IC Design Lab7

- *Siddharth Joshi(22110109)*
- *Siya Patil(22110254)*

Code:

Execute Processor:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 31.03.2025 13:56:19
// Design Name:
// Module Name: Ex_Processor
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Ex_Processor(dlx_processor if1);

always_ff@(posedge if1.clk or negedge if1.rst)
begin
   //reset
   if(!if1.rst)
   begin
      if1.aluin1 = 0; if1.aluin2 = 0; if1.operation = 0; if1.opselect = 0; if1.shift_number = 0;
      if1.enable_arith = 0; if1.enable_shift = 0;
   end

   if(if1.enable_ex)
   begin
```

```
//operation and opselect
if1.operation = if1.control_in[6:4];
if1.opselect = if1.control_in[2:0];

//aluin1
if1.aluin1 = if1.src1;

//aluin2
case(if1.opselect)
if1.ARITH_LOGIC: if (if1.control_in[3]) if1.aluin2 = if1.imm;
                 else if1.aluin2 = if1.src2;
if1.MEM_READ: if (if1.control_in[3]) if1.aluin2 = if1.mem_data_read_in;
              else if1.aluin2 = if1.aluin2;
default: if1.aluin2 = 0;
endcase

//shift_number
if(if1.opselect == if1.SHIFT_REG)
begin
   if(if1.imm[2]) if1.shift_number = if1.src2[4:0];
   else if1.shift_number = if1.imm[10:6];
end
else if1.shift_number = if1.shift_number;

//enable_arith
case(if1.opselect)
if1.ARITH_LOGIC: if1.enable_arith = 1;
if1.MEM_READ: if(if1.control_in[3]) if1.enable_arith = 1;
              else if1.enable_arith = 0;
default: if1.enable_arith = 0;
endcase

//enable_shift
if(if1.opselect == if1.SHIFT_REG) if1.enable_shift = 1;
else if1.enable_shift = 0;

end

else
begin
   if1.aluin1 = if1.aluin1;
   if1.aluin2 = if1.aluin2;
   if1.operation = if1.operation;
   if1.opselect = if1.opselect;
```

```
        if1.shift_number = if1.shift_number;
        if1.enable_arith = if1.enable_arith;
        if1.enable_shift = if1.enable_shift;
    end
end

