that of single precision that it has 8-bits of exponent field.

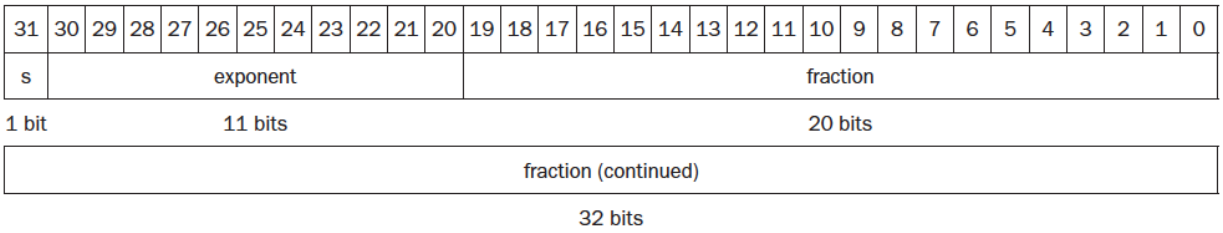

Figure 2. Double Precision Representation

Let's look at the maximum/minimum (approximately) value that each representation can have. For double precision case, it can represent a number within the following range:

$$2.0_{ten} \times 10^{-308} \leq x \leq 2.0_{ten} \times 10^{308}$$

whereas the single precision value has a range (approximately):

$$2.0_{ten} \times 10^{-38} \leq x \leq 2.0_{ten} \times 10^{38}$$

Biased Notation

The desired notation must represent the most negative exponent as

$$00\,..\,00_{two}$$

and the most positive exponent as

$$11\,..\,11_{two}$$

with the bias being the number subtracted from the normal, unsigned representation to determine the real value.

For single precision, it uses a bias of 127. Now, our final representation would look like the following:

$$(-1)^{S} \times (1 + Fraction) \times 2^{(Exponent - Bias)}$$

and the range of single precision numbers is then from as small as

$$\pm 1.0000\ 0000\ 0000\ 0000\ 0000\ 000_{two} \times 2^{-126}$$

to as large as

$$\pm 1.1111\ 1111\ 1111\ 1111\ 1111\ 111_{two} \times 2^{+126}$$

Examples

Example1:

Represent -0.75_{ten} into binary using the bias notation in single precision

$$-0.75_{ten} = -3/4$$
$$-\frac{3}{4} = -\frac{11_{two}}{2^2}$$

Using scientific notation,

$$-\frac{11_{two}}{2^2} = -1.1_{two} \times 2^{-1}$$

Next, we need to take this form and compare to the following general form:

$$(-1)^S \times (1 + Fraction) \times 2^{(Exponent - Bias)}$$

which turns out that

$$S = 1$$
$$F = 0.1_{two}$$
$$E = 126_{ten} = 0111\ 1110_{two} \because bias = 127$$

S (1-bits)	Exponent (8-bits)	Fraction (23-bits)
1	0111 1110	1000 0000 0000 0000 000

Example2:

Convert 639.6875_{ten} to binary using the IEEE754 single precision Floating point representation.

First, we have to separate 639 and 0.6875 and convert each to binary and then recombine them. First take 639 and keep divide it by 2. Details are shown in the below.

$$639 = 1\ 0011\ 1111_{two}$$

Value	Quotient	Remainder
639	319	1
319	159	1
159	79	1
79	39	1
39	19	1
19	9	1
9	4	1
4	2	0

2	1	0
1	0	1

If we read the remainder column from the bottom to top, you will see the same value at the top of the table, which is $1\,0011\,1111_{two}$.

Now, for the floating point number, the processes are quite similar to what we did with the integer part except that we are now multiplying by 2 and details are shown below.

$$0.6875 = 0.1011_{two}$$

Value	Result	Fraction	Remainder(?)
0.6875	1.375	0.375	1
0.375	0.75	0.75	0
0.75	1.5	0.5	1
0.5	1.0	0	1
0	0	0	0

This time we are reading the remainder column from top to bottom, which is 0.1011_{two}

As a result, $1\,0011\,1111_{two} + 0.1011_{two} = 1\,0011\,1111.1011_{two}$. Now we have make this value in scientific notation, which looks like the following:

$$1.0011\,1111\,1011 \times 2^{9}$$

Now, let's compare this representation to the IEEE754 single precision format:

$$(-1)^{S} \times (1 + Fraction) \times 2^{(Exponent - Bias)}$$

Since we know that $Bias = 127$, the rest of calculation is below:

Sign (1-bit)	Exponent (8-bits)	Fraction (23-bits)
0	136_{ten}	$0011\,1111\,1011\,0000\cdots_{two}$
0	$1000\,1000_{two}$	$0011\,1111\,1011\,0000\cdots_{two}$