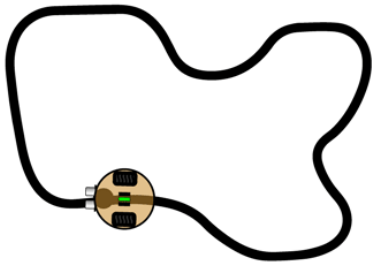
EXPERIMENT- 6
Control Path Modelling and Design

Objective:

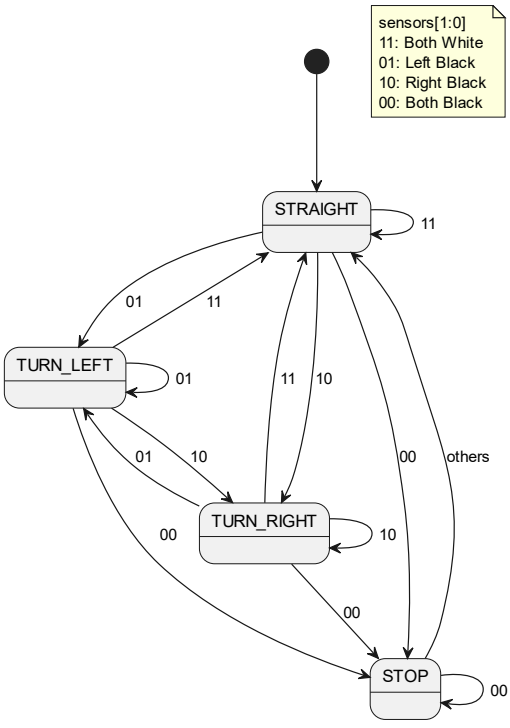
To understand the concepts related to FSM (Moore/Mealy machine) and design the controller for a robot with the given specifications. Write the Verilog Programs for the same using any modeling style.

Exercise Problems

1. **A Line follower Robot uses 2 Infra-Red (IR) sensors to detect the colour of the surface. Sensor output is High when white surface is detected and Low on a black surface. Try to model the Robot controller and then implement its hardware?**



State Diagram



Source Code

```
module linefollower (
    input wire clk, reset,
    input wire sensorLeft,sensorRight,// 1:White surface, 0:Black surface
    output reg motorLeft, motorRight
);
parameter [3:0]
        STRAIGHT   = 4'b0001,
        TURN_LEFT  = 4'b0010,
        TURN_RIGHT = 4'b0100,
        STOP       = 4'b1000;

    reg [3:0] current_state, next_state;
    wire [1:0] sensors;
    assign sensors = {sensorLeft, sensorRight};
    always @(posedge clk or posedge reset) begin
        if (reset)
            current_state <= STRAIGHT;
        else
            current_state <= next_state;
    end
    always @(*) begin
        case (current_state)
            STRAIGHT: begin
                case (sensors)
                    2'b11:   next_state = STRAIGHT;
                    2'b01:   next_state = TURN_LEFT;
                    2'b10:   next_state = TURN_RIGHT;
                    2'b00:   next_state = STOP;
                    default: next_state = STRAIGHT;
                endcase
            end
            TURN_LEFT: begin
                case (sensors)
                    2'b01:   next_state = TURN_LEFT;
                    2'b10:   next_state = TURN_RIGHT;
                    2'b11:   next_state = STRAIGHT;
                    2'b00:   next_state = STOP;
                    default: next_state = TURN_LEFT;
                endcase
            end
```

