SVKM'S NMIMS, School of Technology Management & Engineering
Navi-Mumbai
B-Tech (A.Y. 2020-21)
Lecture Notes
Unit-4 Computer Arithmetic

Content:

1) IEEE 754 format
2) Signed Integer addition and subtraction
3) Addition – **Ripple carry adder, Save Adder Carry, Carry-look-ahead adder, Carry Select Adder.**
4) Multiplication – **Array Multiplier, Shift-and-Add Multiplier, Booth multiplier, Carry Save Multiplier, Wallace Tree Multiplier.**
5) Division restoring and Non-restoring techniques
6) Floating point arithmetic

IEEE 754 standard format:

- The IEEE-754 was published on 12"October 1985 and superseded in 1987, 2008 and 2019. This standard is the product of 754-WG-Working group for floating point arithmetic and sponsored by C/MSC-Microprocessor standard committee.
- The standard specifies interchange and arithmetic formats and methods for binary and decimal floating-point arithmetic in a computer programming environment.
- The standard can be implemented entirely in software or in hardware or in any combination of software and hardware.
- This standard specifies:
  - ➢ Formats for binary and decimal floating-point data, for computation and data interchange.

- Addition, subtraction, multiplication, division, fused multiply-add, square root, compare, and other operations.
- Conversions between integer and floating-point formats.
- Conversions between different floating-point formats.
- Conversions between floating-point formats and external representations as character sequences.
- Floating-point exceptions and their handling, including data that are not numbers (NaNs).

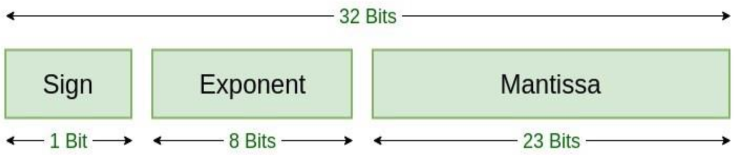
- The standard specifies five basic formats of:

    - Three binary formats, with encodings in lengths of 32, 64, and 128 bits.
    - Two decimal formats, with encodings in lengths of 64 and 128 bits.
    - Each number in both formats is represented by:

    $$n = [(-1)^{s}] \times [b^{e}] \times m$$

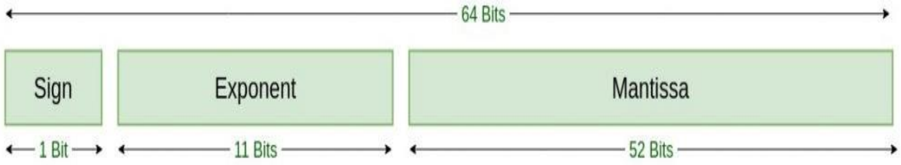
    Where; (s) is the sign bit which can take either 1 or 0 value, (b) is the base for binary b=2 and for decimal b=10, (e) is the exponent, (m) is the mantissa.
- The format supports two infinities (- ∞, + ∞) and two NaNqNaN (quiet) and sNaN (signalling).
- The most commonly used precision in the format are:
    - Single precision (32 bit).
    - Double precision (64 bit).

- **Single Precision:**
    - In single precision format, using binary base, the least significant 23 bits from bit D0 to D22 are used to store the fractional part i. e. Mantissa.
    - Next bits from D23 to D30 are used to store the 8-bit exponent.
    - Bit 31 is used to store the sign bit.



Single Precision
IEEE 754 Floating-Point Standard

- Double Precision:
  - In double precision format, using binary base, the least significant 52 bits from bit D0 to D51 are used to store the fractional part i.e. Mantissa.
  - Next bits from D52 to D62 are used to store the 11-bit exponent.
  - Bit 63 is used to store the sign bit.

