实验六 CPU 综合设计

一、实验目的

- 1 掌握复杂系统设计方法。
- 2 深刻理解计算机系统硬件原理。

二、实验内容

- 1) 设计一个基于 MIPS 指令集的 CPU, 支持以下指令: {add, sub, addi, lw, sw, beq, j, nop};
- 2) CPU 需要包含寄存器组、RAM 模块、ALU 模块、指令译码模块;
- 3) 该 CPU 能运行基本的汇编指令; (D~C+)
- 以下为可选内容:
- 4) 实现多周期 CPU (B-~B+);
- 5) 实现以下高级功能之一 $(A-^{\sim}A+)$:
- (1) 实现 5 级流水线 CPU;
- (2)实现超标量;
- (3)实现 4 路组相联缓存;

可基于 RISC V 、ARM 指令集实现。

如发现代码为抄袭代码,成绩一律按不及格处理。

三、实验要求

编写相应测试程序,完成所有指令测试。

四、实验代码及结果

根据《Digital Design and Computer Architecture》书中的对 MIPS 流水线介绍与原理图加以改进,本实验实现了五级流水线的 CPU。

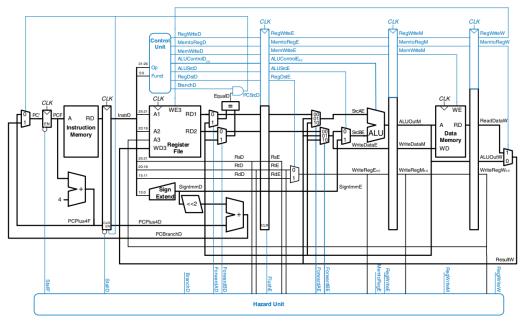


Figure 7.58 Pipelined processor with full hazard handling

✓ ■ ■ PipelineMIPSCPU (PipelineMIPSCPU.v) (12)

- PC_1 : PCreg (PC.v)
- L1I_cache : IMem (IMem.v)
- IF_IDreg : IF_ID (IF_ID.v)
- Register_1 : RegFile (RegFile.v)
- > ctrl : Controller (Controller.v) (2)
 - ID_EXreg : ID_EX (ID_EX.v)
 - alu_forward_1 : ALU_Forward (ALU_Forward.v)
- > **alu**: ALU_8 (ALU_8.v) (9)
 - EX_MEMreg : EX_MEM (EX_MEM.v)
- > Dmem_1 : RAM_4Kx32_inout (RAM_4Kx32_inout.v) (9)
 - MEM_WBreg : MEM_WB (MEM_WB.v)
 - HZ : Hazard_unit (Hazard_Unit.v)

实验代码如下:

```
module PipelineMIPSCPU;
reg clk,rst;

wire [31:0] instruction;

wire CF,SF,PF,OF,Zero;

wire [5:0] opcode;
wire [4:0] rs, rt, rd;
wire [5:0] Funct;
wire [31:0] Data;
wire [31:0] PCF,PC_in,pcp4F,PCbranchD;

wire [31:0] ResultW;
wire stallF,stallD,flushE;
wire [1:0] forwardAE,forwardBE,forwardAD,forwardBD;
```

```
wire [31:0] SrcA, SrcB;
wire [31:0] pcp4D,IR;
wire RegWriteD, MemtoRegD, MemWriteD, ALUSrcD, RegDstD, BranchD, JumpD;
wire EqualD,PCSrcD;
wire [2:0] ALUControlD;
wire [31:0] Imm_extend;
wire RegWriteE, MemtoRegE, MemWriteE, ALUSrcE, RegDstE;
wire [2:0] ALUControlE;
wire [31:0] RD1E, RD2E, ImmE;
wire [4:0] rsE,rtE,rdE;
wire [31:0] SrcAE,SrcBE,alu_result,WriteDataE;
wire [4:0] WriteRegM, WriteRegE;
wire [31:0] ALUOutM, WriteDataM;
wire RegWriteM, MemtoRegM, MemWriteM;
wire [31:0] ReadDataW,ALUOutW;
wire [4:0] WriteRegW;
wire MemtoRegW, RegWriteW;
initial begin
   clk=0; rst=0;
end
always #10 clk=~clk;
assign PC_in=PCSrcD?PCbranchD:pcp4F;
PCreg PC_1(
   .clk(clk),
   .rst(rst),
   .pc(PCF),
   .pc_in(PC_in),
   .stall(stallF)
);
assign pcp4F=PCF+4;
assign PCbranchD=JumpD?IR[25:0]:(Imm_extend<<2)+pcp4D;</pre>
IMem L1I_cache(
   .A(PCF>>2),
   .RD(instruction)
);
```

```
IF_ID IF_IDreg(
   .clk(clk),
   .rst(rst),
   .In_pc(pcp4F),
   .In_Ins(instruction),
   .stall(stallD),
   .clr(PCSrcD),
   .out_pc(pcp4D),
   .out_Ins(IR)
);
assign
         opcode = IR[31:26];
         rs = IR[25:21];
assign
assign
         rt = IR[20:16];
assign
         rd = IR[15:11];
assign
         Funct = IR[5:0];
RegFile Register_1(
   .CLK(clk),
   .WE3(RegWriteW),
   .RA1(rs),
   .RA2(rt),
   .WA3(WriteRegW),
   .WD3(ResultW),
   .RD1(SrcA),
   .RD2(SrcB),
   .RST(rst)
);
/*Control_Unit ctrl (
   .op(opcode),
   .funct(Funct),
   .RegWriteD(RegWriteD),
   .MemtoRegD(MemtoRegD),
   .MemWriteD(MemWriteD),
   .AluSrcD(ALUSrcD),
   .RegDstD(RegDstD),
   .BranchD(BranchD),
   .ALUControlD(ALUControlD)
```

```
);*/
Controller ctrl (
   .Op(opcode),
   .Funct(Funct),
   .RegWrite(RegWriteD),
   .MemToReg(MemtoRegD),
   .MemWrite(MemWriteD),
   .ALUSrc(ALUSrcD),
   .RegDst(RegDstD),
   .Branch(BranchD),
   .Jump(JumpD),
   .ALUControl(ALUControlD)
);
assign EqualD=( ( forwardAD==(2'b10)?ResultW:(forwardAD==(2'b01)?ALUOutM:SrcA))
           ( forwardBD==(2'b10)?ResultW:(forwardBD==(2'b01)?ALUOutM:SrcB)) );
assign PCSrcD=((BranchD|JumpD)&EqualD);
assign Imm_extend={ {16{IR[15]}},IR[15:0] };
ID_EX ID_EXreg(
   .clk(clk),
   .clr(flushE),
   .RD1((RegWriteW&&rs==WriteRegW)?ResultW:SrcA),
   .RD2((RegWriteW&&rt==WriteRegW)?ResultW:SrcB),
   .ImmD(Imm_extend),
   .rsD(rs),
   .rtD(rt),
   .rdD(rd),
   .RegWriteD(RegWriteD), .MemtoRegD(MemtoRegD), .MemWriteD(MemWriteD),
   .ALUSrcD(ALUSrcD), .RegDstD(RegDstD),
   .ALUControlD(ALUControlD),
   .rsE(rsE),.rtE(rtE),.rdE(rdE),
   .RD1E(RD1E),.RD2E(RD2E),.ImmE(ImmE),
   .RegWriteE(RegWriteE), .MemtoRegE(MemtoRegE), .MemWriteE(MemWriteE),
   .ALUSrcE(ALUSrcE), .RegDstE(RegDstE),
   .ALUControlE(ALUControlE)
);
//The ALUsrc here is used to control the sign-extend immediate expansion component,
//which means that for all non I-type instructions, immediate expansion is not required.
```

```
//At this point, the value of this component (ALUsrc) is 0, and for I-type instructions, its value is
1
   ALU_Forward alu_forward_1(
       .forwardAE(forwardAE),.forwardBE(forwardBE),
       .ALUSrcE(ALUSrcE),
       .ALUOutM(ALUOutM),.ResultW(ResultW),
       .RD1E(RD1E),.RD2E(RD2E),.SignlmmE(ImmE),
       .SrcAE(SrcAE),.SrcBE(SrcBE),.WriteDataE(WriteDataE)
   );
   ALU_8 alu (
       .A(SrcAE),
       .B(SrcBE),
       .OP(ALUControlE),
       .F(alu_result),
       .ZF(Zero),
       .CF(CF),
       .SF(SF),
       .PF(PF),
       .OF(OF)
   );
   assign WriteRegE=RegDstE?rdE:rtE;
   EX_MEM EX_MEMreg(
       .alu_output(alu_result),.WriteDataE(WriteDataE),
       .WriteRegE(WriteRegE),
       .clk(clk),.rst(rst),
       .RegWriteE(RegWriteE),.MemtoRegE(MemtoRegE),.MemWriteE(MemWriteE),
       .ALUOutM(ALUOutM),.WriteDataM(WriteDataM),
       .WriteRegM(WriteRegM),
       .RegWriteM(RegWriteM),.MemtoRegM(MemtoRegM),.MemWriteM(MemWriteM)
   );
   Dmemory Dmem_1 (
```

.Addr(ALUOutM),

.R_W(MemWriteM),
.Data_out(Data),

.Data_in(WriteDataM),

```
.clk(clk)
);
MEM_WB MEM_WBreg(
   .ALUOutM(ALUOutM),.ReadDataM(MemWriteM?32'b0:Data),
   .clk(clk),.rst(rst),
   .WriteRegM(WriteRegM),
   .RegWriteM(RegWriteM),.MemtoRegM(MemtoRegM),
   .ReadDataW(ReadDataW),.ALUOutW(ALUOutW),
   .WriteRegW(WriteRegW),
   .RegWriteW(RegWriteW),.MemtoRegW(MemtoRegW)
);
assign ResultW=MemtoRegW?ReadDataW:ALUOutW;
Hazard_unit HZ(
   .rsD(rs),.rtD(rt),
   .rsE(rsE),.rtE(rtE),
   .BranchD(BranchD|JumpD),
   .MemtoRegE(MemtoRegE),.RegWriteE(RegWriteE),
   .MemtoRegM(MemtoRegM),.RegWriteM(RegWriteM),
   .RegWriteW(RegWriteW),
   .WriteRegE(WriteRegE),.WriteRegM(WriteRegM),.WriteRegW(WriteRegW),
   .stallF(stallF),.stallD(stallD),
   .forwardAD(forwardAD),.forwardBD(forwardBD),.flushE(flushE),
   .forwardAE(forwardAE),.forwardBE(forwardBE)
);
```

endmodule

```
各模块代码:
module PCreg(
   input wire clk, rst,
   input wire[31:0] pc_in,
   input wire stall,
   output reg[31:0] pc
);
   initial begin
       pc = 32'b0;
   end
   always@(posedge clk)begin
       if(stall==1'b0)
           pc <= pc_in;</pre>
       else
           pc <= pc;</pre>
   end
endmodule
`define DATA_WIDTH 32
module IMem(
   input [31:0] A,
   output [31:0] RD);
   parameter IMEM_SIZE=256;
   reg [`DATA_WIDTH-1:0] RAM[IMEM_SIZE-1:0];
   initial
       $readmemh("C:/Users/nyodo/Desktop/cnt.txt",RAM);
   assign RD=RAM[A];
endmodule
module IF_ID(
   input wire clk,rst,
   input wire[31:0] In_pc,In_Ins,
   input wire stall,clr,//branch_stall/clr determines whether it is a branch adventure
   output reg[31:0] out_pc,out_Ins
);
   initial begin
       out_pc = 32'b0;
       out_Ins= 32'b0;
```

```
always@(posedge clk)begin
   if(stall)begin
       out_pc<=out_pc;</pre>
       out_Ins<=out_Ins;</pre>
   end
   else if(clr) begin //clear val of pipeline reg for branch ins & next ins
           out_pc<=32'b0;
           out_Ins<=32'b0;
   end
   else begin
           out_pc<=In_pc;</pre>
           out_Ins<=In_Ins;</pre>
   end
end
   //Pipeline pause, pause PC and lock IF/ID registers
   //pause for one clock cycle, lw is in mem stage,
   //The next instruction remains in the ID/EX pipeline,
   //decodes the value again, and the second instruction executes the IF operation again
```

