中国科学技术大学计算机学院
《计算机组成原理实验》报告

实验题目：流水线 CPU 设计

学生姓名：吴毅龙

学生学号：PB19111749

完成日期：2021/6/1

计算机实验教学中心制

2020 年 09 月

【实验题目】

流水线 CPU 设计

【实验目的】

- 理解 CPU 的功能、结构和工作原理
- 掌握流水线 CPU 的设计和调试方法，特别是流水线中数据相关和控制相关的处理
- 熟练掌握数据通路和控制器的设计和描述方法

【实验环境】

FPGAOL: fpgaol.ustc.edu.cn

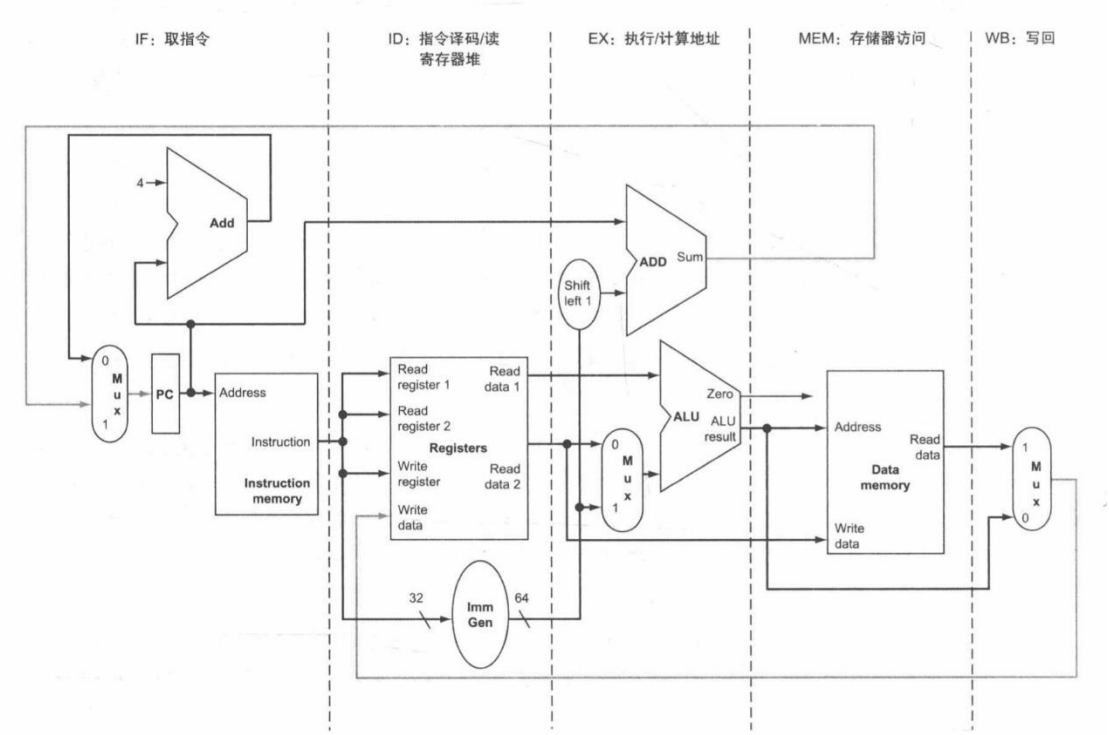
Vivado

【实验内容】

1. 修改 Lab4 寄存器堆模块，使其满足写优先(Write First)，即在对同一寄存器读写时，写数据可立即从读数据输出
2. 设计无数据和控制相关处理的流水线 CPU
3. 设计仅有数据相关处理的流水线 CPU
4. 设计完整的有数据和控制相关处理的流水线 CPU
    - 对 CPU 进行功能仿真
    - 将 CPU 和 PDU 连接并下载至 FPGA 中测试

