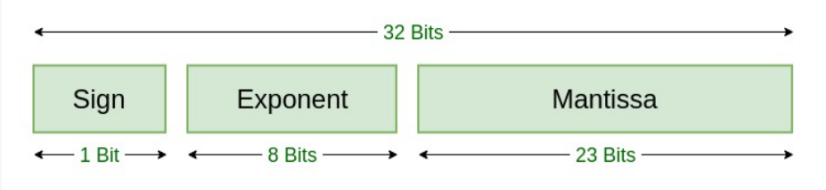
Submitted by:
Abhinandan Sharma (2018A3PS0095P)
Tanvi Shewale (2018A3PS0298P

FLOATING POINT DIVISION

Representation of Single-Precision Floating-Point Numbers (32 bit)



Performing Floating-Point Division

Performing a floating-point division involves three significant steps. The exceptions and special cases have been discussed later.

1) **Assigning the Sign bit**
   The sign bit depends on the sign bits of the Dividend and the Divisor. When both these sign bits are identical, i.e. both zeros or both ones, the sign bit of the result is a zero. Conversely, when the two sign bits have different values, the sign bit of the result is a one. This can be performed using an XOR operation.

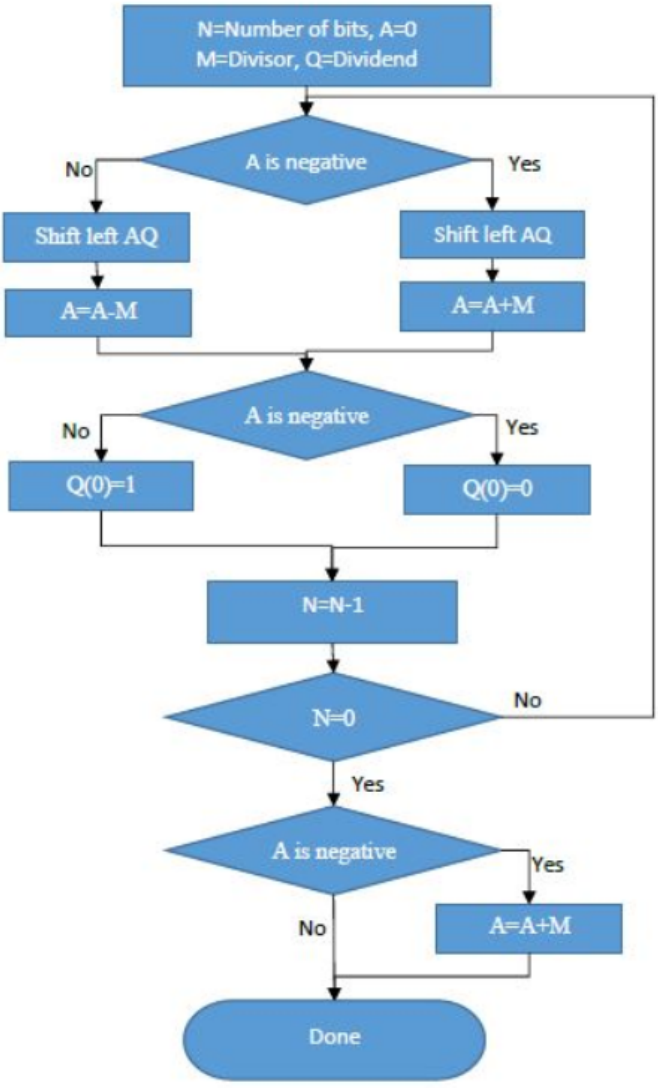
2) **Assigning the Exponent**
   To calculate the exponent of the result, we need to subtract the exponent of the divisor from the exponent of the dividend. Care needs to be taken here since the exponents in the floating-point representation are biased.

3) **Assigning the Mantissa**
   The mantissa of the result is determined by performing a division of the mantissa of the dividend and the divisor. The non-restoring algorithm is used for this computation. This is explained below.

Non-Restoring Algorithm

The steps for performing non-restoring division are as follows.
Step1: The registers are initialized with appropriate values(Q = Dividend, M = Divisor, A = 0, N = Number of bits in dividend)
Step2: Check the sign bit of register A
Step3: If the sign bit is 1, shift left the contents of AQ and perform A = A+M. Otherwise, shift left the contents of AQ and perform A = A-M

Step4: Check the sign bit of register A
Step5: If the sign bit is 1, put Q[0] = 0, else put Q[0] = 1
Step6: Decrement N
Step7: If N is zero, go to Step2, else go to next step\
Step8: Check sign bit of register A, if it is one, perform A = A+M
Step9: Register Q contains the quotient and register A contains the remainder
The ASM for this algorithm is shown in the following figure



Checking for test cases and extreme values

Testing is an integral part of checking the validity of the design. Apart from the usual cases given as inputs, there are several extreme cases that need to be kept in mind while calculating the result. Some of these need to be handled casewise to get the desired result. The table below shows the extreme cases considered.