 ${\tt Endmodule}$

```
module RegFile(CLK,WE3,RA1,RA2,WA3,WD3,RD1,RD2,RST);
  parameter ADDR_SIZE=5;
  input CLK,WE3;
  input [ADDR_SIZE-1:0] RA1,RA2,WA3;
  input [31:0] WD3;
  output [31:0] RD1,RD2;
  input RST;
  //regfile
```

```
reg [31:0] rf[2**ADDR_SIZE-1:0];
   //data in
   integer i;
   initial begin for(i=0;i<32;i=i+1) rf[i]<=32'b0; end
   always @(posedge CLK,posedge RST) begin
       if(RST) for(i=0;i<2**ADDR_SIZE;i=i+1) rf[i]<=0;</pre>
       else if(WE3) rf[WA3]<=WD3;</pre>
   end
   //data rdout
   assign RD1=(RA1!=0)?rf[RA1]:32'b0;
   assign RD2=(RA2!=0)?rf[RA2]:32'b0;
endmodule
module Controller(
   input [5:0] Op, Funct,
   input Zero,
   output MemToReg, MemWrite,
   output Branch, ALUSrc,
   output RegDst,RegWrite,
   output Jump,
   output [2:0] ALUControl);
   wire [1:0] ALUOp;
   MainDec MainDec_1(Op,MemToReg,MemWrite,Branch,ALUSrc,RegDst,RegWrite,Jump,ALUOp);
   ALUDec ALUDec_1(Funct,ALUOp,ALUControl);
endmodule
module MainDec(
   input [5:0] Op,
   output MemToReg, MemWrite,
   output Branch, ALUSrc,
   output RegDst,RegWrite,
   output Jump,
   output [1:0] ALUOp);
```

```
reg [8:0] Controls;
   assign {RegWrite, RegDst, ALUSrc, Branch, MemWrite, MemToReg, Jump, ALUOp}=Controls;
   always @(*)
       case(Op)
           6'b000000: Controls <=9'b110000010;//R(ADD,SUB,or,xor,nor)TYPE
           6'b100011: Controls <=9'b101001000;//LW
           6'b101011: Controls <=9'b001010000;//SW
           6'b000100: Controls <=9'b000100001;//BEQ
           6'b001000: Controls <=9'b101000000;//ADDI
           6'b000010: Controls <=9'b000000100;//J
           default: Controls <=9'bxxxxxxxxx;//illegal Op</pre>
       endcase
endmodule
module ALUDec(
   input [5:0] Funct,
   input [1:0] ALUOp,
   output reg [2:0] ALUControl);
   always @(*)
       case(ALUOp)
           2'b00:ALUControl <= 3'b010;// add for lw/sw/addi
           2'b01:ALUControl <= 3'b110;//sub for beq
           default: case(Funct)
               6'b100000: ALUControl <=3'b010; //add 2
               6'b100010: ALUControl <=3'b110; //sub 6
               6'b100100: ALUControl <=3'b000; //and 0
               6'b100101: ALUControl <=3'b001; //or 1
               6'b101010: ALUControl <=3'b111; //slt,cmp 7
               6'b100110: ALUControl <=3'b011; //xor 3
               6'b100111: ALUControl <=3'b100; //nor 4
               6'b000000: ALUControl <=3'b101; //sll 5
               default: ALUControl <=3'bxxx; //no idea</pre>
           endcase
       endcase
endmodule
module ID_EX(
   //input
```

```
input wire clk,clr,
   input wire [31:0] RD1, RD2, ImmD,
   input wire [4:0] rsD,rtD,rdD,
   input wire RegWriteD, MemtoRegD, MemWriteD, ALUSrcD, RegDstD,
    input wire [2:0] ALUControlD,
    //output
   output reg [31:0] RD1E, RD2E, ImmE,
   output reg [4:0] rsE,rtE,rdE,
   output reg RegWriteE, MemtoRegE, MemWriteE, ALUSrcE, RegDstE,
   output reg [2:0] ALUControlE
);
    initial begin
       RD1E<=32'b0;
       RD2E<=32'b0;
       ImmE<=32'b0;
       rsE<=5'b0;
       rtE<=5'b0;
       rdE<=5'b0;
       RegWriteE<=0;</pre>
       MemtoRegE<=0;</pre>
       MemWriteE<=0;</pre>
       ALUSrcE<=0;
       RegDstE<=0;</pre>
       ALUControlE<=3'b0;
   end
    always@(posedge clk) begin
       if(clr) begin
            RD1E<=32'b0;
            RD2E<=32'b0;
            ImmE<=32'b0;
            rsE<=5'b0;
            rtE<=5'b0;
            rdE<=5'b0;
            RegWriteE<=0;</pre>
           MemtoRegE<=0;</pre>
           MemWriteE<=0;</pre>
           ALUSrcE<=0;
            RegDstE<=0;</pre>
            ALUControlE<=3'b0;
```

```
end
       else begin
           RD1E<=RD1;
            RD2E<=RD2;
           ImmE<=ImmD;</pre>
            rsE<=rsD;
            rtE<=rtD;
            rdE<=rdD;
            RegWriteE<=RegWriteD;</pre>
           MemtoRegE<=MemtoRegD;</pre>
           MemWriteE<=MemWriteD;</pre>
           ALUSrcE<=ALUSrcD;
            RegDstE<=RegDstD;</pre>
           ALUControlE<=ALUControlD;
       end
   end
endmodule
module ALU_Forward(
   input[1:0]forwardAE,forwardBE,
   input ALUSrcE,
   input[31:0] ALUOutM,ResultW,RD1E,RD2E,SignlmmE,
   output reg[31:0] SrcAE,SrcBE,WriteDataE
);
   always@(*) begin
        case(forwardAE)
            2'b00:SrcAE<=RD1E;
            2'b01:SrcAE<=ResultW;
            2'b10:SrcAE<=ALUOutM;
       endcase
        case(forwardBE)
            2'b00:WriteDataE<=RD2E;
            2'b01:WriteDataE<=ResultW;
            2'b10:WriteDataE<=ALUOutM;
       endcase
       if(ALUSrcE)
            SrcBE<=SignlmmE;</pre>
```

```
else
    SrcBE<=WriteDataE;
end</pre>
```

endmodule

```
module ALU_8(F,CF,A,B,OP,OF,SF,PF,ZF);
   parameter SIZE=32;
   output reg [SIZE-1:0] F;
   output wire CF;
   input [SIZE-1:0] A,B;
   input [3:0] OP;
   output wire OF,SF,PF,ZF;
   parameter ALU_AND=4'b0000;//0
   parameter ALU_OR=4'b0001;//1
   parameter ALU_XOR=4'b0011;//3
   parameter ALU_NOR=4'b0100;//4
   parameter ALU_ADD=4'b0010;//2
   parameter ALU_SUB=4'b0110;//6
   parameter ALU_SLT=4'b0111;//7
   parameter ALU_SLL=4'b0101;//5
   wire [7:0] EN;
   wire [SIZE-1:0] FW;
   always @(*) begin
       //ALU_AND: begin F<=FA;end</pre>
       //ALU_OR: begin F<=A|B;end</pre>
       //ALU_XOR: begin F<=A^B;end</pre>
       //ALU_NOR: begin F <= \sim (A|B); end
       F<=FW;
   end
   decoder38 decoder38_1(OP[2:0],EN);
   NOR nor1(FW,A,B,EN[4]);
   DIV div1(FW,A,B,EN[3]);
   AND and1(FW,A,B,EN[0]);
   OR or1(FW,A,B,EN[1]);
   ADD add1(FW,CF,A,B,EN[2]);
   SUB sub1(FW,CF,A,B,EN[6]);
   CMP cmp1(FW,A,B,EN[7]);//slt
```

```
SLL sll1(FW,A,B,EN[5]);
      assign ZF=(F==32'b0);//F==0
      assign OF=A[SIZE-1]^B[SIZE-1]^F[SIZE-1]^CF;//overflow
      assign SF=F[SIZE-1];//sign
      assign PF=~^F;//odd or even of F's 1
   endmodule
   module NOR(FW,A,B,EN);
      parameter N=32;
      input [N-1:0] A,B;
      input EN;
      output reg [N-1:0] FW;
      always @(A,B,EN) begin
          if(EN==1) FW <= \sim (B|A);
          else FW<=32'bz;
      end
   endmodule
   ALU 的各个操作例如上文 NOR 是由-,*,/等直接封装成模块实现的,加法器、减法器、比较器是由二级进位
先行加法器实现的,实现代码在之前的实验ALU Design 中己上传,在此省略
   module EX MEM(
      input wire[31:0] alu_output,WriteDataE,
      input wire[4:0] WriteRegE,
      input clk, rst,
      input wire RegWriteE, MemtoRegE, MemWriteE,
      output reg [31:0] ALUOutM, WriteDataM,
      output reg [4:0] WriteRegM,
      output reg RegWriteM, MemtoRegM, MemWriteM
   );
       initial begin
         ALUOutM <= 32'b0;
         WriteDataM <= 32'b0;
         WriteRegM<=4'b0;
         RegWriteM<=1'b0;
         MemtoRegM<=1'b0;</pre>
         MemWriteM<=1'b0;</pre>
      end
```

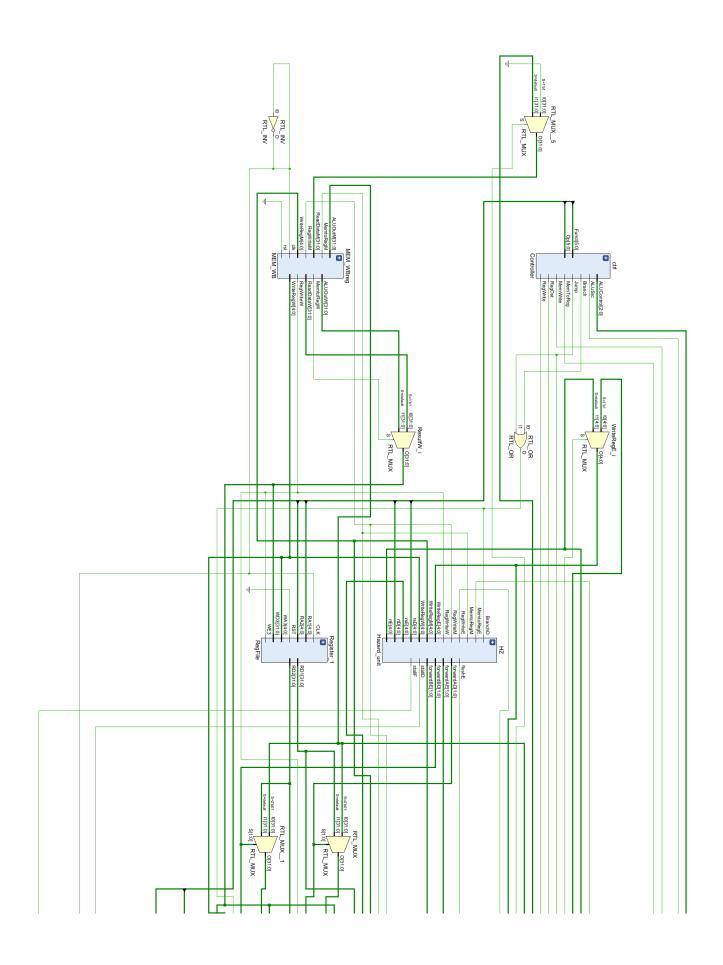
always@(posedge clk) begin

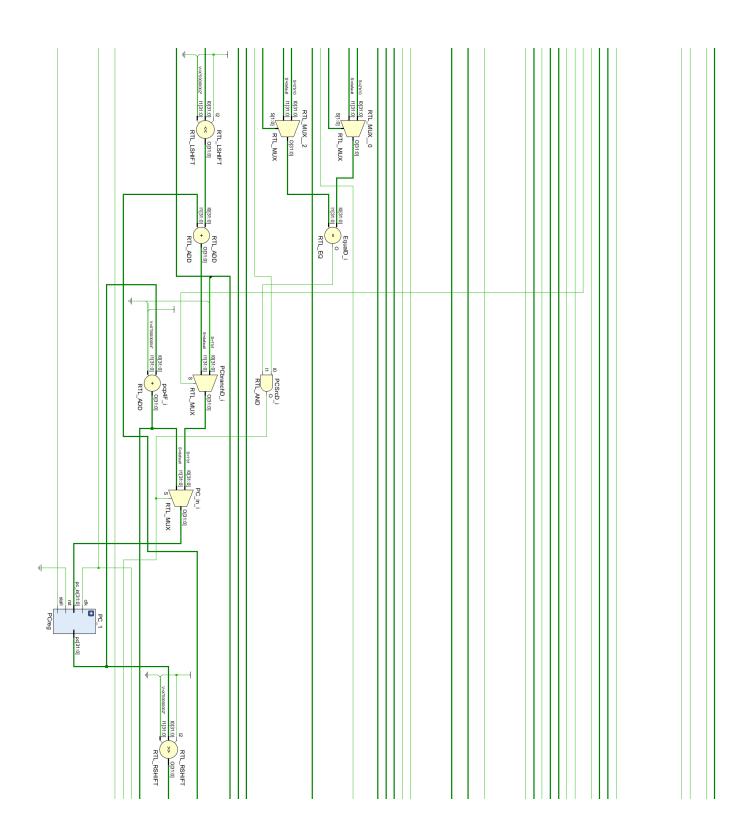
```
ALUOutM <= alu_output;
        WriteDataM <= WriteDataE;</pre>
        WriteRegM <= WriteRegE;</pre>
        RegWriteM <= RegWriteE;</pre>
        MemtoRegM <= MemtoRegE;</pre>
        MemWriteM <= MemWriteE;</pre>
   end
{\tt endmodule}
module Dmemory(Data_out,clk,R_W,Addr,Data_in);
   output [31:0] Data_out;
   input clk;
   input R_W;
   input [31:0] Addr;
   input [31:0] Data_in;
   reg [31:0] data_memory[1023:0];
   integer i;
   initial begin
        for(i=0;i<1024;i=i+1) data_memory[i]=32'b0;</pre>
   end
   assign Data_out = R_W?32'b0:data_memory[Addr];
   always@(posedge clk) begin
        if(R_W)
            data_memory[Addr] <= Data_in;</pre>
   end
{\tt endmodule}
```