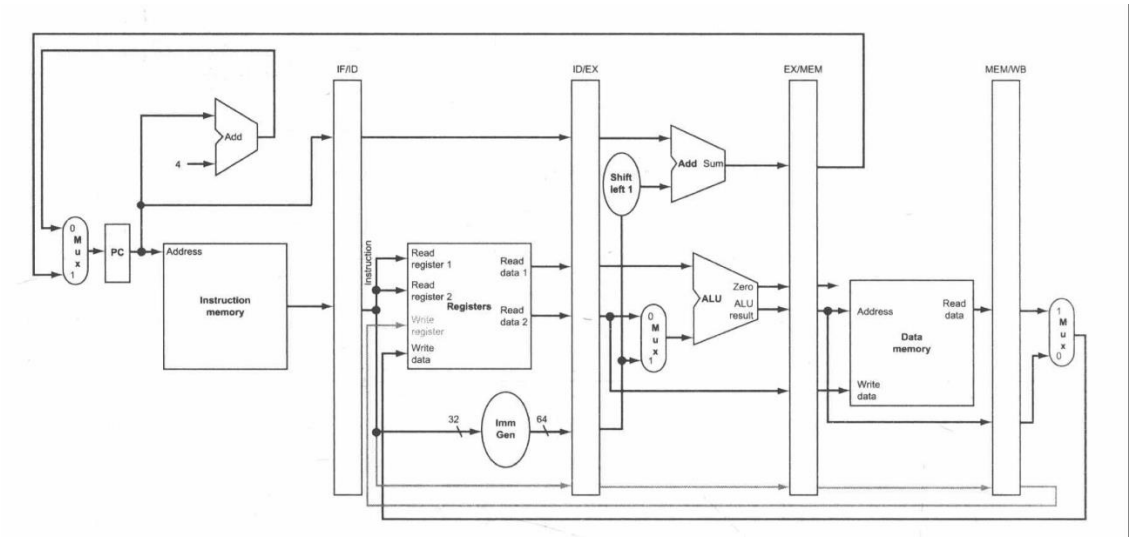
【实验练习】

流水线分级：本次实验实现的流水线 CPU 包含有五级流水线，意味着在任意单时钟周期里最多执行五条指令，相应的，我们需要把数据通路划分为五个部分，将五个部分用相应的指令执行阶段来命名；



段间寄存器引入：我们可以通过引入寄存器保存数据的方式，是的部分数据通路可以在指令执行的过程中被共享。举例来说，指令存储器只在指令的五个阶段的一个阶段被使用，而在其他四个阶段中允许被其他指令共享。为了保留在其他四个阶段的指令的值，必须把从指令存储器中读取出来的数据保存在寄存器中。类似的理由适用于每个流水线阶段，所以我们必须将寄存器置于上图每个阶段的分割线上。下图显示了流水线寄存器的分布，所有指令都会在每一个时钟周期里从一个流水线寄存器前进到下一个寄存器中。

我们需要在加载指令的流水线寄存器中保留目标寄存器的编号。就像存储指令为了 MEM 阶段的使用而将寄存器的值从 ID/EX 中传递到 EX/MEM 流水线寄存器中那样，加载指令为了 WB 阶段的使用而将寄存器编号从 ID/EX 通过 EX/MEM 传递到 MEM/WB 流水线寄存器。寄存器编号在 WB 阶段被使用，指定了要写入的寄存器。




```
1. module IF_ID(
2.     input clk, IF_en, IF_Flush,
3.     input [31:0] PC, PCadd4, instruction,
4.     output reg [31:0] pcd, pcin, ir
5.     );
6.     initial begin
7.         ir = 0;
8.     end
9.     always @(posedge clk)
10.    begin
11.        if (IF_Flush)    //清除 IF 阶段的指令
12.        begin
13.            ir <= 0;
14.            pcd <= 0;
15.            pcin <= 0;
16.        end
17.        else if(IF_en)
18.        begin
19.            ir <= instruction;
20.            pcd <= PC;
21.            pcin <= PCadd4;
22.        end
23.        else
24.        begin
25.            ir <= ir;
26.            pcd <= pcd;
27.            pcin <= pcin;
28.        end
29.    end
30.endmodule
```

```
1. module ID_EX(
2.     input clk, ID_en, ID_Flush,
3.     input [31:0] control,
4.     input [31:0] pcin, pcd,
5.     input [31:0] rs1, rs2,
6.     input [31:0] ir, Imm,
7.     output reg[31:0] ctrl,
8.     output reg[31:0] pce, pcin2,
9.     output reg[31:0] a, b,
10.    output reg[31:0] imm,
11.    output reg[4:0] rd, ra1, ra2
12.    );
13.    initial begin
14.        ctrl=0;
15.        a=0;
16.        b=0;
17.    end
18.    always @(posedge clk)
19.    begin
20.        if(ID_Flush)
21.        begin
22.            ctrl  <= 0;
23.            pcin2 <= 0;
24.            pce   <= 0;
25.            a     <= 0;
26.            b     <= 0;
27.            imm   <= 0;
28.            rd    <= 0;
29.            ra1   <= 0;
30.            ra2   <= 0;
31.        end
32.        else if(ID_en)
33.        begin
34.            ctrl  <= control;
35.            pcin2 <= pcin;
36.            pce   <= pcd;
37.            a     <= rs1;
38.            b     <= rs2;
39.            imm   <= Imm;
40.            rd    <= ir[11:7];
41.            ra1   <= ir[19:15];
```

```
            ra2   <= ir[24:20];
        end
        else
        begin
            ctrl  <= ctrl ;
            pcin2 <= pcin2;
            pce   <= pce  ;
            a     <= a    ;
            b     <= b    ;
            imm   <= imm  ;
            rd    <= rd   ;
            ra1   <= ra1  ;
            ra2   <= ra2  ;
        end
    end
endmodule
```

```
module EX_MEM(
    input clk,
    input [31:0] ALUresult, op2, ctrl, pcin2,
    input [4:0] rd,
    output reg [31:0] ctrlm, y, bm, pcin3,
    output reg [4:0] rdm
    );
    always @(posedge clk) begin
        y <= ALUresult;
        bm <= op2;
        ctrlm <= ctrl;
        pcin3 <= pcin2;
        rdm <=rd;
    end
endmodule

