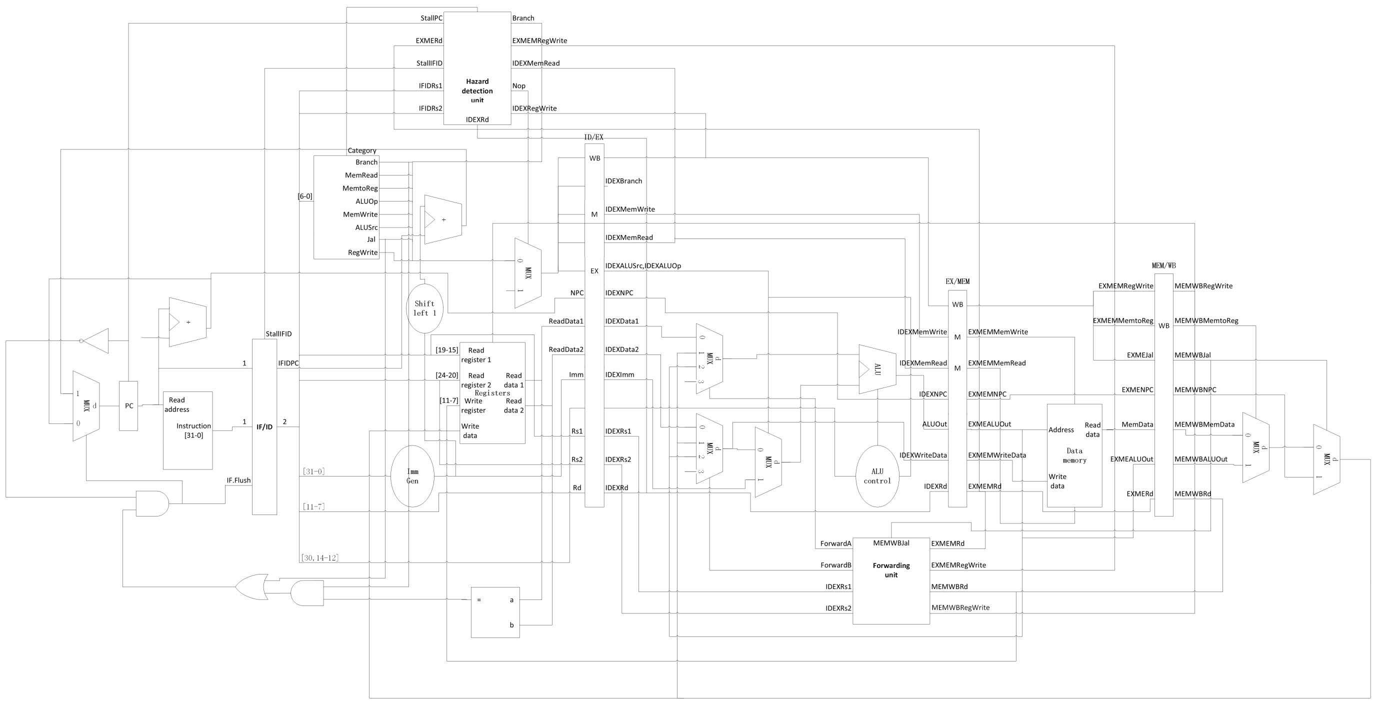
实验5 流水线

PB20000156
徐亦昶

数据通路



控制逻辑

Control

Opcode	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	Jal	RegWrite	Category
0010011	0	0	0	11	0	1	0	1	0
0110011	0	0	0	10	0	0	0	1	1
1101111	0	0	0	11	0	0	1	1	1
1100011	1	0	0	01	0	0	0	0	0
0000011	0	1	1	00	0	1	0	1	1
0100011	0	0	0	00	1	1	0	0	1

0：2个操作数 1：1个操作数

ALU control

ALUOp	instruction[30,14-12]	ALU_sel
00	xxxx	001(+)
01	xxxx	000(-)

ALUOp	instruction[30,14-12]	ALU_sel
10	0000	001(+)
10	1000	000(-)
10	0001	110(<<)
10	0100	100(^)
10	0101	101(>>)
10	1101	111(>>>)
10	0110	011(or)
10	0111	010(and)
11	x000	001(+)
11	x100	100(^)
11	x110	011(or)
11	x111	010(and)

Imm Gen

instruction[6:0]	Imm
0010011	{{20{instruction[31]}},instruction[31:20]}
1100011	{{20{instruction[31]}},instruction[7],instruction[30:25],instruction[11:8]}
0000011	{{20{instruction[31]}},instruction[31:20]}
0100011	{{20{instruction[31]}},instruction[31:25],instruction[12:8]}
1101111	{{12{instruction[31]}},instruction[19:12],instruction[20],instruction[30:21],1'b0}
others	32'b0

Hazard Detection Unit

```
if(Category==0) //本指令有两个源寄存器(add/addi/beq)
{
    if(Branch==0)
    {
        if(IDEXMemRead==1 && IDEXRd!=0 && (IDEXRd==IFIDRs1||IDEXRd==IFIDRs2)) //上一条指
令是lw
        {
            StallPC=1;
            StallIFID=1;
            Nop=1;
        }
        else
        {
            StallPC=0;
            StallIFID=0;
            Nop=0;
        }
    }
    else //branch指令
    {
        if(IDEXRegWrite==1 && IDEXRd!=0 &&
(IDEXRd==IFIDRs1||IDEXRd==IFIDRs2)||EXMEMRegWrite==1 && EXMEMRd!=0 &&
        (EXMEMRd==IFIDRs1||EXMEMRd==IFIDRs2))
        {
            StallPC=1;
            StallIFID=1;
            Nop=1;
        }
        else //不停顿
        {
            StallPC=0;
            StallIFID=0;
            Nop=0;
        }
    }
}
else
{
    if(IDEXMemRead==1 && IDEXRd!=0 && IDEXRd==IFIDRs1)
    {
        StallPC=1;
        StallID=1;
        Nop=1;
    }
    else
    {
        StallPC=0;
        StallIFID=0;
        Nop=0;
    }
}
```

Forwarding unit

```
if(MEMWBJal==0)
{
    if(EXMEMRegWrite==1&&EXMEMRd!=0&&MEMWBRd==IDEXRs1) ForwardA=10;
    else if(MEMWBRegWrite==1&&MEMWBRd!=0&&MEMWBRd==IDEXRs1) ForwardA=01;
    else ForwardA=00;
    if(EXMEMRegWrite==1&&EXMEMRd!=0&&MEMWBRd==IDEXRs2) ForwardB=10;
    else if(MEMWBRegWrite==1&&MEMWBRd!=0&&MEMWBRd==IDEXRs2) ForwardB=01;
    else ForwardA=00;
}
else
{
    if(MEMWBRd==IDEXRs1) ForwardA=10;
    else ForwardA=00;
    ForwardB=00;
}
```

chk_addr和chk_data对应关系

当chk_addr为0x00xx时和ppt上略有不同

chk_addr[7:0]	chk_data	解释
0	pcin	PC_driver的输入PC
1	pc	IFID寄存器的输入PC
2	pcd	ID阶段使用的PC
3	ir	IF阶段取出的指令
4	ird	ID阶段执行的指令
5	a	寄存器堆ReadData1
6	b	寄存器堆ReadData2
7	imm	立即数单元产生的立即数
8	y	ALU输出
9	mdr	数据存储器读出的数据
a	pce	EX阶段对应的PC
b	pcm	MEM阶段对应的PC
c	pcw	WB阶段对应的PC

代码实现

CPU包含两个单端口RAM：dist_mem_gen_0/1，其中0是数据存储器，1是指令存储器。它们的Depth和Data Width均设为12288和32，指令寄存器初始化coe文件为text.coe，数据存储器不使用coe文件初始化。

ALU模块

```
//ALU.v
`timescale 1ns / 1ps
module ALU #(
parameter WIDTH=32
)(
input [WIDTH-1:0] a,b,
input [2:0] s,
output reg [WIDTH-1:0] y,
output reg [2:0] f
);
always@(*)
begin
    if(s==3'b000)
    begin
        f[0]=(a==b)?1:0;
        f[2]=(a<b)?1:0;
        case({a[WIDTH-1],b[WIDTH-1]})
            2'b00:
                f[1]=(a<b)?1:0;
            2'b01:
                f[1]=0;
            2'b10:
                f[1]=1;
            2'b11:
                f[1]=(a[WIDTH-2:0]<b[WIDTH-2:0])?1:0;
            default:
                f[1]=0;
        endcase
    end
    else
        f=3'b000;
end
always@(*)
begin
    case(s)
        3'b000:y=a-b;
        3'b001:y=a+b;
        3'b010:y=a&b;
        3'b011:y=a|b;
        3'b100:y=a^b;
        3'b101:y=a>>b;
        3'b110:y=a<<b;
        3'b111:y=a>>>b;
        default:y=a;
    endcase
end
endmodule
```

