Lab Report

On

Restoring and Non-Restoring Division of Two Four-Bit Integer Numbers

(Computer Organization and Architecture)

By

Aadarsha Bhusal

080BEI001

BEI II/II

To

Er. Sujan Karki

Department of Electronics and Computer Engineering

Institute of Engineering

Purwanchal Campus, Dharan

Tribhuvan University

July 24, 2025

1. Restoring Division of Two Four-Bit Integer Numbers

1.1. Introduction

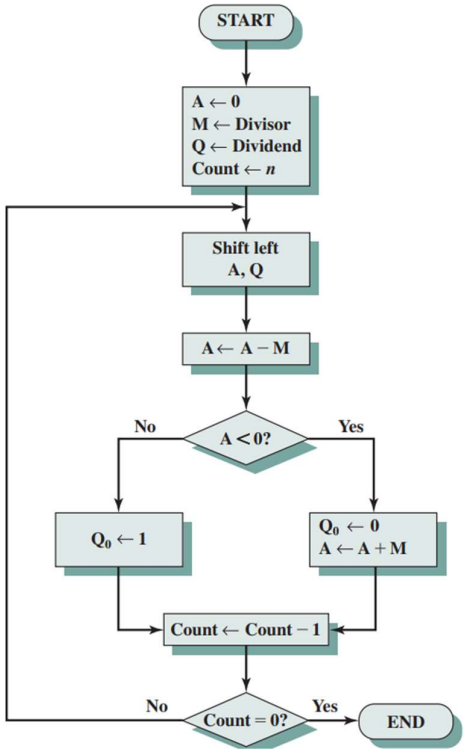
Division is somewhat more complex than multiplication, but is based on the same general principles. For Restoring Division, the divisor is placed in the M register, the dividend in the Q register. At each step, the A and Q registers together are shifted to the left 1 bit. M is subtracted from A to determine whether A divides the partial remainder. If it does, then Q_0 gets a 1 bit. Otherwise, Q_0 gets a 0 bit and M must be added back to A to restore the previous value. The count is then decremented, and the process continues for n steps. At the end, the quotient is in the Q register and the remainder is in the A register.

1.2. Objective

- To design a restoring division algorithm circuit in *Verilog.*
- To test and visualize the simulation using *Icarus Verilog 12.0* and *EPWave.*

1.3. Algorithm

1. Load the twos complement of the divisor into the M register; that is, the M register contains the negative of the divisor. Load the dividend into the A, Q registers.
2. Shift A, Q left 1-bit position.
3. Perform A = A − M. This operation subtracts the divisor from the contents of A.
4. If clause:
    a. If the result is nonnegative (most significant bit of A = 0), then set $Q_0 = 1$.
    b. If the result is negative (most significant bit of A = 1), then set $Q_0 = 0$. and restore the previous value of A.
5. Repeat steps 2 through 4 as many times as there are bit positions in Q.
6. The remainder is in A and the quotient is in Q.



1.4. Design

The design code contains a *restoring_ division_ 4bit* module which takes in *dividend* and *divisor* as input and *quotient* and *remainder* as output. The code implies the above given algorithm in Verilog.

```
module restoring_division_4bit(dividend, divisor, quotient,
remainder);
  input [3:0] dividend;
  input [4:0] divisor;
  output reg [3:0] quotient;
  output reg [4:0] remainder;

  reg [4:0] accumulator;
  reg [4:0] neg_divisor;

  assign neg_divisor = ~divisor + 1;

  always @* begin
    accumulator = 5'b0;
    quotient = dividend;
    remainder = 5'b0;

    for (int i = 0; i < 4; i = i + 1) begin
      {accumulator, quotient} = {accumulator, quotient} <<
1;
      accumulator = accumulator + neg_divisor;

      if (accumulator[4]) begin
        quotient[0] = 1'b0;
        accumulator = accumulator + divisor;
      end
      else begin
        quotient[0] = 1'b1;
      end
    end
    remainder = accumulator;
  end
endmodule
```