```

```
module MEM_WB(
    input clk,
    input [4:0] rdm,
    input [31:0] y, ReadData, ctrlm, pcin3,
    output reg [31:0] yw, mdr, ctrlw, pcin4,
```

```
6.      output reg [4:0] rdw
7.      );
8.      always @(posedge clk) begin
9.          rdw <= rdm;
10.         yw <= y;
11.         mdr <= ReadData;
12.         ctrlw <= ctrlm;
13.         pcin4 <= pcin3;
14.     end
15.endmodule
```

流水线控制：由于控制线从 EX 阶段开始，可以在指令译码阶段为之后的阶段创建控制信号。传递这些控制信号最简单的方式就是扩展流水线寄存器以包含这些控制信息。随着指令沿着流水线向下流动，这些控制信号被用于适当的流水线阶段。在本次实验中，控制信号被规定为如下格式

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
fstall	dstall	dflush	eflush	0	0	a_fwd		0	0	b_fwd		0	rf_wr	wb_sel	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	m_rd	m_wr	0	0	jal	br	0	0	a_sel	b_sel	alu_op			

```
1. module Control(
2.      input [6:0] opcode,
3.      input [1:0] ForwardA,ForwardB,
4.      output reg[31:0] control
5.      );
6.      localparam ALUresult = 2'b01, DataMem=2'b10, PCadd4=2'b11,IDI
   E=2'b00;
7.      always @(*) begin
8.          control[25:24] = ForwardA;
9.          control[21:20] = ForwardB;
10.         control[31:26] = 0;
11.         control[23:22] = 0;
12.         control[19] = 0;
13.         control[15:14] = 0;
14.         control[11:10] = 0;
15.         control[7:6] = 0;
16.         case(opcode)
```

```
            7'b0110011: begin   //add
                control[3:0] = 4'b0010;     //ALU_op
                control[5:4] = 2'b11;       //a_sel b_sel
                control[9:8] = 2'b00;       //jal branch
                control[13:12] = 2'b00;//memory_read memory_write
                control[18] = 1;       //register file write
                control[17:16] = ALUresult;    //wb_sel
            end
            7'b0010011: begin   //addi
                control[3:0] = 4'b0010;        //ALU_op
                control[5:4] = 2'b10;          //a_sel b_sel
                control[9:8] = 2'b00;          //jal branch
                control[13:12] = 2'b00;//memory_read memory_write
                control[18] = 1;       //register file write
                control[17:16] = ALUresult;    //wb_sel
            end
            7'b0000011: begin   //lw
                control[3:0] = 4'b0010;        //ALU_op
                control[5:4] = 2'b10;          //a_sel b_sel
                control[9:8] = 2'b00;          //jal branch
                control[13:12] = 2'b10;//memory_read memory_write
                control[18] = 1;       //register file write
                control[17:16] = DataMem;      //wb_sel
            end
            7'b0100011: begin   //sw
                control[3:0] = 4'b0010;        //ALU_op
                control[5:4] = 2'b10;          //a_sel b_sel
                control[9:8] = 2'b00;          //jal branch
                control[13:12] = 2'b01;//memory_read memory_write
                control[18] = 0;       //register file write
                control[17:16] = IDIE;         //wb_sel
            end
            7'b1100011: begin   //beq
                control[3:0] = 4'b0110;        //ALU_op
                control[5:4] = 2'b11;          //a_sel b_sel
                control[9:8] = 2'b01;          //jal branch
                control[13:12] = 2'b00;//memory_read memory_write
                control[18] = 0;       //register file write
                control[17:16] = IDIE;         //wb_sel
            end
            7'b1101111: begin   //jal
                control[3:0] = 4'b0010;        //ALU_op
                control[5:4] = 2'b00;          //a_sel b_sel
                control[9:8] = 2'b10;          //jal branch
```

