DSLab.16 Design at Register Transfer Level

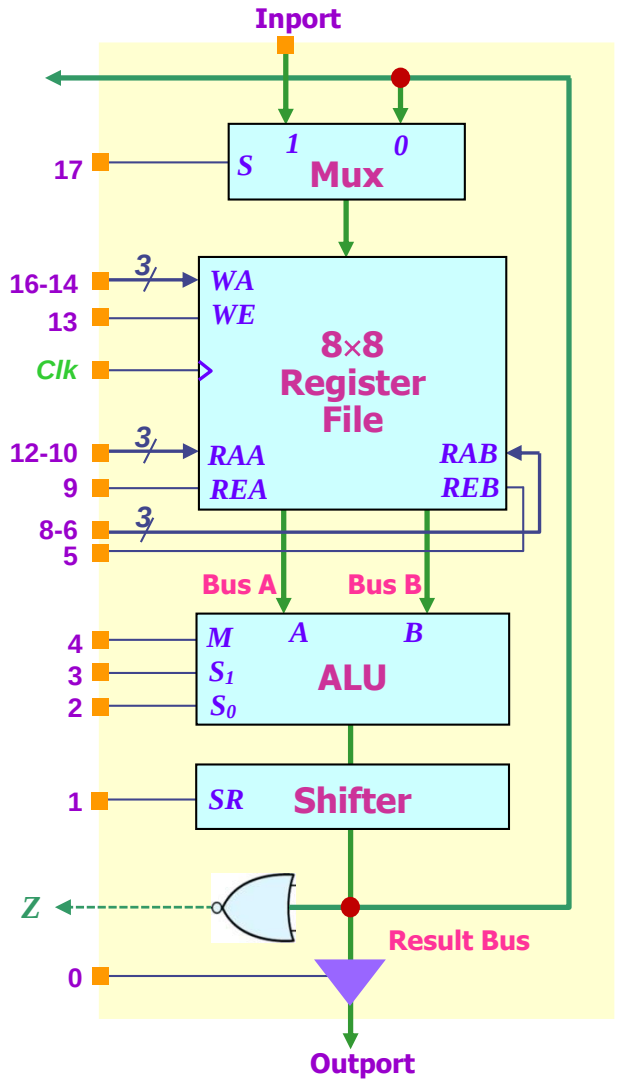
Lab. 16 Design at Register Transfer Level

- Design and Verify the following circuits using Verilog HDL
  - One's Counter

- This lab does not need to upload the lab report

Example 1: 8 x 8 Register file with 2 read and 1 write ports

```
module regfile(WA, WE, RAA, REA, RAB, REB, DATA_W, DATA_A, DATA_B, clk, rst);

input [2:0] WA, RAA, RAB;
input WE, REA, REB, clk, rst;
input [7:0] DATA_W;
output [7:0] DATA_A, DATA_B;

reg [7:0] DATA_A, DATA_B;
reg [7:0] registers[7:0];

 assign DATA_A = registers[RAA];
 assign DATA_B = registers[RAB];

always @ (posedge clk or negedge rst)
begin
  if (~rst)
     registers[0] = 0;
  else
  begin
     if (WE) registers[WA] <= DATA_W;
  end
end

endmodule
```



Example 2: ALU

```
module alu (input [7:0] A, input [7:0] B, input [2:0] sel, output reg [7:0] F);

 always@(A or B or sel)
 begin
    case(sel)
      3'b000: F = ~A;
      3'b001: F = A & B;
      3'b010: F = A ^ B;
      3'b011: F = A | B;
      3'b100: F = A - 1;
      3'b101: F = A + B;
      3'b110: F = A - B;
      3'b111: F = A + 1;
    endcase
  end
endmodule
```

sel[2:0]			*ALU Operations*
0	0	0	complement A
0	0	1	AND
0	1	0	EX−OR
0	1	1	OR
1	0	0	decrement A
1	0	1	add
1	1	0	subtract
1	1	1	increment A