always_comb
begin
    if1.mem_data_write_out = if1.src2;
    if1.mem_data_wr_en = (if1.opselect == if1.MEM_WRITE) && (if1.control_in[3] == 1);
end
endmodule
```

ALU Unit:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 31.03.2025 16:11:16
// Design Name:
// Module Name: ALU_Unit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ALU_Unit(dlx_processor if2);

logic [31:0]shift_aluout, arith_aluout;
logic h_carry;
logic [if2.IMMEDIATE_WIDTH -1:0]h_add;
always_ff@(posedge if2.clk or negedge if2.rst)
begin
    //reset
```

```
   if(!if2.rst)
   begin
      if2.aluout = 0; if2.carry = 0; shift_aluout = 0; arith_aluout = 0;
   end

   //shift ALU
   if(if2.enable_shift)
   begin
      case(if2.opselect[1:0])
      if2.SHLEFTLOG : shift_aluout = {1'b0, if2.aluin1 << if2.operation};
      if2.SHLEFTART : shift_aluout = {if2.aluin1[if2.REGISTER_WIDTH-1], if2.aluin1 <<
if2.operation};
      if2.SHRGHTLOG : shift_aluout = {1'b0, if2.aluin1 >> if2.operation };
      if2.SHRGHTART : shift_aluout = {1'b0, $signed(if2.aluin1) >>> if2.operation };
      default       : shift_aluout = shift_aluout ;
      endcase
   end
   else
   begin
      shift_aluout = shift_aluout;
   end

   //arith ALU
   if(if2.enable_arith)
   begin
      case({if2.opselect,if2.operation})
      {if2.ARITH_LOGIC , if2.ADD}  :  arith_aluout =  $signed(if2.aluin1) + $signed(if2.aluin2) ;
      {if2.ARITH_LOGIC , if2.HADD} : begin
                                        {h_carry,h_add}   = if2.aluin1[if2.IMMEDIATE_WIDTH-1:0] +
if2.aluin2[if2.IMMEDIATE_WIDTH-1:0];
                                        arith_aluout = {h_carry, $signed(h_add)};
                                     end
      {if2.ARITH_LOGIC , if2.SUB}  : arith_aluout =  $signed(if2.aluin1) - $signed(if2.aluin2);
      {if2.ARITH_LOGIC , if2.NOT}  : arith_aluout = {1'b0 ,~if2.aluin2};
      {if2.ARITH_LOGIC , if2.AND}  : arith_aluout = {1'b0 ,if2.aluin1 & if2.aluin2};
      {if2.ARITH_LOGIC , if2.OR}   : arith_aluout = {1'b0 ,if2.aluin1 | if2.aluin2};
      {if2.ARITH_LOGIC , if2.XOR}  : arith_aluout = {1'b0 ,if2.aluin1 ^ if2.aluin2};
      {if2.ARITH_LOGIC , if2.LHG}  : arith_aluout = {1'b0,if2.aluin2<<if2.IMMEDIATE_WIDTH};
      {if2.MEM_READ , if2.LOADBYTE}: arith_aluout = {1'b0,$signed(if2.aluin2[7:0])};
      {if2.MEM_READ , if2.LOADBYTEU}:arith_aluout = {1'b0,{(if2.REGISTER_WIDTH-8){1'b0}},
if2.aluin2[7:0]};
      {if2.MEM_READ , if2.LOADHALF}:arith_aluout = {1'b0,$signed(if2.aluin2[15:0])};
      {if2.MEM_READ , if2.LOADHALFU}:arith_aluout = {1'b0,{(if2.IMMEDIATE_WIDTH){1'b0}},
if2.aluin2[15:0]};
```

```
        {if2.MEM_READ , if2.LOADWORD}:arith_aluout = {1'b0,if2.aluin2};
        default                      :arith_aluout = (if2.opselect == if2.MEM_READ) ? {1'b0, if2.aluin2} :
arith_aluout;

        endcase
    end
    else
    begin
        arith_aluout = arith_aluout;
        if2.carry = if2.carry;
    end

    //final MUX
    {if2.carry, if2.aluout} = (if2.enable_arith)? arith_aluout : shift_aluout;
end
endmodule
```

Interface:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 31.03.2025 13:57:19
// Design Name:
// Module Name: Interface
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


interface dlx_processor();

// processor inputs
//input reg clk;
```

```
logic clk, rst;
logic enable_ex;
logic [31:0] src1, src2, imm, mem_data_read_in;
logic [6:0] control_in;

// processor output and ALU input
logic [2:0] opselect, operation;
logic enable_arith, enable_shift;
logic [31:0] aluin1, aluin2;
logic [4:0] shift_number;

//outputs
logic carry, mem_data_wr_en;
logic [31:0] aluout, mem_data_write_out;

modport execute_processor(input clk, rst, input enable_ex, src1, src2, imm, mem_data_read_in,
control_in,
output operation, opselect, enable_arith, enable_shift, aluin1, aluin2, shift_number,
mem_data_wr_en, mem_data_write_out);

modport ALU(input clk, rst, input enable_ex, operation, opselect, enable_arith, enable_shift,
aluin1, aluin2, shift_number,
output aluout, carry);

parameter CLK_PERIOD      = 10;
parameter REGISTER_WIDTH  = 32;
parameter INSTR_WIDTH     = 32;
parameter IMMEDIATE_WIDTH = 16;

// Instruction Types
parameter MEM_READ   = 3'b101;
parameter MEM_WRITE  = 3'b100;
parameter ARITH_LOGIC = 3'b001;
parameter SHIFT_REG  = 3'b000;

// ARITHMETIC
parameter ADD  = 3'b000;
parameter HADD = 3'b001;
parameter SUB  = 3'b010;
parameter NOT  = 3'b011;
parameter AND  = 3'b100;
parameter OR   = 3'b101;
parameter XOR  = 3'b110;
parameter LHG  = 3'b111;
```

