

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 | 劉杰閔

● Part I

a. R_PipelineCPU.v

```

28 //
29 module R_PipelineCPU(
30     // Outputs
31     output wire [31:0] AddrOut,
32     // Inputs
33     input wire [31:0] AddrIn,
34     input wire clk
35 );
36
37 // Instruction Memory
38 wire [31:0] Instr;
39 wire [31:0] Instr_out;
40
41 // ID/EX.
42 wire [31:0] ALU_result;
43 wire [31:0] RsData;
44 wire [31:0] RsData_out;
45 wire [31:0] RtData;
46 wire [31:0] RtData_out;
47 wire [1:0] ALUOp;
48 wire [1:0] ALUOp_out;
49 wire [5:0] funct_out;
50 wire [4:0] shamt_out;
51 wire [5:0] Funct;
52 wire [4:0] RdAddr_out;
53 wire RegWrite;
54 wire RegWrite_out;
55
56 // EX/MEM
57 wire RegWrite_mem_out;
58 wire [31:0] ALU_result_mem_out;
59 wire [4:0] RdAddr_mem_out;

```

```

61 // EX/MEM
62 wire RegWrite_wb_out;
63 wire [31:0] ALU_result_wb_out;
64 wire [4:0] RdAddr_wb_out;
65
66

```

```

74 IM Instr_Memory(
75     // Outputs
76     .Instr(Instr),
77     // Inputs
78     .InstrAddr(AddrIn)
79 );
80 Adder adder(
81     // Outputs
82     .AddrOut(AddrOut),
83     // Inputs
84     .AddrIn(AddrIn)
85 );
86 IF ID Fetch_Decode(
87     .Instr(Instr),
88     .InstrOut(Instr_out),
89     .clk(clk)
90 );

```

```

91 ID_EX Decode_Execute(
92     // Inputs
93     .RegWrite_in(RegWrite),
94     .clk(clk),
95     .ALUOp_in(ALUOp),
96     .RsData_in(RsData),
97     .RtData_in(RtData),
98     .funct_in(Instr_out[5:0]),
99     .shamt_in(Instr_out[10:6]),
100     .RdAddr_in(Instr_out[15:11]),
101     // Outputs
102     .ALUOp_out(ALUOp_out),
103     .RsData_out(RsData_out),
104     .RtData_out(RtData_out),
105     .funct_out(funct_out),
106     .shamt_out(shamt_out),
107     .RdAddr_out(RdAddr_out),
108     .RegWrite_out(RegWrite_out)
109 );
110
111 Control_controller(
112     // Inputs
113     .OpCode(Instr_out[31:26]),
114     // Outputs
115     .RegWrite(RegWrite), // To ID/EX.
116     .ALUOp(ALUOp) // To ID/EX.
117 );
118
119 /*
120  * Declaration of Register File.
121  * CAUTION: DONT MODIFY THE NAME.
122  */
123 RF Register_File(
124     // Outputs
125     .RsData(RsData), // TO ID/EX.
126     .RtData(RtData), // TO ID/EX.
127     // Inputs
128     .clk(clk),
129     .RegWrite(RegWrite_wb_out), // From EX/MEM
130     .RsAddr(Instr_out[25:21]),
131     .RtAddr(Instr_out[20:16]),
132     .RdAddr(RdAddr_wb_out),
133     .RdData(ALU_result_wb_out) // From EX/MEM
134 );

```

```

136     ALU_alu(
137         // Inputs
138         .Src1(RsData_out),
139         .Src2(RtData_out),
140         .Shamt(shamt_out),
141         .Funct(Funct),
142         // Outputs
143         .result(ALU_result)
144     );
145
146     ALU_Control_ALU_controller(
147         // Inputs
148         .funct(funct_out),
149         .ALUOp(ALUOp_out),
150         // Outputs
151         .Funct(Funct)
152     );
153
154     EX_MEM_Execute_Memory(
155         // Inputs
156         .clk(clk),
157         .ALU_result_in(ALU_result),
158         .RdAddr_in(RdAddr_out),
159         .RegWrite_in(RegWrite_out),
160         //Outputs
161         .RdAddr_out(RdAddr_mem_out),
162         .ALU_result_out(ALU_result_mem_out),
163         .RegWrite_out(RegWrite_mem_out)
164     );
165
166     MEM_WB_Memory_WriteBack(
167         // Inputs
168         .clk(clk),
169         .ALU_result_in(ALU_result_mem_out),
170         .RdAddr_in(RdAddr_mem_out),
171         .RegWrite_in(RegWrite_mem_out),
172         //Outputs
173         .RdAddr_out(RdAddr_wb_out),
174         .ALU_result_out(ALU_result_wb_out),
175         .RegWrite_out(RegWrite_wb_out)
176     );
177
178 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	RF.out
1	1 // my
2	2 00000000
3	3 00000001
4	4 00000002
5	5 77777777
6	6 7e7e7e7e
7	7 fe7e7e7e
8	8 7fffffff
9	9 80000000
10	10 ffffffff0000
11	11 0000ffff
12	12 00000011
13	13 00000023
14	14 00000017
15	15 00000090
16	16 00000100
17	17 00000250
18	18 00000300
19	19 00000037
20	20 00000064
21	21 00000030
22	22 00000034
23	23 00000079
24	24 00000024
25	25 00040000
26	26 00000000
27	27 00000000
28	28 00000000
29	29 00000000
30	30 00000000
31	31 00000000
32	32 ffffffff
33	33 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我程式碼的方式去抓{0, 1, 2, 3}.

```
29 `define INSTR_MEM_SIZE 128 // Bytes
30 /*
31  * Declaration of Instruction Memory for this project.
32  * CAUTION: DONT MODIFY THE NAME.
33  */
34 module IM(
35     // Outputs
36     output reg [31:0] Instr,
37     // Inputs
38     input [31:0] InstrAddr
39 );
40
41 /*
42  * Declaration of instruction memory.
43  * CAUTION: DONT MODIFY THE NAME AND SIZE.
44  */
45 reg [7:0] InstrMem[0:`INSTR_MEM_SIZE - 1];
46
47 always@(InstrAddr)
48 begin
49     Instr[31:0] = {InstrMem[InstrAddr], InstrMem[InstrAddr+1], InstrMem[InstrAddr+2], InstrMem[InstrAddr+3]};
50 end
51
52 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的地址去抓值去輸出。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。）

```
29 `define REG_MEM_SIZE 32 // Words
30 /*
31  * Declaration of Register File for this project.
32  * CAUTION: DONT MODIFY THE NAME.
33  */
34 module RF(
35     // Outputs
36     output [31:0] RsData,
37     output [31:0] RtData,
38     // Inputs
39     input RegWrite,
40     input clk,
41     input [4:0] RsAddr,
42     input [4:0] RtAddr,
43     input [4:0] RdAddr,
44     input [31:0] RdData
45 );
46
47 /*
48  * Declaration of inner register.
49  * CAUTION: DONT MODIFY THE NAME AND SIZE.
50  */
51 reg [31:0] R[0:`REG_MEM_SIZE - 1];
52
53 assign RsData = R[RsAddr];
54 assign RtData = R[RtAddr];
55
56 always@(posedge clk)
57 begin
58     if(RegWrite == 1)
59     begin
60         R[RdAddr] = RdData;
61     end
62     else
63     begin
64         R[RdAddr] = R[RdAddr];
65     end
66 end
67
68 endmodule
```

```

Adder.v
You, 3 days ago | 1 author (You)
1  module Adder
2  (
3      input  [31:0] AddrIn,
4      output [31:0] AddrOut
5  );
6
7      assign AddrOut = AddrIn + 32'd 4;
8
9  endmodule
10

```

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。

```

1  `define addu 6'b 001011
2  `define subu 6'b 001101
3  `define AND 6'b 010010
4  `define sll 6'b 100110
5
6  module ALU(
7      input  [31:0] Src1,
8      input  [31:0] Src2,
9      input  [4:0] Shamt,
10     input  [5:0] Funct,
11     output reg [31:0] result
12 );
13
14 always@(Funct or Shamt or Src1 or Src2)
15 begin
16     case (Funct)
17         `addu : result = Src1 + Src2;
18         `subu : result = Src1 - Src2;
19         `AND  : result = Src1 & Src2;
20         `sll  : result = Src1 << Shamt;
21         default: result = result; // if i
22     endcase
23 end
24
25
26 endmodule

```

```

You, 3 days ago | 1 author (You)
1  `define addu 6'b 001011
2  `define subu 6'b 001101
3  `define AND 6'b 010010
4  `define sll 6'b 100110
5
6  `define input_addu 6'b 001011
7  `define input_subu 6'b 001101
8  `define input_and 6'b 010010
9  `define input_sll 6'b 100110
10 `define R_type 2'b10
11
12 module ALU_Control(
13     input  [5:0] funct,
14     input  [1:0] ALUOp,
15     output reg [5:0] Funct
16 );
17
18 always@(funct or ALUOp)
19 begin
20     case(ALUOp)
21         `R_type:
22             begin
23                 case(funct)
24                     `input_addu : Funct = `addu;
25                     `input_subu : Funct = `subu;
26                     `input_and  : Funct = `AND;
27                     `input_sll  : Funct = `sll;
28                     default: Funct = 0;
29                 endcase
30             end
31         default: Funct = 0;
32     endcase
33 end
34 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)
1  module Control(
2      input  [5:0]  OpCode,
3      output reg    RegWrite,
4      output reg [1:0]  ALUOp
5  );
6
7  always@ (OpCode)
8  begin
9      case (OpCode)
10         6'd 4:
11             begin
12                 RegWrite = 1'b 1; // R format.
13                 ALUOp = 2'b10;
14             end
15         default:
16             begin
17                 RegWrite = 0; // no this opcode.
18                 ALUOp = 2'b00;
19             end
20         endcase
21     end
22 end
23
24 endmodule

