

計算機組織 作業報告

作業二(PA2)

B10807005 朱育辰

一、第三部分

a. 各模組功能描述

i. RF

1. 程式碼

```
35 module RF(  
36     // Outputs  
37     output [31:0] Rs_Data,  
38     output [31:0] Rt_Data,  
39     // Inputs  
40     input RegWrite,  
41     input clk,  
42     input [4:0] Rd_Addr,  
43     input [4:0] Rt_Addr,  
44     input [4:0] Rs_Addr,  
45     input [31:0] Rd_Data  
46  
47 );  
48  
49 /*  
50  * Declaration of inner register.  
51  * CAUTION: DONT MODIFY THE NAME AND SIZE.  
52  */  
53 reg [31:0] R[0:`REG_MEM_SIZE - 1];  
54 assign Rs_Data = R[Rs_Addr];  
55 assign Rt_Data = R[Rt_Addr];  
56 always @(posedge clk) begin  
57     if (RegWrite == 1) begin  
58         R[Rd_Addr] = Rd_Data;  
59     end  
60 end  
61  
62 endmodule
```

2. 功能

根據接收到的位址，將 RS 與 RT 的資料取出。在時脈觸發時，若寫入控制訊號為 1，則將 RD 輸入資料存入 RD 位址。

ii. DM

1. 程式碼

```

module DM(
    // Outputs
    output reg [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];
always @(posedge clk) begin
    if (MemWrite == 1) begin
        {DataMem[MemAddr],DataMem[MemAddr+1],DataMem[MemAddr+2],DataMem[MemAddr+3]} = MemWriteData;
    end
    if (MemRead == 1) begin
        MemReadData = {DataMem[MemAddr],DataMem[MemAddr+1],DataMem[MemAddr+2],DataMem[MemAddr+3]};
    end
end
endmodule

```

2. 功能

根據接收到的位址與輸出控制訊號，將記憶體內相對應位址的資料讀出，在時脈觸發時，若寫入控制訊號為 1，將輸入資料存入相對應位址。

iii. IM

1. 程式碼

```

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] Instruction,
    // Inputs
    input [31:0] Instr_Addr
);

/*
 * Declaration of instruction memory.
 * CAUTION: DONT MODIFY THE NAME AND SIZE.
 */
reg [7:0]InstrMem[0:`INSTR_MEM_SIZE - 1];
always @(Instr_Addr) begin
    Instruction[31:0]
    = {InstrMem[Instr_Addr],InstrMem[Instr_Addr+1]
    },InstrMem[Instr_Addr+2],InstrMem[Instr_Addr+3]};
end
endmodule

```

2. 功能

根據輸入位址，將相對應位址的指令讀出。

iv. Control

1. 程式碼

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

always @(Opcode) begin

    case (Opcode)
        6'd0:
            begin
                RegWrite = 1;
                ALUOp = 2'b10;
                RegDst = 1;
                ALUSrc = 0;
                MemWrite = 0;
                MemRead = 0;
                MemToReg = 0;
                Jump = 0;
                Branch = 0;
            end
        6'b001100:
            begin
                RegWrite = 1;
                ALUOp = 2'b00;
                RegDst = 0;
                ALUSrc = 1;
                MemWrite = 0;
                MemRead = 0;
                MemToReg = 0;
                Jump = 0;
                Branch = 0;
            end
        6'b001101:
            begin
                RegWrite = 1;
                ALUOp = 2'b01;
                RegDst = 0;
                ALUSrc = 1;
                MemWrite = 0;
                MemRead = 0;
                MemToReg = 0;
                Jump = 0;
                Branch = 0;
            end
        6'b010000:
            begin
                RegWrite = 0;
                ALUOp = 2'b00;
                ALUSrc = 1;
                MemWrite = 1;
                MemRead = 0;
                Jump = 0;
                Branch = 0;
            end
        6'b010001:
            begin
                RegWrite = 1;
                ALUOp = 2'b00;
                RegDst = 0;
                ALUSrc = 1;
                MemWrite = 0;
                MemRead = 1;
                MemToReg = 1;
                Jump = 0;
                Branch = 0;
            end
        6'b011001:
            begin
                RegWrite = 0;
                ALUOp = 2'b01;
                ALUSrc = 0;
                MemWrite = 0;
                MemRead = 0;
                Jump = 0;
                Branch = 1;
            end
        6'b011100:
            begin
                RegWrite = 0;
                ALUOp = 2'b01;
                ALUSrc = 1;
                MemWrite = 0;
                MemRead = 0;
                Jump = 1;
                Branch = 0;
            end
        default:
            begin
                MemWrite = 0;
                RegWrite = 0;
                Jump = 0;
                Branch = 0;
            end
    endcase
end

