國立臺灣科技大學

電機工程系

計算機組織

作業報告

PA3

四電機三乙 | B10707128 | 劉杰閎

● Part I

a. R_PipelineCPU.v

```verilog
module R_PipelineCPU(
    // Outputs
    output  wire    [31:0]  AddrOut,
    // Inputs
    input   wire    [31:0]  AddrIn,
    input   wire            clk
);

//  Instruction Memory
    wire [31:0] Instr;
    wire [31:0] Instr_out;

// ID/EX.
    wire [31:0] ALU_result;
    wire [31:0] RsData;
    wire [31:0] RsData_out;
    wire [31:0] RtData;
    wire [31:0] RtData_out;
    wire [1:0] ALUOp;
    wire [1:0] ALUOp_out;
    wire [5:0] funct_out;
    wire [4:0] shamt_out;
    wire [5:0] Funct;
    wire [4:0] RdAddr_out;
    wire RegWrite;
    wire RegWrite_out;

// EX/MEM
    wire RegWrite_mem_out;
    wire [31:0] ALU_result_mem_out;
    wire [4:0] RdAddr_mem_out;
```

```verilog
// EX/MEM
    wire RegWrite_wb_out;
    wire [31:0] ALU_result_wb_out;
    wire [4:0] RdAddr_wb_out;
```

```verilog
    IM_Instr_Memory(
        // Outputs
        .Instr(Instr),
        // Inputs
        .InstrAddr(AddrIn)
    );
    Adder adder(
        // Outputs
        .AddrOut(AddrOut),
        // Inputs
        .AddrIn(AddrIn)
    );
    IF_ID_Fetch_Decode(
        .Instr(Instr),
        .InstrOut(Instr_out),
        .clk(clk)
    );
```

```verilog
    ID_EX_Decode_Execute(
        // Inputs
        .RegWrite_in(RegWrite),
        .clk(clk),
        .ALUOp_in(ALUOp),
        .RsData_in(RsData),
        .RtData_in(RtData),
        .funct_in(Instr_out[5:0]),
        .shamt_in(Instr_out[10:6]),
        .RdAddr_in(Instr_out[15:11]),
        // Outputs
        .ALUOp_out(ALUOp_out),
        .RsData_out(RsData_out),
        .RtData_out(RtData_out),
        .funct_out(funct_out),
        .shamt_out(shamt_out),
        .RdAddr_out(RdAddr_out),
        .RegWrite_out(RegWrite_out)
    );

    Control_controller(
        // Inputs
        .OpCode(Instr_out[31:26]),
        // Outputs
        .RegWrite(RegWrite), // To ID/EX.
        .ALUOp(ALUOp) // To ID/EX.
    );

    /*
     * Declaration of Register File.
     * CAUTION: DONT MODIFY THE NAME.
     */
    RF_Register_File(
        // Outputs
        .RsData(RsData),// TO ID/EX.
        .RtData(RtData),// TO ID/EX.
        // Inputs
        .clk(clk),
        .RegWrite(RegWrite_wb_out), // Fr
        .RsAddr(Instr_out[25:21]),
        .RtAddr(Instr_out[20:16]),
        .RdAddr(RdAddr_wb_out),
        .RdData(ALU_result_wb_out) // Fro
    );
```

```
    ALU alu(
        // Inputs
        .Src1(RsData_out),
        .Src2(RtData_out),
        .Shamt(shamt_out),
        .Funct(Funct),
        // Outputs
        .result(ALU_result)
    );

    ALU_Control ALU_controller(
        // Inputs
        .funct(funct_out),
        .ALUOp(ALUOp_out),
        // Outputs
        .Funct(Funct)
    );

    EX_MEM Execute_Memory(
        // Inputs
        .clk(clk),
        .ALU_result_in(ALU_result),
        .RdAddr_in(RdAddr_out),
        .RegWrite_in(RegWrite_out),
        //Outputs
        .RdAddr_out(RdAddr_mem_out),
        .ALU_result_out(ALU_result_mem_out),
        .RegWrite_out(RegWrite_mem_out)
    );

    MEM_WB Memory_WriteBack(
        // Inputs
        .clk(clk),
        .ALU_result_in(ALU_result_mem_out),
        .RdAddr_in(RdAddr_mem_out),
        .RegWrite_in(RegWrite_mem_out),
        //Outputs
        .RdAddr_out(RdAddr_wb_out),
        .ALU_result_out(ALU_result_wb_out),
        .RegWrite_out(RegWrite_wb_out)
    );

endmodule
```

tb_R_formatCPU 與題目所提供之檔案相同，因篇幅限制而不另行截圖。這個 R_PipelineCPU 是將所有模組集大成之結果。我在裡面宣告了一些 wire 讓他去相互連接。而下圖為 RF.out 之輸出結果。右圖是經過執行後輸出出來的結果，利用 notepad++之 compare 工具，而對比於題目提供之檔案(左圖)可以發現完全相同，因此 R_PipelineCPU 到此順利做完。

RF.out

```
00000000
00000001
00000002
77777777
7f7f7f7f
f7f7f7f7
7fffffff
80000000
ffff0000
0000ffff
00000011
00000023
00000017
00000090
00000100
00000250
00000300
00000037
00000064
00000030
00000034
00000079
00000024
00040000
00000000
00000000
00000000
00000000
00000000
00000000
ffffffff
ffffffff
```

RF.out

```
// my
00000000
00000001
00000002
77777777
7f7f7f7f
f7f7f7f7
7fffffff
80000000
ffff0000
0000ffff
00000011
00000023
00000017
00000090
00000100
00000250
00000300
00000037
00000064
00000030
00000034
00000079
00000024
00040000
00000000
00000000
00000000
00000000
00000000
00000000
ffffffff
ffffffff
```

b. Instruction Memory

IM這個module其實很簡單，只要輸入Instruction Address，接著我就到該位置去抓Instruction，因為這個系統是BIG-ENDIAN，因此我抓的順序就會是如我程式碼的方式去抓{0,1,2,3}.

```
`define INSTR_MEM_SIZE  128 // Bytes
/*
 * Declaration of Instruction Memory for this project.
 * CAUTION: DONT MODIFY THE NAME.
 */
module IM(
    // Outputs
    output reg [31:0]   Instr,
    // Inputs
    input  [31:0]   InstrAddr
);

    /*
     * Declaration of instruction memory.
     * CAUTION: DONT MODIFY THE NAME AND SIZE.
     */
    reg [7:0]   InstrMem[0:`INSTR_MEM_SIZE - 1];

    always@(InstrAddr)
        begin
            Instr[31:0] = {InstrMem[InstrAddr], InstrMem[InstrAddr+1],InstrMem[InstrAddr+2],InstrMem[InstrAddr+3]};
        end

endmodule
```

c. Register File

RM這個module其實不難，只要判斷RegWrite是否為1，來決定是否可以將值寫入Reg，那其他部分就是到Register的address去抓值去輸出。我將RsData與RtData使用assign而非放在always裡面是因為我發現放在裡面會等到正緣觸發才把值送入ALU，那這樣的話就無法達到我們想要的效果了。

d. Adder

Adder所做的事情非常簡單，因為在R-type的AdderOut僅僅需要能夠將AddrIn+4，因此我就只做了加4的動作，然後輸出。(我的加法是unsigned的加法，因為Address沒有負的。)

```
`define REG_MEM_SIZE    32  // Words

/*
 * Declaration of Register File for this project.
 * CAUTION: DONT MODIFY THE NAME.
 */
module RF(
    // Outputs
    output [31:0] RsData,
    output [31:0] RtData,
    // Inputs
    input RegWrite,
    input clk,
    input [4:0]  RsAddr,
    input [4:0]  RtAddr,
    input [4:0]  RdAddr,
    input [31:0] RdData
);

    /*
     * Declaration of inner register.
     * CAUTION: DONT MODIFY THE NAME AND SIZE.
     */
    reg [31:0]R[0:`REG_MEM_SIZE - 1];

    assign  RsData  = R[RsAddr];
    assign  RtData  = R[RtAddr];      You, 3 days

    always@(posedge clk)
        begin
            if(RegWrite == 1)
                begin
                    R[RdAddr] = RdData;
                end
            else
                begin
                    R[RdAddr] = R[RdAddr];
                end
        end


endmodule
```

```
Adder.v
You, 3 days ago | 1 author (You)
module Adder
(
    input   [31:0]  AddrIn,
    output  [31:0]  AddrOut
);

    assign AddrOut = AddrIn + 32'd 4;

endmodule
```

e. ALU

ALU 主要是用來做Src1 and Src2的運算，由輸入的Funct來決定要做甚麼工作。

```
`define addu 6'b 001011
`define subu 6'b 001101
`define AND  6'b 010010
`define sll  6'b 100110

module ALU(
    input    [31:0]  Src1,
    input    [31:0]  Src2,
    input    [4:0]   Shamt,
    input    [5:0]   Funct,
    output  reg  [31:0]  result
);

always@(Funct or Shamt or Src1 or Src2)
    begin
        case (Funct)
            `addu : result = Src1 +  Src2;
            `subu : result = Src1 -  Src2;
            `AND  : result = Src1 &  Src2;
            `sll  : result = Src1 << Shamt;
            default: result = result; // if
        endcase
    end


endmodule      You, 3 days ago • part1 is 87%
```

```
You, 3 days ago | 1 author (You)
`define addu 6'b 001011
`define subu 6'b 001101
`define AND  6'b 010010
`define sll  6'b 100110

`define input_addu 6'b 001011
`define input_subu 6'b 001101
`define input_and  6'b 010010
`define input_sll  6'b 100110
`define R_type 2'b10

module ALU_Control(
    input     [5:0] funct,
    input     [1:0] ALUOp,
    output  reg   [5:0] Funct
);
    You, 3 days ago • part1 is 87% finished, only
always@(funct or ALUOp)
    begin
        case(ALUOp)
            `R_type:
                begin
                    case(funct)
                        `input_addu : Funct = `addu;
                        `input_subu : Funct = `subu;
                        `input_and  : Funct = `AND;
                        `input_sll  : Funct = `sll;
                        default: Funct = 0;
                    endcase
                end
            default:    Funct = 0;
        endcase
    end
endmodule
```

f. ALU_Control

ALU_Control主要是用來控制ALU工作與否，及將instr的指令轉為ALU懂得function code。比較特別的是在上一個作業的input_addu與ALU所要求的funct code不同，而這次作業的code是相同的，導致我在debug時花了一些時間才發現。

g. Control

Control這個module在R-type沒什麼太複雜的工作，只要依照Opcode的要求，來確認會不會把值寫入Reg裡面，來決定是否要送RegWrite的訊號。而因為R-type的所有指令都會使用到ALU，因此我們ALUOp只要是對的Opcode，我們一律送2‘b10.

```
Control.v
You, 3 days ago | 1 author (You)
module Control(
    input   [5:0]   OpCode,
    output reg          RegWrite,
    output reg  [1:0]   ALUOp
);

always@(OpCode)
    begin
        case (OpCode)
            6'd 4:
                begin       You, 3 days ago • part1 is
                    RegWrite =  1'b 1; // R format.
                    ALUOp = 2'b10;
                end
            default:
                begin
                    RegWrite = 0; // no this opcode.
                    ALUOp = 2'b00;
                end
        endcase

    end

endmodule
```

h. IF/ID, ID/EX, EX/MEM, MEM/WB

```
You, 3 days ago | 1 author (You)
module ID_EX(       You, 3 days ago •
    input RegWrite_in, // WB
    input clk,
    input [1:0] ALUOp_in, // EX
    input [5:0] funct_in,
    input [4:0] shamt_in,
    input [4:0] RdAddr_in,
    input [31:0] RsData_in,
    input [31:0] RtData_in,
    output reg [1:0] ALUOp_out, //EX
    output reg [31:0] RsData_out,
    output reg [31:0] RtData_out,
    output reg [4:0] RdAddr_out,
    output reg [5:0] funct_out,
    output reg [4:0] shamt_out,
    output reg RegWrite_out // WB
);

    reg WB;
    reg [1:0] EX;
    reg [5:0] RdAddr_reg;
    reg [5:0] funct_reg;
    reg [5:0] shamt_reg;
    reg [31:0] RsData_reg;
    reg [31:0] RtData_reg;
```

```
always@(posedge clk or negedge clk)
    begin
        if(clk == 1) // put them in to th
            begin
                WB = RegWrite_in;
                EX = ALUOp_in;
                RdAddr_reg = RdAddr_in;
                funct_reg = funct_in;
                shamt_reg = shamt_in;
                RsData_reg = RsData_in;
                RtData_reg = RtData_in;
            end
        else
            begin
                RegWrite_out = WB;
                ALUOp_out = EX;
                RdAddr_out = RdAddr_reg;
                funct_out = funct_reg;
                shamt_out = shamt_reg;
                RsData_out = RsData_reg;
                RtData_out = RtData_reg;
            end
    end
endmodule       You, 3 days ago • part1 is
```

IF_ID.v

You, 3 days ago | 1 author (You)

```
module IF_ID(
    input [31:0] Instr,
    input clk,
    output reg [31:0] InstrOut
);
    reg [31:0] Instr_reg;

always@(posedge clk or negedge clk)
    begin
        if(clk == 1)  //when posedge
            Instr_reg = Instr;
        else // when negedge clk, out
            InstrOut = Instr_reg;
    end
endmodule
```

You, 3 days ago | 1 author (You)

```
module MEM_WB(
    input clk,
    input RegWrite_in, // WB
    input [31:0] ALU_result_in,
    input [4:0] RdAddr_in,
    output reg [4:0] RdAddr_out,
    output reg [31:0] ALU_result_out,
    output reg RegWrite_out // WB
);
    reg WB;
    reg [4:0] RdAddr_reg;
    reg [31:0] ALU_result_reg;
    reg RegWrite_reg;

always@(posedge clk or negedge clk)
    begin
        if(clk == 1) // put them in to the reg
            begin
                WB = RegWrite_in;
                RdAddr_reg = RdAddr_in;
                ALU_result_reg = ALU_result_in;
            end
        else
            begin
                RegWrite_out = WB;
                RdAddr_out = RdAddr_reg;
                ALU_result_out = ALU_result_reg;
            end
    end
endmodule
```

EX_MEM.v

You, seconds ago | 1 author (You)

```
module EX_MEM(
    input clk,
    input RegWrite_in, // WB
    input [31:0] ALU_result_in,
    input [4:0] RdAddr_in,
    output reg [4:0] RdAddr_out,
    output reg [31:0] ALU_result_out,
    output reg RegWrite_out // WB
);
    reg WB;
    reg [4:0] RdAddr_reg;
    reg [31:0] ALU_result_reg;

always@(posedge clk or negedge clk)
    begin
        if(clk == 1) // put them in to the reg
            begin
                WB = RegWrite_in;
                RdAddr_reg = RdAddr_in;
                ALU_result_reg = ALU_result_in;

            end
        else
            begin
                RegWrite_out = WB;
                RdAddr_out = RdAddr_reg;
                ALU_result_out = ALU_result_reg;

            end
    end
```

這幾個module都是Pipeline register.因為性質相同，且無任何特別的，因此在這個地方，就將他們放上來，所有的pipeline register都是正緣觸發，將值傳入register,當負緣的時候才更新輸出的值。確保不會影響到下一刻的output.

● Part II.

a. I_ PipelineCPU

```fortran
module I_PipelineCPU(
    // Outputs
    output  wire    [31:0]  AddrOut,
    // Inputs
    input   wire    [31:0]  AddrIn,
    input   wire            clk
);

//  Instruction Memory
    wire [31:0] Instr;
    wire [31:0] Instr_out;

// ID/EX.

    // WB

    wire RegWrite;
    wire MemtoReg;
    wire MemtoReg_out;
    wire RegWrite_out;


    // memory.

    wire MemWrite;
    wire MemWrite_out;

    wire MemRead;
    wire MemRead_out;

    // EX.
    wire [1:0] ALUOp;
    wire [1:0] ALUOp_out;

    wire RegDst;
    wire RegDst_out;

    wire ALUSrc;
    wire ALSrc_out;
```

```fortran
    // Others.
    wire [31:0] RsData;
    wire [31:0] RsData_out;
    wire [31:0] RtData;
    wire [31:0] RtData_out;
    wire [31:0] Sign_Extend; // immed
    wire [31:0] immediate_out;
    wire [4:0] RdAddr_out;
    wire [4:0] RtAddr_out;

// EX/MEM

    //WB.
    wire RegWrite_mem_out;
    wire MemtoReg_mem_out;
    //Memory
    wire MemWrite_mem_out;
    wire MemRead_mem_out;
    // Others.
    wire [31:0] ALU_result;
    wire [31:0] MemAddr;

    wire [31:0] MemWriteData; // RtDa

    wire [4:0] RdAddr_mem_out;

// MEM/WB

    // WB.
    wire RegWrite_wb_out;
    wire MemtoReg_wb_out;
    // Others
    wire [31:0] MemAddr_wb_out;
    wire [4:0] RdAddr_wb_out;
    wire [31:0] MemReadData;
    wire [31:0] MemReadData_wb_out;
```

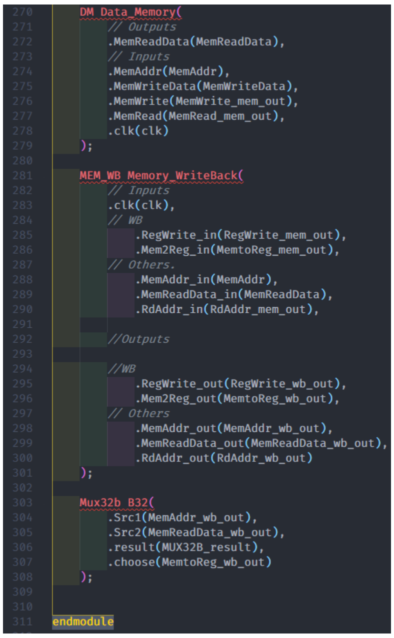
```fortran
// ALU controller
    wire [5:0]Funct;
// MUX
    wire [31:0] MUX32A_result;
    wire [31:0] MUX32B_result;
    wire [4:0]  MUX5_result;

    /*
     * Declaration of Instruction
     * CAUTION: DONT MODIFY THE M
     */
    IM_Instr_Memory(
        // Outputs
        .Instr(Instr),
        // Inputs
        .InstrAddr(AddrIn)
    );

    Adder adder(
        // Outputs        You, 3 da
        .AddrOut(AddrOut),
        // Inputs
        .AddrIn(AddrIn)
    );

    IF_ID_Fetch_Decode(
        .Instr(Instr),
        .InstrOut(Instr_out),
        .clk(clk)
    );

    Control_controller(
        .OpCode(Instr_out[31:26])
        .RegWrite(RegWrite),
        .RegDst(RegDst),
        .ALUSrc(ALUSrc),
        .MemWrite(MemWrite),
        .MemRead(MemRead),
        .MemtoReg(MemtoReg),
        .ALUOp(ALUOp)
    );
```

```fortran
    RF_Register_File(
        // Outputs
        .RsData(RsData),// TO ID/EX.
        .RtData(RtData),// TO ID/EX.
        // Inputs
        .clk(clk),
        .RegWrite(RegWrite_wb_out), // From MEM/WB.
        .RsAddr(Instr_out[25:21]),
        .RtAddr(Instr_out[20:16]),
        .RdAddr(RdAddr_wb_out),
        .RdData(MUX32B_result) // From MEM/WB.
    );

    assign Sign_Extend[31:0] = Instr_out[15]?{16'hFFFF,Instr_out[15:0]}:{16'h0000,Instr_out[15:0]};
```

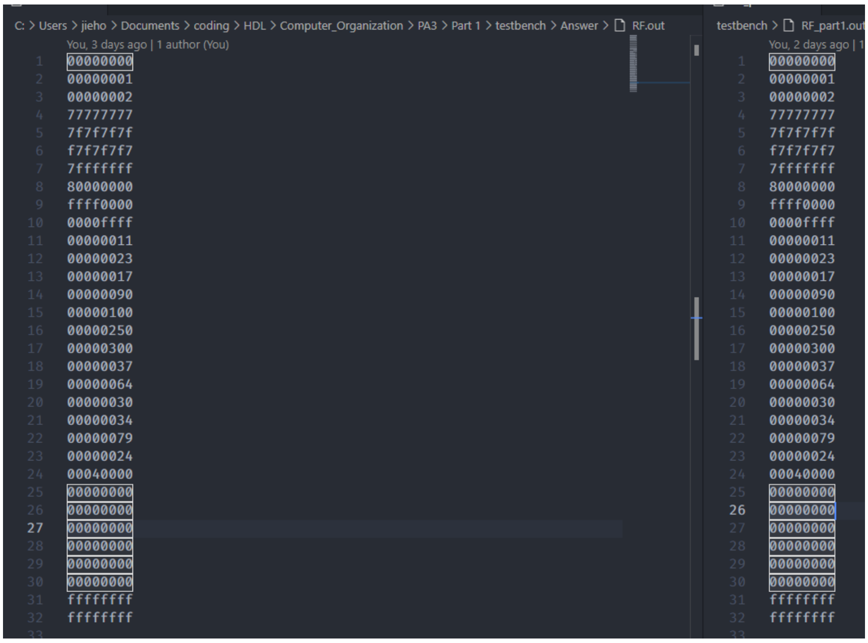
```fortran
    ID_EX_Decode_Execute(
        // Inputs
        .clk(clk),
        //WB
            .RegWrite_in(RegWrite),
            .Mem2Reg_in(MemtoReg),
        // Memory
            .MemRead_in(MemRead),
            .MemWrite_in(MemWrite),
        // EX
            .ALUOp_in(ALUOp),
            .RegDst_in(RegDst),
            .ALU_Src_in(ALUSrc),        You, 2
        // Others.
            .RsData_in(RsData),
            .RtData_in(RtData),
            .immediate_in(Sign_Extend),
            .RdAddr_in(Instr_out[15:11]),
            .RtAddr_in(Instr_out[20:16]),

        // Outputs

        //WB
            .RegWrite_out(RegWrite_out),
            .Mem2Reg_out(MemtoReg_out),
        // Memory
            .MemRead_out(MemRead_out),
            .MemWrite_out(MemWrite_out),
        // EX
            .ALUOp_out(ALUOp_out),
            .RegDst_out(RegDst_out),
            .ALU_Src_out(ALUSrc_out),
        // Others
            .RsData_out(RsData_out),
            .RtData_out(RtData_out),
            .immediate_out(immediate_out),
            .RdAddr_out(RdAddr_out),
            .RtAddr_out(RtAddr_out)
    );
```

```fortran
    Mux32b A32(
        .Src1(RtData_out),
        .Src2(immediate_out),
        .result(MUX32A_result),
        .choose(ALUSrc_out)
    );

    ALU alu(
        // Inputs
        .Src1(RsData_out),
        .Src2(MUX32A_result),
        .Shamt(immediate_out[10:6]),
        .Funct(Funct),
        // Outputs
        .result(ALU_result)
    );

    ALU_Control ALU_controller(
        // Inputs
        .funct(immediate_out[5:0]),
        .ALUOp(ALUOp_out),
        // Outputs
        .Funct(Funct)
    );

    Mux5b mux5b(
        .Src1(RtAddr_out),
        .Src2(RdAddr_out),
        .choose(RegDst_out),
        .result(MUX5_result)
    );
```

```fortran
    EX_MEM_Execute_Memory(
        // Inputs
            .clk(clk),
        // WB
            .RegWrite_in(RegWrite_out),
            .Mem2Reg_in(MemtoReg_out),
        // MEM
            .MemRead_in(MemRead_out),
            .MemWrite_in(MemWrite_out),
        // Others
            .ALU_result_in(ALU_result),
            .RtData_in(RtData_out),
            .RdAddr_in(MUX5_result),

        //Outputs

        // WB
            .RegWrite_out(RegWrite_mem_out),
            .Mem2Reg_out(MemtoReg_mem_out),
        // Memory
            .MemRead_out(MemRead_mem_out),
            .MemWrite_out(MemWrite_mem_out),
        // Others
            .ALU_result_out(MemAddr),
            .RtData_out(MemWriteData),
            .RdAddr_out(RdAddr_mem_out)
    );
```



```
DM Data_Memory(
    // Outputs
    .MemReadData(MemReadData),
    // Inputs
    .MemAddr(MemAddr),
    .MemWriteData(MemWriteData),
    .MemWrite(MemWrite_mem_out),
    .MemRead(MemRead_mem_out),
    .clk(clk)
);

MEM_WB Memory_WriteBack(
    // Inputs
    .clk(clk),
    // WB
        .RegWrite_in(RegWrite_mem_out),
        .Mem2Reg_in(MemtoReg_mem_out),
    // Others.
        .MemAddr_in(MemAddr),
        .MemReadData_in(MemReadData),
        .RdAddr_in(RdAddr_mem_out),

    //Outputs

    //WB
        .RegWrite_out(RegWrite_wb_out),
        .Mem2Reg_out(MemtoReg_wb_out),
    // Others
        .MemAddr_out(MemAddr_wb_out),
        .MemReadData_out(MemReadData_wb_out),
        .RdAddr_out(RdAddr_wb_out)
);

Mux32b B32(
    .Src1(MemAddr_wb_out),
    .Src2(MemReadData_wb_out),
    .result(MUX32B_result),
    .choose(MemtoReg_wb_out)
);


endmodule
```



C: > Users > jieho > Documents > coding > HDL > Computer_Organization > PA3 > Part 1 > testbench > Answer > RF.out

You, 3 days ago | 1 author (You)

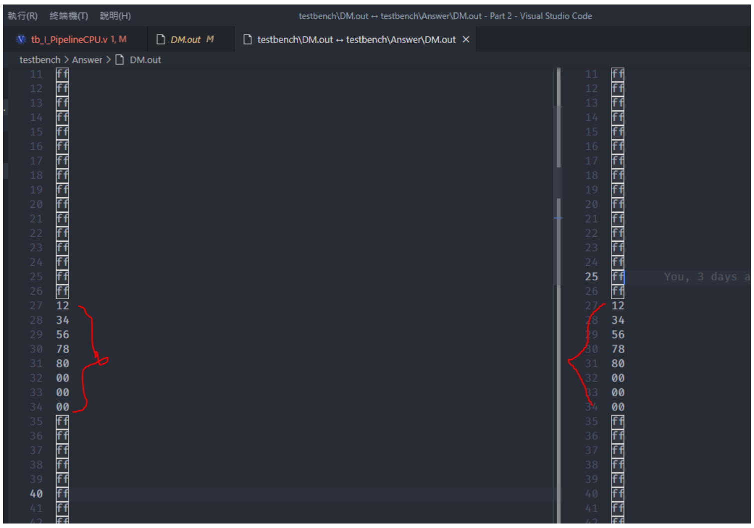
```
00000000
00000001
00000002
77777777
7f7f7f7f
f7f7f7f7
7fffffff
80000000
ffff0000
0000ffff
00000011
00000023
00000017
00000090
00000100
00000250
00000300
00000037
00000064
00000030
00000034
00000079
00000024
00040000
00000000
00000000
00000000
00000000
00000000
00000000
ffffffff
ffffffff
```

testbench > RF_part1.out

You, 2 days ago | 1

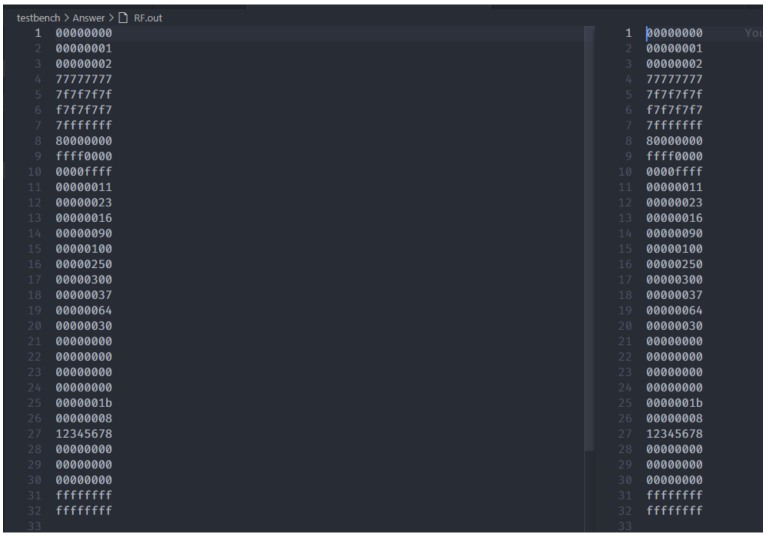
```
00000000
00000001
00000002
77777777
7f7f7f7f
f7f7f7f7
7fffffff
80000000
ffff0000
0000ffff
00000011
00000023
00000017
00000090
00000100
00000250
00000300
00000037
00000064
00000030
00000034
00000079
00000024
00040000
00000000
00000000
00000000
00000000
00000000
00000000
ffffffff
ffffffff
```

I_PipelineCPU，testbench 因篇幅限制而不特別截圖。因此在這個地方，只放上 DM.out 與 RF.out 來確認模擬結果是正確的。而下方左圖是 DM.out 經過執行後輸出出來的結果，而對比於題目提供之檔案(右圖)可以發現完全相同，除了 27~34 以外，其他結果皆為 FF。上方右圖 RF.out 是經過執行後輸出出來的結果，而對比於題目提供之檔案(左圖)可以發現完全相同。



```
11 ff
12 ff
13 ff
14 ff
15 ff
16 ff
17 ff
18 ff
19 ff
20 ff
21 ff
22 ff
23 ff
24 ff
25 ff
26 ff
27 12
28 34
29 56
30 78
31 80
32 00
33 00
34 00
35 ff
36 ff
37 ff
38 ff
39 ff
40 ff
41 ff
```

```
11 ff
12 ff
13 ff
14 ff
15 ff
16 ff
17 ff
18 ff
19 ff
20 ff
21 ff
22 ff
23 ff
24 ff
25 ff    You, 3 days a
26 ff
27 12
28 34
29 56
30 78
31 80
32 00
33 00
34 00
35 ff
36 ff
37 ff
38 ff
39 ff
40 ff
41 ff
```



testbench > Answer > RF.out

```
00000000
00000001
00000002
77777777
7f7f7f7f
f7f7f7f7
7fffffff
80000000
ffff0000
0000ffff
00000011
00000023
00000016
00000090
00000100
00000250
00000300
00000037
00000064
00000030
00000000
00000000
00000000
00000000
0000001b
00000008
12345678
00000000
00000000
00000000
ffffffff
ffffffff
```

```
00000000
00000001
00000002
77777777
7f7f7f7f
f7f7f7f7
7fffffff
80000000
ffff0000
0000ffff
00000011
00000023
00000016
00000090
00000100
00000250
00000300
00000037
00000064
00000030
00000000
00000000
00000000
00000000
0000001b
00000008
12345678
00000000
00000000
00000000
ffffffff
ffffffff
```

接著實測R_format在I-type CPU上模擬之結果。可以看到完全相等，因此到這裡可以確認成功。

b. Instruction Memory

IM這個module其實很簡單，只要輸入Instruction Address，接著我就到該位置去抓Instruction，因為這個系統是BIG-ENDIAN，因此我抓的順序就會是如我程式碼的方式去抓{0,1,2,3}.

```
`define INSTR_MEM_SIZE  128 // Bytes
/*
 * Declaration of Instruction Memory for this project.
 * CAUTION: DONT MODIFY THE NAME.
 */
module IM(
    // Outputs
    output reg [31:0]   Instr,
    // Inputs
    input  [31:0]   InstrAddr
);
    /*
     * Declaration of instruction memory.
     * CAUTION: DONT MODIFY THE NAME AND SIZE.
     */
    reg [7:0]   InstrMem[0:`INSTR_MEM_SIZE - 1];

    always@(InstrAddr)
        begin
            Instr[31:0] = {InstrMem[InstrAddr], InstrMem[InstrAddr+1],InstrMem[InstrAddr+2],InstrMem[InstrAddr+3]};
        end

endmodule       You, 3 days ago • part1 is 87% finished, only left test bench
```

c. Register File

RM這個module其實不難，只要判斷RegWrite是否為1，來決定是否可以將值寫入Reg，那其他部分就是到Register的address去抓值去輸出。我將RsData與RtData使用assign而非放在always裡面是因為我發現放在裡面會等到正緣觸發才把值送入ALU，那這樣的話就無法達到我們想要的效果了。

```
`define REG_MEM_SIZE    32  // Words

/*
 * Declaration of Register File for this project.
 * CAUTION: DONT MODIFY THE NAME.
 */
module RF(
    // Outputs
    output [31:0] RsData,
    output [31:0] RtData,
    // Inputs
    input RegWrite,
    input clk,
    input [4:0]  RsAddr,
    input [4:0]  RtAddr,
    input [4:0]  RdAddr,
    input [31:0] RdData
);

    /*
     * Declaration of inner register.
     * CAUTION: DONT MODIFY THE NAME AND SIZE.
     */
    reg [31:0]R[0:`REG_MEM_SIZE - 1];

    assign  RsData  = R[RsAddr];
    assign  RtData  = R[RtAddr];        You, 3 days

    always@(posedge clk)
        begin
            if(RegWrite == 1)
                begin
                    R[RdAddr] = RdData;
                end
            else
                begin
                    R[RdAddr] = R[RdAddr];
                end
        end


endmodule
```

d. Adder

Adder所做的事情非常簡單，因為在R-type的AdderOut僅僅需要能夠將AddrIn+4，因此我就只做了加4的動作，然後輸出。(我的加法是unsigned的加法，因為Address沒有負的。)

Adder.v

You, 3 days ago | 1 author (You)

```
module Adder
(
    input   [31:0]  AddrIn,
    output  [31:0]  AddrOut
);

    assign AddrOut = AddrIn + 32'd 4;

endmodule
```

e. ALU

ALU 主要是用來做Src1 and Src2的運算，由輸入的Funct來決定要做甚麼工作。

```
`define addu 6'b 001011
`define subu 6'b 001101
`define AND  6'b 010010
`define sll  6'b 100110

module ALU(
    input    [31:0]  Src1,
    input    [31:0]  Src2,
    input    [4:0]   Shamt,
    input    [5:0]   Funct,
    output  reg  [31:0]  result
);

always@(Funct or Shamt or Src1 or Src2)
    begin
        case (Funct)
            `addu : result = Src1 +  Src2;
            `subu : result = Src1 -  Src2;
            `AND  : result = Src1 &  Src2;
            `sll  : result = Src1 << Shamt;
            default: result = result; // if
        endcase
    end


endmodule      You, 3 days ago • part1 is 87%
```

f. ALU_Control

ALU_Control主要是用來控制ALU工作與否，及將instr的指令轉為ALU懂得function code。比較特別的是在上一個作業的input_addu與ALU所要求的funct code不同，而這次作業的code是相同的，導致我在debug時花了一些時間才發現。

ALU_Control.v

You, 3 days ago | 1 author (You)

```
`define addu 6'b 001011
`define subu 6'b 001101
`define AND  6'b 010010
`define sll  6'b 100110
`define input_addu 6'b 001011
`define input_subu 6'b 001101
`define input_and  6'b 010010
`define input_sll  6'b 100110
`define R_type 2'b10
// I-type sub
`define I_type_sub 2'b00
// I-type add
`define I_type_add 2'b01
module ALU_Control(
    input     [5:0] funct,
    input     [1:0] ALUOp,
    output  reg   [5:0] Funct
);
always@(funct or ALUOp)
    begin
        case(ALUOp)
            `R_type:
                begin
                    case(funct)
                        `input_addu : Funct = `addu;
                        `input_subu : Funct = `subu;
                        `input_and  : Funct = `AND;
                        `input_sll  : Funct = `sll;
                        default: Funct = 0;
                    endcase
                end
            `I_type_sub:
                begin
                    Funct = `subu;
                end
            `I_type_add:
                begin
                    Funct = `addu;
                end
            default:;
        endcase
    end
endmodule
```

g. Data Memory

DM這個module其實也不難，只要判斷MemWrite跟MemRead是否為1，來決定是否可以將將值寫入Memory或是把Memory的值給讀出來。我將MemReadData使用assign而非放在always裡面，與RF的原因相同。我發現放在裡面會等到下個正緣觸發才把值送出，那這樣的話就無法達到我們想要的效果了。

```
`define DATA_MEM_SIZE   128 // Bytes

/*
 * Declaration of Data Memory for this project.
 * CAUTION: DONT MODIFY THE NAME.
 */
module DM(
    // Outputs
    output [31:0] MemReadData,
    // Inputs
    input [31:0] MemAddr,
    input [31:0] MemWriteData,
    input MemWrite,
    input MemRead,
    input clk
);

    /*
     * Declaration of data memory.
     * CAUTION: DONT MODIFY THE NAME AND SIZE.
     */
    reg [7:0]DataMem[0:`DATA_MEM_SIZE - 1];

    assign MemReadData = MemRead? {DataMem[MemAddr],DataMem[MemAddr+1],DataMem[MemAddr+2],DataMem[MemAddr+3]}:32'b0;
      You, a day ago • PART3 controller still working
    always@(posedge clk)
        begin
            if(MemWrite == 1)
                begin
                    {DataMem[MemAddr],DataMem[MemAddr+1],DataMem[MemAddr+2],DataMem[MemAddr+3]} = MemWriteData;
                end
            else;
        end

endmodule
```

h. Control

Control這個module在整個CPU裡面是一個非常重要的角色。他掌管整顆CPU現在要做甚麼，不要做甚麼。雖然他極其重要，但是其實沒有甚麼太複雜的工作，只要依照Opcode的要求，來決定是否要送各種訊號。而因為R-type的所有指令都會使用到ALU，因此我們ALUOp只要是對的Opcode，我們一律送2‘b10.而對於I-type指令來說，只會有加法跟減法，因此除了subiu以外(2‘b00)，其他的ALUOP都為2‘b01，剩下的是J-type指令，因為branch用到的是減法，因此ALUOP為0。

Control.v

You, 3 days ago | 1 author (You)

```
// I-type sub
`define I_type_sub 2'b00
// I-type add
`define I_type_add 2'b01
module Control(
    input   [5:0]   OpCode,
    output reg          RegWrite,
    output reg  [1:0]   ALUOp,
    output reg          RegDst,
    output reg          ALUSrc,
    output reg          MemWrite,
    output reg          MemRead,
    output reg          MemtoReg
);
```

You, 3 days ago • part1 is 87

```
always@(OpCode)
    begin
        case (OpCode)
            6'd 4:
                begin
                    RegWrite =  1'b 1;
                    ALUOp = 2'b 10;
                    RegDst = 1; // R for
                    ALUSrc = 0;
                    MemWrite = 0;
                    MemRead = 0;
                    MemtoReg = 0;
                end
            // I format.
            6'd 12: // addiu
                begin
                    RegWrite = 1'b 1;
                    ALUOp = `I_type_add;
                    RegDst = 0; // I for
                    ALUSrc = 1;
                    MemWrite = 0;
                    MemRead = 0;
                    MemtoReg = 0;
                end
            6'd 13: // subiu
                begin
                    RegWrite = 1'b 1;
                    ALUOp = `I_type_sub;
                    RegDst = 0; // I for
                    ALUSrc = 1;
                    MemWrite=0;
                    MemRead = 0;
                    MemtoReg = 0;
                end
            6'd 16: // sw
                begin
                    RegWrite = 1'b 0;
                    ALUOp = `I_type_add;
                    RegDst = 1'b x; // I
                    ALUSrc = 1;
                    MemWrite = 1;
                    MemRead = 0;
                    MemtoReg = 1'b x; //
                end
```

```
            6'd 16: // sw
                begin
                    RegWrite = 1'b 0;
                    ALUOp = `I_type_add;
                    RegDst = 1'b x; // I
                    ALUSrc = 1;
                    MemWrite = 1;
                    MemRead = 0;
                    MemtoReg = 1'b x; //
                end
            6'd 17: // lw
                begin
                    RegWrite = 1'b 1;
                    ALUOp = `I_type_add;
                    RegDst = 0; // I for
                    ALUSrc = 1;
                    MemWrite = 0;
                    MemRead = 1;
                    MemtoReg= 1;
                end
            default:
                begin
                    MemWrite = 0;
                    RegWrite = 0;
                end
        endcase

    end

endmodule
```

i. MUX

MUX這二個module其實非常簡單，一個是5bit,一個是32bit.只要判斷choose選的是多少，就決定輸出要送哪一個輸入出來。

Mux5b.v

You, 3 days ago | 1 author (You)

```
module Mux5b(
    input [4:0]Src1,//0
    input [4:0]Src2,//1
    input choose,
    output [4:0]result
);
assign result = choose? Src2:Src1;

endmodule
```

Mux32b.v

You, 3 days ago | 1 author (You)

```
module Mux32b(
    input [31:0]Src1, // 0
    input [31:0]Src2, // 1
    input choose,
    output [31:0]result
);

assign result = choose? Src2:Src1;

endmodule
```

i. IF/ID, ID/EX, EX/MEM, MEM/WB

```
module ID_EX(

    // Write Back.
    input RegWrite_in, // WB
    input Mem2Reg_in,  //WB
    output reg RegWrite_out, // WB
    output reg Mem2Reg_out,// WB
    // Memory
    input MemRead_in,
    input MemWrite_in,
    output reg MemWrite_out,
    output reg MemRead_out,
    // EX.
    input [1:0] ALUOp_in, // EX
    input RegDst_in,
    input ALU_Src_in,

    output reg [1:0] ALUOp_out, //EX
    output reg RegDst_out,
    output reg ALU_Src_out,
    // Others.
    input clk,
    input [4:0] RdAddr_in,
    input [4:0] RtAddr_in,
    input [31:0] RsData_in,
    input [31:0] RtData_in,
    input [31:0] immediate_in,

    output reg [31:0] immediate_out,
    output reg [31:0] RsData_out,
    output reg [31:0] RtData_out,
    output reg [4:0] RdAddr_out,
    output reg [4:0] RtAddr_out

);
```

```
    // Temp register part.

    // Write Back.
    reg RegWrite_reg;
    reg Mem2Reg_reg;
    // Memory
    reg MemRead_reg;
    reg MemWrite_reg;
    // Execution.
    reg [1:0] ALUOp_reg; // EX
    reg RegDst_reg;
    reg ALU_Src_reg;
    // Others.
    reg [5:0] RdAddr_reg;
    reg [4:0] RtAddr_reg;
    reg [31:0] RsData_reg;
    reg [31:0] RtData_reg;
    reg [31:0] immediate_reg;
```

```
always@(posedge clk or negedge clk)
    begin
        if(clk == 1) // put them in to the reg
            begin
                // Write Back.
                Mem2Reg_reg = Mem2Reg_in;
                RegWrite_reg = RegWrite_in;
                // Memory.
                MemRead_reg = MemRead_in;
                MemWrite_reg = MemWrite_in;
                // Execution
                ALUOp_reg = ALUOp_in;
                RegDst_reg = RegDst_in;
                ALU_Src_reg =ALU_Src_in;
                // Others.
                RdAddr_reg = RdAddr_in;
                RtAddr_reg = RtAddr_in;
                RsData_reg = RsData_in;
                RtData_reg = RtData_in;
                immediate_reg = immediate_in;
            end
        else
            begin
                // Write Back.
                RegWrite_out = RegWrite_reg;
                Mem2Reg_out = Mem2Reg_reg;
                // Memory.
                MemRead_out = MemRead_reg;
                MemWrite_out = MemWrite_reg;
                // Execution
                ALUOp_out = ALUOp_reg;
                RegDst_out = RegDst_reg;
                ALU_Src_out =ALU_Src_reg;
                // Others.
                RdAddr_out = RdAddr_reg;
                RtAddr_out = RtAddr_reg;
                RsData_out = RsData_reg;
                RtData_out = RtData_reg;
                immediate_out = immediate_reg;
            end
    end      You, 3 days ago • part1 is 87% finished

endmodule
```

IF_ID.v

You, 3 days ago | 1 author (You)

```
module IF_ID(
    input [31:0] Instr,
    input clk,
    output reg [31:0] InstrOut
);
    reg [31:0] Instr_reg;

always@(posedge clk or negedge clk)
    begin
        if(clk == 1)  //when posedge
            Instr_reg = Instr;
        else // when negedge clk, out
            InstrOut = Instr_reg;
    end
endmodule      You, 3 days ago • part
```

EX_MEM.v

You, 2 days ago | 1 author (You)

```
module EX_MEM(
    // Write Back.
    input RegWrite_in, // WB
    input Mem2Reg_in,  //WB
    output reg RegWrite_out, // WB
    output reg Mem2Reg_out,// WB
    // Memory
    input MemRead_in,
    input MemWrite_in,
    output reg MemWrite_out,
    output reg MemRead_out,
    // Others.
    input clk,
    input [31:0] ALU_result_in,
    input [4:0] RdAddr_in,
    input [31:0] RtData_in,
    output reg [31:0] RtData_out,
    output reg [4:0] RdAddr_out,
    output reg [31:0] ALU_result_out
);
```

```
always@(posedge clk or negedge clk)
    begin
        if(clk == 1) // put them in to the reg
            begin
                // Write Back.
                RegWrite_reg = RegWrite_in;
                Mem2Reg_reg = Mem2Reg_in;
                // Memory.
                RegWrite_reg = RegWrite_in;
                Mem2Reg_reg = Mem2Reg_in;
                MemRead_reg = MemRead_in;
                MemWrite_reg = MemWrite_in;
                // Others
                RdAddr_reg = RdAddr_in;
                ALU_result_reg = ALU_result_in;
                RtData_reg = RtData_in;
            end
        else
            begin
                // Write Back.
                RegWrite_out = RegWrite_reg;
                Mem2Reg_out = Mem2Reg_reg;
                // Memory
                RegWrite_out = RegWrite_reg;
                Mem2Reg_out = Mem2Reg_reg;
                MemRead_out = MemRead_reg;
                MemWrite_out = MemWrite_reg;
                // Others      You, 3 days ago •
                RdAddr_out = RdAddr_reg;
                ALU_result_out = ALU_result_reg;
                RtData_out = RtData_reg;
            end
    end
endmodule
```

```
    // Temp register part.
        You, 3 days ago • part
    // Write Back.
    reg RegWrite_reg;
    reg Mem2Reg_reg;
    // Memory
    reg MemRead_reg;
    reg MemWrite_reg;
    // Others.
    reg [5:0] RdAddr_reg;
    reg [31:0] RtData_reg;
    reg [31:0] ALU_result_reg;
```

```
You, 2 days ago | 1 author (You)
module MEM_WB(
    // Write Back.
    input RegWrite_in, // WB
    input Mem2Reg_in,  //WB
    output reg RegWrite_out, // WB
    output reg Mem2Reg_out,// WB
    // Others.
    input clk,
    input [31:0] MemAddr_in,
    input [4:0] RdAddr_in,
    input [31:0] MemReadData_in,
    output reg [31:0] MemReadData_out,
    output reg [4:0] RdAddr_out,
    output reg [31:0] MemAddr_out
);
    // Temp register part.
    // Write Back.
    reg RegWrite_reg;
    reg Mem2Reg_reg;
    // Others.
    reg [5:0] RdAddr_reg;
    reg [31:0] MemAddr_reg;
    reg [31:0] MemReadData_reg;
```

```
always@(posedge clk or negedge clk)
    begin
        if(clk == 1) // put them in to the reg
            begin
                // Write Back.
                RegWrite_reg = RegWrite_in;
                Mem2Reg_reg = Mem2Reg_in;

                // Others
                RdAddr_reg = RdAddr_in;
                MemAddr_reg = MemAddr_in;
                MemReadData_reg = MemReadData_in;
            end
        else
            begin
                // Write Back.
                RegWrite_out = RegWrite_reg;
                Mem2Reg_out = Mem2Reg_reg;

                // Others
                RdAddr_out = RdAddr_reg;
                MemAddr_out = MemAddr_reg;
                MemReadData_out = MemReadData_reg;
            end
    end


endmodule
```

這幾個module都是Pipeline register.因為性質相同，且無任何特別的，因此在這個地方，就將他們放上來，所有的pipeline register都是正緣觸發，將值傳入register,當負緣的時候才更新輸出的值。確保不會影響到下一刻的output.

● Part III.

a. FinalCPU

tb_FinalCPU 與題目所提供之檔案相同，因篇幅限制而不另行截圖。這個 Final 是將所有模組集大成之結果。我在裡面宣告了一些 wire 讓他去相互連接。這裡面比較特別的是有一個被我宣告的 wire 叫作 SignExtend,是去把輸進來的 16bit 的 Immediate 值，延長到 32bit。與 part2 不同的除了多了 hazard detection unit, forwarding unit 以外，在控制 ALU 的地方也有些許不同，則其他基本上都相同。

```
module FinalCPU(
    // Outputs
    output  wire            PCWrite,
    output  wire    [31:0]  AddrOut,
    // Inputs
    input   wire    [31:0]  AddrIn,
    input   wire            clk
);

// HazardDetectionUnit
    wire Stall;
    wire IF_ID_Write;

// Forwarding Unit
    wire [1:0] ForwardA;
    wire [1:0] ForwardB;
    wire [31:0] MUX3to1A_result;
    wire [31:0] MUX3to1B_result;
//  Instruction Memory
    wire [31:0] Instr;
    wire [31:0] Instr_out;
// ID/EX.

    // WB

    wire RegWrite;
    wire RegWrite_in; // MUXSTALL
    wire RegWrite_out;
    wire MemtoReg;
    wire MemtoReg_in;
    wire MemtoReg_out;
```

```
    wire MemtoReg_out;
    // memory.
    wire MemWrite;
    wire MemWrite_in;
    wire MemWrite_out;

    wire MemRead;
    wire MemRead_in;
    wire MemRead_out;

    // EX.
    wire [1:0] ALUOp;
    wire [1:0] ALUOp_in;
    wire [1:0] ALUOp_out;

    wire RegDst;
    wire RegDst_in;
    wire RegDst_out;

    wire ALUSrc;
    wire ALUSrc_in;
    wire ALUSrc_out;
```

```
    // Others.        You, 2 days a
    wire [31:0] RsData;
    wire [31:0] RsData_out;
    wire [31:0] RtData;
    wire [31:0] RtData_out;
    wire [31:0] Sign_Extend; //
    wire [31:0] immediate_out;
    wire [4:0] RdAddr_out;
    wire [4:0] RtAddr_out;
    wire [4:0] RsAddr_out;
// EX/MEM

    //WB.
    wire RegWrite_mem_out;
    wire MemtoReg_mem_out;
    //Memory
    wire MemWrite_mem_out;
    wire MemRead_mem_out;
    // Others.
    wire [31:0] ALU_result;
    wire [31:0] MemAddr;

    wire [31:0] MemWriteData; //

    wire [4:0] RdAddr_mem_out;
```

```
// MEM/WB

    // WB.
    wire RegWrite_wb_out;
    wire MemtoReg_wb_out;
    // Others
    wire [31:0] MemAddr_wb_out;
    wire [4:0] RdAddr_wb_out;
    wire [31:0] MemReadData;
    wire [31:0] MemReadData_wb_out;



// ALU controller
    wire [5:0]Funct;
// MUX
    wire [4:0]  MUX5_result;

    wire [31:0] MUX32A_result;
    wire [31:0] MUX32B_result;
```

```
    IM_Instr_Memory(
        // Outputs
        .Instr(Instr),
        // Inputs
        .InstrAddr(AddrIn)
    );

    Adder adder(
        // Outputs
        .AddrOut(AddrOut),
        // Inputs
        .AddrIn(AddrIn)
    );

    IF_ID_Fetch_Decode(
        .Instr(Instr),
        .InstrOut(Instr_out),
        .IF_ID_Write(IF_ID_Write),
        .clk(clk)
    );
```

```
    HazardDetectionUnit HDU(
        // Inputs.
        .ID_EX_MemRead(MemRead_out),
        .ID_EX_RegisterRt(RtAddr_out),
        .IF_ID_RegisterRs(Instr_out[25:21]),
        .IF_ID_RegisterRt(Instr_out[20:16]),
        // Outputs
        .Stall(Stall),
        .PCWrite(PCWrite),
        .IF_ID_Write(IF_ID_Write)
    );

    Control controller(
        // Inputs
        .OpCode(Instr_out[31:26]),
        // Outputs
        .RegWrite(RegWrite),
        .RegDst(RegDst),
        .ALUSrc(ALUSrc),
        .MemWrite(MemWrite),
        .MemRead(MemRead),
        .MemtoReg(MemtoReg),
        .ALUOp(ALUOp)
    );
```

```
    RF_Register_File(
        // Outputs
        .RsData(RsData),// TO ID/EX.
        .RtData(RtData),// TO ID/EX.
        // Inputs
        .clk(clk),
        .RegWrite(RegWrite_wb_out), // From MEM/WB.
        .RsAddr(Instr_out[25:21]),
        .RtAddr(Instr_out[20:16]),
        .RdAddr(RdAddr_wb_out),
        .RdData(MUX32B_result) // From MEM/WB.
    );
    assign Sign_Extend[31:0] = Instr_out[15]?{16'hFFFF,Instr_out[15:0]}:{16'h0000,Instr_out[15:0]};
```

```
    MuxStall_MS(
        // inputs.
        .RegDst_in(RegDst),
        .MemRead_in(MemRead),
        .MemtoReg_in(MemtoReg),
        .ALUOp_in(ALUOp),
        .MemWrite_in(MemWrite),
        .ALUSrc_in(ALUSrc),
        .RegWrite_in(RegWrite),
        .Stall_choose(Stall),
        // Outputs
        .RegDst_out(RegDst_in),
        .MemRead_out(MemRead_in),
        .MemtoReg_out(MemtoReg_in),
        .ALUOp_out(ALUOp_in),
        .MemWrite_out(MemWrite_in),
        .ALUSrc_out(ALUSrc_in),
        .RegWrite_out(RegWrite_in)
    );
```

```
    ID_EX_Decode_Execute(
        // Inputs
        .clk(clk),
        //WB
            .RegWrite_in(RegWrite_in),
            .Mem2Reg_in(MemtoReg_in),
        // Memory
            .MemRead_in(MemRead_in),
            .MemWrite_in(MemWrite_in),
        // EX
            .ALUOp_in(ALUOp_in),
            .RegDst_in(RegDst_in),
            .ALU_Src_in(ALUSrc_in),
        // Others.
            .RsData_in(RsData),
            .RtData_in(RtData),
            .immediate_in(Sign_Extend),
            .RdAddr_in(Instr_out[15:11]),
            .RtAddr_in(Instr_out[20:16]),
            .RsAddr_in(Instr_out[25:21]),
        // Outputs      You, 2 days ago • th
        //WB
            .RegWrite_out(RegWrite_out),
            .Mem2Reg_out(MemtoReg_out),
        // Memory
            .MemRead_out(MemRead_out),
            .MemWrite_out(MemWrite_out),
        // EX
            .ALUOp_out(ALUOp_out),
            .RegDst_out(RegDst_out),
            .ALU_Src_out(ALUSrc_out),
        // Others
            .RsData_out(RsData_out),
            .RtData_out(RtData_out),
            .immediate_out(immediate_out),
            .RdAddr_out(RdAddr_out),
            .RtAddr_out(RtAddr_out),
            .RsAddr_out(RsAddr_out)
    );
```

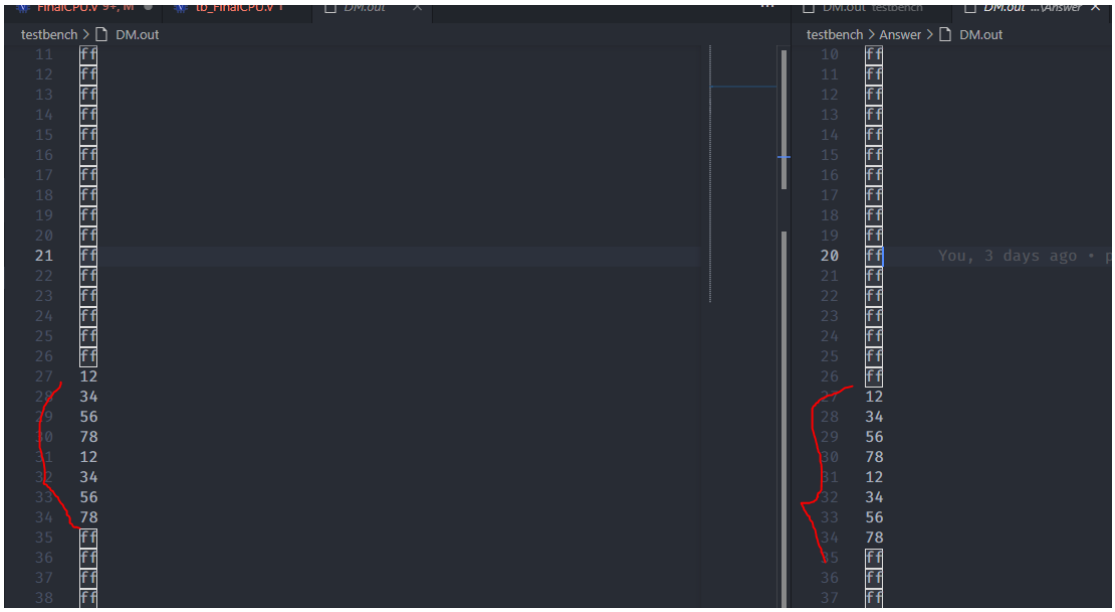
```
MUX3to1_A(
    .Src1(RsData_out),
    .Src2(MUX32B_result),
    .Src3(MemAddr),
    .choose(ForwardA),
    .result(MUX3to1A_result)
);

MUX3to1_B(
    .Src1(RtData_out),
    .Src2(MUX32B_result),
    .Src3(MemAddr),
    .choose(ForwardB),
    .result(MUX3to1B_result)
);

Mux32b_A32(
    .Src1(MUX3to1B_result),
    .Src2(immediate_out),
    .result(MUX32A_result),
    .choose(ALUSrc_out)
);

ALU alu(
    // Inputs
    .Src1(MUX3to1A_result),
    .Src2(MUX32A_result),
    .Shamt(immediate_out[10:6]),
    .Funct(Funct),
    // Outputs
    .result(ALU_result)
);
```

```
ALU_Control ALU_controller(
    // Inputs
    .funct(immediate_out[5:0]),
    .ALUOp(ALUOp_out),
    // Outputs
    .Funct(Funct)
);

Mux5b mux5b(
    .Src1(RtAddr_out),
    .Src2(RdAddr_out),
    .choose(RegDst_out),
    .result(MUX5_result)
);

ForwardingUnit FU(
    .EX_MEM_RegWrite(RegWrite_mem_out),
    .EX_MEM_RegisterRd(RdAddr_mem_out),
    .ID_EX_RegisterRs(RsAddr_out),
    .ID_EX_RegisterRt(RtAddr_out),
    .MEM_WB_RegWrite(RegWrite_wb_out),
    .MEM_WB_RegisterRd(RdAddr_wb_out),
    .ForwardA(ForwardA[1:0]),
    .ForwardB(ForwardB[1:0])
);
```

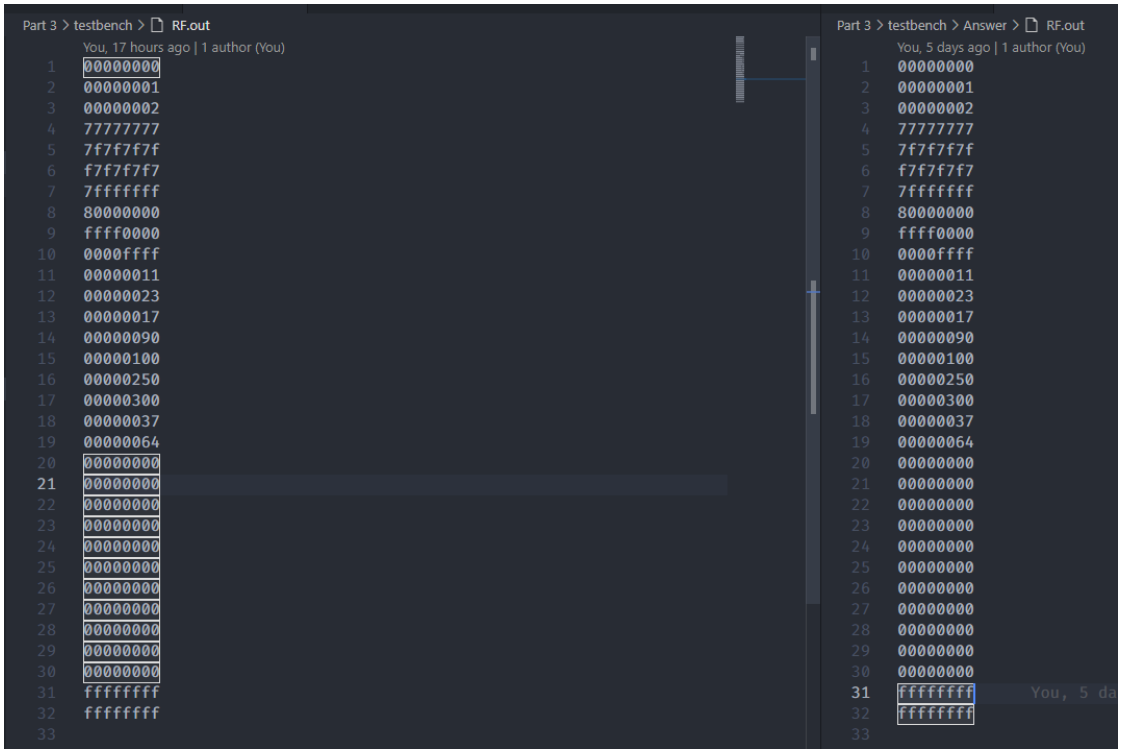
```
EX_MEM_Execute_Memory(        You, 2 days
    // Inputs
        .clk(clk),
    // WB
        .RegWrite_in(RegWrite_out),
        .Mem2Reg_in(MemtoReg_out),
    // MEM
        .MemRead_in(MemRead_out),
        .MemWrite_in(MemWrite_out),
    // Others
        .ALU_result_in(ALU_result),
        .RtData_in(MUX3to1B_result),
        .RdAddr_in(MUX5_result),

    //Outputs

    // WB
        .RegWrite_out(RegWrite_mem_out),
        .Mem2Reg_out(MemtoReg_mem_out),
    // Memory
        .MemRead_out(MemRead_mem_out),
        .MemWrite_out(MemWrite_mem_out),
    // Others
        .ALU_result_out(MemAddr),
        .RtData_out(MemWriteData),
        .RdAddr_out(RdAddr_mem_out)
);
```

```
DM_Data_Memory(
    // Outputs
    .MemReadData(MemReadData),
    // Inputs
    .MemAddr(MemAddr),
    .MemWriteData(MemWriteData),
    .MemWrite(MemWrite_mem_out),
    .MemRead(MemRead_mem_out),
    .clk(clk)
);

MEM_WB_Memory_WriteBack(
    // Inputs
    .clk(clk),
    // WB
        .RegWrite_in(RegWrite_mem_out),
        .Mem2Reg_in(MemtoReg_mem_out),
    // Others.
        .MemAddr_in(MemAddr),
        .MemReadData_in(MemReadData),
        .RdAddr_in(RdAddr_mem_out),

    //Outputs

    //WB
        .RegWrite_out(RegWrite_wb_out),
        .Mem2Reg_out(MemtoReg_wb_out),
    // Others
        .MemAddr_out(MemAddr_wb_out),
        .MemReadData_out(MemReadData_wb_out),
        .RdAddr_out(RdAddr_wb_out)
);

Mux32b_B32(
    .Src1(MemAddr_wb_out),
    .Src2(MemReadData_wb_out),
    .result(MUX32B_result),
    .choose(MemtoReg_wb_out)
);


endmodule
```

而下方右圖是DM.out經過執行後輸出出來的結果(全部都為FF) 而對比於題目提供之檔案(右圖)可以發現完全相同。



而下方右圖是RF.out經過執行後輸出出來的結果，而對比於題目提供之檔案(右圖)可以發現完全相同，因此Final到此順利做完。



下圖是Rtype指令在SimpleCPU上模擬後輸出的結果，我們可以對比part1的RF.out輸出，可以發現一模一樣，因此可以確認成功，

b. Instruction Memory

IM這個module其實很簡單，只要輸入Instruction Address，接著我就到該位置去抓Instruction，因為這個系統是BIG-ENDIAN，因此我抓的順序就會是如我程式碼的方式去抓{0,1,2,3}.

```
`define INSTR_MEM_SIZE  128 // Bytes
/*
 * Declaration of Instruction Memory for this project.
 * CAUTION: DONT MODIFY THE NAME.
 */
module IM(
    // Outputs
    output reg [31:0]   Instr,
    // Inputs
    input  [31:0]   InstrAddr
);

    /*
     * Declaration of instruction memory.
     * CAUTION: DONT MODIFY THE NAME AND SIZE.
     */
    reg [7:0]   InstrMem[0:`INSTR_MEM_SIZE - 1];

    always@(InstrAddr)
        begin
            Instr[31:0] = {InstrMem[InstrAddr], InstrMem[InstrAddr+1],InstrMem[InstrAddr+2],InstrMem[InstrAddr+3]};
        end

endmodule
```

c. Register File

RM這個module其實不難，只要判斷RegWrite是否為1，來決定是否可以將值寫入Reg，那其他部分就是到Register的address去抓值去輸出。我將RsData與RtData使用assign而非放在always裡面是因為我發現放在裡面會等到正緣觸發才把值送入ALU，那這樣的話就無法達到我們想要的效果了。

```
`define REG_MEM_SIZE    32  // Words

/*
 * Declaration of Register File for this project.
 * CAUTION: DONT MODIFY THE NAME.
 */
module RF(
    // Outputs
    output [31:0] RsData,
    output [31:0] RtData,
    // Inputs
    input RegWrite,
    input clk,
    input [4:0]  RsAddr,
    input [4:0]  RtAddr,
    input [4:0]  RdAddr,
    input [31:0] RdData
);

    /*
     * Declaration of inner register.
     * CAUTION: DONT MODIFY THE NAME AND SIZE.
     */
    reg [31:0]R[0:`REG_MEM_SIZE - 1];

    assign  RsData  = R[RsAddr];
    assign  RtData  = R[RtAddr];

    always@(posedge clk)
        begin
            if(RegWrite == 1)
                begin
                    R[RdAddr] = RdData;
                end
            else
                begin
                    R[RdAddr] = R[RdAddr];
                end
        end


endmodule
```

d. Data Memory

DM這個module其實也不難，只要判斷MemWrite跟MemRead是否為1，來決定是否可以將將值寫入Memory或是把Memory的值給讀出來。我將MemReadData使用assign而非放在always裡面，與RF的原因相同。我發現放在裡面會等到下個正緣觸發才把值送出，那這樣的話就無法達到我們想要的效果了。

```
`define DATA_MEM_SIZE   128 // Bytes

/*
 * Declaration of Data Memory for this project.
 * CAUTION: DONT MODIFY THE NAME.
 */
module DM(
    // Outputs
    output [31:0] MemReadData,
    // Inputs
    input [31:0] MemAddr,
    input [31:0] MemWriteData,
    input MemWrite,
    input MemRead,
    input clk
);

    /*
     * Declaration of data memory.
     * CAUTION: DONT MODIFY THE NAME AND SIZE.
     */
    reg [7:0]DataMem[0:`DATA_MEM_SIZE - 1];

    assign MemReadData = MemRead? {DataMem[MemAddr],DataMem[MemAddr+1],DataMem[MemAddr+2],DataMem[MemAddr+3]}:32'b0;
      You, a day ago • PART3 controller still working
    always@(posedge clk)
        begin
            if(MemWrite == 1)
                begin
                    {DataMem[MemAddr],DataMem[MemAddr+1],DataMem[MemAddr+2],DataMem[MemAddr+3]} = MemWriteData;
                end
            else;
        end

endmodule
```

e. Adder

Adder所做的事情非常簡單，因為在R-type的AdderOut僅僅需要能夠將AddrIn+4，因此我就只做了加4的動作，然後輸出。(我的加法是unsigned的加法，因為Address沒有負的。)

```
Adder.v
You, 3 days ago | 1 author (You)
module Adder
(
    input   [31:0]  AddrIn,
    output  [31:0]  AddrOut
);

    assign AddrOut = AddrIn + 32'd 4;

endmodule
```

f. ALU

ALU 主要是用來做Src1 and Src2的運算，由輸入的Funct來決定要做甚麼工作。

```
`define addu 6'b 001011
`define subu 6'b 001101
`define AND  6'b 010010
`define sll  6'b 100110

module ALU(
    input    [31:0]  Src1,
    input    [31:0]  Src2,
    input    [4:0]   Shamt,
    input    [5:0]   Funct,
    output  reg  [31:0]  result
);

always@(Funct or Shamt or Src1 or Src2)
    begin
        case (Funct)
            `addu : result = Src1 +  Src2;
            `subu : result = Src1 -  Src2;
            `AND  : result = Src1 &  Src2;
            `sll  : result = Src1 << Shamt;
            default: result = result; // if /
        endcase
    end


endmodule      You, 3 days ago • part1 is 87%
```

g. ALU_Control

ALU_Control主要是用來控制ALU工作與否，及將instr的指令轉為ALU懂得function code。比較特別的是在上一個作業的input_addu與ALU所要求的funct code不同，而這次作業的code是相同的，導致我在debug時花了一些時間才發現。

```
You, 3 days ago | 1 author (You)
`define addu 6'b 001011
`define subu 6'b 001101
`define AND  6'b 010010
`define sll  6'b 100110

`define input_addu 6'b 001011
`define input_subu 6'b 001101
`define input_and  6'b 010010
`define input_sll  6'b 100110
`define R_type 2'b10

module ALU_Control(
    input     [5:0] funct,
    input     [1:0] ALUOp,
    output  reg   [5:0] Funct
);
      You, 3 days ago • part1 is 87% finished, only l
always@(funct or ALUOp)
    begin
        case(ALUOp)
            `R_type:
                begin
                    case(funct)
                        `input_addu : Funct = `addu;
                        `input_subu : Funct = `subu;
                        `input_and  : Funct = `AND;
                        `input_sll  : Funct = `sll;
                        default: Funct = 0;
                    endcase
                end
            default:    Funct = 0;
        endcase
    end
endmodule
```

j. Control

Control這個module在整個CPU裡面是一個非常重要的角色。他掌管整顆CPU現在要做甚麼，不要做甚麼。雖然他極其重要，但是其實沒有甚麼太複雜的工作，只要依照Opcode的要求，來決定是否要送各種訊號。而因為R-type的所有指令都會使用到ALU，因此我們ALUOp只要是對的Opcode，我們一律送2‘b10.而對於I-type指令來說，只會有加法跟減法，因此除了subiu以外(2‘b00)，其他的ALUOP都為2‘b01。

```
You, 2 days ago | 1 author (You)
// I-type sub
`define I_type_sub 2'b00
// I-type add
`define I_type_add 2'b01
module Control(
    input   [5:0]   OpCode,
    output reg          RegWrite,
    output reg  [1:0]   ALUOp,
    output reg          RegDst,
    output reg          ALUSrc,
    output reg          MemWrite,
    output reg          MemRead,
    output reg          MemtoReg
);
```

```
always@(OpCode)
    begin
        case (OpCode)
            6'd 4:
                begin
                    RegWrite =  1'b 1; //
                    ALUOp = 2'b 10;
                    RegDst = 1; // R form
                    ALUSrc = 0;
                    MemWrite = 0;
                    MemRead = 0;
                    MemtoReg = 0;
                end
            // I format.
            6'd 12: // addiu
                begin
                    RegWrite = 1'b 1;
                    ALUOp = `I_type_add;
                    RegDst = 0; // I form
                    ALUSrc = 1;
                    MemWrite = 0;
                    MemRead = 0;
                    MemtoReg = 0;
                end
            6'd 13: // subiu
                begin
                    RegWrite = 1'b 1;
                    ALUOp = `I_type_sub;
                    RegDst = 0; // I form
                    ALUSrc = 1;
                    MemWrite=0;
                    MemRead = 0;
                    MemtoReg = 0;
                end
            6'd 16: // sw
                begin
                    RegWrite = 1'b 0;
                    ALUOp = `I_type_add;
                    RegDst = 0; // I form
                    ALUSrc = 1;
                    MemWrite = 1;
                    MemRead = 0;
                    MemtoReg = 0; // Sinc
                end
```

```
            end
            6'd 17: // lw
                begin
                    RegWrite = 1'b 1;
                    ALUOp = `I_type_add;
                    RegDst = 0; // I forma
                    ALUSrc = 1;
                    MemWrite = 0;
                    MemRead = 1;
                    MemtoReg= 1;
                end
            default:
                begin
                    MemWrite = 0;
                    RegWrite = 0;
                end
        endcase

    end

endmodule
```

k. MUX

MUX這幾個module其實非常簡單，一個是5bit,一個是32bit.只要判斷choose選的是多少，就決定輸出要送哪一個輸入出來。比較特別的是MUX STALL與MUX 3 to 1.是左上角Hazard Detection Unit底下的mux.

Mux5b.v

You, 3 days ago | 1 author (You)

```
module Mux5b(
    input [4:0]Src1,//0
    input [4:0]Src2,//1
    input choose,
    output [4:0]result
);
assign result = choose? Src2:Src1;

endmodule
```

Mux32b.v

You, 3 days ago | 1 author (You)

```
module Mux32b(
    input [31:0]Src1, // 0
    input [31:0]Src2, // 1
    input choose,
    output [31:0]result
);

assign result = choose? Src2:Src1;

endmodule
```

MuxStall.v

You, a day ago | 1 author (You)

```
module MuxStall(
input RegDst_in,
input MemRead_in,
input MemtoReg_in,
input[1:0] ALUOp_in,
input MemWrite_in,
input ALUSrc_in,
input RegWrite_in,
input Stall_choose,
output reg RegDst_out,
output reg MemRead_out,
output reg MemtoReg_out,
output reg[1:0] ALUOp_out,
output reg MemWrite_out,
output reg ALUSrc_out,
output reg RegWrite_out
);

always@(*) begin
    if (Stall_choose==0)
    begin
        RegDst_out   =RegDst_in;
        MemRead_out =MemRead_in;
        MemtoReg_out     =MemtoReg_in;
        ALUOp_out    =ALUOp_in;
        MemWrite_out     =MemWrite_in;
        ALUSrc_out   =ALUSrc_in;
        RegWrite_out     =RegWrite_in;
    end
    if (Stall_choose==1)
    begin
        RegDst_out   =0;
        MemRead_out =0;
        MemtoReg_out     =0;
        ALUOp_out    =2'd0;
        MemWrite_out     =0;
        ALUSrc_out   =0;
        RegWrite_out     =0;
    end
end
endmodule
```

You, 2 days ago | 1 author (You)

```
module MUX3to1(
    input [31:0] Src1,
    input [31:0] Src2,
    input [31:0] Src3,
    input [1:0] choose,
    output reg [31:0]result
);

always@(*)
begin
    case(choose)
        2'b00:result = Src1;
        2'b01:result = Src2;
        2'b10:result = Src3;
        default:;
    endcase
end

endmodule
```

j. IF/ID, ID/EX, EX/MEM, MEM/WB

IF_ID.v

```verilog
...
module IF_ID(
    input [31:0] Instr,
    input clk,
    input IF_ID_Write,
    output reg [31:0] InstrOut
);
    reg [31:0] Instr_reg;

always@(posedge clk or negedge clk)
    begin
        if(clk == 1 && IF_ID_Write == 1)
            Instr_reg = Instr;
        else // when negedge clk, output
            InstrOut = Instr_reg;
    end
endmodule
```

ID_EX.v

You, a minute ago | 1 author (You)

```verilog
module ID_EX(
    // Write Back.
    input RegWrite_in, // WB
    input Mem2Reg_in,  //WB

    output reg RegWrite_out, // WB
    output reg Mem2Reg_out,// WB
    // Memory
    input MemRead_in,
    input MemWrite_in,
    output reg MemWrite_out,
    output reg MemRead_out,
    // EX.
    input [1:0] ALUOp_in, // EX
    input RegDst_in,
    input ALU_Src_in,

    output reg [1:0] ALUOp_out, //EX
    output reg RegDst_out,
    output reg ALU_Src_out,
    // Others
    input clk,
    input [4:0] RdAddr_in,
    input [4:0] RtAddr_in,
    input [4:0] RsAddr_in,
    input [31:0] RsData_in,
    input [31:0] RtData_in,
    input [31:0] immediate_in,
    output reg [31:0] immediate_out,
    output reg [31:0] RsData_out,
    output reg [31:0] RtData_out,
    output reg [4:0] RdAddr_out,
    output reg [4:0] RtAddr_out,
    output reg [4:0] RsAddr_out
);
```

```verilog
    // Write Back.
    reg RegWrite_reg;
    reg Mem2Reg_reg;
    // Memory
    reg MemRead_reg;
    reg MemWrite_reg;
    // Execution.
    reg [1:0] ALUOp_reg; // EX
    reg RegDst_reg;
    reg ALU_Src_reg;
    // Others.
    reg [4:0] RdAddr_reg;
    reg [4:0] RtAddr_reg;
    reg [4:0] RsAddr_reg;
    reg [31:0] RsData_reg;
    reg [31:0] RtData_reg;
    reg [31:0] immediate_reg;
```

```verilog
always@(posedge clk or negedge clk)
    begin
        if(clk == 1) // put them in to the reg
            begin
                // Write Back.
                Mem2Reg_reg = Mem2Reg_in;
                RegWrite_reg = RegWrite_in;
                // Memory.
                MemRead_reg = MemRead_in;
                MemWrite_reg = MemWrite_in;
                // Execution
                ALUOp_reg = ALUOp_in;
                RegDst_reg = RegDst_in;
                ALU_Src_reg =ALU_Src_in;
                // Others.
                RdAddr_reg = RdAddr_in;
                RtAddr_reg = RtAddr_in;
                RsAddr_reg = RsAddr_in;

                RsData_reg = RsData_in;
                RtData_reg = RtData_in;
                immediate_reg = immediate_in;
            end
        else
            begin
                // Write Back.
                RegWrite_out = RegWrite_reg;
                Mem2Reg_out = Mem2Reg_reg;
                // Memory.
                MemRead_out = MemRead_reg;
                MemWrite_out = MemWrite_reg;
                // Execution
                ALUOp_out = ALUOp_reg;
                RegDst_out = RegDst_reg;
                ALU_Src_out =ALU_Src_reg;
                // Others.
                RdAddr_out = RdAddr_reg;
                RtAddr_out = RtAddr_reg;
                RsAddr_out = RsAddr_reg;

                RsData_out = RsData_reg;
                RtData_out = RtData_reg;
                immediate_out = immediate_reg;
            end
    end
```

You, seconds ago | 1 author (You)

```verilog
module EX_MEM(

    // Write Back.
    input RegWrite_in, // WB
    input Mem2Reg_in,  //WB

    output reg RegWrite_out, // WB
    output reg Mem2Reg_out,// WB

    // Memory
    input MemRead_in,
    input MemWrite_in,
    output reg MemWrite_out,
    output reg MemRead_out,

    // Others.

    input clk,
    input [31:0] ALU_result_in,
    input [4:0] RdAddr_in,
    input [31:0] RtData_in,
    output reg [31:0] RtData_out,
    output reg [4:0] RdAddr_out,
    output reg [31:0] ALU_result_out
)    You, seconds ago • Uncommitted

    // Temp register part.

    // Write Back.
    reg RegWrite_reg;
    reg Mem2Reg_reg;
    // Memory
    reg MemRead_reg;
    reg MemWrite_reg;
    // Others.
    reg [5:0] RdAddr_reg;
    reg [31:0] RtData_reg;
    reg [31:0] ALU_result_reg;
```

```verilog
always@(posedge clk or negedge clk)
    begin
        if(clk == 1) // put them in to the reg
            begin
                // Write Back.
                RegWrite_reg = RegWrite_in;
                Mem2Reg_reg = Mem2Reg_in;
                // Memory.
                RegWrite_reg = RegWrite_in;
                Mem2Reg_reg = Mem2Reg_in;
                MemRead_reg = MemRead_in;
                MemWrite_reg = MemWrite_in;
                // Others
                RdAddr_reg = RdAddr_in;
                ALU_result_reg = ALU_result_in;
                RtData_reg = RtData_in;

            end
        else
            begin
                // Write Back.
                RegWrite_out = RegWrite_reg;
                Mem2Reg_out = Mem2Reg_reg;
                // Memory
                RegWrite_out = RegWrite_reg;
                Mem2Reg_out = Mem2Reg_reg;
                MemRead_out = MemRead_reg;
                MemWrite_out = MemWrite_reg;
                // Others
                RdAddr_out = RdAddr_reg;
                ALU_result_out = ALU_result_reg;
                RtData_out = RtData_reg;

            end
    end


endmodule
```

```
MEM_WB.v
You, 2 days ago | 1 author (You)
module MEM_WB(        You, 2 days ago •

    // Write Back.
    input RegWrite_in, // WB
    input Mem2Reg_in,  //WB

    output reg RegWrite_out, // WB
    output reg Mem2Reg_out,// WB

    // Others.

    input clk,
    input [31:0] MemAddr_in,
    input [4:0] RdAddr_in,
    input [31:0] MemReadData_in,


    output reg [31:0] MemReadData_out,
    output reg [4:0] RdAddr_out,
    output reg [31:0] MemAddr_out
);

    // Temp register part.

    // Write Back.
    reg RegWrite_reg;
    reg Mem2Reg_reg;

    // Others.
    reg [5:0] RdAddr_reg;
    reg [31:0] MemAddr_reg;
    reg [31:0] MemReadData_reg;
```

```
always@(posedge clk or negedge clk)
    begin
        if(clk == 1) // put them in to the reg
            begin
                // Write Back.
                RegWrite_reg = RegWrite_in;
                Mem2Reg_reg = Mem2Reg_in;

                // Others
                RdAddr_reg = RdAddr_in;
                MemAddr_reg = MemAddr_in;
                MemReadData_reg = MemReadData_in;
            end
        else
            begin
                // Write Back.
                RegWrite_out = RegWrite_reg;
                Mem2Reg_out = Mem2Reg_reg;

                // Others
                RdAddr_out = RdAddr_reg;
                MemAddr_out = MemAddr_reg;
                MemReadData_out = MemReadData_reg;
            end
    end


endmodule
```

這幾個module都是Pipeline register.因為性質相同，且無任何特別的，因此在這個地方，就將他們放上來，所有的pipeline register都是正緣觸發，將值傳入register,當負緣的時候才更新輸出的值。確保不會影響到下一刻的output.

4. 作業總結與心得

這次的作業，從6/2星期三出的。因為疫情的關係，全台所有學校進行遠距教學，讓我原本排滿滿的生活，有了充裕的時間可以自行運用。因此這次的作業，我從出作業當天晚上開始思考如何做起，當天晚上把part1做完。而隔天早上把part2做完。下午開始處理Part3，相比part2與part3，真的是難度相差非常多，因為他不僅要處理pipeline, 又要可以偵測hazard，還要做forwarding.在那個晚上在兜完線以後，滿心期待地去跑模擬，結果出來有三個register，是錯誤的，於是就開始了整個晚上的debug時間。

一直忙到大概隔天的凌晨五點，我一直反覆追溯bug的來源，把所有波形放出來看，卻都解決不了問題。(這時候助教還沒有改線)，後來五點的時候我覺得實在是太累了，必須休息，但是我還是不死心的google了一下，結果躺在床上的時候居然找到答案了，這時候我就立刻跳起來打開電腦試試看新學到的方法，結果真的解決了1/3的bug. 但還是有兩個臭蟲仍然存活於我的code之中。不過我還是心滿意足地去睡覺了。到了隔天助教重新發了一張新的接線圖，我就開始依照這張圖重新施工。不過就算花了好幾小時重新施工，我仍然有相同的問題。後來我就有點進入放棄狀態，打算請求同學的幫助，並問他們是否接線圖有問題。結果同學回我說照著最新的圖是沒問題的。後來發現原來一開始我是照著上課投影片多工器的方法去接線，最後才發現原來課本的投影片多工器那邊出了問題。而更改過後，模擬結果還真的對了。

這次的題目主要是以PA2為基礎，增加pipeline register來減少clock數量。雖然因為part3的圖我沒有搞清楚就開始做所以多花了很多時間以外，其他部分都很順利的過五關斬六將了。我在做最後一個project才突然發現一些增進debug的技巧，可以從Model Sim那邊看到各條wire和reg的值，之前大部分的project我都會與同學討論，但這次我一人鑽研居然就成功了。這學期有修計算機組織真的很開心，希望這堂課的期末考試，也可以像這次一樣順利度過。