Computer Organization and Architecture

Module 4-a

Data Path and Control Path Example with Verilog

Prof. Indranil Sengupta

Dr. Sarani Bhattacharya

Department of Computer Science and Engineering

IIT Kharagpur

Data Path and Control Path

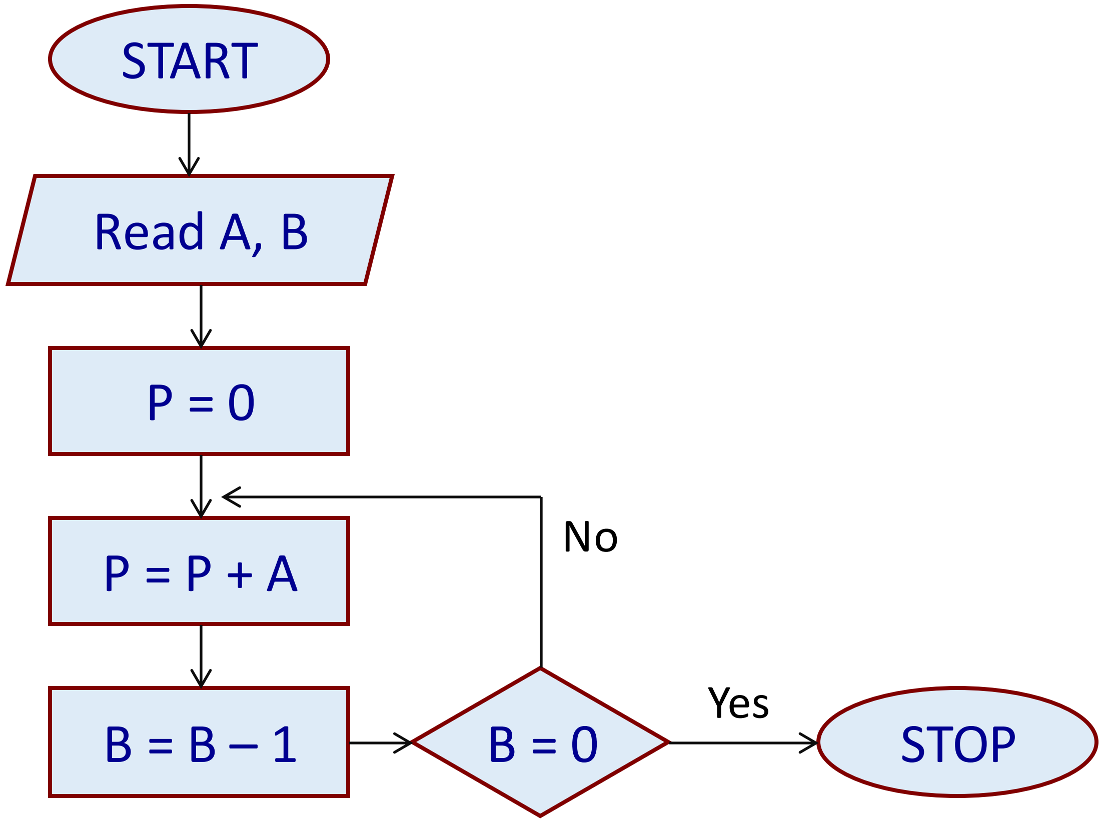
Introduction

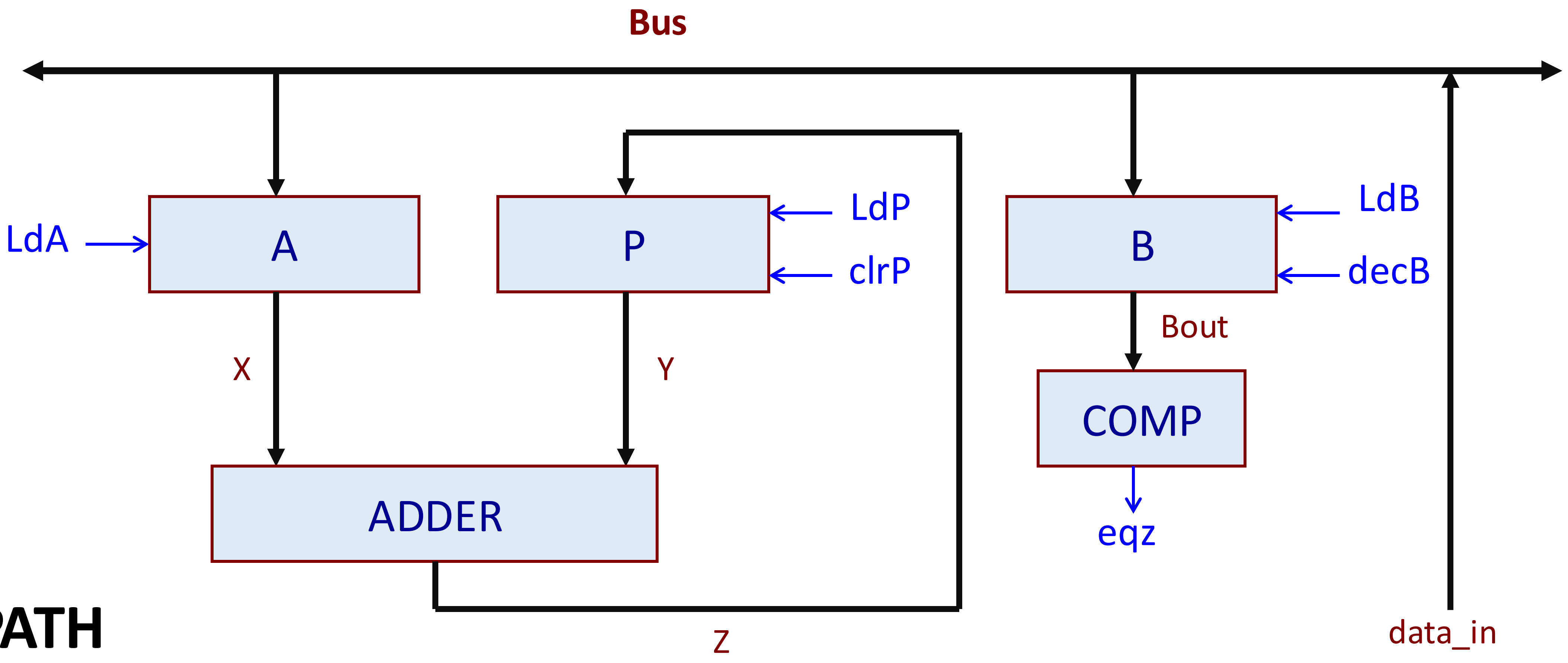
- In a complex digital system, the hardware is typically partitioned into two parts:
  a) *Data Path*, which consists of the functional units where all computations are carried out.
     - Typically consists of registers, multiplexers, bus, adders, multipliers, counters, and other functional blocks.

  b) *Control Path*, which implements a finite-state machine and provides control signals to the data path in proper sequence.
     - In response to the control signals, various operations are carried out by the data path.
     - Also takes inputs from the data path regarding various status information.

