計算機組織　作業報告

作業三(PA3)

B10807005 朱育辰

一、第三部分

因為部分模組與 PA2 相同，因此報告僅列出這次新增或有修改過的部分。

a. 各模組功能描述

i. Fetch & Decode Register (IF_ID)

1. 程式碼

```
module IF_ID (
    output reg [31:0] Instr_out,
    input [31:0] Instr,
    input IF_ID_Write,
    input clk
);
    reg [31:0] Instr_reg;
    always @(posedge clk or negedge clk) begin
        if (clk == 1 && IF_ID_Write == 1) begin
            Instr_reg = Instr;
        end
        else
            Instr_out = Instr_reg;
    end
endmodule
```

2. 功能

暫存從記憶體讀出的指令。

ii. Hazard Detection

1. 程式碼

```
module Hazard_Detection (
    output reg PCWrite,
    output reg IF_ID_Write,
    output reg Stall,
    input ID_EX_MemRead,
    input [4:0] ID_EX_RtAddr,
    input [4:0] IF_ID_RsAddr,
    input [4:0] IF_ID_RtAddr
);
    initial begin
        Stall = 0;
        IF_ID_Write = 1;
        PCWrite = 1;
    end

    always @(*) begin
        if(ID_EX_MemRead && ((ID_EX_RtAddr == IF_ID_RsAddr)||(ID_EX_RtAddr == IF_ID_RtAddr)))begin
            PCWrite = 0;
            Stall = 1;
            IF_ID_Write = 0;
        end
        else begin
            PCWrite = 1;
            Stall = 0;
            IF_ID_Write = 1;
        end
    end
endmodule
```

2. 功能

利用記憶體讀取旗標以及前後兩條指令使用的暫存器位址判斷有無發生衝突，若有則暫停一個週期。

iii. Control Unit (Control)

1. 程式碼

```
module Control (
    input [5:0] Opcode,
    output reg [1:0] ALUOp,
    output reg RegDst,
    output reg RegWrite,
    output reg ALUSrc,
    output reg MemWrite,
    output reg MemRead,
    output reg MemToReg
);

always @(Opcode) begin

    case (Opcode)
        6'd0:
            begin
                RegWrite = 1;
                ALUOp = 2'b10;
                RegDst = 1;
                ALUSrc = 0;
                MemWrite = 0;
                MemRead = 0;
                MemToReg = 0;
            end
        6'b001100:
            begin
                RegWrite = 1;
                ALUOp = 2'b00;
                RegDst = 0;
                ALUSrc = 1;
                MemWrite = 0;
                MemRead = 0;
                MemToReg = 0;
            end
        6'b001101:
            begin
                RegWrite = 1;
                ALUOp = 2'b01;
                RegDst = 0;
                ALUSrc = 1;
                MemWrite = 0;
                MemRead = 0;
                MemToReg = 0;
            end
        6'b010000:
            begin
                RegWrite = 0;
                ALUOp = 2'b00;
                RegDst = 0;
                ALUSrc = 1;
                MemWrite = 1;
                MemRead = 0;
                MemToReg = 0;
            end
        6'b010001:
            begin
                RegWrite = 1;
                ALUOp = 2'b00;
                RegDst = 0;
                ALUSrc = 1;
                MemWrite = 0;
                MemRead = 1;
                MemToReg = 1;
            end
        default:
            begin
                MemWrite = 0;
                RegWrite = 0;
            end
    endcase
end

endmodule
```

2. 功能

與 PA2 相同，但因為這次沒有分支或跳躍指令，故移除相對應旗標。

iv. 暫停多工器(MUX_Stall)

1. 程式碼

```
module MUX_Stall (
    output reg [1:0] ALUOp_result,
    output reg RegDst_result,
    output reg RegWrite_result,
    output reg ALUSrc_result,
    output reg MemWrite_result,
    output reg MemRead_result,
    output reg MemToReg_result,
    input Stall,
    input [1:0] ALUOp,
    input RegDst,
    input RegWrite,
    input ALUSrc,
    input MemWrite,
    input MemRead,
    input MemToReg
);
    assign ALUOp_result = Stall?2'b00:ALUOp;
    assign RegDst_result = Stall?0:RegDst;
    assign RegWrite_result = Stall?0:RegWrite;
    assign ALUSrc_result = Stall?0:ALUSrc;
    assign MemWrite_result = Stall?0:MemWrite;
    assign MemRead_result = Stall?0:MemRead;
    assign MemToReg_result = Stall?0:MemToReg;
endmodule
```

2. 功能

若接收到暫停旗標，則將控制單元所有的輸出旗標設為 0。

v. Decode & Execute Register (ID_EX)