ALU_control

```
ALU_control.v
`timescale 1ns / 1ps
module ALU_control(
input[1:0] ALUOp,
input[3:0] instruction,
output reg[2:0] ALU_sel);
always@(*)
begin
    case(ALUOp)
        2'b00:ALU_sel=3'b001; //lw/sw
        2'b01:ALU_sel=3'b000; //beq
        2'b10: //register operation
        begin
            case(instruction)
                4'b0000:ALU_sel=3'b001;
                4'b1000:ALU_sel=3'b000;
                4'b0001:ALU_sel=3'b110;
                4'b0100:ALU_sel=3'b100;
                4'b0101:ALU_sel=3'b101;
                4'b1101:ALU_sel=3'b111;
                4'b0110:ALU_sel=3'b011;
                4'b0111:ALU_sel=3'b010;
                default:ALU_sel=3'b000;
                endcase
            end
            2'b11: //imm operation
            begin
                casex(instruction)
                    4'bx000:ALU_sel=3'b001;
                    4'bx100:ALU_sel=3'b100;
                    4'bx110:ALU_sel=3'b011;
                    4'bx111:ALU_sel=3'b010;
                    default:ALU_sel=3'b000;
                endcase
            end
            default:ALU_sel=3'b000;
        endcase
    end
endmodule
```

Add

```
//Add.v
`timescale 1ns / 1ps
module Add #(
parameter WIDTH=32
)(
input[WIDTH-1:0] a,b,
output[WIDTH-1:0] y);
assign y=a+b;
endmodule
```

四个流水线寄存器

IF/ID:

```
//IFID.v
`timescale 1ns / 1ps
module IFID(
input clk,rstn,StallIFID,Flush,
input[31:0] PC,_instruction,
output reg[31:0] instruction,IFIDPC);
always@(posedge clk or negedge rstn)
begin
    if(rstn==0)
    begin
        instruction<=0;
        IFIDPC<=32'h3000;
    end
    else
    begin
        if(Flush==1'b1)
            instruction<=32'b0;
        else if(StallIFID==1'b0)
        begin
```

```
            IFIDPC<=PC;
            instruction<=_instruction;
        end
    end
end
endmodule
```

仿真文件:

```
//test_IFID.v
`timescale 1ns / 1ps
module test_IFID();
reg clk,StallIFID,Flush;
reg [31:0]PC,_instruction;
wire [31:0]instruction,IFIDPC;
IFID IFID(clk,StallIFID,Flush,PC,_instruction,instruction,IFIDPC);
initial
begin
    PC<=32'h3000;
    _instruction<=32'habcd;
    StallIFID=0;
    Flush=0;
    clk=0;
end
initial
    forever #1 clk=~clk;
initial
begin
    #3 PC=PC+32'h4;
    _instruction<=32'h1234;
    #3 Flush=1'b1;
    #3 Flush=0;
    #3 PC=PC+32'h4;
    #0.5 StallIFID=1'b1;
    #5 StallIFID=0;
end
endmodule
```

ID/EX:

```
//IDEX.v
`timescale 1ns / 1ps
module IDEX(
input clk,rstn,Branch,MemRead,MemtoReg,MemWrite,ALUSrc,Jal,RegWrite,
input [1:0]ALUOp,
input [31:0]NPC,ReadData1,ReadData2,Imm,IFIDPC,
input [4:0]Rs1,Rs2,Rd,
input [3:0]ALUCtrl,
output reg
IDEXBranch,IDEXMemRead,IDEXMemtoReg,IDEXMemWrite,IDEXALUSrc,IDEXJal,IDEXRegWrite,
output reg [1:0]IDEXALUOp,
output reg [31:0]IDEXNPC,IDEXData1,IDEXData2,IDEXImm,IDEXPC,
output reg [4:0]IDEXRs1,IDEXRs2,IDEXRd,
output reg [3:0]IDEXALUCtrl);
always@(posedge clk or negedge rstn)
if(rstn==0)

{IDEXBranch,IDEXMemRead,IDEXMemtoReg,IDEXMemWrite,IDEXALUSrc,IDEXJal,IDEXRegWrite,IDEXA
LUOp,IDEXNPC,IDEXData1,IDEXData2,IDEXImm,IDEXRs1,IDEXRs2,IDEXRd,IDEXALUCtrl,IDEXPC}<=0;
else

{IDEXBranch,IDEXMemRead,IDEXMemtoReg,IDEXMemWrite,IDEXALUSrc,IDEXJal,IDEXRegWrite,IDEXA
LUOp,IDEXNPC,IDEXData1,IDEXData2,IDEXImm,IDEXRs1,IDEXRs2,IDEXRd,IDEXALUCtrl,IDEXPC}<=
{Branch,MemRead,MemtoReg,MemWrite,ALUSrc,Jal,RegWrite,ALUOp,NPC,ReadData1,ReadData2,Imm
,Rs1,Rs2,Rd,ALUCtrl,IFIDPC};
endmodule
```

EX/MEM

```
//EXMEM.v
`timescale 1ns / 1ps
module EXMEM(
input clk,rstn,IDEXRegWrite,IDEXMemtoReg,IDEXJal,IDEXMemWrite,IDEXMemRead,
input [31:0]IDEXNPC,ALUOut,IDEXWriteData,IDEXPC,
input [4:0]IDEXRd,
output reg EXMEMRegWrite,EXMEMMemtoReg,EXMEMJal,EXMEMMemWrite,EXMEMMemRead,
output reg[31:0] EXMEMNPC,EXMEMALUOut,EXMEMWriteData,EXMEMPC,
output reg[4:0] EXMEMRd);
always@(posedge clk or negedge rstn)
if(rstn==0)

{EXMEMRegWrite,EXMEMMemtoReg,EXMEMJal,EXMEMMemWrite,EXMEMMemRead,EXMEMNPC,EXMEMALUOut,E
XMEMWriteData,EXMEMRd,EXMEMPC}<=0;
else

{EXMEMRegWrite,EXMEMMemtoReg,EXMEMJal,EXMEMMemWrite,EXMEMMemRead,EXMEMNPC,EXMEMALUOut,E
XMEMWriteData,EXMEMRd,EXMEMPC}<=
{IDEXRegWrite,IDEXMemtoReg,IDEXJal,IDEXMemWrite,IDEXMemRead,IDEXNPC,ALUOut,IDEXWriteDat
a,IDEXRd,IDEXPC};
endmodule
```

MEM/WB

```
//MEMWB.v
`timescale 1ns / 1ps
module MEMWB(
input clk,rstn,EXMEMRegWrite,EXMEMMemtoReg,EXMEMJal,
input[31:0] EXMEMNPC,MemData,EXMEMALUOut,EXMEMPC,
input[4:0] EXMEMRd,
output reg MEMWBRegWrite,MEMWBMemtoReg,MEMWBJal,
output reg[31:0] MEMWBNPC,MEMWBMemData,MEMWBALUOut,MEMWBPC,
output reg[4:0]MEMWBRd);
always@(posedge clk or negedge rstn)
if(rstn==0)
    {MEMWBRegWrite,MEMWBMemtoReg,MEMWBJal,MEMWBNPC,MEMWBMemData,MEMWBALUOut,MEMWBRd}
<=0;
```

```
else

{MEMWBRegWrite,MEMWBMemtoReg,MEMWBJal,MEMWBNPC,MEMWBMemData,MEMWBALUOut,MEMWBRd,MEMWBPC
}<={EXMEMRegWrite,EXMEMMemtoReg,EXMEMJal,EXMEMNPC,MemData,EXMEMALUOut,EXMEMRd,EXMEMPC};
endmodule
```

前递控制单元

```
//ForwardingUnit.v
`timescale 1ns / 1ps
module ForwardingUnit(
input EXMEMRegWrite,MEMWBRegWrite,MEMWBJal,
input[4:0] EXMEMRd,MEMWBRd,IDEXRs1,IDEXRs2,
output reg[1:0]ForwardA,ForwardB);
always@(*)
begin
    if(MEMWBJal==0)
    begin
        if(EXMEMRegWrite==1'b1&&EXMEMRd!=0&&EXMEMRd==IDEXRs1) ForwardA=2'b10;
        else if(MEMWBRegWrite==1'b1&&MEMWBRd!=0&&MEMWBRd==IDEXRs1) ForwardA=2'b01;
        else ForwardA=00;
        if(EXMEMRegWrite==1'b1&&EXMEMRd!=0&&EXMEMRd==IDEXRs2) ForwardB=2'b10;
        else if(MEMWBRegWrite==1'b1&&MEMWBRd!=0&&MEMWBRd==IDEXRs2) ForwardB=2'b01;
        else ForwardB=2'b00;
    end
    else //the previous instruction is jal
    begin
        if(MEMWBRd==IDEXRs1) ForwardA=2'b10;
        else ForwardA=2'b00;
        ForwardB=2'b00;
    end
end
endmodule
```