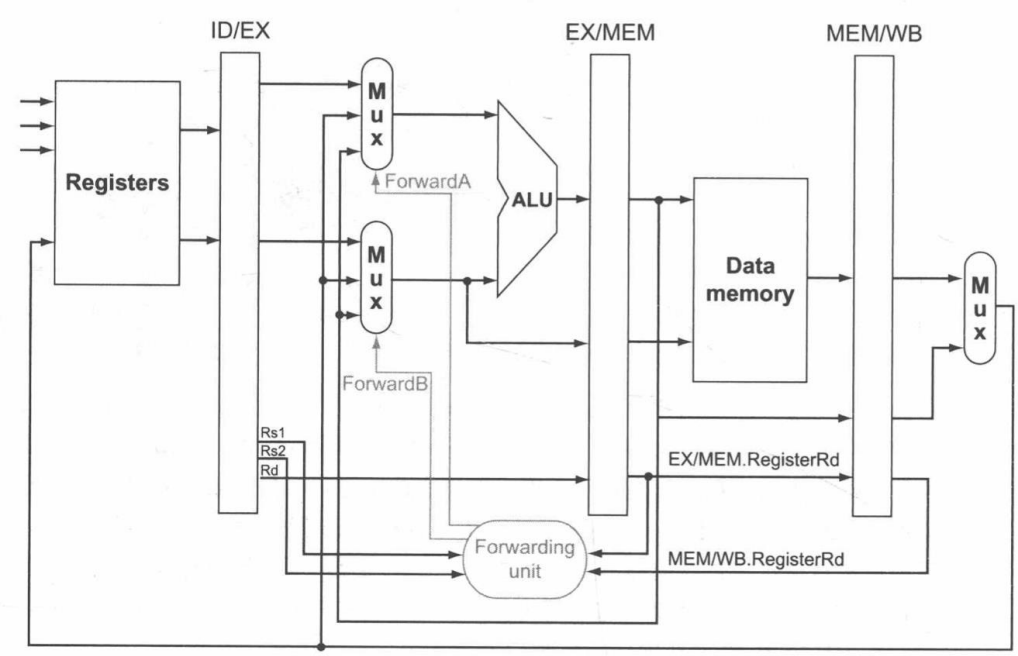
```
61.                 control[13:12] = 2'b00;//memory_read memory_write
62.                 control[18] = 1;       //register file write
63.                 control[17:16] = PCadd4;       //wb_sel
64.             end
65.             default: begin
66.                 control[3:0] = 4'b0000;        //ALU_op
67.                 control[5:4] = 2'b00;          //a_sel b_sel
68.                 control[9:8] = 2'b00;          //jal branch
69.                 control[13:12] = 2'b00;//memory_read memory_write
70.                 control[18] = 0;       //register file write
71.                 control[17:16] = IDIE;         //wb_sel
72.             end
73.         endcase
74.     end
75.endmodule
76.
```

数据冒险（前递）：当一个指令试图在 EX 阶段使用的寄存器是一个较早的指令在 WB 阶段要写入的寄存器时，我们需要将该数据作为 ALU 的输入，所以就可以得到两对冒险的条件：

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs1
1b. EX/MEM.RegisterRd = ID/EX.RegisterRs2
2a. MEM/WB.RegisterRd = ID/EX.RegisterRs1
2b. MEM/WB.RegisterRd = ID/EX. RegisterRs2

因为不是所有的指令都会写回寄存器，因此还需要添加必要的判断标准，一种简单的解决方案是检查 RegWrite 信号是否有效：检查流水线寄存器在 EX 和 MEM 阶段的 WB 控制字段以确定 RegWrite 是否有效。于是就可以给出检测冒险的条件了。

