PRE-LAB 6

6.3 DATAPATH DESIGN (WEEK 14)

- module decoder(wrt_addr, A, B, C, D);
```
input [1:0] wrt_addr;
output reg A, B, C, D;

always @(wrt_addr)
 begin
  case(wrt_addr)
   00: begin A=1; B=0; C=0; D=0; end
   01: begin A=0; B=1; C=0; D=0; end
   10: begin A=0; B=0; C=1; D=0; end
   default: begin A=0; B=0; C=0; D=1; end
  endcase
 end
endmodule
```

- module MUX_2(S, Y, Q, F);
```
 parameter size = 4;
 input [size-1:0] Y, Q;
 input S;
output reg [size-1:0] F;

always @(Y or Q or S)
begin
    case(S)
        0: F <= Q;
        1: F <= Y;
    endcase
  end
endmodule
```

- module MUX_4(S, Reg0, Reg1, Reg2, Reg3, M);
```
 parameter size = 4;
 input [size-1:0] Reg0, Reg1, Reg2, Reg3;
 input [1:0] S;
 output reg [size-1:0] M;

 always @(Reg0 or Reg1 or Reg2 or Reg3 or S)
  begin
   case(S)
    00: M <= Reg0;
    01: M <= Reg1;
    10: M <= Reg2;
    default: M <= Reg3;
   endcase
  end
endmodule
```

- ```
  module AND_Gates (wrt_en, A, S);
    input wrt_en, A;
    output S;
    and G(S, wrt_en, A);
  endmodule
  ```

- ```
  module DFF(Clk, F, Reg);
    parameter size = 4;
    input [size-1:0] F;
    input Clk;
    output reg [size-1:0] Reg;

    always @(posedge Clk)
      begin
        Reg <= F;
      end
  endmodule
  ```

- ```
  module Registers (Clk, F0, F1, F2, F3, Reg0, Reg1, Reg2, Reg3);
      parameter size = 4 ;
      input wire [size-1:0] F0, F1, F2, F3;
      input Clk;
      output wire [size-1:0] Reg0, Reg1, Reg2, Reg3;
      reg [size-1:0] D0, D1, D2, D3;

      // Creating a register with 4 D-FFs:
      DFF FF0 (Clk, D0, Reg0);
      DFF FF1 (Clk, D1, Reg1);
      DFF FF2 (Clk, D2, Reg2);
      DFF FF3 (Clk, D3, Reg3);

      always @(posedge Clk)
        begin
          D0 <= F0;
          D1 <= F1;
          D2 <= F2;
          D3 <= F3;
        end
    endmodule
  ```

- ```
  module ALU (M1, M0, alu_opcode, F4, zero_flag);
      parameter size = 4;
      input [size-1:0] M0, M1;
      input [size-2:0] alu_opcode;
  ```

```
output reg [size-1:0] F4;
output reg zero_flag;

always @(M0 or M1 or alu_opcode)
 begin
  case(alu_opcode)
   001: F4 <= 1;   //set
   010: F4 <= M1 + 1;    //increment
   011: F4 <= M1 - 1 ;    //decrement
   101: F4 <= M1;    //store -> R[xx] to data_out
   110: F4 <= M1 + M0;   //add
   111: F4 <= M0;   //copy -> R[yy] to R[xx]
   default: F4 <= 0;
  endcase
 end

always @(F4)
 begin
  if (!F4)
   zero_flag = 1;
  else
   zero_flag = 0;
 end
endmodule
```

