Design and Implementation of Slow and Fast Division Algorithms in Computer Architecture

Slow Division Algorithms Explored

- Restoring
  - This algorithm works by first subtracting the divisor from the dividend and then restoring the dividend to its original value if the subtraction did not result in a negative number.
  - The restoring division algorithm is simple to implement, but it is not very efficient.
- Non-Restoring
  - This algorithm works by subtracting the divisor from the dividend and then leaving the dividend in its negative state if the subtraction did not result in a negative number.
  - The non-restoring division algorithm is more efficient than the restoring division algorithm, but it is more difficult to implement.

Fast Division Algorithms Explored

- Newton-Raphson Method
  - This algorithm works by iteratively estimating the quotient of the dividend and the divisor.
  - The Newton-Raphson division algorithm is more efficient than the slow division algorithms, but it is more difficult to implement.
- Binary Method
  - This algorithm works by repeatedly dividing the dividend by 2 until the quotient is less than the divisor.
  - The binary division algorithm is simpler to implement than the Newton-Raphson division algorithm, but it is not as efficient.

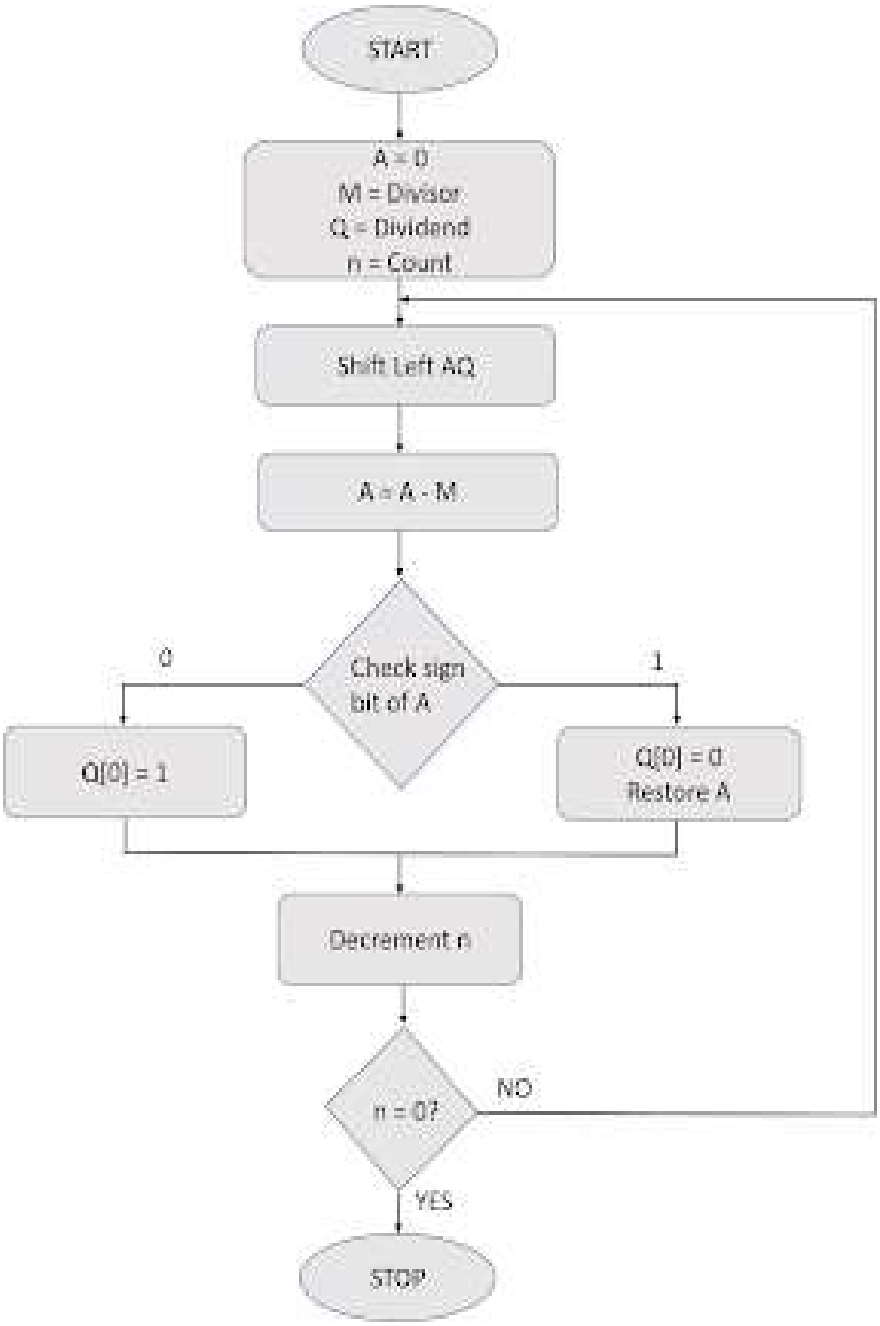
- Slow Division Algorithm (Restoring)

  Algorithm:

  Registers used: A, M, Q, n (counter)
  Step 1: Load the initial values for the registers.
  A = 0 (Accumulator), Qres = 0, M = Divisor, Q = Dividend and n is the count value which equals the number of bits of dividend.
  Step 2: Shift left {A,Q}.
  Step 3: Perform A = A - M.
  Step 4: Check the sign bit of A. If 0, goto step 5. If 1, goto step 6.