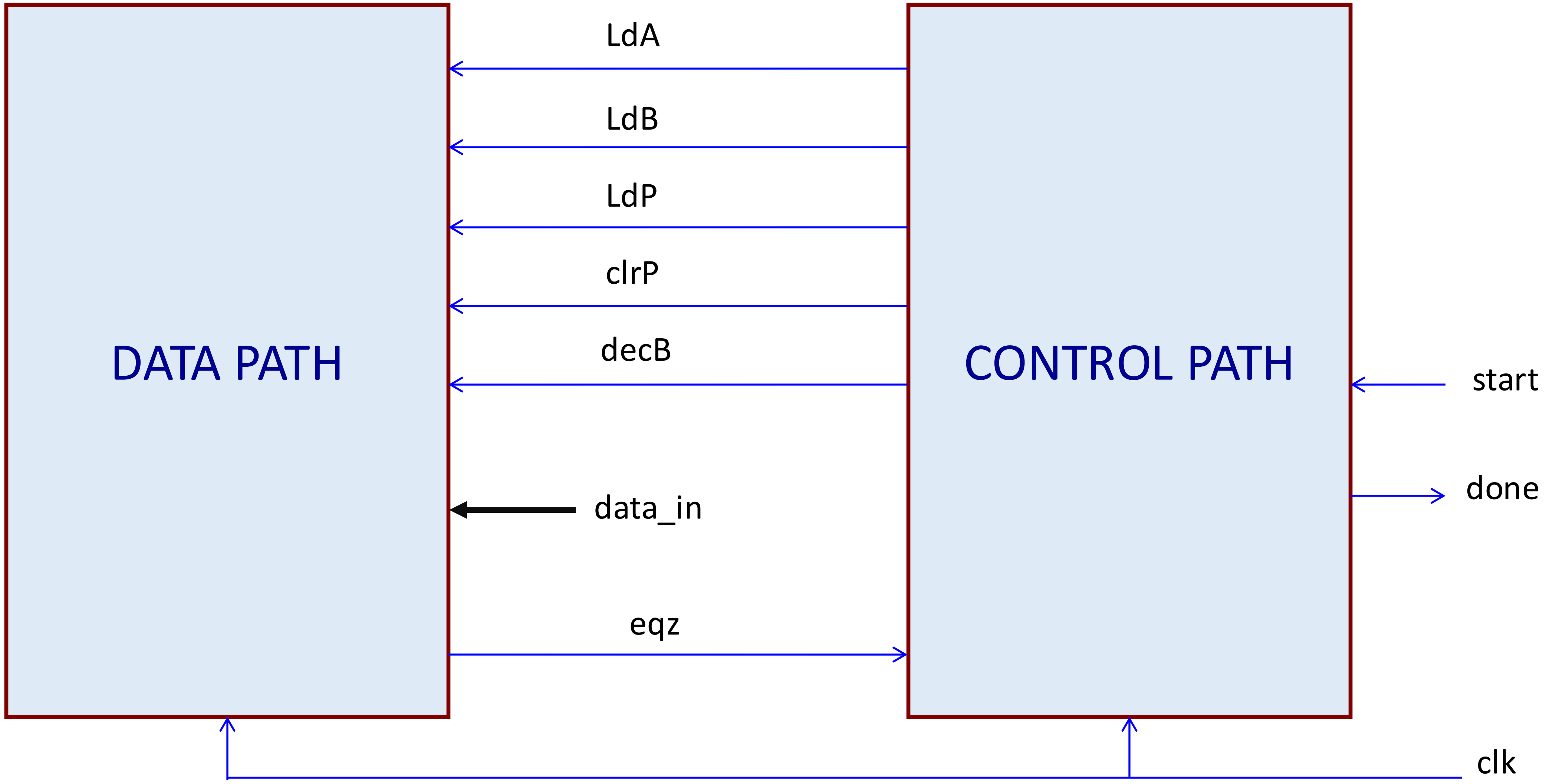
Example 1: Multiplication by Repeated Addition

- We consider a simple algorithm using repeated addition.
  - Assume B is non-zero.
- We identify the functional blocks required in the data path, and the corresponding control signals.
- Then we design the FSM to implement the multiplication algorithm using the data path.

