武汉大学计算机学院
本科生课程设计报告

RISC_V CPU 设计

专 业 名 称 ：计算机弘毅

课 程 名 称 ：计算机系统综合设计

指 导 教 师 ：蔡朝晖

学 生 学 号 ：2019300003043

学 生 姓 名 ：郭志浩

二〇二一年七月

郑 重 声 明

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名：郭志浩　　日期：2021.7.9

目录

1 实验目的和意义

1.1 实验目的

本实验通过对 RISC-V CPU 设计并通过 FPGA 开发板仿真，了解计算机的组成和设计，了解完整的 SoC 设计流程，实现一个五级流水线的 RISC-V 架构 CPU.

2 实验环境介绍

2.1 Verilog HDL

Verilog 是一种用于描述、设计电子系统（特别是数字电路）的硬件描述语言，主要用于在集成电路设计，特别是超大规模集成电路的计算机辅助设计。Verilog 是电气电子工程师学会（IEEE）的 1364 号标准。

Verilog 能够在多种抽象级别对数字逻辑系统进行描述：既可以在晶体管级、逻辑门级进行描述，也可以在寄存器传输级对电路信号在寄存器之间的传输情况进行描述。除了对电路的逻辑功能进行描述，Verilog 代码还能够被用于逻辑仿真、逻辑综合，其中后者可以把寄存器传输级的 Verilog 代码转换为逻辑门级的网表，从而方便在现场可编程逻辑门阵列上实现硬件电路，或者让硬件厂商制造具体的专用集成电路。

2.2 ModelSim

ModelSim 是 Mentor 公司提供的多语言 HDL 仿真环境，用于仿真诸如 VHDL。Verilog 和 SystemC 之类的硬件描述语言，并包含一个内置的 C 调试器。ModelSim 可以单独使用，也可以与 Intel Quartus Prime、Xilinx ISE 或 Xilinx Vivado 等软件结合使用。仿真的执行可以使用图形用户界面（GUI）或自动脚本，是 FPGA/ASIC 设计的首选仿真软件。

2.3 Xilinx ISE

Xilinx ISE (Xilinx Integrated Synthesis Environment，Xilinx 集成综合环境) 是一款由 Xilinx 开发的用于合成和分析 HDL 设计的软件工具。开发者可以使用 Xilinx ISE 综合（“编译”）自己的设计、执行时序分析、检查 RTL 图、仿真不同激励下的设计的响应，并使用编程器配置目标设备。

Xilinx ISE 是用于 Xilinx 的 FPGA 产品的设计环境，并与这些芯片的架构紧密联系，而不能用于其他厂家的 FPGA 产品。Xilinx ISE 主要用于电路综

合和设计,而 ISIM 或 ModelSim 等逻辑模拟器则用于系统级测试。Xilinx ISE 附带的其他组件还包括嵌入式开发套件（EDK），软件开发套件（SDK）和 ChipScope Pro。

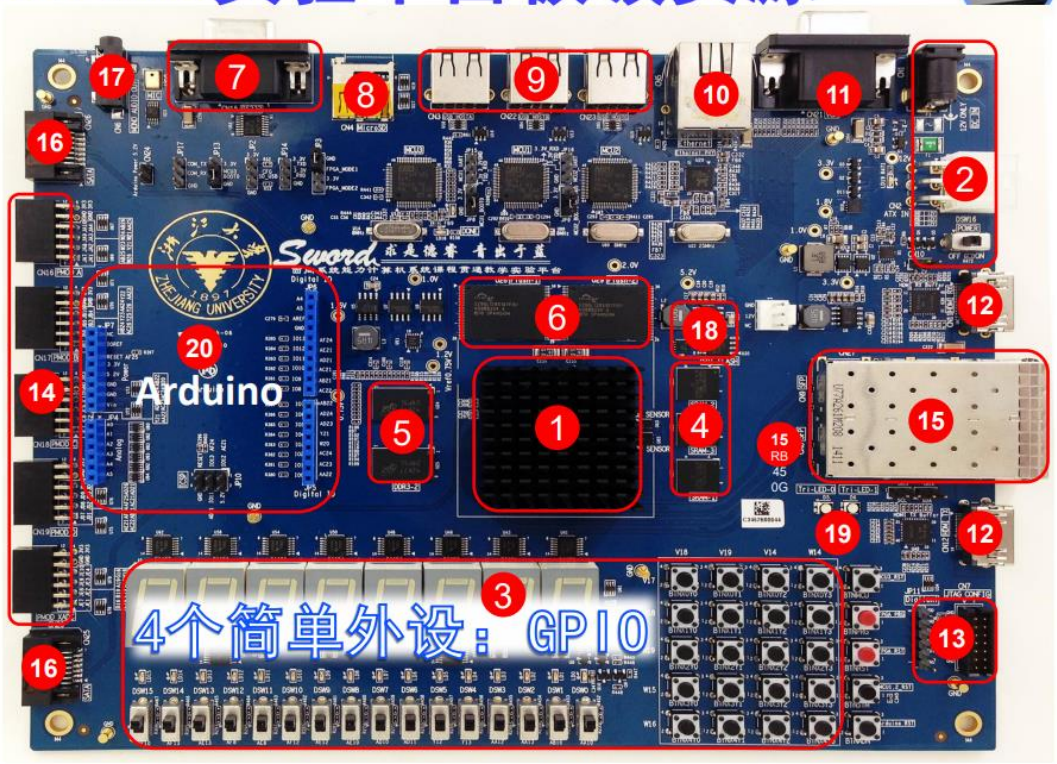
自 2012 年以来，Xilinx ISE 已被 Vivado Design Suite 代替，后者的功能与 ISE 相同，但还带有 SoC 开发的附加功能 Xilinx 于 2013 年 10 月发布了最终版本的 ISE（版本号为 14.7），并声明“ ISE 已进入其产品生命周期的维持阶段，并且不再计划发布 ISE。”

2.4 SWORD

DIGILENT SWORD FPGA 开发板——Simple but Whole cOmputer aRchitecture Design practice platform 简单但完整的计算机体系结构设计平台。

这是一款可由 Vivado 工具链支持的适合计算机学科教学及科研的 FPGA 开发板，带有 Xilinx Kintex-7 FPGA 芯片架构。该款产品是广受欢迎的 Nexys 系列 FPGA 开发板中的升级版本，特别适合计算机学院教学使用，从数字逻辑，计算机组成原理,计算机体系结构,接口技术至操作系统等。SWORD 秉承所有 DIGILENT 开发板一直以来的特色：即用型的硬件，丰富的板载 I/O 口，所有必要的 FPGA 支持电路，免费的软件开发平台，以及适合学生群体的售价。

Sword实验平台板级资源



Sword平台板级资源说明

编号	功能区	编号	功能区
1	XC7K160T-1FFG676C	10	10M/100M/1000M以太网
2	电源输入及开关	11	12位色VGA
3	GPIO（包括按键、拨码开关、LED和7段数码管）	12	HDMI输入（上） HDMI输出（下）
4	SRAM静态存储器	13	FPGA配置（USB-JTAG）
5	DDR3 SDRAM动态存储器	14	Pmod™扩展口
6	NOR 型并行闪存	15	SFP+光模块接口
7	RS232	16	SATA接口
8	MicroSD卡槽	17	麦克风输入和单声道耳机接口
9	USB-OTG、USB-HID	18	SPI闪存（FPGA配置用）

FPGA	XC7K70T	XC7K160T (SWORD)	XC7K325T
Logic Cells	65,600	**162,240**	326,080
BlockRAM (Kb)	4,860	**11,700**	16,020
DSP Slices	240	**600**	840
PCIe® Gen2 Blocks	1	**1**	1
GTX Transceivers (12.5 Gb/s Max Rate)	8	**8**	16
I/O Pins	300	**400**	500

3 单周期处理器概要设计

3.1 总体设计

实验所设计的处理器是一个单周期 RISC-V 处理器，基于 Verilog 硬件设计语言实现，支持除 ecall、ebreak、和控制状态寄存器指令 (csrrc、csrrs、csrrw、csrrci、csrrsi、csrrwi)的 RV32-I 指令集，取指阶段将指令从存储器中读取出来，译码阶段是将存储器取出的指令进行翻译的过程，得到指令中的寄存器索引将寄存器值取出并进行立即数拓展和输出控制信号，执行阶段是对指令进行真正运算处理的阶段，访存阶段根据指令运行的结果访问存储取出数据，或将数据写入存储。写回阶段是将指令执行的结果写回寄存器组的过程。

3.2 PC（程序计数器）

3.2.1 功能描述

保存当前指令的地址，并在时钟上升沿写入下条指令的地址。

3.2.2 模块接口

信号名	方向	描述
clk	input	时钟信号
rst	input	复位信号，使 PC 重置
NPC	input	下条指令的地址
PC	output	当前指令的地址