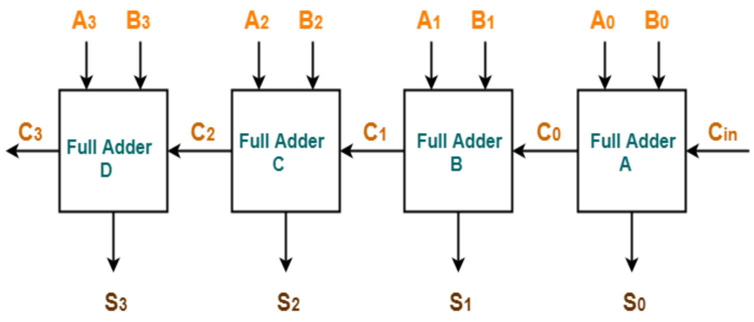


Double Precision
IEEE 754 Floating-Point Standard

High Speed Adders:

1. Ripple Carry Adder (RCA)

- Ripple Carry Adder (RCA) is considered as the most Simplistic approach among all the addition algorithms. A N-bit Ripple Carry Adder requires N number of full adders.
- It is basically a Cascading formation of full adders in series.
- As a full adder block process three inputs along with carry bit and produce two outputs i.e. Sum bit and Carry-out Bits, the carry of one full adder block acts as a carry in for the next full adder.
- Fig. shows Block Diagram of 4-bit Ripple carry adder:



4-bit Ripple Carry Adder

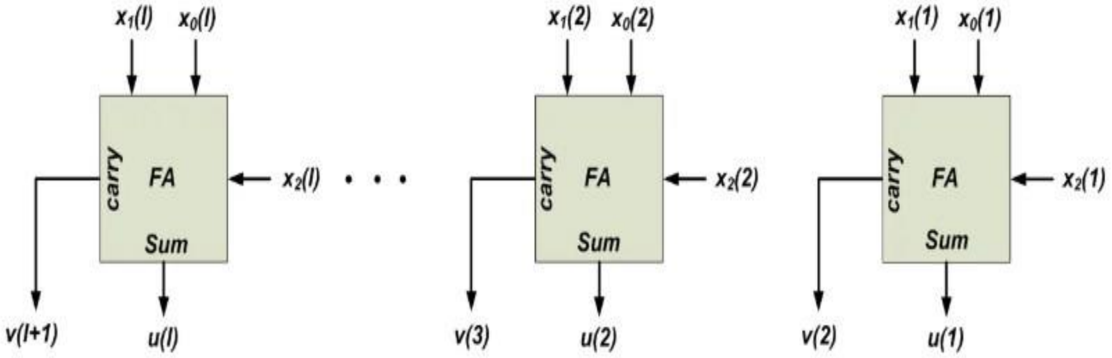
- The Sum bit and the carry bit will be calculated as follows:

Sn = An^Bn^Cn-1

Cout = [(An . Bn) + (Bn . Cn-1) + (An . Cn-1)]

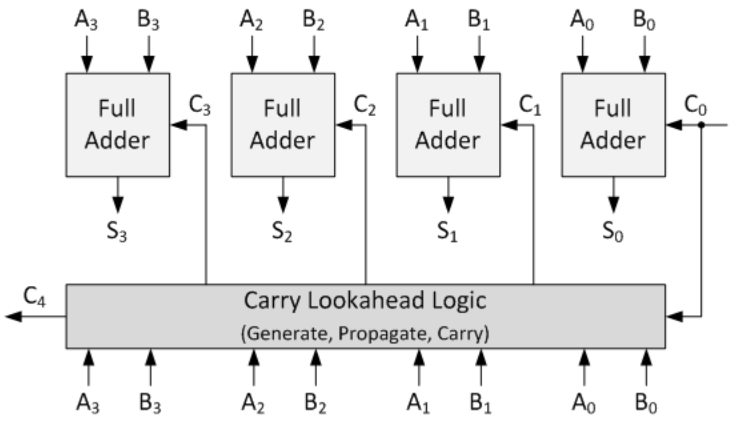
2. Carry Save Adder (CSA):

- Another Algorithm for faster calculation of Addition is Carry Save Adder. It is also same as full Adder.
- From the two inputs we first produce two temporary Outputs named as Sum and Carry.
- For getting sum bit we first perform bitwise XOR and for the Carry bit we execute bitwise AND for the two input numbers. And then finally add them by shifting Carry bit left by one place to Sum bit up to produce final answer.