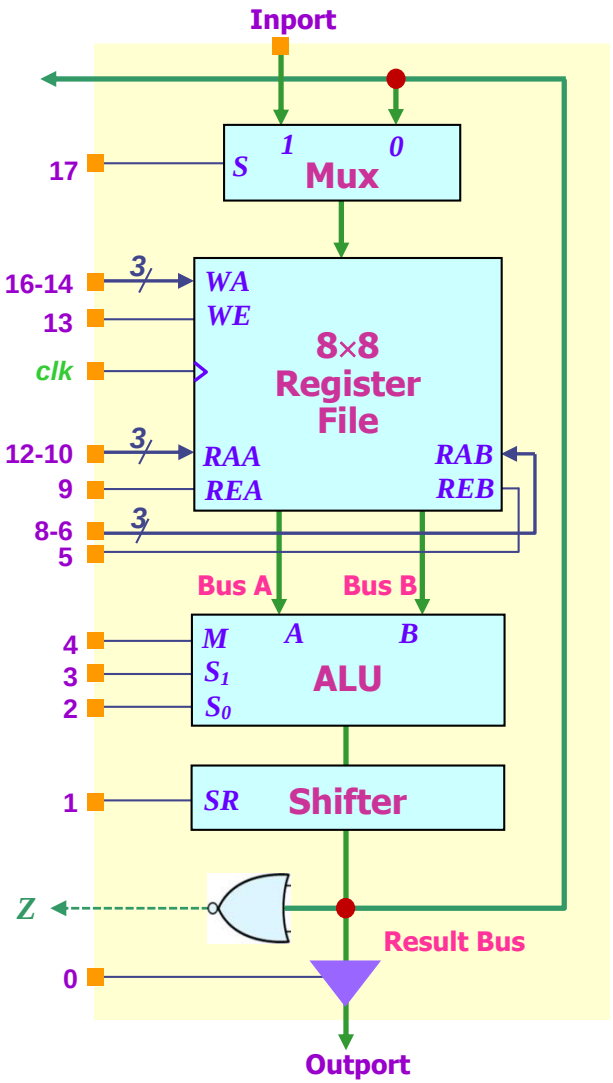
Table of ALU operations

Example 3: Shifter & Multiplexer

```
module shifter (data_in, SR , data_out);
  input [7:0] data_in;
  input SR;
  output [7:0] data_out;
  reg [7:0] data_out;

  always @ (data_in or SR)
     if (SR == 1'b1) data_out<= {1'b0 , data_in[7:1]};
     else  data_out <= data_in;

endmodule
```

```
module mux2(A, B, S, Y);
  input    [7:0]  A, B;
  input    S;
  output  [7:0] Y;

  reg       [7:0] Y;

  always @(S or A or B)
  begin
     case (S)
        1'b0: Y = A;
        1'b1: Y = B;
     endcase
  end

endmodule
```

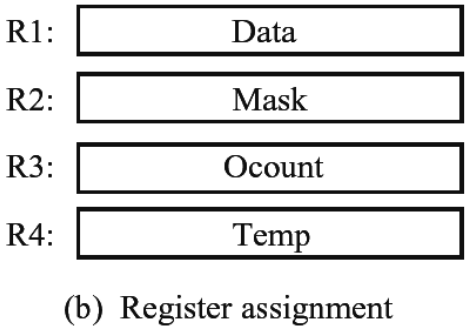
Exercise : One's Counter (1/4)



17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IE	*Write address*			*WE*	*Read address A*			*REA*	*Read address B*			*REB*	*ALU operation*			*SR*	*OE*

Control word

1. Data := Inport
2. Ocount := 0
3. Mask := 1
   while Data :≠ 0 **repeat**
4. Temp := Data AND Mask
5. Ocount : = Ocount + Temp
6. Data := Data >> 1
   end while
7. Outport := Ocount



