# 計算機組織 作業報告

作業二(PA2)

# 一、第三部分

- a. 各模組功能描述
  - i. RF
- 1. 程式碼

```
module RF(
         // Outputs
         output [31:0] Rs_Data,
         output [31:0] Rt_Data,
         // Inputs
         input RegWrite,
          input clk,
         input [4:0] Rd Addr,
         input [4:0] Rt_Addr,
         input [4:0] Rs Addr,
          input [31:0] Rd_Data
     );
          * Declaration of inner register.
          * CAUTION: DONT MODIFY THE NAME AND SIZE.
          reg [31:0]R[0:`REG_MEM_SIZE - 1];
          assign Rs_Data = R[Rs_Addr];
         assign Rt_Data = R[Rt_Addr];
          always @(posedge clk) begin
              if (RegWrite == 1) begin
                  R[Rd_Addr] = Rd_Data;
59
              end
         end
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

# 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

#### 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

```
output reg [1:0] ALUOp,
output reg RegDst,
output reg Regust,
output reg Branch,
output reg Regwhite,
output reg ALUSrc,
output reg MemWrite,
output reg MemRead,
output reg MemToReg,
 output reg Jump
                                                                                                                              RegWrite = 1;
ALUOp = 2'b00;
                                                                                                                               RegDst = 0;
             MemWrite = 0;
                                                                                                                               MemRead = 1;
                                                                                                                               MemToReg = 1;
                                                                                                                              Jump = 0;
Branch = 0;
                     MemWrite = 0;
MemRead = 0;
MemToReg = 0;
Jump = 0;
Branch = 0;
                                                                                                                               RegWrite = 0;
       end
6'b001100:
                    Pagnetic = 1;
ALUOp = 2'b00;
RegDst = 0;
ALUSrc = 1;
MemMrite = 0;
MemRead = 0;
MemToReg = 0;
Jump = 0;
Branch = 0;
                                                                                                                               MemWrite = 0;
                                                                                                                               MemRead = 0;
                                                                                                                              Jump = 0;
Branch = 1;
                                                                                                              6'b011100:
                                                                                                                               RegWrite = 0;
                                                                                                                               ALUOp = 2'b01;
       end
6'b001101:
              MemWrite = 0;
                                                                                                                               MemRead = 0;
                                                                                                                              Jump = 1;
Branch = 0;
                                                                                                                              MemWrite = 0;
RegWrite = 0;
       end
6'b010000:
               begin

RegWrite = 0;

ALUOP = 2'b00;

ALUSrc = 1;

MemWrite = 1;

MemRead = 0;
                      Jump = 0;
Branch = 0;
```

# 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

#### 2. 功能

根據 ALUOp 的值,判斷指令類型。若為 R 類,再根據 func\_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

```
module ALU (
   input [31:0] Src1,Src2,
input [5:0] funct,
   input [4:0] shamt,
   output reg [31:0] aluResult,
   output reg Zero
   always @(funct or shamt or Src1 or Src2) begin
       case (funct)
          6'b001001: aluResult = Src1+Src2;
            6'b001010: aluResult = Src1-Src2;
           6'b010010: aluResult = Src1 | Src2;
           6'b100010: aluResult = Src1 >> shamt;
           default:aluResult = aluResult;
        if (aluResult == 0) begin
           Zero = 1;
        end
       else begin
          Zero = 0;
```

#### 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

```
module Adder (
input [31:0] Src1,Src2,
output [31:0] adderResult

input [31:0] adderResult

assign adderResult = Src1+Src2;
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

```
module MUX_5bit (
input [4:0] Src0,Src1,
input sel,
output [4:0] muxResult

);
assign muxResult = sel?Src1:Src0;
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;
   reg RegWrite;
   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];
       .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)
       $readmemh("testbench/RF.dat",R);
        for ( i=0 ; i<32 ; i=i+1 ) begin
            Register_File.R[i] = R[i];
       #5 clk <= ~clk;
       #0 RegWrite = 1;
       #0 Rs_Addr = 5'd0;
#0 Rt_Addr = 5'd1;
       #0 Rd_Addr = 5'd2;
       #0 Rd_Data = 32'hFFFFFFF;
       #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;
```