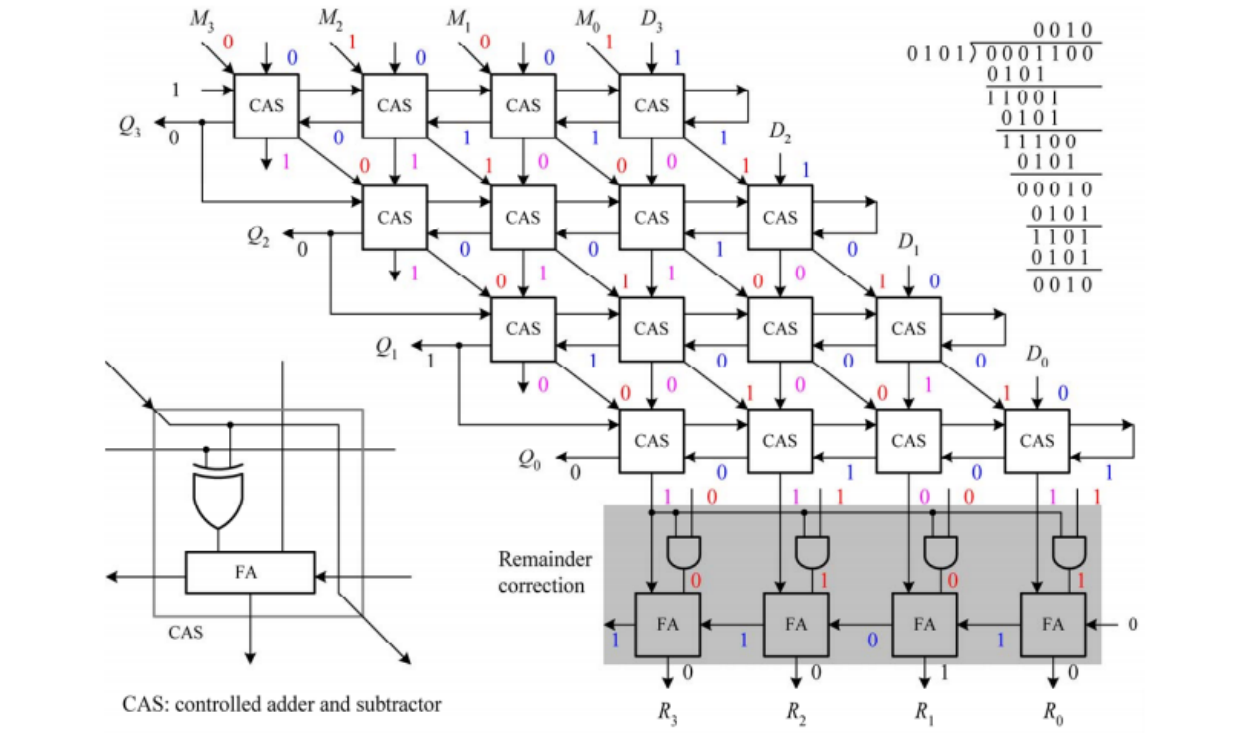
Dividend	Divisor	Result	Exception
±∞	±∞	NaN	Invalid Operands
±∞	±n	±∞	Invalid Operands
±n	±∞	±0	Invalid Operands
±∞	±0	±∞	Invalid Operands
±0	±∞	±0	Invalid Operands
±0	±0	NaN	Divide by zero
±0	±n	±0	-
±n	±0	±∞	Divide by zero
Either is NaN		NaN	Invalid Operands
±n	±n	±∞	Overflow
±n	±n	±0	Underflow

Apart from these, cases that involve the highest/lowest normal and subnormal numbers require special handling to maintain precision and get the correct result.

Design Implementation

The basic format of the block is as follows

```
module fpdiv(AbyB,DONE,EXCEPTION,InputA,InputB,CLOCK,RESET);
input CLOCK,RESET ; // Active High Synchronous Reset
input [31:0] InputA,InputB ;
output [31:0]AbyB;
output DONE ; // ‘0’ while calculating, ‘1’ when the result is ready
output [1:0]EXCEPTION; // Used to output exceptions
```



Generating an exceptions register for each of the inputs

The above mentioned border cases are handled separately. For this, exceptions_A and exceptions_B signals each of 5 bit are generated. These are encoded in one hot scheme with each bit signifying the presence of inf, zero, NaN, subnormal or usual dividend cases.

Generating the sign bit

Irrespective of the exceptions, the sign bit is generated based on the sign bits of the inputs

AbyB_sign = A_sign ^ B_sign

Generating the exponent

A temporary exponent is generated by subtracting the exponent of the divisor from the dividend. The biasing is maintained by adding 127 to the subtracted result. Based on the subnormal condition, this exponent will be further modified.

Generating the mantissa

The mantissa for the answer is obtained by dividing the mantissa of A by mantissa of B. This division is performed by the non-restoring algorithm described before. Structural modelling is used for this part. To maintain the precision of the result, extra zeros are appended to the dividend and the divisor to make it a 51-bit number. The non-restoring division is performed on this extended value. The result is a 50-bit number which will later be trimmed based on the cases.

Normalization
The mantissa obtained from this is normalized after appending extra zeros at the end for maintaining precision. The extra exponent is also calculated here.

Overflow Check
The overflow bit is set if the next exponent is more than 127

Underflow Check
If the net exponent is -127, the subnormal bit is set. If the net exponent is less than -127, the underflow bit is set.

Final Assignment
The final assignment is done based on the exception signals, underflow and overflow signals and the usual result calculated using the case statement