2023秋P5·CPU设计文档

Part 1 CPU设计草稿

1.1 总体概述

以下是参考的结构图：

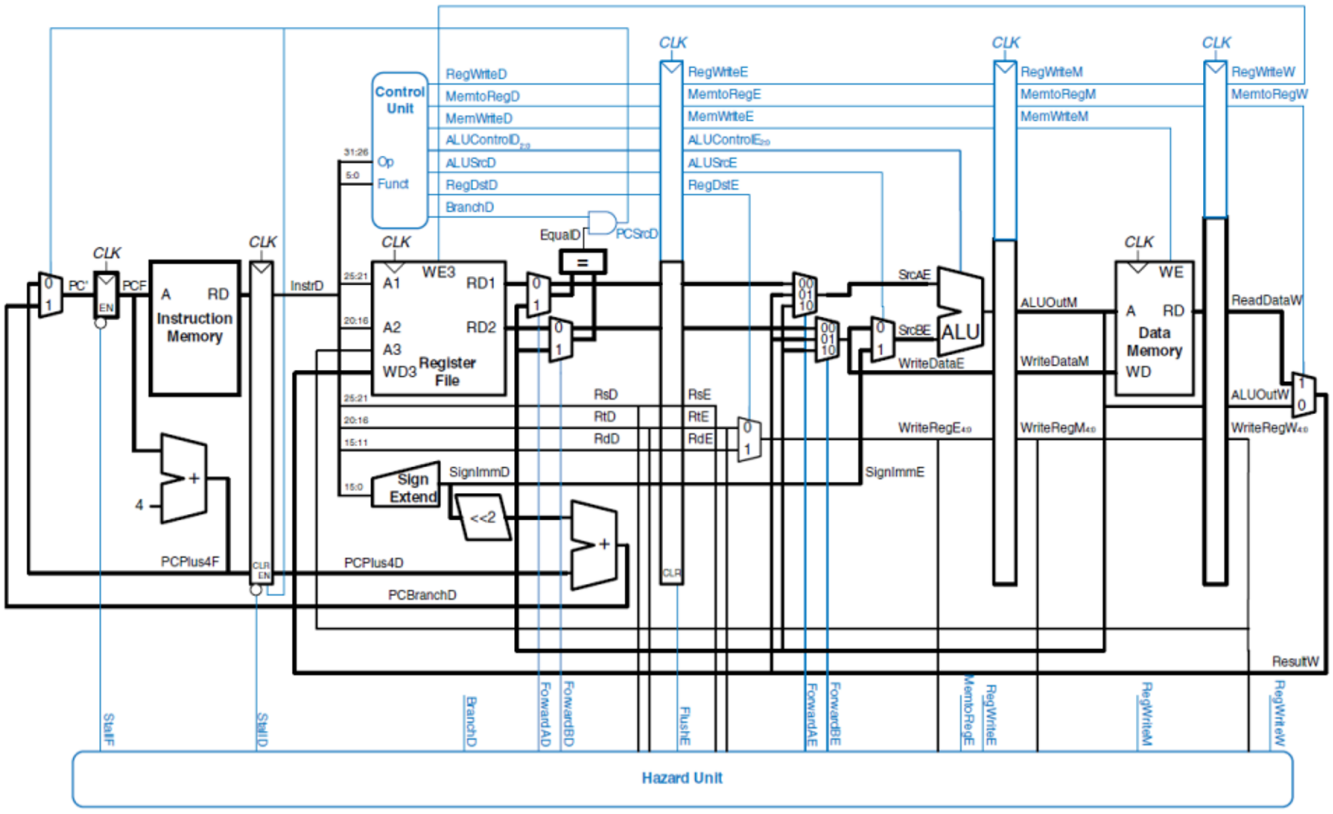

Figure 7.58 Pipelined processor with full hazard handling

1.1.1 CPU类型

本CPU为**五级流水线MIPS - CPU**，目前由**Verilog**的**ISE**工具链实现。

1.1.2 CPU结构

CPU包含**IFU、GRF、CMP、EXT、ALU、DM、Controller、各级之间的流水线寄存器**等多个模块**以及冒险处理单元。**

> MUXs.v文件集成了4个多路选择器。
> forward_MUXs.v可以看作是冒险处理单元的附属模块，集成了转发机制需要的所有多路选择器。

1.1.3 CPU支持的指令集

{**add, sub, ori, lw, sw, beq, lui, jal, jr, nop**}

> 要求中字面上是add和sub指令，是为了方便以后扩展功能。但是实际上本次开发只需考虑无符号数即可，更接近于addu和subu指令。

1.2 模块设计

1.2.1 IFU

1.2.1.1 端口定义

端口名	方向	位宽	信号描述
clk	I	1	时钟信号
reset	I	1	同步复位信号，复位使PC的值为0x3000（起始地址）
enable	I	1	使能信号，当enable为1时，IFU开始工作；否则，IFU停止工作
zero	I	1	相等状态码：1，相等；0，不相等
offset	I	16	跳转偏移量
instr_index	I	26	跳转目标地址
rsIn	I	32	GRF[rs]读出的值
D_pcPlus4	I	32	D阶段(译码阶段)的pc+4值
jumpOp	I	3	跳转操作类型：000，无跳转；001，beq；010；jal；011，jr
nInstr	O	32	下一条32位指令数据
pc	O	32	PC值，用于显示当前指令的地址(按字节)

> 在本文档中，形如`D_xxx`的信号表示该信号是输入信号且来自D阶段，形如`xxx_D`的信号表示该信号是输出信号且送往D阶段。
> 对于其他阶段也是如此。
> 该部分被分为**pc、npc、im三个子模块**,本设计将其写在同一个文件中，用注释区分。

1.2.1.2 功能定义

- 同步复位
  - 当reset为1时，将PC的值置为0x3000（起始地址）
- 计算下一条指令地址（调整PC值）
  - 当beq指令有效时（zero == 1 && jumpOp == 001），PC = PC + 4 + offset * 4
  - 当jal指令有效时（jumpOp == 010），PC = pc[31:28] || instr_index || 00
  - 当jr指令有效时（jumpOp == 011），PC = rsIn
  - 否则，PC = PC + 4
- 取指令
  - 根据PC的值，从IM中读出下一条指令nInstr

1.2.1.3 模块代码

```
`timescale 1ns / 1ps
module IFU (
    input             clk,
    input             reset,
    input             enable,
    input             zero,
    input      [15:0] offset,
    input      [25:0] instr_index,
    input      [31:0] rsIn,
    input      [31:0] D_pcPlus4,
    input      [ 2:0] jumpOp,
    output reg [31:0] nInstr,
    output reg [31:0] pc
);
  initial begin
    pc = 32'h00003000;
  end
  reg [31:0] rom  [0:4095];
  reg [31:0] pc_;
  reg [11:0] addr;
```

```
    // Sub-module1 npc
    reg [31:0] npc;
    always @(*) begin
      case (jumpOp)
        3'd0: begin
          // none
          npc = pc + 4;
        end
        3'd1: begin
          //beq
          if (zero == 1'b1) begin
            npc = D_pcPlus4 + ({{16{offset[15]}}, offset} << 2);
          end else begin
            npc = pc + 4;
          end
        end
        3'd2: begin
          //jal
          npc = {D_pcPlus4[31:28], instr_index, {2{1'b0}}};
        end
        3'd3: begin
          //jr
          npc = rsIn;
        end
        default: npc = pc + 4;
      endcase
    end

    // Sub-module2 pc
    always @(posedge clk) begin
      if (reset == 1'b1) begin
        pc <= 32'h00003000;
      end else if (enable) begin
        pc <= npc;
      end
    end

    // Sub-module3 im
    initial begin
      $readmemh("code.txt", rom);
    end
    always @(*) begin
      pc_    = pc - 32'h00003000;
      addr   = pc_[13:2];
      nInstr = rom[addr];
    end
  endmodule
```

1.2.2 IF_ID(流水线寄存器)

1.2.2.1 端口定义

端口名	方向	位宽	信号描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
enable	I	1	使能信号
F_pc	I	32	F级（取指阶段）的pc值
F_nInstr	I	32	F级（取指阶段）的nInstr值

端口名	方向	位宽	信号描述
pc_D	O	32	D级（译码阶段）的pc值
pcPlus4_D	O	32	D级（译码阶段）的pc+4值
pcPlus8_D	O	32	D级（译码阶段）的pc+8值
nInstr_D	O	32	D级（译码阶段）的nInstr值

