COMPUTER ORGANIZATION : LAB 5

REPORT

109550083 楊皓宇

109550121 温柏萱

PART 1 : IMPLEMENTATION DETAILS

Adder.v

```
module Adder(
    input  [32-1:0] src1_i,
    input  [32-1:0] src2_i,
    output reg [32-1:0] sum_o
);

always @(*) begin
    sum_o = src1_i + src2_i;
end

endmodule
```

Just sum it up.

MUX2to1.v, MUX3to1.v

```
module MUX_2to1(
    input  [31:0] data0_i,
    input  [31:0] data1_i,
```

```
    input         select_i,
    output reg [31:0] data_o
);

always @(*) begin
    case (select_i)
        2'b00: data_o = data0_i;
        2'b01: data_o = data1_i;
        default: data_o = 32'b0;
    endcase
end

endmodule

module MUX_3to1(
    input       [32-1:0] data0_i,
    input       [32-1:0] data1_i,
    input       [32-1:0] data2_i,
    input       [ 2-1:0] select_i,
    output  reg [32-1:0] data_o
);

always @(*) begin
    case (select_i)
        2'b00: data_o = data0_i;
        2'b01: data_o = data1_i;
        2'b10: data_o = data2_i;
        default: data_o = 32'b0;
    endcase
end

endmodule
```

MUX2to1

If the selection input is 0, output data_0, otherwise output data_1.

MUX3to1

3 cases, based on the selection line.

ALU_Ctrl.v

```
`timescale 1ns/1ps
module ALU_Ctrl(
    input       [4-1:0] instr,
    input       [2-1:0] ALUOp,
    output reg     [4-1:0] ALU_Ctrl_o
);
wire [2:0] func3;
assign func3 = instr[2:0];

    wire [6-1:0] allInputs;
  assign allInputs = {ALUOp, instr};

  always @(*) begin
    casez (allInputs)
      6'b100000: ALU_Ctrl_o = 4'b0010; //ADD
      6'b101000: ALU_Ctrl_o = 4'b0110; //SUB
      6'b100100: ALU_Ctrl_o = 4'b0101; //XOR
      6'b100110: ALU_Ctrl_o = 4'b0001; //OR
      6'b100111: ALU_Ctrl_o = 4'b0000; //AND
      6'b100010: ALU_Ctrl_o = 4'b0111; //SLT
      6'b10?000: ALU_Ctrl_o = 4'b0010; //ADDI
      6'b100001: ALU_Ctrl_o = 4'b0100; //SLLI
      6'b10?010: ALU_Ctrl_o = 4'b0111; //SLTI
      6'b00?010: ALU_Ctrl_o = 4'b0010; //LW, SW
      6'b01?000: ALU_Ctrl_o = 4'b0111; //BEQ
      default: ALU_Ctrl_o = 4'b0000;
    endcase
  end

endmodule
```

We can build the signals based on the table below.

instructions	ALU_Ctrl output
ADD	0010
SUB	0110
XOR	0101
OR	0001
AND	0111
SLT	0010
ADDI	0100
SLLI	0111
SLTI	0111
LW/SW	0010
BEQ	0111

Shift_Left_1.v

```
`timescale 1ns/1ps

module Shift_Left_1(
    input      [32-1:0] data_i,
    output reg  [32-1:0] data_o
    );

always @(*) begin
    data_o = data_i << 1;
end
endmodule
```

Use the operator in verilog to shift the data by 1 bit.

Decoder

```
`timescale 1ns/1ps

module Decoder(
    input [32-1:0]  instr_i,
    output reg         Branch,
    output reg         ALUSrc,
    output reg         RegWrite,
    output reg [2-1:0] ALUOp,
    output reg         MemRead,
    output reg         MemWrite,
    output reg         MemtoReg,
    output reg         Jump
);

//Internal Signals
wire    [7-1:0]     opcode = instr_i[6:0];
wire    [3-1:0]     funct3 = instr_i[14:12];
wire    [3-1:0]     Instr_field;
wire    [9:0]       Ctrl_o;

always @(*) begin
    case (opcode)
        7'b0110011: begin // R-type
            Branch = 1'b0;
            ALUSrc = 1'b0;
            RegWrite = 1'b1;
            ALUOp = 2'b10;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            MemtoReg = 1'b0;
            Jump = 1'b0;
        end
        7'b0010011: begin // I-type
            Branch = 1'b0;
            ALUSrc = 1'b1;
            RegWrite = 1'b1;
            ALUOp = 2'b10;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            MemtoReg = 1'b0;
```

```
        Jump = 1'b0;
    end
    7'b0000011: begin // LW
        Branch = 1'b0;
        ALUSrc = 1'b1;
        RegWrite = 1'b1;
        ALUOp = 2'b00;
        MemRead = 1'b1;
        MemWrite = 1'b0;
        MemtoReg = 1'b1;
        Jump = 1'b0;
    end
    7'b0100011: begin // SW
        Branch = 1'b0;
        ALUSrc = 1'b1;
        RegWrite = 1'b0;
        ALUOp = 2'b00;
        MemRead = 1'b0;
        MemWrite = 1'b1;
        MemtoReg = 1'b0;
        Jump = 1'b0;
    end
    7'b1100011: begin // BEQ
        Branch = 1'b1;
        ALUSrc = 1'b0;
        RegWrite = 1'b0;
        ALUOp = 2'b01;
        MemRead = 1'b0;
        MemWrite = 1'b0;
        MemtoReg = 1'b0;
        Jump = 1'b0;
    end
    7'b1101111: begin // JAL
        Branch = 1'b0;
        ALUSrc = 1'b0;
        RegWrite = 1'b0;
        ALUOp = 2'b11;
        MemRead = 1'b0;
        MemWrite = 1'b0;
        MemtoReg = 1'b0;
        Jump = 1'b1;
    end
```

```
    endcase

end
endmodule
```

instr	R-type	I-type	LW	SW	BEQ	JAL
Branch	0	0	0	0	1	0
ALUSrc	0	1	1	1	0	0
RegWrite	1	1	1	0	0	0
ALUOp	10	10	00	00	01	11
MemRead	0	0	1	0	0	0
MemWrite	0	0	0	1	0	0
MemtoReg	0	0	1	0	0	0
Jump	0	0	0	0	0	1

Based on the opcode, we can define each signal as the above table.

Forwarding.v

```
`timescale 1ns/1ps
module ForwardingUnit (
    input [5-1:0] IDEXE_RS1,
    input [5-1:0] IDEXE_RS2,
    input [5-1:0] EXEMEM_RD,
    input [5-1:0] MEMWB_RD,
    input [2-1:0] EXEMEM_RegWrite,
    input [2-1:0] MEMWB_RegWrite,
    output reg [2-1:0] ForwardA,
    output reg [2-1:0] ForwardB
);

always @(*) begin
    ForwardA = 2'b00;
    ForwardB = 2'b00;

    if (EXEMEM_RegWrite && EXEMEM_RD != 0 && EXEMEM_RD == IDEXE_RS1)
```

```
        ForwardA = 2'b10;

    if (EXEMEM_RegWrite && EXEMEM_RD != 0 && EXEMEM_RD == IDEXE_RS2)
        ForwardB = 2'b10;

    if (MEMWB_RegWrite && MEMWB_RD != 0 && !(EXEMEM_RegWrite) &&
EXEMEM_RD != 0 && EXEMEM_RD != IDEXE_RS1 && MEMWB_RD == IDEXE_RS1)
        ForwardA = 2'b01;

    if (MEMWB_RegWrite && MEMWB_RD != 0 && !(EXEMEM_RegWrite) &&
EXEMEM_RD != 0 && EXEMEM_RD != IDEXE_RS2 && MEMWB_RD == IDEXE_RS2)
        ForwardB = 2'b01;
end

endmodule
```

Set the default of `ForwardA` and `ForwardB` to be 0, since if the following conditions of forwarding are not met, then the CPU should not perform forwarding. There are 2 conditions for each forwarding line, Either

`EXEMEM_RegWrite` is not zero and (1). `EXEMEM_RD` is the same as `IDEXE_RS1` or (2). `MEMEWB_RegWrite` $\cap$ `EXEMEM_RD` $\neq$ `IDEXE_RS1` $\cap$ `MEMWB_RD` $=$ `IDEXE_RS1`.

Hazard_detection.v

```
`timescale 1ns/1ps
module Hazard_detection(
    input [4:0] IFID_regRs,
    input [4:0] IFID_regRt,
    input [4:0] IDEXE_regRd,
    input IDEXE_memRead,
    output reg PC_write,
    output reg IFID_write,
    output reg control_output_select
);
always @(*) begin
    if(IDEXE_memRead == 1'b1 && ((IDEXE_regRd == IFID_regRs) ||
(IDEXE_regRd == IFID_regRt))) begin
        PC_write = 1'b1;
        IFID_write = 1'b1;
```

```
        control_output_select = 1'b1;
    end
    else begin
        PC_write = 1'b0;
        IFID_write = 1'b0;
        control_output_select = 1'b0;
    end
end
endmodule
```

If `IDEXE_memRead` is 1 and `IDEXE_regRd` $=$ `IFID_regRs` or `IDEXE_regRd` $=$ `IFID_regRt`, then there may be a hazard, thus we let the outputs to be 1, otherwise the outputs are 0.

Imm_Gen.v

```
`timescale 1ns/1ps

module Imm_Gen(
    input      [31:0] instr_i,
    output reg [31:0] Imm_Gen_o
);
wire  [6:0]opcode;
assign opcode = instr_i[6:0];
always @(*) begin
    case (opcode)
        7'b0110011 :begin // R type
            Imm_Gen_o = 32'b0;
        end
        7'b0010011, 7'b0000011, 7'b1100111 :begin // I, load, jalr
            Imm_Gen_o = {{20{instr_i[31]}}, instr_i[31:20]};
        end
        7'b0100011: begin // Store
            Imm_Gen_o = {{21{instr_i[31]}}, instr_i[30:25],
instr_i[11:8], instr_i[7]};
        end
        7'b1100011: begin // Branch
            Imm_Gen_o = {{21{instr_i[31]}}, instr_i[7],
instr_i[30:25], instr_i[11:8], 1'b0};
        end
        7'b1101111: begin
```

```
            Imm_Gen_o = {{12{instr_i[31]}}, instr_i[19:12],
instr_i[20], instr_i[30:25], instr_i[24:21], 1'b0};
        end
        default:begin
            Imm_Gen_o = 32'b0;
        end
    endcase
end
endmodule
```

We can determine the value of the immediate by the opcode and the values are listed in the picture below.

The 4 pipeline registers

```
module IFID_register (
    input clk_i,
    input rst_i,
    input flush,
    input IFID_write,
    input [31:0] address_i,
    input [31:0] instr_i,
    input [31:0] pc_add4_i,

    output reg [31:0] address_o,
    output reg [31:0] instr_o,
    output reg [31:0] pc_add4_o
);
  always@(clk_i) begin
      if (rst_i || flush) begin
          pc_add4_o = 32'b0;
          instr_o = 32'b0;
          address_o = 32'b0;
      end
      else if(IFID_write) begin
          pc_add4_o = pc_add4_i;
          instr_o = instr_i;
          address_o = address_i;
      end
  end
```

```
endmodule

module IDEXE_register (
    input clk_i,
    input rst_i,
    input [31:0] instr_i,
    input [2:0] WB_i,
    input [1:0] Mem_i,
    input [2:0] Exe_i,
    input [31:0] data1_i,
    input [31:0] data2_i,
    input [31:0] immgen_i,
    input [3:0] alu_ctrl_instr,
    input [4:0] WBreg_i,
    input [31:0] pc_add4_i,

    output reg [31:0] instr_o,
    output reg [2:0] WB_o,
    output reg [1:0] Mem_o,
    output reg [2:0] Exe_o,
    output reg [31:0] data1_o,
    output reg [31:0] data2_o,
    output reg [31:0] immgen_o,
    output reg [3:0] alu_ctrl_input,
    output reg [4:0] WBreg_o,
    output reg [31:0] pc_add4_o
);
/* Write your code HERE */
  always @(clk_i) begin
      if(rst_i)begin
          instr_o = 32'b0;
          WB_o = 3'b0;
          Mem_o = 2'b0;
          Exe_o = 3'b0;
          data1_o = 32'b0;
          data1_o = 32'b0;
          immgen_o = 32'b0;
          alu_ctrl_input = 4'b0;
          WBreg_o = 5'b0;
          pc_add4_o = 32'b0;
      end
      else begin
          instr_o = instr_i;
```

```
            WB_o = WB_i;
            Mem_o = Mem_i;
            Exe_o = Exe_i;
            data1_o = data1_i;
            data1_o = data2_i;
            immgen_o = immgen_i;
            alu_ctrl_input = alu_ctrl_instr;
            WBreg_o = WBreg_i;
            pc_add4_o = pc_add4_i;
        end
    end
endmodule

module EXEMEM_register (
    input clk_i,
    input rst_i,
    input [31:0] instr_i,
    input [2:0] WB_i,
    input [2:0] Mem_i,
    input zero_i,
    input [31:0] alu_ans_i,
    input [31:0] rtdata_i,
    input [4:0] WBreg_i,
    input [31:0] pc_add4_i,

    output reg [31:0] instr_o,
    output reg [2:0] WB_o,
    output reg [2:0] Mem_o,
    output reg zero_o,
    output reg [31:0] alu_ans_o,
    output reg [31:0] rtdata_o,
    output reg [4:0] WBreg_o,
    output reg [31:0] pc_add4_o
);
    always @(clk_i) begin
        if(rst_i)begin
            instr_o = 32'b0;
            WB_o = 3'b0;
            Mem_o = 3'b0;
            zero_o = 1'b0;
            alu_ans_o = 32'b0;
            rtdata_o = 32'b0;
            WBreg_o = 5'b0;
```

```
      pc_add4_o = 32'b0;
    end
    else begin
      instr_o = instr_i;
      WB_o = WB_i;
      Mem_o = Mem_i;
      zero_o = zero_i;
      alu_ans_o = alu_ans_i;
      rtdata_o = rtdata_i;
      WBreg_o = WBreg_i;
      pc_add4_o = pc_add4_i;
    end
  end
endmodule

module MEMWB_register (
    input clk_i,
    input rst_i,
    input [2:0] WB_i,
    input [31:0] DM_i,
    input [31:0] alu_ans_i,
    input [4:0] WBreg_i,
    input [31:0] pc_add4_i,

    output reg [2:0] WB_o,
    output reg [31:0] DM_o,
    output reg [31:0] alu_ans_o,
    output reg [4:0] WBreg_o,
    output reg [31:0] pc_add4_o
);
  always @(clk_i) begin
      if(rst_i)begin
          WB_o = 3'b0;
          DM_o = 32'b0;
          alu_ans_o = 32'b0;
          WBreg_o = 5'b0;
          pc_add4_o = 32'b0;
      end
      else begin
          WB_o = WB_i;
          DM_o = DM_i;
          alu_ans_o = alu_ans_i;
          WBreg_o = WBreg_i;
```

```
            pc_add4_o = pc_add4_i;
        end
    end
endmodule
```