停顿处理单元

```
//HazardDetectionUnit.v
`timescale 1ns / 1ps
module HazardDetectionUnit(
input Branch,EXMEMRegWrite,IDEXMemRead,IDEXRegWrite,Category,
input [4:0]IFIDRs1,IFIDRs2,IDEXRd,EXMEMRd,
output reg StallPC,StallIFID,Nop);
always@(*)
begin
    if(Category==0)
    begin
        if(Branch==0)
        begin
            if(IDEXMemRead==1'b1 && IDEXRd!=0 && (IDEXRd==IFIDRs1||IDEXRd==IFIDRs2))
            begin
                StallPC=1'b1;
                StallIFID=1'b1;
                Nop=1'b1;
            end
            else
            begin
                StallPC=0;
                StallIFID=0;
                Nop=0;
            end
        end
        else
        begin
            if(IDEXRegWrite==1 && IDEXRd!=0 &&
(IDEXRd==IFIDRs1||IDEXRd==IFIDRs2)||EXMEMRegWrite==1 && EXMEMRd!=0 &&
(EXMEMRd==IFIDRs1||EXMEMRd==IFIDRs2))
            begin
                StallPC=1'b1;
                StallIFID=1'b1;
                Nop=1'b1;
            end
            else
            begin
                StallPC=0;
                StallIFID=0;
                Nop=0;
            end
        end
    end
    else
    begin
        if(IDEXMemRead==1'b1 && IDEXRd!=0 && IDEXRd==IFIDRs1)
        begin
            StallPC=1'b1;
            StallIFID=1'b1;
            Nop=1'b1;
        end
        else
        begin
            StallPC=0;
            StallIFID=0;
            Nop=0;
        end
    end
end
endmodule
```

仿真文件：

```
//test_HazardDetectionUnit.v
`timescale 1ns / 1ps
module test_HazardDetectionUnit();
reg Branch,EXMEMRegWrite,IDEXMemRead,IDEXRegWrite,Category;
reg [4:0]IFIDRs1,IFIDRs2,IDEXRd,EXMEMRd;
wire StallPC,StallIFID,Nop;
HazardDetectionUnit
HazardDetectionUnit(Branch,EXMEMRegWrite,IDEXMemRead,IDEXRegWrite,Category,IFIDRs1,IFID
Rs2,IDEXRd,EXMEMRd,StallPC,StallIFID,Nop);
initial
{Branch,EXMEMRegWrite,IDEXMemRead,IDEXRegWrite,Category,IFIDRs1,IFIDRs2,IDEXRd,EXMEMRd}
=0;
initial
begin
    #1 Branch=1'b1;
    IDEXRegWrite=1'b1; //test branch after lw
    IDEXRd=5'hc;
    IFIDRs2=5'hc;
    #1 IDEXRegWrite=0;
    EXMEMRegWrite=1'b1;
    EXMEMRd=5'hc;
    IDEXRd=0;
    #1 IDEXRd=0; //test normal branch
    EXMEMRd=0;
    #1 Branch=0; //test hazard without branch
    IDEXMemRead=1'b1;
    IDEXRd=5'ha;
    IFIDRs2=5'ha;
    #1 Category=1'b1; //test hazard with one source register
    #1 IFIDRs2=0;
    IFIDRs1=5'ha;
end
endmodule
```

立即数生成

```
//Imm_Gen.v
`timescale 1ns / 1ps
module Imm_Gen(
input[31:0] instruction,
output reg[31:0] Imm);
always@(*)
begin
    casex(instruction[6:0])
        7'b0010011:Imm={{20{instruction[31]}},instruction[31:20]};
        7'b1100111:Imm={{20{instruction[31]}},instruction[31:20]};
        7'b1100011:Imm=
{{20{instruction[31]}},instruction[7],instruction[30:25],instruction[11:8]};
        7'b0000011:Imm={{20{instruction[31]}},instruction[31:20]}; //lw
        7'b0100011:Imm={{20{instruction[31]}},instruction[31:25],instruction[11:8]};
//sw
        7'b1101111:Imm=
{{13{instruction[31]}},instruction[19:12],instruction[20],instruction[30:21]}; //jal
        default:Imm=32'b0;
    endcase
end
endmodule
```

PC控制

```
//PC_driver.v
`timescale 1ns / 1ps
module PC_driver(
input[31:0] NPC,
input clk,rstn,StallPC,
output reg[31:0] PC);
always@(posedge clk or negedge rstn)
begin
    if(rstn==0)
        PC<=32'h3000;
    else
        if(StallPC==0)
            PC<=NPC;
end
endmodule
```

主控部分（产生控制信号）

```
//control.v
`timescale 1ns / 1ps
module control(
input[6:0] Opcode,
output reg Branch,MemRead,MemtoReg,MemWrite,ALUSrc,Jal,RegWrite,Category,
output reg[1:0] ALUOp);
always@(*)
begin
    case(Opcode)
        7'b0010011:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jal,RegWrite,Category}=10'b0001101010;
        7'b0110011:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jal,RegWrite,Category}=10'b0001000011;
        7'b1101111:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jal,RegWrite,Category}=10'b0001100111;
        7'b1100011:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jal,RegWrite,Category}=10'b1000100000;
        7'b0000011:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jal,RegWrite,Category}=10'b0110001011;
        7'b0100011:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jal,RegWrite,Category}=10'b0000011001;
        default:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jal,RegWrite,Category}=10'b0;
    endcase
end
endmodule
```

数据寄存器

```
//data_memory.v
`timescale 1ns / 1ps
module data_memory(
input MemWrite,MemRead,clk,
input [31:0]Address,WriteData,io_din,
output reg [31:0]ReadData,io_dout,
output reg[7:0] io_addr,
output reg io_we,io_rd);
wire[31:0] shifted=Address>>2;
wire[31:0] RealData;
reg RealWe;
dist_mem_gen_0 memory (
 .a(shifted[13:0]), // input wire [15 : 0] a
 .d(WriteData), // input wire [31 : 0] d
 .clk(clk), // input wire clk
 .we(RealWe), // input wire we
 .spo(RealData) // output wire [31 : 0] spo
);
always@(*)
begin
 if(Address>=32'hff00)
 {ReadData,RealWe,io_dout,io_addr,io_we,io_rd}=
{io_din,1'b0,WriteData,Address[7:0],MemWrite,MemRead};
 else
 {ReadData,RealWe,io_dout,io_addr,io_we,io_rd}={RealData,MemWrite,32'b0,8'b0,2'b0};
end
endmodule
```

指令寄存器

```
//instruction_memory.v
`timescale 1ns / 1ps
module instruction_memory(
input clk,
input[31:0] PC,
output[31:0] instruction);
wire [31:0] shifted=(PC-32'h3000)>>2;
dist_mem_gen_1 IM (
 .a(shifted[13:0]), // input wire [13 : 0] a
 .d(0), // input wire [31 : 0] d
 .clk(clk), // input wire clk
 .we(0), // input wire we
 .spo(instruction) // output wire [31 : 0] spo
 );
endmodule
```

寄存器堆

```
//registers.v
`timescale 1ns / 1ps
module registers #(
    parameter AW=5,
    parameter DW=32
)(
    input clk,
    input [AW-1:0] ra0,ra1,
    output [DW-1:0] rd0,rd1,
    input [AW-1:0] wa,
    input [DW-1:0] wd,
    input we,rstn
);
reg [DW-1:0] rf [0:(1<<AW)-1];
integer i;
initial
begin
    for(i=0;i<32;i=i+1)
        rf[i]=0;
end
always@(negedge clk or negedge rstn)
begin
    if(!rstn)
        for(i=0;i<32;i=i+1)
           rf[i]=0;
    else
        if(we&&wa!=0)
            rf[wa]<=wd;
end
assign rd0=rf[ra0];
assign rd1=rf[ra1];
endmodule
```

数据选择器

MUXaxb表示位宽为b的a选器。

```
//MUX2x32.v
`timescale 1ns / 1ps
module MUX2x32(
input[31:0] s0,s1,
input sel,
output[31:0] d);
assign d=sel?s1:s0;
endmodule
```

```
//MUX2x5.v
`timescale 1ns / 1ps
module MUX2x5(
input [4:0]s0,s1,
input sel,
output [4:0]d);
assign d=sel?s1:s0;
endmodule
```

```
//MUX2x9.v
`timescale 1ns / 1ps
module MUX2x9(
input [8:0]s0,s1,
input sel,
output [8:0]d);
assign d=sel?s1:s0;
endmodule
```

```
//MUX4x32.v
`timescale 1ns / 1ps
module MUX4x32(
input[31:0] s0,s1,s2,s3,
input [1:0]sel,
output[31:0] d);
assign d=sel[1]?(sel[0]?s3:s2):(sel[0]?s1:s0);
endmodule
```

```
//MUX2x1.v
`timescale 1ns / 1ps
module MUX2x1(
input s0,s1,
input sel,
output d);
assign d=sel?s1:s0;
endmodule
```

