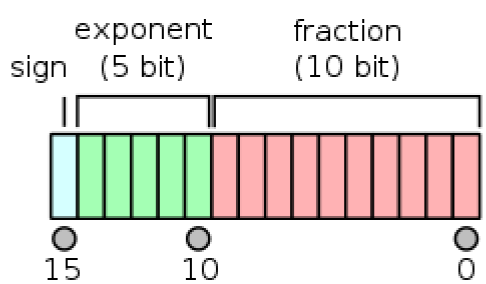
16-BIT FLOATING-POINT UNIT (FPU)

Operation Code	Operation
000	A + B
001	A * B
010	A / B
011	toInt(A)
100	A > B
101	A < B
110	A == B

Introduction:

16-bit floating-point (binary16):



The floating-point binary representation is similar to the scientific notation (example: 3.65 $\times 10^2$.)

The number 15_{10} in binary as an integer is represented as 1111_2, but as a float, it is $1.111_2 \times 2^3$.

Binary16 format:

- Sign bit: 1 bit
- Exponent width: 5 bits
- Significand precision: 11 bits (10 explicitly stored)

Some definitions:

- Normalization: a floating-point number is said to be normalized if there exists only one 1-bit on the left side of the floating point. Example: 1.111×2^3 is said to be normalized, while 11.11×2^2 and 0.1111×2^4 are not; despite all of them having the same value.
- Mantissa: the bits after the floating point. It represents the precision.
- Exponent: represents the amount by which to shift the floating point.

Special notes:

- Floating-point value calculation:

$$(-1)^{sign} \times 2^{exponent-15_{10}} \times 1.Mantissa$$

- Exponent bias: the exponent is biased by 15_{10}. That is, to get the true value of the exponent, 15_{10} must be subtracted from it.
- Implicitly stored bit: its assumed that there's an implicitly stored 1-bit before the mantissa unless the exponent is set to 00000.

Examples:

$$1.5_{10} = 0\ 01111\ 1000000000 = 1.1 \times 2^{15-15}$$

$$0.5_{10} = 0\ 01110\ 0000000000 = 1.0 \times 2^{14-15}$$

$$3.0_{10} = 0\ 10000\ 1000000000 = 1.1 \times 2^{16-15}$$

$$2.0_{10} = 0\ 10000\ 0000000000 = 1.0 \times 2^{16-15}$$

Operations' Implementation:

- "000": Addition/Subtraction (A + B)

Addition is carried out in 4 steps:

1) Setting the sign:

If both A and B have the same sign, then the sign is set the same

Else if the exponent of A is greater than that of B, then the sign is set to that of A

Else if the exponent of B is greater than that of A, then the sign is set to that of B

Else if the mantissa of A is greater than that of B, then the sign is set to that of A

Else it is set to that of B

2) Setting the exponent:

The exponent is set tentatively to the higher exponent, or if both exponents are equal, it is set the same.

3) Setting the mantissa:

First, in order to add or subtract the mantissas, they are re-arranged if necessary such that the number with the higher magnitude is placed in A. The floating point in B is shifted as necessary such that both numbers have the same exponent and then B is subtracted or added to A based on the signs of the numbers: if they have the same sign, they are added, and if they have opposite signs, they are subtracted.

4) The result is normalized if necessary.

Example: A – B = 4 – 1 = 0 10001 0000000000 + 1 01111 0000000000

1) The sign is set to 0 the same as A since the exponent of A is greater than that of B

   0 UUUUU UUUUUUUUUU

2) The tentative exponent is set to 10001 the same as A since the exponent of A is greater than that of B

   0 10001 UUUUUUUUUU

3) The floating point of B is shifted to the right so that both A and B have the same exponent.
   1 01111 0000000000 –> 1 10001 0100000000
   The significands are subtracted since A.sign xor B.sign = 1 (opposite signs)
   1.0000000000 – 0.0100000000 = 0.1100000000

   Therefore, the mantissa of output is tentatively set to the result 1100000000.

   0 10001 1100000000
4) The output is normalized:

   0 10000 1000000000

- "001": Multiplication (A * B)

Multiplication is carried out in 4 steps:

1) Setting the sign:

The sign is set by xor–ing the signs of A and B.

X.sign = A.sign xor B.sign;

2) Setting the exponent:

The exponents of A and B are simply added and the additional bias is removed

X.exp = A.exp + B.exp – 15_{10}

3) Setting the significand:

The significands of A and B are multiplied and the result is placed in the output mantissa.

4) Normalizing the output if necessary

- “010”: Division (A / B)

Division is carried out in 4 steps:

1) Setting the sign:

The sign is set by xor–ring the signs of A and B

X.sign = A.sign xor B.sign;

2) Setting the exponent:

The tentative exponent is set by subtracting the exponent of B from A and adding the bias

X.exp = A.exp – B.exp + 15_{10};

3) Setting the significand

The significand is calculated in 2 steps:

1. Integer division to set the most significant implicitly stored bit
2. Long division to set the mantissa

4) Normalizing the result if necessary

- “011”: toInt(A)

The conversion to integer is fairly simple. The floating point is shifted until the exponent is equal to 01111_2 , which is equivalent to a real value of 0, and the value on the left side of the floating point is set to the integer.

- “100”: A > B

The result is either set to 1, which represents True, or 0 which represents False.

The comparison takes place in 3 ordered steps until the result is known:

1) The signs are compared:
   True is returned if A is +ve and B is –ve
2) The exponents are compared if both the signs are equal:
   True is returned if A.exp > B.exp
3) The mantissas are compared if both the exponents are also equal
   True is returned if A.mantissa > B.mantissa

- “101”: A < B

Its implementation is as simple as switching the positions of A and B in the previous operation

- “110”: A == B

All the bits of A and B are compared and True is returned if they are all the same.