多选器控制	源	解释
ForwardA = 00	ID/EX	ALU的第一个操作数来自寄存器堆
ForwardA = 10	EX/MEM	ALU的第一个操作数来自上一个ALU计算结果的前递
ForwardA = 01	MEM/WB	ALU的第一个操作数来自数据存储器或者更早的ALU计算结果的前递
ForwardB = 00	ID/EX	ALU的第二个操作数来自寄存器堆
ForwardB = 10	EX/MEM	ALU的第二个操作数来自上一个ALU计算结果的前递
ForwardB = 01	MEM/WB	ALU的第二个操作数来自数据存储器或者更早的ALU计算结果的前递



1. EX 冒险

```
if  (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 10

if  (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 10
```

2. MEM 冒险

```
if  (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 01
if  (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01
```

```
1. module Forwarding(
2.     input [4:0] ra1, ra2, rdm, rdw, rdf,
3.     input rf_wr_m, rf_wr_w, rf_wr_f,
4.     output reg [1:0] ForwardA, ForwardB
5.     );
6.     always @(*)
7.     begin
8.         if(rf_wr_m && ra1 == rdm ) ForwardA = 2'b01;
9.         else if(rf_wr_w && ra1 == rdw) ForwardA = 2'b10;
10.        else if(rf_wr_f && ra1 == rdf) ForwardA = 2'b11;
11.        else ForwardA = 2'b00;
12.    end
```

```
13.     always @(*)
14.     begin
15.         if(rf_wr_m && ra2 == rdm) ForwardB = 2'b01;
16.         else if(rf_wr_w && ra2 == rdw) ForwardB = 2'b10;
17.         else if(rf_wr_f && ra2 == rdf) ForwardB = 2'b11;
18.         else ForwardB = 2'b00;
19.     end
20.endmodule
```

数据冒险（停顿）：冒险控制单元的控制逻辑满足以下条件：

```
if  (ID/EX.MemRead and
    ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
      (ID/EX.RegisterRd = IF/ID.RegisterRs2)))
      stall the pipeline
```

使流水线停顿只需要禁止 PC 寄存器和 IF/ID 流水线寄存器的改变就可以阻止这两条指令的执行。如果这些寄存器被保护，在 IF 阶段的指令就会继续使用相同的 PC 值来取指令，同时 ID 段的寄存器会继续使用 IF/ID 流水线寄存器中相同的字段来读寄存器。另外需要解除 EX、MEM 和 WB 阶段的控制信号（将其设置为 0）就可以产生一个“没有任何操作”的指令，也就是空指令。

控制冒险：一种提升分支阻塞效率的方法是预测分支条件不发生并持续执行顺序指令流。一旦条件分支发生，已经被读取和译码的指令就将被丢弃，流水线继续从分支目标处开始执行。想要丢弃指令，只需要将初始控制值变为 0 即可，这与指令停顿以解决加载-使用的数据冒险类似。不同的是，丢弃指令的同时也需要改变当分支指令到达 MEM 阶段时 IF、ID 和 EX 阶段的三条指令；而在加载-使用数据冒险的停顿中，只需要将 ID 段的控制信号变为 0 并且精该阶段的指令从流水线中过滤出去即可。

```
module Hazard(
    input m_rd_idex, m_wr_ifid,
    input [4:0]rd, ra1, ra2,
    input zero, beq_d, jal_d, beq,jal,
    input a_sel, b_sel,
    output reg PC_en, IF_en, ID_en, IF_Flush, ID_Flush
    );
    always @(*) begin
        if(jal || (zero && beq))
        begin
            PC_en = 1;
            IF_Flush = 1;
            IF_en = 0;
            ID_en = 0;
            ID_Flush = 1;
        end
        else if( m_rd_idex && ( (a_sel && rd == ra1) || ((b_sel | | m_wr_ifid ) && rd == ra2) ) )
        begin
            PC_en = 0;
            IF_Flush = 0;
            IF_en = 0;
            ID_en = 0;
            ID_Flush = 1;
        end
        else
        begin
            PC_en = 1;
            IF_Flush = 0;
            IF_en = 1;
            ID_en = 1;
            ID_Flush = 0;
        end
    end
endmodule

