

Lab7 - mips-cpu 设计

金泽文 PB15111604

实验目的：

设计更为全面，更为优秀的 mips cpu，以加深对 Computer Organization and Design 的理解。

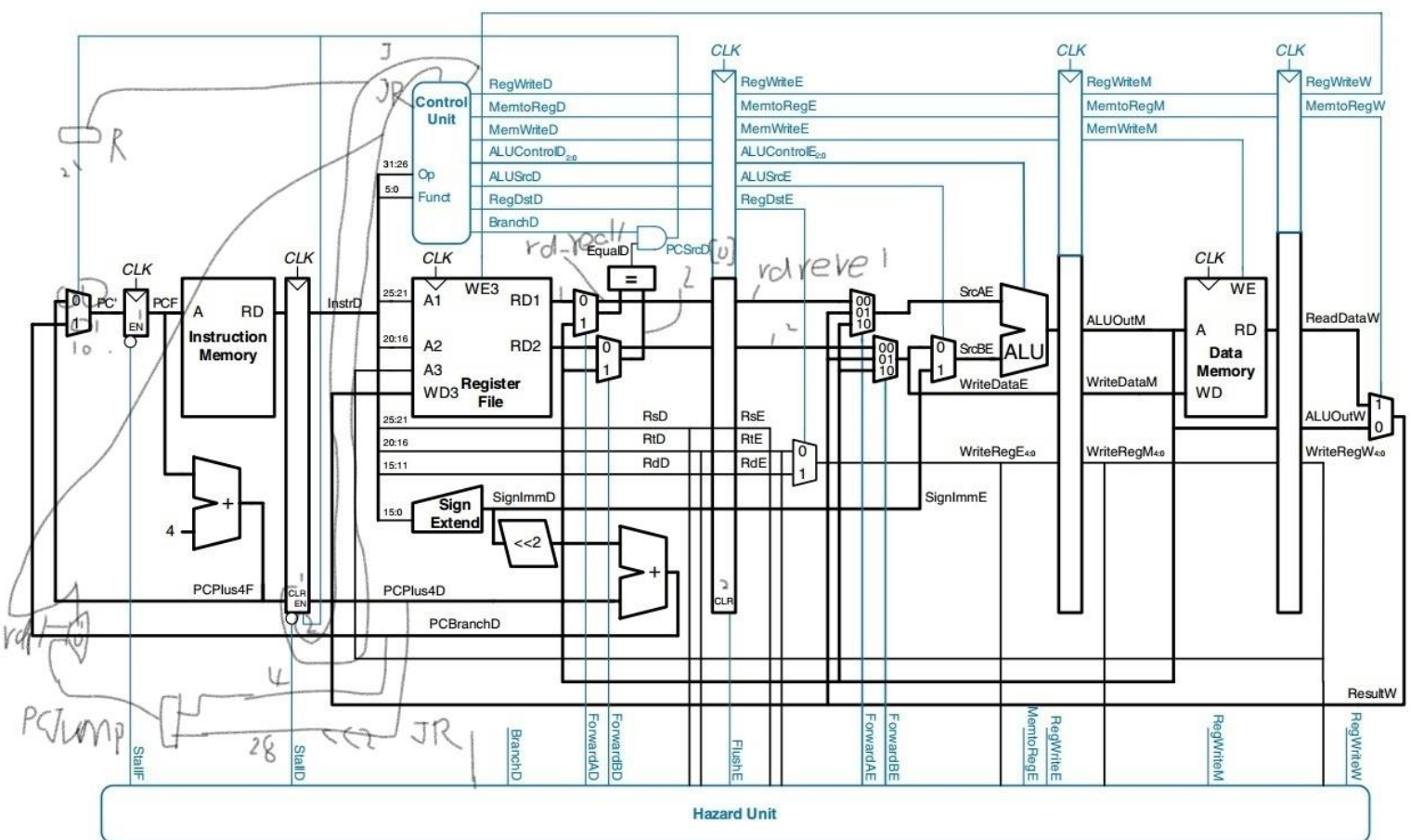
实验内容：

设计并实现流水线、静态分支预测功能，与对 16 个基本指令基础上的 20 条指令的处理功能。

实验分析与设计：

首先是对流水线整体的思考。根据 COD 书上的内容和 David Money Harris 的《Digital Design and Computer Architecture》上面的讲解，进行设计。