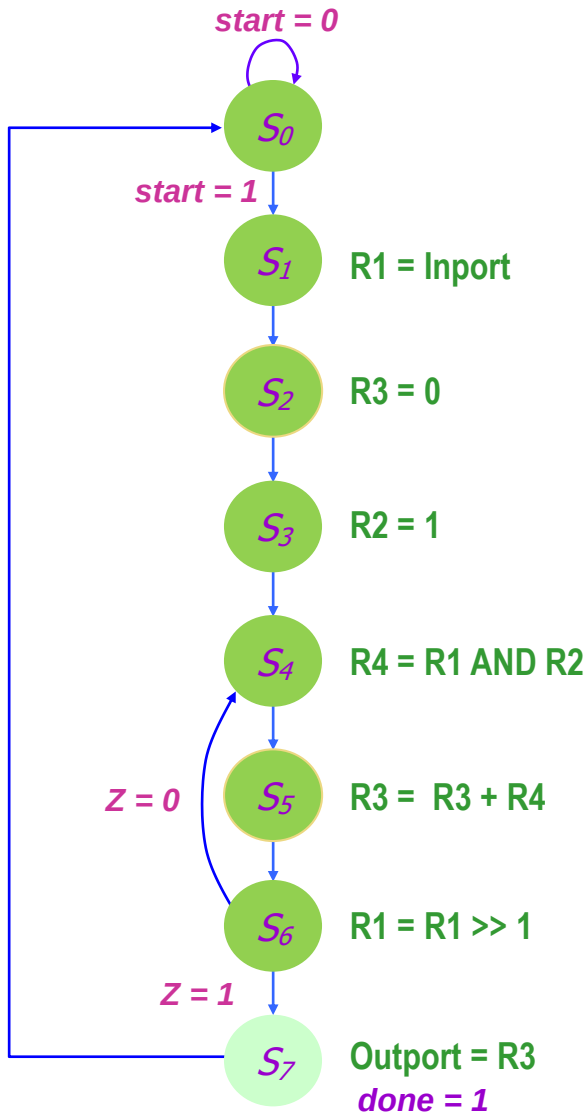
(b) Register assignment

Exercise : One's Counter (2/4)

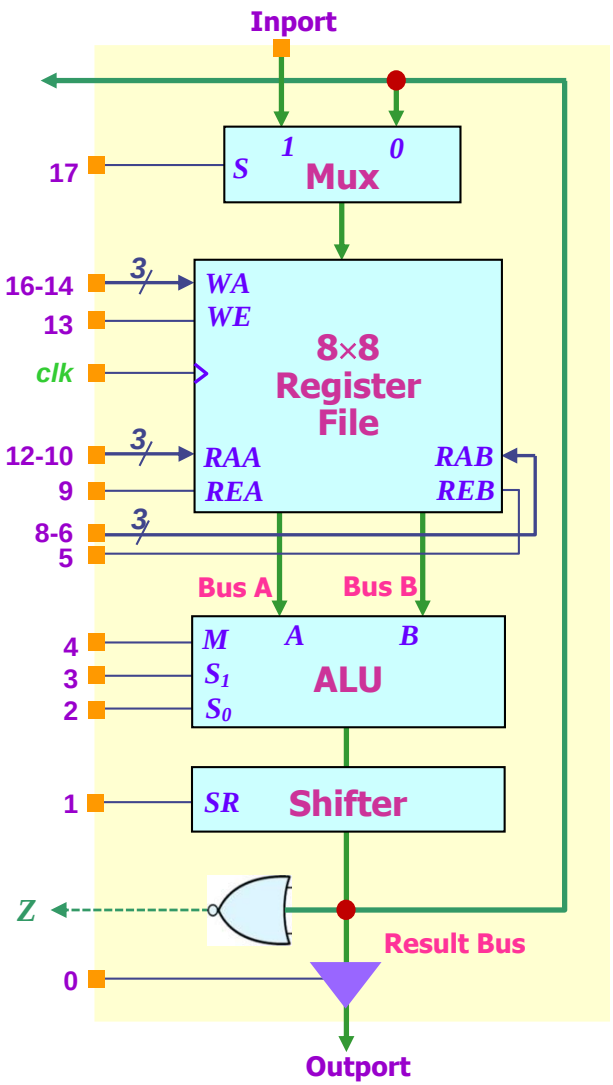
```
1.  Data := Inport
2.  Ocount := 0
3.  Mask := 1
    while Data :≠ 0  repeat
4.        Temp := Data AND Mask
5.        Ocount : =  Ocount + Temp
6.        Data := Data >> 1
    end while
7.  Outport := Ocount
```

```
1.  R1 := Inport
2.  R3 := 0
3.  R2 := 1
    while ~Z repeat
4.        R4 := R1 AND R2
5.        R3 : =  R3 + R4
6.        R1 := R1 >> 1
    end while
7.  Outport := R3
```