```

最终连线构成 CPU

```
module CPU(
    input clk, rst,

    //IO_BUS
    output [7:0] io_addr,
```

```
6.      output [31:0] io_dout,
7.      output io_we,
8.      input [31:0] io_din,
9.
10.     //Debug_BUS
11.     input [7:0] m_rf_addr,
12.     output [31:0] rf_data,
13.     output [31:0] m_data,
14.
15.     output [31:0] pcin, pc, pcd, pce,
16.     output [31:0] ir, imm, mdr,
17.     output [31:0] a, b, y, bm, yw,
18.     output [4:0]  rd, rdm, rdw,
19.     output [31:0] ctrl, ctrlm, ctrlw
20.     );
21.
22.     reg [31:0] PC;
23.     assign pc = PC;
24.     wire [31:0] PCN;
25.     wire PC_en;
26.     always @(posedge clk, posedge rst) begin
27.         if(rst)PC <= 0;
28.         else if(PC_en) PC <= PCN;
29.         else PC <= PC;
30.     end
31.
32.     wire [31:0] instruction;
33.     instruction_mem Instruction_mem(
34.         .a(PC[9:2]),
35.         .spo(instruction)
36.     );
37.
38.     wire [31:0] PCadd4;
39.     assign PCadd4 = PC + 4;
40.
41.     wire [31:0] PCjmp;
42.     wire isPCjmp;
43.     assign PCN = isPCjmp ? PCjmp : PCadd4;
44.
45.     wire IF_en, IF_Flush;
46.     IFID IF_ID(
47.         .clk(clk),
48.         .PC(PC),
49.         .PCadd4(PCadd4),
```

```
50.         .instruction(instruction),
51.         .pcin(pcin),
52.         .pcd(pcd),
53.         .ir(ir),
54.         .IF_en(IF_en),
55.         .IF_Flush(IF_Flush)
56.     );
57.
58.     wire [31:0]rs1, rs2;
59.     reg [31:0] WriteData;
60.     Registers RF(
61.         .clk(clk),
62.         .we(ctrlw[18]),                 //register file write
63.         .wa(rdw),
64.         .ra0(ir[19:15]),
65.         .ra1(ir[24:20]),
66.         .ra2(m_rf_addr),
67.         .rd0(rs1),
68.         .rd1(rs2),
69.         .rd2(rf_data),
70.         .wd(WriteData)
71.     );
72.
73.     wire [31:0]Imm;
74.     ImmGen ImmGen(ir,Imm);
75.
76.     wire [31:0] control,pcin2;
77.     wire [4:0] ra1,ra2;
78.
79.     wire ID_en,ID_Flush;
80.     IDEX ID_EX(
81.         .clk(clk),
82.         .control(control),.ctrl(ctrl),
83.         .rs1(rs1),.a(a),
84.         .rs2(rs2),.b(b),
85.         .Imm(Imm),.imm(imm),
86.         .ir(ir),.rd(rd),
87.         .pcd(pcd),.pce(pce),
88.         .pcin(pcin),.pcin2(pcin2),
89.         .ra1(ra1),.ra2(ra2),
90.         .ID_en(ID_en),
91.         .ID_Flush(ID_Flush)
92.     );
93.
```

```
94.     wire [1:0] ForwardB,ForwardB;
95.     Control Controler(
96.         .opcode(ir[6:0]),
97.         .ForwardA(ForwardA),
98.         .ForwardB(ForwardB),
99.         .control(control)
100.     );
101.
102.     reg [31:0] op1;
103.     wire [31:0] op2;
104.     reg [31:0] op2_temp;
105.     wire [31:0] ALUresult;
106.     wire zero;
107.     wire [31:0] LastWriteData;
108.
109.     always @(*) begin
110.         case (control[25:24])        //ForwardA
111.             2'b00:op1 = a;
112.             2'b01:op1 = y;
113.             2'b10:op1 = WriteData;
114.             default:op1 = LastWriteData;
115.         endcase
116.     end
117.     always @(*) begin
118.         case (control[21:20])        //ForwardB
119.             2'b00:op2_temp = b;
120.             2'b01:op2_temp = y;
121.             2'b10:op2_temp = WriteData;
122.             default:op2_temp = LastWriteData;
123.         endcase
124.     end
125.
126.     ALU #(32)alu(