调试数据接口

```
//chk_data_driver.v
`timescale 1ns / 1ps
module chk_data_driver(
input[15:0] chk_addr,
input[31:0] pcin,pc,pcd,ir,ird,a,b,imm,y,mdr,
input[8:0] ctrl,
output reg[31:0] chk_data,
input[31:0] pce,pcm,pcw);
always@(*)
begin
    if(chk_addr[15:12]==4'h0)
    begin
        case(chk_addr[7:0])
            8'h0:chk_data=pcin;
            8'h1:chk_data=pc;
            8'h2:chk_data=pcd;
            8'h3:chk_data=ir;
            8'h4:chk_data=ird;
            8'h5:chk_data=a;
            8'h6:chk_data=b;
            8'h7:chk_data=imm;
            8'h8:chk_data=y;
            8'h9:chk_data=mdr;
            8'ha:chk_data=pce;
```

```
                8'hb:chk_data=pcm;
                8'hc:chk_data=pcw;
                default:chk_data=32'b0;
            endcase
        end
        else if(chk_addr[15:12]==4'h1)
            chk_data=a;
        else
            chk_data=mdr;
    end
endmodule
```

CPU

```
//CPU.v
`timescale 1ns / 1ps
module CPU(
input clk,rst,
output[7:0] io_addr,
output[31:0] io_dout,
output io_we,io_rd,
input[31:0] io_din,
output[31:0] chk_pc,
input[15:0] chk_addr,
output[31:0] chk_data);
wire
[31:0]NPC,PC_addimm,PC_in,PC_out,ReadData1,ReadData2,instruction,IFIDinstruction,IFIDPC
,Imm,shifted,RegWriteData,IDEXNPC,IDEXData1,IDEXData2,IDEXImm,ALUSrc1,ALUSrc2,MUXALUSrc
2_0,
ALUOut,EXMEMNPC,EXMEMALUOut,EXMEMWriteData,MemData,MEMWBNPC,MEMWBMemData,MEMWBALUOut,MU
XMemALUOut,MemAddr,IDEXPC,EXMEMPC,MEMWBPC;
wire
Flush,StallPC,Branch,equal,Jal,StallIFID,Nop,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite,
Category,
_IDEXRegWrite,_IDEXBranch,_IDEXMemtoReg,_IDEXJal,_IDEXMemRead,_IDEXALUSrc,_IDEXMemWrite
,IDEXBranch,IDEXMemRead,IDEXMemtoReg,IDEXMemWrite,IDEXALUSrc,IDEXJal,IDEXRegWrite,
EXMEMRegWrite,EXMEMMemtoReg,EXMEMJal,EXMEMMemWrite,EXMEMMemRead,MEMWBRegWrite,MEMWBMemt
oReg,MEMWBJal,rstn,DebugMem,DebugReg;
wire [1:0]ALUOp,_IDEXALUOp,IDEXALUOp,ForwardA,ForwardB;
wire [2:0]ALUControlSignal;
wire [4:0]IFIDRs1,IFIDRs2,EXMEMRd,IDEXRs1,IDEXRs2,IDEXRd,MEMWBRd,RegAddr;
wire [3:0]IDEXALUCtrl;
assign chk_pc=PC_out;
assign DebugMem=chk_addr[13]?1'b1:1'b0;
assign DebugReg=chk_addr[12]?1'b1:1'b0;
assign rstn=~rst;
assign equal=(ReadData1==ReadData2)?1'b1:1'b0;
assign Flush=~StallPC&(Jal|Branch&equal);
assign shifted=Imm<<1;
assign IFIDRs1=IFIDinstruction[19:15];
assign IFIDRs2=IFIDinstruction[24:20];
MUX2x32 MUXPC(.s0(NPC),.s1(PC_addimm),.sel(Flush),.d(PC_in));
PC_driver PC_driver(.clk(clk),.rstn(rstn),.NPC(PC_in),.StallPC(StallPC),.PC(PC_out));
instruction_memory instruction_memory(.clk(clk),.PC(PC_out),.instruction(instruction));
Add Add1(.a(PC_out),.b(32'h4),.y(NPC));
```

```
IFID
IFID(.clk(clk),.rstn(rstn),.StallIFID(StallIFID),.Flush(Flush),.PC(PC_out),._instructio
n(instruction),.instruction(IFIDinstruction),.IFIDPC(IFIDPC));
control
control(.Opcode(IFIDinstruction[6:0]),.Branch(Branch),.MemRead(MemRead),.MemtoReg(Memto
Reg),.ALUOp(ALUOp),.MemWrite(MemWrite),.ALUSrc(ALUSrc),.Jal(Jal),.RegWrite(RegWrite),.C
ategory(Category));
Imm_Gen Imm_Gen(.instruction(IFIDinstruction),.Imm(Imm));
Add Add2(.a(shifted),.b(IFIDPC),.y(PC_addimm));
registers
registers(.clk(clk),.rstn(rstn),.ra0(RegAddr),.ra1(IFIDinstruction[24:20]),.rd0(ReadDat
a1),.rd1(ReadData2),.wa(MEMWBRd),.wd(RegWriteData),.we(MEMWBRegWrite));
MUX2x9
MUXControl(.s0({Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jal,RegWrite}),.s1(0),.se
l(Nop),.d({_IDEXBranch,_IDEXMemRead,_IDEXMemtoReg,_IDEXALUOp,_IDEXMemWrite,_IDEXALUSrc,
_IDEXJal,_IDEXRegWrite}));
HazardDetectionUnit
HazardDetectionUnit(.Branch(Branch),.EXMEMRegWrite(EXMEMRegWrite),.IDEXMemRead(IDEXMemR
ead),.IDEXRegWrite(IDEXRegWrite),.Category(Category),.IFIDRs1(IFIDRs1),.IFIDRs2(IFIDRs2
),.IDEXRd(IDEXRd),.EXMEMRd(EXMEMRd),.StallPC(StallPC),.StallIFID(StallIFID),.Nop(Nop));
IDEX
IDEX(.clk(clk),.rstn(rstn),.Branch(_IDEXBranch),.MemRead(_IDEXMemRead),.MemtoReg(_IDEXM
emtoReg),.MemWrite(_IDEXMemWrite),.ALUSrc(_IDEXALUSrc),.Jal(_IDEXJal),.RegWrite(_IDEXRe
gWrite),.ALUOp(_IDEXALUOp),.NPC(NPC),.ReadData1(ReadData1),.ReadData2(ReadData2),.Imm(I
mm),.ALUCtrl({IFIDinstruction[30],IFIDinstruction[14:12]}),
.Rs1(IFIDinstruction[19:15]),.Rs2(IFIDinstruction[24:20]),.Rd(IFIDinstruction[11:7]),
.IDEXBranch(IDEXBranch),.IDEXMemRead(IDEXMemRead),.IDEXMemtoReg(IDEXMemtoReg),.IDEXMemW
rite(IDEXMemWrite),.IDEXALUSrc(IDEXALUSrc),.IDEXJal(IDEXJal),.IDEXRegWrite(IDEXRegWrite
),.IDEXALUOp(IDEXALUOp),.IDEXNPC(IDEXNPC),.IDEXData1(IDEXData1),.IDEXData2(IDEXData2),.
IDEXImm(IDEXImm),
.IDEXRs1(IDEXRs1),.IDEXRs2(IDEXRs2),.IDEXRd(IDEXRd),.IDEXALUCtrl(IDEXALUCtrl),.IFIDPC(I
FIDPC),.IDEXPC(IDEXPC));
MUX4x32
MUXForward1(.s0(IDEXData1),.s1(RegWriteData),.s2(EXMEMALUOut),.s3(0),.sel(ForwardA),.d(
ALUSrc1));
MUX4x32
MUXForward2(.s0(IDEXData2),.s1(RegWriteData),.s2(EXMEMALUOut),.s3(0),.sel(ForwardB),.d(
MUXALUSrc2_0));
MUX2x32 MUXALUSrc2(.s0(MUXALUSrc2_0),.s1(IDEXImm),.sel(IDEXALUSrc),.d(ALUSrc2));
ALU ALU(.a(ALUSrc1),.b(ALUSrc2),.s(ALUControlSignal),.y(ALUOut));
ALU_control
ALU_control(.ALUOp(IDEXALUOp),.instruction(IDEXALUCtrl),.ALU_sel(ALUControlSignal));
EXMEM
EXMEM(.clk(clk),.rstn(rstn),.IDEXRegWrite(IDEXRegWrite),.IDEXMemtoReg(IDEXMemtoReg),.ID
EXJal(IDEXJal),.IDEXMemWrite(IDEXMemWrite),.IDEXMemRead(IDEXMemRead),.IDEXNPC(IDEXNPC),
.ALUOut(ALUOut),.IDEXWriteData(MUXALUSrc2_0),.IDEXRd(IDEXRd),
.EXMEMRegWrite(EXMEMRegWrite),.EXMEMMemtoReg(EXMEMMemtoReg),.EXMEMJal(EXMEMJal),.EXMEMM
emWrite(EXMEMMemWrite),.EXMEMMemRead(EXMEMMemRead),.EXMEMNPC(EXMEMNPC),.EXMEMALUOut(EXM
EMALUOut),.EXMEMWriteData(EXMEMWriteData),.EXMEMRd(EXMEMRd),.IDEXPC(IDEXPC),.EXMEMPC(EX
MEMPC));
ForwardingUnit
ForwardingUnit(.EXMEMRegWrite(EXMEMRegWrite),.MEMWBRegWrite(MEMWBRegWrite),.MEMWBJal(ME
MWBJal),.EXMEMRd(EXMEMRd),.MEMWBRd(MEMWBRd),.IDEXRs1(IDEXRs1),.IDEXRs2(IDEXRs2),.Forwar
dA(ForwardA),.ForwardB(ForwardB));
MUX2x5
MUXDebugReg(.s0(IFIDinstruction[19:15]),.s1(chk_addr[4:0]),.sel(DebugReg),.d(RegAddr));
MUX2x32
```