endmodule
```

2. 功能

根據 Opcode 判斷指令類型與內容設定 ALUOp 後，依照讀寫需求設立相對應旗標，同樣依照運算與執行資料來源，設定多工器的選擇線。

v. ALU_Control

1. 程式碼

```
module ALU_Control (
    input [5:0] funct_ctrl,
    input [1:0] ALUOp,
    output reg [5:0] funct
);
    always @(funct_ctrl or ALUOp) begin
        case (ALUOp)
            2'b10:
                begin
                    case (funct_ctrl)
                        6'b001001: funct = 6'b001001;
                        6'b001101: funct = 6'b001010;
                        6'b100101: funct = 6'b010010;
                        6'b000010: funct = 6'b100010;
                        default: funct = 0;
                    endcase
                end
            2'b00:
                begin
                    funct = 6'b001001;
                end
            2'b01:
                begin
                    funct = 6'b001010;
                end
        endcase
    end
endmodule
```

2. 功能

根據 ALUOp 的值，判斷指令類型。若為 R 類，再根據 funct_ctrl 的值判斷運算需求，將 funct 設為相對應代碼；若為 I 類，則將 funct 設為減法；若為 J 類，則設為加法。

vi. ALU

1. 程式碼

```
1 module ALU (
2     input [31:0] Src1,Src2,
3     input [5:0] funct,
4     input [4:0] shamt,
5     output reg [31:0] aluResult,
6     output reg Zero
7 );
8     always @(funct or shamt or Src1 or Src2) begin
9         case (funct)
10             6'b001001: aluResult = Src1+Src2;
11             6'b001010: aluResult = Src1-Src2;
12             6'b010010: aluResult = Src1 | Src2;
13             6'b100010: aluResult = Src1 >> shamt;
14             default: aluResult = aluResult;
15         endcase
16         if (aluResult == 0) begin
17             Zero = 1;
18         end
19         else begin
20             Zero = 0;
21         end
22     end
23 endmodule
```

2. 功能

根據 funct 傳入代碼，對兩個來源進行相對應的運算，並將結果存入 aluResult，根據結果設立零旗標。若為位移運算，則根據 shamt 的量進行位移。

vii. Adder

1. 程式碼

```
1  module Adder (  
2      input [31:0] Src1,Src2,  
3      output [31:0] adderResult  
4  );  
5      assign adderResult = Src1+Src2;  
6  endmodule
```

2. 功能

將兩個來源資料相加後存入 adderResult。

viii. MUX_32bit

1. 程式碼

```
module MUX_32bit (  
    input [31:0] Src0,Src1,  
    input sel,  
    output [31:0] muxResult  
);  
    assign muxResult = sel?Src1:Src0;  
endmodule
```

2. 功能

根據 sel 輸入，決定將來源 0 或來源 1 存入 muxResult。

ix. MUX_5bit

1. 程式碼

```
1  module MUX_5bit (  
2      input [4:0] Src0,Src1,  
3      input sel,  
4      output [4:0] muxResult  
5  );  
6      assign muxResult = sel?Src1:Src0;  
7  endmodule
```

2. 功能

同上，但輸入與輸出資料長度改為 5 位元。

b. 測試資料與結果

i. RF

1. 測試資料