3.3 RF（寄存器堆）

3.3.1 功能描述

处理器的 32 个通用寄存器位于被称为寄存器堆的结构中，寄存器堆是寄存器的集合，其中的寄存器可以通过指定相应的寄存器号来进行读写。

3.3.2 模块接口

信号名	方向	描述

clk	input	时钟信号
rst	input	复位信号
RFWr	input	寄存器写使能信号
A1,A2	input	所读寄存器号 A1、A2
A3	input	所写寄存器号 A3
WD	input	写入数据
RD1,RD2	output	读出的寄存器 A1、A2 的值

3.4 EXT（立即数生成）

3.4.1 功能描述

根据不同指令所产生的控制信号生成对应的立即数

3.4.2 模块接口

信号名	方向	描述
iimm	input	i 型指令立即数
simm	input	s 型指令立即数
bimm	input	b 型指令立即数
uimm	input	u 型指令立即数
jimm	input	j 型指令立即数
EXTOP	input	立即数生成单元控制信号
immout	output	输出的立即数

3.5 ctrl（控制信号生成）

3.5.1 功能描述

生成控制信号

3.5.2 模块接口

信号名	方向	描述
OP	input	OPCode
Funct7	input	Funct7Code
Funct3	input	Funct3Code
Zero	input	ALU 运算结果是否为 0
RegWrite	output	寄存器写信号
MemWrite	output	内存写信号
EXTOp	output	立即数单元控制信号

ALUOp	output	ALU 单元控制信号
NPCOp	output	NPC 单元控制信号
ALUSrc	output	立即数/寄存器 选择信号
ls	output	字节 半字 字 读写信号
WDSel	output	写回多选器控制信号
ALUSrc_A	output	ALU 源操作数 1 选择信号

3.6 mux_from_rs2_EXT_to_alu（ALU 输入 B 多选器）

3.6.1 功能描述

根据控制信号选择输出立即数或读出的寄存器 rs2

3.6.2 模块接口

信号名	方向	描述
ALUSrc	input	立即数/寄存器 选择信号
RD2	input	寄存器 rs2 的值
imm	input	立即数
B	output	ALU 输入 B

3.7 mux_from_rs1_pc_to_alu（ALU 输入 A 多选器）

3.7.1 功能描述

根据控制信号选择输出 PC 或读出的寄存器 rs1

3.7.2 模块接口

信号名	方向	描述
ALUSrc_A	input	PC/寄存器 选择信号
RD1	input	寄存器 rs 的值
PC	input	当前指令地址
A	output	ALU 输入 A

3.8 alu（算术逻辑单元）

3.8.1 功能描述

根据控制信号进行对应的算术逻辑运算

3.8.2 模块接口

信号名	方向	描述
A	input	算数逻辑单元操作数 A
B	input	算数逻辑单元操作数 B
ALUOp	input	算数逻辑单元控制信号
C	output	算数逻辑单元运算结果
Zero	output	运算结果是否为 0

3.8.3 模块说明

ALUOp	操作
4'b0000	A
4'b0001	A+B
4'b0010	A-B
4'b0011	A&B
4'b0100	A|B
4'b0101	A^B
4'b0110	A<<B
4'b0111	A>>B(逻辑)
4'b1000	A>>B(算数)
4'b1001	A<B?1: 0(有符号)
4'b1010	A<B?1: 0（无符号）

3.9 NPC（下条指令地址生成）

3.9.1 功能描述

根据控制信号进行对应的算术逻辑运算

3.9.2 模块接口

信号名	方向	描述
PC	input	当前指令地址
NPCOp	input	NPC 控制信号
IMM	input	立即数
C	input	算术逻辑单元运算结果
NPC	output	下条指令地址

3.10 mux_from_dm_alu_to_RF（寄存器堆写回数据选择多选器）

3.10.1 功能描述

根据控制信号进行选择寄存器堆写回的数据

3.10.2 模块接口

信号名	方向	描述
WDSel	input	控制信号
C	input	算术逻辑单元运算结果
readdata	input	内存读到的数据
PC	input	当前指令地址
WD	output	寄存器堆写回的数据

4 单周期处理器详细设计

4.1 CPU 总体结构

```
module sccpu(clk,
             rst,
             instr,
             readdata,
             PC,
             MemWrite,
             aluout,
             writedata,
         ls,
             reg_sel,
             reg_data);

    input      clk;          // clock
    input      rst;          // reset

    input [31:0]  instr;     // instruction
    input [31:0]  readdata;  // data from data memory

    output [31:0] PC;        // PC address
    output        MemWrite;  // memory write
    output [31:0] aluout;    // ALU output
    output [31:0] writedata; // data to data memory
    output [3:0]   ls;       // load and write type whb

    input  [4:0] reg_sel;    // register selection (for debug use)
    output [31:0] reg_data;  // selected register data (for debug use)

    wire        RegWrite;    // control signal to register write
    wire [4:0]  EXTOp;       // control signal to signed extension
    wire [3:0]  ALUOp;       // ALU opertion
    wire [1:0]  NPCOp;       // next PC operation
    wire [1:0]  ALUSrc_A;    // ALU source for A
    wire [1:0]  WDSel;       // (register) write data selection
    wire [1:0]  GPRSel;      // general purpose register selection

    wire        ALUSrc;      // ALU source for B
    wire        Zero;        // ALU ouput zero
```

```
wire [31:0] NPC;         // next PC

wire [4:0]  rs1;          // rs
wire [4:0]  rs2;          // rt
wire [4:0]  rd;          // rd
wire [6:0]  Op;          // opcode
wire [6:0]  Funct7;       // funct7
wire [2:0]  Funct3;       // funct3
wire [11:0] Imm12;       // 12-bit immediate
wire [31:0] Imm32;       // 32-bit immediate
wire [19:0] IMM;         // 20-bit immediate (address)
wire [31:0] WD;          // register write data
wire [31:0] RD1;         // register data specified by rs
wire [31:0] A;           // operator for ALU A
wire [31:0] B;           // operator for ALU B


assign Op     = instr[6:0];  // instruction
assign Funct7 = instr[31:25]; // funct7
assign Funct3 = instr[14:12]; // funct3
assign rs1    = instr[19:15];  // rs1
assign rs2    = instr[24:20];  // rs2
assign rd     = instr[11:7];  // rd
assign Imm12  = instr[31:20];// 12-bit immediate
assign IMM    = instr[31:12];  // 20-bit immediate

RF U_RF(
   .clk(clk),
   .rst(rst),
   .RFWr(RegWrite),
   .A1(rs1),.A2(rs2),
   .A3(rd),
   .WD(WD),
   .RD1(RD1),
   .RD2(writedata),
   .reg_sel(reg_sel),
   .reg_data(reg_data)
   );
// instantiation of control unit
ctrl U_ctrl(
     .Op(Op),
     .Funct7(Funct7),
     .Funct3(Funct3),
```

```
      .Zero(Zero),
      .RegWrite(RegWrite),
      .MemWrite(MemWrite),
      .EXTOp(EXTOp),
      .ALUOp(ALUOp),
      .NPCOp(NPCOp),
      .ALUSrc(ALUSrc),
      .ALUSrc_A(ALUSrc_A),
      .ls(ls),
      .WDSel(WDSel)
      );
// instantiation of pc unit
NPC U_NPC(
.PC(PC),
.NPCOp(NPCOp),
.IMM(Imm32),
    .C(aluout),
.NPC(NPC)
);
PC U_PC(
   .clk(clk),
   .rst(rst),
   .NPC(NPC),
   .PC(PC)
   );
// instantiation of alu unit
EXT U_EXT(
.iimm(Imm12),
.simm({Funct7,rd}),
.bimm({Funct7[6],rd[0],Funct7[5:0],rd[4:1]}),
.uimm(IMM),.jimm(IMM),
.EXTOp(EXTOp),
.immout(Imm32)
);
mux_from_rs2_EXT_to_alu U_mux_from_EXT_im_to_alu(
            .ALUSrc(ALUSrc),
            .RD2(writedata),
            .imm(Imm32),
            .B(B)
            );
alu U_alu(
.A(A),
.B(B),
.ALUOp(ALUOp),
```

```
    .C(aluout),
    .Zero(Zero)
    );

    mux_from_dm_alu_to_RF U_mux_from_dm_alu_to_RF(
              .WDSel(WDSel),
              .C(aluout),
              .readdata(readdata),
              .PC(PC),
              .WD(WD)
              );
    mux_from_rs1_pc_to_alu U_mux_from_rs1_pc_to_alu(
               .ALUSrc_A(ALUSrc_A),
               .RD1(RD1),
               .PC(PC),
               .A(A)
               );

endmodule
```

4.2 PC（程序计数器）

```
module PC(clk,
          rst,
          NPC,
          PC);

    input              clk;
    input              rst;
    input       [31:0] NPC;
    output reg  [31:0] PC;

    always @(posedge clk, posedge rst)
        if (rst)
            PC         <= 32'h0000_0000;
            //      PC <= 32'h0000_3000;
        else
            PC <= NPC;

endmodule
```