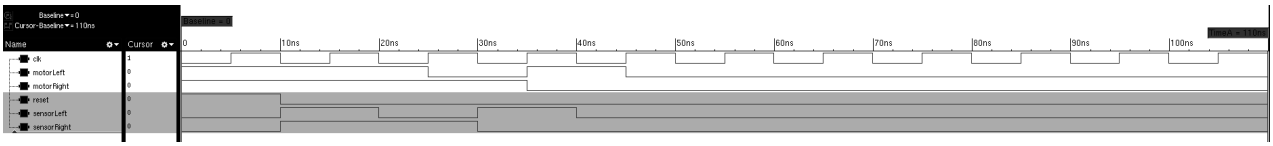
```
            TURN_RIGHT: begin
                case (sensors)
                    2'b10:   next_state = TURN_RIGHT;
                    2'b01:   next_state = TURN_LEFT;
                    2'b11:   next_state = STRAIGHT;
                    2'b00:   next_state = STOP;
                    default: next_state = TURN_RIGHT;
                endcase
            end
            STOP: begin
                next_state = (sensors == 2'b00) ? STOP : STRAIGHT;
            end
            default: begin
                next_state = STRAIGHT;
            end
        endcase
    end
    always @(*) begin
        case (current_state)
            STRAIGHT:   {motorLeft, motorRight} = 2'b11;
            TURN_LEFT:  {motorLeft, motorRight} = 2'b01;
            TURN_RIGHT: {motorLeft, motorRight} = 2'b10;
            STOP:       {motorLeft, motorRight} = 2'b00;
            default:    {motorLeft, motorRight} = 2'b11;
        endcase
    end
endmodule
```

Testbench

```
module tb();
  reg clk, reset;
  reg sensorLeft, sensorRight;
  wire motorLeft, motorRight;
  linefollower uut (  .clk(clk),
                      .reset(reset),
                      .sensorLeft(sensorLeft),
                      .sensorRight(sensorRight),
                      .motorLeft(motorLeft),
                      .motorRight(motorRight)
                    );
```

```
  always #5 clk = ~clk;
  initial begin
$monitor($time," | clk = %b | reset = %b | sensorLeft = %b | sensorRight
= %b | motorLeft = %b | motorRight = %b",
clk,reset,sensorLeft,sensorRight,motorLeft,motorRight);
    clk=0;
    reset = 0;
    sensorLeft = 0;
    sensorRight = 0;
    reset = 1; #10;
    reset = 0;
    sensorLeft = 1;
    sensorRight = 1; #10; // Both sensors detect white
    sensorLeft = 0;
    sensorRight = 1; #10; // Left sensor detects black
    sensorLeft = 1;
    sensorRight = 0;#10; // Right sensor detects black
    sensorLeft = 0;
    sensorRight = 0;#10; // Both sensors detect black
    #60;
    $finish;
  end
endmodule
```