Step 5: Set LSB of Q as 0. Goto step 7.
Step 6: Set LSB of Q as 1. Restore the value of A which was present before the subtraction.
Step 7: Decrement count.
Step 8: Check if counter value n is zero. If yes, goto next step. Else, goto step 3.
Step 9: Stop

Flowchart:

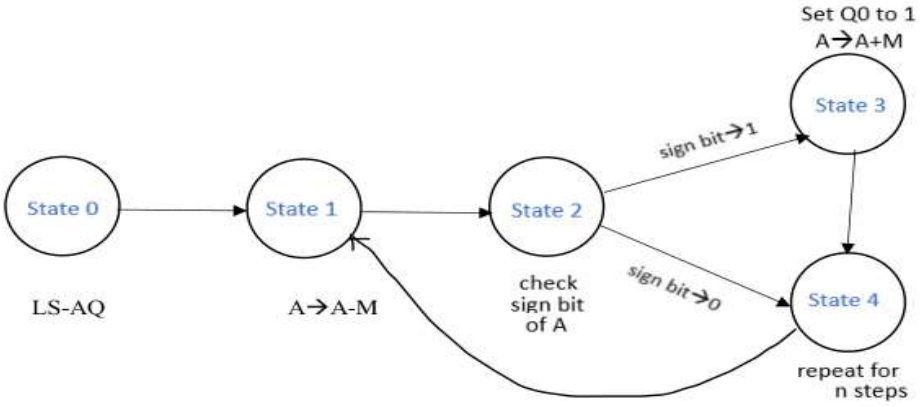


Example Operation:

Consider the following: Dividend = 15 (place in Q) and Divisor = 8 (place in M)
Count n = 4

Count	Operation	A	Q
4	Initial Values	0000	1111
	Shift Left A,Q	0001	1110
	A = A - M	1001	1110
	Q[0] = 0, Restore A	0001	1110
3	Shift Left A,Q	0011	1100
	A = A - M	1011	1100
	Q[0] = 0, Restore A	0011	1100
2	Shift Left A,Q	0111	1000
	A = A - M	1111	1000
	Q[0] = 0, Restore A	0111	1000
1	Shift Left A,Q	1111	0000
	A = A - M	0111	0000
	Q[0] = 1	0111	0001
0	Final Result: Remainder = 0111 = 7, Quotient = 0001 = 1		