4.3 RF（寄存器堆）

```
  module RF(   input         clk,
               input         rst,
               input         RFWr,
               input  [4:0]  A1, A2, A3,
               input  [31:0] WD,
               output [31:0] RD1, RD2,
               input  [4:0]  reg_sel,
               output [31:0] reg_data);

  reg [31:0] rf[31:0];

  integer i;

  always @(posedge clk, posedge rst)begin
    if (rst) begin    //  reset
      for (i=0; i<32; i=i+1)
        rf[i] <= 0; //  i;
    end

    else
      if (RFWr) begin
        rf[A3] <= WD;
        $display("r[%2d] = 0x%8X,", A3, WD);
      end
  end


  assign RD1 = (A1 != 0) ? rf[A1] : 0;
  assign RD2 = (A2 != 0) ? rf[A2] : 0;
  assign reg_data = (reg_sel != 0) ? rf[reg_sel] : 0;

endmodule
```

4.4 EXT（立即数生成）

```
`include "ctrl_encode_def.v"

module EXT(input    [11:0]          iimm,
           input    [11:0]          simm,         //instr[31:25, 11:7],
 12 bits
```

```
            input    [11:0]          bimm,         //instrD[31], instrD[
7], instrD[30:25], instrD[11:8], 12 bits
            input    [19:0]          uimm,
            input    [19:0]          jimm,
            input    [4:0]           EXTOp,
            output   reg [31:0]   immout);

    always  @(*)
        case (EXTOp)
            `EXT_CTRL_ITYPE:    immout    <= {{20{iimm[11]}}, iimm[11:0
]};
            `EXT_CTRL_STYPE:    immout    <= {{20{simm[11]}}, simm[11:0
]};
            `EXT_CTRL_BTYPE:    immout <= {{19{bimm[11]}}, bimm[11:0],1
'b0};
            `EXT_CTRL_UTYPE:    immout    <= {uimm[19:0], 12'b0};
            `EXT_CTRL_JTYPE:    immout    <= {{12{jimm[19]}},jimm[7:0],
jimm[8],jimm[18:9],1'b0};
            `SHAMT:             immout          <= {{26{iimm[11]}}, iim
m[4:0]};
            default:            immout    <= 32'b0;
        endcase


endmodule
```

4.5 ctrl（控制信号生成）

```
// `include "ctrl_encode_def.v"


module ctrl(Op,
            Funct7,
            Funct3,
            Zero,
            RegWrite,
            MemWrite,
            EXTOp,
            ALUOp,
            NPCOp,
            ALUSrc,
            ALUSrc_A,
```

```
            ls,
            WDSel);

    input  [6:0] Op;       // opcode
    input  [6:0] Funct7;    // funct7
    input  [2:0] Funct3;    // funct3
    input        Zero;

    output       RegWrite; // control signal for register write
    output       MemWrite; // control signal for memory write
    output [4:0] EXTOp;    // control signal to signed extension
    output [3:0] ALUOp;    // ALU opertion
    output [1:0] NPCOp;    // next pc operation
    output       ALUSrc;   // ALU source for B
    output [1:0] ALUSrc_A; // ALU source for A
    output [1:0] WDSel;    // (register) write data selection
    output [3:0] ls;   // load and write type whb

    // r format
    wire rtype  = ~Op[6]&Op[5]&Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];
    wire i_add  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&~Funct3[1]&~Funct3[0]; // add
    wire i_sub  = rtype& ~Funct7[6]& Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&~Funct3[1]&~Funct3[0]; // sub
    wire i_sll  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&~Funct3[1]&Funct3[0]; //sll
    wire i_slt  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&Funct3[1]&~Funct3[0];//slt
    wire i_sltu = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&Funct3[1]&Funct3[0];//sltu
    wire i_xor  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&Funct3[2]&~Funct3[1]&~Funct3[0];//xor
    wire i_srl  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&Funct3[2]&~Funct3[1]&Funct3[0];//srl
    wire i_sra  = rtype& ~Funct7[6]& Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&Funct3[2]&~Funct3[1]&Funct3[0];//sra
    wire i_or   = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]& Funct3[2]& Funct3[1]&~Funct3[0]; // or
    wire i_and  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]& Funct3[2]& Funct3[1]& Funct3[0]; // and

    // i format
    wire itype   = ~Op[6]&~Op[5]&Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];
    wire i_addi  = itype&~Funct3[2]&~Funct3[1]&~Funct3[0]; // addi
```

```
    wire i_slti  = itype&~Funct3[2]& Funct3[1]&~Funct3[0]; // slti
    wire i_sltiu = itype&~Funct3[2]& Funct3[1]& Funct3[0]; // sltiu
    wire i_xori  = itype& Funct3[2]&~Funct3[1]&~Funct3[0]; // xori
    wire i_ori   = itype& Funct3[2]& Funct3[1]&~Funct3[0]; // ori
    wire i_andi  = itype& Funct3[2]& Funct3[1]& Funct3[0]; // andi

    wire sxli      = ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~Funct
7[2]&~Funct7[1]&~Funct7[0];
    wire sxai      = ~Funct7[6]& Funct7[5]&~Funct7[4]&~Funct7[3]&~Funct
7[2]&~Funct7[1]&~Funct7[0];
    wire shamttype = (sxli|sxai)&itype;
    wire i_slli    = shamttype&sxli&~Funct3[2]&~Funct3[1]& Funct3[0]; /
/slli
    wire i_srli    = shamttype&sxli& Funct3[2]&~Funct3[1]& Funct3[0]; /
/srli
    wire i_srai    = shamttype&sxai& Funct3[2]&~Funct3[1]& Funct3[0]; /
/srai


    // i format
    wire ltype = ~Op[6]&~Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];
    wire i_lb  = ltype& ~Funct3[2]&~Funct3[1]&~Funct3[0]; // lb
    wire i_lh  = ltype& ~Funct3[2]&~Funct3[1]& Funct3[0]; // lh
    wire i_lw  = ltype& ~Funct3[2]& Funct3[1]&~Funct3[0]; // lw
    wire i_lbu = ltype&  Funct3[2]&~Funct3[1]&~Funct3[0]; // lbu
    wire i_lhu = ltype&  Funct3[2]&~Funct3[1]& Funct3[0]; // lhu

    // i format
    wire i_jalr = Op[6]& Op[5]&~Op[4]&~Op[3]& Op[2]& Op[1]& Op[0]&~Func
t3[2]&~Funct3[1]&~Funct3[0];//jalr

    // s format
    wire stype = ~Op[6]&Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];
    wire i_sb  = stype& ~Funct3[2]&~Funct3[1]&~Funct3[0]; // sb
    wire i_sh  = stype& ~Funct3[2]&~Funct3[1]& Funct3[0]; // sh
    wire i_sw  = stype& ~Funct3[2]& Funct3[1]&~Funct3[0]; // sw

    // sb format
    wire sbtype = Op[6]&Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];
    wire i_beq  = sbtype& ~Funct3[2]& ~Funct3[1]&~Funct3[0]; // beq
    wire i_bne  = sbtype& ~Funct3[2]& ~Funct3[1]& Funct3[0]; // bne
    wire i_blt  = sbtype&  Funct3[2]& ~Funct3[1]&~Funct3[0]; // blt
    wire i_bge  = sbtype&  Funct3[2]& ~Funct3[1]& Funct3[0]; // bge
    wire i_bltu = sbtype&  Funct3[2]&  Funct3[1]&~Funct3[0]; // bltu
```

```
    wire i_bgeu = sbtype&  Funct3[2]&  Funct3[1]& Funct3[0]; // bgeu

    // u format
    wire i_lui   =~Op[6]& Op[5]& Op[4]&~Op[3]& Op[2]& Op[1]& Op[0];  //
 lui
    wire i_auipc = ~Op[6]&~Op[5]& Op[4]& ~Op[3]& Op[2]& Op[1]& Op[0];
// auipc

    // j format
    wire i_jal =  Op[6]& Op[5]&~Op[4]& Op[3]& Op[2]& Op[1]& Op[0];  //
jal

    // generate control signals
    assign RegWrite = rtype | ltype |itype| i_jalr | i_jal | i_auipc |
i_lui; // register write

    assign MemWrite = stype;                          // memory write
    assign ALUSrc   = ltype|itype|stype|i_jal|i_jalr|i_lui|i_auipc;   /
/ ALU B is from instruction immediate

    // signed extension
    // SHAMT            5'b11111
    // EXT_CTRL_ITYPE   5'b10000
    // EXT_CTRL_STYPE   5'b01000
    // EXT_CTRL_BTYPE   5'b00100
    // EXT_CTRL_UTYPE   5'b00010
    // EXT_CTRL_JTYPE   5'b00001
    assign EXTOp[4] = itype | ltype|i_jalr|shamttype;
    assign EXTOp[3] = stype|shamttype;
    assign EXTOp[2] = sbtype|shamttype;
    assign EXTOp[1] = i_lui|i_auipc|shamttype;
    assign EXTOp[0] = i_jal|shamttype;



    // WDSel_FromALU 2'b00
    // WDSel_FromMEM 2'b01
    // WDSel_FromPC  2'b10
    assign WDSel[0] = ltype;
    assign WDSel[1] = i_jal|i_jalr;

    // NPC_PLUS4   2'b00
    // NPC_BRANCH  2'b01
    // NPC_JUMP    2'b10
```

```
    assign NPCOp[0] = (i_beq & Zero)|(i_bne & ~Zero)|(i_blt&~Zero)|(i_b
ge& Zero)|(i_bltu&~Zero)|(i_bgeu& Zero)|i_jalr;
    assign NPCOp[1] = i_jal|i_jalr;


    // ALU_NOP   4'b0000
    // ALU_ADD   4'b0001
    // ALU_SUB   4'b0010
    // ALU_AND   4'b0011
    // ALU_OR    4'b0100
    // ALU_XOR   4'b0101
    // ALU_SL    4'b0110
    // ALU_SRL   4'b0111
    // ALU_SRA   4'b1000
    // ALU_LT    4'b1001
    // ALU_LTU   4'b1010
    assign ALUOp[0] = i_add | i_lw | i_lh |i_lb |i_lbu | i_lhu | i_sw |
 i_addi | i_and |i_slt|i_srl|i_slti|i_xor|i_xori|i_andi|i_srli | i_jalr
 |stype|i_blt |i_bge|i_lui|i_auipc;
    assign ALUOp[1] = i_sub | i_beq | i_and | i_sll| i_sltu|i_srl|i_slt
iu|i_andi|i_slli|i_srli|i_bne|i_bltu|i_bgeu;
    assign ALUOp[2] = i_or | i_ori | i_sll|i_srl|i_xor|i_xori|i_ori|i_s
lli|i_srli;
    assign ALUOp[3] = i_slt | i_sltu |i_sra||i_slti|i_sltiu|i_srai|i_bl
t |i_bge|i_bltu|i_bgeu;


    // ALUSRC_A
    // x[rs1]       2'b00
    // {32{1'b0}}   2'b01
    //pc            2'b10
    assign ALUSrc_A[0] = i_lui;
    assign ALUSrc_A[1] = i_auipc;

    //lstype
    //w             4'b0000
    //h             4'b1000
    //b             4'b0100
    //hu            4'b0010
    //bu            4'b0001
    assign ls[3] = i_sh|i_lh;
    assign ls[2] = i_sb|i_lb;
    assign ls[1] = i_lhu;
    assign ls[0] = i_lbu;
```

```
endmodule
```