1.2.2.2 功能定义

- 暂存F阶段结果，向D阶段提供数据。

1.2.2.3 模块代码

```
`timescale 1ns / 1ps
module IF_ID (
    input             clk,
    input             reset,
    input             enable,
    input      [31:0] F_pc,
    input      [31:0] F_nInstr,
    output reg [31:0] pc_D,
    output reg [31:0] pcPlus4_D,
    output reg [31:0] pcPlus8_D,
    output reg [31:0] nInstr_D
);
  always @(posedge clk) begin
    if (reset == 1) begin
      pc_D      <= 32'h00003000;
      pcPlus4_D <= 32'h00003004;
      pcPlus8_D <= 32'h00003008;
      nInstr_D  <= 32'h00000000;
    end else if (enable) begin
      pc_D      <= F_pc;
      pcPlus4_D <= F_pc + 4;
      pcPlus8_D <= F_pc + 8;
      nInstr_D  <= F_nInstr;
    end
  end
endmodule
```

1.2.3 GRF

1.2.3.1 端口定义

端口名	方向	位宽	信号描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
WE	I	1	写使能信号：1，允许写入；0，不可写入
A1	I	5	读出数据的地址1
A2	I	5	读出数据的地址2
A3	I	5	写入数据的地址
WD	I	32	写入数据

端口名	方向	位宽	信号描述
RD1	O	32	读出数据1
RD2	O	32	读出数据2

1.2.3.2 功能定义

- 读数据
  - 读数据1：根据地址A1，读出对应的数据RD1
  - 读数据2：根据地址A2，读出对应的数据RD2
- 写数据
  - 写数据：WE为1、reset不为1且时钟上升来临时，根据地址A3，写入数据WD
  - 会按照教程所要求格式输出写入信息
- 同步复位
  - 当reset为1时，将所有寄存器清零

1.2.3.3 模块代码

```
`timescale 1ns / 1ps
module GRF (
    input         clk,
    input         reset,
    input         WE,
    input  [ 4:0] A1,
    input  [ 4:0] A2,
    input  [ 4:0] A3,
    input  [31:0] WD,
    input  [31:0] pc,
    output [31:0] RD1,
    output [31:0] RD2
);
  reg     [31:0] regfile[0:31];
  integer        i = 0;
  always @(posedge clk) begin
    if (reset == 1'b1) begin
      for (i = 0; i < 32; i = i + 1) begin
        regfile[i] <= 32'd0;
      end
    end else begin
      if (WE == 1'b1) begin
        if (A3 != 5'd0) begin
          regfile[A3] <= WD;
          //display for test
          $display("%d@%h: $%d <= %h", $time, pc, A3, WD);
        end else begin
          regfile[0] <= 32'd0;
        end
      end
    end
  end
  assign RD1 = (WE == 1 && A3 != 5'd0 && A3 == A1) ? WD : regfile[A1];
  assign RD2 = (WE == 1 && A3 != 5'd0 && A3 == A2) ? WD : regfile[A2];
endmodule
```

1.2.4 CMP

1.2.4.1 端口定义

端口名	方向	位宽	信号描述
A	I	32	第一个操作数（A）
B	I	32	第二个操作数（B）
equal	O	1	相等状态码：1，相等；0，不相等；

1.2.4.2 功能定义

- 判断相等
  - 判断A和B是否相等，相等时equal为1，不相等时equal为0

1.2.4.3 模块代码

```
`timescale 1ns / 1ps
module CMP (
    input  [31:0] A,
    input  [31:0] B,
    output        equal
);
  assign equal = (A == B) ? 1'b1 : 1'b0;
endmodule
```

1.2.5 EXT

1.2.5.1 端口定义

端口名	方向	位宽	信号描述
dataIn	I	16	输入数据,16位立即数
extOp	I	2	扩展操作类型：00，无符号扩展；01，符号扩展；10，将低16位加载至高16位且低位补0
dataOut	O	32	输出数据,32位立即数

1.2.5.2 功能定义

- 位扩展
  - 将16位的立即数扩展为32位的立即数，提供符号扩展、无符号扩展、加载至高位三种方式

1.2.5.3 模块代码

```
`timescale 1ns / 1ps
module EXT (
    input      [15:0] dataIn,
    input      [ 1:0] extOp,
    output reg [31:0] dataOut
);
  always @(*) begin
    if (extOp == 2'b00) begin
      dataOut = {{16{1'b0}}, dataIn};
    end else if (extOp == 2'b01) begin
      dataOut = {{16{dataIn[15]}}, dataIn};
    end else if (extOp == 2'b10) begin
      dataOut = {dataIn, {16{1'b0}}};
    end else begin
      dataOut = 0;
    end
```

```
    end
endmodule
```

1.2.6 ID_EX(流水线寄存器)

1.2.6.1 端口定义

端口名	方向	位宽	信号描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
enable	I	1	使能信号
D_nInstr	I	32	D阶段的nInstr值
D_pc	I	32	D阶段的pc值
D_pcPlus4	I	32	D阶段的pc+4值
D_pcPlus8	I	32	D阶段的pc+8值
D_RD1	I	32	D阶段从rs寄存器读出的数据RD1
D_RD2	I	32	D阶段从rt寄存器读出的数据RD2
D_dataOut	I	32	D阶段的扩展后的立即数dataOut
nInstr_E	O	32	E阶段的nInstr值
pc_E	O	32	E阶段的pc值
pcPlus4_E	O	32	E阶段的pc+4值
pcPlus8_E	O	32	E阶段的pc+8值
rsData_E	O	32	E阶段的从rs寄存器读出的数据
rtData_E	O	32	E阶段的从rt寄存器读出的数据
extImm_E	O	32	E阶段的扩展后的立即数extImm

1.2.6.2 功能定义

- 暂存D阶段结果，向E阶段提供数据。

1.2.6.3 模块代码

```
`timescale 1ns / 1ps
module ID_EX (
    input             clk,
    input             reset,
    input             enable,
    input      [31:0] D_nInstr,
    input      [31:0] D_pc,
    input      [31:0] D_pcPlus4,
    input      [31:0] D_pcPlus8,
    input      [31:0] D_RD1,
    input      [31:0] D_RD2,
    input      [31:0] D_dataOut,
    output reg [31:0] nInstr_E,
    output reg [31:0] pc_E,
    output reg [31:0] pcPlus4_E,
    output reg [31:0] pcPlus8_E,
```

```
    output reg [31:0] rsData_E,
    output reg [31:0] rtData_E,
    output reg [31:0] extImm_E
);

  always @(posedge clk) begin
    if (reset) begin
      nInstr_E  <= 0;
      pc_E      <= 32'h00003000;
      pcPlus4_E <= 32'h00003004;
      pcPlus8_E <= 32'h00003008;
      rsData_E  <= 0;
      rtData_E  <= 0;
      extImm_E  <= 0;
    end else if (enable) begin
      nInstr_E  <= D_nInstr;
      pc_E      <= D_pc;
      pcPlus4_E <= D_pcPlus4;
      pcPlus8_E <= D_pcPlus8;
      rsData_E  <= D_RD1;
      rtData_E  <= D_RD2;
      extImm_E  <= D_dataOut;
    end
  end
endmodule
```

1.2.7 ALU

1.2.7.1 端口定义

端口名	方向	位宽	信号描述
A	I	32	第一个操作数
B	I	32	第二个操作数
aluOp	I	3	ALU操作类型：000，加；001，减；010，按位与；011，按位或；100，判断相等
R	O	32	运算结果
zero	O	1	相等状态码：1，相等；0，不相等

1.2.7.2 功能定义

- 算术运算
  - 加法：R = A+B
  - 减法：R = A-B
- 位运算
  - 按位与：R = A&B
  - 按位或：R = A|B
- 判断相等
  - 判断A和B是否相等，相等时zero为1，不相等时zero为0；**此时R输出端保持原值**

1.2.7.3 模块代码

```
`timescale 1ns / 1ps
module ALU (
    input      [31:0] A,
    input      [31:0] B,
    input      [ 2:0] aluOp,
    output reg [31:0] R,
```

```
    output            zero
);
  always @(*) begin
    case (aluOp)
      3'd0: R = A + B;
      3'd1: R = A - B;
      3'd2: R = A & B;
      3'd3: R = A | B;
      3'd4: R = R;
      default: R = R;
    endcase
  end
  assign zero = (A == B) ? 1 : 0;
endmodule
```

1.2.8 EX_MEM(流水线寄存器)

1.2.8.1 端口定义

端口名	方向	位宽	信号描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
enable	I	1	使能信号
E_nInstr	I	32	E阶段的nInstr值
E_pc	I	32	E阶段的pc值
E_pcPlus4	I	32	E阶段的pc+4值
E_pcPlus8	I	32	E阶段的pc+8值
E_aluRes	I	32	E阶段的ALU运算结果aluRes
E_rtData	I	32	E阶段的从rt寄存器读出的数据
E_extImm	I	32	E阶段的扩展后的立即数extImm
nInstr_M	O	32	M阶段的nInstr值
pc_M	O	32	M阶段的pc值
pcPlus4_M	O	32	M阶段的pc+4值
pcPlus8_M	O	32	M阶段的pc+8值
aluRes_M	O	32	M阶段的ALU运算结果aluRes
rtData_M	O	32	M阶段的从rt寄存器读出的数据
extImm_M	O	32	M阶段的扩展后的立即数extImm