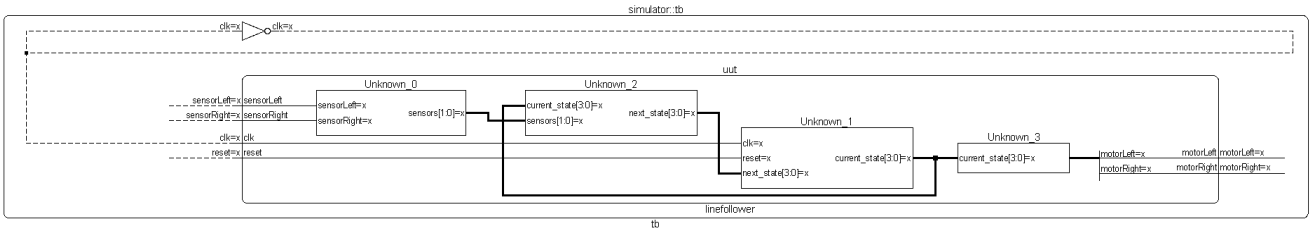
Waveform



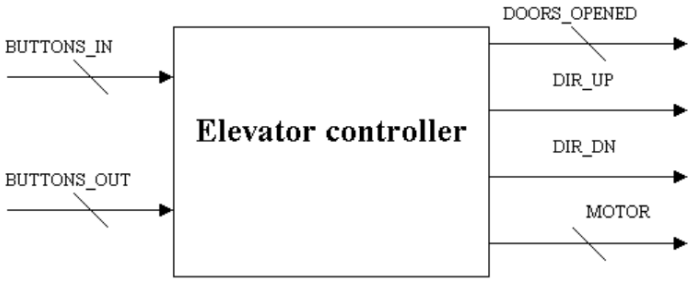
Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                   0 | clk = 0 | reset = 1 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 1 | motorRight = 1
                   5 | clk = 1 | reset = 1 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 1 | motorRight = 1
                  10 | clk = 0 | reset = 0 | sensorLeft = 1 | sensorRight = 1 | motorLeft = 1 | motorRight = 1
                  15 | clk = 1 | reset = 0 | sensorLeft = 1 | sensorRight = 1 | motorLeft = 1 | motorRight = 1
                  20 | clk = 0 | reset = 0 | sensorLeft = 0 | sensorRight = 1 | motorLeft = 1 | motorRight = 1
                  25 | clk = 1 | reset = 0 | sensorLeft = 0 | sensorRight = 1 | motorLeft = 0 | motorRight = 1
                  30 | clk = 0 | reset = 0 | sensorLeft = 1 | sensorRight = 0 | motorLeft = 0 | motorRight = 1
                  35 | clk = 1 | reset = 0 | sensorLeft = 1 | sensorRight = 0 | motorLeft = 1 | motorRight = 0
                  40 | clk = 0 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 1 | motorRight = 0
                  45 | clk = 1 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                  50 | clk = 0 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                  55 | clk = 1 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                  60 | clk = 0 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                  65 | clk = 1 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                  70 | clk = 0 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                  75 | clk = 1 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                  80 | clk = 0 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                  85 | clk = 1 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                  90 | clk = 0 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                  95 | clk = 1 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                 100 | clk = 0 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
                 105 | clk = 1 | reset = 0 | sensorLeft = 0 | sensorRight = 0 | motorLeft = 0 | motorRight = 0
Simulation complete via $finish(1) at time 110 NS + 0
./linefollower_tb.v:31      $finish;\r
ncsim>
```

Schematic



2. Model and implement an Elevator control system handling 4 floors (0 to 3)



Source Code

```
module elevator(
    input clk,
    input reset,
    input [1:0] current_floor,
    input [3:0] buttons_in,
    input [3:0] buttons_out,
    output reg [3:0] motor,
    output reg dir_up,
    output reg dir_down
  );
  reg [3:0] floor_requests;
  reg [1:0] next_floor;
  always @(posedge clk or posedge reset)
  begin
    if (reset)
    begin
      motor <= 4'b0000;
      dir_up <= 1'b0;
      dir_down <= 1'b0;
      next_floor <= current_floor;
      floor_requests <= 4'b0000;
    end
    else
```

```
    begin
      floor_requests <= buttons_in | buttons_out;
      if (floor_requests[current_floor])
      begin
        motor <= 4'b0000;  // Stop at current floor
        dir_up <= 1'b0;
        dir_down <= 1'b0;
      end
      else
      begin
        if (current_floor == 2'b00 && floor_requests[1])
        begin
          next_floor <= 2'b01;
          motor <= 4'b0010;
          dir_up <= 1'b1;
          dir_down <= 1'b0;
        end
        else if (current_floor <= 2'b01 && floor_requests[2])
        begin
          next_floor <= 2'b10;
          motor <= 4'b0100;
          dir_up <= 1'b1;
          dir_down <= 1'b0;
        end
        else if (current_floor <= 2'b10 && floor_requests[3])
        begin
          next_floor <= 2'b11;
          motor <= 4'b1000;
          dir_up <= 1'b1;
          dir_down <= 1'b0;
        end
        else if (current_floor >= 2'b01 && floor_requests[0])
        begin
          next_floor <= 2'b00;
          motor <= 4'b0001;
          dir_up <= 1'b0;
          dir_down <= 1'b1;
        end
        else if (current_floor >= 2'b10 && floor_requests[1])
        begin
          next_floor <= 2'b01;
          motor <= 4'b0010;
```