4.6 mux_from_rs2_EXT_to_alu（ALU 输入 B 多选器）

```
module mux_from_rs2_EXT_to_alu(input ALUSrc,
                               input [31:0] RD2,
                               input [31:0] imm,
                               output [31:0] B);
    reg [31:0] B_r;
    always @(*) begin
        case (ALUSrc)
            1'b0:begin
                B_r <= RD2;
            end
            1'b1:begin
                B_r <= imm;
            end
        endcase
    end
    assign B = B_r;
endmodule
```

4.7 mux_from_rs1_pc_to_alu（ALU 输入 A 多选器）

```
module mux_from_rs1_pc_to_alu(input [1:0] ALUSrc_A,
                              input [31:0] RD1,
                              input [31:0] PC,
                              output [31:0] A);
    reg [31:0] A_r;
    always @(*) begin
        case (ALUSrc_A)
            2'b00:begin
                A_r <= RD1;
            end
            2'b01:begin
                A_r <= {32{1'b0}};
            end
            2'b10:begin
                A_r <= PC;
            end
        endcase
```

```
    end
    assign A = A_r;
endmodule
```

4.8 alu（算术逻辑单元）

```
`include "ctrl_encode_def.v"

module alu(A,
           B,
           ALUOp,
           C,
           Zero);

    input  signed [31:0] A, B;
    input         [3:0]  ALUOp;
    output signed [31:0] C;
    output Zero;

    reg [31:0] C;
    integer i;

    always @(*)
    begin
        case (ALUOp)
            `ALU_NOP:
            C = A;
            `ALU_ADD:
            C = A+B;
            `ALU_SUB:
            C = A-B;
            `ALU_AND:
            C = A&B;
            `ALU_OR:
            C = A|B;
            `ALU_XOR:
            C = A^B;
            `ALU_SL:
            C = A<<B;
            `ALU_SRL:
            C = A>>B;
            `ALU_SRA:
```

```
            C = A>>>B;
            `ALU_LT:
            begin
                if (A[31]<B[31])
                    C = 0;
                else if (A[31]>B[31])
                    C = 1;
                else
                begin
                    if (A[30:0]<B[30:0])
                        C = 1;
                    else
                        C = 0;
                end
            end
            `ALU_LTU:
            begin
                if (A[31:0]<B[31:0])
                    C = 1;
                else
                    C = 0;
            end
            default:
            C = A;
        endcase
    end // end always

    assign Zero = (C == 32'b0);

endmodule
```

4.9 NPC（下条指令地址生成）

```
`include "ctrl_encode_def.v"

module NPC(PC,
           NPCOp,
           IMM,
           C,
           NPC);  // next pc module

    input  [31:0] PC;        // pc
```

```
    input  [1:0]  NPCOp;     // next pc operation
    input  [31:0] IMM;       // immediate
    input  [31:0] C;         //C from ALU for jalr
    output reg [31:0] NPC;   // next pc

    wire [31:0] PCPLUS4;

    assign PCPLUS4 = PC + 4; // pc + 4

    always @(*) begin
        case (NPCOp)
            `NPC_PLUS4:  NPC = PCPLUS4;
            `NPC_BRANCH: NPC = PC + {{19{IMM[12]}}, IMM[12:2], 2'b00};
            `NPC_JUMP:   NPC = PC + {{11{IMM[20]}}, IMM[20:2], 2'b00};
            `NPC_JALR:   NPC = C;
            default:     NPC = PCPLUS4;
        endcase
    end // end always

endmodule
```

4.10 mux_from_dm_alu_to_RF（寄存器堆写回数据选择多选器）

```
module mux_from_rs1_pc_to_alu(input [1:0] ALUSrc_A,
                              input [31:0] RD1,
                              input [31:0] PC,
                              output [31:0] A);
    reg [31:0] A_r;
    always @(*) begin
        case (ALUSrc_A)
            2'b00:begin
                A_r <= RD1;
            end
            2'b01:begin
                A_r <= {32{1'b0}};
            end
            2'b10:begin
                A_r <= PC;
            end
        endcase
    end
    assign A = A_r;
endmodule
```

5 单周期处理器测试及结果分析

5.1 仿真代码及分析

测试 1：

```
# Test the RISC-V processor in simulation
# 待验证：lui, auipc, addi
# 待验证： sb, sh, sw, lb, lh, lw, lbu, lhu
# 本测试只验证单条指令的功能，不考察转发和冒险检测的功能，所以在相关指令之间添加了足够多的 nop 指令

#      Assembly              Description
main:  lui    x5, 0xF1F2F    #x5 寄存器的高 20 位设置为 0xF1F2F
       auipc  x6, 0x6        #x6 寄存器的结果设置为 0x6004
       addi   x0, x0, 0      #nop 指令
       addi   x0, x0, 0
       addi   x0, x0, 0
       addi   x5, x5, 0x3F4  #将 x5 寄存器的值设置为 0xF1F2F3F4
       addi   x0, x0, 0
       addi   x0, x0, 0
       addi   x0, x0, 0
       addi   x0, x0, 0
       sb     x5, 0(x0)      #数据内存地址 0 处的字节设置为 0xF4
       sb     x5, 1(x0)      #数据内存地址 1 处的字节设置为 0xF4
       sh     x5, 2(x0)      #数据内存地址 2 处的双字节设置为 0xF3F4
       sw     x5, 4(x0)      #数据内存地址 4 处的四字节设置为 0xF1F2F3F4
       addi   x0, x0, 0
       addi   x0, x0, 0
       addi   x0, x0, 0
       addi   x0, x0, 0
       lb     x7, 4(x0)      #x7 寄存器的值设置为 0xFFFFFFF4
       lb     x8, 5(x0)      #x8 寄存器的值设置为 0xFFFFFFF3
       lb     x9, 6(x0)      #x9 寄存器的值设置为 0xFFFFFFF2
       lb     x10, 7(x0)     #x10 寄存器的值设置为 0xFFFFFFF1
       lh     x11, 0(x0)     #x11 寄存器的值设置为 0xFFFFF4F4
       lh     x12, 2(x0)     #x12 寄存器的值设置为 0xFFFFF3F4
       lw     x13, 4(x0)     #x13 寄存器的值设置为 0xF1F2F3F4
       lbu    x14, 0(x0)     #x14 寄存器的值设置为 0xF4
       lbu    x15, 1(x0)     #x15 寄存器的值设置为 0xF4
```

```
    lbu    x16, 2(x0)     #x16 寄存器的值设置为0xF4
    lbu    x17, 3(x0)     #x17 寄存器的值设置为0xF3
    lhu    x18, 4(x0)     #x18 寄存器的值设置为0xF3F4
    lhu    x19, 6(x0)     #x19 寄存器的值设置为0xF1F2
```