```

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

```

You, 3 days ago | 1 author (You)
1  module ID_EX(
2      input RegWrite_in, // WB
3      input clk,
4      input [1:0] ALUOp_in, // EX
5      input [5:0] funct_in,
6      input [4:0] shamt_in,
7      input [4:0] RdAddr_in,
8      input [31:0] RsData_in,
9      input [31:0] RtData_in,
10     output reg [1:0] ALUOp_out, //EX
11     output reg [31:0] RsData_out,
12     output reg [31:0] RtData_out,
13     output reg [4:0] RdAddr_out,
14     output reg [5:0] funct_out,
15     output reg [4:0] shamt_out,
16     output reg RegWrite_out // WB
17 );
18
19 reg WB;
20 reg [1:0] EX;
21 reg [5:0] RdAddr_reg;
22 reg [5:0] funct_reg;
23 reg [5:0] shamt_reg;
24 reg [31:0] RsData_reg;
25 reg [31:0] RtData_reg;

```

```

27 always@(posedge clk or negedge clk)
28 begin
29     if(clk == 1) // put them in to th
30     begin
31         WB = RegWrite_in;
32         EX = ALUOp_in;
33         RdAddr_reg = RdAddr_in;
34         funct_reg = funct_in;
35         shamt_reg = shamt_in;
36         RsData_reg = RsData_in;
37         RtData_reg = RtData_in;
38     end
39     else
40     begin
41         RegWrite_out = WB;
42         ALUOp_out = EX;
43         RdAddr_out = RdAddr_reg;
44         funct_out = funct_reg;
45         shamt_out = shamt_reg;
46         RsData_out = RsData_reg;
47         RtData_out = RtData_reg;
48     end
49 end
50 endmodule

```

```

1  module IF_ID(
2      input [31:0] Instr,
3      input clk,
4      output reg [31:0] InstrOut
5  );
6      reg [31:0] Instr_reg;
7
8      always@(posedge clk or negedge clk)
9      begin
10         if(clk == 1) //when posedge
11             Instr_reg = Instr;
12         else // when negedge clk, out
13             InstrOut = Instr_reg;
14         end
15     endmodule

```

```

1  module MEM_WB(
2      input clk,
3      input RegWrite_in, // WB
4      input [31:0] ALU_result_in,
5      input [4:0] RdAddr_in,
6      output reg [4:0] RdAddr_out,
7      output reg [31:0] ALU_result_out,
8      output reg RegWrite_out // WB
9  );
10     reg WB;
11     reg [4:0] RdAddr_reg;
12     reg [31:0] ALU_result_reg;
13     reg RegWrite_reg;
14
15     always@(posedge clk or negedge clk)
16     begin
17         if(clk == 1) // put them in to the reg
18         begin
19             WB = RegWrite_in;
20             RdAddr_reg = RdAddr_in;
21             ALU_result_reg = ALU_result_in;
22         end
23         else
24         begin
25             RegWrite_out = WB;
26             RdAddr_out = RdAddr_reg;
27             ALU_result_out = ALU_result_reg;
28         end
29     end
30 endmodule

```

```

1  module EX_MEM(
2      input clk,
3      input RegWrite_in, // WB
4      input [31:0] ALU_result_in,
5      input [4:0] RdAddr_in,
6      output reg [4:0] RdAddr_out,
7      output reg [31:0] ALU_result_out,
8      output reg RegWrite_out // WB
9  );
10     reg WB;
11     reg [4:0] RdAddr_reg;
12     reg [31:0] ALU_result_reg;
13
14     always@(posedge clk or negedge clk)
15     begin
16         if(clk == 1) // put them in to the reg
17         begin
18             WB = RegWrite_in;
19             RdAddr_reg = RdAddr_in;
20             ALU_result_reg = ALU_result_in;
21         end
22         else
23         begin
24             RegWrite_out = WB;
25             RdAddr_out = RdAddr_reg;
26             ALU_result_out = ALU_result_reg;
27         end
28     end
29 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 II.

a. I_PipelineCPU

```
29 module I_PipelineCPU(  
30     // Outputs  
31     output wire [31:0] AddrOut,  
32     // Inputs  
33     input wire [31:0] AddrIn,  
34     input wire clk  
35 );  
36  
37 // Instruction Memory  
38 wire [31:0] Instr;  
39 wire [31:0] Instr_out;  
40  
41 // ID/EX.  
42  
43 // WB  
44  
45 wire RegWrite;  
46 wire MemtoReg;  
47 wire MemtoReg_out;  
48 wire RegWrite_out;  
49  
50  
51 // memory.  
52  
53 wire MemWrite;  
54 wire MemWrite_out;  
55  
56 wire MemRead;  
57 wire MemRead_out;  
58  
59 // EX.  
60 wire [1:0] ALUOp;  
61 wire [1:0] ALUOp_out;  
62  
63 wire RegDst;  
64 wire RegDst_out;  
65  
66 wire ALUSrc;  
67 wire ALUSrc_out;
```

```
68 // Others.  
69 wire [31:0] RsData;  
70 wire [31:0] RsData_out;  
71 wire [31:0] RtData;  
72 wire [31:0] RtData_out;  
73 wire [31:0] Sign_Extend; // immedi  
74 wire [31:0] immediate_out;  
75 wire [4:0] RdAddr_out;  
76 wire [4:0] RtAddr_out;  
77  
78  
79 // EX/MEM  
80  
81 // WB.  
82 wire RegWrite_mem_out;  
83 wire MemtoReg_mem_out;  
84 // Memory  
85 wire MemWrite_mem_out;  
86 wire MemRead_mem_out;  
87 // Others.  
88 wire [31:0] ALU_result;  
89 wire [31:0] MemAddr;  
90  
91 wire [31:0] MemWriteData; // RtDa  
92  
93 wire [4:0] RdAddr_mem_out;  
94  
95 // MEM/WB  
96  
97 // WB.  
98 wire RegWrite_wb_out;  
99 wire MemtoReg_wb_out;  
100 // Others  
101 wire [31:0] MemAddr_wb_out;  
102 wire [4:0] RdAddr_wb_out;  
103 wire [31:0] MemReadData;  
104 wire [31:0] MemReadData_wb_out;  
105
```

```
106  
107 // ALU controller  
108 wire [5:0] Funct;  
109 // MUX  
110 wire [31:0] MUX32A_result;  
111 wire [31:0] MUX32B_result;  
112 wire [4:0] MUX5_result;  
113  
114 /*  
115 * Declaration of Instruction  
116 * CAUTION: DONT MODIFY THE M  
117 */  
118 IM Instr_Memory(  
119     // Outputs  
120     .Instr(Instr),  
121     // Inputs  
122     .InstrAddr(AddrIn)  
123 );  
124  
125 Adder adder(  
126     // Outputs  
127     .AddrOut(AddrOut),  
128     // Inputs  
129     .AddrIn(AddrIn)  
130 );  
131  
132 IF_ID_Fetch_Decode(  
133     .Instr(Instr),  
134     .InstrOut(Instr_out),  
135     .clk(clk)  
136 );  
137  
138 Control_controller(  
139     .OpCode(Instr_out[31:26]),  
140     .RegWrite(RegWrite),  
141     .RegDst(RegDst),  
142     .ALUSrc(ALUSrc),  
143     .MemWrite(MemWrite),  
144     .MemRead(MemRead),  
145     .MemtoReg(MemtoReg),  
146     .ALUOp(ALUOp)  
147 );  
148
```

```
154 RF_Register_File(  
155     // Outputs  
156     .RsData(RsData), // TO ID/EX.  
157     .RtData(RtData), // TO ID/EX.  
158     // Inputs  
159     .clk(clk),  
160     .RegWrite(RegWrite_wb_out), // From MEM/WB.  
161     .RsAddr(Instr_out[25:21]),  
162     .RtAddr(Instr_out[20:16]),  
163     .RdAddr(RdAddr_wb_out),  
164     .RdData(MUX32B_result) // From MEM/WB.  
165 );  
166  
167 assign Sign_Extend[31:0] = Instr_out[15]?{16'hFFFF, Instr_out[15:0]}:{16'h0000, Instr_out[15:0]};
```