```
          dir_up <= 1'b0;
          dir_down <= 1'b1;
        end
        else if (current_floor >= 2'b11 && floor_requests[2])
        begin
          next_floor <= 2'b10;
          motor <= 4'b0100;
          dir_up <= 1'b0;
          dir_down <= 1'b1;
        end
        else
        begin
          motor <= 4'b0000; // No movement if no requests in any
direction
          dir_up <= 1'b0;
          dir_down <= 1'b0;
        end
      end
    end
  end
endmodule
```

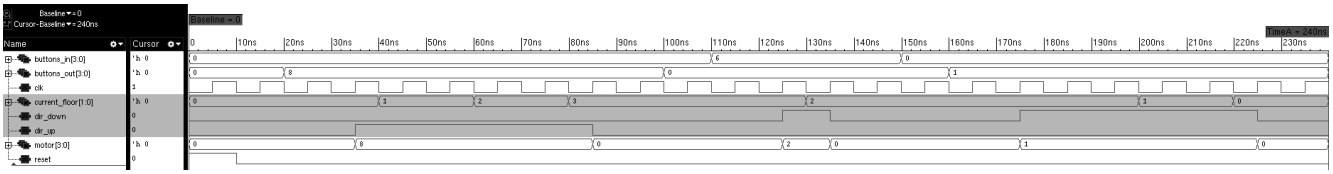
Testbench

```
module tb();
  reg clk;
  reg reset;
  reg [1:0] current_floor;
  reg [3:0] buttons_in;
  reg [3:0] buttons_out;
  wire [3:0] motor;
  wire dir_up;
  wire dir_down;


  elevator uut (
              .clk(clk),
              .reset(reset),
              .current_floor(current_floor),
              .buttons_in(buttons_in),
              .buttons_out(buttons_out),
              .motor(motor),
```

```
            .dir_up(dir_up),
            .dir_down(dir_down)
          );


  always #5 clk = ~clk;

  initial begin
$monitor($time," |Floor=%d | Buttons In=%b | Buttons Out=%b | Motor=%b |
Dir_Up=%b | Dir_Down=%b", current_floor, buttons_in, buttons_out, motor,
dir_up, dir_down);
    clk = 0;
    reset = 1;
    current_floor = 2'b00;
    buttons_in = 4'b0000;
    buttons_out = 4'b0000;
    #10;
    reset = 0;
    // Request to go to the 3rd floor (external)
    #10;
    buttons_out = 4'b1000;  // Request for 3rd floor
    #20;
    current_floor = 2'b01;  // Move to 1st floor
    #20;
    current_floor = 2'b10;  // Move to 2nd floor
    #20;
    current_floor = 2'b11;  // Reach 3rd floor
    #20;
    buttons_out = 4'b0000;  // Clear requests

    // Request from 1st and 2nd floors (internal)
    #10;
    buttons_in = 4'b0110;   // Requests for 1st and 2nd floors
    #20;
    current_floor = 2'b10;  // Move to 2nd floor
    #20;
    buttons_in = 4'b0000;   // Clear requests

    // Request to go down to ground floor (external)
    #10;
    buttons_out = 4'b0001;  // Request for ground floor
    #20;
```