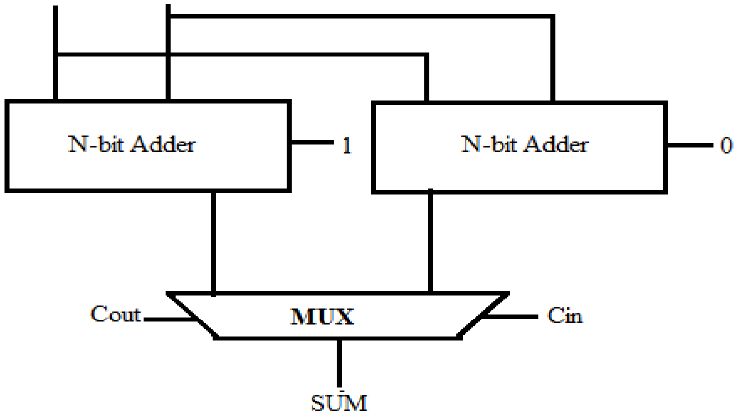
3. Carry Look Ahead Adder

- Another fast addition topology is Carry Look Ahead Adder.
- The main advantage of Carry Look Ahead Adder over Ripple Carry Adder is it improves the speed of operation by reducing the time needed to determine the carry bits.
- Carry Look Ahead adder calculate the sum and carry simultaneously by using a separate carry generation unit.
- Finally, the sum bit is calculated by XOR-ing the Propagation bit and its previous stage carry bit.



4. Carry Select Adder (CSLA):

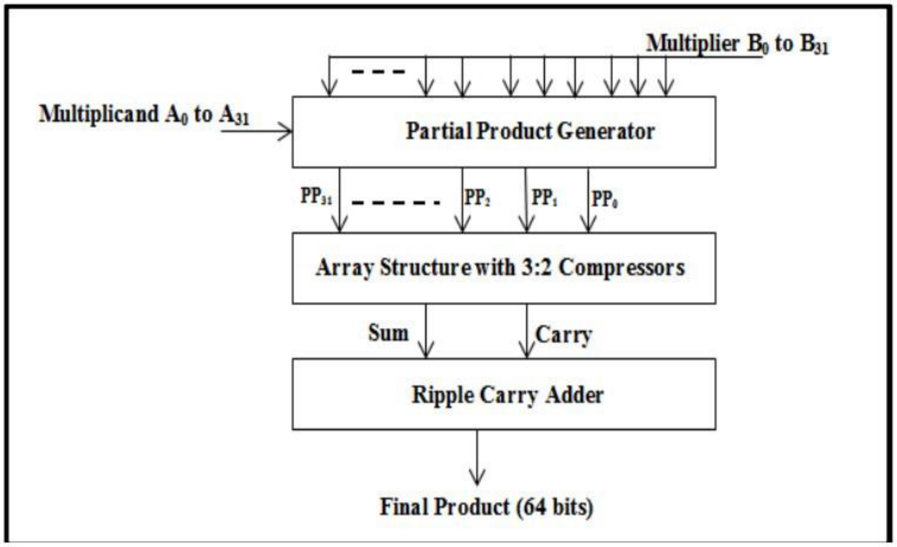
- Carry Select Adder algorithm is nothing but calculating sum by guessing the carry input which we will be getting from previous stage.
- Two adder work simultaneously in this adder algorithms where one is calculating sum by taking carry input as 0 from the previous stage and the other adder does so by taking Carry bit as 1.
- Several multiplexers are used to choose the appropriate sum bit corresponding to its previous carry out bit.



High Speed Multipliers:

1. Array Multiplier

- Partial products are obtained by multiplying each bit of the multiplier with the multiplicand at the output of a partial product generator.
- These partial products are added using an array structure of 3:2 Compressors.
- The final adder used is a Ripple Carry Adder to add the sequence of sum and carry obtained as the output of array structure with 3:2 compressors.
- The block diagram of 32× 32 bit array multiplier is shown below in fig.