测试2

```
# Test the RISC-V processor in simulation
# 已经能正确执行：lui, addi
# 待验证：slt, sltu, slti, sltiu
# 本测试只验证单条指令的功能，不考察转发和冒险检测的功能，所以在相关指令之间添加了足够多的nop 指令

#           Assembly             Description
main:  lui    x5, 0xfffff          #x5 <== 0xFFFFF000
    lui    x6, 0xffffe          #x6 <== 0xFFFFE000
    addi   x0, x0, 0
    addi   x0, x0, 0
    addi   x0, x0, 0
    addi   x0, x0, 0
```

```
    slt    x7, x5, x6           #x7 <== 0x00000000
    slti   x8, x5, 1            #x8 <== 0x00000001

    sltu   x9, x5, x6           #x9 <== 0x00000000
    sltiu  x10, x5, -2048       #x10 <== 0x00000001
    sltiu  x11, x5, 1           #x11 <== 0x00000000
```

测试3

```
# Test the RISC-V processor in simulation
# 已经能正确执行：addi, lui, jalr
# 待验证：beq, bne, blt, bge, bltu, bgeu
# 本测试只验证单条指令的功能，不考察转发和冒险检测的功能，所以在相关指令之间添加了足够多的nop 指令

#           Assembly             Description
main:  addi   x5, x0, 0           #x5 <== 0x0
    addi   x6, x0, 0            #x6 <== 0x0
    lui    x7, 0xfffff          #x7 <== 0xFFFFF000
    addi   x0, x0, 0            #instr 00000013
    addi   x0, x0, 0
    addi   x0, x0, 0
```

```
      beq     x6, x0, br1          #beq taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br1ret: beq   x7, x0, br2ret       #beq not taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x5, x5, 1            #x5 = 2
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br2ret: bne   x7, x0, br3          #bne taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br3ret: bne   x6, x0, br4          #bne not taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x5, x5, 1            #x5 = 4
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br4ret: blt   x7, x6, br5          #blt taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br5ret: blt   x6, x7, br6          #blt not taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x5, x5, 1            #x5 = 6
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br6ret: bge   x6, x0, br7          #bge taken
      addi    x0, x0, 0
```

```
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br7ret: bge     x6, x7, br8          #bge taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br8ret: bge     x7, x0, br9          #bge not taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x5, x5, 1            #x5 = 9
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br9ret: bltu    x6, x7, br10         #bltu taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br10ret:bltu    x7, x6, br11         #bltu not taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x5, x5, 1            #x5 = 0xB
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br11ret:bgeu    x7, x6, br12         #bgtu taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
br12ret:bgeu    x6, x7, br13         #bgtu not taken
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x5, x5, 1            #x5 = 0xD
      addi    x0, x0, 0
      addi    x0, x0, 0
```

```
      addi    x0, x0, 0
br13ret:jalr    x0, x0, end          #x5 should be 0xD for correct implementation
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
```

```
br1:   addi    x5, x5, 1            #x5 = 1
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      jalr    x0, x0, br1ret
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
```

```
br2:   jalr    x0, x0, br2ret
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
```

```
br3:   addi    x5, x5, 1            #x5 = 3
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      jalr    x0, x0, br3ret
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
```

```
br4:   jalr    x0, x0, br4ret
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
```

```
br5:   addi    x5, x5, 1            #x5 = 5
      addi    x0, x0, 0
      addi    x0, x0, 0
      addi    x0, x0, 0
      jalr    x0, x0, br5ret
      addi    x0, x0, 0
```

```
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
```

```
br6:    jalr    x0, x0, br6ret
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
```

```
br7:    addi    x5, x5, 1           #x5 = 7
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
        jalr    x0, x0, br7ret
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
```

```
br8:    addi    x5, x5, 1           #x5 = 8
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
        jalr    x0, x0, br8ret
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
```

```
br9:    jalr    x0, x0, br9ret
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
```

```
br10:   addi    x5, x5, 1           #x5 = 0xA
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
        jalr    x0, x0, br10ret
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
```

```
        addi    x0, x0, 0

br11:   jalr    x0, x0, br11ret
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0

br12:   addi    x5, x5, 1           #x5 = 0xC
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
        jalr    x0, x0, br12ret
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0

br13:   jalr    x0, x0, br13ret
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0
        addi    x0, x0, 0

end:
```

测试4

```
# Test the RISC-V processor in simulation
# 已经能正确执行：addi,lui
# 待验证：add,sub,sll,xor,srl,sra,or,and,slli,srli,srai,xori,ori,andi,jal

#       Assembly              Description

addi x2,x0,1              #x2=1=0x00000001
addi x3,x0,-1             #x3=-1=0xffffffff
add x4,x2,x3              #x4=x2+x3=0x00000000
sub x5,x2,x3              #x5=x2-x3=0x00000002
addi x20,x0,20            #x20=20=0x00000014
sll x6,x2,x20             #x6=x2<<x20=0x00100000
sll x7,x3,x20             #x7=x3<<x20=0xfff00000
lui x8,0x0ffff            #x8=0x0ffff000
lui x9,0xfffff            #x9=0xfffff000
srl x10,x8,x20            #x10=x8>>x20=0x000000ff
```

```
srl x11,x9,x20          #x11=x9>>x20=0x00000fff
sra x12,x8,x20          #x12=x8>>>x20=0x000000ff
sra x13,x9,x20          #x13=x9>>>x20=0xffffffff

slli x14,x2,20          #x14=x2<<20=0x00100000
slli x15,x3,20          #x7=x3<<20=0xfff00000
srli x16,x8,20          #x10=x8>>20=0x000000ff
srli x17,x9,20          #x11=x9>>20=0x00000fff
srai x18,x8,20          #x12=x8>>>20=0x000000ff
srai x19,x9,20          #x13=x9>>>20=0xffffffff

xor x21,x2,x0           #x21=0x00000001
xor x22,x2,x2           #x22=0x00000000
and x23,x2,x2           #x23=0x00000001
or  x24,x2,x0           #x24=0x00000001

xori x25,x2,0           #x25=0x00000001
xori x26,x2,2           #x26=0x00000003
andi x27,x2,1           #x27=0x00000001
ori  x28,x2,2           #x28=0x00000003
jal x29,exit        #x29=0x00000070
addi x30,x0,1           #跳过此语句，x30=0!=1
exit:
addi x31,x0,1       #x31=0x00000001
```

5.2 仿真测试结果

结果 1

```
VSIM 2> run -all
# r[ 5] = 0xf1f2f000,
# r[ 6] = 0x00006004,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 5] = 0xf1f2f3f4,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 7] = 0xffffffff4,
# r[ 8] = 0xffffffff3,
# r[ 9] = 0xffffffff2,
# r[10] = 0xffffffff1,
# r[11] = 0xfffff4f4,
# r[12] = 0xfffff3f4,
# r[13] = 0xf1f2f3f4,
# r[14] = 0x000000f4,
# r[15] = 0x000000f4,
# r[16] = 0x000000f4,
# r[17] = 0x000000f3,
# r[18] = 0x0000f3f4,
# r[19] = 0x0000f1f2,
```

结果 2

```
VSIM 5> run -all
# r[ 5] = 0xfffff000,
# r[ 6] = 0xffffe000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 7] = 0x00000000,
# r[ 8] = 0x00000001,
# r[ 9] = 0x00000000,
# r[10] = 0x00000001,
# r[11] = 0x00000000,
# ** Note: $stop    : C:/Users/Administrator/Desktop/RVSCPU/sccomp_tb.v(51)
#    Time: 1200 ns  Iteration: 0  Instance: /sccomp_tb
```

结果 3

```
VSIM 13> run -all
# r[ 5] = 0x00000000,
# r[ 6] = 0x00000000,
# r[ 7] = 0xfffff000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 5] = 0x00000001,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x000001a0,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,
# r[ 5] = 0x00000002,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 0] = 0x00000210,
# r[ 0] = 0x00000000,  # r[ 5] = 0x00000003,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 5] = 0x00000006,
# r[ 0] = 0x00000000,  # r[ 0] = 0x000001d8,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 5] = 0x00000007,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 5] = 0x00000004,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 0] = 0x00000248,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 5] = 0x00000008,
# r[ 0] = 0x00000000,  # r[ 5] = 0x00000005,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 5] = 0x00000002,  # r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,  # r[ 0] = 0x0000026c,
```

```
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 5] = 0x00000009,  # r[ 5] = 0x0000000c,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 5] = 0x0000000a,  # r[ 0] = 0x000002dc,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x000002a4,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 5] = 0x0000000d,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 0] = 0x00000000,  # r[ 0] = 0x00000000,
# r[ 5] = 0x0000000b,  # r[ 0] = 0x00000180,
```