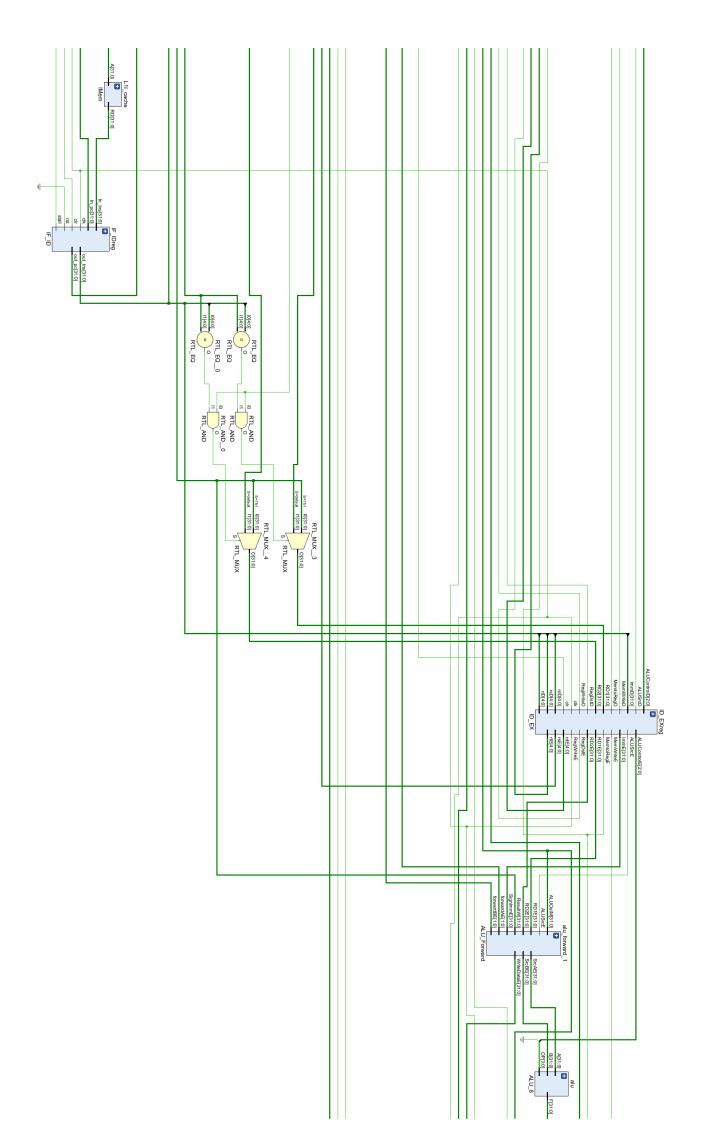
```
module MEM_WB(
    //input
    input [31:0] ALUOutM,ReadDataM,
    input clk,rst,
    input [4:0] WriteRegM,
    input RegWriteM,MemtoRegM,
```

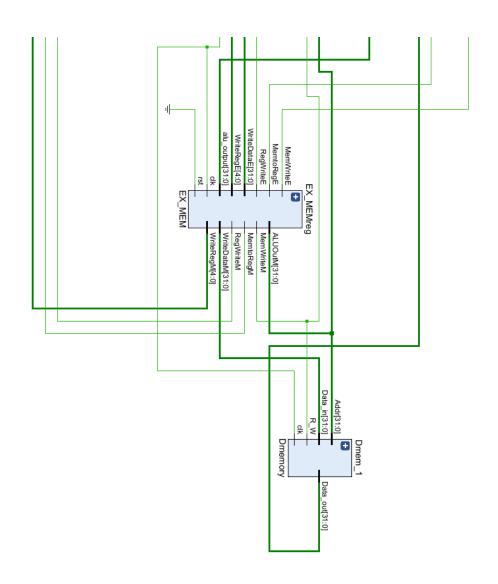
```
//output
   output reg[31:0] ReadDataW,ALUOutW,
   output reg[4:0] WriteRegW,
   output reg RegWriteW,MemtoRegW
);
   initial begin
      ALUOutW <= 32'b0;
      ReadDataW <= 32'b0;</pre>
      WriteRegW <= 5'b0;
      RegWriteW<=1'b0;</pre>
      MemtoRegW<=1'b0;</pre>
   end
   always@(posedge clk ) begin
      ALUOutW <= ALUOutM;
      ReadDataW <= ReadDataM;</pre>
      WriteRegW <= WriteRegM;
      RegWriteW <= RegWriteM;</pre>
      MemtoRegW <= MemtoRegM;</pre>
   end
endmodule
module Hazard_unit(
   input [4:0] rsD,rtD,rsE,rtE,WriteRegE,WriteRegM,WriteRegW,
  input BranchD,MemtoRegE,RegWriteE,MemtoRegM,RegWriteM,RegWriteW,
  output reg stallF, stallD, flushE,
  output reg [1:0] forwardAE, forwardBE, forwardAD, forwardBD
);
   always @(*) begin//priority:M,W,E
       if ((rsE!=0) && (rsE==WriteRegM) && RegWriteM)
           forwardAE<=2'b10;</pre>
       else if ((rsE!=0) && (rsE==WriteRegW) && RegWriteW)
           forwardAE<=2'b01;</pre>
       else forwardAE<=2'b00;</pre>
   end
   always @(*) begin
       if ((rtE!=0) && (rtE==WriteRegM) && RegWriteM)
           forwardBE<=2'b10;</pre>
       else if ((rtE!=0) && (rtE==WriteRegW) && RegWriteW)
```

```
forwardBE<=2'b01;</pre>
       else forwardBE<=2'b00;
   end
   wire lwstall,branchstall;
   //handle decode data dependency for branch read regfile
   //result in WB, alu ins MEM,OK
   //result of alu ins in EX, lw in MEM, stall 1 cycle
   always @(*)begin
       if((rsD!=5'b0)&&(rsD==WriteRegM)&&RegWriteM)
           forwardAD<=2'b01;</pre>
       else if((rsD!=5'b0)&&(rsD==WriteRegW)&&RegWriteW)
           forwardAD<=2'b10;</pre>
       else forwardAD<=2'b00;
   end
   always @(*)begin
       if((rtD!=5'b0)&&(rtD==WriteRegM)&&RegWriteM)
           forwardBD<=2'b01;</pre>
       else if((rtD!=5'b0)&&(rtD==WriteRegW)&&RegWriteW)
           forwardBD<=2'b10;</pre>
       else forwardBD<=2'b00;
   end
   //lw ins doesn't finish rd data til end of MEM stage
   //e.g. lw produce $50 at end of MEM but next and need $50 at start EX(same cycle) ,STALL!
   //load or branch stall
   assign lwstall=((rsD==rtE)||(rtD==rtE))&&MemtoRegE;
   assign branchstall=(BranchD && RegWriteE && (WriteRegE==rsD || WriteRegE==rtD) )||
                      (BranchD && MemtoRegM && (WriteRegM==rsD || WriteRegM==rtD) );
   always @(*)begin
       {stallF,stallD,flushE}<={lwstall||branchstall,lwstall||branchstall,lwstall||branchstall};
   end
endmodule 实验代码的仿真电路图:
```









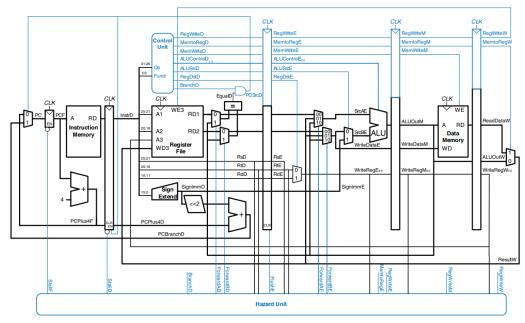


Figure 7.58 Pipelined processor with full hazard handling

本流水线处理器在上图的改动是:

将 PCSrcD=((BranchD|JumpD)&EqualD);, 判断多增加一个 jump 选项,由于 jump 指令使用 imm[15:0]跳转,所以此时 equalD 一定是 1,满足 PCSrcD=1,在 Hazard_unit 中,入口 BranchD= ControlUnit 里面的 BranchD|JumpD

所有级寄存器实现是简单的存贮与读取,而存储、逻辑、运算单元与单周期 CPU 基本一致。主要是 Hazard_Unit 单元的设计

首先涉及到的是数据的转发,

对于 Branch,如果判断数据 WB 阶段,前半周期就会写到寄存器,后半周期读取,没有冒险存在,但是在 ID 阶段收到 MEM 的数据转发,用于判断 EqualD。如果判断数据在 EX 阶段,或者是 lw 的结果,那么流水线必须停顿至数据准备完成。

所以判断逻辑为:

ForwardAD = (rsD != 0) AND (rsD == WriteRegM) AND RegWriteM

ForwardBD = (rtD != 0) AND (rtD ==WriteRegM) AND RegWriteM

说明 branch 的 rs, rd 与 MEM 阶段的 rs, rd 地址相同

lwstall= ((rsD==rtE)||(rtD==rtE))&&MemtoRegE

如果 ID 与 EX 操作数的调用有重叠且 EX 准备向 reg 中写入(说明是 lw),有 RAW 冒险,必须要停顿 1 cycle,

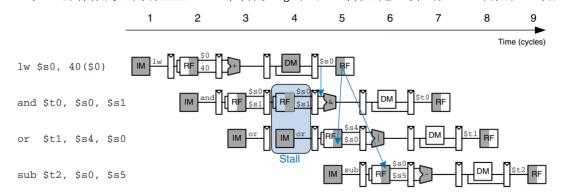


Figure 7.52 Abstract pipeline diagram illustrating stall to solve hazards

对于 branch lw 连续使用,则会停顿两个周期

branchstall =BranchD AND RegWriteE AND (WriteRegE == rsD OR WriteRegE == rtD)

BranchD AND MemtoRegM AND (WriteRegM == rsD OR WriteRegM== rtD)

如果 ID 是 branch, EX 阶段有向 branch 的 rs, rd 写入操作或者 lw 在 MEM 阶段准备写入 rs, rd 则 branch stall,

StallF = StallD = FlushE = Iwstall OR branchstall

IF/ID 寄存器与 PC 取指必须停顿,ID/EX 清空为 nop 气泡

对于 ALU 运算,如果所需操作数在 MEM, WB 阶段,必须进行转发 所以这里的多路选择器判断逻辑为

以下为 forwardBE 判断逻辑, forwardAE 与之相同, rt 变为 rs 即可 if ((rtE!=0) && (rtE==WriteRegM) && RegWriteM)

forwardBE<=2'b10:

else if ((rtE!=0) && (rtE==WriteRegW) && RegWriteW)

forwardBE<=2'b01;

else forwardBE<=2'b00;

即最先选择 MEM 的转发,

之后选择 WB 的转发,最后选择当前读出来的寄存器数,是有优先级的考虑的

还有一个改动逻辑是

ID/EX 寄存器读入寄存器的转发,

如下代码为例

addi \$3,\$2,20

sw \$3,3

addi \$4, \$2, 4

add \$4, \$2, \$3

由于 ID/EX 寄存器读入的时机为周期前半段,上段代码\$3 在时钟末期写入寄存器,所以必须要将 ResultW 提前半个周期转发给 ID/EX 读入口,逻辑为:

- . RD1 ((RegWriteW&&rs==WriteRegW)?ResultW:SrcA),
- .RD2((RegWriteW&&rt==WriteRegW)?ResultW:SrcB),

课本中的理想化的寄存器堆采用前半个周期写后半个周期读,所以不存在这种数据冒险。

实际实现中,寄存器堆在 WB 级的结束时才写入值,所以对于 **Iw 后跟 branch** 指令在 **Iw 的 WB 周期**读取的寄存器的值是旧值,存在数据冒险在 id 级产生两个旁路信号,选择正确的数据源