1.5. Test Bench

The test bench initializes the design module with registers and wires. The simulation is initialized where any changes in variables are dumped into a *"dump.vcd"* file and the dump variable hierarchy is set to 1. We gave two different test cases and displayed the result on console as well as graph.

```
module test_bench;
  reg [3:0] dividend;
```

```
  reg [4:0] divisor;
  wire [3:0] quotient;
  wire [4:0] remainder;

  restoring_division_4bit dut(dividend, divisor, quotient,
remainder);

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);

    dividend = 4'd10;
    divisor = 4'd5;
    #10
    $display("dividend=%d, divisor=%d, quotient=%d,
remainder=%d", dividend, divisor, quotient, remainder);

    dividend = 4'd8;
    divisor = 4'd6;
    #10
    $display("dividend=%d, divisor=%d, quotient=%d,
remainder=%d", dividend, divisor, quotient, remainder);
  end
endmodule
```

1.6. Results

The output on the console was printed as:

```
dividend=10, divisor= 5, quotient= 2, remainder= 0

dividend= 8, divisor= 6, quotient= 1, remainder= 2
```

The *EPWave* result was displayed as:

In the above simulation graph, we obtain the correct result for the first test case and it takes 10 time units. Similarly, we obtain the correct result for the second test case which also takes 10 time units.

1.7. Conclusion

The restoring division algorithm circuit designed using *Verilog* and simulated using *EDAPlayground* correctly outputs the desired result. We display the result both on the console and *EPWave* graphical representation.

2. Non-Restoring Division of Two Four-Bit Integer Numbers

2.1. Introduction

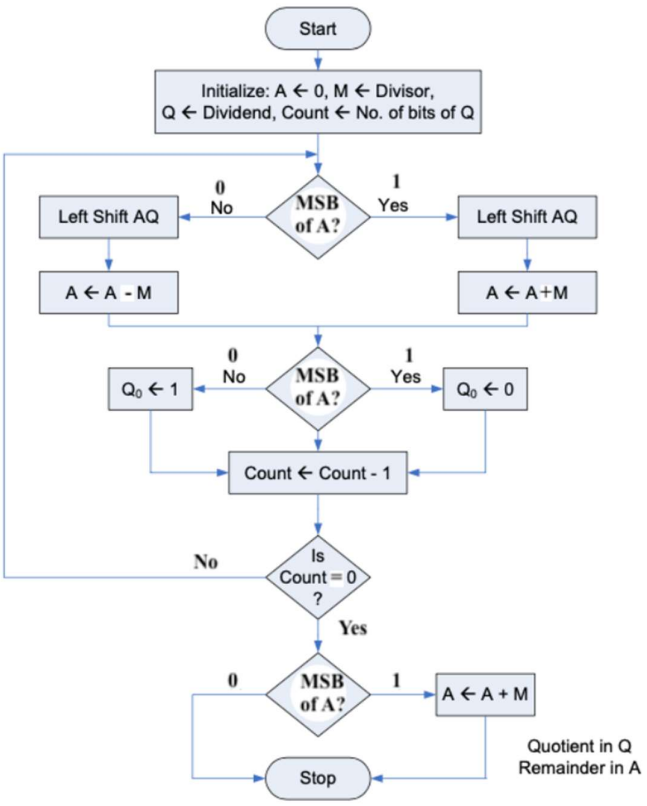
The non-restoring division is a division technique for unsigned binary values that simplifies the procedure by eliminating the restoring phase. The non-restoring division is simpler and more effective than restoring division since it just employs addition and subtraction operations instead of restoring division, which requires extra steps to restore the original result after a failed subtraction.

2.2. Objective

- To implement non-restoring division algorithm in *Verilog*.
- To test and visualize the circuit using *EPWave* and *EDAPlayground*.