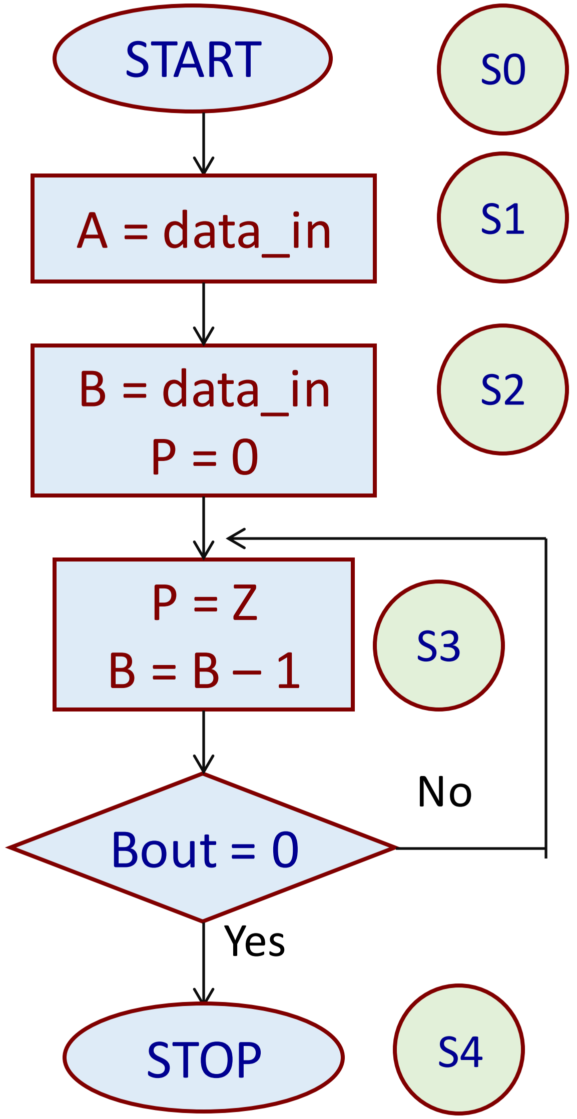




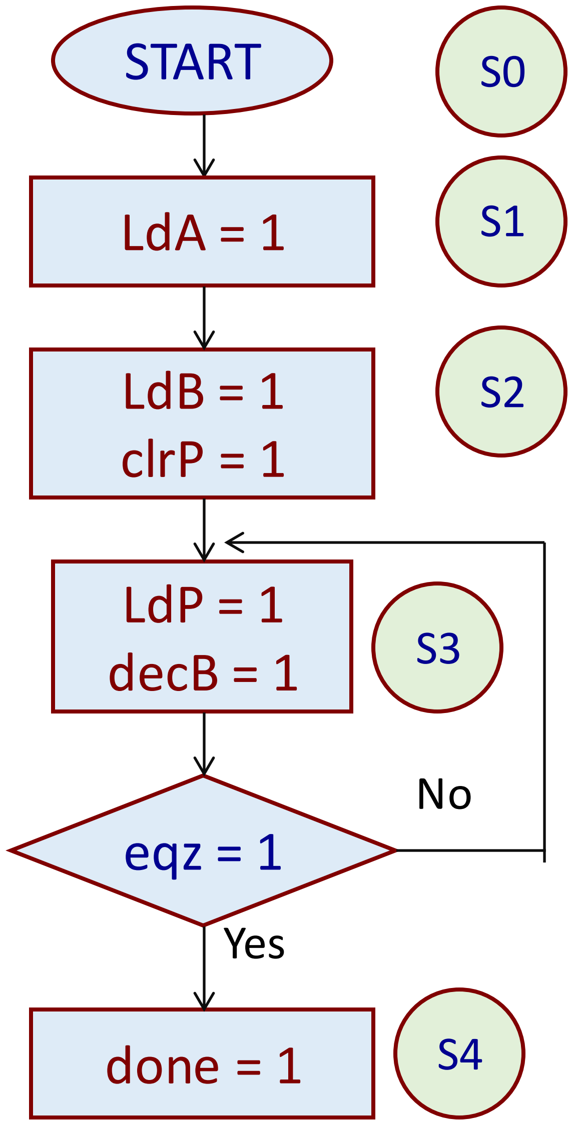
DATA PATH


LdA
LdB
LdP
clrP
decB
DATA PATH
CONTROL PATH
start
done
data_in
eqz
clk

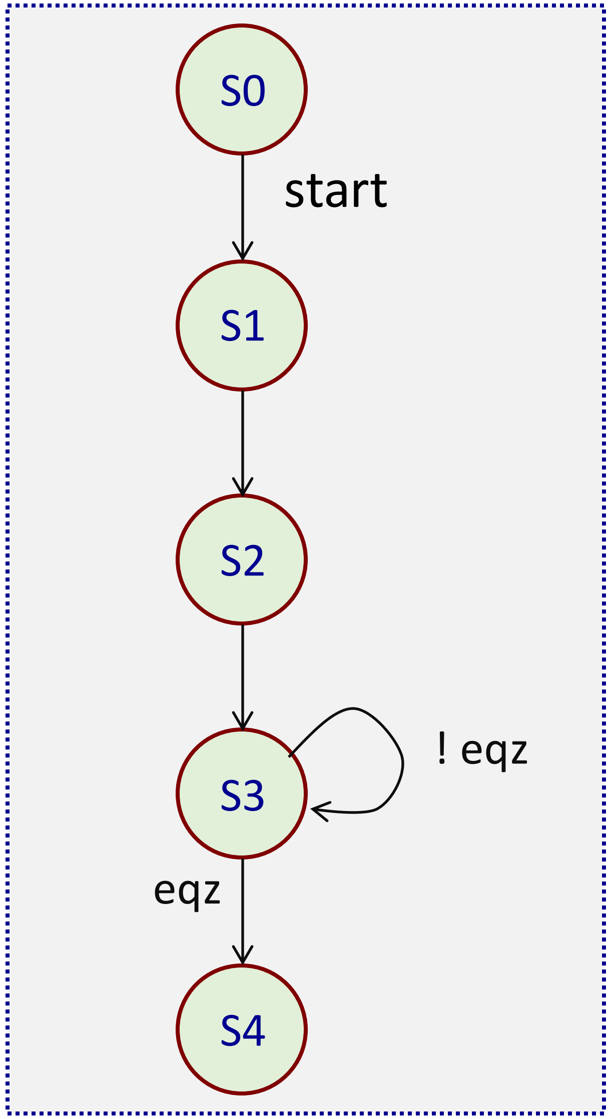
CONTROL PATH



Micro-operations



Control Signals



State Transitions

```
module MUL_datapath (eqz, LdA, LdB, LdP, clrP, decB, data_in, clk);
  input LdA, LdB, LdP, clrP, decB, clk;
  input [15:0] data_in;
  output eqz;
  wire [15:0] X, Y, Z, Bout, Bus;

  PIPO1 A (X, Bus, LdA, clk);
  PIPO2 P (Y, Z, LdP, clrP, clk);
  CNTR  B (Bout, Bus, LdB, decB, clk);
  ADD   AD (Z, X, Y);
  EQZ COMP (eqz, Bout);
endmodule
```

THE DATA PATH

```
module PIPO1 (dout, din, ld, clk);
  input [15:0] din;
  input ld, clk;
  output reg [15:0] dout;
  always @(posedge clk)
    if (ld) dout <= din;
endmodule

module ADD (out, in1, in2);
  input [15:0] in1, in2;
  output reg [15:0] out;
  always @(*)
    out = in1 + in2;
endmodule
```

```
module PIPO2 (dout, din, ld,
                     clr, clk);
  input [15:0] din;
  input ld, clr, clk;
  output reg [15:0] dout;
  always @(posedge clk)
    if (clr) dout <= 16'b0;
    else if (ld) dout <= din;
endmodule

module EQZ (eqz, data);
  input [15:0] data;
  output eqz;
  assign  eqz = (data == 0);
endmodule
```

```
module CNTR (dout, din, ld, dec, clk);
  input [15:0] din;
  input ld, dec, clk;
  output reg [15:0] dout;

  always @(posedge clk)
    if (ld) dout <= din;
    else if (dec) dout <= dout - 1;
endmodule
```

```
module controller (LdA, LdB, LdP, clrP, decB, done, clk, eqz, start);