127.         .a(op1),
128.         .b(op2),
129.         .s(ctrl[3:0]),
130.         .y(ALUresult),
131.         .zf(zero)
132.     );
133.
134.     wire [31:0] pcin3;
135.     EXMEM EX_MEM(
136.         .clk(clk),
137.         .ALUresult(ALUresult),.y(y),
```

```
138.         .op2(op2_temp),.bm(bm),
139.         .pcin2(pcin2),.pcin3(pcin3),
140.         .ctrl(ctrl),.ctrlm(ctrlm),
141.         .rd(rd),.rdm(rdm)
142.     );
143.
144.     wire [31:0] ReadData_temp;//map device
145.     wire DM_we;
146.     wire [31:0] ReadData;
147.     data_mem Data_mem(
148.         .a(y[9:2]),
149.         .d(bm),
150.         .dpra(m_rf_addr),
151.         .clk(clk),
152.         .we(DM_we),                    //ctrlm[12]  memory_write
153.         .spo(ReadData_temp),
154.         .dpo(m_data)
155.     );
156.
157.     //memory map device
158.     assign DM_we = ctrlm[12] & (~y[10]);
159.     assign io_addr = y[7:0];
160.     assign io_dout = bm;
161.     assign io_we = ctrlm[12];
162.     assign ReadData = y[10] ? io_din : ReadData_temp;
163.
164.     wire [31:0] pcin4;
165.     MEMWB MEM_WB(
166.         .clk(clk),
167.         .rdm(rdm),.rdw(rdw),
168.         .y(y),.yw(yw),
169.         .pcin3(pcin3),.pcin4(pcin4),
170.         .ctrlm(ctrlm),.ctrlw(ctrlw),
171.         .ReadData(ReadData),.mdr(mdr)
172.     );
173.
174.     wire [4:0] rdf;
175.     wire rf_wr_f;
176.
177.     Forwarding forwarding_unit(
178.         .ForwardA(ForwardA),
179.         .ForwardB(ForwardB),
180.         .ra1(ra1),
181.         .ra2(ra2),
```

```
182.         .rdm(rdm),
183.         .rdw(rdw),
184.         .rdf(rdf),
185.         .rf_wr_m(ctrlm[18]),
186.         .rf_wr_w(ctrlw[18]),
187.         .rf_wr_f(rf_wr_f)
188.     );
189.
190.     always @(*) begin
191.         case(ctrlw[17:16])          //wb_sel
192.             2'b01:WriteData = yw;     //ALUresult
193.             2'b10:WriteData = mdr;    //DataMem
194.             2'b11:WriteData = pcin4;  //PCadd4
195.             default:WriteData = 0;
196.         endcase
197.     end
198.
199.
200.     assign op2 = ctrl[4] ? op2_temp : imm;  //ctrl[4] b_sel
201.
202.     assign PCjmp = (imm << 1) + pce;
203.     assign isPCjmp = ctrl[9] | (zero & ctrl[8]);
204.
205.
206.     WBIF WB_IF(
207.         .clk(clk),
208.         .WriteData(WriteData),
209.         .LastWriteData(LastWriteData),
210.         .rdw(rdw),
211.         .rdf(rdf),
212.         .rf_wr_w(ctrlw[18]),
213.         .rf_wr_f(rf_wr_f)
214.     );
215.
216.     Hazard Hazard_Unit(
217.         .m_rd_idex(ctrl[13]),
218.         .m_wr_ifid(control[12]),
219.         .zero(zero),
220.         .beq_d(control[8]),
221.         .jal_d(control[9]),
222.         .beq(ctrl[8]),
223.         .jal(ctrl[9]),
224.         .a_sel(control[5]),
225.         .b_sel(control[4]),
```

```
226.         .rd(rd),
227.         .ra1(ir[19:15]),
228.         .ra2(ir[24:20]),
229.         .PC_en(PC_en),
230.         .IF_en(IF_en),
231.         .IF_Flush(IF_Flush),
232.         .ID_en(ID_en),
233.         .ID_Flush(ID_Flush)
234.     );
235.endmodule
```

【总结与思考】

通过本次实验，我利用 FPGAOL 平台进行实验，并使用 IP 核，实现了五级流水线的 CPU 设计。本次实验难度不大，根据实验说明的指导就可以完成实验操作，实验题目的难度层层递进，有基础操作的考核，也有所学知识的综合，难易结合，既有复习又有思考，让所学在实践中得以运用，加深了我对逻辑电路知识的理解。希望今后实验可以保持本次实验中详细实验指导描述的优点，辅助完成每项试验内容。