注意：x0 寄存器虽然显示有赋值，但实际上 x0 一直保持输出 0

结果 4

```
VSIM 16> run -all
# r[ 2] = 0x00000001,
# r[ 3] = 0xffffffff,
# r[ 4] = 0x00000000,
# r[ 5] = 0x00000002,
# r[20] = 0x00000014,
# r[ 6] = 0x00100000,
# r[ 7] = 0xfff00000,
# r[ 8] = 0x0ffff000,  # r[19] = 0xffffffff,
# r[ 9] = 0xfffff000,  # r[21] = 0x00000001,
# r[10] = 0x000000ff,  # r[22] = 0x00000000,
# r[11] = 0x00000fff,  # r[23] = 0x00000001,
# r[12] = 0x000000ff,  # r[24] = 0x00000001,
# r[13] = 0xffffffff,  # r[25] = 0x00000001,
# r[14] = 0x00100000,  # r[26] = 0x00000003,
# r[15] = 0xfff00000,  # r[27] = 0x00000001,
# r[16] = 0x000000ff,  # r[28] = 0x00000003,
# r[17] = 0x00000fff,  # r[29] = 0x00000070,
# r[18] = 0x000000ff,  # r[31] = 0x00000001,
```

6 流水线处理器概要设计

6.1 总体设计

实验所设计的处理器是一个 RISC-V 处理器，基于 Verilog 硬件设计语言实现，支持除 ecall、ebreak、和控制状态寄存器指令 (csrrc、csrrs、csrrw、csrrci、csrrsi、csrrwi)的 RV32-I 指令集，采用五级流水线设计，取指阶段将指令从存储器中读取出来，译码阶段是将存储器取出的指令进行翻译的过程，得到指令中的寄存器索引将寄存器值取出并进行立即数拓展和输出控制信号，执行阶段是对指令进行真正运算处理的阶段，访存阶段根据指令运行的结果访问存储取出数据，或将数据写入存储。写回阶段是将指令执行的结果写回寄存器组的过程。同时采用数据前推单元，提前从内部缓冲取到数据，而不是等到数据达到寄存器或存储器，当一条指令试图在加载指令写入一个寄存器之后读取这个寄存器时 ，阻塞一个时钟周期。控制冒险采用静态预测，假设分支不发生，如果发生，清楚掉指令重新取指。

6.2 PC（程序计数器）

6.2.1 功能描述

保存当前指令的地址，并在时钟上升沿写入下条指令的地址。

6.2.2 模块接口

信号名	方向	描述
clk	input	时钟信号
rst	input	复位信号，使 PC 重置
write_enable	input	使能信号，为 1 时写入下条指令，为 0 误操作
NPC	input	下条指令的地址
PC	output	当前指令的地址

6.3 mux_from_exe_pcplus4_to_pipeif

6.3.1 功能描述

根据控制信号选择输出下条指令的地址。

6.3.2 模块接口

信号名	方向	描述
PCOP	input	下条指令多选器的控制信号
NPC	input	EX 阶段输出的 NPC（PCOP 为 1 时选择输出）
PC	input	当前指令的地址（PCOP 为 0 时选择输出 PC+4）
muxNPC	output	多选器输出下条指令的地址

6.4 RF（寄存器堆）

6.4.1 功能描述

处理器的 32 个通用寄存器位于被称为寄存器堆的结构中，寄存器堆是寄存器的集合，其中的寄存器可以通过指定相应的寄存器号来进行读写。

6.4.2 模块接口

信号名	方向	描述
clk	input	时钟信号
rst	input	复位信号
RFWr	input	寄存器写使能信号
A1,A2	input	所读寄存器号 A1、A2
A3	input	所写寄存器号 A3
WD	input	写入数据
RD1,RD2	output	读出的寄存器 A1、A2 的值

6.5 EXT（立即数生成）

6.5.1 功能描述

根据不同指令所产生的控制信号生成对应的立即数

6.5.2 模块接口

信号名	方向	描述
iimm	input	i 型指令立即数
simm	input	s 型指令立即数
bimm	input	b 型指令立即数
uimm	input	u 型指令立即数
jimm	input	j 型指令立即数
EXTOP	input	立即数生成单元控制信号
immout	output	输出的立即数

6.6 ctrl（控制信号生成）

6.6.1 功能描述

生成控制信号

6.6.2 模块接口

信号名	方向	描述
OP	input	OPCode
Funct7	input	Funct7Code
Funct3	input	Funct3Code
RegWrite	output	寄存器写信号
MemWrite	output	内存写信号
MemRead	output	内存读信号
EXTOp	output	立即数单元控制信号
ALUOp	output	ALU 单元控制信号
NPCOp	output	NPC 单元控制信号
ALUSrc	output	立即数/寄存器 选择信号
ls	output	字节 半字 字 读写信号
WDSel	output	写回多选器控制信号
Zero_1	output	跳转指令是否需要 ALU 单元 Zero 信号为 1 信号

6.7 hazard_detection（阻塞检测单元）

6.7.1 功能描述

根据流水线状态输出是否阻塞的信号

6.7.2 模块接口

信号名	方向	描述

ID_EX_MemRead	input	ID/EX 流水线寄存器 内存读信号
ID_EX_RegisterRd	input	ID/EX 流水线寄存器 目的寄存器号
IF_ID_RegisterRs1	input	IF/ID 流水线寄存器 源寄存器 1 号
IF_ID_RegisterRs2	input	IF/ID 流水线寄存器 源寄存器 2 号
data_nstall	output	流水线不阻塞输出 1，阻塞输出 0

6.8 mux_from_rs2_EXT_to_alu（ALU 输入 B 多选器）

6.8.1 功能描述

根据控制信号选择输出立即数或读出的寄存器 rs2

6.8.2 模块接口

信号名	方向	描述
ALUSrc	input	立即数/寄存器 选择信号
RD2	input	寄存器 rs2 的值
imm	input	立即数
B	output	ALU 输入 B

6.9 forwardingunit（前推控制信号生成）

6.9.1 功能描述

生成前推控制信号

6.9.2 模块接口

信号名	方向	描述
EX_MEM_RegWrite	input	EX/MEM 流水线寄存器 寄存器写信号
EX_MEM_RegisterRd,	input	EX/MEM 流水线寄存器 目的寄存器号
ID_EX_RegisterRs1	input	ID/EX 流水线寄存器 源寄存器 1 号
ID_EX_RegisterRs2	input	ID/EX 流水线寄存器 源寄存器 2 号
MEM_WB_RegWrite	input	MEM/WB 流水线寄存器 寄存器写信号
MEM_WB_RegisterRd	input	MEM/WB 流水线寄存器 目的寄存器号
ForwardA	output	源寄存器 1 前推控制信号
ForwardB	output	源寄存器 2 前推控制信号

6.10 alu（算术逻辑单元）

6.10.1 功能描述

根据控制信号进行对应的算术逻辑运算

6.10.2 模块接口

信号名	方向	描述
A	input	算数逻辑单元操作数 A
B	input	算数逻辑单元操作数 B
ALUOp	input	算数逻辑单元控制信号
C	output	算数逻辑单元运算结果
Zero	output	运算结果是否为 0

6.10.3 模块说明

ALUOp	操作
4'b0000	A
4'b0001	A+B
4'b0010	A-B
4'b0011	A&B
4'b0100	A|B
4'b0101	A^B
4'b0110	A<<B
4'b0111	A>>B(逻辑)
4'b1000	A>>B(算数)
4'b1001	A<B?1: 0(有符号)
4'b1010	A<B?1: 0（无符号）
4'b1011	B

6.11 NPC（下条指令地址生成）

6.11.1 功能描述

根据控制信号进行对应的算术逻辑运算

6.11.2 模块接口

信号名	方向	描述
PC	input	当前指令地址
NPCOp	input	NPC 控制信号
IMM	input	立即数
C	input	算术逻辑单元运算结果
branch	input	是否为分支/跳转指令
NPC	output	下条指令地址
jump	output	分支/跳转是否发生

6.12 mux_from_dm_alu_to_RF（寄存器堆写回数据选择多选器）

6.12.1 功能描述

根据控制信号进行选择寄存器堆写回的数据

6.12.2 模块接口

信号名	方向	描述
WDSel	input	控制信号
C	input	算术逻辑单元运算结果
readdata	input	内存读到的数据
PC	input	当前指令地址
WD	output	寄存器堆写回的数据

6.13 GRE_array（通用流水线寄存器）

6.13.1 功能描述

流水线寄存器

6.13.2 模块接口

信号名	方向	描述

Clk	input	时钟信号
Rst	input	复位信号
write_enable	input	写使能信号
flush	input	清除信号
in	input	流水线寄存器输入
out	output	流水线寄存器输出