```
168 ID_EX_Decode_Execute(  
169     // Inputs  
170     .clk(clk),  
171     // WB  
172     .RegWrite_in(RegWrite),  
173     .Mem2Reg_in(MemtoReg),  
174     // Memory  
175     .MemRead_in(MemRead),  
176     .MemWrite_in(MemWrite),  
177     // EX  
178     .ALUOp_in(ALUOp),  
179     .RegDst_in(RegDst),  
180     .ALUSrc_in(ALUSrc),  
181     // Others  
182     .RsData_in(RsData),  
183     .RtData_in(RtData),  
184     .immediate_in(Sign_Extend),  
185     .RdAddr_in(Instr_out[15:11]),  
186     .RtAddr_in(Instr_out[20:16]),  
187  
188     // Outputs  
189     .RegWrite_out(RegWrite_out),  
190     .Mem2Reg_out(MemtoReg_out),  
191     // Memory  
192     .MemRead_out(MemRead_out),  
193     .MemWrite_out(MemWrite_out),  
194     // EX  
195     .ALUOp_out(ALUOp_out),  
196     .RegDst_out(RegDst_out),  
197     .ALUSrc_out(ALUSrc_out),  
198     // Others  
199     .RsData_out(RsData_out),  
200     .RtData_out(RtData_out),  
201     .immediate_out(immediate_out),  
202     .RdAddr_out(RdAddr_out),  
203     .RtAddr_out(RtAddr_out)  
204 );  
205
```

```
206 Mux32b_A32(  
207     .Src1(RtData_out),  
208     .Src2(immediate_out),  
209     .result(MUX32A_result),  
210     .choose(ALUSrc_out)  
211 );  
212  
213 ALU_alu(  
214     // Inputs  
215     .Src1(RsData_out),  
216     .Src2(MUX32A_result),  
217     .Shamt(immediate_out[10:6]),  
218     .Funct(Funct),  
219     // Outputs  
220     .result(ALU_result)  
221 );  
222  
223 ALU_Control_ALU_controller(  
224     // Inputs  
225     .funct(immediate_out[5:0]),  
226     .ALUOp(ALUOp_out),  
227     // Outputs  
228     .Funct(Funct)  
229 );  
230  
231 Mux5b_mux5b(  
232     .Src1(RtAddr_out),  
233     .Src2(RdAddr_out),  
234     .choose(RegDst_out),  
235     .result(MUX5_result)  
236 );  
237  
238
```

```
241 EX_MEM_Execute_Memory(  
242     // Inputs  
243     .clk(clk),  
244     // WB  
245     .RegWrite_in(RegWrite_out),  
246     .Mem2Reg_in(MemtoReg_out),  
247     // MEM  
248     .MemRead_in(MemRead_out),  
249     .MemWrite_in(MemWrite_out),  
250     // Others  
251     .ALU_result_in(ALU_result),  
252     .RtData_in(RtData_out),  
253     .RdAddr_in(MUX5_result),  
254     // Outputs  
255     .RegWrite_out(RegWrite_mem_out),  
256     .Mem2Reg_out(MemtoReg_mem_out),  
257     // Memory  
258     .MemRead_out(MemRead_mem_out),  
259     .MemWrite_out(MemWrite_mem_out),  
260     // Others  
261     .ALU_result_out(MemAddr),  
262     .RtData_out(MemWriteData),  
263     .RdAddr_out(RdAddr_mem_out)  
264 );  
265  
266
```



```

270 DM_Data_Memory(
271 // Outputs
272 .MemReadData(MemReadData),
273 // Inputs
274 .MemAddr(MemAddr),
275 .MemWriteData(MemWriteData),
276 .MemWrite(MemWrite_mem_out),
277 .MemRead(MemRead_mem_out),
278 .clk(clk)
279 );
280
281 MEM_WB_Memory_WriteBack(
282 // Inputs
283 .clk(clk),
284 // WB
285 .RegWrite_in(RegWrite_mem_out),
286 .Mem2Reg_in(MemtoReg_mem_out),
287 // Others
288 .MemAddr_in(MemAddr),
289 .MemReadData_in(MemReadData),
290 .RdAddr_in(RdAddr_mem_out),
291 //Outputs
292 //WB
293 .RegWrite_out(RegWrite_wb_out),
294 .Mem2Reg_out(MemtoReg_wb_out),
295 // Others
296 .MemAddr_out(MemAddr_wb_out),
297 .MemReadData_out(MemReadData_wb_out),
298 .RdAddr_out(RdAddr_wb_out)
299 );
300
301 Mux32b_B32(
302 .Src1(MemAddr_wb_out),
303 .Src2(MemReadData_wb_out),
304 .result(MUX32b_result),
305 .choose(MemtoReg_wb_out)
306 );
307
308 endmodule

```

```

C:\Users\jieho> Documents > coding > HDL > Computer_Organization > PA3 > Part 1 > testbench > Answer > RF.out
You, 3 days ago | 1 author (You)
1 00000000
2 00000001
3 00000002
4 77777777
5 7f7f7f7f
6 f7f7f7f7
7 7fffffff
8 80000000
9 ffff0000
10 0000ffff
11 00000011
12 00000023
13 00000017
14 00000090
15 00000100
16 00000250
17 00000300
18 00000037
19 00000064
20 00000030
21 00000034
22 00000079
23 00000024
24 00040000
25 00000000
26 00000000
27 00000000
28 00000000
29 00000000
30 00000000
31 ffffffff
32 ffffffff

```

```

testbench > RF_part1.out
You, 2 days ago | 1
1 00000000
2 00000001
3 00000002
4 77777777
5 7f7f7f7f
6 f7f7f7f7
7 7fffffff
8 80000000
9 ffff0000
10 0000ffff
11 00000011
12 00000023
13 00000017
14 00000090
15 00000100
16 00000250
17 00000300
18 00000037
19 00000064
20 00000030
21 00000034
22 00000079
23 00000024
24 00040000
25 00000000
26 00000000
27 00000000
28 00000000
29 00000000
30 00000000
31 ffffffff
32 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。

```

testbench > Answer > DM.out
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

```

```

testbench > Answer > RF.out
1 00000000
2 00000001
3 00000002
4 77777777
5 7f7f7f7f
6 f7f7f7f7
7 7fffffff
8 80000000
9 ffff0000
10 0000ffff
11 00000011
12 00000023
13 00000016
14 00000090
15 00000100
16 00000250
17 00000300
18 00000037
19 00000064
20 00000030
21 00000000
22 00000000
23 00000000
24 00000000
25 0000001b
26 00000008
27 12345678
28 00000000
29 00000000
30 00000000
31 ffffffff
32 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}.

```
29 `define INSTR_MEM_SIZE 128 // Bytes
30 /*
31  * Declaration of Instruction Memory for this project.
32  * CAUTION: DONT MODIFY THE NAME.
33  */
34 module IM(
35     // Outputs
36     output reg [31:0] Instr,
37     // Inputs
38     input [31:0] InstrAddr
39 );
40 /*
41  * Declaration of instruction memory.
42  * CAUTION: DONT MODIFY THE NAME AND SIZE.
43  */
44 reg [7:0] InstrMem[0:`INSTR_MEM_SIZE - 1];
45
46 always@(InstrAddr)
47 begin
48     Instr[31:0] = {InstrMem[InstrAddr], InstrMem[InstrAddr+1], InstrMem[InstrAddr+2], InstrMem[InstrAddr+3]};
49 end
50
51 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的地址去抓值去輸出。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。

```
29 `define REG_MEM_SIZE 32 // Words
30 /*
31  * Declaration of Register File for this project.
32  * CAUTION: DONT MODIFY THE NAME.
33  */
34 module RF(
35     // Outputs
36     output [31:0] RsData,
37     output [31:0] RtData,
38     // Inputs
39     input RegWrite,
40     input clk,
41     input [4:0] RsAddr,
42     input [4:0] RtAddr,
43     input [4:0] RdAddr,
44     input [31:0] RdData
45 );
46
47 /*
48  * Declaration of inner register.
49  * CAUTION: DONT MODIFY THE NAME AND SIZE.
50  */
51 reg [31:0] R[0:`REG_MEM_SIZE - 1];
52
53 assign RsData = R[RsAddr];
54 assign RtData = R[RtAddr];
55
56 always@(posedge clk)
57 begin
58     if(RegWrite == 1)
59     begin
60         R[RdAddr] = RdData;
61     end
62     else
63     begin
64         R[RdAddr] = R[RdAddr];
65     end
66 end
67
68
69
70 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。）

```
1 module Adder
2 (
3     input  [31:0] AddrIn,
4     output [31:0] AddrOut
5 );
6
7     assign AddrOut = AddrIn + 32'd 4;
8
9 endmodule
10
```

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。

```
1 `define addu 6'b 001011
2 `define subu 6'b 001101
3 `define AND 6'b 010010
4 `define sll 6'b 100110
5
6 module ALU(
7     input  [31:0] Src1,
8     input  [31:0] Src2,
9     input  [4:0]  Shamt,
10    input  [5:0]  Funct,
11    output reg [31:0] result
12 );
13
14 always@(Funct or Shamt or Src1 or Src2)
15 begin
16     case (Funct)
17         `addu : result = Src1 + Src2;
18         `subu : result = Src1 - Src2;
19         `AND  : result = Src1 & Src2;
20         `sll  : result = Src1 << Shamt;
21         default: result = result; // if no operation
22     endcase
23 end
24
25
26 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。

```
ALU_Control.v
You, 3 days ago | 1 author (You)
1  `define addu 6'b 001011
2  `define subu 6'b 001101
3  `define AND 6'b 010010
4  `define sll 6'b 100110
5  `define input_addu 6'b 001011
6  `define input_subu 6'b 001101
7  `define input_and 6'b 010010
8  `define input_sll 6'b 100110
9  `define R_type 2'b10
10 // I-type sub
11 `define I_type_sub 2'b00
12 // I-type add
13 `define I_type_add 2'b01
14 module ALU_Control(
15     input [5:0] funct,
16     input [1:0] ALUOp,
17     output reg [5:0] Funct
18 );
19 always@(funct or ALUOp)
20 begin
21     case(ALUOp)
22     `R_type:
23     begin
24         case(funct)
25         `input_addu : Funct = `addu;
26         `input_subu : Funct = `subu;
27         `input_and : Funct = `AND;
28         `input_sll : Funct = `sll;
29         default: Funct = 0;
30         endcase
31     end
32     `I_type_sub:
33     begin
34         Funct = `subu;
35     end
36     `I_type_add:
37     begin
38         Funct = `addu;
39     end
40     default:;
41     endcase
42 end
43 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。