```
    current_floor = 2'b10;  // Move to 2nd floor
    #20;
    current_floor = 2'b01;  // Move to 1st floor
    #20;
    current_floor = 2'b00;  // Reach ground floor
    #20;
    buttons_out = 4'b0000;  // Clear requests
    $finish;
  end
endmodule
```

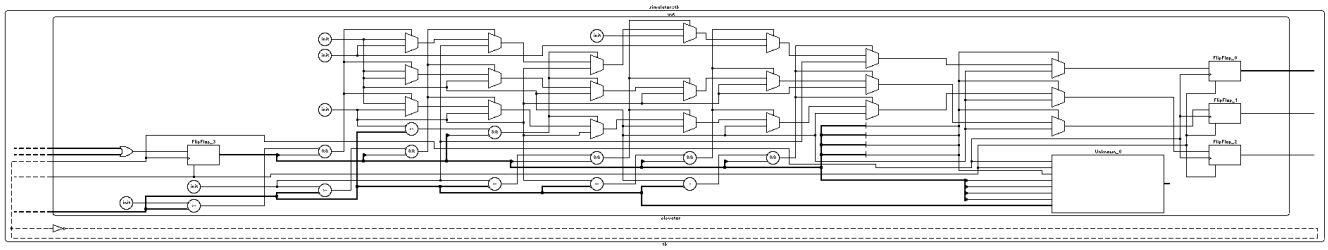
Waveform



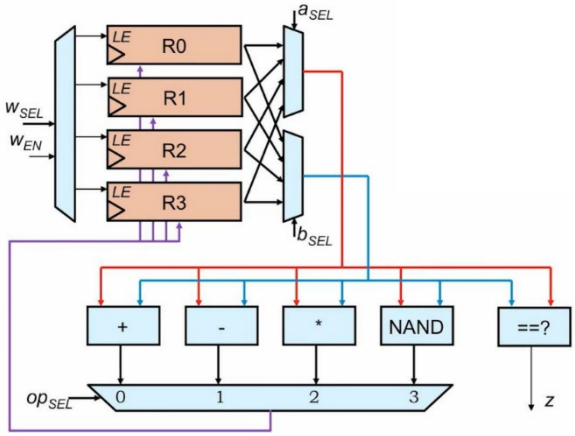
Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> database -open waves -into waves.shm -default
probe -create -shm tb.buttons_in tb.buttons_out tb.clk tb.current_floor tb.dir_down tb.dir_up tb.motor tb.reset
Created default SHM database waves
ncsim> Created probe 1
ncsim> run
                   0 |Floor=0 | Buttons In=0000 | Buttons Out=0000 | Motor=0000 | Dir_Up=0 | Dir_Down=0
                  20 |Floor=0 | Buttons In=0000 | Buttons Out=1000 | Motor=0000 | Dir_Up=0 | Dir_Down=0
                  35 |Floor=0 | Buttons In=0000 | Buttons Out=1000 | Motor=1000 | Dir_Up=1 | Dir_Down=0
                  40 |Floor=1 | Buttons In=0000 | Buttons Out=1000 | Motor=1000 | Dir_Up=1 | Dir_Down=0
                  60 |Floor=2 | Buttons In=0000 | Buttons Out=1000 | Motor=1000 | Dir_Up=1 | Dir_Down=0
                  80 |Floor=3 | Buttons In=0000 | Buttons Out=1000 | Motor=1000 | Dir_Up=1 | Dir_Down=0
                  85 |Floor=3 | Buttons In=0000 | Buttons Out=1000 | Motor=0000 | Dir_Up=0 | Dir_Down=0
                 100 |Floor=3 | Buttons In=0000 | Buttons Out=0000 | Motor=0000 | Dir_Up=0 | Dir_Down=0
                 110 |Floor=3 | Buttons In=0110 | Buttons Out=0000 | Motor=0000 | Dir_Up=0 | Dir_Down=0
                 125 |Floor=3 | Buttons In=0110 | Buttons Out=0000 | Motor=0010 | Dir_Up=0 | Dir_Down=1
                 130 |Floor=2 | Buttons In=0110 | Buttons Out=0000 | Motor=0010 | Dir_Up=0 | Dir_Down=1
                 135 |Floor=2 | Buttons In=0110 | Buttons Out=0000 | Motor=0000 | Dir_Up=0 | Dir_Down=0
                 150 |Floor=2 | Buttons In=0000 | Buttons Out=0000 | Motor=0000 | Dir_Up=0 | Dir_Down=0
                 160 |Floor=2 | Buttons In=0000 | Buttons Out=0001 | Motor=0000 | Dir_Up=0 | Dir_Down=0
                 175 |Floor=2 | Buttons In=0000 | Buttons Out=0001 | Motor=0001 | Dir_Up=0 | Dir_Down=1
                 200 |Floor=1 | Buttons In=0000 | Buttons Out=0001 | Motor=0001 | Dir_Up=0 | Dir_Down=1
                 220 |Floor=0 | Buttons In=0000 | Buttons Out=0001 | Motor=0001 | Dir_Up=0 | Dir_Down=1
                 225 |Floor=0 | Buttons In=0000 | Buttons Out=0001 | Motor=0000 | Dir_Up=0 | Dir_Down=0
Simulation complete via $finish(1) at time 240 NS + 0
./elevator_tb.v:66      $finish;\r
ncsim>
```