6.14 pipeif（if 阶段封装）

6.15 pipeid（id 阶段封装）

6.16 pipeexe（exe 阶段封装）

7　流水线处理器详细设计

7.1 CPU 总体结构

```
module mccpu(
    clk,
    rst,
    instr,
    readdata,
    PC,
    MemWrite,
    aluout,
    writedata,
    ls,
    reg_sel,
    reg_data
    );

    input      clk;          // clock
    input      rst;          // reset

    input [31:0]  instr;     // instruction
    input [31:0]  readdata;  // data from data memory

    output [31:0] PC;        // PC address
    output        MemWrite;  // memory write
    output [31:0] aluout;    // ALU output
    output [31:0] writedata; // data to data memory
    output [3:0]   ls;       // load and write type whb

    input  [4:0] reg_sel;    // register selection (for debug use)
    output [31:0] reg_data;  // selected register data (for debug use)

    //IF


    wire [109:0] EXEMEMIN,EXEMEMOUT;
    wire data_nstall;
    wire [31:0] NPC2if;
    wire jump;
    pipeif U_pipeif(
```

```
        .clk(clk),
        .rst(rst),
        .PCOP(jump),
        .write_enable(data_nstall),
        .NPC(NPC2if),
        .PC(PC)
    );
    wire IFIDflush=(~data_nstall)|jump;
    wire [63:0] IFIDOUT;
    GRE_array #(. WIDTH(64)) IFID(.Clk(clk),.Rst(rst),.write_enable(dat
a_nstall),.flush(IFIDflush),.in({instr,PC}),.out(IFIDOUT));
    wire [31:0] instr2id=IFIDOUT[63:32];


    //ID
    wire [159:0] IDEXIN,IDEXOUT;
    wire Memread;



    wire [39:0]MEMWBIN,MEMWBOUT;
    wire [31:0] WD2id=MEMWBOUT[39:8];
    wire [4:0] rd2id=MEMWBOUT[7:3];
    wire Regwrite2id=MEMWBOUT[2];

    pipeid U_pipeid(
        .clk(clk),
        .rst(rst),
        .instr(instr2id),
        .RegWrite(Regwrite2id),
        .rd(rd2id),
        .WD(WD2id),
        .RD12exe(IDEXIN[116:85]),
        .RD22exe(IDEXIN[84:53]),
        .ALUOp2exe(IDEXIN[52:49]),
        .NPCOp2exe(IDEXIN[48:47]),
        .ALUSrc2exe(IDEXIN[46]),
        .Imm322exe(IDEXIN[45:14]),
        .Zero_12exe(IDEXIN[13]),
        .MemWrite2mem(IDEXIN[12]),
        .ls2mem(IDEXIN[11:8]),
        .rd2wb(IDEXIN[7:3]),
        .RegWrite2wb(IDEXIN[2]),
        .WDSel2wb(IDEXIN[1:0]),
        .Memread(Memread)
```

```
    );
    hazard_detection U_hazard_detection(
        .ID_EX_MemRead(IDEXOUT[149]),
        .ID_EX_RegisterRd(IDEXOUT[7:3]),
        .IF_ID_RegisterRs1(instr2id[19:15]),
        .IF_ID_RegisterRs2(instr2id[24:20]),
        .data_nstall(data_nstall)
    );

    wire IDEXflush=jump|(~data_nstall);
    assign IDEXIN[154:150]=instr2id[19:15];
    assign IDEXIN[159:155]=instr2id[24:20];
    assign IDEXIN[149]=Memread;
    assign IDEXIN[148:117]=IFIDOUT[31:0];
    GRE_array #(. WIDTH(160)) IDEX(.Clk(clk),.Rst(rst),.write_enable(1'
b1),.flush(IDEXflush),.in(IDEXIN),.out(IDEXOUT));
    wire ALUSrc2exe=IDEXOUT[46];
    wire [31:0] RD12exe=IDEXOUT[116:85];
    wire [31:0] RD22exe=IDEXOUT[84:53];
    wire [31:0] Imm322exe=IDEXOUT[45:14];
    wire [3:0] ALUOp2exe=IDEXOUT[52:49];
    wire [31:0] PC2exe=IDEXOUT[148:117];
    wire [1:0] NPCOp2exe=IDEXOUT[48:47];
    wire Zero_12exe=IDEXOUT[13];

    //EXE
    wire [1:0] ForwardA,ForwardB;
    pipeexe U_pipeexe(
        .ALUSrc(ALUSrc2exe),
        .RD1(RD12exe),
        .RD2(RD22exe),
        .Imm32(Imm322exe),
        .ALUOp(ALUOp2exe),
        .PC(PC2exe),
        .NPCOp(NPCOp2exe),
        .Zero_1(Zero_12exe),
        .ForwardA(ForwardA),
        .ForwardB(ForwardB),
        .EX_MEM_Register(EXEMEMOUT[76:45]),
        .MEM_WB_Register(MEMWBOUT[39:8]),
        .aluout(EXEMEMIN[76:45]),
        .NPC2if(NPC2if),
        .jump2if(jump),
        .Bs(EXEMEMIN[44:13])
```

```
    );
    forwardingunit U_forwardingunit(
        .EX_MEM_RegWrite(EXEMEMOUT[2]),
        .EX_MEM_RegisterRd(EXEMEMOUT[7:3]),
        .ID_EX_RegisterRs1(IDEXOUT[154:150]),
        .ID_EX_RegisterRs2(IDEXOUT[159:155]),
        .MEM_WB_RegWrite(MEMWBOUT[2]),
        .MEM_WB_RegisterRd(MEMWBOUT[7:3]),
        .ForwardA(ForwardA),
        .ForwardB(ForwardB)
    );

    assign EXEMEMIN[7:0]=IDEXOUT[7:0]; //wbreg
    assign EXEMEMIN[12:8]=IDEXOUT[12:8];//memreg     EXEMEMIN[44:13] me
mwritedata
    assign EXEMEMIN[108:77]=PC2exe;
    GRE_array #(. WIDTH(109)) EXEMEM(.Clk(clk),.Rst(rst),.write_enable(
1'b1),.flush(1'b0),.in(EXEMEMIN),.out(EXEMEMOUT));


    //MEM

    assign MemWrite=EXEMEMOUT[12];
    assign aluout=EXEMEMOUT[76:45];
    assign writedata=EXEMEMOUT[44:13];
    assign ls=EXEMEMOUT[11:8];

    wire [1:0] WDSel=EXEMEMOUT[1:0];
    wire [31:0] PC2muxRF=EXEMEMOUT[108:77];
    wire [31:0] WD2RF;
    mux_from_dm_alu_to_RF U_mux_from_dm_alu_to_RF(
        .WDSel(WDSel),
        .C(aluout),
        .readdata(readdata),
        .PC(PC2muxRF),
        .WD(WD2RF)
    );

    assign MEMWBIN[7:0]=EXEMEMOUT[7:0];
    assign MEMWBIN[39:8]=WD2RF;
    GRE_array #(. WIDTH(40)) MEMWB(.Clk(clk),.Rst(rst),.write_enable(1'
b1),.flush(1'b0),.in(MEMWBIN),.out(MEMWBOUT));
    //WB
    //write back from MEMWB Reg
```

```
endmodule
```

7.2 PC（程序计数器）

```
module PC(clk,
          rst,
          write_enable,
          NPC,
          PC);

    input              clk;
    input              rst;
    input              write_enable;
    input       [31:0] NPC;
    output reg  [31:0] PC;

    always @(posedge clk, posedge rst)
        if (rst)
            PC         <= 32'h0000_0000;
            //      PC <= 32'h0000_3000;
        else if(write_enable)
            PC <= NPC;
endmodule
```

7.3 mux_from_exe_pcplus4_to_pipeif

```
module mux_from_exe_pcplus4_to_pipeif(
    input  PCOP,
    input [31:0] NPC,
    input [31:0] PC,
    output reg [31:0] muxNPC);
    always @(*) begin
        case (PCOP)
            1'b0:begin
                muxNPC <= PC+4;
            end
            1'b1:begin
                muxNPC <= NPC;
```

```
            end
            default:begin
                muxNPC <= PC+4;
            end
        endcase
    end
endmodule
```

7.4 RF（寄存器堆）

```
module RF(   input         clk,
             input         rst,
             input         RFWr,
             input  [4:0]  A1, A2, A3,
             input  [31:0] WD,
             output [31:0] RD1, RD2,
             input  [4:0]  reg_sel,
             output [31:0] reg_data);

reg [31:0] rf[31:0];

integer i;

always @(negedge clk, posedge rst)begin
  if (rst) begin    //  reset
    for (i=0; i<32; i=i+1)
      rf[i] <= 0; //  i;
  end

  else
    if (RFWr) begin
      rf[A3] <= WD;
    rf[0]<=0;
      $display("r[%2d] = 0x%8X,", A3, WD);
    end
end


assign RD1 = (A1 != 0) ? rf[A1] : 0;
assign RD2 = (A2 != 0) ? rf[A2] : 0;
```

```
  assign reg_data = (reg_sel != 0) ? rf[reg_sel] : 0;

endmodule
```