```
29 `define DATA_MEM_SIZE 128 // Bytes
30
31 /*
32  * Declaration of Data Memory for this project.
33  * CAUTION: DONT MODIFY THE NAME.
34  */
35 module DM(
36     // Outputs
37     output [31:0] MemReadData,
38     // Inputs
39     input [31:0] MemAddr,
40     input [31:0] MemWriteData,
41     input MemWrite,
42     input MemRead,
43     input clk
44 );
45
46 /*
47  * Declaration of data memory.
48  * CAUTION: DONT MODIFY THE NAME AND SIZE.
49  */
50 reg [7:0] DataMem[0:`DATA_MEM_SIZE - 1];
51
52 assign MemReadData = MemRead? {DataMem[MemAddr],DataMem[MemAddr+1],DataMem[MemAddr+2],DataMem[MemAddr+3]}:32'b0;
53 // You, a day ago + PART3 controller still working
54 always@(posedge clk)
55 begin
56     if(MemWrite == 1)
57     begin
58         {DataMem[MemAddr],DataMem[MemAddr+1],DataMem[MemAddr+2],DataMem[MemAddr+3]} = MemWriteData;
59     end
60     else;
61 end
62
63 endmodule
64
```

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)
1 // I-type sub
2 `define I_type_sub 2'b00
3 // I-type add
4 `define I_type_add 2'b01
5 module Control(
6     input [5:0] OpCode,
7     output reg RegWrite,
8     output reg [1:0] ALUOp,
9     output reg RegDst,
10    output reg ALUSrc,
11    output reg MemWrite,
12    output reg MemRead,
13    output reg MemtoReg
14 );
15

```

```

16    end
17    6'd 16: // sw
18    begin
19        RegWrite = 1'b 0;
20        ALUOp = `I_type_add;
21        RegDst = 1'b x; // I format.
22        ALUSrc = 1;
23        MemWrite = 1;
24        MemRead = 0;
25        MemtoReg = 1'b x; // I format.
26    end
27    6'd 17: // lw
28    begin
29        RegWrite = 1'b 1;
30        ALUOp = `I_type_add;
31        RegDst = 0; // I format.
32        ALUSrc = 1;
33        MemWrite = 0;
34        MemRead = 1;
35        MemtoReg = 1;
36    end
37    default:
38    begin
39        MemWrite = 0;
40        RegWrite = 0;
41    end
42 endcase
43 end
44 endmodule

```

```

16 always@(OpCode)
17 begin
18     case (OpCode)
19     6'd 4:
20     begin
21         RegWrite = 1'b 1; // I format.
22         ALUOp = 2'b 10;
23         RegDst = 1; // R format.
24         ALUSrc = 0;
25         MemWrite = 0;
26         MemRead = 0;
27         MemtoReg = 0;
28     end
29     // I format.
30     6'd 12: // addiu
31     begin
32         RegWrite = 1'b 1;
33         ALUOp = `I_type_add;
34         RegDst = 0; // I format.
35         ALUSrc = 1;
36         MemWrite = 0;
37         MemRead = 0;
38         MemtoReg = 0;
39     end
40     6'd 13: // subi
41     begin
42         RegWrite = 1'b 1;
43         ALUOp = `I_type_sub;
44         RegDst = 0; // I format.
45         ALUSrc = 1;
46         MemWrite = 0;
47         MemRead = 0;
48         MemtoReg = 0;
49     end
50     6'd 16: // sw
51     begin
52         RegWrite = 1'b 0;
53         ALUOp = `I_type_add;
54         RegDst = 1'b x; // I format.
55         ALUSrc = 1;
56         MemWrite = 1;
57         MemRead = 0;
58         MemtoReg = 1'b x; // I format.
59     end

```

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)
1 module Mux5b(
2   input [4:0]Src1, //0
3   input [4:0]Src2, //1
4   input choose,
5   output [4:0]result
6 );
7 assign result = choose? Src2:Src1;
8
9 endmodule

Mux32b.v
You, 3 days ago | 1 author (You)
1 module Mux32b(
2   input [31:0]Src1, // 0
3   input [31:0]Src2, // 1
4   input choose,
5   output [31:0]result
6 );
7
8 assign result = choose? Src2:Src1;
9
10 endmodule
You, 3 days ago • part
```

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

```
1 module ID_EX(
2
3   // Write Back.
4   input RegWrite_in, // WB
5   input Mem2Reg_in, //WB
6   output reg RegWrite_out, // WB
7   output reg Mem2Reg_out, // WB
8   // Memory
9   input MemRead_in,
10  input MemWrite_in,
11  output reg MemWrite_out,
12  output reg MemRead_out,
13  // EX.
14  input [1:0] ALUOp_in, // EX
15  input RegDst_in,
16  input ALU_Src_in,
17
18  output reg [1:0] ALUOp_out, //EX
19  output reg RegDst_out,
20  output reg ALU_Src_out,
21  // Others.
22  input clk,
23  input [4:0] RdAddr_in,
24  input [4:0] RtAddr_in,
25  input [31:0] RsData_in,
26  input [31:0] RtData_in,
27  input [31:0] immediate_in,
28
29  output reg [31:0] immediate_out,
30  output reg [31:0] RsData_out,
31  output reg [31:0] RtData_out,
32  output reg [4:0] RdAddr_out,
33  output reg [4:0] RtAddr_out
34
35
```

```
35
36 // Temp register part.
37
38 // Write Back.
39 reg RegWrite_reg;
40 reg Mem2Reg_reg;
41 // Memory
42 reg MemRead_reg;
43 reg MemWrite_reg;
44 // Execution.
45 reg [1:0] ALUOp_reg; // EX
46 reg RegDst_reg;
47 reg ALU_Src_reg;
48 // Others.
49 reg [5:0] RdAddr_reg;
50 reg [4:0] RtAddr_reg;
51 reg [31:0] RsData_reg;
52 reg [31:0] RtData_reg;
53 reg [31:0] immediate_reg;
```



```

56 always@(posedge clk or negedge clk)
57 begin
58     if(clk == 1) // put them in to the reg
59     begin
60         // Write Back.
61         Mem2Reg_reg = Mem2Reg_in;
62         RegWrite_reg = RegWrite_in;
63         // Memory.
64         MemRead_reg = MemRead_in;
65         MemWrite_reg = MemWrite_in;
66         // Execution
67         ALUOp_reg = ALUOp_in;
68         RegDst_reg = RegDst_in;
69         ALU_Src_reg = ALU_Src_in;
70         // Others.
71         RdAddr_reg = RdAddr_in;
72         RtAddr_reg = RtAddr_in;
73         RsData_reg = RsData_in;
74         RtData_reg = RtData_in;
75         immediate_reg = immediate_in;
76     end
77 else
78     begin
79         // Write Back.
80         RegWrite_out = RegWrite_reg;
81         Mem2Reg_out = Mem2Reg_reg;
82         // Memory.
83         MemRead_out = MemRead_reg;
84         MemWrite_out = MemWrite_reg;
85         // Execution
86         ALUOp_out = ALUOp_reg;
87         RegDst_out = RegDst_reg;
88         ALU_Src_out = ALU_Src_reg;
89         // Others.
90         RdAddr_out = RdAddr_reg;
91         RtAddr_out = RtAddr_reg;
92         RsData_out = RsData_reg;
93         RtData_out = RtData_reg;
94         immediate_out = immediate_reg;
95     end
96 end
97
98 endmodule

```

```

1 module IF_ID(
2     input [31:0] Instr,
3     input clk,
4     output reg [31:0] InstrOut
5 );
6     reg [31:0] Instr_reg;
7
8     always@(posedge clk or negedge clk)
9     begin
10         if(clk == 1) //when posedge
11             Instr_reg = Instr;
12         else // when negedge clk, out
13             InstrOut = Instr_reg;
14         end
15 endmodule

```

```

1 module EX_MEM(
2     // Write Back.
3     input RegWrite_in, // WB
4     input Mem2Reg_in, //WB
5     output reg RegWrite_out, // WB
6     output reg Mem2Reg_out, //WB
7     // Memory
8     input MemRead_in,
9     input MemWrite_in,
10    output reg MemWrite_out,
11    output reg MemRead_out,
12    // Others.
13    input clk,
14    input [31:0] ALU_result_in,
15    input [4:0] RdAddr_in,
16    input [31:0] RtData_in,
17    output reg [31:0] RtData_out,
18    output reg [4:0] RdAddr_out,
19    output reg [31:0] ALU_result_out
20 );

```