FSM Diagram



A → ACCUMULATOR

Q → DIVIDEND

M → DIVISOR

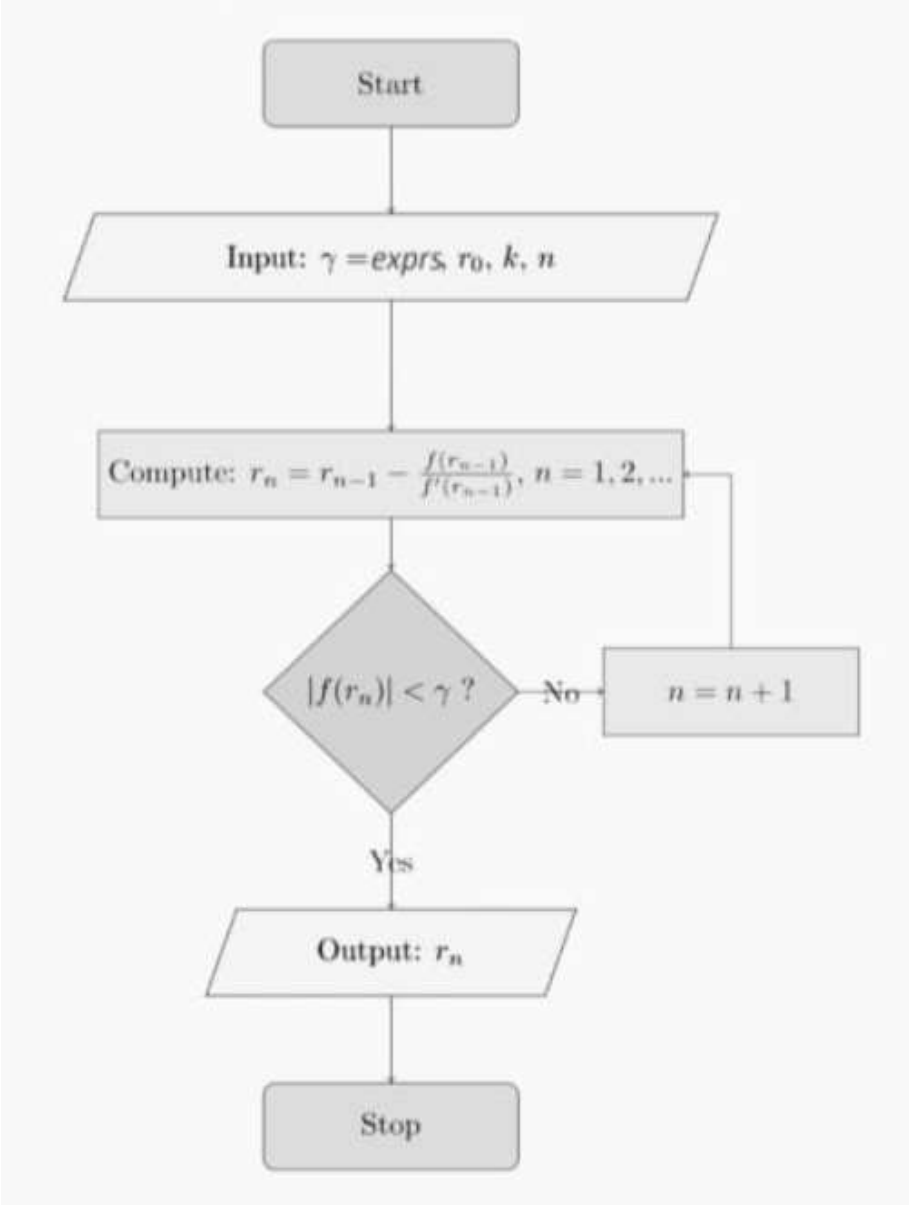
n → NUMBER OF BITS IN DIVISOR

- Fast Division Algorithm (Newton-Raphson)

Algorithm:

Step 1: Initialize the inputs - dividend, divisor and count(n=1,2,...).
Step 2: Compute r(n) using the formula.
Step 3: If the value of |f(r(n))| is the most closest value to the dividend, displaying the value which is the output. If not, increment the value of count.
Step 4: Repeat the steps 3 and 4 until the condition is met.
Step 5: The output should display the corresponding quotient and remainder.
Step 6: Check the results.

Flowchart:



Example Operation:

Consider the following: Dividend: 10 (in binary: 1010) Divisor: 3 (in binary: 0011)

Step 1: Initialize the variables:
x0 = 0
x1 = 0
x2 = 0
x3 = 0
x4 = 0

Step 2: Assign the values of the dividend and divisor to x0 and divisor, respectively:
x0 = 1010 (dividend)
divisor = 0011 (divisor)

Step 3: Calculate x1:
x1 = x0 - (x0 / divisor) * divisor
x1 = 1010 - (1010 / 0011) * 0011
x1 = 1010 - (5 * 0011)
x1 = 1010 - 0110
x1 = 0100

Step 4: Calculate x2:
x2 = x1 - (x1 / divisor) * divisor
x2 = 0100 - (0100 / 0011) * 0011
x2 = 0100 - (2 * 0011)
x2 = 0100 - 0110
x2 = 1110 (carry out of MSB)

Step 5: Calculate x3:
x3 = x2 - (x2 / divisor) * divisor
x3 = 1110 - (1110 / 0011) * 0011
x3 = 1110 - (6 * 0011)
x3 = 1110 - 1101
x3 = 0001

Step 6: Calculate x4 (quotient):
x4 = x0 / divisor
x4 = 1010 / 0011
x4 = 0011

The final results are:
quotient = 0011 (3 in decimal)
remainder = 0001 (1 in decimal)
So, when dividing the dividend 10 by the divisor 3 using the provided module, the quotient is 3, and the remainder is 1, which matches the expected results.

FSM

Since the Newton-Raphson division algorithm does not lend itself directly to a finite state machine (FSM) representation, it is not possible to create a traditional FSM diagram for it. The Newton-Raphson division algorithm is an iterative numerical method that relies on repeated calculations and comparisons until a desired level of precision is achieved.

FSMs are typically used to model systems with a finite number of states and discrete transitions between those states based on input conditions. In the case of the Newton-Raphson division algorithm, the calculation process involves a loop with iterative steps, rather than discrete states and transitions.

Instead of an FSM diagram, it would be more appropriate to represent the Newton-Raphson division algorithm as a flowchart or a procedural description, which can illustrate the step-by-step process involved in the algorithm's execution.

Validation Methods for Slow and Fast Division Algorithms

- Restoring Method
    - The restoring division algorithm can be validated by checking that the remainder is 0 after the division is complete.
- Newton-Raphson Method
    - The Newton-Raphson division algorithm can be validated by comparing the estimated quotient to the actual quotient.

Conclusion

This documentation provides a comprehensive overview of the slow and fast division algorithms that were explored in this project. The flowcharts and input/output table provide a visual representation of how the algorithms work, and the example problems demonstrate how the algorithms can be used to divide two numbers. The validation methods for the slow and fast division algorithms provide a way to verify that the algorithms are producing the correct results.