逻辑为:

```
if((rsD!=5'b0)&&(rsD==WriteRegM)&&RegWriteM)
    forwardAD<=2'b01;
else if((rsD!=5'b0)&&(rsD==WriteRegW)&&RegWriteW)
    forwardAD<=2'b10;
else forwardAD<=2'b00;</pre>
```

基础功能测试: main: addi \$2,\$2,10 sw \$2,2 addi \$3,\$2,20 sw \$3,3 addi \$4, \$2, 4 add \$4, \$2, \$3 sw \$4, 4 addi \$0,\$4,44 1w \$4, 4(\$2) sw \$0,8(\$2) sub \$2, \$4, \$3 and \$1, \$2, \$4 or \$5, \$6, \$2 sub \$7,\$2,\$8 or \$2,\$4,\$3 and \$2, \$4, \$3 s1t \$2,\$4,\$3 1w \$0,40(\$0) and \$1, \$0, \$2 or \$3,\$4,\$0 sub \$5, \$0, \$6 or \$3,\$4,\$0 beq \$3, \$3, equ 1w \$2,0(\$3) equ: beq \$3,\$4,exit or \$2, \$4, \$3 and \$2,\$4,\$3 s1t \$2, \$4, \$3 sw \$2,0(\$3) exit: j main 以上为一段包含所有指令的循环代码, 翻译为机器码为 2042000a ac020002

20430014 ac030003 20440004 00432020

ac040004

2080002c

8c440004

ac400008

00831022

00440824

00c22825

00483822

00831025

00831024

0083102a

8c000028

00020824

00801825

00062822

00801825

10630001

8c620000

10640004

00831025

00831024

0083102a

ac620000

08000000

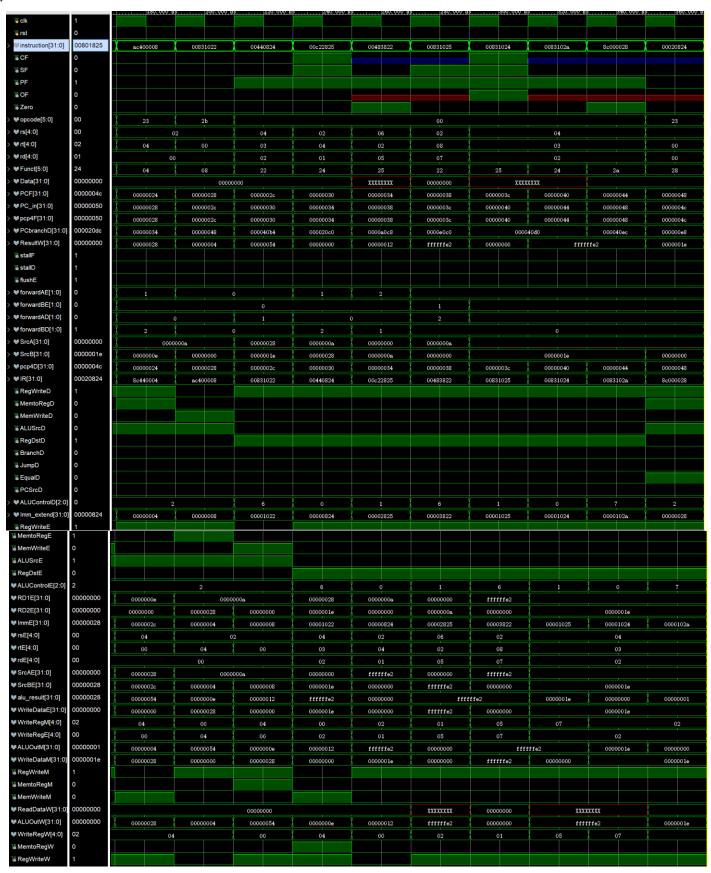
Address	Code	Basic	
0x00000000	0x2042000a	addi \$2,\$2,10	2: addi \$2,\$2,10
0x00000004	0xac020002	sw \$2,2(\$0)	3: sw \$2,2
0x00000008	0x20430014	addi \$3,\$2,20	4: addi \$3,\$2,20
0x0000000c	0xac030003	sw \$3,3(\$0)	5: sw \$3,3
0x00000010	0x20440004	addi \$4,\$2,4	6: addi \$4,\$2,4
0x00000014	0x00432020	add \$4, \$2, \$3	7: add \$4, \$2, \$3
0x00000018	0xac040004	sw \$4,4(\$0)	8: sw \$4, 4
0x0000001c	0x2080002c	addi \$0,\$4,44	9: addi \$0,\$4,44
0x00000020	0x8c440004	lw \$4,4(\$2)	10: lw \$4,4(\$2)
0x00000024	0xac400008	sw \$0,8(\$2)	11: sw \$0,8(\$2)
0x00000028	0x00831022	sub \$2,\$4,\$3	12: sub \$2,\$4,\$3
0x0000002c	0x00440824	and \$1, \$2, \$4	13: and \$1,\$2,\$4
0x00000030	0x00c22825	or \$5, \$6, \$2	14: or \$5,\$6,\$2
0x00000034	0x00483822	sub \$7,\$2,\$8	15: sub \$7,\$2,\$8
0x00000038	0x00831025	or \$2,\$4,\$3	16: or \$2,\$4,\$3
0x0000003c	0x00831024	and \$2, \$4, \$3	17: and \$2,\$4,\$3
0x00000040	0x0083102a	slt \$2,\$4,\$3	18: slt \$2,\$4,\$3
0x00000044	0x8c000028	lw \$0,40(\$0)	19: lw \$0,40(\$0)
0x00000048	0x00020824	and \$1, \$0, \$2	20: and \$1,\$0,\$2
0x0000004c	0x00801825	or \$3,\$4,\$0	21: or \$3,\$4,\$0
0x00000050	0x00062822	sub \$5, \$0, \$6	22: sub \$5,\$0,\$6
0x00000054	0x00801825	or \$3,\$4,\$0	23: or \$3,\$4,\$0
0x00000058	0x10630001	beq \$3,\$3,1	24: beq \$3,\$3, equ
0x0000005c	0x8c620000	lw \$2,0(\$3)	25: lw \$2,0(\$3)
0x00000060	0x10640004	beq \$3,\$4,4	27: beq \$3,\$4, exit
0x00000064	0x00831025	or \$2,\$4,\$3	28: or \$2,\$4,\$3
0x00000068	0x00831024	and \$2, \$4, \$3	29: and \$2,\$4,\$3
0x0000006c	0x0083102a	slt \$2,\$4,\$3	30: slt \$2,\$4,\$3
0x00000070	0xac620000	sw \$2,0(\$3)	31: sw \$2,0(\$3)
0x00000074	0x08000000	j 0x00000000	33: j main