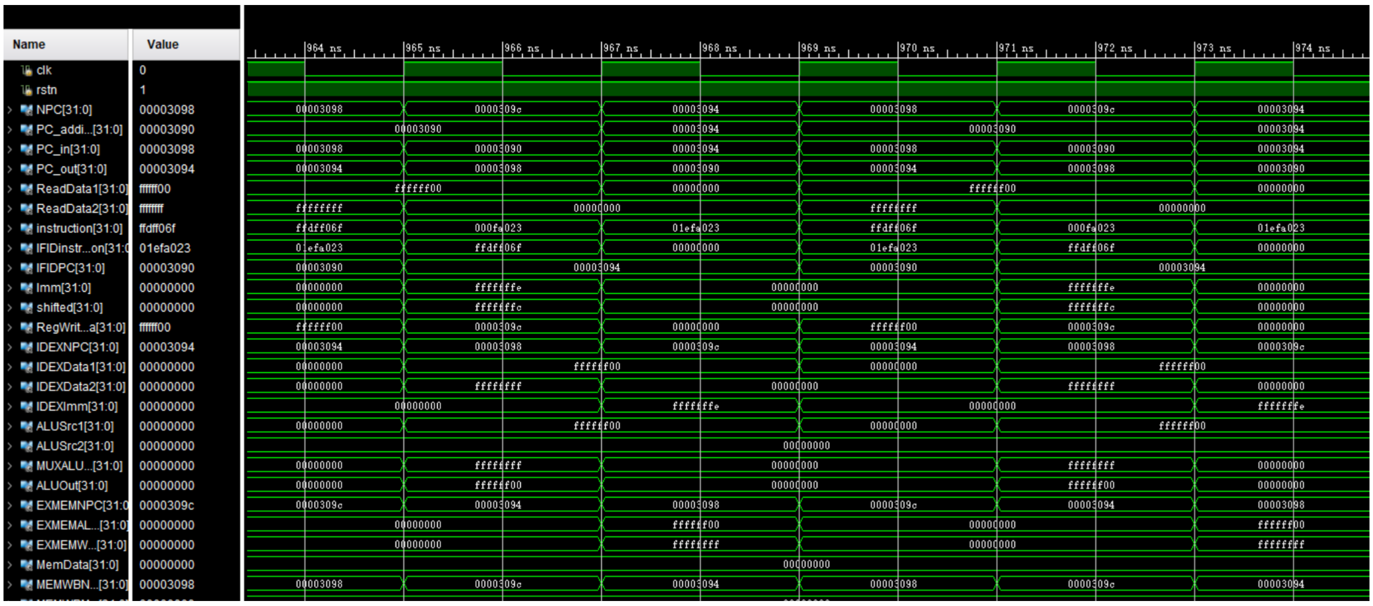
```
MUXDebugMem(.s0(EXMEMALUOut),.s1({20'b0,chk_addr[11:0]}),.sel(DebugMem),.d(MemAddr));
data_memory
data_memory(.clk(clk),.MemWrite(EXMEMMemWrite),.MemRead(EXMEMMemRead|DebugMem),.Address
(MemAddr),.WriteData(EXMEMWriteData),.ReadData(MemData),.io_din(io_din),.io_dout(io_dou
t),.io_addr(io_addr),.io_we(io_we),.io_rd(io_rd));
MEMWB
MEMWB(.clk(clk),.rstn(rstn),.EXMEMRegWrite(EXMEMRegWrite),.EXMEMMemtoReg(EXMEMMemtoReg)
,.EXMEMJal(EXMEMJal),.EXMEMNPC(EXMEMNPC),.MemData(MemData),.EXMEMALUOut(EXMEMALUOut),.E
XMEMRd(EXMEMRd),
.MEMWBRegWrite(MEMWBRegWrite),.MEMWBMemtoReg(MEMWBMemtoReg),.MEMWBJal(MEMWBJal),.MEMWBN
PC(MEMWBNPC),.MEMWBMemData(MEMWBMemData),.MEMWBALUOut(MEMWBALUOut),.MEMWBRd(MEMWBRd),.E
XMEMPC(EXMEMPC),.MEMWBPC(MEMWBPC));
MUX2x32
MUXMemALU(.s0(MEMWBALUOut),.s1(MEMWBMemData),.sel(MEMWBMemtoReg),.d(MUXMemALUOut));
MUX2x32 MUXRightMost(.s0(MUXMemALUOut),.s1(MEMWBNPC),.sel(MEMWBJal),.d(RegWriteData));
chk_data_driver
chk_data_driver(chk_addr,PC_in,PC_out,IFIDPC,instruction,IFIDinstruction,ReadData1,Read
Data2,Imm,ALUOut,MemData,
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jal,RegWrite},chk_data,IDEXPC,EXMEMPC,ME
MWBPC);
endmodule
```