1.2.8.2 功能定义

- 暂存E阶段结果，向M阶段提供数据。

1.2.8.3 模块代码

```
`timescale 1ns / 1ps
module EX_MEM (
    input             clk,
    input             reset,
```

```
    input             enable,
    input      [31:0] E_nInstr,
    input      [31:0] E_pc,
    input      [31:0] E_pcPlus4,
    input      [31:0] E_pcPlus8,
    input      [31:0] E_rtData,
    input      [31:0] E_aluRes,
    input      [31:0] E_extImm,
    output reg [31:0] nInstr_M,
    output reg [31:0] pc_M,
    output reg [31:0] pcPlus4_M,
    output reg [31:0] pcPlus8_M,
    output reg [31:0] rtData_M,
    output reg [31:0] aluRes_M,
    output reg [31:0] extImm_M
);
  always @(posedge clk) begin
    if (reset) begin
      nInstr_M  <= 0;
      pc_M      <= 0;
      pcPlus4_M <= 0;
      pcPlus8_M <= 0;
      rtData_M  <= 0;
      aluRes_M  <= 0;
      extImm_M  <= 0;
    end else if (enable) begin
      nInstr_M  <= E_nInstr;
      pc_M      <= E_pc;
      pcPlus4_M <= E_pcPlus4;
      pcPlus8_M <= E_pcPlus8;
      rtData_M  <= E_rtData;
      aluRes_M  <= E_aluRes;
      extImm_M  <= E_extImm;
    end
  end
endmodule
```

1.2.9 DM

1.2.9.1 端口定义

端口名	方向	位宽	信号描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
WE	I	1	写使能信号：1，允许写入；0，不可写入
RE	I	1	读使能信号：1，允许读出；0，不可读出
A	I	32	读写地址：输入值是按字节为单位；有效范围（以字节为单位）是0x0000-0x2fff，共3072字；需要利用以字为单位的地址进行整字存取
WD	I	32	写入数据
RD	O	32	读出数据

1.2.9.2 功能定义

- 读数据
  - 读数据：RE为1时且时钟上升来临时，根据地址A，读出数据RD;否则读出0。

- 写数据
  - 写数据：WE为1时且时钟上升来临时，根据地址A，写入数据WD；否则保持原值。
  - 会按照教程所要求格式输出写入信息
- 同步复位
  - 当reset为1时，将所有内容清零

1.2.9.3 模块代码

```
`timescale 1ns / 1ps
module DM (
    input         clk,
    input         reset,
    input         WE,
    input         RE,
    input  [31:0] A,
    input  [31:0] pc,
    input  [31:0] WD,
    output [31:0] RD
);
  reg     [31:0] ram   [0:3071];
  // real address
  wire    [11:0] addr;
  integer        i = 0;
  assign addr = A[13:2];
  always @(posedge clk) begin
    if (reset == 1'b1) begin
      for (i = 0; i < 3072; i = i + 1) begin
        ram[i] <= 32'd0;
      end
    end else begin
      if (WE == 1'b1) begin
        ram[addr] <= WD;
        // display for test
        $display("%d@%h: *%h <= %h",$time, pc, A, WD);
      end else begin
        ram[addr] <= ram[addr];
      end
    end
  end
  assign RD = (RE == 1'b1) ? ram[addr] : 32'b0;
endmodule
```

1.2.10 MEM_WB(流水线寄存器)

1.2.10.1 端口定义

端口名	方向	位宽	信号描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
enable	I	1	使能信号
M_nInstr	I	32	M阶段的nInstr值
M_pc	I	32	M阶段的pc值
M_pcPlus4	I	32	M阶段的pc+4值
M_pcPlus8	I	32	M阶段的pc+8值

端口名	方向	位宽	信号描述
M_aluRes	I	32	M阶段的ALU运算结果aluRes
M_rtData	I	32	M阶段的从rt寄存器读出的数据
M_extImm	I	32	M阶段的扩展后的立即数extImm
M_dmData	I	32	M阶段的从DM读出的数据dmData
nInstr_W	O	32	W阶段的nInstr值
pc_W	O	32	W阶段的pc值
pcPlus4_W	O	32	W阶段的pc+4值
pcPlus8_W	O	32	W阶段的pc+8值
aluRes_W	O	32	W阶段的ALU运算结果aluRes
rtData_W	O	32	W阶段的从rt寄存器读出的数据
extImm_W	O	32	W阶段的扩展后的立即数extImm
dmData_W	O	32	W阶段的从DM读出的数据dmData

1.2.10.2 功能定义

- 暂存M阶段结果，向W阶段提供数据。

1.2.10.3 模块代码

```
`timescale 1ns / 1ps
module MEM_WB (
    input             clk,
    input             reset,
    input             enable,
    input      [31:0] M_nInstr,
    input      [31:0] M_pc,
    input      [31:0] M_pcPlus4,
    input      [31:0] M_pcPlus8,
    input      [31:0] M_rtData,
    input      [31:0] M_aluRes,
    input      [31:0] M_extImm,
    input      [31:0] M_dmData,
    output reg [31:0] nInstr_W,
    output reg [31:0] pc_W,
    output reg [31:0] pcPlus4_W,
    output reg [31:0] pcPlus8_W,
    output reg [31:0] rtData_W,
    output reg [31:0] aluRes_W,
    output reg [31:0] extImm_W,
    output reg [31:0] dmData_W
);
  always @(posedge clk) begin
    if (reset) begin
      nInstr_W  <= 0;
      pc_W      <= 0;
      pcPlus4_W <= 0;
      pcPlus8_W <= 0;
      rtData_W  <= 0;
      aluRes_W  <= 0;
      extImm_W  <= 0;
      dmData_W  <= 0;
    end else if (enable) begin
```

```
        nInstr_W  <= M_nInstr;
        pc_W      <= M_pc;
        pcPlus4_W <= M_pcPlus4;
        pcPlus8_W <= M_pcPlus8;
        rtData_W  <= M_rtData;
        aluRes_W  <= M_aluRes;
        extImm_W  <= M_extImm;
        dmData_W  <= M_dmData;
      end
    end
  endmodule
```

1.2.11 Controller

1.2.11.1 端口定义

端口名	方向	位宽	信号描述
op	I	6	操作码
fn	I	6	功能码
regDst	O	2	寄存器堆写入地址选通信号：00，选用rt为地址写入；01，选用rd为地址写入；10，选用$ra为目标写入
aluSrc	O	1	ALU第二个操作数（B）选通信号：1，选用立即数；0，选用寄存器堆读出的数据
aluOp	O	3	ALU操作类型：000，加；001，减；010，按位与；011，按位或；100，判断相等
memToR	O	2	数据写入寄存器堆时的数据来源：00，来自ALU运算结果；01，来自DM；10，来自lui的立即数；11，写入pc+4的值
memWrite	O	1	DM写使能信号：1，允许写入；0，不可写入
memRead	O	1	DM读使能信号：1，允许读出；0，不可读出
regWrite	O	1	寄存器堆写使能信号：1，允许写入；0，不可写入
extOp	O	2	扩展操作类型：00，无符号扩展；01，符号扩展；10，将16位立即数加载至高位且低位补0。仅负责ALU两个操作数的符号扩展
jumpOp	O	3	跳转操作类型：000，无跳转；001，beq；010，jal；011，jr

1.2.11.2 功能定义

- 控制信号生成
  - 识别指令的操作码和功能码，根据指令的不同，生成不同的控制信号

1.2.11.3 控制信号与译码生成对应表

所有指令类型与控制信号以及译码结果的对应关系如下表所示。

> 对于nop，指令码为0x00000000，op[5:0]为000000，fn[5:0]为000000，均为0，故不在表中列出。

信号名-op-fn	regDst	aluSrc	aluOp[2:0]	memToR[1:0]	memWrite	memRead	regWrite	extOp[1:0]	jumpOp[2:0]
add-000000-100000	01	0	000	00	0	0	1	x	000

信号名-op-fn	regDst	aluSrc	aluOp[2:0]	memToR[1:0]	memWrite	memRead	regWrite	extOp[1:0]	jumpOp[2:0]
sub-000000-100010	01	0	001	00	0	0	1	x	000
ori-001101-x	00	1	011	00	0	0	1	00	000
lw-100011-x	00	1	000	01	0	1	1	01	000
sw-101011-x	x	1	000	x	1	0	0	01	000
beq-000100-x	x	0	100	x	0	0	0	x	001
lui-001111-x	00	x	x	10	0	0	1	10	000
jal-000011-x	10	x	x	11	0	0	1	x	010
jr-000000-001000	x	x	x	x	0	0	0	x	011