Schematic



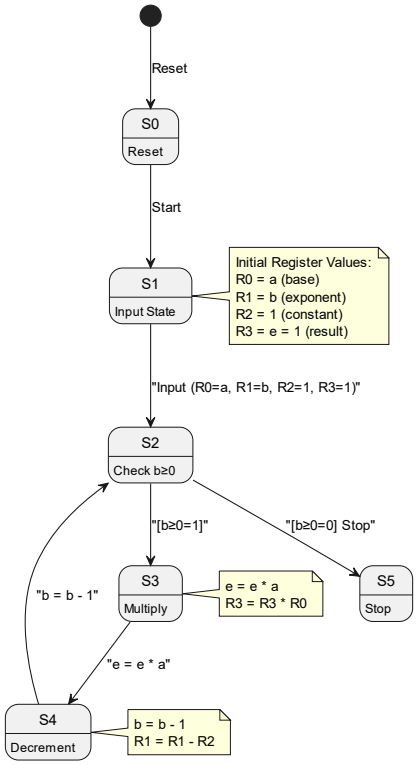
3. Design the control flow and control signals required to configure this programmable data path given in the figure to perform Exponentiation function.



Pseudocode

```
e = 1;
while (b >= 1) {
    e = e * a;
    b = b - 1;
}
```

State Diagram



Control Signal

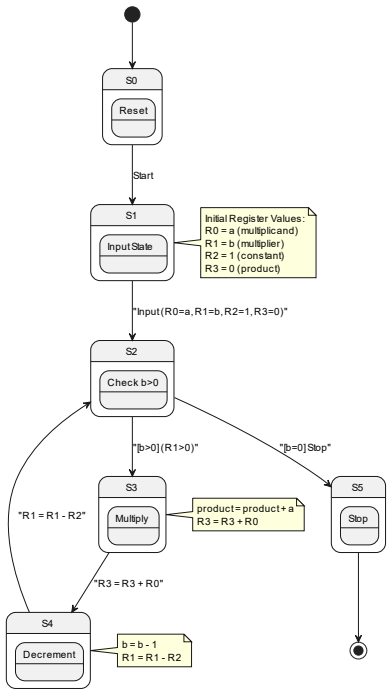
State	a_sel	b_sel	op_sel	wsel	wen
S0	10	10	10	1	1
S1	10	10	10	1	1
S2	10	01	10	0	1
S3	11	00	10	1	1
S4	01	10	01	0	1
S5	10	10	10	1	0

4. Design the state diagram and control path for programmable data path given in Q No.3 to perform any of (a) Multiplication (b) Division (c) Square root (d) factorial

Multiplication -- Pseudocode

```
p = 0;
while (b >= 1) {
    p = p + a;
    b = b - 1;
}
```