```

37 always@(posedge clk or negedge clk)
38 begin
39     if(clk == 1) // put them in to the reg
40     begin
41         // Write Back.
42         RegWrite_reg = RegWrite_in;
43         Mem2Reg_reg = Mem2Reg_in;
44         // Memory.
45         RegWrite_reg = RegWrite_in;
46         Mem2Reg_reg = Mem2Reg_in;
47         MemRead_reg = MemRead_in;
48         MemWrite_reg = MemWrite_in;
49         // Others
50         RdAddr_reg = RdAddr_in;
51         ALU_result_reg = ALU_result_in;
52         RtData_reg = RtData_in;
53     end
54 else
55     begin
56         // Write Back.
57         RegWrite_out = RegWrite_reg;
58         Mem2Reg_out = Mem2Reg_reg;
59         // Memory
60         RegWrite_out = RegWrite_reg;
61         Mem2Reg_out = Mem2Reg_reg;
62         MemRead_out = MemRead_reg;
63         MemWrite_out = MemWrite_reg;
64         // Others
65         RdAddr_out = RdAddr_reg;
66         ALU_result_out = ALU_result_reg;
67         RtData_out = RtData_reg;
68     end
69 end
70 endmodule

```

```

22 // Temp register part.
23
24 // Write Back.
25 reg RegWrite_reg;
26 reg Mem2Reg_reg;
27 // Memory
28 reg MemRead_reg;
29 reg MemWrite_reg;
30 // Others.
31 reg [5:0] RdAddr_reg;
32 reg [31:0] RtData_reg;
33 reg [31:0] ALU_result_reg;

```



```

1  module MEM_WB(
2      // Write Back.
3      input RegWrite_in, // WB
4      input Mem2Reg_in, //WB
5      output reg RegWrite_out, // WB
6      output reg Mem2Reg_out, // WB
7      // Others.
8      input clk,
9      input [31:0] MemAddr_in,
10     input [4:0] RdAddr_in,
11     input [31:0] MemReadData_in,
12     output reg [31:0] MemReadData_out,
13     output reg [4:0] RdAddr_out,
14     output reg [31:0] MemAddr_out
15 );
16 // Temp register part.
17 // Write Back.
18 reg RegWrite_reg;
19 reg Mem2Reg_reg;
20 // Others.
21 reg [5:0] RdAddr_reg;
22 reg [31:0] MemAddr_reg;
23 reg [31:0] MemReadData_reg;
24

```

```

25 always@(posedge clk or negedge clk)
26 begin
27     if(clk == 1) // put them in to the reg
28     begin
29         // Write Back.
30         RegWrite_reg = RegWrite_in;
31         Mem2Reg_reg = Mem2Reg_in;
32     end
33     // Others
34     RdAddr_reg = RdAddr_in;
35     MemAddr_reg = MemAddr_in;
36     MemReadData_reg = MemReadData_in;
37 end
38 else
39 begin
40     // Write Back.
41     RegWrite_out = RegWrite_reg;
42     Mem2Reg_out = Mem2Reg_reg;
43 end
44 // Others
45 RdAddr_out = RdAddr_reg;
46 MemAddr_out = MemAddr_reg;
47 MemReadData_out = MemReadData_reg;
48 end
49 end
50
51
52 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。

```

29 module FinalCPU(
30     // Outputs
31     output wire      PCWrite,
32     output wire [31:0] AddrOut,
33     // Inputs
34     input  wire [31:0] AddrIn,
35     input  wire      clk
36 );
37
38 // HazardDetectionUnit
39 wire Stall;
40 wire IF_ID_Write;
41
42 // Forwarding Unit
43 wire [1:0] ForwardA;
44 wire [1:0] ForwardB;
45 wire [31:0] MUX3to1A_result;
46 wire [31:0] MUX3to1B_result;
47 // Instruction Memory
48 wire [31:0] Instr;
49 wire [31:0] Instr_out;
50 // ID/EX.
51
52 // WB
53
54 wire RegWrite;
55 wire RegWrite_in; // MUXSTALL
56 wire RegWrite_out;
57 wire MemtoReg;
58 wire MemtoReg_in;
59 wire MemtoReg_out;
60

```

```

59 wire MemtoReg_out;
60 // memory.
61 wire MemWrite;
62 wire MemWrite_in;
63 wire MemWrite_out;
64
65 wire MemRead;
66 wire MemRead_in;
67 wire MemRead_out;
68
69 // EX.
70 wire [1:0] ALUOp;
71 wire [1:0] ALUOp_in;
72 wire [1:0] ALUOp_out;
73
74 wire RegDst;
75 wire RegDst_in;
76 wire RegDst_out;
77
78 wire ALUSrc;
79 wire ALUSrc_in;
80 wire ALUSrc_out;

```

```

82 // Others.
83 wire [31:0] RsData;
84 wire [31:0] RsData_out;
85 wire [31:0] RtData;
86 wire [31:0] RtData_out;
87 wire [31:0] Sign_Extend; //
88 wire [31:0] immediate_out;
89 wire [4:0] RdAddr_out;
90 wire [4:0] RtAddr_out;
91 wire [4:0] RsAddr_out;
92 // EX/MEM
93
94 //WB.
95 wire RegWrite_mem_out;
96 wire MemtoReg_mem_out;
97 //Memory
98 wire MemWrite_mem_out;
99 wire MemRead_mem_out;
100 // Others.
101 wire [31:0] ALU_result;
102 wire [31:0] MemAddr;
103
104 wire [31:0] MemWriteData; //
105
106 wire [4:0] RdAddr_mem_out;
107

```

```

107 // MEM/WB
108 // MEM/WB
109
110 // WB.
111 wire RegWrite_wb_out;
112 wire MemtoReg_wb_out;
113 // Others
114 wire [31:0] MemAddr_wb_out;
115 wire [4:0] RdAddr_wb_out;
116 wire [31:0] MemReadData;
117 wire [31:0] MemReadData_wb_out;
118
119
120
121 // ALU controller
122 wire [5:0] Funct;
123 // MUX
124 wire [4:0] MUX5_result;
125
126 wire [31:0] MUX32A_result;
127 wire [31:0] MUX32B_result;
128

```

```

132     IM Instr_Memory(
133         // Outputs
134         .Instr(Instr),
135         // Inputs
136         .InstrAddr(AddrIn)
137     );
138
139     Adder adder(
140         // Outputs
141         .AddrOut(AddrOut),
142         // Inputs
143         .AddrIn(AddrIn)
144     );
145
146     IF_ID Fetch_Decode(
147         .Instr(Instr),
148         .InstrOut(Instr_out),
149         .IF_ID_Write(IF_ID_Write),
150         .clk(clk)
151     );

```

```

153     HazardDetectionUnit HDU(
154         // Inputs.
155         .ID_EX_MemRead(MemRead_out),
156         .ID_EX_RegisterRt(RtAddr_out),
157         .IF_ID_RegisterRs(Instr_out[25:21]),
158         .IF_ID_RegisterRt(Instr_out[20:16]),
159         // Outputs
160         .Stall(Stall),
161         .PCWrite(PCWrite),
162         .IF_ID_Write(IF_ID_Write)
163     );
164
165     Control_controller(
166         // Inputs
167         .OpCode(Instr_out[31:26]),
168         // Outputs
169         .RegWrite(RegWrite),
170         .RegDst(RegDst),
171         .ALUSrc(ALUSrc),
172         .MemWrite(MemWrite),
173         .MemRead(MemRead),
174         .MemtoReg(MemtoReg),
175         .ALUOp(ALUOp)
176     );

```

```

182     RF Register_File(
183         // Outputs
184         .RsData(RsData), // TO ID/EX.
185         .RtData(RtData), // TO ID/EX.
186         // Inputs
187         .clk(clk),
188         .RegWrite(RegWrite_wb_out), // From MEM/WB.
189         .RsAddr(Instr_out[25:21]),
190         .RtAddr(Instr_out[20:16]),
191         .RdAddr(RdAddr_wb_out),
192         .RdData(MUX32B_result) // From MEM/WB.
193     );
194     assign Sign_Extend[31:0] = Instr_out[15]?{16'hFFFF, Instr_out[15:0]}:{16'h0000, Instr_out[15:0]};

```

```

195
196     MuxStall MS(
197         // inputs.
198         .RegDst_in(RegDst),
199         .MemRead_in(MemRead),
200         .MemtoReg_in(MemtoReg),
201         .ALUOp_in(ALUOp),
202         .MemWrite_in(MemWrite),
203         .ALUSrc_in(ALUSrc),
204         .RegWrite_in(RegWrite),
205         .Stall_choose(Stall),
206         // Outputs
207         .RegDst_out(RegDst_in),
208         .MemRead_out(MemRead_in),
209         .MemtoReg_out(MemtoReg_in),
210         .ALUOp_out(ALUOp_in),
211         .MemWrite_out(MemWrite_in),
212         .ALUSrc_out(ALUSrc_in),
213         .RegWrite_out(RegWrite_in)
214     );