⇒



Exercise : One's Counter (3/4)




```
1. R1 := Inport
2. R3 := 0
3. R2 := 1
   while ~Z repeat
4.       R4 := R1 AND R2
5.       R3 : =  R3 + R4
6.       R1 := R1 >> 1
   end while
7. Outport:= R3
```

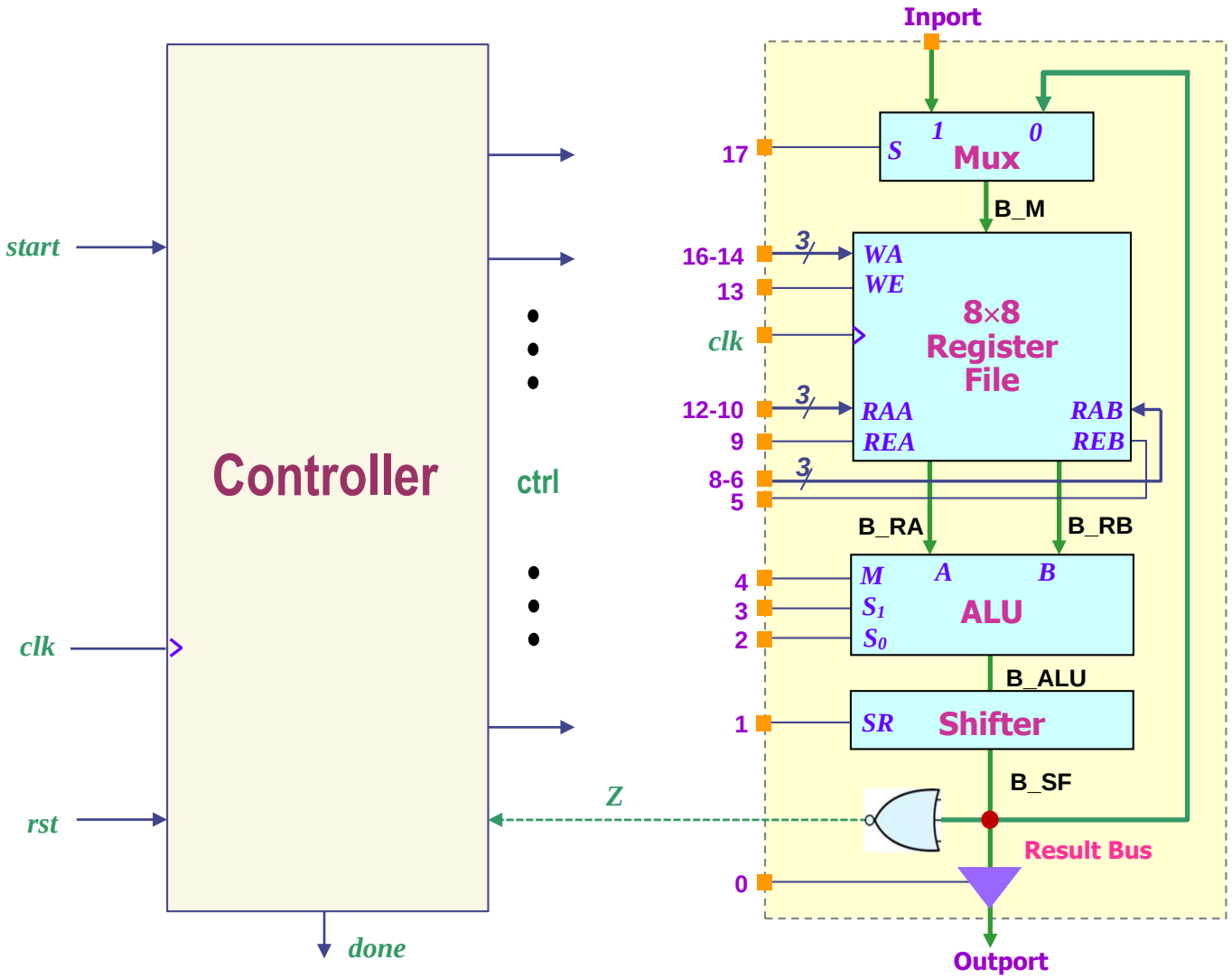
R1: Data
R2: Mask
R3: Ocount
R4: Temp

(b) Register assignment

	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	IE	*Write address*			*WE*	*Read address A*			*REA*	*Read address B*			*REB*	*ALU operation*			*SR*	*OE*
1	1	R1			1	x			0	x			0	x			x	0
2	0	R3			1	R0			1	R0			1	add			0	0
3	0	R2			1	R0			1	x			0	inc			0	0
4	0	R4			1	R1			1	R2			1	AND			0	0
5	0	R3			1	R3			1	R4			1	add			0	0
6	0	R1			1	R1			1	R0			1	add			1	0
7	0	x			0	R3			1	R0			1	add			0	1