1.2.11.4 模块代码

```
`timescale 1ns / 1ps
module Controller (
    input      [5:0] op,
    input      [5:0] fn,
    output reg [1:0] regDst,
    output reg       aluSrc,
    output reg [2:0] aluOp,
    output reg [1:0] memToR,
    output reg       memWrite,
    output reg       memRead,
    output reg       regWrite,
    output reg [1:0] extOp,
    output reg [2:0] jumpOp
);
  always @(*) begin
    case (op)
      6'b000000: begin
        //R-type
        regDst   = 2'b01;
        aluSrc   = 0;
        memRead  = 0;
        memWrite = 0;
        extOp    = extOp;
        case (fn)
          6'b100000: begin
```

```
            //add
            aluOp    = 3'b000;
            memToR   = 2'b00;
            regWrite = 1;
            jumpOp   = 3'b000;
          end
          6'b100010: begin
            //sub
            aluOp    = 3'b001;
            memToR   = 2'b00;
            regWrite = 1;
            jumpOp   = 3'b000;
          end
          6'b001000: begin
            //jr
            aluOp    = aluOp;
            memToR   = 2'b00;  //default,not real
            regWrite = 0;
            jumpOp   = 3'b011;
          end
          default: begin
            regDst   = 0;
            aluSrc   = 0;
            aluOp    = 0;
            memToR   = 0;
            memWrite = 0;
            memRead  = 0;
            regWrite = 0;
            extOp    = 2'b00;
            jumpOp   = 0;
          end
        endcase
      end
      6'b100011: begin
        //lw
        regDst   = 2'b00;
        aluSrc   = 1;
        aluOp    = 3'b000;
        memToR   = 2'b01;
        memWrite = 0;
        memRead  = 1;
        regWrite = 1;
        extOp    = 2'b01;
        jumpOp   = 3'b000;
      end
      6'b101011: begin
        //sw
        regDst   = regDst;
        aluSrc   = 1;
        aluOp    = 3'b000;
        memToR   = 2'b00;  //default,not real
        memWrite = 1;
        memRead  = 0;
        regWrite = 0;
        extOp    = 2'b01;
        jumpOp   = 3'b000;
      end
      6'b000100: begin
        //beq
        regDst   = regDst;
        aluSrc   = 0;
        aluOp    = 3'b100;
        memToR   = 2'b00;  //default,not real
        memWrite = 0;
        memRead  = 0;
```

```
          regWrite = 0;
          extOp    = extOp;
          jumpOp   = 3'b001;
        end
        6'b001101: begin
          //ori
          regDst   = 2'b00;
          aluSrc   = 1;
          aluOp    = 3'b011;
          memToR   = 2'b00;
          memWrite = 0;
          memRead  = 0;
          regWrite = 1;
          extOp    = 2'b00;
          jumpOp   = 3'b000;
        end
        6'b001111: begin
          //lui
          regDst   = 2'b00;
          aluSrc   = aluSrc;
          aluOp    = aluOp;
          memToR   = 2'b10;
          memWrite = 0;
          memRead  = 0;
          regWrite = 1;
          extOp    = 2'b10;
          jumpOp   = 3'b000;
        end
        6'b000011: begin
          //jal
          regDst   = 2'b10;
          aluSrc   = aluSrc;
          aluOp    = aluOp;
          memToR   = 2'b11;
          memWrite = 0;
          memRead  = 0;
          regWrite = 1;
          extOp    = extOp;
          jumpOp   = 3'b010;
        end
        default: begin
          regDst   = 0;
          aluSrc   = 0;
          aluOp    = 0;
          memToR   = 0;
          memWrite = 0;
          memRead  = 0;
          regWrite = 0;
          extOp    = 2'b00;
          jumpOp   = 0;
        end
      endcase
    end
  endmodule
```

1.3 冒险单元设计

1.3.1 总体思路

需要实现两个核心模块，**Haz_Decoder（冒险译码器）以及 Haz_Processor （冒险处理器）**。
冒险译码器负责解析各个阶段指令的信息，**本设计采用分布式译码**，传递给冒险处理器来生成暂停和转发逻辑的信号。

> 解析的信息是：某个具体的指令属于以下类别的那一种：
> Cali(lui/ori), Calr(add/sub), Jal(jal), Jr(jr), Load(lw), Store(sw), Beq(beq)

冒险处理器负责生成暂停和转发逻辑的信号，消除冒险现象。

> 根据策略表，确定哪些组合需要stall(S)信号，哪些组合需要对应的哪些转发数据。
> `forward_MUXs`辅助其完成真正的数据转发。

1.3.2 Haz_Decoder 冒险译码器

具体设计如下：

```
`timescale 1ns / 1ps
`include "Define.v"
module Haz_Decoder (
    input  [31:0] nInstr,
    output        isCali,
    output        isCalr,
    output        isJal,
    output        isJr,
    output        isLoad,
    output        isStore,
    output        isBeq
);
  assign isCali  = (nInstr[`op] == `lui || nInstr[`op] == `ori);
  assign isCalr  = (nInstr[`op] == `R && nInstr[`fn] != `jr);
  assign isJal   = (nInstr[`op] == `jal);
  assign isJr    = (nInstr[`op] == `R && nInstr[`fn] == `jr);
  assign isLoad  = (nInstr[`op] == `lw);
  assign isStore = (nInstr[`op] == `sw);
  assign isBeq   = (nInstr[`op] == `beq);
endmodule
```

> Cali是计算型指令。且有操作数是立即数（这里将lui视为计算型指令）；Calr是计算型指令。且操作数均为寄存器值。

1.3.3 Haz_Processor 冒险处理器

另附上冒险单元处理表：

> 说明：
> **对于某一个指令的某一个数据需求，我们定义需求时间**`Tu`: 这条指令位于 D 级的时候，再经过多少个时钟周期就必须要使用相应的数据。beq的`Tu`为0；sw的`rs_Tu`为1，`rt_Tu`为2。
> **对于某个指令的数据产出，我们定义供给时间**`Tn`:位于某个流水级的某个指令，它经过多少个时钟周期可以算出结果并且存储到流水级寄存器里。例如，对于 add 指令，当它处于 E 级，此时结果还没有存储到流水级寄存器里，所以此时它的`Tn`为1。
> **递推公式：下一级的`Tn'` = max{`Tn`-1 ,0}**

暂停策略表:

D阶段可能需要暂停的指令			E阶段当前指令			M阶段当前指令
译码器信息	源寄存器	Tu	Tn			
			calr-rd	cali-rt	load-rt	load-rt
			1	1	2	1
calr	rs/rt	1			S	
cali	rs	1			S	
load	rs	1			S	
store	rs	1			S	
store	rt	2				
beq	rs/rt	0	S	S	S	S
jr	rs	0	S	S	S	S

转发策略表:

源寄存器	译码器信息	对应多路选择器名	对应选择信号名	默认输出	转发源阶段							
					E	M			W			
					jal	calr-rd	cali-rt	jal	calr-rd	cali-rt	load-rt	jal
rs	calr	Drs_fsel	DrsSel	RD1	pcPlus8_E	aluRes_M	aluRes_M	pcPlus8_M	WD	WD	WD	pcPlus8_W
	cali											
	beq											
	load											
	store											
	jr											
rt	calr	Drt_fsel	DrtSel	RD2	pcPlus8_E	aluRes_M	aluRes_M	pcPlus8_M	WD	WD	WD	pcPlus8_W
	store											
	beq											
rs	calr	Ers_fsel	ErsSel	rsData_E		aluRes_M	aluRes_M	pcPlus8_M	WD	WD	WD	pcPlus8_W
	cali											
	load											
	store											
rt	calr	Ert_fsel	ErtSel	rtData_E		aluRes_M	aluRes_M	pcPlus8_M	WD	WD	WD	pcPlus8_W
	store											
rt	store	Mrt_fsel	MrtSel	rtData_M					WD	WD	WD	pcPlus8_W