总体按照下图设计。



(原图来自这本书，为了实现 jjr 加了一些东西。大体参考了这本书对于单周期 cpu 的设计。)

为了避免数据冒险，首先要设置前推寄存器 ForwardAE，和 ForwardBE。

```
if((RsE != 0) && (RsE == WriteRegM) && RegWriteM)
    ForwardAE=2;
else if((RsE != 0) && (RsE == WriteRegW) && RegWriteW)
    ForwardAE=1;
```

判断条件就是，注意要优先判断 M，后判断 W。同时为了考虑 lw 对 reg 的影

```
lwstall = ((RsD == RtE) | (RtD == RtE)) & MemtoRegE;  
StallF = lwstall  
StallD = lwstall  
FlushE = lwstall
```

响，需要考虑 stall 和 flush

为了避免控制冒险，需要考虑 beq 等和 jr 等。

```
assign ForwardAD = (RsD != 0) & (RsD == WriteRegM) &  
    RegWriteM;  
assign ForwardBD = (RtD != 0) & (RtD == WriteRegM) &  
    RegWriteM;  
  
wire branchstall = (BranchD | JumpR ) & RegWriteE &(  
    WriteRegE == RsD | WriteRegE == RtD) | (BranchD |  
    JumpR ) & MemtoRegM & (WriteRegM == RsD | WriteRegM  
    == RtD);
```

将以上封装在 hazard 模块中。

对于 16 条指令（以及**额外实现**的 4 条指令），设置 control 模块，在其中类比 lab6 针对 opcode 和 funct。针对运算，由于 funct 对应运算和我们之前实现的 alu 不一致，所以设置 alu_decoder 转码模块。

对于分支预测，我实现了静态**分支预测**。

只需要简单的根据寄存器读出来的两个值 RD1 和 RD2 是否相等，设置 EqualD 与 control 输出的 BranchD 进行比较即可。另外考虑到 J,JR,需要根据 control 输出的 Jump 和 JumpR 进一步判断。

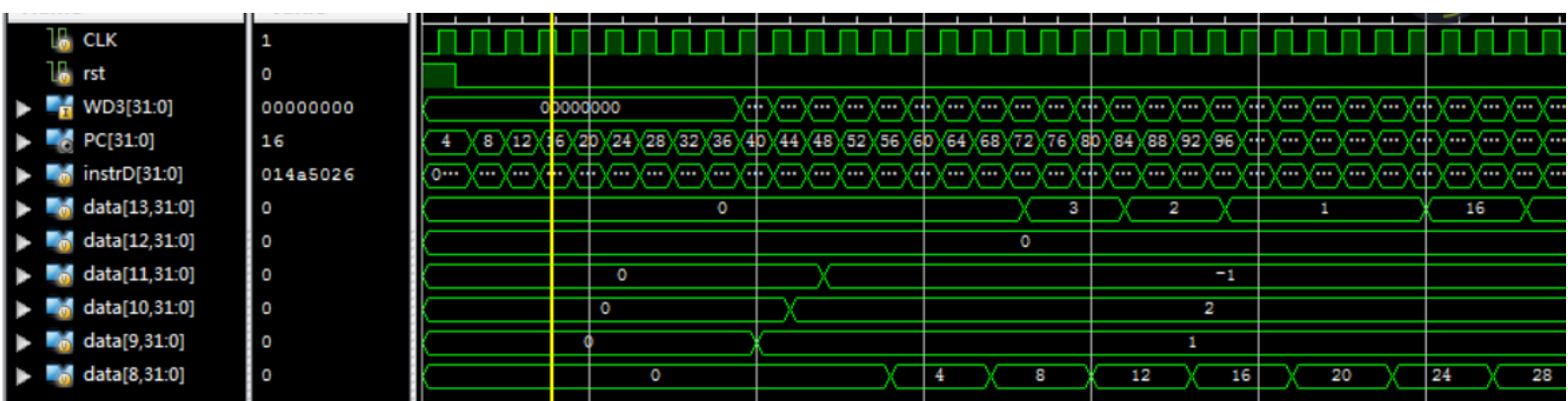
为了针对 beq，bgtz，以及加了 4 条 b 指令后继续沿用 equalD，所以直接设置 equalD 模块得到 equalD 结果，用于判断 PCSrcD。