State Diagram



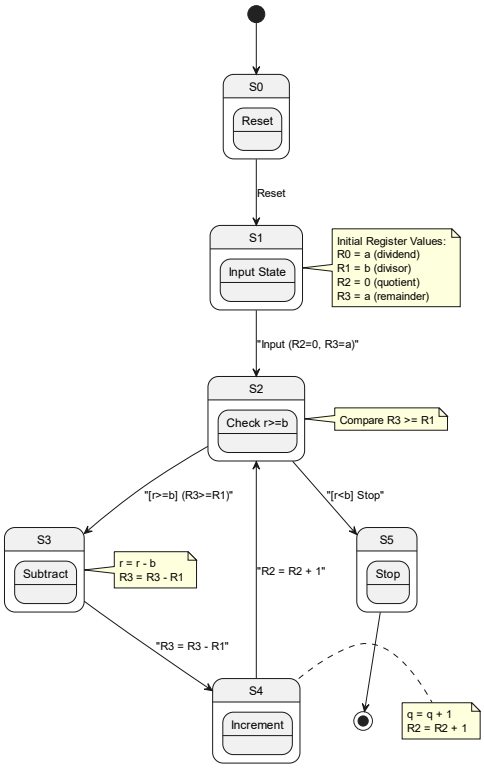
Control Signal

State	a_sel	b_sel	op_sel	wsel	wen
S0	10	10	10	1	1
S1	10	10	10	1	1
S2	10	01	10	0	1
S3	11	00	00	1	1
S4	01	10	01	0	1
S5	10	10	10	1	0

Division -- Pseudocode

```
q = 0;
r = a;
while (r >= b) {
    r = r - b;
    q = q - 1;
}
```

State Diagram



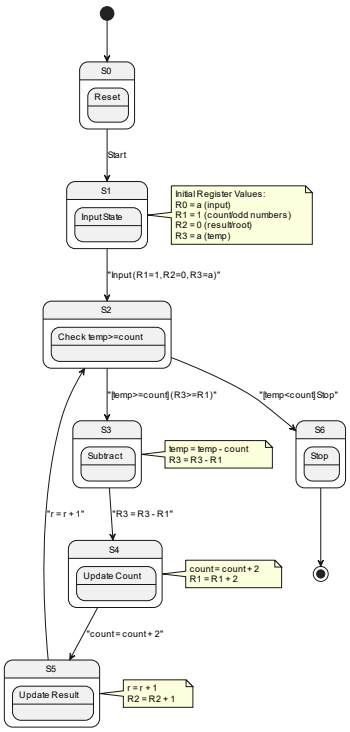
Control Signal

State	a_sel	b_sel	op_sel	wsel	wen
S0	10	10	10	1	1
S1	10	10	10	1	1
S2	10	01	10	0	1
S3	11	00	01	1	1
S4	01	10	00	0	1
S5	10	10	10	1	0

Square root -- Pseudocode

```
r = 0;
temp = a;
count= 1;
while (temp >= count) {
    temp = temp - count;
    count = count + 2;
    r = r + 1;
}
```

State Diagram



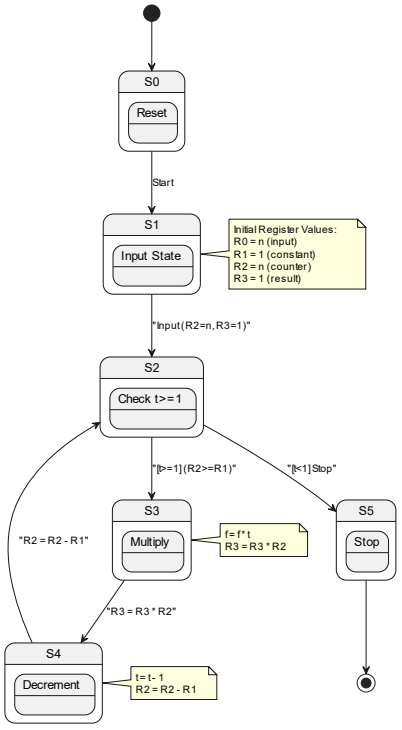
Control Signal

State	a_sel	b_sel	op_sel	wsel	wen
S0	10	10	10	1	1
S1	10	10	10	1	1
S2	10	01	10	0	1
S3	11	00	01	1	1
S4	01	10	00	1	1
S5	01	10	00	0	1
S6	10	10	10	1	0

Factorial -- Pseudocode

```
f = 1;
t = n;
while (t >= 1) {
    f = f * t;
    t = t - 1;
}
```

State Diagram



Control Signal

State	a_sel	b_sel	op_sel	wsel	wen
S0	10	10	10	1	1
S1	10	10	10	1	1
S2	10	01	10	0	1
S3	11	00	10	1	1
S4	01	10	01	0	1
S5	10	10	10	1	0