```

```

216     ID_EX Decode_Execute(
217         // Inputs
218         .clk(clk),
219         //WB
220         .RegWrite_in(RegWrite_in),
221         .Mem2Reg_in(MemtoReg_in),
222         // Memory
223         .MemRead_in(MemRead_in),
224         .MemWrite_in(MemWrite_in),
225         // EX
226         .ALUOp_in(ALUOp_in),
227         .RegDst_in(RegDst_in),
228         .ALU_Src_in(ALUSrc_in),
229         // Others.
230         .RsData_in(RsData),
231         .RtData_in(RtData),
232         .immediate_in(Sign_Extend),
233         .RdAddr_in(Instr_out[15:11]),
234         .RtAddr_in(Instr_out[20:16]),
235         .RsAddr_in(Instr_out[25:21]),
236         // Outputs
237         //WB
238         .RegWrite_out(RegWrite_out),
239         .Mem2Reg_out(MemtoReg_out),
240         // Memory
241         .MemRead_out(MemRead_out),
242         .MemWrite_out(MemWrite_out),
243         // EX
244         .ALUOp_out(ALUOp_out),
245         .RegDst_out(RegDst_out),
246         .ALU_Src_out(ALUSrc_out),
247         // Others
248         .RsData_out(RsData_out),
249         .RtData_out(RtData_out),
250         .immediate_out(immediate_out),
251         .RdAddr_out(RdAddr_out),
252         .RtAddr_out(RtAddr_out),
253         .RsAddr_out(RsAddr_out)
254     );

```

```

256     MUX3to1 A(
257         .Src1(RsData_out),
258         .Src2(MUX32B_result),
259         .Src3(MemAddr),
260         .choose(ForwardA),
261         .result(MUX3to1A_result)
262     );
263
264     MUX3to1 B(
265         .Src1(RtData_out),
266         .Src2(MUX32B_result),
267         .Src3(MemAddr),
268         .choose(ForwardB),
269         .result(MUX3to1B_result)
270     );
271
272     Mux32b_A32(
273         .Src1(MUX3to1B_result),
274         .Src2(immediate_out),
275         .result(MUX32A_result),
276         .choose(ALUSrc_out)
277     );
278
279     ALU alu(
280         // Inputs
281         .Src1(MUX3to1A_result),
282         .Src2(MUX32A_result),
283         .Shamt(immediate_out[10:6]),
284         .Funct(Funct),
285         // Outputs
286         .result(ALU_result)
287     );

```

```

288
289     ALU_Control ALU_controller(
290         // Inputs
291         .funct(immediate_out[5:0]),
292         .ALUOp(ALUOp_out),
293         // Outputs
294         .Funct(Funct)
295     );
296
297     Mux5b mux5b(
298         .Src1(RtAddr_out),
299         .Src2(RdAddr_out),
300         .choose(RegDst_out),
301         .result(MUX5_result)
302     );
303
304     ForwardingUnit FU(
305         .EX_MEM_RegWrite(RegWrite_mem_out),
306         .EX_MEM_RegisterRd(RdAddr_mem_out),
307         .ID_EX_RegisterRs(RsAddr_out),
308         .ID_EX_RegisterRt(RtAddr_out),
309         .MEM_WB_RegWrite(RegWrite_wb_out),
310         .MEM_WB_RegisterRd(RdAddr_wb_out),
311         .ForwardA(ForwardA[1:0]),
312         .ForwardB(ForwardB[1:0])
313     );

```

```

315     EX_MEM Execute_Memory(
316         // Inputs
317         .clk(clk),
318         // WB
319         .RegWrite_in(RegWrite_out),
320         .Mem2Reg_in(MemtoReg_out),
321         // MEM
322         .MemRead_in(MemRead_out),
323         .MemWrite_in(MemWrite_out),
324         // Others
325         .ALU_result_in(ALU_result),
326         .RtData_in(MUX3to1B_result),
327         .RdAddr_in(MUX5_result),
328         //Outputs
329
330         // WB
331         .RegWrite_out(RegWrite_mem_out),
332         .Mem2Reg_out(MemtoReg_mem_out),
333         // Memory
334         .MemRead_out(MemRead_mem_out),
335         .MemWrite_out(MemWrite_mem_out),
336         // Others
337         .ALU_result_out(MemAddr),
338         .RtData_out(MemWriteData),
339         .RdAddr_out(RdAddr_mem_out)
340     );

```

```

344     DM Data_Memory(
345         // Outputs
346         .MemReadData(MemReadData),
347         // Inputs
348         .MemAddr(MemAddr),
349         .MemWriteData(MemWriteData),
350         .MemWrite(MemWrite_mem_out),
351         .MemRead(MemRead_mem_out),
352         .clk(clk)
353     );
354
355     MEM_WB_Memory_WriteBack(
356         // Inputs
357         .clk(clk),
358         // WB
359         .RegWrite_in(RegWrite_mem_out),
360         .Mem2Reg_in(MemtoReg_mem_out),
361         // Others.
362         .MemAddr_in(MemAddr),
363         .MemReadData_in(MemReadData),
364         .RdAddr_in(RdAddr_mem_out),
365
366         //Outputs
367
368         //WB
369         .RegWrite_out(RegWrite_wb_out),
370         .Mem2Reg_out(MemtoReg_wb_out),
371         // Others
372         .MemAddr_out(MemAddr_wb_out),
373         .MemReadData_out(MemReadData_wb_out),
374         .RdAddr_out(RdAddr_wb_out)
375     );
376
377     Mux32b_B32(
378         .Src1(MemAddr_wb_out),
379         .Src2(MemReadData_wb_out),
380         .result(MUX32B_result),
381         .choose(MemtoReg_wb_out)
382     );
383
384
385 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。

```
testbench > DM.out
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 12
32 34
33 56
34 78
35 ff
36 ff
37 ff
38 ff

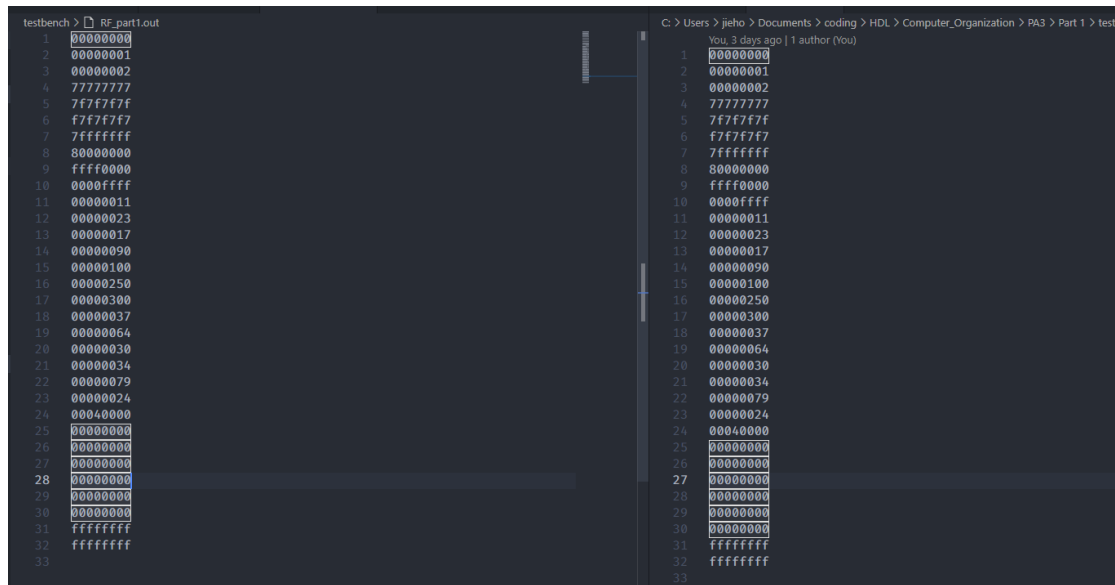
testbench > Answer > DM.out
10 ff
11 ff
12 ff
13 ff
14 ff
15 ff
16 ff
17 ff
18 ff
19 ff
20 ff
21 ff
22 ff
23 ff
24 ff
25 ff
26 ff
27 12
28 34
29 56
30 78
31 12
32 34
33 56
34 78
35 ff
36 ff
37 ff
```

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。

```
Part 3 > testbench > RF.out
You, 17 hours ago | 1 author (You)
1 00000000
2 00000001
3 00000002
4 77777777
5 7f7f7f7f
6 f7f7f7f7
7 7fffffff
8 80000000
9 ffff0000
10 0000ffff
11 00000011
12 00000023
13 00000017
14 00000090
15 00000100
16 00000250
17 00000300
18 00000037
19 00000064
20 00000000
21 00000000
22 00000000
23 00000000
24 00000000
25 00000000
26 00000000
27 00000000
28 00000000
29 00000000
30 00000000
31 ffffffff
32 ffffffff
33

Part 3 > testbench > Answer > RF.out
You, 5 days ago | 1 author (You)
1 00000000
2 00000001
3 00000002
4 77777777
5 7f7f7f7f
6 f7f7f7f7
7 7fffffff
8 80000000
9 ffff0000
10 0000ffff
11 00000011
12 00000023
13 00000017
14 00000090
15 00000100
16 00000250
17 00000300
18 00000037
19 00000064
20 00000000
21 00000000
22 00000000
23 00000000
24 00000000
25 00000000
26 00000000
27 00000000
28 00000000
29 00000000
30 00000000
31 ffffffff
32 ffffffff
33
```

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，



The image shows two side-by-side terminal windows. The left window has a title bar 'testbench > RF_part1.out' and displays a list of 33 lines of hexadecimal data. The right window has a title bar 'C:\Users\jheho\Documents\coding\HDL\Computer_Organization\PA3\Part 1> testbench > RF_part1.out' and displays the same 33 lines of hexadecimal data. The data is as follows:

Line	Hex Value
1	00000000
2	00000001
3	00000002
4	77777777
5	7f7f7f7f
6	f7f7f7f7
7	7fffffff
8	80000000
9	ffff0000
10	0000ffff
11	00000011
12	00000023
13	00000017
14	00000090
15	00000100
16	00000250
17	00000300
18	00000037
19	00000064
20	00000030
21	00000034
22	00000079
23	00000024
24	00040000
25	00000000
26	00000000
27	00000000
28	00000000
29	00000000
30	00000000
31	ffffffff
32	ffffffff
33	

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我程式碼的方式去抓{0, 1, 2, 3}.