具体设计如下:

```
`timescale 1ns / 1ps
`include "Define.v"
module Haz_Processor (
    input  [31:0] D_nInstr,
    input  [31:0] E_nInstr,
    input  [31:0] M_nInstr,
    input  [31:0] W_nInstr,
    // stall logic outputs
    output        enableIFU,
    output        enableD,
    output        clearE,
    // forward logic outputs
    output [ 2:0] DrsSel,
    output [ 2:0] DrtSel,
    output [ 2:0] ErsSel,
    output [ 2:0] ErtSel,
    output [ 2:0] MrtSel
);
  // register Address information
```

```
    wire [4:0] rsD, rsE, rtD, rtE, rsM, rtM, rdM, rsW, rtW, rdW, rdE;
    assign rsD = D_nInstr[`rs];
    assign rtD = D_nInstr[`rt];
    assign rsE = E_nInstr[`rs];
    assign rdE = E_nInstr[`rd];
    assign rtE = E_nInstr[`rt];
    assign rsM = M_nInstr[`rs];
    assign rtM = M_nInstr[`rt];
    assign rdM = M_nInstr[`rd];
    assign rsW = W_nInstr[`rs];
    assign rtW = W_nInstr[`rt];
    assign rdW = W_nInstr[`rd];

    // Decoders information
    // D stage
    wire beqD, calrD, caliD, loadD, storeD, jrD, jalD;
    Haz_Decoder D_Decoder (
        .nInstr (D_nInstr),
        .isBeq  (beqD),
        .isCalr (calrD),
        .isCali (caliD),
        .isLoad (loadD),
        .isStore(storeD),
        .isJr   (jrD),
        .isJal  (jalD)
    );
    // E stage
    wire beqE, calrE, caliE, loadE, storeE, jrE, jalE;
    Haz_Decoder E_Decoder (
        .nInstr (E_nInstr),
        .isBeq  (beqE),
        .isCalr (calrE),
        .isCali (caliE),
        .isLoad (loadE),
        .isStore(storeE),
        .isJr   (jrE),
        .isJal  (jalE)
    );
    // M stage
    wire beqM, calrM, caliM, loadM, storeM, jrM, jalM;
    Haz_Decoder M_Decoder (
        .nInstr (M_nInstr),
        .isBeq  (beqM),
        .isCalr (calrM),
        .isCali (caliM),
        .isLoad (loadM),
        .isStore(storeM),
        .isJr   (jrM),
        .isJal  (jalM)
    );
    // W stage
    wire beqW, calrW, caliW, loadW, storeW, jrW, jalW;
    Haz_Decoder W_Decoder (
        .nInstr (W_nInstr),
        .isBeq  (beqW),
        .isCalr (calrW),
        .isCali (caliW),
        .isLoad (loadW),
        .isStore(storeW),
        .isJr   (jrW),
        .isJal  (jalW)
    );

    // stall logic
    wire stall_rs;
```

```
  wire stall_rt;
  wire stall;
  assign stall_rt = (beqD && loadE && rtE == rtD && rtD) || (beqD && caliE && rtE == rtD &&
rtD) || (beqD && calrE && rdE == rtD && rtD) || (beqD && loadM && rtM == rtD && rtD) ||
(calrD && loadE && rtD == rtE && rtE);
  assign stall_rs = (beqD && loadE && rtE == rsD && rsD) ||
                        (beqD && caliE && rtE == rsD && rsD) ||
                        (beqD && calrE && rdE == rsD && rsD) ||
                        (beqD && loadM && rtM == rsD && rsD) ||
                        (calrD && loadE && rsD == rtE && rsD) ||
                        (caliD && loadE && rsD == rtE && rsD) ||
                        (loadD && loadE && rsD == rtE && rsD) ||
                        (storeD && loadE && rsD == rtE && rsD) ||
                        (jrD && loadE && rtE == rsD && rsD) ||
                        (jrD && caliE && rtE == rsD && rsD) ||
                        (jrD && calrE && rdE == rsD && rsD) ||
                        (jrD && loadM && rtM == rsD && rsD);
  assign stall = stall_rs || stall_rt;
  assign enableIFU = ~stall;
  assign enableD = ~stall;
  assign clearE = stall;

  // forward logic  default:0
  assign DrsSel = (rsD == 5'd31 && jalE && rsD) ? 1 : (rsD == rtM && caliM && rsD) || (rsD
== rdM && calrM && rsD) ? 2 : (rsD == 5'd31 && jalM && rsD) ? 3 : (rsD == rtW && (loadW ||
caliW) && rsD) || (rsD == rdW && calrW && rsD) ? 4 : (rsD == 31 && jalW && rsD) ? 5 : 0;
  assign DrtSel = (rtD == 31 && jalE && rtD) ? 1 : (rtD == rtM && caliM && rtD) || (rtD ==
rdM && calrM && rtD) ? 2 : (rtD == 31 && jalM && rtD) ? 3 : (rtD == rtW && (loadW || caliW)
&& rtD) || (rtD == rdW && calrW && rtD) ? 4 : (rtD == 31 && jalW && rtD) ? 5 : 0;
  assign ErsSel = (rsE == rtM && caliM && rsE) || (rsE == rdM && calrM && rsE) ? 1 : (rsE ==
31 && jalM && rsE) ? 2 : (rsE == rtW && (loadW || caliW) && rsE) || (rsE == rdW && calrW &&
rsE) ? 3 : (rsE == 31 && jalW && rsE) ? 4 : 0;
  assign ErtSel = (rtE == rtM && caliM && rtE) || (rtE == rdM && calrM && rtE) ? 1 : (rtE ==
31 && jalM && rtE) ? 2 : (rtE == rtW && (loadW || caliW) && rtE) || (rtE == rdW && calrW &&
rtE) ? 3 : (rtE == 31 && jalW && rtE) ? 4 : 0;
  assign MrtSel = (rtM == rtW && (loadW || caliW) && rtM) || (rtM == rdW && calrW && rtM) ?
1 : (rtM == 31 && jalW && rtM) ? 2 : 0;

endmodule
```

Part 2 测试方案

基本方法:

- 构造一段MIPS程序，输入Mars中运行；
- 利用Mars导出机器码，形成code.txt文件，用来载入自己的cpu的IM中，仿真运行；
- 比对两者按统一格式输出的内容。即如下两种格式：

```
//grf
$display("@%h: $%d <= %h", WPC, Waddr, WData);
//dm
$display("@%h: *%h <= %h", pc, addr, din);
```

> 注意:
> 需要用到魔改版的Mars来获取标准的输出
> 这里统一规定：对于0号寄存器的写入不进行输出

以下谈谈测试数据应该如何生成。

2.1 针对单指令功能的测试

2.1.1 构造策略

这部分实现比较简单，可以用高级语言写一段程序，通过一些循环语句连续构造大量单种指令的序列。主要测试除了**beq/jal/jr的其余六种指令**的功能。当然，在最后也可以简单手动构造测试剩下的三种指令。

2.1.2 示例代码

```
sub $27, $27, $15
sub $28, $28, $15
sub $29, $29, $15
sub $30, $30, $15
sub $31, $31, $15
ori $5, $0, 4
sw $0, 0($5)
sw $1, 4($5)
sw $2, 8($5)
sw $3, 12($5)
sw $4, 16($5)
# there can be more like this...
```

2.2 针对暂停和转发策略的测试

2.2.1 构造策略

根据自己的策略表，明确总共有哪些组合会出现冒险的情况。列出所有的这些情况，逐个构造相应的测试程序，如下所示（下面没有完全列举）。

2.2.2 示例代码

```
#beq cali
ori $1 $0 12
beq $1 $0 next2
ori $4 $0 1234
next2:addu $1 $1 $1

#beq load
ori $1 $0 12
sw $1 0($0)
lw $2 0($0)
beq $2 $0 next3
ori $4 $0 1234
next3:addu $1 $1 $1

#cal_r load
ori $1 $0 12
sw $1 0($0)
lw $2 0($0)
subu $3 $2 $1
```

