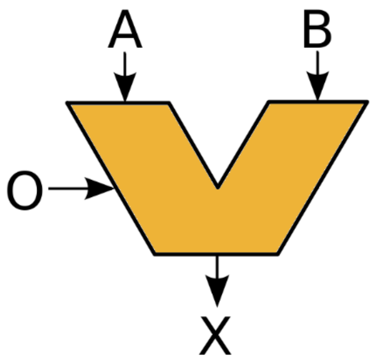
16–BIT ARITHMETIC LOGICAL UNIT (ALU)

Operation Code	Operation
0000	A + B
0001	A – B
0010	A + 1
0011	A – 1
0100	NOT A
0101	A and B
0110	A or B
0111	A xor B
1000	A * B
1001	A / B
1010	A > B
1011	A == B
1100	A < B
1101	toFloat(A)
1110	rand()
1111	rand(seed)



Implementation:

The operations from code 0000 to 1100 were implemented using the standard IEEE libraries provided with Xilinx:

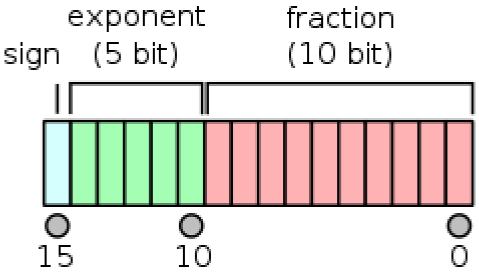
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use IEEE.NUMERIC_STD.ALL;

As for the operations from code 1101 to 1111; These were coded explicitly.

- “1101”: toFloat()



The floating–point representation is similar to the scientific notation for decimal numbers.

The conversion to a 16–bit float takes place in 4 steps:

1) Conversion from 2’s complement to a signed representation.
2) Setting the sign:
   The sign is set to the most significant bit of the integer.
3) Setting the exponent:
   The number of significant bits in the integer is counted by shifting logically to the right until only 1 significant bit remains in the number. The exponent is then biased by adding $(15)_{10}$.

4) Setting the mantissa/fraction:
   The 10 most significant bits after the implicitly stored 1–bit are set to the mantissa.

- “1110” and “1111”: Rand operations

The rand operations are implemented using a simple linear–feedback shift register (LFSR).

Example: 8–Bit LFSR

X7	X6	X5	X4	X3	X2	X1	X0

The bits are shifted to the right and the output bit X0 is used to set the input bit X7 such that

$$X7 = X7 \text{ xor } X0$$

The random 16–bit combination is divided into 2 of such registers.