仿真文件：

```
//test_CPU.v
`timescale 1ns / 1ps
module test_CPU();
reg clk,rstn;
wire
[31:0]NPC,PC_addimm,PC_in,PC_out,ReadData1,ReadData2,instruction,IFIDinstruction,IFIDPC
,Imm,shifted,RegWriteData,IDEXNPC,IDEXData1,IDEXData2,IDEXImm,ALUSrc1,ALUSrc2,MUXALUSrc
2_0,
ALUOut,EXMEMNPC,EXMEMALUOut,EXMEMWriteData,MemData,MEMWBNPC,MEMWBMemData,MEMWBALUOut,MU
XMemALUOut;
wire
Flush,StallPC,Branch,equal,Jal,StallIFID,Nop,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite,
Category,
_IDEXRegWrite,_IDEXBranch,_IDEXMemtoReg,_IDEXJal,_IDEXMemRead,_IDEXALUSrc,_IDEXMemWrite
,IDEXBranch,IDEXMemRead,IDEXMemtoReg,IDEXMemWrite,IDEXALUSrc,IDEXJal,IDEXRegWrite,
EXMEMRegWrite,EXMEMMemtoReg,EXMEMJal,EXMEMMemWrite,EXMEMMemRead,MEMWBRegWrite,MEMWBMemt
oReg,MEMWBJal;
wire [1:0]ALUOp,_IDEXALUOp,IDEXALUOp,ForwardA,ForwardB;
wire [2:0]ALUControlSignal;
wire [4:0]IFIDRs1,IFIDRs2,EXMEMRd,IDEXRs1,IDEXRs2,IDEXRd,MEMWBRd;
wire [3:0]IDEXALUCtrl;
assign equal=(ReadData1==ReadData2)?1'b1:1'b0;
assign Flush=~StallPC&(Jal|Branch&equal);
assign shifted=Imm<<1;
assign IFIDRs1=IFIDinstruction[19:15];
assign IFIDRs2=IFIDinstruction[24:20];
MUX2x32 MUXPC(.s0(NPC),.s1(PC_addimm),.sel(Flush),.d(PC_in));
PC_driver PC_driver(.clk(clk),.rstn(rstn),.NPC(PC_in),.StallPC(StallPC),.PC(PC_out));
instruction_memory instruction_memory(.clk(clk),.PC(PC_out),.instruction(instruction));
Add Add1(.a(PC_out),.b(32'h4),.y(NPC));
IFID
```

```
IFID(.clk(clk),.rstn(rstn),.StallIFID(StallIFID),.Flush(Flush),.PC(PC_out),._instructio
n(instruction),.instruction(IFIDinstruction),.IFIDPC(IFIDPC));
control
control(.Opcode(IFIDinstruction[6:0]),.Branch(Branch),.MemRead(MemRead),.MemtoReg(Memto
Reg),.ALUOp(ALUOp),.MemWrite(MemWrite),.ALUSrc(ALUSrc),.Jal(Jal),.RegWrite(RegWrite),.C
ategory(Category));
Imm_Gen Imm_Gen(.instruction(IFIDinstruction),.Imm(Imm));
Add Add2(.a(shifted),.b(IFIDPC),.y(PC_addimm));
registers
registers(.clk(clk),.rstn(rstn),.ra0(IFIDinstruction[19:15]),.ra1(IFIDinstruction[24:20
]),.rd0(ReadData1),.rd1(ReadData2),.wa(MEMWBRd),.wd(RegWriteData),.we(MEMWBRegWrite));
MUX2x9
MUXControl(.s0({Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jal,RegWrite}),.s1(0),.se
l(Nop),.d({_IDEXBranch,_IDEXMemRead,_IDEXMemtoReg,_IDEXALUOp,_IDEXMemWrite,_IDEXALUSrc,
_IDEXJal,_IDEXRegWrite}));
HazardDetectionUnit
HazardDetectionUnit(.Branch(Branch),.EXMEMRegWrite(EXMEMRegWrite),.IDEXMemRead(IDEXMemR
ead),.IDEXRegWrite(IDEXRegWrite),.Category(Category),.IFIDRs1(IFIDRs1),.IFIDRs2(IFIDRs2
),.IDEXRd(IDEXRd),.EXMEMRd(EXMEMRd),.StallPC(StallPC),.StallIFID(StallIFID),.Nop(Nop));
IDEX
IDEX(.clk(clk),.rstn(rstn),.Branch(_IDEXBranch),.MemRead(_IDEXMemRead),.MemtoReg(_IDEXM
emtoReg),.MemWrite(_IDEXMemWrite),.ALUSrc(_IDEXALUSrc),.Jal(_IDEXJal),.RegWrite(_IDEXRe
gWrite),.ALUOp(_IDEXALUOp),.NPC(NPC),.ReadData1(ReadData1),.ReadData2(ReadData2),.Imm(I
mm),.ALUCtrl({IFIDinstruction[30],IFIDinstruction[14:12]}),
.Rs1(IFIDinstruction[19:15]),.Rs2(IFIDinstruction[24:20]),.Rd(IFIDinstruction[11:7]),
.IDEXBranch(IDEXBranch),.IDEXMemRead(IDEXMemRead),.IDEXMemtoReg(IDEXMemtoReg),.IDEXMemW
rite(IDEXMemWrite),.IDEXALUSrc(IDEXALUSrc),.IDEXJal(IDEXJal),.IDEXRegWrite(IDEXRegWrite
),.IDEXALUOp(IDEXALUOp),.IDEXNPC(IDEXNPC),.IDEXData1(IDEXData1),.IDEXData2(IDEXData2),.
IDEXImm(IDEXImm),
.IDEXRs1(IDEXRs1),.IDEXRs2(IDEXRs2),.IDEXRd(IDEXRd),.IDEXALUCtrl(IDEXALUCtrl));
MUX4x32
MUXForward1(.s0(IDEXData1),.s1(RegWriteData),.s2(EXMEMALUOut),.s3(0),.sel(ForwardA),.d(
ALUSrc1));
MUX4x32
MUXForward2(.s0(IDEXData2),.s1(RegWriteData),.s2(EXMEMALUOut),.s3(0),.sel(ForwardB),.d(
MUXALUSrc2_0));
MUX2x32 MUXALUSrc2(.s0(MUXALUSrc2_0),.s1(IDEXImm),.sel(IDEXALUSrc),.d(ALUSrc2));
ALU ALU(.a(ALUSrc1),.b(ALUSrc2),.s(ALUControlSignal),.y(ALUOut));
ALU_control
ALU_control(.ALUOp(IDEXALUOp),.instruction(IDEXALUCtrl),.ALU_sel(ALUControlSignal));
EXMEM
EXMEM(.clk(clk),.rstn(rstn),.IDEXRegWrite(IDEXRegWrite),.IDEXMemtoReg(IDEXMemtoReg),.ID
EXJal(IDEXJal),.IDEXMemWrite(IDEXMemWrite),.IDEXMemRead(IDEXMemRead),.IDEXNPC(IDEXNPC),
.ALUOut(ALUOut),.IDEXWriteData(MUXALUSrc2_0),.IDEXRd(IDEXRd),
.EXMEMRegWrite(EXMEMRegWrite),.EXMEMMemtoReg(EXMEMMemtoReg),.EXMEMJal(EXMEMJal),.EXMEMM
emWrite(EXMEMMemWrite),.EXMEMMemRead(EXMEMMemRead),.EXMEMNPC(EXMEMNPC),.EXMEMALUOut(EXM
EMALUOut),.EXMEMWriteData(EXMEMWriteData),.EXMEMRd(EXMEMRd));
ForwardingUnit
ForwardingUnit(.EXMEMRegWrite(EXMEMRegWrite),.MEMWBRegWrite(MEMWBRegWrite),.MEMWBJal(ME
MWBJal),.EXMEMRd(EXMEMRd),.MEMWBRd(MEMWBRd),.IDEXRs1(IDEXRs1),.IDEXRs2(IDEXRs2),.Forwar
dA(ForwardA),.ForwardB(ForwardB));
data_memory
data_memory(.clk(clk),.MemWrite(EXMEMMemWrite),.MemRead(EXMEMMemRead),.Address(EXMEMALU
Out),.WriteData(EXMEMWriteData),.ReadData(MemData),.io_din(0));
MEMWB
MEMWB(.clk(clk),.rstn(rstn),.EXMEMRegWrite(EXMEMRegWrite),.EXMEMMemtoReg(EXMEMMemtoReg)
,.EXMEMJal(EXMEMJal),.EXMEMNPC(EXMEMNPC),.MemData(MemData),.EXMEMALUOut(EXMEMALUOut),.E
XMEMRd(EXMEMRd),
.MEMWBRegWrite(MEMWBRegWrite),.MEMWBMemtoReg(MEMWBMemtoReg),.MEMWBJal(MEMWBJal),.MEMWBN
PC(MEMWBNPC),.MEMWBMemData(MEMWBMemData),.MEMWBALUOut(MEMWBALUOut),.MEMWBRd(MEMWBRd));
MUX2x32
MUXMemALU(.s0(MEMWBALUOut),.s1(MEMWBMemData),.sel(MEMWBMemtoReg),.d(MUXMemALUOut));
MUX2x32 MUXRightMost(.s0(MUXMemALUOut),.s1(MEMWBNPC),.sel(MEMWBJal),.d(RegWriteData));
initial
begin
    clk=0;
    rstn=0;
end
initial
forever #1 clk=~clk;
initial
    #1.5 rstn=1'b1;
endmodule
```

仿真结果：



Name	Value
clk	0
rstn	1
NPC[31:0]	00003098
PC_addi...[31:0]	00003090
PC_in[31:0]	00003098
PC_out[31:0]	00003094
ReadData1[31:0]	ffffff00
ReadData2[31:0]	ffffffff
instruction[31:0]	ffdff06f
IFIDinstr...on[31:0]	01efa023
IFIDPC[31:0]	00003090
Imm[31:0]	00000000
shifted[31:0]	00000000
RegWrit...a[31:0]	ffffff00
IDEXNPC[31:0]	00003094
IDEXData1[31:0]	00000000
IDEXData2[31:0]	00000000
IDEXImm[31:0]	00000000
ALUSrc1[31:0]	00000000
ALUSrc2[31:0]	00000000
MUXALU...[31:0]	00000000
ALUOut[31:0]	00000000
EXMEMNPC[31:0]	0000309c
EXMEMAL...[31:0]	00000000
EXMEMW...[31:0]	00000000
MemData[31:0]	00000000
MEMWBN...[31:0]	00003098