2.3 综合测试

利用**1000行左右的测试程序**全面地测试所有的指令。在P4文档中，已经提供了一份详尽的代码，类似于P4，我们可以再次构造一份代码。

```
ori $0, $0, 156
ori $1, $0, 100
ori $2, $0, 88
```

```
ori $3, $0, 64
ori $4, $0, 124
ori $5, $0, 72
ori $6, $0, 12
ori $7, $0, 64
ori $8, $0, 148
ori $9, $0, 16
ori $10, $0, 100
ori $11, $0, 32
ori $12, $0, 56
ori $13, $0, 8
ori $14, $0, 152
ori $15, $0, 44
ori $16, $0, 136
ori $17, $0, 72
ori $18, $0, 72
ori $19, $0, 60
ori $20, $0, 48
ori $21, $0, 152
ori $22, $0, 136
ori $23, $0, 64
ori $24, $0, 4
ori $25, $0, 92
ori $26, $0, 28
ori $27, $0, 48
ori $28, $0, 136
ori $29, $0, 28
ori $30, $0, 88
ori $31, $0, 56

lui $4,20
sw $2,144($0)
jal Test_jal1
ori $2,$31,52
beq $0,$0,Test_beq1
add $0,$31,$3
Test_jal1: jr $ra
Test_beq1:
ori $4,$0,72
jal Test1
sw $1,68($0)
Back1:
ori $4,$1,104
ori $3,$1,96
jal Test_jal2
lw $0,-12288($31)
beq $0,$0,Test_beq2
sw $4,-12288($31)
Test_jal2: jr $ra
Test_beq2:
add $4,$3,$2
jal Test2
lui $2,124
Back2:
add $0,$4,$2
add $0,$1,$3
jal Test_jal3
add $2,$31,$0
beq $0,$0,Test_beq3
lw $0,-12288($31)
Test_jal3: jr $ra
Test_beq3:
sub $3,$3,$3
sw $1,140($3)
jal Test3
```

```
lw $4,116($0)
Back3:
sub $1,$2,$3
lui $3,128
jal Test_jal4
sw $4,-12288($31)
beq $0,$0,Test_beq4
sw $4,-12288($31)
Test_jal4: jr $ra
Test_beq4:
ori $4,$0,56
jal Test4
sw $2,4($0)
Back4:
add $3,$4,$3
ori $0,$3,96
jal Test_jal5
lw $2,-12288($31)
beq $0,$0,Test_beq5
add $2,$31,$1
Test_jal5: jr $ra
Test_beq5:
ori $4,$4,128
jal Test5
lw $3,88($0)
Back5:
sub $3,$3,$3
ori $3,$3,12
sw $3,0($0)
lw $3,0($0)
lw $3,72($3)
sw $3,56($0)
sub $2,$2,$2
ori $2,$2,52
sw $2,-20($2)
jal Test_jal6
lw $3,-12288($31)
beq $0,$0,Test_beq6
lw $1,-12288($31)
Test_jal6: jr $ra
Test_beq6:
sub $4,$4,$4
ori $4,$4,12
sw $4,0($0)
lw $4,0($0)
lw $4,16($4)
sw $4,92($0)
jal Test6
ori $4,$1,88
Back6:
sub $0,$0,$0
sw $2,68($0)
sw $2,72($0)
lui $0,12
jal Test7
sub $4,$2,$3
Back7:
ori $4,$3,16
sub $4,$4,$4
sw $4,12($4)
add $0,$2,$1
jal Test8
add $0,$0,$1
Back8:
lui $0,84
```

```
add $1,$0,$3
jal Test_jal9
ori $3,$31,16
beq $0,$0,Test_beq9
lw $2,-12288($31)
Test_jal9: jr $ra
Test_beq9:
lui $4,56
jal Test9
sw $0,56($0)
Back9:
sub $3,$3,$3
sw $0,88($3)
sub $4,$0,$1
lw $2,112($0)
jal Test10
sw $4,100($0)
Back10:
sw $2,32($0)
ori $3,$2,140
lw $2,156($0)
jal Test11
lui $3,88
Back11:
sub $4,$4,$4
lw $1,24($4)
ori $1,$3,0
sw $4,0($0)
jal Test12
lui $1,56
Back12:
add $3,$1,$0
sub $1,$1,$1
ori $1,$1,76
lw $2,-12($1)
ori $0,$1,24
jal Test13
sw $2,108($0)
Back13:
lw $1,52($0)
ori $4,$3,52
jal Test_jal14
sw $4,-12288($31)
beq $0,$0,Test_beq14
ori $1,$31,88
Test_jal14: jr $ra
Test_beq14:
add $0,$0,$1
jal Test14
sub $2,$4,$2
Back14:
sub $4,$4,$4
ori $4,$4,100
sw $4,0($0)
ori $4,$4,100
lw $0,0($0)
sw $4,80($0)
ori $0,$4,44
jal Test_jal15
sw $2,-12288($31)
beq $0,$0,Test_beq15
sw $4,-12288($31)
Test_jal15: jr $ra
Test_beq15:
ori $0,$2,48
```

```
jal Test15
add $0,$1,$4
jal End
ori $1,$4,136

Test1:
ori $1,$31,56
sub $16,$16,$16
sub $24,$24,$24
ori $24,$24,10
sub $12,$12,$12
ori $12,$12,1
SubTest1:
add $16,$16,$12
beq $16,$24,EndTest1
lui $0,92
sub $0,$2,$4
sub $3,$4,$1
lui $2,140
lw $2,48($0)
ori $4,$1,76
lui $4,12
sub $4,$4,$2
beq $1,$0,SubTest1
add $2,$0,$2
sub $2,$2,$2
ori $2,$2,44
sw $2,-28($2)
sub $3,$3,$3
ori $3,$3,12
sw $3,0($0)
lw $1,0($0)
lw $3,140($1)
sw $3,148($0)
sub $3,$3,$3
sw $2,56($3)
sub $2,$2,$2
sw $2,32($2)
lw $0,68($0)
sw $1,44($0)
sub $1,$1,$1
ori $1,$1,12
sw $1,0($0)
lw $3,0($0)
lw $1,32($3)
sw $1,76($0)
sub $4,$2,$4
lui $0,16
sub $2,$4,$4
beq $2,$2,SubTest1
sub $4,$0,$2
ori $4,$2,76
ori $0,$4,68
ori $0,$1,16
sub $2,$2,$2
ori $2,$2,120
lw $2,-52($2)
beq $1,$0,SubTest1
lui $2,52
lw $0,156($0)
EndTest1:
sub $3,$3,$3
add $2,$31,$3
add $31,$31,$31
sub $31,$31,$2
```

```
sw $0,32($0)
jr $ra
add $2,$0,$1


Test2:
beq $0,$31,EndTest2
sub $2,$2,$2
ori $2,$2,152
sw $2,-132($2)
sub $2,$0,$4
lui $0,148
lw $3,68($0)
lui $4,40
sw $2,140($0)
sw $0,120($0)
lui $4,52
sw $3,88($0)
sw $3,4($0)
add $0,$1,$4
sub $1,$4,$4
sw $3,120($0)
lui $4,124
add $1,$4,$3
sub $3,$2,$1
ori $3,$3,1
lui $3,0
lw $2,32($3)
sub $3,$0,$0
lui $3,1
lw $2,-65492($3)
lui $2,108
beq $0,$3,EndTest2
lw $4,76($0)
beq $4,$1,EndTest2
sub $3,$3,$2
sub $4,$4,$4
sw $3,148($4)
sub $4,$0,$1
lui $1,124
ori $4,$1,84
EndTest2:
sub $4,$4,$4
add $4,$31,$4
sub $31,$31,$31
add $31,$0,$4
jr $ra
ori $4,$3,52


Test3:
ori $0,$31,0
sw $2,12($0)
sw $3,128($0)
sub $2,$3,$3
ori $3,$1,116
sub $0,$0,$0
lw $2,40($0)
lw $0,52($0)
sub $2,$2,$2
sw $2,136($2)
sub $0,$0,$0
ori $0,$0,108
sw $0,0($0)
ori $0,$0,88
```

```
lw $2,0($0)
sw $0,148($2)
sub $0,$0,$0
sw $1,116($0)
sub $1,$1,$1
lw $3,136($1)
sub $4,$0,$1
sub $3,$3,$3
ori $3,$3,12
sw $3,0($0)
lw $3,0($0)
lw $3,0($3)
sw $3,44($0)
beq $4,$3,EndTest3
lui $1,80
sub $2,$2,$2
ori $2,$2,12
sw $2,0($0)
lw $2,0($0)
lw $2,76($2)
sw $2,120($0)
beq $4,$3,EndTest3
add $0,$1,$2
add $0,$1,$2
ori $2,$2,1
lui $2,0
lw $3,100($2)
sub $4,$4,$4
sw $3,132($4)
lui $2,156
sub $4,$4,$4
sw $1,104($4)
lui $3,104
lw $0,88($0)
ori $3,$2,152
beq $1,$1,EndTest3
lui $3,64
lui $0,68
EndTest3:
sub $2,$2,$2
sw $31,96($2)
sub $31,$31,$31
lw $31,96($2)
lui $4,76
jr $ra
ori $0,$2,36


Test4:
add $4,$31,$4
sub $16,$16,$16
sub $21,$21,$21
ori $21,$21,10
sub $20,$20,$20
ori $20,$20,1
SubTest4:
add $16,$16,$20
beq $16,$21,EndTest4
sw $0,0($0)
sub $0,$0,$0
sw $0,52($0)
lui $3,64
lui $0,112
lw $3,40($0)
lui $2,48
```

```
sw $3,104($0)
sub $4,$4,$4
sw $3,92($4)
lw $1,156($0)
sw $1,104($0)
sub $2,$0,$4
beq $1,$3,SubTest4
lw $0,16($0)
sub $4,$3,$4
lui $2,8
lw $0,28($0)
add $0,$3,$2
lw $4,116($0)
beq $4,$3,SubTest4
lw $4,36($0)
sub $1,$1,$1
beq $3,$2,SubTest4
sub $2,$4,$4
sub $2,$2,$2
lw $1,96($2)
lw $3,72($0)
lui $4,120
lui $2,60
beq $0,$1,SubTest4
sub $2,$2,$4
sub $2,$2,$2
ori $2,$2,148
sw $2,0($0)
ori $2,$2,12
lw $2,0($0)
sw $2,132($2)
EndTest4:
sub $3,$3,$3
add $3,$31,$3
sub $31,$31,$31
add $31,$0,$3
jr $ra
add $1,$2,$2


Test5:
ori $0,$31,4
sub $1,$0,$2
lw $1,100($0)
sub $0,$0,$0
ori $0,$0,24
sw $0,0($0)
ori $0,$0,52
lw $3,0($0)
sw $0,56($3)
sub $4,$1,$4
add $1,$2,$4
sub $0,$0,$0
sw $0,4($0)
lui $4,36
add $1,$3,$1
lw $1,44($0)
beq $0,$1,EndTest5
sub $3,$1,$3
sub $0,$0,$0
ori $0,$0,32
sw $0,0($0)
ori $0,$0,144
lw $2,0($0)
sw $0,64($2)
```

```
sub $4,$3,$3
sub $1,$4,$3
add $4,$2,$0
sub $4,$4,$4
ori $4,$4,128
sw $2,-104($4)
sub $4,$0,$0
beq $0,$2,EndTest5
lui $4,136
sub $4,$4,$4
ori $4,$4,64
sw $4,0($0)
ori $4,$4,108
lw $1,0($0)
sw $4,56($1)
lui $2,32
ori $0,$2,80
add $4,$1,$2
add $3,$1,$1
lw $0,92($0)
add $4,$2,$1
beq $3,$4,EndTest5
ori $2,$2,60
EndTest5:
sub $3,$3,$3
add $3,$31,$3
sub $31,$31,$31
add $31,$0,$3
jr $ra
add $3,$1,$2


Test6:
ori $4,$31,128
sub $18,$18,$18
sub $22,$22,$22
ori $22,$22,10
sub $12,$12,$12
ori $12,$12,1
SubTest6:
add $18,$18,$12
beq $18,$22,EndTest6
lw $0,116($0)
sub $0,$1,$2
lui $0,28
add $1,$0,$4
add $0,$2,$2
ori $1,$3,124
sw $0,152($0)
sub $3,$0,$2
lw $4,44($0)
sw $4,84($0)
add $2,$3,$0
sub $4,$4,$4
ori $4,$4,12
sw $4,0($0)
lw $2,0($0)
lw $4,104($2)
sw $4,156($0)