For the 4 registers, we put the input into the output if the reset signal is off and when a clock hits, otherwise assume them to be 0.

ALU.v

```
module alu(
    input                   rst_n,         // negative reset
   (input)
    input signed       [32-1:0]   src1,          // 32 bits source 1
         (input)
    input signed       [32-1:0]   src2,          // 32 bits source 2
         (input)
    input        [ 4-1:0]   ALU_control,   // 4 bits ALU control
input  (input)
    output reg   [32-1:0]   result,        // 32 bits result
   (output)
    output reg              zero           // 1 bit when the output
is 0, zero must be set (output)
);

  always @(*) begin
      zero = src1 == src2;

      case (ALU_control)
          4'b0000: result = src1 & src2; //AND
          4'b0001: result = src1 | src2; //OR
          4'b0010: result = src1 + src2; //ADD ADDI LW SW
          4'b0110: result = src1 - src2; //SUB
          4'b0111: result = src1 < src2; //SLT SLTI BEQ
          4'b0100: result = src1 << src2; //SLLI
          4'b0101: result = src1 ^ src2; //XOR
          default: result = 32'b0;
      endcase
  end
endmodule
```

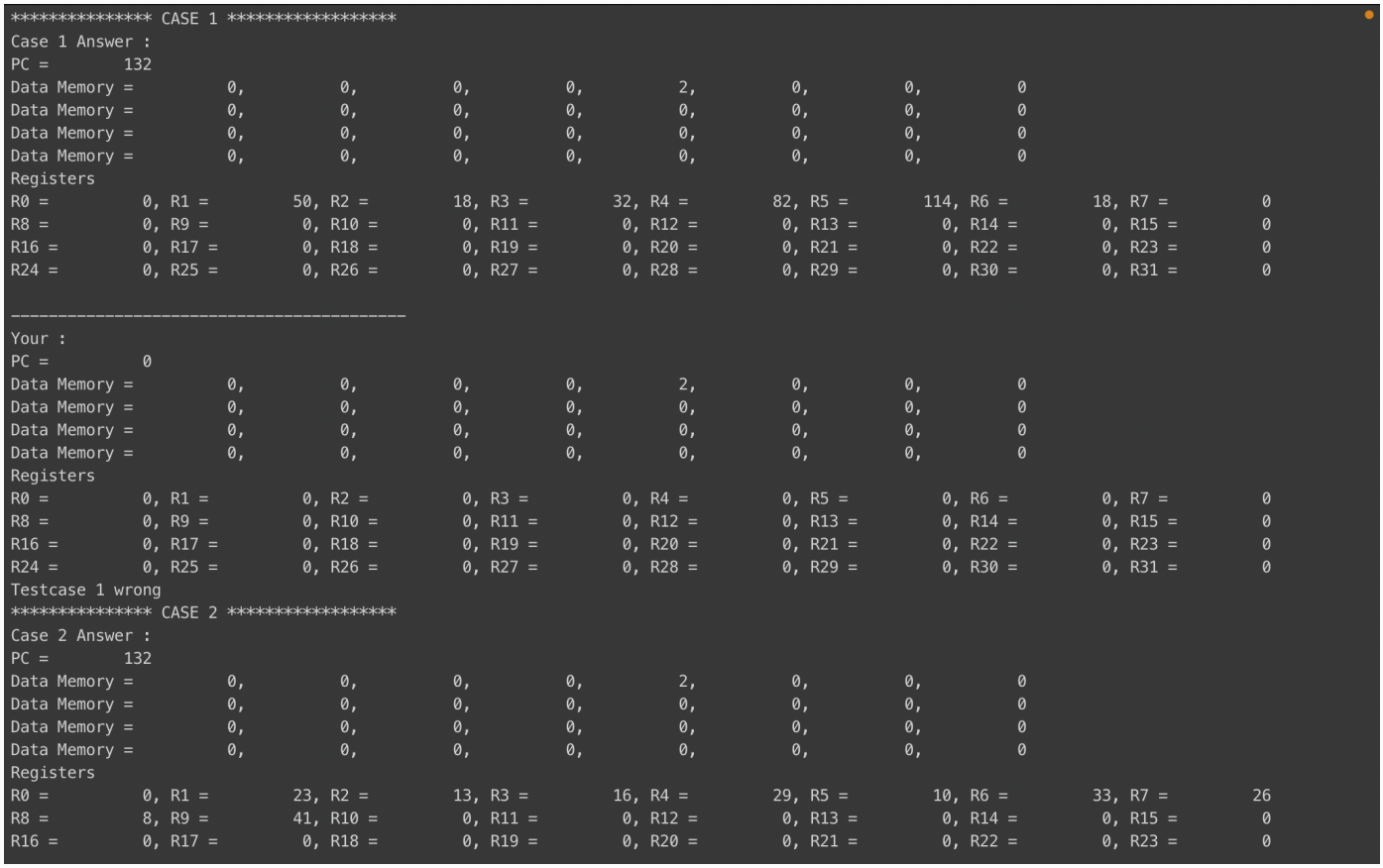
Since we are allowed to use operators directly rather than combining 32 1-bit ALUs to produce a 32 bit ALU, we decided to make the code shorter by using operators directly.

PART 2 : RESULTS

```
==========================================
********** CASE 1 **********
Testcase 1 pass
********** CASE 2 **********
Testcase 2 pass
********** CASE 3 **********
Testcase 3 pass
********** CASE 4 **********
Testcase 4 pass
********** CASE 5 **********
Testcase 5 pass
********** CASE 6 **********
Testcase 6 pass
********** CASE 7 **********
Testcase 7 pass
********** CASE 8 **********
Testcase 8 pass
********** CASE 9 **********
Testcase 9 pass
********** CASE 10 **********
Testcase 10 pass
********** CASE 11 **********
Testcase 11 pass
********** CASE 12 **********
Testcase 12 pass
********** CASE 13 **********
Testcase 13 pass
==========================================
Basic Score:30
Medium Score:40
Advanced Score:30
Total Score:100
```

PART 3 : ENCOUNTERED PROBLEMS

At first when we both finished writing the code, the result wasn't ideal, all the values of the registers were 0s. And we asked our classmates but they didn't have the same problem; the initial result was as the following picture.



After some examination, we realized it was the problem in the pipeline module, which the reset signals could not be processed correctly, and some typo in the wires causing the problem. After fixing those, we could get a 60% of the total grade.

Then after staring at GTKWave for some time, we finally fixed the problem.