clk										\$ <u>1</u> 00.000.µ	
not	0										
instruction[31:0]	ac020002	/V		<u> </u>			V				
	Z	2042000a	ac020002	20430014	ac030003	20440004	00432020	ac040004	2080002c	8c440004	ac400008
CF	0										
SF	1										
PF	l '										
OF	X										
Zero	1				 					ļ	
opcode[5:0]	08	00	08	2b	08	2 b	80	00	2b	80	23
rs[4:0]	02	00	02	00	02	00	c	2	00	04	<u> </u>
rt[4:0]	02	00		02	, 0	3	04	03	04	00	04
rd[4:0]	00	(00			04		0	00
Funct[5:0]	0a	00	0a	02	14	03	04	20	04	2c	04
Data[31:0]	00000000							00000000			
PCF[31:0]	00000004	00000000	00000004	80000000	0000000c	00000010	00000014	00000018	0000001c	00000020	00000024
PC_in[31:0]	00000008	00000004	80000000	0000000c	00000010	00000014	00000018	0000001c	00000020	00000024	00000028
pcp4F[31:0]	8000000	00000004	80000000	0000000c	00000010	00000014	00000018	0000001c	00000020	00000024	00000028
PCbranchD[31:0]	0000002c	00000000	0000002c	00000010	0000005c	0000001c	00000024	00008098	0000002c	000000d0	00000034
ResultW[31:0]	00000000			00000000	1	0000000a	0000002	0000001e	0000003	0000000e	00000028
stallF	0										
stallD	0										
flushE	0										
forwardAE[1:0]	0										₩
forwardBE[1:0]	0	—		0	V	1		· · · · · · · · · · · · · · · · · · ·	0	V	1
forwardAD[1:0]	0	<u> </u>	. 0		2	0	2		0	2	
			. 0		1			0		1	
orwardBD[1:0]	0	<u> </u>			0			2	1	0	2
SrcA[31:0]	00000000			00000000			0000	000a	0000	00000	
SrcB[31:0]	00000000					00000000					0000000e
pcp4D[31:0]	00000004	00000000	00000004	8000000	0000000с	00000010	00000014	00000018	0000001c	00000020	00000024
R[31:0]	2042000a	00000000	2042000a	ac020002	20430014	ac030003	20440004	00432020	ac040004	2080002c	8c440004
RegWriteD	1										
MemtoRegD	0										
MemWriteD	0										
ALUSrcD	1										
RegDstD	0										
BranchD	0										
JumpD	0										
	1										
EqualD											
EqualD PCSrcD	0										
PCSrcD	0	5						,			
PCSrcD ALUControlD[2:0]	2	5	0000000	0000000	0000014	00000002	/	2	0000004	000002	0000000
PCSrcD ALUControlD[2:0] lmm_extend[31:0]	2	5 00000000	0000000a	0000002	00000014	0000003	0000004	2 00002020	0000004	000002c	0000004
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE	2		0000000a	0000002	0000014	00000003	/	· · · · · · · · · · · · · · · · · · ·	0000004	0000002c	0000004
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE	2 00000000a 1		0000000a	0000002	0000014	0000003	/	· · · · · · · · · · · · · · · · · · ·	0000004	0000002c	0000004
PCSrcD ALUControID[2:0] mm_extend[31:0] RegWriteE emtoRegE emWriteE	2] 0000000a 1		0000000a	0000002	0000014	0000003	/	· · · · · · · · · · · · · · · · · · ·	0000004	0000002c	0000004
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emWriteE LUSrcE	2 00000000a 1		0000000a	0000002	0000014	0000003	/	· · · · · · · · · · · · · · · · · · ·	0000004	000002c	0000004
PCSrcD ALUControlD[2:0] Rm_extend[31:0] RegWriteE emtoRegE emWriteE _USrcE egDstE	2 0000000a 1 1 0 1	0000000		0000002	0000014	0000003	/	00002020		000002c	0000004
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emWriteE LUSrcE egDstE LUControlE[2:0]	2 0000000a 1 1 0 1 0		0000000a	0000002		0000003	/	00002020	2		
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emWriteE LUSrcE egDstE LUControlE[2:0] D1E[31:0]	2 0000000a 1 1 0 1 0 2 0000000a	0000000		0000002	0000000	0000003	/	00002020	2 20000a	0000000	000000
PCSrcD ALUControlD[2:0]	2 0000000a 1 1 0 0 1 0 2 0000000a 00000028	0000000		0000002	0000000	0000003	/	00002020	2	0000000	
CSrcD LUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emWriteE LUSrcE egDstE LUControlE[2:0] D1E[31:0] mE[31:0]	2 0000000a 1 1 0 1 0 2 0000000a 00000028	0		0000002	0000000	0000003	/	00002020	2 20000a	0000000	000000
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emWriteE LUSrcE egDstE LUControlE[2:0] D1E[31:0] D2E[31:0] mmE[31:0]	2 0000000a 1 1 0 0 1 0 2 0000000a 00000028	0	5		0000000		0000004	0000000	2 00000a 0000001e	0000000	000000
PCSrcD ALUControlD[2:0] ALUControlD[2:0] ALUControlD[2:0] ALUCONTrolE[2:0]	2 0000000a 1 1 0 1 0 2 0000000a 00000028	0	5	0000000a	00000000	00000014 02	0000004	0000000	2 0000a 0000001e 00002020	0000000 0000004	0000000
PCSrcD ALUControlD[2:0] ALUControlD[2:0] ALUControlD[2:0] ALUCONTrolE[2:0]	2 0000000a 1 1 0 1 0 2 0000000a 00000028 00000004 02	0	5	0000000a	00000000 00000000 00000002 00	00000014 02	0000004	0000004	2 00000 0000001e 00002020	0000000 0000004 00	0000000
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtWriteE .USrcE egDstE .UControlE[2:0] D1E[31:0] D2E[31:0] pmE[31:0] E[4:0] E[4:0]	2 0000000a 1 1 0 1 0 2 0000000a 00000028 00000004 02 04	0	5	0000000a	00000000	00000014 02	0000004	00000004	2 00000 0000001e 00002020 02	0000000 0000004 00	0000000 000000 00000002 04
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtWriteE .USrcE egDstE .UControlE[2:0] D1E[31:0] D2E[31:0] immE[31:0] E[4:0] E[4:0] E[4:0] cAE[31:0]	2 0000000a 1 1 0 2 0000000a 00000028 00000004 02 04 00	0	5	0000000a	00000000	00000014	0000003	00000004	2 000000 0000001e 00002020 02 03 04	00000000 00000004 00	0000000 000000 0000002 04 00
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtWriteE .USrcE egDstE .UControlE[2:0] D1E[31:0] D2E[31:0] pmE[31:0] E[4:0] E[4:0] E[4:0] cAE[31:0] cBE[31:0]	2 0000000a 1 1 0 2 0000000a 00000028 00000004 02 04 00 0000000a	0	5	0000000a 02 00000000 00000000a	00000000 00000000 00000002 00	00000014 02 0000000a	00000004 00000003 00 00 0000000 0000000	00000004 004	2 0000001e 000002020 02 03 04 0000001e	00000000 00000004 00 04 00000000	0000000 000000 0000002 04 00 0000002
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtoRegE emtwriteE LUSrcE eguDEE LUSrcE guDE[31:0] poeE[31:0] poeE[31:0] poeE[4:0] poeE[4:0] poeE[31:0]	2 0000000a 1 1 0 0 2 00000004 00000000 a 0000000 a 0000000 a 00000000	0	5 0000000 00 00 0000000	0000000a 02 00000000 0000000a	00000000 00000000 00000002 00 22 00 000000	00000014 02 0000000a 00000014	00000004 00000003 06 00000000 00000000 00000003	00000004 00000004 04	2 2 0000001e 0000001e 000000000000000000	00000000 00000004 00 04 00000000 0000000	0000000 000000 0000002 04 00 0000002 0000002
CSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtoRegE emtwriteE .USrcE eg.USrcE eg.USrcE[31:0] DDE[31:0] DDE[31:0] DE[4:0] E[4:0] E[4:0] cAE[31:0] cBE[31:0] cresult[31:0] result[31:0] result[31:0]	2 0000000a 1 1 0 0 2 00000004 00000000 0 0000000 0 0000000 0 000000	0	5 0000000 00 00 0000000 0000000	0000000a 02 00000000 0000000a	00000000 00000000 00000002 00 22 00 000000	00000014 02 0000000a 00000014 0000001e	00000004 00000003 00 0000000 00000003 00000003 00000001e	00000004 00000004 00000004 00000004 000000	2 0000001e 000002020 02 03 04 0000001e	00000000 00000004 00 04 00000000 0000000	0000000 000000 0000002 04 00 0000002 0000002
CSrcD LUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtWriteE .USrcE egDetE .UControlE[2:0] D1E[31:0] D2E[31:0] E[4:0] E[4:0] E[4:0] cAE[31:0] cBE[31:0] cPE[31:0] criteRegM[4:0] riteRegM[4:0]	2 0000000a 1 1 0 1 0 2 0000000a 00000004 0000000a 00000004 00000000	0	5 0000000 00 00 0000000 0000000 0000000	0000000a 02 00000000 0000000a	00000000 000000002 000 000 000 000 000000	00000014 02 0000000a 00000014 0000001e	00000004 00000003 06 0000000 00000003 00000003 0000001e	00000004 00000004 04	2 2 0000001e 0000001e 000000000000000000	00000000 00000004 00 04 00000000 0000000	0000000 000000 0000002 04 00 0000002 0000002
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtWriteE .USrcE egDstE .UControlE[2:0] D1E[31:0] D2E[31:0] E[4:0] E[4:0] E[4:0] E[4:0] cAE[31:0] cBE[31:0] criteRegM[4:0] riteRegM[4:0] riteRegM[4:0]	2 0000000a 1 1 0 0 1 0 0 00000028 00000004 00000004 00000004 000000028 00 04	0	5 0000000 00 00 0000000 0000000 0000000	0000000a 02 0000000a 0000000a	00000000 000000002 000 000 22 00000002 000000	00000014 02 0000000a 0000014 0000001e 00000000	00000004 00000003 00000000 00000003 00000003 000000	00000004 00000004 04 00000004 00000004 000000	2 000001e 000001e 00002020 02 03 04 000001e 00000028 0000001e	00000000 00000004 00 04 00000000 0000000	00000000 0000000 00000000 04 00000000 000000
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtWriteE .USrcE egDstE .UControlE[2:0] D1E[31:0] D2E[31:0] pmE[31:0] E[4:0] E[4:0] eAE[31:0] cAE[31:0] cFE[31:0] cF	2 0000000a 1 1 0 1 0 2 00000004 00000004 00000004 00000004 000000	0	5 0000000 00 00 0000000 0000000 0000000	0000000a 022 0000000a 0000000a	00000000 000000002 000 000 000 000 000000	00000014 02 0000000a 00000014 0000001e 00000000	00000004 00000003 00000000 00000003 00000003 000000	00000004 00000004 04 00000004 00000000 000000	2 0000001e 0000001e 000002020 03 04 0000001e 0000001e 0000001e 0000001e	00000000 00000004 00 04 00000004 0000004 000000	0000000 0000000 04 00 0000000 0000000 000000
PCSrcD ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtWriteE .USrcE egDstE .UControlE[2:0] D1E[31:0] D2E[31:0] E[4:0] E[4:0] E[4:0] E[4:0] cAE[31:0] cBE[31:0] criteDataE[31:0] friteDataE[31:0] riteRegM[4:0] riteRegE[4:0] .UOutM[31:0]	2 0000000a 1 1 0 1 0 2 00000004 00000004 00000004 000000028 00 00000028 00 04 000000054 000000054 000000000000	0	5 0000000 00 00 0000000 0000000 0000000	0000000a 02 0000000a 0000000a	00000000 000000002 000 000 22 00000002 000000	00000014 02 0000000a 0000014 0000001e 00000000	00000004 00000003 00000000 00000003 00000003 000000	00000004 00000004 04 00000004 00000004 000000	2 000001e 000001e 00002020 02 03 04 000001e 00000028 0000001e	00000000 00000004 00 04 00000000 0000000	0000000 0000000 04 00 0000000 0000000 000000
ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtVriteE LUSrcE egDstE LUControlE[2:0] D1E[31:0] D2E[31:0] D2E[31:0] D2E[4:0] E[4:0] E[4:0] E[4:0] rcAE[31:0] rricBe[31:0] /riteDataE[31:0] /riteRegM[4:0] /riteRegE[4:0] LUOutM[31:0] LUOutM[31:0] egWriteM	2 0000000a 1 1 0 0 1 0 0 0000000a 00000000	0	5 0000000 00 00 0000000 0000000 0000000	0000000a 022 0000000a 0000000a	00000000 000000002 000 000 22 00000002 000000	00000014 02 0000000a 00000014 0000001e 00000000	00000004 00000003 00000000 00000003 00000003 000000	00000004 00000004 04 00000004 00000000 000000	2 0000001e 0000001e 000002020 03 04 0000001e 0000001e 0000001e 0000001e	00000000 00000004 00 04 00000004 0000004 000000	0000000 0000000 04 00 0000000 0000000 000000
ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtVriteE LUSrcE egDstE LUControlE[2:0] D1E[31:0] D2E[31:0] D2E[31:0] D2E[4:0] E[4:0] E[4:0] E[4:0] rcAE[31:0] rricBet[31:0] /riteDataE[31:0] /ritePataE[31:0] LUOutM[31:0] LUOutM[31:0] egWriteM emtoRegM	2 0000000a 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0	0	5 0000000 00 00 0000000 0000000 0000000	0000000a 022 0000000a 0000000a	00000000 000000002 000 000 22 00000002 000000	00000014 02 0000000a 00000014 0000001e 00000000	00000004 00000003 00000000 00000003 00000003 000000	00000004 00000004 04 00000004 00000000 000000	2 0000001e 0000001e 000002020 03 04 0000001e 0000001e 0000001e 0000001e	00000000 00000004 00 04 00000004 0000004 000000	0000000 0000000 04 00 0000000 0000000 000000
ALUControlD[2:0] mm_extend[31:0] RegWriteE emtoRegE emtv/riteE LUSrcE egDstE LUControlE[2:0] D1E[31:0] D2E[31:0] D2E[31:0] DE[4:0] E[4:0] E[4:0] E[4:0] rocAE[31:0] riteDataE[31:0] /ritePataE[31:0] /riteRegM[4:0] /riteRegE[4:0] LUOutM[31:0] /riteDataM[31:0] egWriteM emtoRegM emtvriteM	2 0000000a 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0	0	5 0000000 00 00 0000000 0000000 0000000	0000000a 022 0000000a 0000000a	00000000 000000002 000 000 22 00000002 000000	00000014 02 0000000a 00000014 0000001e 00000000	00000004 00000003 00000000 00000003 00000003 000000	00000004 00000004 04 00000004 00000000 000000	2 0000001e 0000001e 000002020 03 04 0000001e 0000001e 0000001e 0000001e	00000000 00000004 00 04 00000004 0000004 000000	0000000 0000000 04 00 0000000 0000000 000000
PCSrcD ALUControlD[2:0] Imm_extend[31:0] RegWriteE lemtoRegE lemtwriteE LUSrcE egDstE LUControlE[2:0] D1E[31:0] D2E[31:0] D2E[31:0] D2E[31:0] D2E[4:0] E[4:0] E[4:0] IE[4:0] IF[4:0] IF	2 0000000a 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0	0	5 0000000 00 00 0000000 0000000 0000000	0000000a 022 0000000a 0000000a	00000000 000000002 000 000 22 00000002 000000	00000014 02 0000000a 00000014 0000001e 00000000	00000004 00000003 00000000 00000003 00000003 000000	00000004 00000004 04 00000004 00000000 000000	2 0000001e 0000001e 000002020 03 04 0000001e 0000001e 0000001e 0000001e	00000000 00000004 00 04 00000004 0000004 000000	0000000 0000000 04 00 0000000 0000000 000000
PCSrcD ALUControlD[2:0] imm_extend[31:0] RegWriteE lemtoRegE lemtWriteE LUSrcE legDstE LUControlE[2:0] D1E[31:0] D2E[31:0] D2E[31:0] DE[4:0] IE[4:0] IE[4:0] IrcAE[31:0] IvriteDataE[31:0] VriteRegE[4:0] LUOutM[31:0] VritePataM[31:0] VriteDataM[31:0] VriteDataM[31:0] VriteDataM[31:0] VriteDataM[31:0] IcegWriteM IdemtoRegM IdemtWriteM IdemtoRegM Idemto	2 0000000a 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0	0	5 0000000 00 00 0000000 0000000 0000000	0000000a 022 0000000a 0000000a	00000000 000000002 000 000 22 00000002 000000	00000014 02 0000000a 00000014 0000001e 00000000	00000004 00000003 00000000 00000003 00000003 000000	00000004 00000004 04 00000004 00000000 000000	0000001e 0000001e 00002020 02 03 04 0000001e 00000028 0000001e	00000000 00000004 00 04 00000004 0000004 000000	00000000000000000000000000000000000000
PCSrcD ALUControlD[2:0] imm_extend[31:0] RegWriteE lemtoRegE lemtWriteE LUSrcE legDstE LUControlE[2:0] D1E[31:0] D2E[31:0] D2E[31:0] SE[4:0] SE[4:0] SE[4:0] SE[4:0] SE[4:0] VritePataE[31:0] VriteRegE[4:0] LUCutM[31:0] VritePataM[31:0] VriteDataM[31:0]	2 0000000a 1 1 0 1 0 2 00000004 00000004 00000004 00000004 000000	0	5 0000000 00 00 0000000 0000000 0000000	0000000a 02 0000000a 0000000a 0000000a	00000000 000000002 000 000 22 00000002 000000	00000014 02 0000000a 00000014 00000000 2 000000002 00000000a	00000004 00000003 00000000 00000003 00000003 000000	00000004 00000004 04 00000004 00000000 000000	00000000 0000000000000000000000000000	00000000 00000004 00 04 00000000 0000000	00000000 00000020 04 00 0000020 00000020 00000054
PCSrcD ALUControlD[2:0] imm_extend[31:0] RegWriteE lemtoRegE lemtWriteE LUSrcE legDstE LUControlE[2:0] .D1E[31:0] .D2E[31:0] .D2E[31:0] .DE[4:0] .E[4:0] .E[4:0] .rcAE[31:0] .vriteDataE[31:0] .vriteDataE[31:0] .vritePagM[4:0] .vritePagM[4:0] .vriteDataM[31:0] .uresult[31:0] .	2 0000000a 1 1 0 0 2 00000004 00000004 000000054 000000054 00000000	0	5 0000000 00 00 0000000 0000000 0000000	0000000a 02 00000000 0000000a 0000000a	00000000 000000002 000 000 22 00000002 000000	00000014 02 0000000a 00000014 00000000 2 000000002 00000000a	00000004 00000003 000 00000000 00000003 0000000	00000004 00000004 04 00000004 00000000 000000	2 2 0000a 0000001e 00002020 22 03 04 0000001e 00000001e 04 0000000e 000000000	00000000 00000004 00 04 00000000 0000000	00000000 0000000 00000002 000000000 000000

