Lab Report

On

Division Using Restoring Algorithm

Computer Organization and Architecture

By

Spandan Guragain
PUR078BCT086

To

Mr. Sujan Karki

Tribhuwan University

Institute of Engineering

Faculty of Computer and Eletronics Engineering

Purwanchal Campus

Dharan, Nepal

2024-06-30

1 Division Using Restoring Algorithm

1.1 Introduction

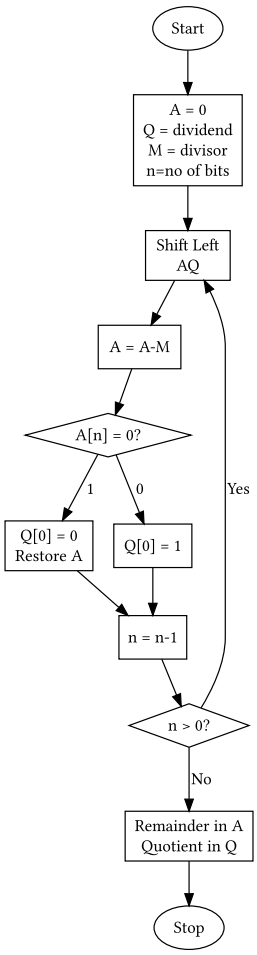
Restoring division is usually performed on the fixed point fractional numbers. When we perform division operations on two numbers, the division algorithm will give us two things, i.e., quotient and remainder.

1.2 Objective

To design and implement an unsigned binary divider using the restoring algorithm in SystemVerilog and to understand its workings.

1.3 Algorithm

1. Initialize registers A and Q to zero, and M to the divisor. Load the dividend into Q. Set counter to the number of bits in the dividend (n).
2. Shift A and Q left by one bit.
3. Subtract M from A, placing the result back in A.
   - If the A[n] is 1, set Q[0] to 0 and restore A by adding M back.
   - If the A[n] is 0, set Q[0] to 1.
4. Decrease the counter by 1. If the counter is greater than 0, repeat from step 2. Otherwise, stop.
5. The final remainder will be in A and the quotient in Q.



1.4 Design

The RestoringDivider module performs 4-bit unsigned division using the restoring algorithm. The module takes a 4-bit dividend and a 5-bit divisor as inputs and outputs a 4-bit quotient and a 5-bit remainder.

```
module RestoringDivider(
    input [3:0] dividend, // Dividend (4-bit)
    input [4:0] divisor,  // Divisor (5-bit)
    output reg [3:0] quotient, // Quotient (4-bit)
    output reg [4:0] remainder // Remainder (5-bit)
);

reg [4:0] accumulator;
reg [4:0] neg_divisor; // Negative of divisor

// Initialize variables
assign neg_divisor = ~divisor + 1;

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

    // Division using Restoring algorithm
    for (int i = 0; i < 4; i = i + 1) begin
        accumulator = {accumulator[3:0], quotient[3]};
        quotient = {quotient[2:0], 1'b0};
        accumulator = accumulator + neg_divisor;

        if (!accumulator[4]) begin
            quotient[0] = 1'b1; // Set quotient bit
        end else begin
            quotient[0] = 1'b0; // Clear quotient bit
            accumulator = accumulator + divisor;
        end
    end

    remainder = accumulator;
end

endmodule
```

1.5 Testbench

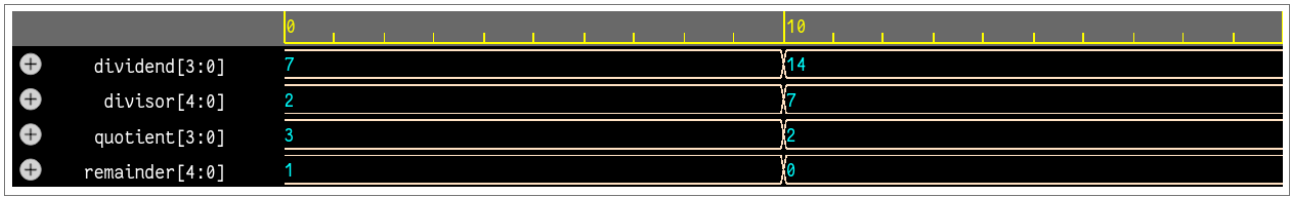
The Testbench module is used to verify the functionality of the RestoringDivider4Bit module. It initializes various test cases, simulates the division, and displays the results.

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

```
    RestoringDivider divider (dividend, divisor, quotient, remainder);

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

        // Test case 1: Divide 7 by 2
        dividend = 4'b0111;
        divisor = 5'b00010;
        #10;
        // Display inputs and outputs
        $display("%d/%d = %d and remainder = %d"
               , dividend, divisor, quotient, remainder);

        // Test case 2: Divide 14 by 7
        dividend = 4'b1110;
        divisor = 5'b00111;
        #10;
        // Display inputs and outputs
        $display("%d/%d = %d and remainder = %d"
               , dividend, divisor, quotient, remainder);
    end
endmodule
```

1.6 Results

The simulation was conducted in EDA Playground. The results were as follows:



Output:

```
# KERNEL:   7/ 2 =  3 and remainder =  1
# KERNEL:  14/ 7 =  2 and remainder =  0
```

The results are the expected outputs.

1.7 Conclusion

The `RestoringDivider` module accurately performs unsigned binary division using the restoring algorithm in SystemVerilog. The testbench confirms its correctness through various test cases, demonstrating reliable computation of quotient and remainder.