```
// SHIFTING
parameter SHLEFTLOG  = 3'b000;
parameter SHLEFTART  = 3'b001;
parameter SHRGHTLOG  = 3'b010;
parameter SHRGHTART  = 3'b011;

// DATA TRANSFER
parameter LOADBYTE   = 3'b000;
parameter LOADBYTEU  = 3'b100;
parameter LOADHALF   = 3'b001;
parameter LOADHALFU  = 3'b101;
parameter LOADWORD   = 3'b011;
endinterface
```

Testbench:

```
//`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//// Company:
//// Engineer:
////
//// Create Date: 31.03.2025 17:11:04
//// Design Name:
//// Module Name: dlx_processor_tb
//// Project Name:
//// Target Devices:
//// Tool Versions:
//// Description:
////
//// Dependencies:
////
//// Revision:
//// Revision 0.01 - File Created
//// Additional Comments:
////
//////////////////////////////////////////////////////////////////////////////////


//module dlx_processor_tb();

//    // Instantiate the interface
//    dlx_processor if_inst();

//    // Instantiate the Execute Preprocessor and ALU modules.
```

```
//    Ex_Processor uut1 (.if1(if_inst));
//    ALU_Unit     uut2 (.if2(if_inst));

//    // Generate a clock with period = 10 ns.
//    initial begin
//        if_inst.clk = 0;
//        forever #5 if_inst.clk = ~if_inst.clk;
//    end

//    // Stimulus generation
//    initial begin
//        // Initialize reset and all signals
//        if_inst.rst = 0;
//        if_inst.enable_ex = 0;
//        if_inst.src1 = 32'd0;
//        if_inst.src2 = 32'd0;
//        if_inst.imm  = 32'd0;
//        if_inst.control_in = 7'd0;
//        if_inst.mem_data_read_in = 32'd0;

//        // Hold reset for a few clock cycles
//        #15;
//        if_inst.rst = 1;
//        #10;

//        // ----- Test Case 1: Arithmetic Addition -----
//        // Using ARITH_LOGIC operation with ADD.
//        // Definitions from the appendix:
//        //   ARITH_LOGIC = 3'b001, ADD = 3'b000.
//        // control_in = {operation, immediate flag, opselect}
//        // For addition with src2 (immediate flag = 0):
//        //   control_in = {3'b001, 1'b0, 3'b000} = 7'b0010_000.
//        if_inst.enable_ex = 1;
//        if_inst.src1 = 32'd10;
//        if_inst.src2 = 32'd20;
//        if_inst.imm  = 32'd100; // Not used when immediate flag is 0.
//        if_inst.control_in = 7'b0010000;
//        #10;

//        // ----- Test Case 2: Arithmetic Subtraction -----
//        // For subtraction, set opselect = 3'b010 (SUB) while keeping ARITH_LOGIC.
//        if_inst.src1 = 32'd50;
//        if_inst.src2 = 32'd15;
//        // control_in = {3'b001, 1'b0, 3'b010} = 7'b0010010.
```

```
//        if_inst.control_in = 7'b0010010;
//        #10;

//        // ----- Test Case 3: Shift Operation -----
//        // For shifting, opselect should be SHIFT_REG (3'b000).
//        // The testbench demonstrates selection of the shift amount via imm[10:6] when imm[2] is
0.
//        if_inst.src1 = 32'h0000000F;   // Data to shift.
//        if_inst.src2 = 32'h00000002;   // Alternative source for shift amount if imm[2]==1.
//        // Set imm such that bit2==0 and imm[10:6] = shift amount (e.g., shift by 2)
//        if_inst.imm = 32'b0;
//        if_inst.imm[10:6] = 5'b00010;
//        // control_in = {operation, immediate flag, opselect} where opselect is SHIFT_REG.
//        // Here, we use operation = 3'b001 (arbitrarily chosen) and immediate flag = 0.
//        if_inst.control_in = {3'b001, 1'b0, 3'b000};
//        #10;