<b>≨</b> 1 +	Msgs		1-		
+> /tb_RF/Rs_Data	-No Data-	00000000		fffffff	77777777
<b>≖</b> - <b>/</b> /tb_RF/Rt_Data	-No Data-	00000001			
/tb_RF/RegWrite	-No Data-				
√ /tb_RF/dk	-No Data-				
<b>≖</b> – <b>∜</b> /tb_RF/Rd_Addr	-No Data-	02		03	
<b>II</b> - /tb_RF/Rt_Addr	-No Data-	01			
<b>II</b> - /tb_RF/Rs_Addr	-No Data-	00		02	03
<b>II</b> - <b>◇</b> /tb_RF/Rd_Data	-No Data-	fffffff		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

```
wire [31:0] MemReadData;
reg [31:0] MemAddr;
reg [31:0] MemWriteData;
reg MemWrite;
reg MemRead;
reg clk;
reg [7:0] DataMem[0:127];
    .MemReadData(MemReadData),
    .MemAddr(MemAddr),
    .MemWriteData(MemWriteData),
.MemWrite(MemWrite),
    .MemRead(MemRead),
    .clk(clk)
    $readmemh("testbench/DM.dat",DataMem);
    for (i = 0; i < 128; i = i+1) begin
       Data_Memory.DataMem[i] = DataMem[i];
   #0 MemWriteData = 32'h01010101;
    #0 MemAddr = 32'd0;
#0 MemWrite = 1;
    #0 MemRead = 0;
    #10 MemWrite = 0;
    #10 MemRead = 1;
```

<b>II</b> → /tb_DM/MemReadD	01010101			01010101
<b>≖</b> – <b>∜</b> /tb_DM/MemAddr	0	0		
<b>±</b> - <b>∜</b> /tb_DM/MemWriteD	01010101	01010101		
/tb_DM/MemWrite	0			
/tb_DM/MemRead	1			
/tb_DM/dk	1			

#### 3. 分析

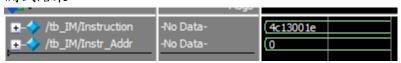
先設立寫入控制旗標嘗試寫入資料,再設立讀取旗標嘗試讀取 並驗證寫入結果,可以看到讀寫皆正確執行。

#### iii. IM

1. 測試資料

```
module tb_IM ();
         // Outputs
         wire [31:0] Instruction;
         // Inputs
         reg [31:0] Instr_Addr;
         IM Instr_Memory(
             .Instr_Addr(Instr_Addr),
             .Instruction(Instruction)
         );
         reg [127:0] InstrMem[0:8];
         integer i;
L4
         initial begin
L5
             $readmemh("testbench/IM.dat", InstrMem);
             for (i = 0; i < 128; i = i+1) begin
                 Instr_Memory.InstrMem[i] = InstrMem[i];
             end
         end
20
21
         initial begin
22
             #0 Instr_Addr = 0;
23
             #10 $finish;
24
         end
25
26
     endmodule
```

#### 2. 測試結果



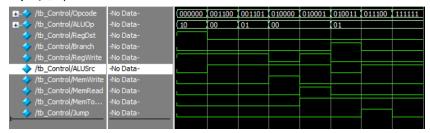
3. 分析

對照 IM. dat 內容,可以知道讀取正確。

#### iv. Control

1. 測試資料

```
module tb Control ();
         reg [5:0] Opcode;
         wire [1:0] ALUOp;
         wire RegDst;
         wire Branch;
         wire RegWrite;
         wire ALUSrc;
         wire MemWrite;
         wire MemRead;
         wire MemToReg;
11
         wire Jump;
12
13
         Control ctrl(
14
              .Opcode(Opcode),
              .ALUOp(ALUOp),
              .RegDst(RegDst),
              .Branch(Branch),
              .RegWrite(RegWrite),
              .ALUSrc(ALUSrc),
              .MemWrite(MemWrite),
21
              .MemRead(MemRead),
              .MemToReg(MemToReg),
23
              .Jump(Jump)
24
         );
         initial fork
             #0 Opcode = 6'd0;
             #5 Opcode = 6'b001100;
             #10 Opcode = 6'b001101;
             #15 Opcode = 6'b010000;
             #20 Opcode = 6'b010001;
             #25 \ Opcode = 6'b010011;
             #30 Opcode = 6'b011100;
             #35 Opcode = 6'b111111;
             #40 $finish;
35
         join
     endmodule
```