第一个是 addi 指令,在第二周期可以看到 rs=2, rt=2, 第三周期看到 immE=0xa, SrcBE=0xa, aluoutputE 所得为 0xa, 说明流水线工作正常,i 型指令正常

第二个是 sw \$2,2,可以看到第三周期 rs=0, rt=2, imm=2, 第四周期 writeDataE=0xa, 说明把刚才的 addi 的值存进了 Dmem 中

第六周期 0x00432020 是 add \$4,\$2,\$3 指令, 第七周期中,rs=2,rt=3,rt=4,解码正确,按照代码,此时 \$2=10,\$3=30,那么第八周期时候 alu 为 0x1e+0xa=0x28, add 等 R 型指令工作正常,前面所述 WB 至 ID 转发逻辑正常



0xac400008 至 0x0083102a 均为 R 型指令 从解码、读取,运算的值来看都是正常的

0x8c000028 是 1w 指令

1w \$0,40(\$0)

and \$1, \$0, \$2

or \$3,\$4,\$0

sub \$5, \$0, \$6

k 1		360.000 ns	380.000 ns	400.000	ns	s.,,,,,,,,,,,,440,,000,p	s460.,000,n	s480.000 t	is 500.000.	ns. 520
t 0										
	0831025	00020824	00801	825	00062822	00801825	10630001	8c6:	20000	10640004
Z 0										
1	-									
×	1									
1	1									
ode[5:0] 04	1	23			00				04	00
:0] 03	3		00		04	00	04		03	00
:0] 04	- 1 - 1	00	02		00	06	00		03	00
:0] 00	- 1 ≥1	00	01		03	05	03			· ·
nct[5:0] 04	0000000	28	24		25	22	25	<u> </u>	01	00
	0000064	00000048	00000	040	00000050	00000054	00000058	0000	0005c	0000000
	0000074	00000048	00000		00000054	00000058	0000005c		00060	0000006
	0000068	0000004c	00000		00000054	00000058	0000005c		00060	0000006
oranchD[31:0] 00	0000074	000000e8	00002		000060e4	0000a0dc	000060ec	-	00060	0000000
ultW[31:0] 00	0000000	0000001e	00000000	00000001						
F 0	1									
0										
E 0										
ardAE[1:0] 0	=									1
vardBE[1:0] 0 vardAD[1:0] 0	-								v :	1
ardBD[1:0] 0	-	0	1	2	0	0		2	1	↓
	0000000	<u> </u>			00000	· · · · · · · · · · · · · · · · · · ·		0000001e		<u> </u>
	0000000	00000000	0000001e			00000		0000001e		
4D[31:0] 00	0000064	00000048	00000	04c	00000050	00000054	00000058	-	0005c	0000000
1:0] 10	0640004	8c000028	00020	824	00801825	00062822	00801825	1060	30001	0000000
WriteD 0										
ntoRegD 0										
WriteD 0										
SrcD 0	1									
DstD 0										
pD 0	1									
alD 1	1									
SrcD 1	1									
ControlD[2:0] 6	7	2	0		1	6	1		6	5
	0000004	00000028	00000	824	00001825	00002822	00001825	0000	00001	0000000
NriteE 1 oRegE 0									1	
WriteE 0										
SrcE 0										
OstE 1										
ControlE[2:0] 5	<u> </u>	7	2		0	1	6	1	O	6
	000000	- V			·	,			00000000	
	000000	0000001e	000000		00000001				V	·
[31:0] 000 :0] 00		0000102a	00000028	00000000	00000824	00001825	00002822	00001825	00000000	000000
0] 00		04	00	00	02	04	00	04	00	03
:0] 00	_	03	00		01	03	05	03	<u> </u>	03
•	000000	1			T			i i	00000000	
	000000	0000001e	00000028	00000000	00000001					
esult[31:0] 000	000000	00000001	00000028							000000
DataE[31:0] 000		0000001e	000000	000	00000001					
RegM[4:0] 03	_	02			00	01	03	05	03	00
RegE[4:0] 00	575	02	00		01	03	05	03	00	03
	000000	00000000	00000001	00000028						
WriteM 0	-	000000	le	000	00000	00000001				
toRegM 0										
WriteM 0										
	000000									000000
	000000	0000001e	00000000	00000001	00000028					
	- 1		02		·	0	01	03	05	03
eRegW[4:0] 00										
ntoRegW 0										

我们可以看到,在 0x0080125 的时候出触发了 stall,此时 EX 阶段的 alu_result 是 0x28,正是 40(\$0),它即将作为 Dmem 的 addr,可以看到,在停顿的第二周期, ID/EX 寄存器的内容全部被清零,例如 rsE,rtE,AluSrcE,MemtoRegE 等信号都是 0,它不与上一周期的 1w 的 ID/EX 寄存器值相同,再下一个周期,才开始解码 or 指令

从前面的代码我们知道,1w 指令前\$0 值为 44(0x2c),这个可以从图中第十周期 1w 的 A1uResultE 看出,那么在 1w 后调用的 add,or,sub 都是 1w 后加载的新值(\$0)=0, 说明 RAW 问题已经得到解决,数据的转发也是没有问题的。

之后是一段跳转指令

beq \$3, \$3, equ

1w \$2,0(\$3)

equ:

beq \$3,\$4, exit

or \$2, \$4, \$3

and \$2, \$4, \$3

s1t \$2, \$4, \$3

sw \$2,0(\$3)

exit:

j main

		460,000 B	s 480.000 ns	500.000	ns	940.000.1	ış 560.000 n	s ԵԾՄ. ՄՄՄ ns	600.000 h	s 620.000 n	s 640. VVV. no
₩ clk	1										
instruction[31:0]	00831025	. 10630001	8c620000		10640004	00831025	08000000	XXXXX	XXX	2042000a	ac020002
1⊌ CF	Z	100000			10010001	***************************************				2012000	43020002
™ SF	0										
™ PF	1										
% OF	X 1										
₩ opcode[5:0]	04	00	04		00	04	00	02		00	08
₩ rs[4:0]	03	. 04	03		00	03		00		· · ·	02
₩ rt[4:0]	04	. 00	03		00	04		00			0
₩ rd[4:0]	00	. 03							00		
₩ Funct[5:0]₩ Data[31:0]	04	. 25	01		00	04	i	. 00			0a
● PCF[31:0]	00000064	. 00000058	00000050		00000060	00000064	00000074	00000	0.78	00000000	00000004
₩ PC_in[31:0]	00000074	. 0000005c	00000060		00000064	00000074	00000078	00000		00000004	00000008
₩ pcp4F[31:0]	00000068	. 0000005с	00000060		00000064	00000068	00000078	00000	07c	00000004	8000000
₩ PCbranchD[31:0]	00000074	. 000060ec	00000060		00000000	00000074	X	00000	000		0000002c
W ResultW[31:0] StallF	00000000						000000	000			
stallD	0										
% flushE	0										
♥ forwardAE[1:0]	0		0		1)	
♥ forwardBE[1:0]	0		0		1				o		
# forwardAD[1:0] # forwardBD[1:0]	0	0	2	1				. 0			
■ SrcA[31:0]	00000000	00000000	2 0000001e	1	·		00000000			0	00000001
■ SrcB[31:0]	00000000	00000000	0000001e				00000000				0000001
■ pcp4D[31:0]	00000064	. 00000058	0000005c		00000000	00000064	00000000	00000	078	00000000	00000004
₩ IR[31:0]	10640004	. 00801825	10630001		00000000	10640004	00000000	08000	000	00000000	2042000a
™ RegWriteD	0										
MemtoRegD MemWriteD	0										
MemoviteD ALUSrcD	0										
™ RegDstD	0										
™ BranchD	1										
™ JumpD	0										
le EqualD	1										
₩ ALUControlD[2:0]	6	6 1	6		<u> </u>		5	2		5	
■ Imm_extend[31:0]	00000004	. 00001825	00000001		00000000	6 0000004		00000	000	5	0000000a
™ RegWriteE	1										
™ MemtoRegE	0										
I MemWriteE	0										
Te ALUSrcE Te RegDstE	0										
_	5	1 6	1	0	6	5	6	5	0	2	5
₩RD1E[31:0]	00000000	-1 -		<u>`</u>	- Y	00000		1 · · · · · · · · · · · · · · · · · · ·	·		1—— š
₩ RD2E[31:0]	00000000					00000					
₩ ImmE[31:0]	00000000	. 00002822	00001825	00000000	00000001	00000000	00000004			0000	
₩ rsE[4:0]	00	. 00	04	00	03	00	03	·		0	·i
rtE[4:0] rdE[4:0]	00	. 06	00		03	00	04	·	, 0	00	
₩ SrcAE[31:0]	00000000	. 05	US A			00000	0000			00	
♥ SrcBE[31:0]	00000000					00000					
₩ alu_result[31:0]	00000000					00000	0000				
₩ WriteDataE[31:0]	00000000					00000	0000				
₩ WriteRegM[4:0]	03	. 03	05	03	00	03	00	04			00
WriteRegE[4:0] ₩ ALUOutM[31:0]	00000000	. 05	03	00	03	00	00000000		0	0	
₩ WriteDataM[31:0]	00000000						00000000				
™ RegWriteM	0										
MemtoRegM	0										
MemWriteM	0										
■ ReadDataW[31:0] ■ ALUOutW[31:0]	00000000									00000000	
₩ ALUOutW[31:0] WriteRegW[4:0]	00000000	. 01	03	05	03	00	00000	000	04		-
MemtoRegW	0	1	i 1								
le RegWriteW	0										

beq \$3,\$3,equ 的机器码为 0x10630001,可以看到,在下一周期 beq 的 ID 阶段流水线停顿,这是因为 beq 用到的\$3 是上一个指令 or 的结果,此次指令分支要跳到 equ 处,即 10640001,而且,我们可以看到, lw 的 rs=3, rt=2,然而在解码阶段没有 rs=3, rt=2 出现,仅仅是 beq \$3,\$3,equ 的 rs=3, rt=3 后, IF/ID 寄存器被刷新,紧接着是 beq \$3,\$4,exit 解码出来的 rs=3,rt=4, lw 仅仅存在在 IF 阶段后就被清除了

0x8c620000 有一次停顿,其实这是因为 branch 要用到 or 的操作结果\$3 所以停顿的。\$3 停顿的时候值为 0x1e,停顿后的值为 0x0

equ函数中, (\$3)==(\$4)=0,

所以仅在 IF 阶段读取了 or \$2,\$4,\$3 指令后至今 branch 到 exit 函数, pc in 从

0x64 变到 0x74。下一个周期,IF 取指为 0x080000000, jump 到第一条指令,可以看到,此时 IF 指令停顿 2 个周期,这是因为 jump 指令到 ID 阶段后指令空了,IF 取不到正确的指令了,然后 jump 又在 ID 阶段暂停一个周期,这是因为我把 jump 和 branch 的冒险信号做到了一起,理论上分开的话是不用停顿一个周期的,所以两个周期后,PC 重新变为 0x0,开始新一轮循环,整个流程是正确的。