//        // ----- Test Case 4: Memory Read (Load Byte) -----
//        // For a memory read operation, opselect should be MEM_READ (3'b101) with immediate
flag = 1.
//        // The specification indicates that when control_in[3]==1, aluin2 is taken from
mem_data_read_in.
//        if_inst.src2 = 32'd0;  // Not used here.
//        if_inst.mem_data_read_in = 32'h000000FF;  // Simulate reading 0xFF from memory.
//        // control_in = {operation, immediate flag, opselect} = {3'b101, 1'b1, 3'b101} = 7'b1011101.
//        if_inst.control_in = 7'b1011101;
//        #10;

//        // End simulation after a few more cycles.
////         #200;
//        $finish;
//    end

//endmodule


`timescale 1ns/1ps

module tb_dlx;
  // Instantiate the shared interface
  dlx_processor dut_if();

  // Instantiate DUT modules
  Ex_Processor ex_proc (dut_if);
```

```
  ALU_Unit      alu_unit (dut_if);

  // Clock Generation: 10 ns period (5 ns high, 5 ns low)
  initial begin
    dut_if.clk = 0;
    forever #5 dut_if.clk = ~dut_if.clk;
  end

  // Define arrays for opselect and operation values (0 to 7)
  int unsigned opselect_vals[8] = '{0,1,2,3,4,5,6,7};
  int unsigned operation_vals[8]  = '{0,1,2,3,4,5,6,7};
  bit control_bit[2] = '{0,1}; // Represents control_in[3]

  // Task: Loop first over opselect values, then for every opselect, sweep all operation values
  task run_by_opselect();
    int unsigned op_sel;
    int unsigned oper;
    bit ctrl_bit;
    foreach (opselect_vals[i]) begin
      op_sel = opselect_vals[i];
      foreach (control_bit[j]) begin
        ctrl_bit = control_bit[j];
        foreach (operation_vals[k]) begin
          oper = operation_vals[k];
          // Form the 7-bit control signal:
          // [6:4] = operation, [3] = ctrl_bit, [2:0] = op_sel
          dut_if.control_in = {oper[2:0], ctrl_bit, op_sel[2:0]};
          $display("Time=%0t | (ByOpSelect) opselect=%b, control_bit=%b, operation=%b =>
control_in=%b",
                   $time, op_sel[2:0], ctrl_bit, oper[2:0], dut_if.control_in);
          #10; // Wait one clock cycle for evaluation
        end
      end
    end
  endtask

  // Task: Loop first over operation values, then for every operation, sweep all opselect values
  task run_by_operation();
    int unsigned op_sel;
    int unsigned oper;
    bit ctrl_bit;
    foreach (operation_vals[i]) begin
      oper = operation_vals[i];
      foreach (control_bit[j]) begin
```

```
        ctrl_bit = control_bit[j];
        foreach (opselect_vals[k]) begin
          op_sel = opselect_vals[k];
          // Form the 7-bit control signal:
          // [6:4] = operation, [3] = ctrl_bit, [2:0] = op_sel
          dut_if.control_in = {oper[2:0], ctrl_bit, op_sel[2:0]};
          $display("Time=%0t | (ByOperation) operation=%b, control_bit=%b, opselect=%b =>
control_in=%b",
                   $time, oper[2:0], ctrl_bit, op_sel[2:0], dut_if.control_in);
          #10; // Wait one clock cycle for evaluation
        end
      end
    end
  endtask

  // Main test stimulus
  initial begin
    // Initialize/reset signals and drive large test values
    dut_if.rst = 0;
    dut_if.enable_ex = 0;
    dut_if.src1 = 32'hA5A5_A5A5;          // Large value for aluin1
    dut_if.src2 = 32'h5A5A_5A5A;          // Large value for aluin2
    dut_if.imm  = 32'hFFFF_0000;          // Large immediate value
    dut_if.mem_data_read_in = 32'h1234_5678; // Arbitrary test value

    // Apply reset for a few cycles
    #20;
    dut_if.rst = 1;
    #10;
    dut_if.enable_ex = 1;

    // Call tasks to exercise control_in
    $display("\nStarting test using opselect outer loop:");
    run_by_opselect();

    $display("\nStarting test using operation outer loop:");
    run_by_operation();

    $finish;
  end

endmodule
```

Simulation Waveform: