計算機組織

PA3

班級:四電機三乙

姓名:王翊峻

學號:B10707137

Part1

R_PipelineCPU

1. IM.v

沿用 PA2 模組，讀取 IM.dat 內的指令並儲存在暫存器裡。

2. RF.v

沿用 PA2 模組，讀取 RF.dat 內的資料並儲存在暫存器裡，並提供寫入及讀出的功能。

3. Control.v

沿用 PA2 模組，依照當前指令產生對應的控制信號。

4. ALU_Control.v

沿用 PA2 模組，依照當前指令使 ALU 進行對應的運算。

5. ALU.v

沿用 PA2 模組，進行邏輯或數學運算。

6. Adder.v

沿用 PA2 模組，將兩個輸入相加並輸出結果，此處用來將 PC 指向 PC+4。

7. IFID.v

```
module IFID(clk, Instr, Instr_out);
    input clk;
    input [31:0] Instr;
    output reg [31:0] Instr_out;
    always @(posedge clk) begin
        Instr_out = Instr;
    end
endmodule

```

第一層 Pipeline，用來儲存指令。

8. IDEX.v

```
module IDEX(clk, RsData, RtData, ALUOp, RdAddr, Shamt, funct, RegWrite, MemWrite, MemRead,
    Data_out, RdAddr_out, WB, M, EX);
    input clk;
    input [31:0] RsData, RtData;
    input [1:0] ALUOp;
    input [5:0] funct;
    input [4:0] RdAddr, Shamt;
    input RegWrite, MemWrite, MemRead;
    output reg [63:0] Data_out;
    output reg [4:0] RdAddr_out;
    output reg WB;
    output reg [1:0] M;
    output reg [12:0] EX;
    always@(posedge clk) begin
        Data_out = {RsData, RtData};
        RdAddr_out = RdAddr;
        WB = RegWrite;
        M = {MemWrite, MemRead};
        EX = {ALUOp, Shamt, funct};
    end
endmodule
```

第二層 Pipeline，用來儲存各個控制信號、RF 所讀出的資料等等。

9. EXMEM.v

```
module EXMEM(clk, WB, M, ALU_Result, RdAddr, WB_out, M_out, ALU_ResultOut, RdAddr_out);
    input clk;
    input WB;
    input[1:0] M;
    input[31:0] ALU_Result;
    input[4:0] RdAddr;
    output reg WB_out;
    output reg[1:0] M_out;
    output reg[31:0] ALU_ResultOut;
    output reg[4:0] RdAddr_out;
    always@(posedge clk)begin
        WB_out = WB;
        M_out = M;
        ALU_ResultOut = ALU_Result;
        RdAddr_out = RdAddr;
    end
endmodule
```

第三層 Pipeline，用來儲存後面所需的控制信號，以及 ALU 的運算結果等等。

10. MEMWB.v

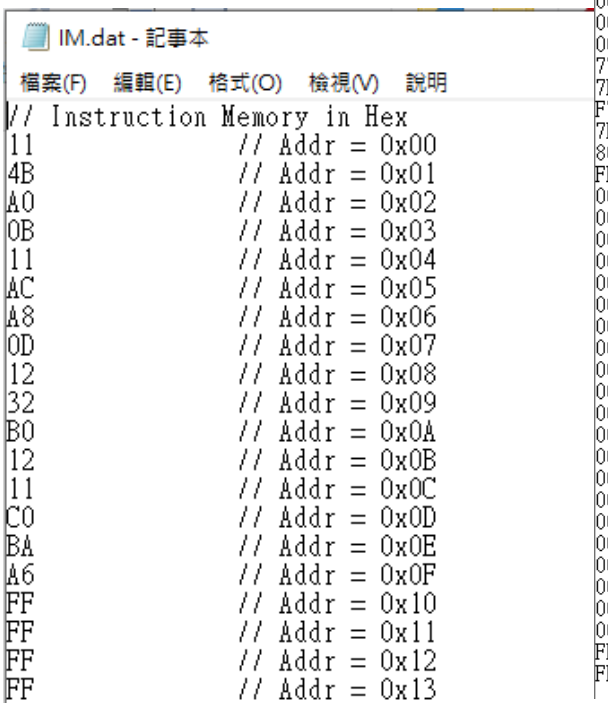
第四層 Pipeline，用來儲存後面所需的控制信號，以及 Memory 讀取結果等等。

11. R_PipelineCPU.v

```
module R_PipelineCPU(
    // Outputs
    output  wire    [31:0]  AddrOut,
    // Inputs
    input   wire    [31:0]  AddrIn,
    input   wire            clk
);
    wire[31:0] Instr, Instr_out, RsData, RtData;
    wire[1:0] ALUOp;
    wire RegWrite, MemWrite, MemRead;
    wire[63:0] Data_out;
    wire[4:0] RdAddr_out_EX, RdAddr_out_MEM, RdAddr_out_WB;
    wire WB_EX, WB_MEM, WB_WB;
    wire[1:0] M_EX, M_MEM;
    wire[12:0] EX;
    wire[31:0] ALU_Result;
    wire[5:0] Funct;
    wire[31:0] ALU_ResultOut_MEM, ALU_ResultOut_WB;
    /*
     * Declaration of Instruction Memory.
     * CAUTION: DONT MODIFY THE NAME.
     */
    Adder Add1(4, AddrIn, AddrOut);
    IM Instr_Memory(
        // Outputs
        Instr,
        // Inputs
        AddrIn
    );
    IFID Register(clk, Instr, Instr_out);
    /*
     * Declaration of Register File.
     * CAUTION: DONT MODIFY THE NAME.
     */
    RF Register_File(clk, WB_WB, Instr_out[25:21], Instr_out[20:16], RdAddr_out_WB, ALU_ResultOut_WB, RsData, RtData);

    Control C1(Instr_out[31:26], ALUOp, RegWrite, MemWrite, MemRead);

    IDEX Register2(clk, RsData, RtData, ALUOp, Instr_out[15:11], Instr_out[10:6], Instr_out[5:0], RegWrite, MemWrite, MemRead,
        Data_out, RdAddr_out_EX, WB_EX, M_EX, EX);

    ALU_Control C2(EX[5:0], EX[12:11], Funct);

    ALU A1(Data_out[63:32], Data_out[31:0], EX[10:6], Funct, ALU_Result);

    EXMEM Register3(clk, WB_EX, M_EX, ALU_Result, RdAddr_out_EX, WB_MEM, M_MEM, ALU_ResultOut_MEM, RdAddr_out_MEM);

    MEMWB Register4(clk, WB_MEM, ALU_ResultOut_MEM, RdAddr_out_MEM, WB_WB, ALU_ResultOut_WB, RdAddr_out_WB);

endmodule
```

將上述模組依照架構圖做連結。

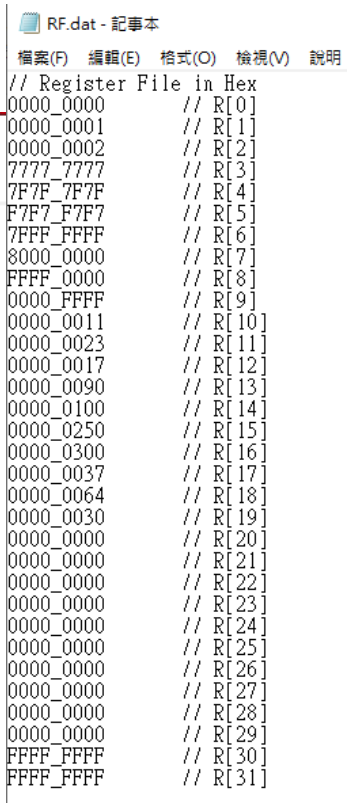
IM.dat & RF.dat



IM.dat - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
// Instruction Memory in Hex
11                  // Addr = 0x00
4B                  // Addr = 0x01
A0                  // Addr = 0x02
0B                  // Addr = 0x03
11                  // Addr = 0x04
AC                  // Addr = 0x05
A8                  // Addr = 0x06
0D                  // Addr = 0x07
12                  // Addr = 0x08
32                  // Addr = 0x09
B0                  // Addr = 0x0A
12                  // Addr = 0x0B
11                  // Addr = 0x0C
C0                  // Addr = 0x0D
BA                  // Addr = 0x0E
A6                  // Addr = 0x0F
FF                  // Addr = 0x10
FF                  // Addr = 0x11
FF                  // Addr = 0x12
FF                  // Addr = 0x13
```



RF.dat - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
// Register File in Hex
0000_0000        // R[0]
0000_0001        // R[1]
0000_0002        // R[2]
7777_7777        // R[3]
7F7F_7F7F        // R[4]
F7F7_F7F7        // R[5]
7FFF_FFFF        // R[6]
8000_0000        // R[7]
FFFF_0000        // R[8]
0000_FFFF        // R[9]
0000_0011        // R[10]
0000_0023        // R[11]
0000_0017        // R[12]
0000_0090        // R[13]
0000_0100        // R[14]
0000_0250        // R[15]
0000_0300        // R[16]
0000_0037        // R[17]
0000_0064        // R[18]
0000_0030        // R[19]
0000_0000        // R[20]
0000_0000        // R[21]
0000_0000        // R[22]
0000_0000        // R[23]
0000_0000        // R[24]
0000_0000        // R[25]
0000_0000        // R[26]
0000_0000        // R[27]
0000_0000        // R[28]
0000_0000        // R[29]
FFFF_FFFF        // R[30]
FFFF_FFFF        // R[31]
```

a. 第一次讀取指令為 114B_A00B

114B_A00B=0001_0001_0100_1011_1010_0000_0000_1011

因此 OpCode = 000100，RsAddr = 5’ b01010 = 10，

RtAddr = 5’ b01011 = 11，RdAddr = 5’ b10100 = 20，

funct = 6’ b001011，因此會執行加法將 R[10]+R[11]存回

R[20]。 目前 R[20] = 0000_0034。

b. 第二次讀取指令為 11AC_A80D

11AC_A80D=0001_0001_1010_1100_1010_1000_0000_1101

因此 OpCode = 000100，RsAddr = 13，RtAddr = 12，

RdAddr = 21，funct = 001101，因此執行減法，R[21] =

R[13] - R[12] = 0000_0079。

c. 第三次讀取指令為 1232_B012

1232_B012 = 0001_0010_0011_0010_1011_0000_0001_0010

Opcode = 000100，RsAddr = 17，RtAddr = 18，RdAddr = 22，funct = 010010，執行 and 運算，因此 R[22] = R[17]&R[18] = 0000_0037&0000_0064 = 0000_0024。

d. 第四次讀取指令為 11C0_BAA6

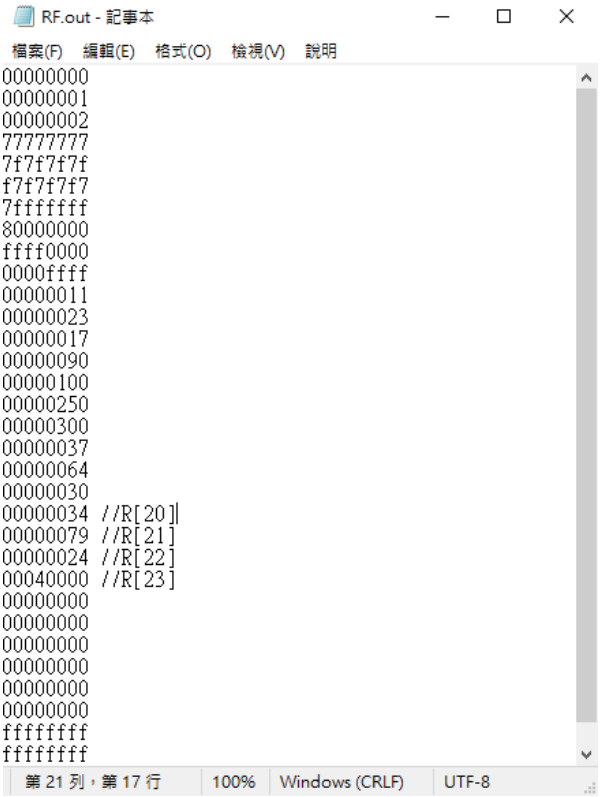
11C0_BAA6=0001_0001_1100_0000_1011_1010_1010_0110

Opcode = 000100，RsAddr = 14，RtAddr = 0，RdAddr = 23

funct= 100110，執行 shift_left_logic，shamt = 01010 = 10，

因此 R[23] = R[14] << 10 = 0004_0000

輸出結果



Part2

I_PipelineCPU

沿用 R_PipelineCPU 的各個模組，並加以擴充，但功能皆相同，並新增多個模組，以下只講解新增模組的部分。

1. DM.v

沿用 PA2 模組，讀取 DM.dat 內資料並儲存在暫存器內，並提供寫入及讀出的功能。

2. Mux32bits.v

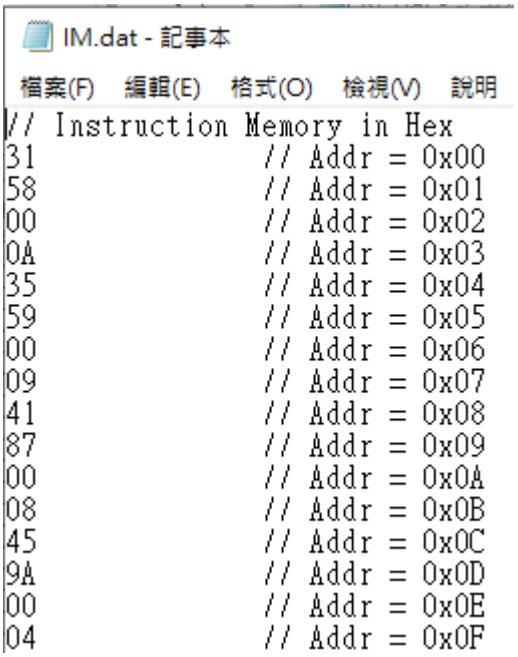
沿用 PA2 模組，32 位元的多工器，由 Control 模組產生的控制信號來決定輸出。

3. I_PipelineCPU.v

```
module I_PipelineCPU(
    // Outputs
    output  wire    [31:0]  AddrOut,
    // Inputs
    input   wire    [31:0]  AddrIn,
    input   wire            clk
);
    wire[31:0] Instr, Instr_out, RsData, RtData;
    wire[1:0] ALUOp;
    wire RegWrite, MemWrite, MemRead, RegDst, ALUSrc, MemtoReg;
    wire[63:0] Data_out;
    wire[4:0] RdAddr_out_EX, RtAddr_out_EX, RdAddr_out_MEM, RdAddr_out_WB;
    wire[1:0] WB_EX, WB_MEM, WB_WB;
    wire[1:0] M_EX, M_MEM;
    wire[14:0] EX;
    wire[31:0] ALU_Result;
    wire[5:0] Funct;
    wire[31:0] ALU_ResultOut_MEM, ALU_ResultOut_WB;
    wire[31:0] Immediate, Immediate_out, MemWriteData, MemReadData, MemReadData_out, mux1_out, mux2_out, mux3_out, mux4_out;
    assign Immediate = {{16{Instr_out[15]}}, Instr_out[15:0]};
    /*
     * Declaration of Instruction Memory.
     * CAUTION: DONT MODIFY THE NAME.
     */
    Adder Add1(4, AddrIn, AddrOut);
    IM Instr_Memory(
        // Outputs
        Instr,
        // Inputs
        AddrIn
    );
    IFID Register(clk, Instr, Instr_out);
    /*
     * Declaration of Register File.
     * CAUTION: DONT MODIFY THE NAME.
     */
    RF Register_File(clk, WB_WB[1], Instr_out[25:21], Instr_out[20:16], RdAddr_out_WB, mux4_out, RsData, RtData);

    Control C1(Instr_out[31:26], ALUOp, RegWrite, MemWrite, MemRead, RegDst, ALUSrc, MemtoReg);

    Mux_32bits mux1(0, {ALUOp, RegWrite, MemWrite, MemRead, RegDst, ALUSrc, MemtoReg}, 1, mux1_out);

    IDEX Register2(clk, RsData, RtData, Immediate, mux1_out[7:6], Instr_out[20:16], Instr_out[15:11], Instr_out[10:6],
        Instr_out[5:0], mux1_out[5], mux1_out[4], mux1_out[3], mux1_out[2], mux1_out[1], mux1_out[0],
        Data_out, Immediate_out, RtAddr_out_EX, RdAddr_out_EX, WB_EX, M_EX, EX);

    ALU_Control C2(EX[5:0], EX[12:11], Funct);

    Mux_32bits mux2(Data_out[31:0], Immediate_out, EX[13], mux2_out);

    ALU A1(Data_out[63:32], mux2_out, EX[10:6], Funct, ALU_Result);

    Mux_32bits mux3(RtAddr_out_EX, RdAddr_out_EX, EX[14], mux3_out);

    EXMEM Register3(clk, WB_EX, M_EX, ALU_Result, Data_out[31:0], mux3_out[4:0], WB_MEM, M_MEM, ALU_ResultOut_MEM, MemWriteData, RdAddr_out_MEM);

    DM Data_Memory(MemReadData, MemWriteData, ALU_ResultOut_MEM[6:0], M_MEM[1], M_MEM[0], clk);

    MEMWB Register4(clk, WB_MEM, ALU_ResultOut_MEM, MemReadData, RdAddr_out_MEM, WB_WB, ALU_ResultOut_WB, MemReadData_out, RdAddr_out_WB);

    Mux_32bits mux4(ALU_ResultOut_WB, MemReadData_out, WB_WB[0], mux4_out);
endmodule
```

將上述模組依照架構圖做連結。

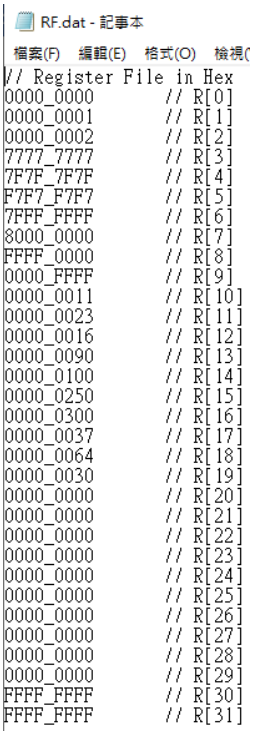
IM.dat & RF.dat



IM.dat - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
// Instruction Memory in Hex
31                  // Addr = 0x00
58                  // Addr = 0x01
00                  // Addr = 0x02
0A                  // Addr = 0x03
35                  // Addr = 0x04
59                  // Addr = 0x05
00                  // Addr = 0x06
09                  // Addr = 0x07
41                  // Addr = 0x08
87                  // Addr = 0x09
00                  // Addr = 0x0A
08                  // Addr = 0x0B
45                  // Addr = 0x0C
9A                  // Addr = 0x0D
00                  // Addr = 0x0E
04                  // Addr = 0x0F
```



RF.dat - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(

```
// Register File in Hex
0000_0000        // R[0]
0000_0001        // R[1]
0000_0002        // R[2]
7777_7777        // R[3]
7F7F_7F7F        // R[4]
F7F7_F7F7        // R[5]
7FFF_FFFF        // R[6]
8000_0000        // R[7]
FFFF_0000        // R[8]
0000_FFFF        // R[9]
0000_0011        // R[10]
0000_0023        // R[11]
0000_0016        // R[12]
0000_0090        // R[13]
0000_0100        // R[14]
0000_0250        // R[15]
0000_0300        // R[16]
0000_0037        // R[17]
0000_0064        // R[18]
0000_0030        // R[19]
0000_0000        // R[20]
0000_0000        // R[21]
0000_0000        // R[22]
0000_0000        // R[23]
0000_0000        // R[24]
0000_0000        // R[25]
0000_0000        // R[26]
0000_0000        // R[27]
0000_0000        // R[28]
0000_0000        // R[29]
FFFF_FFFF        // R[30]
FFFF_FFFF        // R[31]
```

a. 第一次讀取指令為 3158_000A

Opcode = 001100，因此做 addiu，且 immediate = 000A

RsAddr = 10，RtAddr = 24，因此 R[24] = R[10] + 000A = 0000_001B

b. 第二次讀取指令為 3559_0009

Opcode = 001101，因此做 subiu，且 immediate = 0009

RsAddr = 10，RtAddr = 25，因此 R[25] = R[10] - 0009 = 0000_0008

c. 第三次讀取指令為 4187_0008

Opcode = 010000，因此做 sw 指令，immediate = 0008

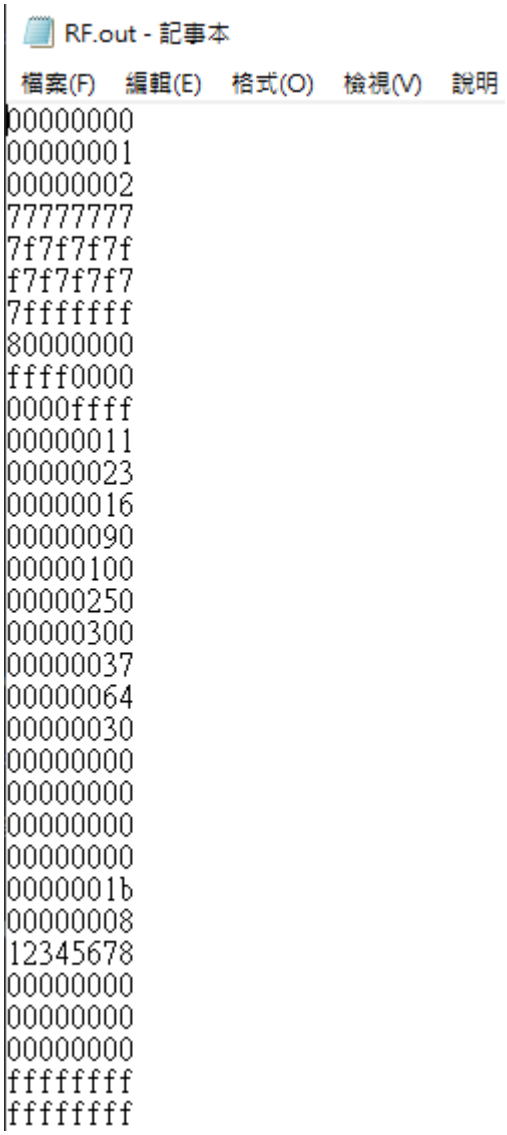
RsAddr=12，RtAddr=7，因此 MemData[30] = 8000_0000

d. 第四次讀取指令為 459A_0004

Opcode = 010001，因此做 LoadWord，immediate = 0004

RsAddr = 12，RtAddr = 26，因此 R[26] = Mem[22 + 4] = 1234_5678

輸出結果



Part3

FinalCPU

沿用 I_PipelineCPU 的各個模組，並加以擴充，但功能皆相同，並新增多個模組，以下只講解新增模組的部分。

1. HazardUnit.v

```
module HazardUnit(MemRead_EX, Rd_Addr_EX, RsAddr_ID, Rt_Addr_ID, PCWrite, IFID_Write);
    input MemRead_EX;
    input[4:0] Rd_Addr_EX, RsAddr_ID, Rt_Addr_ID;
    output reg PCWrite = 1;
    output IFID_Write;
    assign IFID_Write = 1;
    always@(MemRead_EX, Rd_Addr_EX, RsAddr_ID, Rt_Addr_ID) begin
        if(MemRead_EX && ((Rd_Addr_EX == RsAddr_ID) || (Rd_Addr_EX == Rt_Addr_ID))) PCWrite = 0;
        else PCWrite = 1;
    end
endmodule
```

根據當前 ID、EX 階段的特定信號判斷是否要進行 stall，如果要進行 stall 就將 PCWrite 設為 0。

2. ForwardUnit.v

```
module ForwardUnit(RdAddr_MEM, RdAddr_WB, RsAddr_EX, RtAddr_EX, RegWrite_MEM, RegWrite_WB, ForwardA, ForwardB);
    input[4:0] RdAddr_MEM, RdAddr_WB, RsAddr_EX, RtAddr_EX;
    input RegWrite_MEM, RegWrite_WB;
    output reg[1:0] ForwardA = 2'b00, ForwardB = 2'b00;
    always@(RegWrite_MEM, RdAddr_MEM, RegWrite_WB, RdAddr_WB, RsAddr_EX, RtAddr_EX) begin
        if(RegWrite_MEM && (RdAddr_MEM != 0) && (RdAddr_MEM == RsAddr_EX)) begin ForwardA = 2'b10; end
        else if (RegWrite_WB && (RdAddr_WB != 0) && (RdAddr_WB == RsAddr_EX)) begin ForwardA = 2'b01; end
        else begin ForwardA = 2'b00; end

        if(RegWrite_MEM && (RdAddr_MEM != 0) && (RdAddr_MEM == RtAddr_EX)) begin ForwardB = 2'b10; end
        else if (RegWrite_WB && (RdAddr_WB != 0) && (RdAddr_WB == RtAddr_EX)) begin ForwardB = 2'b01; end
        else begin ForwardB = 2'b00; end
    end
endmodule
```

根據當前 EX、MEM、WB 階段的特定信號判斷是否要進行 Forwarding，ForwardA 控制 RsData 的 forwarding，ForwardB 控制 RtData 的 forwarding。

3. Mux_3to1.v

```
1  module Mux_3to1(Input1, Input2, Input3, Output1, select);
2      input[31:0] Input1, Input2, Input3;
3      input[1:0] select;
4      output[31:0] Output1;
5      assign Output1 = (select==2'b00) ? Input1
6                      :(select==2'b01) ? Input2
7                      :(select==2'b10) ? Input3 : Input1;
8  endmodule
```

3 選 1 的多工器，藉由接收到的 Forwarding 信號決定要從 MEM 階段做 Forwarding 還是從 WB 階段做 Forwarding，還是不做 Forwarding。

4. FinalCPU.v

```
module FinalCPU(
    // Outputs
    output  wire            PCWrite,
    output  wire    [31:0]  AddrOut,
    // Inputs
    input   wire    [31:0]  AddrIn,
    input   wire            clk
);
    wire[31:0] Instr, Instr_out, RsData, RtData;
    wire[1:0] ALUOp;
    wire RegWrite, MemWrite, MemRead, RegDst, ALUSrc, MemtoReg, IFID_Write;
    wire[63:0] Data_out;
    wire[4:0] RdAddr_out_EX, RtAddr_out_EX, RsAddr_out_EX, RdAddr_out_MEM, RdAddr_out_WB;
    wire[1:0] WB_EX, WB_MEM, WB_WB;
    wire[1:0] M_EX, M_MEM;
    wire[1:0] ForwardA, ForwardB;
    wire[14:0] EX;
    wire[31:0] ALU_Result;
    wire[5:0] Funct;
    wire[31:0] ALU_ResultOut_MEM, ALU_ResultOut_WB;
    wire[31:0] Immediate, Immediate_out, MemWriteData, MemReadData, MemReadData_out, mux1_out, mux2_out, mux3_out, mux4_out, mux5_out, mux6_out;
    assign Immediate = {{16{Instr_out[15]}}, Instr_out[15:0]};
    /*
     * Declaration of Instruction Memory.
     * CAUTION: DONT MODIFY THE NAME.
     */
    Adder Add1(4, AddrIn, AddrOut);
    IM Instr_Memory(
        // Outputs
        Instr,
        // Inputs
        AddrIn[6:0]
    );
    IFID Register(clk, IFID_Write, Instr, Instr_out);
    /*
     * Declaration of Register File.
     * CAUTION: DONT MODIFY THE NAME.
     */
    RF Register_File(clk, WB_WB[1], Instr_out[25:21], Instr_out[20:16], RdAddr_out_WB, mux4_out, RsData, RtData);

    Control C1(Instr_out[31:26], ALUOp, RegWrite, MemWrite, MemRead, RegDst, ALUSrc, MemtoReg);

    HazardUnit HU(M_EX[0], RtAddr_out_EX, Instr_out[25:21], Instr_out[20:16], PCWrite, IFID_Write);

    Mux_32bits mux1(0, {ALUOp, RegWrite, MemWrite, MemRead, RegDst, ALUSrc, MemtoReg}, PCWrite, mux1_out);

    IDEX Register2(clk, RsData, RtData, Immediate, mux1_out[7:6], Instr_out[20:16], Instr_out[15:11], Instr_out[25:21],
        Instr_out[10:6], Instr_out[5:0], mux1_out[5], mux1_out[4], mux1_out[3], mux1_out[2], mux1_out[1], mux1_out[0],
        Data_out, Immediate_out, RtAddr_out_EX, RdAddr_out_EX, RsAddr_out_EX, WB_EX, M_EX, EX);

    ALU_Control C2(EX[5:0], EX[12:11], Funct);

    ForwardUnit FU(RdAddr_out_MEM, RdAddr_out_WB, RsAddr_out_EX, RtAddr_out_EX, WB_MEM[1], WB_WB[1], ForwardA, ForwardB);

    Mux_3to1 mux_5(Data_out[63:32], mux4_out, ALU_ResultOut_MEM, mux5_out, ForwardA);

    Mux_3to1 mux_6(Data_out[31:0], mux4_out, ALU_ResultOut_MEM, mux6_out, ForwardB);

    Mux_32bits mux2(mux6_out, Immediate_out, EX[13], mux2_out);

    ALU A1(mux5_out, mux2_out, EX[10:6], Funct, ALU_Result);

    Mux_32bits mux3(RtAddr_out_EX, RdAddr_out_EX, EX[14], mux3_out);

    EXMEM Register3(clk, WB_EX, M_EX, ALU_Result, mux6_out, mux3_out[4:0], WB_MEM, M_MEM, ALU_ResultOut_MEM, MemWriteData, RdAddr_out_MEM);

    DM Data_Memory(MemReadData, MemWriteData, ALU_ResultOut_MEM[6:0], M_MEM[1], M_MEM[0], clk);

    MEMWB Register4(clk, WB_MEM, ALU_ResultOut_MEM, MemReadData, RdAddr_out_MEM, WB_WB, ALU_ResultOut_WB, MemReadData_out, RdAddr_out_WB);