将测试代码换为如下:

main:

addi \$2,\$2,10

sw \$2,34

addi \$3,\$2,20

addi \$1,\$2,20

sw \$3,3

addi \$4,\$2,4

add \$4,\$2,\$3

sw \$4, 4

addi \$0,\$4,44

1w \$3, 4(\$3)

beq \$3, \$1, equ

and \$1, \$0, \$2

or \$3,\$4,\$0

sub \$5, \$0, \$6

or \$3,\$4,\$0

beq \$3,\$3, equ

1w \$2,0(\$3)

equ:

beq \$3, \$4, exit

or \$2,\$4,\$3

and \$2, \$4, \$3

s1t \$2,\$4,\$3

sw \$2,0(\$3)

exit:

j main

主要关注这两句,这里

1w \$3,4(\$3)

beq \$3, \$1, equ

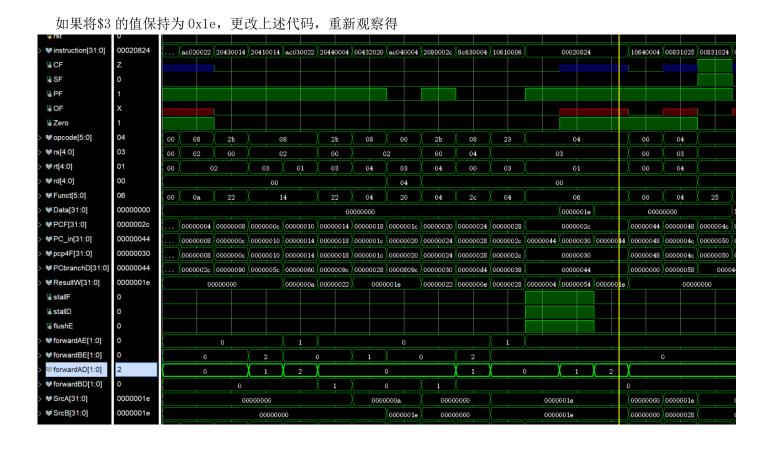
beq要使用lw的值,

		+	-· -· +-/ -
0x00000020		addi \$0,\$4,44	10: addi \$0,\$4,44
0x00000024	0x8c630004	lw \$3,4(\$3)	11: lw \$3,4(\$3)
0x00000028	0x10610006	beq \$3,\$1,6	12: beq \$3,\$1,equ
0x0000002c	0x00020824	and \$1,\$0,\$2	13: and \$1,\$0,\$2
0x00000030	0x00801825	or \$3,\$4,\$0	14: or \$3,\$4,\$0
0x00000034	0x00062822	sub \$5,\$0,\$6	15: sub \$5,\$0,\$6
0x00000038	0x00801825	or \$3,\$4,\$0	16: or \$3,\$4,\$0
0x0000003c	0x10630001	beq \$3,\$3,1	17: beq \$3,\$3,equ
0x00000040	0x8c620000	lw \$2,0(\$3)	18: lw \$2,0(\$3)
0x00000044	0x10640004	beq \$3,\$4,4	20: beq \$3,\$4,exit