1. 程式碼

```
module ID_EX (
    output reg [1:0] ALUOp_out,
    output reg RegDst_out,
    output reg RegWrite_out,
    output reg ALUSrc_out,
    output reg MemWrite_out,
    output reg MemRead_out,
    output reg MemToReg_out,
    input [1:0] ALUOp,
    input RegDst,
    input RegWrite,
    input ALUSrc,
    input MemWrite,
    input MemRead,
    input MemToReg,

    output reg [31:0] RsData_out,
    output reg [31:0] RtData_out,
    output reg [31:0] immediate_out,
    output reg [4:0] RtAddr_out,
    output reg [4:0] RdAddr_out,
    output reg [4:0] RsAddr_out,
    input [31:0] RsData,
    input [31:0] RtData,
    input [31:0] immediate,
    input [4:0] RtAddr,
    input [4:0] RdAddr,
    input [4:0] RsAddr,

    input clk
);
    reg [1:0] ALUOp_reg;
    reg RegDst_reg;
    reg RegWrite_reg;
    reg ALUSrc_reg;
    reg MemWrite_reg;
    reg MemRead_reg;
    reg MemToReg_reg;
    reg [31:0] RsData_reg;
    reg [31:0] RtData_reg;
    reg [31:0] immediate_reg;
    reg [4:0] RtAddr_reg;
    reg [4:0] RdAddr_reg;
    reg [4:0] RsAddr_reg;

    always @(posedge clk or negedge clk) begin
        if (clk == 1) begin
            ALUOp_reg = ALUOp;
            RegDst_reg = RegDst;
            RegWrite_reg = RegWrite;
            ALUSrc_reg = ALUSrc;
            MemWrite_reg = MemWrite;
            MemRead_reg = MemRead;
            MemToReg_reg = MemToReg;
            RsData_reg = RsData;
            RtData_reg = RtData;
            immediate_reg = immediate;
            RtAddr_reg = RtAddr;
            RdAddr_reg = RdAddr;
            RsAddr_reg = RsAddr;
        end
        else begin
            ALUOp_out = ALUOp_reg;
            RegDst_out = RegDst_reg;
            RegWrite_out = RegWrite_reg;
            ALUSrc_out = ALUSrc_reg;
            MemWrite_out = MemWrite_reg;
            MemRead_out = MemRead_reg;
            MemToReg_out = MemToReg_reg;
            RsData_out = RsData_reg;
            RtData_out = RtData_reg;
            immediate_out = immediate_reg;
            RtAddr_out = RtAddr_reg;
            RdAddr_out = RdAddr_reg;
            RsAddr_out = RsAddr_reg;
        end
    end
endmodule
```

2. 功能

暫存解碼後的指令

vi. 三對一多工器（MUX_3to1）

1. 程式碼

```
module MUX_3to1 (
    output reg [31:0] result,
    input [31:0] Src0,
    input [31:0] Src1,
    input [31:0] Src2,
    input [1:0] sel
);
    always @(*) begin
        case (sel)
            2'b00: result = Src0;
            2'b01: result = Src1;
            2'b10: result = Src2;
        endcase
    end
endmodule
```

2. 功能

利用選擇輸入，將相對應來源導至多工器輸出。

vii. Forwarding Unit（Frowarding）

1. 程式碼

```
module Forwarding (
    output reg [1:0]ForwardA,
    output reg [1:0]ForwardB,
    input EX_MEM_RegWrite,
    input [4:0] EX_MEM_RdAddr,
    input [4:0] ID_EX_RsAddr,
    input [4:0] ID_EX_RtAddr,
    input MEM_WB_Regwrite,
    input [4:0] MEM_WB_RdAddr

);
    always @(*) begin
        ForwardA = 2'b00;
        ForwardB = 2'b00;
        if (EX_MEM_RegWrite && (EX_MEM_RdAddr != 0) && (EX_MEM_RdAddr == ID_EX_RsAddr))begin
            ForwardA = 2'b10;
        end

        if (EX_MEM_RegWrite && (EX_MEM_RdAddr != 0) && (EX_MEM_RdAddr == ID_EX_RtAddr)) begin
            ForwardB = 2'b10;
        end

        if (MEM_WB_Regwrite && (MEM_WB_RdAddr != 0) && (MEM_WB_RdAddr == ID_EX_RsAddr)) begin
            ForwardA = 2'b01;
        end

        if (MEM_WB_Regwrite && (MEM_WB_RdAddr != 0) && (MEM_WB_RdAddr == ID_EX_RtAddr)) begin
            ForwardB = 2'b01;
        end
    end
endmodule
```

2. 功能

利用上一條指令的暫存器讀寫旗標、這次與上次的暫存器位址、以及上上次的旗標與暫存器位址，判斷是否衝突，若有則直接將計算結果取代暫存器資料。

viii. Execute & Memory Register (EX_MEM)