Control words for one's counter

Exercise : One's Counter (4/4)



Verilog of One's Counter (1/4)

```
module Ones_Counter(start, inport, clk, rst, outport, done);
  input    start, clk, rst;
  input    [7:0] inport;
  output   done;
  output   [7:0] outport;

  wire     Z;
  wire     [17:0] ctrl;

  Controller Controller( .start(start), .Z(Z), .clk(clk), .rst(rst), .ctrl(ctrl), .done(done));
  Datapath Datapath( .inport(inport), .ctrl(ctrl), .clk(clk), .rst(rst), .outport(outport), .Z(Z));

endmodule
```

Verilog of One's Counter (2/4)

```
module Controller (start, Z, clk, rst, ctrl, done);

input       start, Z, clk, rst;
output      done;
output      [17:0] ctrl;
parameter   S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011;
parameter      S4=3'b100, S5=3'b101, S6=3'b110, S7=3'b111;
reg         done;
reg         [17:0] ctrl;
reg         [2:0] Current_State, Next_State;

always @(posedge clk or negedge rst)
begin
    if(~rst)  Current_State <= S0;
    else  Current_State <= Next_State;
end

always @(Current_State or start or Z)
begin
     case (Current_State)
          S0:
          begin
               ctrl = 18'b0_0000_0000_0000_000_00;
               done = 1'b0;
               if(~start)  Next_State = S0;
               else   Next_State = S1;
          end
          S1:
          begin
               ctrl = 18'b1_0011_0001_0001_000_00;
               done = 1'b0;
               Next_State = S2;
          end
                                  ●
                                  ●
                                  ●
          S6:
          begin
               ctrl = 18'b0_0011_0011_0001_101_10;
               done = 1'b0;
               if(Z)  Next_State = S7;
               else   Next_State = S4;
          end
          S7:
          begin
               ctrl = 18'b0_0000_0111_0001_101_01;
               done = 1'b1;
               Next_State = S0;
          end
     endcase
end
endmodule
```

Verilog of One's Counter (3/4)

```
module Datapath(inport, ctrl, clk, rst, outport, Z);
input       [7:0] inport;
input       [17:0] ctrl;
input       clk, rst;
output      [7:0] outport;
output      Z;

wire        [7:0] B_M, B_RA, B_RB, B_ALU, B_SF;

mux2 U1(.A(B_SF), .B(inport), .S(ctrl[17]), .Y(B_M));

regfile U2(.WA(ctrl[16:14]), .WE(ctrl[13]),.RAA(ctrl[12:10]), .REA(ctrl[9]), .RAB(ctrl[8:6]),
   .REB(ctrl[5]), .DATA_W(B_M), .DATA_A(B_RA), .DATA_B(B_RB),.clk(clk), .rst(rst));

alu U3( .A(B_RA), .B(B_RB), .sel(ctrl[4:2]), .F(B_ALU));

shifter U4(.data_in(B_ALU), .SR(ctrl[1]), .data_out(B_SF) );

assign      outport  =  ctrl[0] ? B_SF : 8'bz;
assign      Z = ~|B_SF;

endmodule
```

Verilog of One's Counter (4/4)

```
module t_Ones_Counter;

  reg  [7:0]   inport;
  reg          start, clk, rst;
  wire [7:0]  outport;
  wire         done;

  Ones_Counter U0 (.start(start), .inport(inport), . clk(clk),
                                  . rst(rst), .outport(outport), .done(done));

   initial #400 $finish;
   initial begin clk = 0; forever #5 clk = ~clk; end

   initial fork
            rst = 0;
            start = 0;
     #3     rst = 1;
     #12   start = 1;  inport=215;        // 215 = 1101 0111
     #30   start = 0;
   join

endmodule
```