sub $2,$2,$2
lw $0,68($2)
lw $2,144($0)
lw $4,72($0)
sw $0,60($0)
ori $3,$3,16
```

```
add $2,$3,$2
sub $1,$0,$0
sub $1,$1,$1
ori $1,$1,76
sw $0,-52($1)
add $2,$2,$0
add $2,$3,$2
ori $4,$4,136
beq $3,$4,SubTest6
lui $3,108
add $1,$1,$1
lw $2,124($0)
EndTest6:
sub $3,$3,$3
add $2,$31,$3
sub $31,$31,$31
add $31,$0,$2
jr $ra
sub $3,$1,$1


Test7:
add $3,$31,$2
ori $0,$0,148
sub $0,$0,$0
lw $4,24($0)
sub $2,$1,$2
lw $3,100($0)
sub $3,$2,$0
lui $3,1
lw $3,-65508($3)
ori $0,$3,16
sub $0,$0,$0
ori $0,$0,140
sw $0,0($0)
ori $0,$0,136
lw $3,0($0)
sw $0,12($3)
ori $2,$3,16
sw $1,60($0)
lui $4,60
lw $3,104($0)
sw $0,36($0)
ori $4,$1,68
sw $2,48($0)
lw $2,108($0)
lw $1,144($0)
sub $2,$1,$4
lui $3,64
sw $3,48($0)
sub $3,$3,$3
lw $2,20($3)
lui $3,68
lui $2,52
add $1,$0,$1
sub $4,$3,$0
sub $0,$0,$0
sw $4,16($0)
EndTest7:
sub $4,$4,$4
add $3,$31,$4
add $31,$31,$31
sub $31,$31,$3
ori $3,$4,152
jr $ra
```

```
ori $0,$4,116


Test8:
beq $3,$31,EndTest8
sub $21,$21,$21
sub $10,$10,$10
ori $10,$10,10
sub $6,$6,$6
ori $6,$6,1
SubTest8:
add $21,$21,$6
beq $21,$10,EndTest8
ori $2,$2,132
sw $2,12($0)
sub $3,$3,$3
ori $3,$3,36
sw $1,-16($3)
sub $1,$1,$1
ori $1,$1,12
sw $1,0($0)
lw $2,0($0)
lw $1,20($2)
sw $1,60($0)
sub $1,$1,$1
sw $0,144($1)
add $4,$2,$4
ori $0,$0,140
lui $4,44
lw $1,112($0)
ori $3,$4,1
lui $3,0
lw $3,64($3)
sub $3,$3,$3
ori $3,$3,24
sw $4,-4($3)
sw $3,68($0)
sub $2,$2,$2
lw $3,88($2)
sw $3,68($0)
sw $0,36($0)
ori $1,$3,156
lw $0,136($0)
ori $0,$3,100
ori $3,$0,44
sw $2,100($0)
lui $1,124
sub $3,$2,$0
add $4,$2,$2
add $3,$4,$1
lui $2,116
sw $1,144($0)
EndTest8:
sub $3,$3,$3
add $3,$31,$3
sub $31,$31,$31
add $31,$0,$3
jr $ra
ori $2,$2,56


Test9:
add $3,$31,$3
add $1,$4,$0
lw $3,52($0)
```

```
lw $2,128($0)
lui $0,128
sub $1,$1,$1
ori $1,$1,12
sw $1,0($0)
lw $2,0($0)
lw $1,152($2)
sw $1,32($0)
ori $2,$0,48
add $3,$3,$3
sub $0,$0,$0
sw $4,28($0)
sub $3,$0,$4
sub $0,$3,$4
lui $3,104
beq $4,$0,EndTest9
ori $1,$2,44
ori $2,$3,76
sub $3,$4,$3
lui $3,1
lw $2,-65432($3)
sub $0,$0,$3
lui $1,148
ori $0,$4,32
sub $4,$4,$2
ori $2,$1,80
add $3,$1,$1
ori $0,$0,120
ori $3,$2,112
ori $2,$4,100
beq $3,$4,EndTest9
lui $0,40
lw $0,96($0)
EndTest9:
sub $4,$4,$4
add $3,$31,$4
sub $31,$31,$31
add $31,$0,$3
lw $0,108($0)
jr $ra
lw $2,0($0)


Test10:
add $1,$31,$3
sub $0,$0,$0
ori $3,$4,1
lui $3,0
lw $3,116($3)
sub $1,$1,$1
sw $0,116($1)
ori $0,$4,140
lui $4,88
sub $1,$1,$1
sw $2,12($1)
sub $2,$2,$2
ori $2,$2,56
sw $2,0($0)
ori $2,$2,56
lw $0,0($0)
sw $2,64($0)
lw $1,128($0)
sub $1,$1,$1
sw $0,56($1)
sub $1,$1,$1
```

```
sw $3,64($1)
sub $4,$4,$4
ori $4,$4,152
sw $1,-68($4)
sub $3,$3,$3
lw $0,104($3)
sw $4,80($0)
sw $2,48($0)
ori $4,$1,28
sub $0,$4,$4
sub $1,$1,$1
ori $1,$1,136
sw $4,-92($1)
add $2,$2,$1
add $4,$0,$2
sub $1,$1,$1
ori $1,$1,12
sw $1,0($0)
lw $2,0($0)
lw $1,112($2)
sw $1,40($0)
sub $0,$1,$2
lui $2,36
ori $2,$4,16
add $0,$4,$1
add $4,$1,$1
EndTest10:
sub $3,$3,$3
add $2,$31,$3
sub $31,$31,$31
add $31,$0,$2
jr $ra
ori $3,$2,48


Test11:
add $3,$31,$0
sub $27,$27,$27
sub $15,$15,$15
ori $15,$15,10
sub $16,$16,$16
ori $16,$16,1
SubTest11:
add $27,$27,$16
beq $27,$15,EndTest11
lw $0,0($0)
lw $1,112($0)
ori $4,$2,16
lui $1,136
add $3,$0,$2
sub $3,$3,$3
lw $4,52($3)
lw $1,144($0)
sw $2,92($0)
sub $0,$0,$0
sw $3,12($0)
sub $1,$1,$1
ori $1,$1,104
sw $0,-52($1)
ori $3,$3,108
lw $1,104($0)
beq $1,$3,SubTest11
ori $3,$2,52
beq $4,$0,SubTest11
sub $3,$4,$2
```

```
ori $4,$3,1
lui $4,0
lw $2,64($4)
lw $2,140($0)
beq $3,$1,SubTest11
sw $4,60($0)
sw $3,152($0)
sub $2,$2,$2
sw $2,92($2)
add $0,$1,$2
add $4,$2,$3
sw $1,136($0)
lw $4,28($0)
lui $0,104
ori $0,$0,152
ori $2,$0,44
EndTest11:
sub $4,$4,$4
add $2,$31,$4
sub $31,$31,$31
ori $31,$2,0
lw $3,116($0)
jr $ra
lw $3,92($0)


Test12:
lw $0,-12288($31)
sub $21,$21,$21
sub $20,$20,$20
ori $20,$20,10
sub $12,$12,$12
ori $12,$12,1
SubTest12:
add $21,$21,$12
beq $21,$20,EndTest12
ori $0,$0,40
add $3,$0,$0
sub $0,$4,$0
sub $0,$0,$0
lw $1,0($0)
lui $0,116
sub $3,$3,$3
sw $3,48($3)
ori $2,$1,0
lw $3,80($0)
ori $4,$3,16
sub $4,$4,$4
ori $4,$4,52
lw $1,-20($4)
add $3,$0,$1
sub $1,$3,$0
beq $4,$1,SubTest12
add $1,$3,$2
sub $2,$2,$2
ori $2,$2,88
sw $2,0($2)
lui $1,20
lw $3,12($0)
sw $3,52($0)
sw $3,76($0)
sw $4,124($0)
beq $2,$3,SubTest12
lw $4,124($0)
sub $3,$3,$4
```

```
add $4,$1,$1
ori $0,$3,136
beq $3,$2,SubTest12
lw $1,80($0)
sub $3,$3,$3
lw $3,128($3)
sw $2,148($0)
EndTest12:
sub $2,$2,$2
sw $31,4($2)
sub $31,$31,$31
lw $31,4($2)
jr $ra
lw $1,24($0)


Test13:
lw $3,-12288($31)
ori $2,$1,1
lui $2,0
lw $3,124($2)
sub $0,$0,$0
sw $4,44($0)
lui $2,60
sub $3,$3,$3
sub $1,$1,$1
sw $3,152($1)
sub $2,$2,$2
ori $2,$2,132
sw $4,-60($2)
lui $2,156
add $2,$2,$4
lw $3,32($0)
sub $1,$1,$1
ori $1,$1,12
sw $1,0($0)
lw $2,0($0)
lw $1,140($2)
sw $1,108($0)
beq $1,$4,EndTest13
lw $4,136($0)
sub $1,$1,$1
ori $1,$1,12
sw $1,0($0)
lw $1,0($0)
lw $1,16($1)
sw $1,156($0)
sub $2,$3,$3
lui $2,1
lw $4,-65404($2)
lui $3,120
add $1,$2,$4
sub $3,$3,$3
ori $3,$3,132
sw $3,0($0)
ori $3,$3,40
lw $2,0($0)
sw $3,32($2)
sub $2,$0,$3
lui $0,120
lw $4,12($0)
sub $3,$3,$3
lw $2,56($3)
sub $0,$1,$1
add $0,$2,$2
```

```
lw $2,120($0)
beq $1,$2,EndTest13
sw $1,148($0)
ori $0,$1,44
EndTest13:
sub $2,$2,$2
sw $31,12($2)
sub $31,$31,$31
lw $31,12($2)
jr $ra
sub $3,$2,$2


Test14:
beq $0,$31,EndTest14
sub $11,$11,$11
sub $17,$17,$17
ori $17,$17,10
sub $23,$23,$23
ori $23,$23,1
SubTest14:
add $11,$11,$23
beq $11,$17,EndTest14
ori $3,$3,96
sub $2,$2,$2
sw $2,88($2)
lui $2,156
add $3,$1,$4
add $2,$2,$3
lui $1,96
lw $0,60($0)
add $4,$2,$3
ori $4,$0,28
sub $1,$1,$1
ori $1,$1,92
sw $4,-52($1)
sub $4,$4,$4
sw $4,84($4)
lui $0,28
ori $0,$3,92
sw $0,132($0)
sub $3,$3,$3
ori $3,$3,148
sw $2,-44($3)
ori $0,$3,104
sub $4,$4,$4
lw $0,124($4)
add $1,$3,$4
lw $0,84($0)
sub $0,$4,$1
sub $4,$1,$3
sub $2,$4,$3
sub $1,$3,$0
ori $3,$2,140
sub $1,$1,$1
ori $1,$1,148
lw $2,-36($1)
add $3,$3,$4
EndTest14:
sub $2,$2,$2
sw $31,28($2)
sub $31,$31,$31
lw $31,28($2)
lw $4,120($0)
jr $ra
```

```
ori $1,$3,152


Test15:
add $4,$31,$0
sub $12,$12,$12
sub $13,$13,$13
ori $13,$13,10
sub $28,$28,$28
ori $28,$28,1
SubTest15:
add $12,$12,$28
beq $12,$13,EndTest15
ori $3,$1,128
sub $0,$0,$0
ori $0,$0,12
sw $0,0($0)
lw $0,0($0)
lw $0,132($0)
sw $0,88($0)
ori $0,$0,12
sub $0,$0,$0
sw $4,20($0)
lw $4,80($0)
ori $4,$0,1
lui $4,0
lw $2,4($4)
ori $4,$1,32
lui $3,152
sub $2,$2,$2
lw $1,24($2)
sw $0,24($0)
sub $4,$4,$4
ori $4,$4,36
sw $4,0($0)
ori $4,$4,112
lw $1,0($0)
sw $4,136($1)
ori $3,$1,56
sub $2,$2,$2
ori $2,$2,12
sw $2,0($0)
lw $0,0($0)
lw $2,36($0)
sw $2,148($0)
ori $3,$1,156
sub $2,$0,$3
add $2,$0,$3
lw $3,64($0)
ori $3,$3,64
sub $4,$1,$0
sub $4,$4,$4
ori $4,$4,76
sw $4,-32($4)
beq $2,$0,SubTest15
sub $4,$4,$0
lw $1,88($0)
ori $1,$1,136
sub $3,$0,$3
sub $3,$3,$3
ori $3,$3,68
sw $2,-64($3)
ori $3,$4,1
lui $3,0
lw $3,132($3)
```

```
EndTest15:
sub $2,$2,$2
sw $31,84($2)
sub $31,$31,$31
lw $31,84($2)
ori $3,$3,36
jr $ra
lw $1,40($0)
End:
```