  input clk, eqz, start;
  output reg LdA, LdB, LdP, clrP, decB, done;
  reg [2:0] state;
  parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100;
  always @(posedge clk)
    begin
      case (state)
        S0:    if (start) state <= S1;
        S1:    state <= S2;
        S2:    state <= S3;
        S3:    #2 if (eqz) state <= S4;
        S4:    state <= S4;
        default: state <= S0;
      endcase
    end
```

THE CONTROL PATH

```
always @(state)
  begin
    case (state)
      S0:   begin #1 LdA = 0;  LdB = 0; LdP = 0; clrP = 0; decB = 0; end
      S1:   begin #1 LdA = 1; end
      S2:   begin #1 LdA = 0; LdB = 1; clrP = 1; end
      S3:   begin #1 LdB = 0; LdP = 1; clrP = 0; decB = 1; end
      S4:   begin #1 done = 1; LdB = 0; LdP = 0; decB = 0; end
    default: begin #1 LdA = 0; LdB = 0; LdP = 0; clrP = 0; decB = 0; end
    endcase
  end
endmodule
```

THE TEST BENCH

```
module MUL_test;
  reg [15:0] data_in;
  reg clk, start;
  wire done;

  MUL_datapath DP (eqz, LdA, LdB, LdP, clrP, decB, data_in, clk);
  controller  CON (LdA, LdB, LdP, clrP, decB, done, clk, eqz, start);

  initial
    begin
      clk = 1'b0;
      #3 start = 1'b1;
      #500 $finish;
    end

  always #5 clk = ~clk;
```

```
  initial
    begin
      #17 data_in = 17;
      #10 data_in = 5;
    end

  initial
    begin
      $monitor ($time, " %d %b", DP.Y, done);
      $dumpfile ("mul.vcd"); $dumpvars (0, MUL_test);
    end

endmodule
```

```
 0      x x
 6      x 0
35      0 0
45     17 0
55     34 0
65     51 0
75     68 0
85     85 0
88     85 1
```