前面的信号经检验完全正常，最后IF/ID段的PC在0x3090和0x3094来回切换，符合预期结果。

PDU

```
//PDU.v
module  PDU(
        input clk,              //clk100mhz
        input rstn,             //cpu_resetn

        input step,             //btnu
        input cont,             //btnd
        input chk,              //btnr
        input data,             //btnc
        input del,              //btnl
        input [15:0] x,         //sw15-0

        output stop,            //led16r
        output [15:0] led,      //led15-0
        output [7:0] an,        //an7-0
        output [6:0] seg,       //ca-cg
```

```
    output [2:0] seg_sel, //led17

    output clk_cpu,       //cpu's clk
    output rst_cpu,       //cpu's rst

    //IO_BUS
    input [7:0] io_addr,
    input [31:0] io_dout,
    input io_we,
    input io_rd,
    output [31:0] io_din,

    //Debug_BUS
    input [31:0] chk_pc,    //连接CPU的npc
    output [15:0] chk_addr,
    input [31:0] chk_data
);

reg [15:0] rstn_r;
wire rst;               //复位信号，高电平有效
wire clk_pdu;           //PDU工作时钟
wire clk_db;            //去抖动计数器时钟
reg run_r, run_n;

reg [19:0] cnt_clk_r;   //时钟分频、数码管刷新计数器
reg [4:0] cnt_sw_db_r;
reg [15:0] x_db_r, x_db_1r;
reg xx_r, xx_1r;
wire x_p;
reg [3:0] x_hd_t;

wire [4:0] btn;
reg [4:0] cnt_btn_db_r;
reg [4:0] btn_db_r, btn_db_1r;
wire step_p, cont_p, chk_p, data_p, del_p;

reg [15:0] led_data_r;  //指示灯led15-0数据
reg [31:0] seg_data_r;  //数码管输出数据
reg seg_rdy_r;          //数码管准备好标志
reg [31:0] swx_data_r;  //开关输入数据
reg swx_vld_r;          //开关输入有效标志
reg [31:0] cnt_data_r;  //性能计数器数据

reg [31:0] tmp_r;       //临时编辑数据
reg [31:0] brk_addr_r;  //断点地址
reg [15:0] chk_addr_r;  //查看地址
reg [31:0] io_din_t;

reg led_sel_r;
reg [2:0] seg_sel_r;
reg [31:0] disp_data_t;
reg [7:0] an_t;
reg [3:0] hd_t;
reg [6:0] seg_t;

assign rst = rstn_r[15];    //经处理后的复位信号，高电平有效
assign rst_cpu = rst;
```

```
    assign clk_pdu = cnt_clk_r[1];      //PDU工作时钟25MHz
    assign clk_db = cnt_clk_r[16];      //去抖动计数器时钟763Hz (周期约1.3ms)
    assign clk_cpu = clk_pdu & run_n;   //CPU工作时钟

    //降低指示灯和数码管亮度 (保护眼睛)
    assign stop = ~run_r & (& cnt_clk_r[15:11]);
    assign led = ((led_sel_r)? chk_addr : led_data_r) & {{16{&cnt_clk_r[15:13]}}};
    assign an = an_t;
    assign seg = seg_t;
    assign seg_sel = seg_sel_r & {{3{&cnt_clk_r[15:11]}}};

    assign io_din = io_din_t;
    assign chk_addr = chk_addr_r;

    assign btn ={step, cont, chk, data, del};
    assign x_p = xx_r ^ xx_1r;
    assign step_p = btn_db_r[4] & ~ btn_db_1r[4];
    assign cont_p = btn_db_r[3] & ~ btn_db_1r[3];
    assign chk_p = btn_db_r[2] & ~ btn_db_1r[2];
    assign data_p = btn_db_r[1] & ~ btn_db_1r[1];
    assign del_p = btn_db_r[0] & ~ btn_db_1r[0];


    ///////////////////////////////////////////////
    //复位处理: 异步复位、同步和延迟释放
    ///////////////////////////////////////////////
    always @(posedge clk, negedge rstn) begin
        if (!rstn)
            rstn_r <= 16'hFFFF;
        else
            rstn_r <= {rstn_r[14:0], 1'b0};
    end


    ///////////////////////////////////////////////
    //时钟分频
    ///////////////////////////////////////////////
    always @(posedge clk or posedge rst) begin
        if (rst)
            cnt_clk_r <= 20'h0;
        else
            cnt_clk_r <= cnt_clk_r + 20'h1;
    end


    ///////////////////////////////////////////////
    //开关sw去抖动
    ///////////////////////////////////////////////
    always @(posedge clk_db or posedge rst) begin
        if (rst)
            cnt_sw_db_r <= 5'h0;
        else if ((|(x ^ x_db_r)) & (~cnt_sw_db_r[4]))
            cnt_sw_db_r <= cnt_sw_db_r + 5'h1;
        else
            cnt_sw_db_r <= 5'h0;
    end
    always@(posedge clk_db or posedge rst) begin
        if (rst) begin
            x_db_r <= x;
            x_db_1r <= x;
            xx_r <= 1'b0;
        end
        else if (cnt_sw_db_r[4]) begin    //信号稳定约21ms后输出
            x_db_r <= x;
            x_db_1r <= x_db_r;
            xx_r <= ~xx_r;
        end
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            xx_1r <= 1'b0;
        else
            xx_1r <= xx_r;
    end


    ///////////////////////////////////////////////
    //按钮btn去抖动
    ///////////////////////////////////////////////
    always @(posedge clk_db or posedge rst) begin
        if (rst)
            cnt_btn_db_r <= 5'h0;
        else if ((|(btn ^ btn_db_r)) & (~cnt_btn_db_r[4]))
            cnt_btn_db_r <= cnt_btn_db_r + 5'h1;
        else
            cnt_btn_db_r <= 5'h0;
    end

    always@(posedge clk_db or posedge rst) begin
        if (rst)
            btn_db_r <= btn;
        else if (cnt_btn_db_r[4])
            btn_db_r <= btn;
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            btn_db_1r <= btn;
        else
            btn_db_1r <= btn_db_r;
    end


    ///////////////////////////////////////////////
    //控制CPU运行方式
    ///////////////////////////////////////////////
    reg [1:0] cs, ns;
    parameter STOP = 2'b00, STEP = 2'b01, RUN = 2'b10;

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            cs <= STOP;
        else
            cs <= ns;
    end

    always @* begin
        ns = cs;
        case (cs)
            STOP: begin
                if (step_p)
                    ns = STEP;
                else if (cont_p)
                    ns = RUN;
            end
            STEP:
                ns = STOP;
            RUN: begin
                if (brk_addr_r == chk_pc)
                    ns = STOP;
            end
            default:
                ns = STOP;
        endcase
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            run_r <= 1'b0;
        else if (ns == STOP)
            run_r <= 1'b0;
        else
            run_r <= 1'b1;
    end

    always @(negedge clk_pdu or posedge rst) begin
        if (rst)
            run_n <= 1'b0;
        else
            run_n <= run_r;
    end


    ///////////////////////////////////////////////
    //CPU输入/输出
    ///////////////////////////////////////////////
    always @(posedge clk_cpu or posedge rst) begin    //CPU输出    //fix
        if (rst) begin
            led_data_r <= 16'hFFFF;
            seg_data_r <= 32'h12345678;
        end
        else if (io_we) begin
            case (io_addr)
                8'h00:
                    led_data_r <= io_dout;
                8'h0C:
                    seg_data_r <= io_dout;
                default: ;
            endcase
        end
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            seg_rdy_r <= 1;
        else if (run_r & io_we & (io_addr == 8'h0C))    //fix
            seg_rdy_r <= 0;
        else if (x_p | del_p)
            seg_rdy_r <= 1;
    end

    always @(*) begin    //CPU输入
        case (io_addr)
            8'h04:
                io_din_t = {{11{1'b0}}, step, cont, chk, data, del, x};
            8'h08:
                io_din_t = {{31{1'b0}}, seg_rdy_r};
            8'h10:
                io_din_t = {{31{1'b0}}, swx_vld_r};
            8'h14:
                io_din_t = swx_data_r;
            8'h18:
                io_din_t = cnt_data_r;
            default:
                io_din_t = 32'h0;
        endcase
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            swx_vld_r <= 0;
        else if (data_p & ~swx_vld_r)
            swx_vld_r <= 1;
        else if (run_r & io_rd & (io_addr == 8'h14))    //fix
            swx_vld_r <= 0;
    end


    ///////////////////////////////////////////////
    //性能计数器
    ///////////////////////////////////////////////
    always@(posedge clk_cpu or posedge rst) begin
        if(rst)
            cnt_data_r <= 32'h0;
        else
            cnt_data_r <= cnt_data_r + 32'h1;
    end


    ///////////////////////////////////////////////
    //开关编辑数据
    ///////////////////////////////////////////////
    always @* begin    //开关输入编码
        case (x_db_r ^ x_db_1r )
            16'h0001:
                x_hd_t = 4'h0;
            16'h0002:
                x_hd_t = 4'h1;
            16'h0004:
                x_hd_t = 4'h2;
            16'h0008:
                x_hd_t = 4'h3;
            16'h0010:
                x_hd_t = 4'h4;
            16'h0020:
                x_hd_t = 4'h5;
            16'h0040:
                x_hd_t = 4'h6;
            16'h0080:
                x_hd_t = 4'h7;
            16'h0100:
                x_hd_t = 4'h8;
            16'h0200:
                x_hd_t = 4'h9;
            16'h0400:
                x_hd_t = 4'hA;
            16'h0800:
                x_hd_t = 4'hB;
            16'h1000:
                x_hd_t = 4'hC;
            16'h2000:
                x_hd_t = 4'hD;
            16'h4000:
                x_hd_t = 4'hE;
            16'h8000:
                x_hd_t = 4'hF;
            default:
                x_hd_t = 4'h0;
        endcase
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            tmp_r <= 32'h0;
        else if (x_p)
            tmp_r <= {tmp_r[27:0], x_hd_t};      //x_hd_t + tmp_r << 4
        else if (del_p)
            tmp_r <= {{4{1'b0}}, tmp_r[31:4]};   //tmp_r >> 4
        else if ((cont_p & ~run_r) | (data_p & ~swx_vld_r))
            tmp_r <= 32'h0;
        else if (chk_p & ~run_r)
            tmp_r <= tmp_r + 32'h1;
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst) begin
            chk_addr_r <= 16'h0;
            brk_addr_r <= 32'h0;
        end
        else if (data_p & ~swx_vld_r)
            swx_data_r <= tmp_r;
        else if (cont_p & ~run_r)
            brk_addr_r <= tmp_r;
        else if (chk_p & ~run_r)
            chk_addr_r <= tmp_r;
    end


    ///////////////////////////////////////////////
    //led15-0指示灯显示
    ///////////////////////////////////////////////
    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            led_sel_r <= 1'b0;
        else if (run_r & io_we && (io_addr == 8'h00))    //fix
            led_sel_r <= 1'b0;
        else if (chk_p & ~run_r)
            led_sel_r <= 1'b1;
    end


    ///////////////////////////////////////////////
    //数码管多用途显示
    ///////////////////////////////////////////////
    always @(posedge clk_pdu or posedge rst) begin    //数码管显示数据选择
        if (rst)
            seg_sel_r <= 3'b001;
        else if (run_r & io_we & (io_addr == 8'h0C))    //fix
            seg_sel_r <= 3'b001;   //输出
        else if (x_p | del_p)
            seg_sel_r <= 3'b010;   //编辑
        else if (chk_p & ~run_r)
            seg_sel_r <= 3'b100;   //调试
    end

    always @* begin
        case (seg_sel_r)
            3'b001:
                disp_data_t = seg_data_r;
            3'b010:
                disp_data_t = tmp_r;
            3'b100:
                disp_data_t = chk_data;
            default:
                disp_data_t = tmp_r;
        endcase
    end

    always @(*) begin          //数码管扫描
        an_t <= 8'b1111_1111;
        hd_t <= disp_data_t[3:0];
        if (&cnt_clk_r[16:15])    //降低亮度
        case (cnt_clk_r[19:17])   //刷新频率约为95Hz
            3'b000: begin
                an_t <= 8'b1111_1110;
                hd_t <= disp_data_t[3:0];
            end
            3'b001: begin
                an_t <= 8'b1111_1101;
                hd_t <= disp_data_t[7:4];
            end
            3'b010: begin
                an_t <= 8'b1111_1011;
                hd_t <= disp_data_t[11:8];
            end
            3'b011: begin
                an_t <= 8'b1111_0111;
                hd_t <= disp_data_t[15:12];
            end
            3'b100: begin
                an_t <= 8'b1110_1111;
                hd_t <= disp_data_t[19:16];
            end
            3'b101: begin
                an_t <= 8'b1101_1111;
                hd_t <= disp_data_t[23:20];
            end
            3'b110: begin
                an_t <= 8'b1011_1111;
                hd_t <= disp_data_t[27:24];
            end
            3'b111: begin
                an_t <= 8'b0111_1111;
                hd_t <= disp_data_t[31:28];
            end
            default:
                ;
        endcase
    end

    always @ (*) begin    //7段译码
        case(hd_t)
            4'b1111:
                seg_t = 7'b0111000;
            4'b1110:
                seg_t = 7'b0110000;
            4'b1101:
                seg_t = 7'b1000010;
            4'b1100:
                seg_t = 7'b0110001;
            4'b1011:
                seg_t = 7'b1100000;
            4'b1010:
                seg_t = 7'b0001000;
            4'b1001:
                seg_t = 7'b0001100;
            4'b1000:
                seg_t = 7'b0000000;
            4'b0111:
                seg_t = 7'b0001111;
            4'b0110:
                seg_t = 7'b0100000;
            4'b0101:
                seg_t = 7'b0100100;
            4'b0100:
                seg_t = 7'b1001100;
            4'b0011:
                seg_t = 7'b0000110;
            4'b0010:
                seg_t = 7'b0010010;
            4'b0001:
                seg_t = 7'b1001111;
            4'b0000:
                seg_t = 7'b0000001;
            default:
                seg_t = 7'b1111111;
        endcase
    end

endmodule
```