另外值得一说的是，为了将单个周期控制在一个时钟周期，所以将寄存器文件的触发沿改为了 CLK 的下降沿。

最后，感谢 llxx 老师和四位助教一个学期以来的知道和陪伴！你们都是坠胖的！

实验结果：

仿真波形：



内存：

0x0	0	0
0x2	3	2
0x4	1	1
0x6	16	3
0x8	0	-2
0xA	1	0
0xC	8	2
0xE	0	0
0x10	0	0
0x12	0	0

实验代码：

Top：

```
1 module top(
2   input CLK,
3   input rst
4 );
5 wire [31:0] PC;
6 wire [31:0] PCPlus4F;
7 reg [31:0] PCPlus4D;
8 wire [31:0] PCBranchD;
9 reg [31:0] PCF;
10 wire BranchD;
11 wire Jump;
12 wire [31:0] instr;
13 reg [31:0] instrD;
14 wire [31:0] SignImmD;
15 reg [31:0] SignImmE;
16 wire EN1;
17 wire EN2;
18 wire RegWriteD;
19 wire RegWriteE;
20 reg RegWriteM;
21 reg RegWriteW;
22 wire MemtoRegD;
23 reg MemtoRegE;
24 wire MemtoRegM;
25 reg MemtoRegW;
26 wire MemWriteD;
27 reg MemWriteE;
28 reg MemWriteM;
29 wire [2:0] ALUControlD;
30 reg [2:0] ALUOp;
31 wire [2:0] ALUControlE;
32 reg [2:0] ALUSrcD;
33 wire ALUSrcE;
34 reg RegDstD;
35 wire RegDstE;
36 reg RD1;
37 wire [31:0] RDreal1;
38 reg [31:0] RDreal2;
39 reg [31:0] RDrealreal1;
40 wire [31:0] RD2;
41 wire [31:0] RDreal2;
42 wire [31:0] RDreal2;
43 wire [1:0] PCSrcD;
44 wire EqualD;
45 wire RsD = instrD[25:21];
46 wire RtD = instrD[20:16];
47 wire RdD = instrD[15:11];
48 reg [4:0] RsE;
49 reg [4:0] RtE;
50 reg [4:0] RdE;
51 wire [31:0] WriteDataE;
52 reg [31:0] WriteDataM;
53 wire [31:0] ALUOutE;
54 reg [31:0] ALUOutM;
55 reg [31:0] ALUOutW;
56 wire [4:0] WriteRegE;
57 reg [4:0] WriteRegM;
58 reg [4:0] WriteRegW;
59 wire [31:0] SrcAE;
60 wire [31:0] SrcBE;
61 wire [31:0] ReadDataM;
62 reg [31:0] ReadDataW;
63 wire StallF;
64 wire StallD;
65 wire ForwardAD;
66 wire ForwardBD;
67 wire [1:0] ForwardAE;
68 wire [1:0] ForwardBE;
69 wire FlushE;
70 wire CLR;
71 wire [31:0] PCJump;
72 wire [31:0] PCJumpR;
73 wire [31:0] ResultW = MemtoRegW ? ReadDataW : ALUOutW;
74 assign PC = PCSrcD == 0 ? PCPlus4F : PCSrcD == 1 ? PCBranchD :
    PCJumpR;
75 assign EN1 = ~StallF;
76 assign EN2 = ~StallD;
77 assign PCPlus4F = PCF + 32'd4;
78 assign PCSrcD[0] = EqualD && BranchD;
79 assign PCSrcD[1] = Jump;
80 assign RDreal1 = ForwardAD ? ALUOutM : RD1;
81 assign RDreal2 = ForwardBD ? ALUOutM : RD2;
82 assign CLR = PCSrcD || Jump || JumpR;
83 assign SrcAE = ForwardAE == 0 ? RDrealreal1 : (
    ForwardAE == 1 ? ResultW : ALUOutM);
84 assign WriteDataE = ForwardBE == 0 ? RDrealreal2 : (
    ForwardBE == 1 ? ResultW : ALUOutM);
85 assign SrcBE = ALUSrcE ? SignImmE : WriteDataE;
86 assign WriteRegE = RegDstE ? RdE : RtE;
87 assign PCJump[27:0] = instrD[25:0] << 2;
88 assign PCJump[31:28] = PCPlus4D[31:28];
89 assign PCJumpR = JumpR ? RDreal1 : PCJump;
90 assign PCBranchD = (SignImmD << 2) + PCPlus4D;
91 always@(posedge CLK or posedge rst) begin
92   if(rst)begin
93     instrD <= 0;
94     PCPlus4D <= 0;
95   end
96   else if(EN2) begin
97     if(CLR)
98       begin
99         instrD <= 0;
100        PCPlus4D <= 0;
101      end
102     else
103       begin
104         instrD <= instr;
105         PCPlus4D <= PCPlus4F;
106       end
107   end
108 end
109 always@(posedge CLK or posedge rst) begin
110   if(rst)begin
111     RDrealreal1 <= 0;
112     RDrealreal2 <= 0;
113     RsE <= 0;
114     RtE <= 0;
115     RdE <= 0;
116     RegWriteE <= 0;
117     MemtoRegE <= 0;
118     MemWriteE <= 0;
119   end
120   else begin
121     RDrealreal1 <= FlushE ? 0 : RD1;
122     RDrealreal2 <= FlushE ? 0 : RD2;
123     RsE <= FlushE ? 0 : RsD;
124     RtE <= FlushE ? 0 : RtD;
125     RdE <= FlushE ? 0 : RdD;
126     RegWriteE <= FlushE ? 0 : RegWriteD;
127     MemtoRegE <= FlushE ? 0 : MemtoRegD;
128     MemWriteE <= FlushE ? 0 : MemWriteD;
129     ALUControlE <= FlushE ? 0 : ALUControlD;
130     ALUSrcE <= FlushE ? 0 : ALUSrcD;
131     RegDstE <= FlushE ? 0 : RegDstD;
132     SignImmE <= FlushE ? 0 : SignImmD;
133   end
134 end
135 always@(posedge CLK or posedge rst) begin
136   if(rst)begin
137     PCF <= 0;
138   end
139   else begin
140     RegWriteM <= 0;
141     MemtoRegM <= 0;
142     MemWriteM <= 0;
143     ALUOutM <= 0;
144     WriteDataM <= 0;
145     WriteRegM <= 0;
146   end
147   else begin
148     if(EN1)
149       PCF <= PC;
150     RegWriteM <= RegWriteE;
151     MemtoRegM <= MemtoRegE;
152     MemWriteM <= MemWriteE;
153     ALUOutM <= ALUOutE;
154     WriteDataM <= WriteDataE;
155     WriteRegM <= WriteRegE;
156   end
157 end
158 always@(posedge CLK or posedge rst) begin
159   if(rst)begin
160     RegWriteW <= 0;
161     MemtoRegW <= 0;
162     ReadDataW <= 0;
163     ALUOutW <= 0;
164     WriteRegW <= 0;
165   end
166   else begin
167     RegWriteW <= RegWriteM;
168     MemtoRegW <= MemtoRegM;
169     ReadDataW <= ReadDataM;
170     ALUOutW <= ALUOutM;
171     WriteRegW <= WriteRegM;
172   end
173 end
174 control u_control(
175   .Op(instrD[31:26]),
176   .Func(instrD[5:0]),
177   .RegWriteD(RegWriteD),
178   .MemtoRegD(MemtoRegD),
179   .MemWriteD(MemWriteD),
180   .ALUOp(ALUOp),
181   .ALUSrcD(ALUSrcD),
182   .RegDstD(RegDstD),
183   .BranchD(BranchD),
184   .Jump(Jump),
185   .JumpR(JumpR)
186 );
187 REG_FILE u_REG_FILE(
188   .CLK(CLK),
189   .rst(rst),
190   .A1(instrD[25:21]),
191   .A2(instrD[20:16]),
192   .A3(WriteRegW),
193   .WD3(ResultW),
194   .WE3(RegWriteW),
195   .RD1(RD1),
196   .RD2(RD2)
197 );
198 InsMem u_InsMem(
199   .PCF[7:2],
200   .instr(instr)
201 );
202 DataMem u_DataMem(
203   .ALUOutM[6:2],
204   .WriteDataM,
205   .CLK,
206   .MemWriteM,
207   .ReadDataM
208 );
209 sign_extend u_sign_extend(
210   .instrD[15:0],
211   .SignImmD
212 );
213 alu u_alu(
214   .SrcAE(SrcAE),
215   .SrcBE(SrcBE),
216   .ALUControlE(ALUControlE),
217   .ALUOutE(ALUOutE)
218 );
219 alu_decoder u_alu_decoder(
220   .ALUOp(ALUOp),
221   .Func(instrD[5:0]),
222   .ALUControl(ALUControlD)
223 );
224 hazard u_hazard(
225   .WriteRegE(WriteRegE),
226   .WriteRegM(WriteRegM),
227   .WriteRegW(WriteRegW),
228   .RegWriteE(RegWriteE),
229   .RegWriteM(RegWriteM),
230   .RegWriteW(RegWriteW),
231   .MemtoRegE(MemtoRegE),
232   .MemtoRegM(MemtoRegM),
233   .BranchD(BranchD),
234   .JumpR(JumpR),
235   .RtE(RtE),
236   .RsE(RsE),
237   .RtD(RtD),
238   .RsD(RsD),
239   .StallF(StallF),
240   .StallD(StallD),
241   .ForwardAD(ForwardAD),
242   .ForwardBD(ForwardBD),
243   .FlushE(FlushE),
244   .ForwardAE(ForwardAE),
245   .ForwardBE(ForwardBE)
246 );
247 equald u_equald(
248   .RDreal1(RDreal1),
249   .RDreal2(RDreal2),
250   .Op(instrD[31:26]),
251   .RtD(RtD),
252   .EqualD(EqualD)
253 );
254 endmodule
```

Reg_file

```

1  module REG_FILE(
2      input          CLK,
3      input          rst,
4      input [4:0]    A1,
5      input [4:0]    A2,
6      input [4:0]    A3,
7      input [31:0]   WD3,
8      input          WE3,
9      output reg [31:0] RD1,
10     output reg [31:0] RD2
11 );
12
13 reg [31:0] data [31:0];
14 always@(negedge CLK or posedge rst)
15 begin
16     if (rst) begin
17         data[0] <= 32'b0;
18         data[1] <= 32'b0;
19         data[2] <= 32'b0;
20         data[3] <= 32'b0;
21         data[4] <= 32'b0;
22         data[5] <= 32'b0;
23         data[6] <= 32'b0;
24         data[7] <= 32'b0;
25         data[8] <= 32'b0;
26         data[9] <= 32'b0;
27         data[10] <= 32'b0;
28         data[11] <= 32'b0;
29         data[12] <= 32'b0;
30         data[13] <= 32'b0;
31         data[14] <= 32'b0;
32         data[15] <= 32'b0;
33         data[16] <= 32'b0;
34         data[17] <= 32'b0;
35         data[18] <= 32'b0;
36         data[19] <= 32'b0;
37         data[20] <= 32'b0;
38         data[21] <= 32'b0;
39         data[22] <= 32'b0;
40         data[23] <= 32'b0;
41         data[24] <= 32'b0;
42         data[25] <= 32'b0;
43         data[26] <= 32'b0;
44         data[27] <= 32'b0;
45         data[28] <= 32'b0;
46         data[29] <= 32'b0;
47         data[30] <= 32'b0;
48         data[31] <= 32'b0;
49     end
50     else if(WE3)
51         data[A3] <= WD3;
52     end
53
54     always@(*)
55     begin
56         if(A1)
57             RD1 = data[A1];
58         else
59             RD1 = 32'h0;
60     end
61
62     always@(*)begin
63         if(A2)
64             RD2 = data[A2];
65         else
66             RD2 = 32'h0;
67     end
68 endmodule
69

```

Alu :

```
1  module alu(  
2  input  signed  [31:0]  SrcAE,  
3  input  signed  [31:0]  SrcBE,  
4  input          [2:0]  ALUControlE,  
5  output reg       [31:0] ALUOutE  
6  );  
7  
8  always@(*)  
9  begin  
10     case(ALUControlE)  
11         3'h0: ALUOutE = 32'h0;  
12         3'h1: ALUOutE = SrcAE + SrcBE;  
13         3'h2: ALUOutE = SrcAE - SrcBE;  
14         3'h3: ALUOutE = SrcAE & SrcBE;  
15         3'h4: ALUOutE = SrcAE | SrcBE;  
16         3'h5: ALUOutE = SrcAE ^ SrcBE;  
17         3'h6: ALUOutE = ~(SrcAE | SrcBE);  
18         default: ALUOutE = 32'h0;  
19     endcase  
20 end  
21  
22 endmodule  
23
```


control

```
1 module control(
2     input [5:0] Op,
3     input [5:0] Funct,
4     output reg RegWriteD,
5     output reg MemtoRegD,
6     output reg MemWriteD,
7     output reg [2:0] ALUOp,
8     output reg ALUSrcD,
9     output reg RegDstD,
10    output reg BranchD,
11    output reg Jump,
12    output reg JumpR
13);
14 parameter Rtype=6'b000000;
15 parameter addi=6'b001000;
16 parameter addiu=6'b001001;
17 parameter andi=6'b001100;
18 parameter ori=6'b001101;
19 parameter xori=6'b001110;
20 parameter lw=6'b100011;
21 parameter sw=6'b101011;
22 parameter beq=6'b000100;
23 parameter bgez=6'b000001;
24 parameter bgtz=6'b000111;
25 parameter blez=6'b000110;
26 parameter bltz=6'b000001;
27 parameter bne=6'b000101;
28 parameter j=6'b000010;
29 always@(*) begin
30     case(Op)
31     Rtype: begin
32         if(Funct==6'b001000)
33             RegWriteD=0;
34             MemtoRegD=0;
35             MemWriteD=0;
36             ALUSrcD=0;
37             RegDstD=0;
38             BranchD=0;
39             Jump=1;
40             JumpR=1;
41             ALUOp=2;
42         end
43     else
44     begin
45         RegWriteD=1;
46         MemtoRegD=0;
47         MemWriteD=0;
48         ALUSrcD=0;
49         RegDstD=1;
50         BranchD=0;
51         Jump=0;
52         JumpR=0;
53         ALUOp=2;
54     end
55 end
56 end
57 addi: begin
58     RegWriteD=1;
59     MemtoRegD=0;
60     MemWriteD=0;
61     ALUSrcD=1;
62     RegDstD=0;
63     BranchD=0;
64     Jump=0;
65     JumpR=0;
66     ALUOp=0;
67 end
68 addiu: begin
69     RegWriteD=1;
70     MemtoRegD=0;
71     MemWriteD=0;
72     ALUSrcD=1;
73     RegDstD=0;
74     BranchD=0;
75     Jump=0;
76     JumpR=0;
77     ALUOp=0;
78 end