Multiplier B_0 to B_{31}
Multiplicand A_0 to A_{31}
Partial Product Generator
PP_{31}
PP_2
PP_1
PP_0
Array Structure with 3:2 Compressors
Sum
Carry
Ripple Carry Adder
Final Product (64 bits)

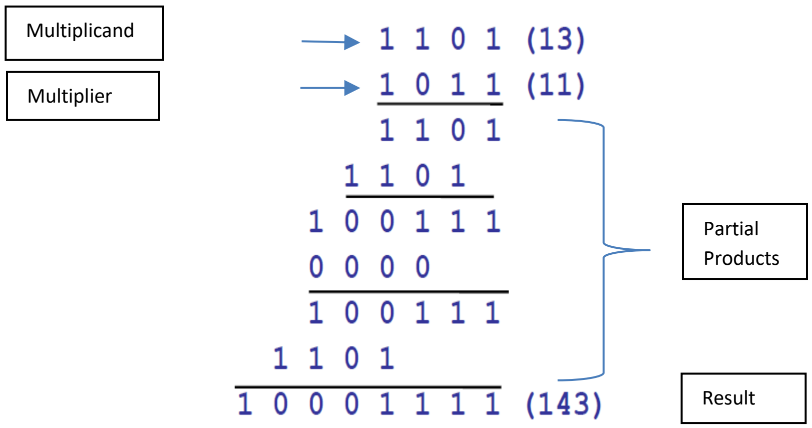
Block diagram of 32× 32 bit Array Multiplier

*Note link: https://www.youtube.com/watch?v=gTxgiJHBfsI

2. Shift-and-Add Multiplication:

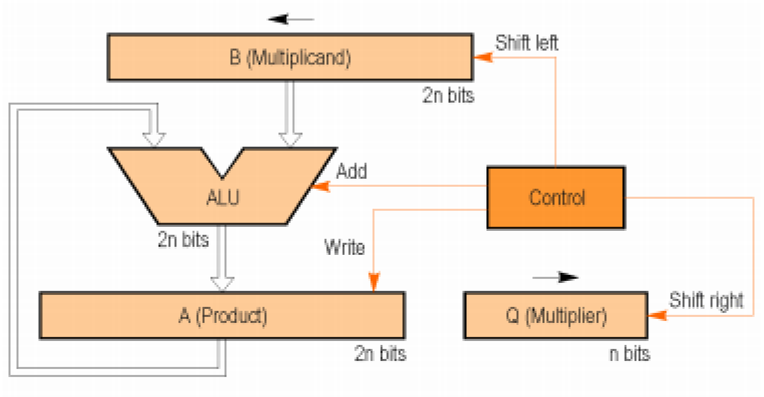
- Shift-and-add multiplication adds the multiplicand X to itself Y times, where Y denotes the multiplier.
- To multiply two numbers by paper and pencil, the algorithm is to take the digits of the multiplier one at a time from right to left is similar to the multiplication performed by paper and pencil.
- This method, multiplying the multiplicand by a single digit of the multiplier and placing the intermediate product in the appropriate positions to the left of the earlier results.

- As an example, consider the multiplication of two unsigned 4-bit numbers, 13 (1101) and 11 (1011).



```
        1 1 0 1  (13)
        1 0 1 1  (11)
        1 1 0 1
      1 1 0 1
    1 0 0 1 1 1
    0 0 0 0
    1 0 0 1 1 1
  1 1 0 1
1 0 0 0 1 1 1 1  (143)
```

- In the case of binary multiplication, since the digits are 0 and 1, each step of the multiplication is simple. If the multiplier digit is 1, a copy of the multiplicand (1 × multiplicand) is placed in the proper positions; if the multiplier digit is 0, a number of 0 digits (0 × multiplicand) are placed in the proper positions.
- Consider the multiplication of positive numbers. The first version of the multiplier circuit, which implements the shift-and-add multiplication method for two n-bit numbers, is shown in figure below:

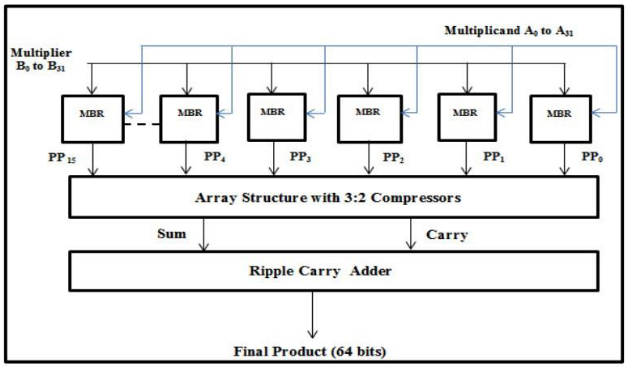


First version of the multiplier circuit.

3. Modified Booth Multiplier:

- The Partial products are generated using the Radix -4 Modified Booth Algorithm in which the multiplier is grouped in blocks of three for recoding and this recoded operation is performed on the multiplicand to obtain partial products and this is done using Modified Booth Recorder (MBR).

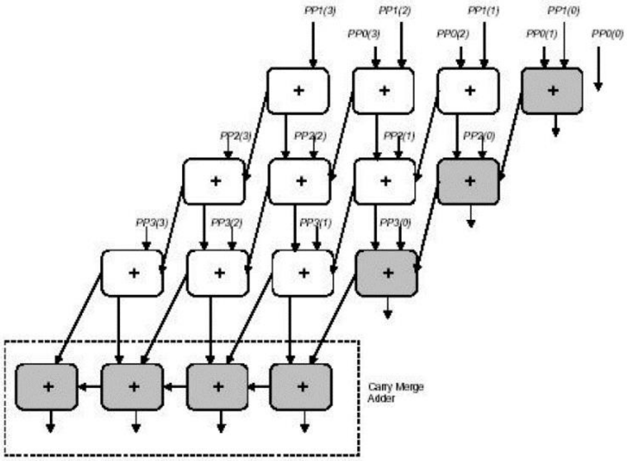
- The no. of partial products is reduced to n/2 where ‘n’ indicates the available multiplier bits. With inputs of 32 bits, 16 partial products obtained. An array made of 3:2 compressors is used to add these partial products.
- The final adder used is the ripple carry adder to add the sequence of sum and carry.
- The fig. below depicts the block diagram a 32×32 bit Modified Booth multiplier.



Block diagram of 32× 32 bit Modified Booth Multiplier

4. Carry Save Array Multiplier

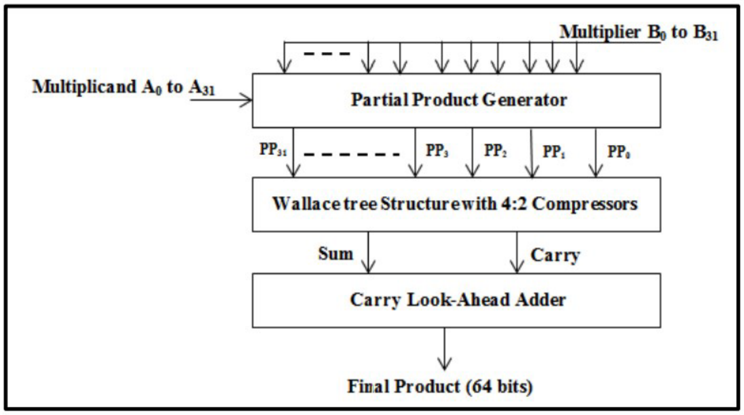
The carry-save array multiplier uses an array of carry-save adders for the accumulation of partial product. It uses a carry-propagate adder for the generation of the final product. This reduces the critical path delay of the multiplier since the carry-save adders pass the carry to the next level of adders rather than the adjacent ones. Figure, shows the structure of the carry-save array multiplier.