```
29 `define INSTR_MEM_SIZE 128 // Bytes
30 /*
31  * Declaration of Instruction Memory for this project.
32  * CAUTION: DONT MODIFY THE NAME.
33  */
34 module IM(
35     // Outputs
36     output reg [31:0] Instr,
37     // Inputs
38     input [31:0] InstrAddr
39 );
40
41 /*
42  * Declaration of instruction memory.
43  * CAUTION: DONT MODIFY THE NAME AND SIZE.
44  */
45 reg [7:0] InstrMem[0:`INSTR_MEM_SIZE - 1];
46
47 always@(InstrAddr)
48 begin
49     Instr[31:0] = {InstrMem[InstrAddr], InstrMem[InstrAddr+1], InstrMem[InstrAddr+2], InstrMem[InstrAddr+3]};
50 end
51
52 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的地址去抓值去輸出。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。

```
29 `define REG_MEM_SIZE 32 // Words
30
31 /*
32  * Declaration of Register File for this project.
33  * CAUTION: DONT MODIFY THE NAME.
34  */
35 module RF(
36     // Outputs
37     output [31:0] RsData,
38     output [31:0] RtData,
39     // Inputs
40     input RegWrite,
41     input clk,
42     input [4:0] RsAddr,
43     input [4:0] RtAddr,
44     input [4:0] RdAddr,
45     input [31:0] RdData
46 );
47
48 /*
49  * Declaration of inner register.
50  * CAUTION: DONT MODIFY THE NAME AND SIZE.
51  */
52 reg [31:0] R[0:`REG_MEM_SIZE - 1];
53
54 assign RsData = R[RsAddr];
55 assign RtData = R[RtAddr];
56
57 always@(posedge clk)
58 begin
59     if(RegWrite == 1)
60     begin
61         R[RdAddr] = RdData;
62     end
63     else
64     begin
65         R[RdAddr] = R[RdAddr];
66     end
67 end
68
69 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。


```

29 `define DATA_MEM_SIZE 128 // Bytes
30
31 /*
32  * Declaration of Data Memory for this project.
33  * CAUTION: DONT MODIFY THE NAME.
34  */
35 module DM(
36     // Outputs
37     output [31:0] MemReadData,
38     // Inputs
39     input [31:0] MemAddr,
40     input [31:0] MemWriteData,
41     input MemWrite,
42     input MemRead,
43     input clk
44 );
45
46 /*
47  * Declaration of data memory.
48  * CAUTION: DONT MODIFY THE NAME AND SIZE.
49  */
50 reg [7:0] DataMem[0:`DATA_MEM_SIZE - 1];
51
52 assign MemReadData = MemRead? {DataMem[MemAddr],DataMem[MemAddr+1],DataMem[MemAddr+2],DataMem[MemAddr+3]}:32'b0;
53 // You, a day ago * PART3 controller still working
54 always@(posedge clk)
55 begin
56     if(MemWrite == 1)
57     begin
58         {DataMem[MemAddr],DataMem[MemAddr+1],DataMem[MemAddr+2],DataMem[MemAddr+3]} = MemWriteData;
59     end
60     else;
61 end
62
63 endmodule
64

```

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)
1  module Adder
2  (
3      input  [31:0] AddrIn,
4      output [31:0] AddrOut
5  );
6
7      assign AddrOut = AddrIn + 32'd 4;
8
9  endmodule
10

```


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。

```
1 `define addu 6'b 001011
2 `define subu 6'b 001101
3 `define AND 6'b 010010
4 `define sll 6'b 100110
5
6 module ALU(
7     input [31:0] Src1,
8     input [31:0] Src2,
9     input [4:0] Shamt,
10    input [5:0] Funct,
11    output reg [31:0] result
12);
13
14 always@(Funct or Shamt or Src1 or Src2)
15 begin
16     case (Funct)
17         `addu : result = Src1 + Src2;
18         `subu : result = Src1 - Src2;
19         `AND : result = Src1 & Src2;
20         `sll : result = Src1 << Shamt;
21         default: result = result; // if
22     endcase
23 end
24
25
26 endmodule
```

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)
1 `define addu 6'b 001011
2 `define subu 6'b 001101
3 `define AND 6'b 010010
4 `define sll 6'b 100110
5
6 `define input_addu 6'b 001011
7 `define input_subu 6'b 001101
8 `define input_and 6'b 010010
9 `define input_sll 6'b 100110
10 `define R_type 2'b10
11
12 module ALU_Control(
13     input [5:0] funct,
14     input [1:0] ALUOp,
15     output reg [5:0] Funct
16 );
17
18 always@(funct or ALUOp)
19 begin
20     case(ALUOp)
21         `R_type:
22         begin
23             case(funct)
24                 `input_addu : Funct = `addu;
25                 `input_subu : Funct = `subu;
26                 `input_and : Funct = `AND;
27                 `input_sll : Funct = `sll;
28                 default: Funct = 0;
29             endcase
30         end
31         default: Funct = 0;
32     endcase
33 end
34 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。

```
1 // I-type sub
2 `define I_type_sub 2'b00
3 // I-type add
4 `define I_type_add 2'b01
5 module Control(
6     input [5:0] OpCode,
7     output reg RegWrite,
8     output reg [1:0] ALUOp,
9     output reg RegDst,
10    output reg ALUSrc,
11    output reg MemWrite,
12    output reg MemRead,
13    output reg MemtoReg
14 );
```

```
59 end
60 6'd 17: // lw
61 begin
62     RegWrite = 1'b 1;
63     ALUOp = `I_type_add;
64     RegDst = 0; // I format
65     ALUSrc = 1;
66     MemWrite = 0;
67     MemRead = 1;
68     MemtoReg = 1;
69 end
70 default:
71 begin
72     MemWrite = 0;
73     RegWrite = 0;
74 end
75 endcase
76 end
77 endmodule
78
79 endmodule
```

```
16 always@(OpCode)
17 begin
18     case (OpCode)
19         6'd 4:
20         begin
21             RegWrite = 1'b 1; //
22             ALUOp = 2'b 10;
23             RegDst = 1; // R format
24             ALUSrc = 0;
25             MemWrite = 0;
26             MemRead = 0;
27             MemtoReg = 0;
28         end
29         // I format.
30         6'd 12: // addiu
31         begin
32             RegWrite = 1'b 1;
33             ALUOp = `I_type_add;
34             RegDst = 0; // I format
35             ALUSrc = 1;
36             MemWrite = 0;
37             MemRead = 0;
38             MemtoReg = 0;
39         end
40         6'd 13: // subiu
41         begin
42             RegWrite = 1'b 1;
43             ALUOp = `I_type_sub;
44             RegDst = 0; // I format
45             ALUSrc = 1;
46             MemWrite = 0;
47             MemRead = 0;
48             MemtoReg = 0;
49         end
50         6'd 16: // sw
51         begin
52             RegWrite = 1'b 0;
53             ALUOp = `I_type_add;
54             RegDst = 0; // I format
55             ALUSrc = 1;
56             MemWrite = 1;
57             MemRead = 0;
58             MemtoReg = 0; // Since
59         end
```

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)
1 module Mux5b(
2   input [4:0]Src1, //0
3   input [4:0]Src2, //1
4   input choose,
5   output [4:0]result
6 );
7 assign result = choose? Src2:Src1;
8
9 endmodule

Mux32b.v
You, 3 days ago | 1 author (You)
1 module Mux32b(
2   input [31:0]Src1, // 0
3   input [31:0]Src2, // 1
4   input choose,
5   output [31:0]result
6 );
7
8 assign result = choose? Src2:Src1;
9
10 endmodule
```