79 andi: begin
80     RegWriteD=1;
81     MemtoRegD=0;
82     MemWriteD=0;
83     ALUSrcD=1;
84     RegDstD=0;
85     BranchD=0;
86     Jump=0;
87     JumpR=0;
88     ALUOp=3;
89 end
90 ori: begin
91     RegWriteD=1;
92     MemtoRegD=0;
93     MemWriteD=0;
94     ALUSrcD=1;
95     RegDstD=0;
96     BranchD=0;
97     Jump=0;
98     JumpR=0;
99     ALUOp=4;
100 end
101 xori: begin
102     RegWriteD=1;
103     MemtoRegD=0;
104     MemWriteD=0;
105     ALUSrcD=1;
106     RegDstD=0;
107     BranchD=0;
108     Jump=0;
109 end
110 Jump=0;
111 JumpR=0;
112 ALUOp=5;
113 end
114 lw: begin
115     RegWriteD=1;
116     MemtoRegD=1;
117     MemWriteD=0;
118     ALUSrcD=1;
119     RegDstD=0;
120     BranchD=0;
121     Jump=0;
122     JumpR=0;
123     ALUOp=0;
124 end
125 sw: begin
126     RegWriteD=0;
127     MemtoRegD=0;
128     MemWriteD=1;
129     ALUSrcD=1;
130     RegDstD=0;
131     BranchD=0;
132     Jump=0;
133     JumpR=0;
134     ALUOp=0;
135 end
136 beq: begin
137     RegWriteD=0;
138     MemtoRegD=0;
139     MemWriteD=0;
140     ALUSrcD=0;
141     RegDstD=0;
142     BranchD=1;
143     Jump=0;
144     JumpR=0;
145     ALUOp=0;
146 end
147 bne: begin
148     RegWriteD=0;
149     MemtoRegD=0;
150     MemWriteD=0;
151     ALUSrcD=0;
152     RegDstD=0;
153     BranchD=1;
154     Jump=0;
155     JumpR=0;
156     ALUOp=0;
157 end
158 blez: begin
159     RegWriteD=0;
160     MemtoRegD=0;
161     MemWriteD=0;
162     ALUSrcD=0;
163     RegDstD=0;
164     BranchD=0;
165     Jump=0;
166     JumpR=0;
167     ALUOp=0;
168 end
169 bgez: begin
170     RegWriteD=0;
171     MemtoRegD=0;
172     MemWriteD=0;
173     ALUSrcD=0;
174     RegDstD=0;
175     BranchD=0;
176     Jump=0;
177     JumpR=0;
178     ALUOp=0;
179 end
180 bltz: begin
181     RegWriteD=0;
182     MemtoRegD=0;
183     MemWriteD=0;
184     ALUSrcD=0;
185     RegDstD=0;
186     BranchD=1;
187     Jump=0;
188     JumpR=0;
189     ALUOp=0;
190 end
191 j: begin
192     RegWriteD=0;
193     MemtoRegD=0;
194     MemWriteD=0;
195     ALUSrcD=0;
196     RegDstD=0;
197     BranchD=0;
198     Jump=1;
199     JumpR=0;
200     ALUOp=0;
201 end
202 default: begin
203     RegWriteD=0;
204     MemtoRegD=0;
205     MemWriteD=0;
206     ALUSrcD=0;
207     RegDstD=0;
208     BranchD=0;
209     Jump=0;
210     JumpR=0;
211     ALUOp=0;
212 end
213 endcase
214 end
215 endmodule
```