    Mux_32bits mux4(ALU_ResultOut_WB, MemReadData_out, WB_WB[0], mux4_out);
endmodule
```

將上述模組依照架構圖做連結。

IM.dat & RF.dat

IM.dat - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
// Instruction Memory in Hex
10                  // Addr = 0x00
43                  // Addr = 0x01
D0                  // Addr = 0x02
0B                  // Addr = 0x03
13                  // Addr = 0x04
41                  // Addr = 0x05
D8                  // Addr = 0x06
0D                  // Addr = 0x07
45                  // Addr = 0x08
9C                  // Addr = 0x09
00                  // Addr = 0x0A
04                  // Addr = 0x0B
41                  // Addr = 0x0C
9C                  // Addr = 0x0D
00                  // Addr = 0x0E
08                  // Addr = 0x0F
45                  // Addr = 0x10
9D                  // Addr = 0x11
00                  // Addr = 0x12
08                  // Addr = 0x13
13                  // Addr = 0x14
AA                  // Addr = 0x15
F0                  // Addr = 0x16
0B                  // Addr = 0x17
```

RF.dat - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V)

```
// Register File in Hex
0000_0000        // R[0]
0000_0001        // R[1]
0000_0002        // R[2]
7777_7777        // R[3]
7F7F_7F7F        // R[4]
F7F7_F7F7        // R[5]
7FFF_FFFF        // R[6]
8000_0000        // R[7]
FFFF_0000        // R[8]
0000_FFFF        // R[9]
0000_0011        // R[10]
0000_0023        // R[11]
0000_0016        // R[12]
0000_0090        // R[13]
0000_0100        // R[14]
0000_0250        // R[15]
0000_0300        // R[16]
0000_0037        // R[17]
0000_0064        // R[18]
0000_0030        // R[19]
0000_0000        // R[20]
0000_0000        // R[21]
0000_0000        // R[22]
0000_0000        // R[23]
0000_0000        // R[24]
0000_0000        // R[25]
0000_0000        // R[26]
0000_0000        // R[27]
0000_0000        // R[28]
0000_0000        // R[29]
FFFF_FFFF        // R[30]
FFFF_FFFF        // R[31]
```

a. 第一次讀取指令為 1043_D00B

執行 add $R26, $R02, $R03

R[26] = R[2] + R[3] = 7777_7779

b. 第二次讀取指令為 1341_D80D

執行 sub $R27, $R26, $R01

因上一個指令結果尚未存入 R[26]，因此需要做 Forward

R[27] = R[26] - R[1] = 7777_7778

c. 第三次讀取指令為 459C_0004

執行 lw $R28, 4($R12)

R[28] = MEM[26] = 1234_5678

d. 第四次讀取指令為 419C_0008

執行 sw　$R28, 8($R12)

因上一個指令結果尚未從 Memory 取出，因此無法做 Forward，需要先做一次 Stall，才能做 Forward。

MEM[30] = R[28] = 1234_5678

e. 第五次讀取指令為 459D_0008

執行 lw　$R29, 8($R12)

R[29]=MEM[30]，不須做 Forward 或 Stall

f. 第六次讀取指令為 13AA_F00B

執行 add　$R30, $R29, $R10

因上一個指令結果尚未從 Memory 取出，因此無法做 Forward，需要先做一次 Stall，才能做 Forward。

R[30] = R[29] + R[10] = 1234_5689

輸出結果

RF.out - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
00000000
00000001
00000002
77777777
7f7f7f7f
f7f7f7f7
7fffffff
80000000
ffff0000
0000ffff
00000011
00000023
00000016
00000090
00000100
00000250
00000300
00000037
00000064
00000030
00000000
00000000
00000000
00000000
00000000
00000000
77777779
77777778
12345678
12345678
12345689
ffffffff
```

DM.out - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
12
34
56
78
12
34
56
78
ff
```

作業總結與心得

這次的作業實作了上課所學到的 Pipeline 架構以及 Hazard Detect 跟 Forwarding，實現電路的過程中我也更加熟悉 Hazard 的判斷條件，以及如何做 Stall，把後續都指令都往後延一個 Clock，也更加清楚 Forwarding 可能會產生哪些衝突，以及要如何避免它發生，這次的作業讓我深刻了解這個單元所要教導的東西，讓我能夠將上課所學到的理論實現成電路。