主模块

```
//top.v
`timescale 1ns / 1ps
module  top (
  input clk,     //clk100mhz
  input rstn,    //cpu_resetn

  input step,    //btnu
  input cont,    //btnd
  input chk,     //btnr
  input data,    //btnc
  input del,     //btnl
  input [15:0]  x,   //sw15-0

  output stop,       //led16r
  output [15:0] led,     //led15-0
  output [7:0] an,       //an7-0
  output [6:0] seg,      //ca-cg
  output [2:0] seg_sel); //led17
wire[7:0] io_addr;
wire[31:0] io_dout,io_din,chk_pc,chk_data;
wire io_we,io_rd,rst_cpu,clk_cpu;
wire[15:0] chk_addr;
PDU
PDU(.clk(clk),.rstn(rstn),.step(step),.cont(cont),.chk(chk),.data(data),.del(del),.x(x)
,.stop(stop),.led(led),.an(an),.seg(seg),.seg_sel(seg_sel),.clk_cpu(clk_cpu),
.rst_cpu(rst_cpu),.io_addr(io_addr),.io_dout(io_dout),.io_we(io_we),.io_rd(io_rd),.io_d
in(io_din),.chk_pc(chk_pc),.chk_addr(chk_addr),.chk_data(chk_data));
CPU CPU(clk_cpu,rst_cpu,io_addr,io_dout,io_we,io_rd,io_din,chk_pc,chk_addr,chk_data);
endmodule
```

出现运行

测试代码：

```
#测试时单步运行
#假定MMIO的起始地址为0xff00
```

```
#test no hazards
addi x30, x0, -1    #x30=0xffffffff
addi x31, x0, -256    #MMIO base address
addi x1, x0, 1    #x1=1
addi x2, x0, 2    #x2=2
addi x3, x0, 3    #x3=3
addi x4, x0, 4    #x4=4
sw x1, 0x00(x31)    #led_data=1
sw x2, 0x00(x31)    #led_data=2
sw x3, 0x00(x31)    #led_data=3
sw x4, 0x00(x31)    #led_data=4
sw x30, 0x00(x31)    #led_data=0xffff
sw x31, 0x00(x31)    #led_data=oxff00

#test data hazards
add x5, x4, x1    #x5=5
add x6, x5, x1    #x6=6
add x7, x5, x2    #x7=7
add x8, x1, x7    #x8=8
add x9, x2, x7    #x9=9
sw x9, 0x00(x31)    #led_data=9
sw x8, 0x00(x31)    #led_data=8
sw x7, 0x00(x31)    #led_data=7
sw x6, 0x00(x31)    #led_data=6
sw x5, 0x00(x31)    #led_data=5

add x10, x5, x6    #x6=11
add x10, x10, x7    #x6=18
add x10, x10, x8    #x6=26
add x10, x10, x9    #x6=35
sw x10, 0x00(x31)    #led_data=35


#test load-use hazard
add x10, x0, x0      #fig
loop:
lw x11, 0x04(x31)    #x11=swt_data
addi x12, x11, 1    #x12=swt_data+1
addi x13, x12, -1    #x13=swt_data
sw x13, 0x00(x31)    #led_data=swt_data


#test control hazard
beq x13, x0, stop    #if (swt_data==0) stop
add x10, x10, x13
sw x10, 0x00(x31)    #led_data=accum of swt_data
jal x0, loop

stop:
sw x30, 0x00(x31)    #led_data=0xffff
jal x0, stop

#do not execute
err:
sw x0, 0x00(x31)    #led_data=0x0000
```

```
    sw x30, 0x00(x31)    #led_data=0xffff
    jal x0, err
```

text.coe文件：

```
memory_initialization_radix  = 16;
memory_initialization_vector =
fff00f13
f0000f93
00100093
00200113
00300193
00400213
001fa023
002fa023
003fa023
004fa023
01efa023
01ffa023
001202b3
00128333
002283b3
00708433
007104b3
009fa023
008fa023
007fa023
006fa023
005fa023
00628533
00750533
00850533
00950533
00afa023
00000533
004fa583
00158613
fff60693
00dfa023
00068863
00d50533
00afa023
fe5ff06f
01efa023
ffdff06f
000fa023
01efa023
ff9ff06f
```

当开关全置为0时，程序跳出循环结束，LED等全亮；否则用单步调试可以看到程序将陷入一个循环，每次LED灯的示数增加按钮表示的那么多（如果不用单步调试直接运行，会发现开关置1的最低位对应数量的LED等全部熄灭，其余LED等变暗（实际上是一直在闪）。开关全部置0并使程序终止后，使用单步调试，会发现LED等不闪，即没有进入err分支，程序运行正常。

更多

本次寄存器堆设计方式有点特别，使用了下降沿写。这样WB阶段数据加载以后，在时钟下降沿会被及时写入寄存器，在下一个时钟周期刚开始流水线寄存器就能存入正确的值。