alu_decoder:

```
1
2 module alu_decoder(
3     input [2:0]ALUOp,
4     input [5:0]Funct,
5     output reg [2:0]ALUControl
6 );
7
8     always@(*)
9     begin
10 case(ALUOp)
11     0: ALUControl<=1;
12     1: ALUControl<=2;
13     2:
14     begin
15         case(Funct)
16             6'b100000: ALUControl<=1;
17             6'b100001: ALUControl<=1;
18             6'b100010: ALUControl<=2;
19             6'b100011: ALUControl<=2;
20             6'b100100: ALUControl<=3;
21             6'b100101: ALUControl<=4;
22             6'b100110: ALUControl<=5;
23             6'b100111: ALUControl<=6;
24             6'b001000: ALUControl<=0;
25             default: ALUControl<=3'b000;
26         endcase
27     end
28     3: ALUControl<=3;
29     4: ALUControl<=4;
30     5: ALUControl<=5;
31     6: ALUControl<=6;
32     7: ALUControl<=7;
33     default: ALUControl<=1;
34 endcase
35 end
36
37 endmodule
38
```

hazard:

```
1 module hazard(  
2   input          BranchD,  
3   input          MemtoRegE,  
4   input          RegWriteE,  
5   input          MemtoRegM,  
6   input          RegWriteM,  
7   input          RegWriteW,  
8   input          JumpR,  
9   input [4:0]    RsD,  
10  input [4:0]    RtD,  
11  input [4:0]    RsE,  
12  input [4:0]    RtE,  
13  input [4:0]    WriteRegE,  
14  input [4:0]    WriteRegM,  
15  input [4:0]    WriteRegW,  
16  output          StallF,  
17  output          StallD,  
18  output          ForwardAD,  
19  output          ForwardBD,  
20  output          FlushE,  
21  output reg [1:0] ForwardAE,  
22  output reg [1:0] ForwardBE  
23 );  
24  
25 wire lwstall = ((RsD == RtE) | (RtD == RtE)) & MemtoRegE;  
26 wire branchstall = (BranchD | JumpR) & RegWriteE &  
    WriteRegE == RsD | WriteRegE == RtD) | (BranchD |  
    JumpR) & MemtoRegM & (WriteRegM == RsD | WriteRegM  
    == RtD);  
27  
28 always@(*) begin  
29     ForwardAE=0;  
30     ForwardBE=0;  
31     if((RsE != 0) && (RsE == WriteRegM) && RegWriteM)  
32         ForwardAE=2;  
33     else if((RsE != 0) && (RsE == WriteRegW) && RegWriteW  
34         )  
35         ForwardAE=1;  
36     if((RtE != 0) && (RtE == WriteRegM) && RegWriteM)  
37         ForwardBE=2;  
38     else if((RtE != 0) && (RtE == WriteRegW) && RegWriteW  
39         )  
40         ForwardBE=1;  
41  
42     assign ForwardAD = (RsD != 0) & (RsD == WriteRegM) &  
        RegWriteM;  
43     assign ForwardBD = (RtD != 0) & (RtD == WriteRegM) &  
        RegWriteM;  
44  
45     assign StallF = lwstall | branchstall;  
46     assign StallD = lwstall | branchstall;  
47     assign FlushE = lwstall | branchstall;  
48  
49 endmodule
```

equald:

```
1 module equald(  
2     input signed [31:0] RDreal1,  
3     input signed [31:0] RDreal2,  
4     input      [5:0] Op,  
5     input      [4:0] RtD,  
6     output reg      EqualD  
7 );  
8  
9     parameter beq = 6'b000100;  
10    parameter bgez = 6'b000001;  
11    parameter bgtz = 6'b000111;  
12    parameter blez = 6'b000110;  
13    parameter bltz = 6'b000001;  
14    parameter bne = 6'b000101;  
15  
16    always@(*)  
17    begin  
18        case(Op)  
19            beq:    if(RDreal1 == RDreal2)  
20                    EqualD = 1;  
21                    else  
22                        EqualD = 0;  
23            bgez:    if(RtD == 5'b000001)//bgez  
24                        begin  
25                            if(RDreal1 >= 0)  
26                                EqualD = 1;  
27                            else  
28                                EqualD = 0;  
29                        end  
30                    else  
31                        begin  
32                            if(RDreal1 < 0)//bltz  
33                                EqualD = 1;  
34                            else  
35                                EqualD = 0;  
36                        end  
37            bgtz:    if(RDreal1 > 0)  
38                    EqualD = 1;  
39                    else  
40                        EqualD = 0;  
41            blez:    if(RDreal1 <= 0)  
42                    EqualD = 1;  
43                    else  
44                        EqualD = 0;  
45            bne:    if(RDreal1 != RDreal2)  
46                    EqualD = 1;  
47                    else  
48                        EqualD = 0;  
49            default: EqualD = 0;  
50        endcase  
51    end  
52  
53 endmodule
```

sign_extend:

```
2  module sign_extend(  
3      input  [15:0]din,  
4      output [31:0]dout  
5  );  
6  
7      assign dout[15:0]=din[15:0];  
8      assign dout[31]=din[15];  
9      assign dout[30]=din[15];  
10     assign dout[29]=din[15];  
11     assign dout[28]=din[15];  
12     assign dout[27]=din[15];  
13     assign dout[26]=din[15];  
14     assign dout[25]=din[15];  
15     assign dout[24]=din[15];  
16     assign dout[23]=din[15];  
17     assign dout[22]=din[15];  
18     assign dout[21]=din[15];  
19     assign dout[20]=din[15];  
20     assign dout[19]=din[15];  
21     assign dout[18]=din[15];  
22     assign dout[17]=din[15];  
23     assign dout[16]=din[15];  
24  
25     endmodule  
26
```

sim:

```
3
4  module sim;
5
6      // Inputs
7      reg CLK;
8      reg rst;
9
10     // Instantiate the Unit Under Test (UUT)
11     top uut (
12         .CLK(CLK),
13         .rst(rst)
14     );
15
16     always #5 CLK=~CLK;
17
18     initial begin
19         // Initialize Inputs
20         CLK = 0;
21         rst = 1;
22
23         // Wait 100 ns for global reset to finish
24         #10;
25
26         rst = 0;
27         // Add stimulus here
28
29     end
30
31 endmodule
```