```
module tb_RF (
);

    wire [31:0] Rs_Data;
    wire [31:0] Rt_Data;
    // Inputs
    reg RegWrite;
    reg clk;
    reg [4:0] Rd_Addr;
    reg [4:0] Rt_Addr;
    reg [4:0] Rs_Addr;
    reg [31:0] Rd_Data;

    reg [31:0] R[0:31] ;

    RF Register_File(
        .Rs_Data(Rs_Data),
        .Rt_Data(Rt_Data),
        .RegWrite(RegWrite),
        .clk(clk),
        .Rd_Addr(Rd_Addr),
        .Rt_Addr(Rt_Addr),
        .Rs_Addr(Rs_Addr),
        .Rd_Data(Rd_Data)
    );
    integer i;
    initial begin : init
        clk = 0;
        $readmemh("testbench/RF.dat",R);

        for ( i=0 ; i<32 ; i = i+1 ) begin
            Register_File.R[i] = R[i];
        end
    end
    always begin
        #5 clk <= ~clk;
    end
    initial fork
        #0 RegWrite = 1;
        #0 Rs_Addr = 5'd0;
        #0 Rt_Addr = 5'd1;
        #0 Rd_Addr = 5'd2;
        #0 Rd_Data = 32'hFFFFFFFF;

        #10 RegWrite = 0;
        #10 Rs_Addr = 5'd2;
        #10 Rd_Addr = 5'd3;
        #10 Rd_Data = 32'h87878787;

        #20 Rs_Addr = 5'd3;

        #25 $finish;
    join
endmodule
```

2. 測試結果

	Msgs				
+ /tb_RF/Rs_Data	-No Data-	00000000	ffffff	77777777	
+ /tb_RF/Rt_Data	-No Data-	00000001			
+ /tb_RF/RegWrite	-No Data-				
+ /tb_RF/dk	-No Data-				
+ /tb_RF/Rd_Addr	-No Data-	02	03		
+ /tb_RF/Rt_Addr	-No Data-	01			
+ /tb_RF/Rs_Addr	-No Data-	00	02	03	
+ /tb_RF/Rd_Data	-No Data-	ffffff	87878787		

3. 分析

對照 RF.dat 檔案可以看到 RS 與 RT 的資料都正確讀出，後將 RS 位址設成原先的 RD 位址可以看到資料被正確寫入。將寫入控制取消後嘗試寫入，後讀取該位址，發現寫入失敗，運作正常。

ii. DM

1. 測試資料

```

module tb_DM ();
    // Outputs
    wire [31:0] MemReadData;
    // Inputs
    reg [31:0] MemAddr;
    reg [31:0] MemWriteData;
    reg MemWrite;
    reg MemRead;
    reg clk;

    reg [7:0] DataMem[0:127];
    integer i;

    DM Data_Memory(
        .MemReadData(MemReadData),
        .MemAddr(MemAddr),
        .MemWriteData(MemWriteData),
        .MemWrite(MemWrite),
        .MemRead(MemRead),
        .clk(clk)
    );

    initial begin
        clk = 0;
        $readmemh("testbench/DM.dat",DataMem);
        for (i = 0;i < 128 ;i = i+1 ) begin
            Data_Memory.DataMem[i] = DataMem[i];
        end
    end

    always begin
        #5 clk = ~clk;
    end

    initial fork
        #0 MemWriteData = 32'h01010101;
        #0 MemAddr = 32'd0;
        #0 MemWrite = 1;
        #0 MemRead = 0;

        #10 MemWrite = 0;
        #10 MemRead = 1;

        #20 $finish;
    join
endmodule

```


2. 測試結果

+/tb_DM/MemReadD...	01010101				01010101
+/tb_DM/MemAddr	0	0			
+/tb_DM/MemWriteD...	01010101	01010101			
+/tb_DM/MemWrite	0				
+/tb_DM/MemRead	1				
+/tb_DM/dk	1				

3. 分析

先設立寫入控制旗標嘗試寫入資料，再設立讀取旗標嘗試讀取並驗證寫入結果，可以看到讀寫皆正確執行。

iii. IM

1. 測試資料