2.3 结论

经对比内容一致，本测试通过，说明CPU的基本功能实现正确。

Part 3 思考题解答

- **我们使用提前分支判断的方法尽早产生结果来减少因不确定而带来的开销，但实际上这种方法并非总能提高效率，请从流水线冒险的角度思考其原因并给出一个指令序列的例子。**

  分支判断提前之后我们会发现：beq指令的Tuse为0。当Tuse的值小于Tnew时，我们不得不采用阻塞策略。而Tuse为0显然会加剧这一情况的发生。这或许会导致原本只需要进行转发的指令因为这一提前而不能及时生成数据导致需要阻塞，例如：

  ```
  sub $t0,$t1,$t2
  beq $t0,$t1,label
  ```

  在这个例子中，原本转发可以解决的更新需要阻塞一个周期。若Beq在EX阶段ALU中判断，则此时EX_MEM流水线寄存器中已经存有了$t0正确的值，可以转发，不需要阻塞。

- **因为延迟槽的存在，对于 jal 等需要将指令地址写入寄存器的指令，要写回 PC + 8，请思考为什么这样设计?**

  因为采用延迟槽的编译优化方法，在D级决定是否进行跳转，这个时候下一条指令已经进入了流水线，所以这条指令仍被执行，下下条指令地址存入PC（需要+8）。这样达到充分利用流水线的性能。

- **我们要求大家所有转发数据都来源于流水寄存器而不能是功能部件（如 DM 、 ALU ），请思考为什么？。**

  每一级流水线寄存器存储对应流水级的指令以及相关数据值起到隔断作用，让每个流水级 中的数据不会受到上一个流水级的数据通路的影响。

- **我们为什么要使用 GPR 内部转发？该如何实现?**

  为了防止W级还未写入GRF的数据在之后的指令中使用导致运行结果出现错误。 实现如下：

  ```
  assign RD1 = (WE == 1 && A3 != 5'd0 && A3 == A1) ? WD : regfile[A1];
  assign RD2 = (WE == 1 && A3 != 5'd0 && A3 == A2) ? WD : regfile[A2];
  ```

- **我们转发时数据的需求者和供给者可能来源于哪些位置？共有哪些转发数据通路?**

  请参见上面的冒险策略表。

- **在课上测试时，我们需要你现场实现新的指令，对于这些新的指令，你可能需要在原有的数据通路上做哪些扩展或修改？提示：你可以对指令进行分类，思考每一类指令可能修改或扩展哪些位置。**

  跳转：可能需要在Controller加入新的控制信号识别该指令,并在IFU中加入完成该指令的方法，并并考虑跳转和阻塞问题。
  计算：可能需要在Controller加入新的ALUop信号计算该指令，并在ALU中加入完成该指令的方法，并考虑跳转和阻塞问题。
  访存：可能需要在Controller加入新的控制信号识别该指令，并在DM中完成该访存的方法，并考虑跳转和阻塞问题。

- **确定你的译码方式，简要描述你的译码器架构，并思考该架构的优势以及不足。**

我选用的是**分布式译码**方式。 译码风格为**指令驱动型**，具体见上Controller模块代码。
优势：不需要流水控制信号，降低整个通路的复杂度。实现起来无论是思路还是代码量上都远远少于集中式译码器。
不足：多次实例化Controller模块，造成了资源的浪费，降低了效率。

- **[P5 选做] 请详细描述你的测试方案及测试数据构造策略**

  可以大致参见上面的**测试方案**模块。