[™] clk									112					
	0													
₩ rst	0	V		$\overline{}$				_ _		$\overline{}$,	
> W instruction[31:0]	00020824	ac040004	2080002c	8c630004	10610006		00020824		00801825	00062822	0080182	5 10630001	8c620000	10640004
¹⊌ CF	Z													
¹⊌ SF	0													
¹⊌ PF	1													
₩ OF	Х													
™ Zero	1													
> ♥ opcode[5:0]	04	00	2b	80	23	X	04		_X	(00		04	00
> W rs[4:0]	03	02	00	04	Χ	0	13		00	04	00	04	03	00
> W rt[4:0]	01	03	04	00	03	Х	01		02	00	06	00	03	00
> W rd[4:0]	00	04	X			00			01	03	05	03	X	
> W Funct[5:0]	06	20	04	2c	04	Х	06		24	25	22	25	01	00
> W Data[31:0]	00000000			00000000			0000000a	000	000000	XXXXXXX	\			
> W PCF[31:0]	0000002c	0000001c	00000020	00000024	00000028	X	0000002c		00000030	00000034	0000003	8 0000003c	00000040	00000044
> W PC_in[31:0]	00000030	00000020	00000024	00000028	0000002c	00000044	000000	30	00000034	00000038	0000003	c 00000040	00000044	00000048
> W pcp4F[31:0]	00000030	00000020	00000024	00000028	0000002c		00000030		00000034	00000038	0000003	c 00000040	00000044	00000048
> PCbranchD[31:0]	00000044	00008090	00000030	000000d4	00000038	X	00000044		000020c0	000060c8	0000a0c	000060d0	00000044	00000000
> ® ResultW[31:0]	0000000a	0000001e	00000003	0000000e	00000028	00000004	00000054	000 <mark>00</mark> :	a 0000	00000	ffffffe	c 00000000	00000028	00000028
™ stallF	0													
™ stallD	0													
¹⊌ flushE	0													
> W forwardAE[1:0]	0		0) 1	X				0				1
> ₩ forwardBE[1:0]	0	<u> </u>	0	2	XX					0				1
> v forwardAD[1:0]	2		0) 1	X	0	1	2			0		2 1	X
> v forwardBD[1:0]	0	0	1					0					2 1	
> W SrcA[31:0]	0000001e	0000000a	000	00000)	0000	001e		00000000	00000028	0000000	0 00000028	0000000a 000000	000000000
> W SrcB[31:0]	0000001e	0000001e	000	00000)	0000	001e		0000000a	X	0000000	0	0000000a 000000	000000000
> W pcp4D[31:0]	0000002c	00000010	00000020	00000024	00000028	(0000002c		00000030	00000034	0000003	8 0000003c	00000040	00000000
> W IR[31:0]	10610006	00432020	ac040004	2080002c	8c630004	(10610006		00020824	00801825	0006282	2 00801825	10630001	00000000
™ RegWriteD	0													
™ MemtoRegD	0													
MemWriteD	0													
™ ALUSrcD	0													
™ RegDstD	0													
[™] BranchD	1													
[™] JumpD	0													
I ⊌ EqualD	0													
№ PCSrcD	0													
> WALUControlD[2:0]	6			2		(6		0	1	6	1	6	
> W Imm_extend[31:0]	00000006	00002020	00000004	0000002c	00000004	X	00000006			11				5
NegWriteE	0								00000824	00001825	0000282	2 00001825	00000001	00000000
16 MemtoRegE	0								00000824	00001825	0000282	2 00001825	00000001	00000000
MemWriteE	0								00000824	00001825	0000282	2 00001825	00000001	00000000
™ ALUSrcE									00000824	00001825	0000282	2 00001825	00000001	00000000
™ RegDstE	0								00000824	00001825	0000282	2 00001825	00000001	\ 5 \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
	0								00000824	V	0000282	V		
₩ ALUControlE[2:0]	0	V		2			0		6	\	1) 6	1 0	\ 6
₩ RD1E[31:0]	0 0 00000000		0000a	00000000)(0000000e		000000		6 0000000a	0	1	X 6 8 X 00000000	\(\bar{1} \) \(\cdot \	6 000 00000028
₩ RD1E[31:0] ₩ RD2E[31:0]	0 0 00000000 00000000	00000000	0000001e	00000000	00000	0000001e	000000	00	6 0000000a 0000001e	0 00000000 00000000a	1	\$ 00000000 0000	\(\) \(\)	6 000 00000028 00000028
₩ RD1E[31:0] ₩ RD2E[31:0] ₩ ImmE[31:0]	0 0 00000000 00000000 00000000	00000000	0000001e	00000000	00000 0000002c	0000001e	000000	00	6 0000000a 0000001e	0 00000000 0000000a 00000824	1 0000002	\$\left\ 6 \\ 8 \left\ 000000000 \\ 00000 \\ 5 \left\ 00002822	1 0 00000028 000000 000000 000001825 000000	6 000 00000028 000000028
₩ RD1E[31:0] ₩ RD2E[31:0] ₩ ImmE[31:0] ₩ rsE[4:0]	0 0 00000000 00000000 00000000	00000000	0000001e 00002020 02	00000000 0000 00000004	00000 0000002c	0000001e 00000004 03	000000 000000 000000	00	6 0000000a 0000001e 00000006	0 00000000 00000000 0000000a 00000824	0000002	\$\left\ 6 \\ 8 \\ \ \ \ \ \ \ \ \ \ \ \ \ \ \	1 0 00000028 000000 000000 00001825 000000 04 00	6 000 00000028 000000001 03
■ RD1E[31:0] ■ RD2E[31:0] ■ ImmE[31:0] ■ rsE[4:0] ■ rtE[4:0]	0 0 00000000 00000000 00000000 00	00000000	0000001e 00002020 02 03	00000000	00000 0000002c	0000001e 00000004 03 03	000000 000000 000000 00	00	6 0000000a 0000001e	0 00000000 00000000 00000024 00	0000002	6 8 00000000 000055 00002822 00 06	1 0 00000028 000000 00000 000000000000000	6 000 00000028 000000028
₩ RD1E[31:0] ₩ RD2E[31:0] ₩ ImmE[31:0] ₩ rsE[4:0] ₩ rtE[4:0]	0 0 00000000 0000000 0000000 00 00 00	00000000	0000001e 00002020 02 03 04	00000000 00000 00000004 00 04	00000 0000002c 04 00	0000001e 00000004 03 03	000000 000000 000000 00 00	00	6 0000000a 0000001e 00000006 03	0 00000000 00000000 000000024 00 02 01	0000002 00000182 04 00	6 8 00000000 0000 5 00002822 00 06	1 0 00000028 000000 00000 000000000000000	6 000 00000028 000 00000028 000 00000001 03 03
₩ RD1E[31:0] ₩ RD2E[31:0] ₩ ImmE[31:0] ₩ rsE[4:0] ₩ rtE[4:0] ₩ rdE[4:0] ₩ SrcAE[31:0]	0 0 00000000 0000000 00000000 00 00 0000	00000000	0000001e 00002020 02 03	00000000 00000 00000004 00 04	00000 0000002c	0000001e 00000004 03 03	000000 000000 000000 00	00	6 0000000a 0000001e 00000006 03	0 00000000 00000000 00000024 00	0000002 00000182 04 00	6 8 00000000 0000 5 00002822 00 06	1 0 00000028 000000 00000 000000000000000	6 000 00000028 000 00000028 000 00000001 03 03
■ RD1E[31:0] ■ RD2E[31:0] ■ ImmE[31:0] ■ rsE[4:0] ■ rtE[4:0] ■ rdE[4:0] ■ SrcAE[31:0] ■ SrcBE[31:0]	0 0 00000000 0000000 0000000 00 00 00000	00000000	0000001e 00002020 02 03 04 0000a	00000000 00000004 00000004 00	00000 0000002c 04 00	0000001e 00000004 03 03 03 0000001e	000000 000000 000000 00 00	oo	6 0000000a 0000000 03 01	0 00000000 00000000 000000024 00 02 01	0000002 00000182 04 00	6 8 00000000 0000 5 00002822 00 06 05 8 00000000	1 0 00000028 000000 00000 000000000000000	6 000 00000028 000 00000028 000 00000001 03 03
■ RD1E[31:0] ■ RD2E[31:0] ■ ImmE[31:0] ■ rsE[4:0] ■ rtE[4:0] ■ rdE[4:0] ■ SrcAE[31:0] ■ SrcBE[31:0] ■ alu_result[31:0]	0 0 00000000 00000000 00000000 00 000000	00000000 00000004 004 00 0000 00000004	0000001e 00002020 02 03 04 00000a	00000000 0000004 00 04 00 04 00000000	000000 0000002e 04 00	0000001e 00000004 03 03 00 0000001e	000000 000000 000000 00 00 00	00 00 00 00	6 0000000a 0000001e 0000000a 0000001e	0 00000000 0000000a 00000024 00 02 01 00000000 00000000	0000002 0000182 04 00 03	6 8 00000000 0000 5 00002822 00 06 05 8 00000000	01 0 00000028 00000000000000000000000000	6 000 00000028 000000001 03 03 000 00000028
■ RD1E[31:0] ■ RD2E[31:0] ■ ImmE[31:0] ■ rsE[4:0] ■ rtE[4:0] ■ rdE[4:0] ■ SrcAE[31:0] ■ SrcBE[31:0] ■ WriteDataE[31:0]	0 0 00000000 0000000 0000000 00 00 00000	00000000 00000004 04 00 00000004 0000000	0000001e 00002020 02 03 04 00000a 0000001e 0000001e	000000004 00000004 0000000000000000000	00000 0000002c 04 00 00000028 0000002e	0000001e 00000004 03 03 00 0000001e 00000004	000000 000000 000000 00 00 000000 000000	00 00 00 00 00	6 0000000a 0000001e 00000006 03 01	0 00000000 0000000a 00000024 00 02 01 00000000 00000000	0000002 0000182 04 00 03	6 8 00000000 0000 5 00002822 00 06 05 8 00000000	01 0 00000028 00000000000000000000000000	6 000 00000028 000000001 03 03 000 00000028
RD1E[31:0] RD2E[31:0] ImmE[31:0] ImmE[31:0] IntE[4:0] InteR[4:0] In	0 0 00000000 0000000 0000000 00 00 00000	00000000 00000004 04 00 00000004 0000000	0000001e 00002020 02 03 04 00000a 0000001e 0000001e	000000004 00000004 0000000000000000000	00000 0000002c 04 00 00000028 0000002c 00000054	0000001e 00000004 03 03 00 0000001e 00000004	000000 000000 000000 00 00 000000 000000	00 00 00 00	6 0000000a 0000001e 00000006 03 01	0 00000000 0000000a 00000824 00 02 01 00000000 0000000a 00000000 00000000	0000002 0000182 04 00 03	6 8 00000000 0000 5 00000000 0000 0000 0	00000028 000000 00000 00000 00000 00000 00000 00000 00000 000000	6 000 00000028 00000028 000 00000001 03 03 000 00000028 00000028
RD1E[31:0] RD2E[31:0] ImmE[31:0] ImmE[31:0] InmE[4:0] In	0 0 00000000 0000000 0000000 00 00 00 0	000000000 00000004 004 00 000000004 000000	0000001e 00002020 02 03 04 00000a 0000001e 0000001e	000000000 000000004 000000000 00000000 000000	00000 0000002c 04 00 00000028 0000002c 00000054	0000001e 00000004 03 03 00 0000001e 00000001e 00000004 000000022 00000001e	000000 000000 000000 00 00 000000 000000	00 00 00 00	0000000a 0000001e 00000000 03 01 00000001e ffffffee 0000001e	0 00000000 0000000a 00000824 00 02 01 00000000 0000000a 00000000 00000000	0000002 00000182 04 00 03 0000002	6 8 00000000 0000 5 00002822 00 06 05 8 00000000 0000 8 00000000	00000028 000000 000000000 000000 00000 00000000	000 00000028 000 00000001 03 03 000 00000028 000000028
RD1E[31:0] RD2E[31:0] ImmE[31:0] ImsE[4:0] IntE[4:0] Int	0 0 00000000 0000000 0000000 00 00 00 0	00000000 00000004)	0000001e 00002020 02 03 04 0000004 0000001e 00000028	000000000 000000004 000000000 00000000 000000	00000 0000002c 04 00 0000002s 0000002s 00000054 00000000000000000000000000000	0000001e 0000004 03 03 00 0000001e 00000004 000000022 0000001e	000000 000000 00 00 00 00 00 00 00 00 0	00 00 00 00 00 00 00 00 00 00 00 00 00	0000000a 0000001e 00000000 03 01 00000001e ffffffee 0000001e	0 00000000 00000000 00000000 00000000 0000	0000002 0000002 04 00 03 0000002 0000002	6 8 00000000 0000 5 00002822 00 06 05 8 00000000 0000 3 00000000 03	00000028 000000 000000 000000 000000 0000000 000000	000 00000028 000 00000001 03 03 000 00000028 000000028 00000028
■ RD1E[31:0] ■ RD2E[31:0] ■ ImmE[31:0] ■ rsE[4:0] ■ rtE[4:0] ■ rdE[4:0] ■ SrcAE[31:0] ■ SrcBE[31:0] ■ WriteDataE[31:0] ■ WriteRegM[4:0] ■ WriteRegE[4:0] ■ ALUOutM[31:0] ■ WriteDataM[31:0]	0 0 00000000 00000000 00000000 00 00 00	00000000 00000004 00 00 00 00000004 00000000	0000001e 00002020 02 03 04 000000 0000001e 04 0000001e	000000000 00000004 00 00 00 04 00000000	00000000000000000000000000000000000000	0000001e 00000004 03 03 00 0000001e 00000004 000000022 0000001e 00	000000 000000 00 00 00 00 000000 000000	00 00 00 00 00 00	6 0000000a 0000001e 00000006 03 01 0000001e ffffffec 0000001e	0 00000000 00000000 00000000 00000000 0000	0000002 04 00 03 0000002 0000002	6 8 00000000 0000 000 0000 0000 0000 00	00000028 000000 000000 0000000 000000 00000000 000000	000 00000028 000 00000001 03 03 000 00000028 000000028 00000028
RD1E[31:0] RD2E[31:0] ImmE[31:0] ImsE[4:0] IntE[4:0] Int	0 0 00000000 00000000 00000000 00 00 00	00000000 00000004 00 00 00 00000004 00000000	0000001e 00002020 02 03 04 000000 0000001e 04 0000001e	000000000 00000004 00 00 00 04 00000000	00000000000000000000000000000000000000	0000001e 00000004 03 03 00 00 0000001e 00000004 00000022 0000001e 00 03 00000054	000000 000000 00 00 00 00 000000 000000	00 00 00 00 00 00 00 00	6 0000000a 00000000 03 03 00000000000000	00000000000000000000000000000000000000	0000002 04 00 03 0000002 0000002	6 8 00000000 0000 000 0000 0000 0000 00	00000028 000000 000000 000000 000000 0000000000	000 00000028 000 00000001 03 03 000 00000028 000000028 00000028
■ RD1E[31:0] ■ RD2E[31:0] ■ ImmE[31:0] ■ rsE[4:0] ■ rtE[4:0] ■ rdE[4:0] ■ SrcAE[31:0] ■ SrcBE[31:0] ■ WriteDataE[31:0] ■ WriteRegM[4:0] ■ WriteRegE[4:0] ■ ALUOutM[31:0] ■ WriteDataM[31:0]	0 0 00000000 00000000 00000000 00 00 00	00000000 00000004 00 00 00 00000004 00000000	0000001e 00002020 02 03 04 000000 0000001e 04 0000001e	000000000 00000004 00 00 00 04 00000000	00000000000000000000000000000000000000	0000001e 00000004 03 03 00 00 0000001e 00000004 00000022 0000001e 00 03 00000054	000000 000000 00 00 00 00 000000 000000	00 00 00 00 00 00 00 00	6 0000000a 00000000 03 03 00000000000000	00000000000000000000000000000000000000	0000002 04 00 03 0000002 0000002	6 8 00000000 0000 000 0000 0000 0000 00	00000028 000000 000000 000000 000000 0000000000	000 00000028 000 00000001 03 03 000 00000028 000000028 00000028
■ RD1E[31:0] ■ RD2E[31:0] ■ ImmE[31:0] ■ rsE[4:0] ■ rtE[4:0] ■ rdE[4:0] ■ SrcAE[31:0] ■ SrcBE[31:0] ■ Ju_result[31:0] ■ WritePataE[31:0] ■ WriteRegM[4:0] ■ ALUOuttM[31:0] ■ WriteDataM[31:0] ■ WriteDataM[31:0] ■ WriteDataM[31:0]	0 0 00000000 00000000 00000000 00 00 00	00000000 00000004 00 00 00 00000004 00000000	0000001e 00002020 02 03 04 000000 0000001e 04 0000001e	000000000 00000004 00 00 00 04 00000000	00000000000000000000000000000000000000	0000001e 00000004 03 03 00 00 0000001e 00000004 00000022 0000001e 00 03 00000054	000000 000000 00 00 00 00 000000 000000	00 00 00 00 00 00 00 00	6 0000000a 00000000 03 03 00000000000000	00000000000000000000000000000000000000	0000002 04 00 03 0000002 0000002	6 8 00000000 0000 000 0000 0000 0000 00	00000028 000000 000000 000000 000000 0000000000	000 00000028 000 00000001 03 03 000 00000028 000000028 00000028
RD1E[31:0] RD2E[31:0] ImmE[31:0] ImmE[31:0] Interpretation of the property	0 0 00000000 00000000 00000000 00 00 00	00000000 00000004 00 00 00 00000004 00000000	0000001e 00002020 02 03 04 000000 0000001e 04 0000001e	000000000 00000004 00 00 00 04 00000000	00000000000000000000000000000000000000	0000001e 00000004 03 03 00 00 0000001e 00000004 00000022 0000001e 00 03 00000054	000000 000000 00 00 00 00 00 00 000000 0000	00 00 00 00 00 00 00 00	6 0000000a 0000000a 03 01 0000000a 0000001e 00000000 0000000	00000000000000000000000000000000000000	0000002 04 00 03 0000002 0000002	6 8 000000000 0000 0000 0000 0000 0000	00000028 000000 000000 000000 000000 0000000000	000 00000028 000 00000001 03 03 000 00000028 000000028 00000028
₩ RD1E[31:0] ₩ RD2E[31:0] ₩ ImmE[31:0] ₩ rsE[4:0] ₩ rtE[4:0] ₩ rdE[4:0] ₩ SrcAE[31:0] ₩ SrcBE[31:0] ₩ Ju_result[31:0] ₩ WriteDataE[31:0] ₩ WriteRegM[4:0] ₩ WriteRegE[4:0] ₩ WriteDataM[31:0] ₩ WriteDataM[31:0] ₩ RegWriteM ¼ MemtoRegM ¼ MemWriteM	0 0 00000000 00000000 00000000 00 00 00	000000000 00000004 00 00 000000000 000000	0000001e 0000001e 0000001e 000001e 04 0000000000	000000000 00000004 00 00000004 00 000000	00000000000000000000000000000000000000	0000001e 00000004 03 03 00 0000001e 00000004 00000022 0000001e 00 03 03	000000 000000 00 00 00 00 00 00 000000 0000	200 200	6 0000000 00000000 03 01 00000000 ffffffee 00000000 00000000 00000000	0 00000000 00000000 00000004 00 00000000	0000002 0000002 04 00 03 0000002 0000002	6 8 000000000 0000 0000 0000 0000 0000000	00000028 000000 000000 000000 000000 0000000000	6 000 00000028 000 00000001 03 03 000 00000028 000000028 000000028 00000028
₩ RD1E[31:0] ₩ RD2E[31:0] ₩ ImmE[31:0] ₩ rsE[4:0] ₩ rtE[4:0] ₩ rdE[4:0] ₩ SrcAE[31:0] ₩ SrcBE[31:0] ₩ WriteDataE[31:0] ₩ WritePataE[31:0] ₩ WriteRegM[4:0] ₩ WriteRegE[4:0] ₩ WriteDataM[31:0] ₩ WriteDataM[31:0] ಔ RegWriteM ಔ MemtoRegM ಔ MemWriteM ₩ ReadDataW[31:0]	0 0 0 00000000 00000000 00 00 00 00 0000	000000000 00000004 00 00 00000000 000000	0000001e 0000001e 0000001e 000001e 04 0000000000	000000000 00000004 00 00000004 00 000000	00000000000000000000000000000000000000	0000001e 00000004 03 03 00 0000001e 00000004 00000022 0000001e 00 03 03	000000 000000 00 00 00 00 00 00 00 00 0	200 200	6 0000000a 0000001e 00000000 000000000000	0 00000000 00000000 00000000 00000000 0000	0000002 0000002 04 00 03 0000002 0000002	6 8 000000000 0000 0000 0000 0000 0000000	1 0 0 00000000 00000000000000000000000	6 000 00000028 000 00000028 000 00000028 000 00000028 000 00000028
₩ RD1E[31:0] ₩ RD2E[31:0] ₩ ImmE[31:0] ₩ rsE[4:0] ₩ rdE[4:0] ₩ rdE[4:0] ₩ SrcAE[31:0] ₩ SrcBE[31:0] ₩ WriteDataE[31:0] ₩ WritePataE[31:0] ₩ WriteRegM[4:0] ₩ WriteRegE[4:0] ₩ ALUOutM[31:0] ₩ WriteDataM[31:0] ₩ RegWriteM ಔ MemtoRegM ಔ MemWriteM ₩ ReadDataW[31:0] ₩ ALUOutW[31:0]	0 0 00000000 00000000 00 00 00 00 00 00	000000004 000000004 000 0000000000 00000000	0000001e 02 03 04 000002e 0000001e 04 00000000000000000000000000	000000000 00000004 00 00000004 00 000000	00000000000000000000000000000000000000	0000001e 00000004 03 03 00 0000001e 00000004 00000022 0000001e 00 03 03	000000 000000 00 00 00 00 00 00 00 00 0	00 000 000 000 000 000 000 000 000 000	6 0000000a 0000001e 00000000 000000000000	0 000000000 000000000 000000000 0000000	0000002 0000002 04 00 03 0000002 0000002	6 8 000000000 0000 0000 0000 0000 0000	000000028 00000000000000000000000000000	6 000 00000028 000 00000028 000 00000028 000 00000028 000 00000028

仿真如下:

0x8c630004 是 lw 指令, lw 在 EX 阶段时候 beq 在 ID 阶段,由于 beq 需要 lw 的\$3, 所以至少要 lw 在 EX 阶段末才能得到\$3,于是流水线停顿一次,lw 在 MEM 阶段时,可以看到 aluoutM 是 0x22,正是 4(\$3)的值,作为 Dmem 的读取地址,此时是停顿的第二个周期,由于 branchstall 里面 MEM 阶段有 ID 要用的操作数,所以 branchstall 又停顿一个周期,直到 lw 进入 WB 阶段,此时寄存器还没有将\$3 的值更新,所以要将 resultW 也就是(\$3)=0xa 前递到 branch 的 EqualD 的计算中,而此时(\$1)=SrcB=0x1e,两者不相等,所以 PC_in 没有变成 PCbranchD,而是普通+4.



可以看到此时便成功地跳到了 PC=0x44 处,说明 1w 和 branch 这种最极端的冒险情况也是正常运行的

五、调试和心得体会

设计流水线 CPU 是一个浩大的工程,除了按图说话设计出整体数据通路外,最主要的是考虑到所有的冒险行为,从普通的 WB/MEM->EX 转发到 branch 和 lw 的停顿与 WB->ID 转发的结合,各个地方都要细致考虑,CPU 设计是一个宏观又极为精细的活。这次实验使我大大增强了对 CPU 基本运行逻辑的理解,也对整个工程设计的思路,从宏观到细节的把握能力有了很大的锻炼,非常有意义。