2.3. Algorithm

1. Load the (positive) divisor into M, load the dividend into Q, and set A = 0.
2. Shift the pair A, Q left by 1 bit.
3. If the MSB of A = 0 then A = A − M; otherwise A = A + M.
4. If the MSB of A = 0 then set $Q_0 = 1$; otherwise set $Q_0 = 0$.
5. Repeat steps 2–4 for as many bits as Q contains.
6. If the final MSB of A = 1, then A = A + M (to correct the remainder).
7. At the end, Q holds the quotient and A holds the (nonnegative) remainder.



2.4. Design

The design code contains a *non_restoring_division_4bit* module which takes in *dividend* and *divisor* as input and *quotient* and *remainder* as output. The code implies the above given algorithm in Verilog.

```
module non_restoring_division_4bit(dividend, divisor,
quotient, remainder);
  input [3:0] dividend;
  input [4:0] divisor;
  output reg [3:0] quotient;
  output reg [4:0] remainder;

  reg [4:0] accumulator;
  reg [4:0] neg_divisor;

  assign neg_divisor = ~divisor + 1;

  always @* begin
    accumulator = 5'b0;
    quotient = dividend;
    remainder = 5'b0;

    for (int i = 0; i < 4; i = i + 1) begin
      if (accumulator[4]) begin
        {accumulator, quotient} = {accumulator, quotient}
<< 1;
        accumulator = accumulator + divisor;
      end
      else begin
        {accumulator, quotient} = {accumulator, quotient}
<< 1;
        accumulator = accumulator - divisor;
      end
      if (accumulator[4]) begin
        quotient[0] = 1'b0;
      end
      else begin
        quotient[0] = 1'b1;
      end
    end
    if (accumulator[4]) begin
      accumulator = accumulator + divisor;
    end
    remainder = accumulator;
  end
endmodule
```

2.5. Test Bench

The test bench initializes the design module with registers and wires. The simulation is initialized where any changes in variables are dumped into a *"dump.vcd"* file and the dump variable hierarchy is set to 1. We gave two different test cases and displayed the result on console as well as graph.

```
module test_bench;
  reg [3:0] dividend;
  reg [4:0] divisor;
  wire [3:0] quotient;
  wire [4:0] remainder;

  non_restoring_division_4bit dut(dividend, divisor,
quotient, remainder);

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);

    dividend = 4'd1;
    divisor = 4'd2;
    #10
    $display("dividend=%d, divisor=%d, quotient=%d,
remainder=%d", dividend, divisor, quotient, remainder);

    dividend = 4'd9;
    divisor = 4'd7;
    #10
    $display("dividend=%d, divisor=%d, quotient=%d,
remainder=%d", dividend, divisor, quotient, remainder);
  end
endmodule
```

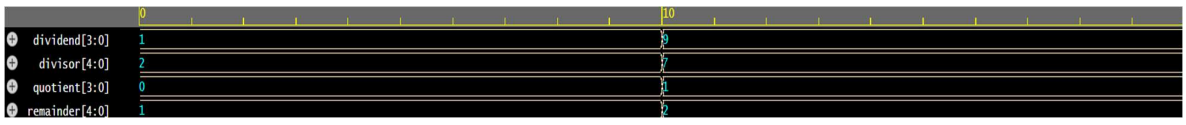
2.6. Result

The output on the console was printed as:

```
dividend= 1, divisor= 2, quotient= 0, remainder= 1

dividend= 9, divisor= 7, quotient= 1, remainder= 2
```

The *EPWave* result was displayed as:



In the above simulation graph, we obtain the correct result for the first test case and it takes 10 time units. Similarly, we obtain the correct result for the second test case which also takes 10 time units.

2.7. Conclusion

The non-restoring division algorithm circuit designed using *Verilog* and simulated using *EDAPlayground* correctly outputs the desired result. We display the result both on the console and *EPWave* graphical representation.