```
1  module tb_IM ();
2      // Outputs
3      wire [31:0] Instruction;
4      // Inputs
5      reg [31:0] Instr_Addr;
6
7      IM Instr_Memory(
8          .Instr_Addr(Instr_Addr),
9          .Instruction(Instruction)
10     );
11
12     reg [127:0] InstrMem[0:8] ;
13     integer i;
14     initial begin
15         $readmemh("testbench/IM.dat",InstrMem);
16         for (i = 0;i < 128 ;i = i+1 ) begin
17             Instr_Memory.InstrMem[i] = InstrMem[i];
18         end
19     end
20
21     initial begin
22         #0 Instr_Addr = 0;
23         #10 $finish;
24     end
25
26 endmodule
27
```

2. 測試結果

+/tb_IM/Instruction	-No Data-	4c13001e
+/tb_IM/Instr_Addr	-No Data-	0

3. 分析

對照 IM.dat 內容，可以知道讀取正確。

iv. Control

1. 測試資料

```
1  module tb_Control ();
2      reg [5:0] Opcode;
3      wire [1:0] ALUOp;
4      wire RegDst;
5      wire Branch;
6      wire RegWrite;
7      wire ALUSrc;
8      wire MemWrite;
9      wire MemRead;
10     wire MemToReg;
11     wire Jump;
12
13     Control ctrl(
14         .Opcode(Opcode),
15         .ALUOp(ALUOp),
16         .RegDst(RegDst),
17         .Branch(Branch),
18         .RegWrite(RegWrite),
19         .ALUSrc(ALUSrc),
20         .MemWrite(MemWrite),
21         .MemRead(MemRead),
22         .MemToReg(MemToReg),
23         .Jump(Jump)
24     );
25
26     initial fork
27         #0 Opcode = 6'd0;
28         #5 Opcode = 6'b001100;
29         #10 Opcode = 6'b001101;
30         #15 Opcode = 6'b010000;
31         #20 Opcode = 6'b010001;
32         #25 Opcode = 6'b010011;
33         #30 Opcode = 6'b011100;
34         #35 Opcode = 6'b111111;
35         #40 $finish;
36     join
37 endmodule
```

2. 測試結果

/tb_Control/Opcod	-No Data-	000000	001100	001101	010000	010001	010011	011100	111111
/tb_Control/ALUOp	-No Data-	10	00	01	00		01		
/tb_Control/RegDst	-No Data-								
/tb_Control/Branch	-No Data-								
/tb_Control/RegWrite	-No Data-								
/tb_Control/ALUSrc	-No Data-								
/tb_Control/MemWrite	-No Data-								
/tb_Control/MemRead	-No Data-								
/tb_Control/MemTo...	-No Data-								
/tb_Control/Jump	-No Data-								

3. 分析

對照作業說明，輸入各種 Opcod，各旗標與 ALUOp 皆正確設立。

v. ALU_Control

1. 測試資料

```

1  module tb_ALU_Control ();
2      reg [5:0] funct_ctrl;
3      reg [1:0] ALUOp;
4      wire [5:0] funct;
5      ALU_Control alu_ctrl(
6          .funct_ctrl(funct_ctrl),
7          .ALUOp(ALUOp),
8          .funct(funct)
9      );
10
11     initial fork
12         #0 funct_ctrl = 6'b001011;
13         #0 ALUOp = 2'b10;
14         #5 funct_ctrl = 6'b001101;
15         #10 funct_ctrl = 6'b100101;
16         #15 funct_ctrl = 6'b000010;
17         #20 ALUOp = 2'b00;
18         #25 ALUOp = 2'b01;
19         #30 $finish;
20     join
21 endmodule

```

2. 測試結果

tb/funct_ctrl	000010	001011	001101	100101	000010		
tb/ALUOp	01	10			00	01	
tb/funct	001010	001001	001010	010010	100010	001001	001010

3. 分析

對照作業說明，對各種指令類型與運算需求正確輸出 funct。

vi. ALU

1. 測試資料