Block diagram of Carry Save Array Multiplier

5. Wallace tree Multiplier:

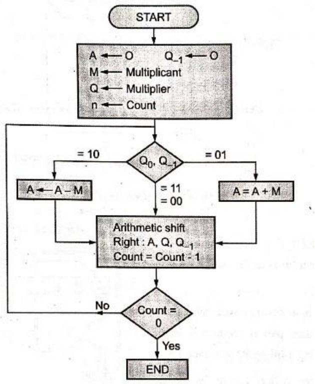
- Partial products are obtained by multiplying each bit of the multiplier with the multiplicand in a partial product generator.
- The multiplier and multiplicand is each of 32 bit, thus generating 32 partial products.
- These partial products are added using a Wallace tree structure made up of several 4:2 Compressors thus increasing the speed of accumulation.
- The sequence of sum and carry bits obtained at the end is given to final carry propagate adder that is Carry look-ahead adder.
- The block diagram of a 32 bit Wallace tree multiplier is shown below:


Multiplier B_0 to B_{31}
Multiplicand A_0 to A_{31}
Partial Product Generator
PP_{31}
PP_3
PP_2
PP_1
PP_0
Wallace tree Structure with 4:2 Compressors
Sum
Carry
Carry Look-Ahead Adder
Final Product (64 bits)

Block diagram of 32× 32 bit Wallace tree Multiplier

Booth's algorithm:

- Booth's algorithm is widely used for multiplication and division as it provides faster result compared to normal approach. Fig. depicts the flow chart of the algorithm.



- Step 1: Initialize the registers i.e. place the multiplier and multiplicand in the Q and M register respectively. Also A and Q1 is initialized to zero.
- Step 2: The control logic scans the bit of the multiplier. Each bit of the multiplier is examined with bit is it right.

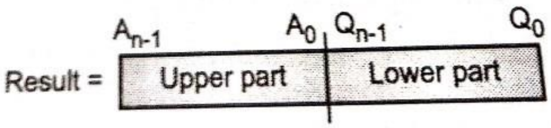
- Step 2.1: If both bits are same i.e. [0-0 and 1-1] then all the bits are shifted to right of register A, Q and Q1 this is called as [Arithmetic shift].

- Step 2.2: If the two bits are of the form [0-1 or 1-0], then the multiplicant is subtracted from register A after addition (or added), or subtraction of the bits of register A, Q and Q1 are shifted right by 1 bit.

Step 3: Decrement count value and check if count equals zero.

- Step 3.1: If count ≠ 0, repeat step 2.

- Step 3.2: If count = 0, register A holds upper n bits including MSB and Register Q holds lower n bits of the result.



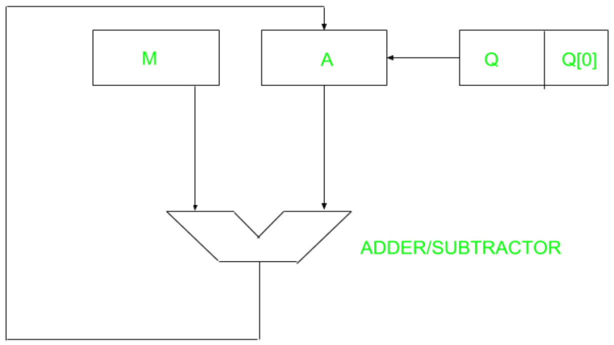
Note: Problem on Booth's algorithm is shared in YouTube video link.
https://www.youtube.com/watch?v=QFXaddi-Ag8

Restoring Division Algorithm for Unsigned Integer:

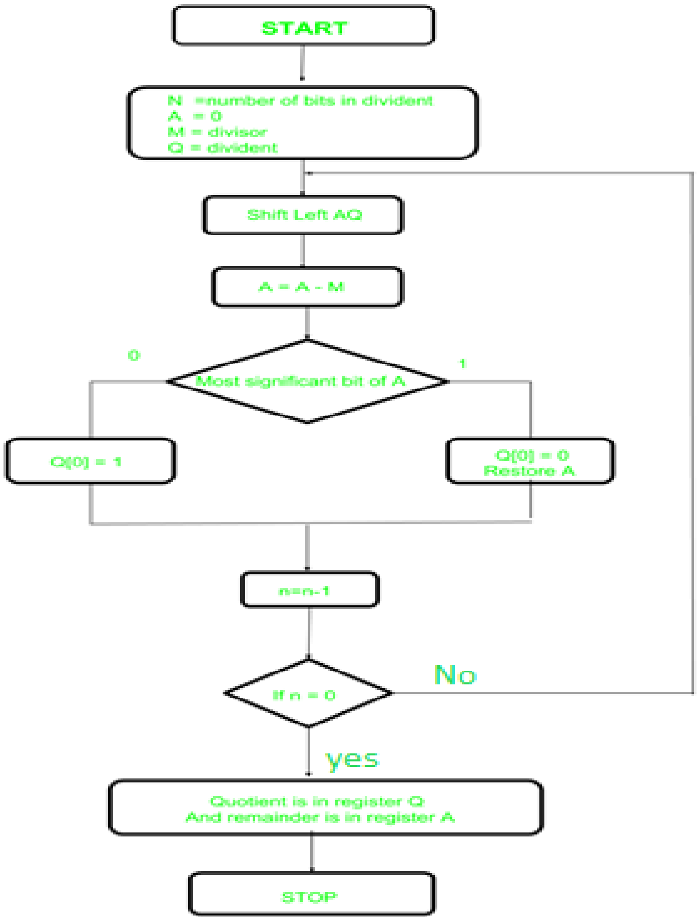
A division algorithm provides a quotient and a remainder when we divide two number. They are generally of two type **slow algorithm and fast algorithm**.

Slow division algorithms are restoring, non-restoring, non-performing restoring, SRT algorithm and under fast comes Newton–Raphson and Goldschmidt.

In this, will be performing restoring algorithm for unsigned integer. Restoring term is due to fact that value of register A is restored after each iteration.



Here, register Q contain quotient and register A contain remainder. Here, n-bit dividend is loaded in Q and divisor is loaded in M. Value of Register is initially kept 0 and this is the register whose value is restored during iteration due to which it is named Restoring.


START
N =number of bits in divident
A = 0
M = divisor
Q = divident
Shift Left AQ
A = A - M
0
Most significant bit of A
1
Q[0] = 1
Q[0] = 0
Restore A
n=n-1
If n = 0
No
yes
Quotient is in register Q
And remainder is in register A
STOP

Let’s pick the step involved:

- **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend).
- **Step-2:** Then the content of register A and Q is shifted left as if they are a single unit.
- **Step-3:** Then content of register M is subtracted from A and result is stored in A.
- **Step-4:** Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e the value of A before the subtraction with M.
- **Step-5:** The value of counter n is decremented.
- **Step-6:** If the value of n becomes zero, we get of the loop otherwise we repeat from step 2.
- **Step-7:** Finally, the register Q contain the quotient and A contain remainder.