- module Fibo_Datapath (Clk, wrt_addr, wrt_en, load_data, rd_addr1, rd_addr2, alu_opcode, count, data, zero_flag);

```
  parameter size = 4;
  input [size-1:0] count;
  input [size-2:0] alu_opcode;
  input [1:0] wrt_addr, rd_addr1, rd_addr2;
  input Clk, wrt_en, load_data;
  output [size-1:0] data;
  output zero_flag;
  wire [size-1:0] Y, Reg0, Reg1, Reg2, Reg3, F0, F1, F2, F3, F4, M0, M1;
  wire A, B, C, D, S0, S1, S2, S3;

  //Connecting the 2-to-4 decoder:
  decoder Decoder (wrt_addr, A, B, C, D);

  //Connecting 4 AND gates:
  AND_Gates G0 (wrt_en, A, S0);
  AND_Gates G1 (wrt_en, B, S1);
  AND_Gates G2 (wrt_en, C, S2);
  AND_Gates G3 (wrt_en, D, S3);

  //Connecting 4 2-to-1 MUXs and 4 D-FFs (a register):
  MUX_2 MUX0 (S0, Y, Reg0, F0);
  MUX_2 MUX1 (S1, Y, Reg1, F1);
```

```
  MUX_2 MUX2 (S2, Y, Reg2, F2);
  MUX_2 MUX3 (S3, Y, Reg3, F3);
  Registers R (Clk, F0, F1, F2, F3, Reg0, Reg1, Reg2, Reg3);

  //Connecting single MUX:
  MUX_2 M (load_data, count, data, Y);

  //Connecting 2 4-to-1 MUX and ALU:
  MUX_4 mux1 (rd_addr1, Reg0, Reg1, Reg2, Reg3, M1);
  MUX_4 mux2 (rd_addr2, Reg0, Reg1, Reg2, Reg3, M0);
  ALU ALU (M1, M0, alu_opcode, F4, zero_flag);

  //Connecting single D-FF at the very bottom:
  DFF DFF (Clk, F4, data);

endmodule
```

- ```
  module Fibo_Datapath_tb ();
    parameter size = 4;
    reg [size-1:0] count;
    reg [size-2:0] alu_opcode;
    reg [1:0] wrt_addr, rd_addr1, rd_addr2;
    reg Clk, wrt_en, load_data;
    wire [size-1:0] data;
    wire zero_flag;

    Fibo_Datapath Fibonacci (Clk, wrt_addr, wrt_en, load_data, rd_addr1, rd_addr2, alu_opcode,
  count, data, zero_flag);

    always #5 Clk = ~Clk;

    initial
      begin

        Clk=0; wrt_addr=1; wrt_en=1; load_data=1; rd_addr1=2'b01; rd_addr2=2'b11;
  alu_opcode=3'b001; count=4'b1010;
                                #10 wrt_addr=1; wrt_en=1; load_data=1; rd_addr1=2'b10; rd_addr2=2'b01;
  alu_opcode=3'b010; count=4'b0100;
                        #10 wrt_addr=1; wrt_en=1; load_data=1; rd_addr1=2'b01; rd_addr2=2'b10;
  alu_opcode=3'b001; count=4'b1001;
                        #10 wrt_addr=1; wrt_en=1; load_data=1; rd_addr1=2'b01; rd_addr2=2'b01;
  alu_opcode=3'b110; count=4'b0010;
                        #10 wrt_addr=1; wrt_en=1; load_data=1; rd_addr1=2'b10; rd_addr2=2'b01;
  alu_opcode=3'b010; count=4'b1011;
                        #10 wrt_addr=1; wrt_en=1; load_data=1; rd_addr1=2'b10; rd_addr2=2'b10;
  alu_opcode=3'b111; count=4'b1100;
                        #10 wrt_addr=1; wrt_en=1; load_data=1; rd_addr1=2'b11; rd_addr2=2'b10;
  alu_opcode=3'b110; count=4'b0101;
  ```
  ```

```
                #10 wrt_addr=1; wrt_en=0; load_data=1; rd_addr1=2'b10; rd_addr2=2'b11;
alu_opcode=3'b110; count=4'b1110;
                #10 wrt_addr=1; wrt_en=1; load_data=1; rd_addr1=2'b01; rd_addr2=2'b01;
alu_opcode=3'b101; count=4'b1111;
                #10 wrt_addr=0; wrt_en=1; load_data=0; rd_addr1=2'b10; rd_addr2=2'b10;
alu_opcode=3'b110; count=4'b1000;
                #10 wrt_addr=1; wrt_en=1; load_data=1; rd_addr1=2'b11; rd_addr2=2'b10;
alu_opcode=3'b101; count=4'b1010;
                #10 wrt_addr=1; wrt_en=0; load_data=1; rd_addr1=2'b11; rd_addr2=2'b11;
alu_opcode=3'b111; count=4'b1101;

                end
endmodule
```