```
1  module tb_ALU ();
2      reg [31:0] Src1,Src2;
3      reg [5:0] funct;
4      reg [4:0] shamt;
5      wire [31:0] aluResult;
6      wire Zero;
7
8      ALU alu(
9          .Src1(Src1),
10         .Src2(Src2),
11         .funct(funct),
12         .shamt(shamt),
13         .aluResult(aluResult),
14         .Zero(Zero)
15     );
16     initial fork
17         #0 Src1 = 32'h000000f0;
18         #0 Src2 = 32'd15;
19         #0 shamt = 5'd4;
20         #0 funct = 6'b001001;
21         #5 funct = 6'b001010;
22         #10 funct = 6'b010010;
23         #15 funct = 6'b100010;
24         #20 Src1 = 32'd15;
25         #20 funct = 6'b001010;
26         #25 funct = 6'b111111;
27         #30 $finish;
28     join
29
30     endmodule
31
32
```

2. 測試結果

/Src1	-No Data-	240				15	
/Src2	-No Data-	15					
/funct	-No Data-	001001	001010	010010	100010	001010	111111
/shamt	-No Data-	00100					
/aluResult	-No Data-	255	225	255	15	0	
/Zero	-No Data-						

3. 分析

根據 funct 進行相對應驗算後，運算結果與旗標正確。

vii. Adder

1. 測試資料

```
1 module tb_Adder ();
2     reg [31:0] Src1,Src2;
3     wire [31:0] adderResult;
4
5     Adder adder(
6         .Src1(Src1),
7         .Src2(Src2),
8         .adderResult(adderResult)
9     );
10
11     initial fork
12         #0 Src1 = 32'd4;
13         #0 Src2 = 32'd23;
14         #5 Src1 = 32'd14;
15         #10 $finish;
16     join
17 endmodule
```

2. 測試結果

r/Src1	-No Data-	4	14
r/Src2	-No Data-	23	
r/adderRe...	-No Data-	27	37

3. 分析

驗算過後，可以確認計算結果正確。

viii. MUX_32bit

1. 測試資料

```
1 module tb_MUX_32bit ();
2     reg [31:0] Src0,Src1;
3     reg sel;
4     wire [31:0] muxResult;
5     MUX_32bit mux(
6         .Src0(Src0),
7         .Src1(Src1),
8         .sel(sel),
9         .muxResult(muxResult)
10    );
11    initial fork
12        #0 Src0 = 32'd0;
13        #0 Src1 = 32'hFFFFFFFF;
14        #0 sel = 0;
15        #5 sel = 1;
16        #10 $finish;
17    join
18 endmodule
```

2. 測試結果

it/Src0	00000000000000...	00000000000000000000000000000000							
it/Src1	11111111111111...	11111111111111111111111111111111							
it/sel	1								
it/mux...	11111111111111...	00000000000000000000000000000000	11111111111111111111111111111111						

3. 分析

可以看見多工器成功根據 sel 訊號選擇輸出。

ix. MUX_5bit

1. 測試資料

```
1  module tb_MUX_5bit ();
2      reg [4:0] Src0,Src1;
3      reg sel;
4      wire [4:0] muxResult;
5      MUX_5bit mux(
6          .Src0(Src0),
7          .Src1(Src1),
8          .sel(sel),
9          .muxResult(muxResult)
10     );
11     initial fork
12         #0 Src0 = 5'd0;
13         #0 Src1 = 5'b11111;
14         #0 sel = 0;
15         #5 sel = 1;
16         #10 $finish;
17     join
18 endmodule
```

2. 測試結果

t/Src0	00000	00000							
t/Src1	11111	11111							
t/sel	1								
t/muxR...	11111	00000	11111						

3. 分析

可以看見多工器成功根據 sel 訊號選擇輸出。

第一與第二部分同第三部分。

二、心得

這一次的作業，實作各個模組本身並不算困難，只需要詳細閱讀作業說明對各個指令的說明與講義對 CPU 的介紹即可，但比較新的部分是對資料的讀寫，雖然在模組的程式裡要寫的地方沒什麼不同，但在測試資料的部分需要參考網路上對資料讀寫的說明與示範才能完成。