***Note: Problem on Restoring algorithm is shared in YouTube video link.**
https://www.youtube.com/watch?v=PzV6gYpVLuc

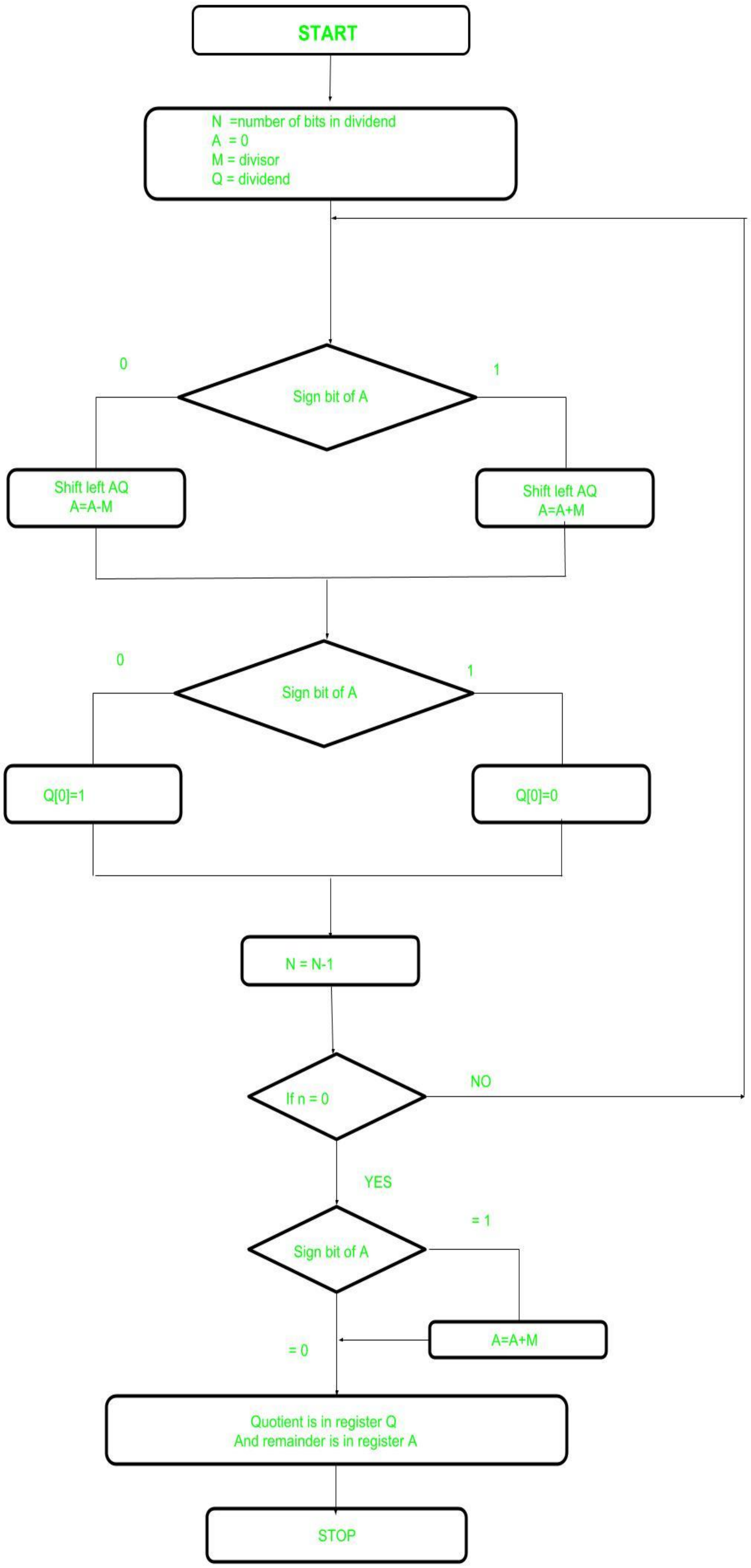
Non-Restoring Division for Unsigned Integer:

In Restoring Division we learned about restoring algorithm.

Now, here perform Non-Restoring division, it is less complex than the restoring one because simpler operations are involved i.e. addition and subtraction, also now restoring step is performed.

In the method, rely on the sign bit of the register which initially contain zero named as A.

Here is the flow chart given below.


START
N =number of bits in dividend
A = 0
M = divisor
Q = dividend
0
Sign bit of A
1
Shift left AQ
A=A-M
Shift left AQ
A=A+M
0
Sign bit of A
1
Q[0]=1
Q[0]=0
N = N-1
If n = 0
NO
YES
= 1
Sign bit of A
A=A+M
= 0
Quotient is in register Q
And remainder is in register A
STOP

Let’s pick the step involved:

- **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)
- **Step-2:** Check the sign bit of register A
- **Step-3:** If it is 1 shift left content of AQ and perform A = A+M, otherwise shift left AQ and perform A = A-M (means add 2’s complement of M to A and store it to A)
- **Step-4:** Again the sign bit of register A
- **Step-5:** If sign bit is 1 Q[0] become 0 otherwise Q[0] become 1 (Q[0] means least significant bit of register Q)
- **Step-6:** Decrements value of N by 1
- **Step-7:** If N is not equal to zero go to **Step 2** otherwise go to next step
- **Step-8:** If sign bit of A is 1 then perform A = A+M
- **Step-9:** Register Q contain quotient and A contain remainder

***Note: Problem on Non-Restoring algorithm is shared in YouTube video link.**

https://www.youtube.com/watch?v=f6A3ySUdT80