```
MuxStall.v
You, a day ago | 1 author (You)
1 module MuxStall(
2   input RegDst_in,
3   input MemRead_in,
4   input MemtoReg_in,
5   input [1:0] ALUOp_in,
6   input MemWrite_in,
7   input ALUSrc_in,
8   input RegWrite_in,
9   input Stall_choose,
10  output reg RegDst_out,
11  output reg MemRead_out,
12  output reg MemtoReg_out,
13  output reg [1:0] ALUOp_out,
14  output reg MemWrite_out,
15  output reg ALUSrc_out,
16  output reg RegWrite_out
17 );
18
19 always@(*) begin
20   if (Stall_choose==0)
21   begin
22     RegDst_out =RegDst_in;
23     MemRead_out =MemRead_in;
24     MemtoReg_out =MemtoReg_in;
25     ALUOp_out =ALUOp_in;
26     MemWrite_out =MemWrite_in;
27     ALUSrc_out =ALUSrc_in;
28     RegWrite_out =RegWrite_in;
29   end
30   if (Stall_choose==1)
31   begin
32     RegDst_out =0;
33     MemRead_out =0;
34     MemtoReg_out =0;
35     ALUOp_out =2'd0;
36     MemWrite_out =0;
37     ALUSrc_out =0;
38     RegWrite_out =0;
39   end
40 end
41 endmodule
```

```

You, 2 days ago | 1 author (You)
1 module MUX3to1(
2   input [31:0] Src1,
3   input [31:0] Src2,
4   input [31:0] Src3,
5   input [1:0] choose,
6   output reg [31:0]result
7 );
8
9 always@(*)
10 begin
11   case(chOOSE)
12     2'b00:result = Src1;
13     2'b01:result = Src2;
14     2'b10:result = Src3;
15     default;;
16   endcase
17 end
18
19 endmodule
```

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

```

1 module IF_ID(
2     input [31:0] Instr,
3     input clk,
4     input IF_ID_Write,
5     output reg [31:0] InstrOut
6 );
7     reg [31:0] Instr_reg;
8
9     always@(posedge clk or negedge clk)
10        begin
11            if(clk == 1 && IF_ID_Write == 1)
12                Instr_reg = Instr;
13            else // when negedge clk, output
14                InstrOut = Instr_reg;
15        end
16 endmodule

```

```

1 module ID_EX(
2     // Write Back.
3     input RegWrite_in, // WB
4     input Mem2Reg_in, //WB
5
6     output reg RegWrite_out, // WB
7     output reg Mem2Reg_out, // WB
8     // Memory
9     input MemRead_in,
10    input MemWrite_in,
11    output reg MemWrite_out,
12    output reg MemRead_out,
13    // EX.
14    input [1:0] ALUOp_in, // EX
15    input RegDst_in,
16    input ALU_Src_in,
17
18    output reg [1:0] ALUOp_out, //EX
19    output reg RegDst_out,
20    output reg ALU_Src_out,
21    // Others
22    input clk,
23    input [4:0] RdAddr_in,
24    input [4:0] RtAddr_in,
25    input [4:0] RsAddr_in,
26    input [31:0] RsData_in,
27    input [31:0] RtData_in,
28    input [31:0] immediate_in,
29    output reg [31:0] immediate_out,
30    output reg [31:0] RsData_out,
31    output reg [31:0] RtData_out,
32    output reg [4:0] RdAddr_out,
33    output reg [4:0] RtAddr_out,
34    output reg [4:0] RsAddr_out
35 );

```

```

38 // Write Back.
39 reg RegWrite_reg;
40 reg Mem2Reg_reg;
41 // Memory
42 reg MemRead_reg;
43 reg MemWrite_reg;
44 // Execution.
45 reg [1:0] ALUOp_reg; // EX
46 reg RegDst_reg;
47 reg ALU_Src_reg;
48 // Others.
49 reg [4:0] RdAddr_reg;
50 reg [4:0] RtAddr_reg;
51 reg [4:0] RsAddr_reg;
52 reg [31:0] RsData_reg;
53 reg [31:0] RtData_reg;
54 reg [31:0] immediate_reg;
55
56

```

```

57 always@(posedge clk or negedge clk)
58     begin
59         if(clk == 1) // put them in to the reg
60             begin
61                 // Write Back.
62                 Mem2Reg_reg = Mem2Reg_in;
63                 RegWrite_reg = RegWrite_in;
64                 // Memory.
65                 MemRead_reg = MemRead_in;
66                 MemWrite_reg = MemWrite_in;
67                 // Execution
68                 ALUOp_reg = ALUOp_in;
69                 RegDst_reg = RegDst_in;
70                 ALU_Src_reg = ALU_Src_in;
71                 // Others.
72                 RdAddr_reg = RdAddr_in;
73                 RtAddr_reg = RtAddr_in;
74                 RsAddr_reg = RsAddr_in;
75
76                 RsData_reg = RsData_in;
77                 RtData_reg = RtData_in;
78                 immediate_reg = immediate_in;
79             end
80         else
81             begin
82                 // Write Back.
83                 RegWrite_out = RegWrite_reg;
84                 Mem2Reg_out = Mem2Reg_reg;
85                 // Memory.
86                 MemRead_out = MemRead_reg;
87                 MemWrite_out = MemWrite_reg;
88                 // Execution
89                 ALUOp_out = ALUOp_reg;
90                 RegDst_out = RegDst_reg;
91                 ALU_Src_out = ALU_Src_reg;
92                 // Others.
93                 RdAddr_out = RdAddr_reg;
94                 RtAddr_out = RtAddr_reg;
95                 RsAddr_out = RsAddr_reg;
96
97                 RsData_out = RsData_reg;
98                 RtData_out = RtData_reg;
99                 immediate_out = immediate_reg;
100             end
101         end

```

```

1 module EX_MEM(
2     // Write Back.
3     input RegWrite_in, // WB
4     input Mem2Reg_in, //WB
5
6     output reg RegWrite_out, // WB
7     output reg Mem2Reg_out, // WB
8
9     // Memory
10    input MemRead_in,
11    input MemWrite_in,
12    output reg MemWrite_out,
13    output reg MemRead_out,
14
15    // Others.
16    input clk,
17    input [31:0] ALU_result_in,
18    input [4:0] RdAddr_in,
19    input [4:0] RtAddr_in,
20    input [31:0] RtData_in,
21    output reg [31:0] RtData_out,
22    output reg [4:0] RdAddr_out,
23    output reg [31:0] ALU_result_out
24 );
25
26 // Temp register part.
27
28 // Write Back.
29 reg RegWrite_reg;
30 reg Mem2Reg_reg;
31 // Memory
32 reg MemRead_reg;
33 reg MemWrite_reg;
34 // Others.
35 reg [5:0] RdAddr_reg;
36 reg [31:0] RtData_reg;
37 reg [31:0] ALU_result_reg;
38
39

```

```

42 always@(posedge clk or negedge clk)
43     begin
44         if(clk == 1) // put them in to the reg
45             begin
46                 // Write Back.
47                 RegWrite_reg = RegWrite_in;
48                 Mem2Reg_reg = Mem2Reg_in;
49                 // Memory.
50                 RegWrite_reg = RegWrite_in;
51                 Mem2Reg_reg = Mem2Reg_in;
52                 MemRead_reg = MemRead_in;
53                 MemWrite_reg = MemWrite_in;
54                 // Others
55                 RdAddr_reg = RdAddr_in;
56                 ALU_result_reg = ALU_result_in;
57                 RtData_reg = RtData_in;
58             end
59         else
60             begin
61                 // Write Back.
62                 RegWrite_out = RegWrite_reg;
63                 Mem2Reg_out = Mem2Reg_reg;
64                 // Memory
65                 RegWrite_out = RegWrite_reg;
66                 Mem2Reg_out = Mem2Reg_reg;
67                 MemRead_out = MemRead_reg;
68                 MemWrite_out = MemWrite_reg;
69                 // Others
70                 RdAddr_out = RdAddr_reg;
71                 ALU_result_out = ALU_result_reg;
72                 RtData_out = RtData_reg;
73             end
74         end
75     end
76
77 endmodule

```

```

1  module MEM_WB(
2      // Write Back.
3      input RegWrite_in, // WB
4      input Mem2Reg_in, // WB
5
6      output reg RegWrite_out, // WB
7      output reg Mem2Reg_out, // WB
8
9      // Others.
10
11      input clk,
12      input [31:0] MemAddr_in,
13      input [4:0] RdAddr_in,
14      input [31:0] MemReadData_in,
15
16
17      output reg [31:0] MemReadData_out,
18      output reg [4:0] RdAddr_out,
19      output reg [31:0] MemAddr_out
20 );
21
22 // Temp register part.
23
24 // Write Back.
25 reg RegWrite_reg;
26 reg Mem2Reg_reg;
27
28 // Others.
29 reg [5:0] RdAddr_reg;
30 reg [31:0] MemAddr_reg;
31 reg [31:0] MemReadData_reg;
32
33

```

```

35 always@(posedge clk or negedge clk)
36 begin
37     if(clk == 1) // put them in to the reg
38     begin
39         // Write Back.
40         RegWrite_reg = RegWrite_in;
41         Mem2Reg_reg = Mem2Reg_in;
42
43         // Others
44         RdAddr_reg = RdAddr_in;
45         MemAddr_reg = MemAddr_in;
46         MemReadData_reg = MemReadData_in;
47     end
48 else
49     begin
50         // Write Back.
51         RegWrite_out = RegWrite_reg;
52         Mem2Reg_out = Mem2Reg_reg;
53
54         // Others
55         RdAddr_out = RdAddr_reg;
56         MemAddr_out = MemAddr_reg;
57         MemReadData_out = MemReadData_reg;
58     end
59 end
60
61
62 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。