1. 程式碼

```
module EX_MEM (
    output reg RegWrite_out,
    output reg MemWrite_out,
    output reg MemRead_out,
    output reg MemToReg_out,
    input RegWrite,
    input MemWrite,
    input MemRead,
    input MemToReg,
    output reg [31:0] ALUresult_out,
    output reg [31:0] RtData_out,
    output reg [4:0] RdAddr_out,
    input [31:0] ALUresult,
    input [31:0] RtData,
    input [4:0] RdAddr,
    input clk
);
    reg RegWrite_reg;
    reg MemWrite_reg;
    reg MemRead_reg;
    reg MemToReg_reg;
    reg [31:0] ALUresult_reg;
    reg [31:0] RtData_reg;
    reg [4:0] RdAddr_reg;

    always @(posedge clk or negedge clk) begin
        if (clk == 1) begin
            RegWrite_reg = RegWrite;
            MemWrite_reg = MemWrite;
            MemRead_reg = MemRead;
            MemToReg_reg = MemToReg;
            ALUresult_reg = ALUresult;
            RtData_reg = RtData;
            RdAddr_reg = RdAddr;
        end
        else begin
            RegWrite_out = RegWrite_reg;
            MemWrite_out = MemWrite_reg;
            MemRead_out = MemRead_reg;
            MemToReg_out = MemToReg_reg;
            ALUresult_out = ALUresult_reg;
            RtData_out = RtData_reg;
            RdAddr_out = RdAddr_reg;
        end
    end
endmodule
```

2. 功能

暫存指令執行後的結果。

ix. Memory & Write-back Register (MEM_WB)

1. 程式碼

```
module MEM_WB (
    output reg RegWrite_out,
    output reg MemToReg_out,
    input RegWrite,
    input MemToReg,
    output reg [31:0] MemAddr_out,
    output reg [31:0] MemReadData_out,
    output reg [4:0] RdAddr_out,
    input [31:0] MemAddr,
    input reg [31:0] MemReadData,
    input reg [4:0] RdAddr,
    input clk
);
    reg RegWrite_reg;
    reg MemToReg_reg;
    reg [31:0] MemAddr_reg;
    reg [31:0] MemReadData_reg;
    reg [4:0] RdAddr_reg;

    always @(posedge clk or negedge clk) begin
        if (clk == 1) begin
            RegWrite_reg = RegWrite;
            MemToReg_reg = MemToReg;
            MemAddr_reg = MemAddr;
            MemReadData_reg = MemReadData;
            RdAddr_reg = RdAddr;
        end
        else begin
            RegWrite_out = RegWrite_reg;
            MemToReg_out = MemToReg_reg;
            MemAddr_out = MemAddr_reg;
            MemReadData_out = MemReadData_reg;
            RdAddr_out = RdAddr_reg;
        end
    end
endmodule
```

2. 功能

暫存記憶體讀出的資料，準備寫回暫存器或供下條指令使用。

b. 測試資料與結果

i. 測試資料

```
; Example of Part 3 (Final CPU)
;
; |  InstrAddr  |   Instruction              |   Binary Code                                |   Hexadecimal  |
; |             |                            |  OPCode | Rs  | Rt  | immediate            |                |
    0x0000_0000     addu    $R26, $R02, $R03    ; 0b000000_00010_00011_11010_00000_001011 => 0x00_43_D0_0B
    0x0000_0004     subu    $R27, $R26, $R01    ; 0b000000_11010_00001_11011_00000_001101 => 0x03_41_D8_0D
    0x0000_0008     lw      $R28, 4($R12)       ; 0b010001_01100_11100_0000000000000100   => 0x45_9C_00_04
    0x0000_000C     sw      $R28, 8($R12)       ; 0b010000_01100_11100_0000000000001000   => 0x41_9C_00_08
    0x0000_0010     lw      $R29, 8($R12)       ; 0b010001_01100_11101_0000000000001000   => 0x45_9D_00_08
    0x0000_0014     addu    $R30, $R29, $R10    ; 0b000000_11101_01010_11110_00000_001011 => 0x03_AA_F0_0B
```

ii. 測試結果

左圖為記憶體結果，右圖為暫存器結果

```
25    ff
26    ff
27    12
28    34
29    56
30    78
31    12
32    34
33    56
34    78
35    ff
36    ff
```

```
26    00000000
27    77777779
28    77777778
29    12345678
30    12345678
31    12345689
32    ffffffff
```

iii. 分析

對照測試用的組合語言指令，可以看到記憶體被正確寫入，暫存器也存入正確的數值，可以知道記憶體被正確讀出。

第一、第二部分與第三部分相同，故不列出。

二、心得

這一次的作業是將上一次的作業各模組分割，加上暫存器，達成管線效果(pipe-line)，希望減少執行所需要的時間。雖然各個部分的實作不困難，可以對照講義與網路資料完成，但接線數量暴增許多，需要更細心地完成，在除厝整整一天後，發現是其中一條線沒有正確接上，令人十分難過。