#### 3. 分析

對照作業說明,輸入各種 Opcode,各旗標與 ALUOp 皆正確設立。

# v. ALU\_Control

#### 1. 測試資料

```
module tb_ALU_Control ();
         reg [5:0] funct_ctrl;
         reg [1:0] ALUOp;
         wire [5:0] funct;
         ALU_Control alu_ctrl(
              .funct_ctrl(funct_ctrl),
              .ALUOp(ALUOp),
              .funct(funct)
         );
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 \text{ ALUOp} = 2'b00;
              #25 ALUOp = 2'b01;
18
19
              #30 $finish;
         join
20
21
     endmodule
```

#### 2. 測試結果

12.4 = 41.12 >1.	maya						
ol/funct_ctrl	000010	001011	001101	100101	000010		
ol/ALUOp	01	10				00	01
ol/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

```
module tb_ALU ();
          reg [31:0] Src1, Src2;
          reg [5:0] funct;
          reg [4:0] shamt;
          wire [31:0] aluResult;
          wire Zero;
          ALU alu(
              .Src1(Src1),
              .Src2(Src2),
11
              .funct(funct),
12
              .shamt(shamt),
13
               .aluResult(aluResult),
               .Zero(Zero)
15
          initial fork
16
              \#0 \text{ Src1} = 32'h000000f0;
17
18
              \#0 \text{ 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;
              #20 Src1 = 32'd15;
24
25
              #20 funct = 6'b001010;
26
              #25 funct = 6'b111111;
27
              #30 $finish;
28
29
30
          join
      endmodule
32
```

#### 2. 測試結果

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

#### 3. 分析

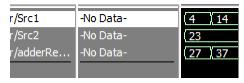
根據 funct 進行相對應驗算後,運算結果與旗標正確。

# vii. Adder

1. 測試資料

```
module tb_Adder ();
2
          reg [31:0] Src1, Src2;
         wire [31:0] adderResult;
          Adder adder(
              .Src1(Src1),
              .Src2(Src2),
              .adderResult(adderResult)
          );
11
          initial fork
12
              #0 Src1 = 32'd4;
              \#0 Src2 = 32'd23;
              \#5 \ Src1 = 32'd14;
15
              #10 $finish;
          join
17
     endmodule
```

2. 測試結果



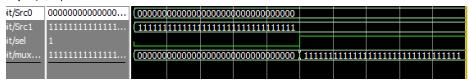
3. 分析

驗算過後,可以確認計算結果正確。

# viii. MUX 32bit

1. 測試資料

```
module tb_MUX_32bit ();
         reg [31:0] Src0,Src1;
         reg sel;
         wire [31:0] muxResult;
         MUX_32bit mux(
              .Src0(Src0),
              .Src1(Src1),
              .sel(sel),
              .muxResult(muxResult)
         initial fork
              \#0 \text{ Src0} = 32'd0;
              #0 Src1 = 32'hFFFFFFF;
              \#0 sel = 0;
              #5 sel = 1;
16
              #10 $finish;
          join
     endmodule
```



3. 分析

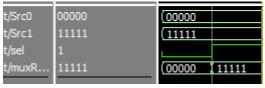
可以看見多工器成功根據 sel 訊號選擇輸出。

# ix. MUX 5bit

1. 測試資料

```
module tb MUX 5bit ();
         reg [4:0] Src0, Src1;
         reg sel;
         wire [4:0] muxResult;
         MUX_5bit mux(
              .Src0(Src0),
              .Src1(Src1),
              .sel(sel),
              .muxResult(muxResult)
10
         );
         initial fork
11
12
              #0 Src0 = 5'd0;
              #0 Src1 = 5'b111111;
13
14
              #0 sel = 0;
15
              #5 sel = 1;
16
              #10 $finish;
17
         join
     endmodule
```

#### 2. 測試結果



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。