7.5 EXT（立即数生成）

```
`include "ctrl_encode_def.v"

module EXT(input    [11:0]          iimm,
           input    [11:0]          simm,         //instr[31:25, 11:7],
 12 bits
           input    [11:0]          bimm,         //instrD[31], instrD[
7], instrD[30:25], instrD[11:8], 12 bits
           input    [19:0]          uimm,
           input    [19:0]          jimm,
           input    [4:0]           EXTOp,
           output   reg [31:0]   immout);

    always  @(*)
        case (EXTOp)
            `EXT_CTRL_ITYPE:    immout    <= {{20{iimm[11]}}, iimm[11:0
]};
            `EXT_CTRL_STYPE:    immout    <= {{20{simm[11]}}, simm[11:0
]};
            `EXT_CTRL_BTYPE:    immout <= {{19{bimm[11]}}, bimm[11:0],1
'b0};
            `EXT_CTRL_UTYPE:    immout    <= {uimm[19:0], 12'b0};
            `EXT_CTRL_JTYPE:    immout    <= {{12{jimm[19]}},jimm[7:0],
jimm[8],jimm[18:9],1'b0};
            `SHAMT:             immout          <= {{26{iimm[11]}}, iim
m[4:0]};
            default:            immout    <= 32'b0;
        endcase


endmodule
```

7.6 ctrl（控制信号生成）

```
// `include "ctrl_encode_def.v"
```

```
module ctrl(Op,
            Funct7,
            Funct3,
            RegWrite,
            MemWrite,
            EXTOp,
            ALUOp,
            NPCOp,
            ALUSrc,
            ls,
            WDSel,
            Zero_1,
            Memread
            );

    input  [6:0] Op;       // opcode
    input  [6:0] Funct7;    // funct7
    input  [2:0] Funct3;    // funct3

    output       RegWrite; // control signal for register write
    output       MemWrite; // control signal for memory write
    output [4:0] EXTOp;    // control signal to signed extension
    output [3:0] ALUOp;    // ALU opertion
    output [1:0] NPCOp;    // next pc operation
    output       ALUSrc;   // ALU source for B
    output [1:0] WDSel;    // (register) write data selection
    output [3:0] ls;   // load and write type whb
    output Zero_1;
    output Memread;

    // r format
    wire rtype  = ~Op[6]&Op[5]&Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];
    wire i_add  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&~Funct3[1]&~Funct3[0]; // add
    wire i_sub  = rtype& ~Funct7[6]& Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&~Funct3[1]&~Funct3[0]; // sub
    wire i_sll  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&~Funct3[1]&Funct3[0]; //sll
    wire i_slt  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&Funct3[1]&~Funct3[0];//slt
    wire i_sltu = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&Funct3[1]&Funct3[0];//sltu
```

```
    wire i_xor  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&Funct3[2]&~Funct3[1]&~Funct3[0];//xor
    wire i_srl  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&Funct3[2]&~Funct3[1]&Funct3[0];//srl
    wire i_sra  = rtype& ~Funct7[6]& Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]&Funct3[2]&~Funct3[1]&Funct3[0];//sra
    wire i_or   = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]& Funct3[2]& Funct3[1]&~Funct3[0]; // or
    wire i_and  = rtype& ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~F
unct7[2]&~Funct7[1]&~Funct7[0]& Funct3[2]& Funct3[1]& Funct3[0]; // and

    // i format
    wire itype   = ~Op[6]&~Op[5]&Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];
    wire i_addi  = itype&~Funct3[2]&~Funct3[1]&~Funct3[0]; // addi
    wire i_slti  = itype&~Funct3[2]& Funct3[1]&~Funct3[0]; // slti
    wire i_sltiu = itype&~Funct3[2]& Funct3[1]& Funct3[0]; // sltiu
    wire i_xori  = itype& Funct3[2]&~Funct3[1]&~Funct3[0]; // xori
    wire i_ori   = itype& Funct3[2]& Funct3[1]&~Funct3[0]; // ori
    wire i_andi  = itype& Funct3[2]& Funct3[1]& Funct3[0]; // andi

    wire sxli      = ~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~Funct
7[2]&~Funct7[1]&~Funct7[0];
    wire sxai      = ~Funct7[6]& Funct7[5]&~Funct7[4]&~Funct7[3]&~Funct
7[2]&~Funct7[1]&~Funct7[0];
    wire shamttype = (sxli|sxai)&itype;
    wire i_slli    = shamttype&sxli&~Funct3[2]&~Funct3[1]& Funct3[0]; /
/slli
    wire i_srli    = shamttype&sxli& Funct3[2]&~Funct3[1]& Funct3[0]; /
/srli
    wire i_srai    = shamttype&sxai& Funct3[2]&~Funct3[1]& Funct3[0]; /
/srai


    // i format
    wire ltype = ~Op[6]&~Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];
    wire i_lb  = ltype& ~Funct3[2]&~Funct3[1]&~Funct3[0]; // lb
    wire i_lh  = ltype& ~Funct3[2]&~Funct3[1]& Funct3[0]; // lh
    wire i_lw  = ltype& ~Funct3[2]& Funct3[1]&~Funct3[0]; // lw
    wire i_lbu = ltype&  Funct3[2]&~Funct3[1]&~Funct3[0]; // lbu
    wire i_lhu = ltype&  Funct3[2]&~Funct3[1]& Funct3[0]; // lhu
    assign Memread=ltype;

    // i format
```

```
    wire i_jalr = Op[6]& Op[5]&~Op[4]&~Op[3]& Op[2]& Op[1]& Op[0]&~Func
t3[2]&~Funct3[1]&~Funct3[0];//jalr

    // s format
    wire stype = ~Op[6]&Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];
    wire i_sb  = stype& ~Funct3[2]&~Funct3[1]&~Funct3[0]; // sb
    wire i_sh  = stype& ~Funct3[2]&~Funct3[1]& Funct3[0]; // sh
    wire i_sw  = stype& ~Funct3[2]& Funct3[1]&~Funct3[0]; // sw

    // sb format
    wire sbtype = Op[6]&Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];
    wire i_beq  = sbtype& ~Funct3[2]& ~Funct3[1]&~Funct3[0]; // beq
    wire i_bne  = sbtype& ~Funct3[2]& ~Funct3[1]& Funct3[0]; // bne
    wire i_blt  = sbtype&  Funct3[2]& ~Funct3[1]&~Funct3[0]; // blt
    wire i_bge  = sbtype&  Funct3[2]& ~Funct3[1]& Funct3[0]; // bge
    wire i_bltu = sbtype&  Funct3[2]&  Funct3[1]&~Funct3[0]; // bltu
    wire i_bgeu = sbtype&  Funct3[2]&  Funct3[1]& Funct3[0]; // bgeu

    // u format
    wire i_lui   =~Op[6]& Op[5]& Op[4]&~Op[3]& Op[2]& Op[1]& Op[0];  //
 lui
    wire i_auipc = ~Op[6]&~Op[5]& Op[4]& ~Op[3]& Op[2]& Op[1]& Op[0];
// auipc

    // j format
    wire i_jal =  Op[6]& Op[5]&~Op[4]& Op[3]& Op[2]& Op[1]& Op[0];  //
jal

    // generate control signals
    assign RegWrite = rtype | ltype |itype| i_jalr | i_jal | i_auipc |
i_lui; // register write

    assign MemWrite = stype;                          // memory write
    assign ALUSrc   = ltype|itype|stype|i_jal|i_jalr|i_lui|i_auipc;   /
/ ALU B is from instruction immediate

    // signed extension
    // SHAMT            5'b11111
    // EXT_CTRL_ITYPE   5'b10000
    // EXT_CTRL_STYPE   5'b01000
    // EXT_CTRL_BTYPE   5'b00100
    // EXT_CTRL_UTYPE   5'b00010
    // EXT_CTRL_JTYPE   5'b00001
    assign EXTOp[4] = itype | ltype|i_jalr|shamttype;
```

```
    assign EXTOp[3] = stype|shamttype;
    assign EXTOp[2] = sbtype|shamttype;
    assign EXTOp[1] = i_lui|i_auipc|shamttype;
    assign EXTOp[0] = i_jal|shamttype;



    // WDSel_FromALU 2'b00
    // WDSel_FromMEM 2'b01
    // WDSel_FromPC  2'b10
     // WDSel_FromPC+ALU(B) 2'b11
    assign WDSel[0] = ltype|i_auipc;
    assign WDSel[1] = i_jal|i_jalr|i_auipc;

    // NPC_PLUS4   2'b00
    // NPC_BRANCH  2'b01
    // NPC_JUMP    2'b10
    // NPC_JALR    2'b11
    assign NPCOp[0] = i_beq|i_bne|i_blt|i_bge|i_bltu|i_bgeu|i_jalr;
    assign NPCOp[1] = i_jal|i_jalr;
    assign Zero_1  = i_beq|i_bge|i_bgeu;



    // ALU_NOP   4'b0000
    // ALU_ADD   4'b0001
    // ALU_SUB   4'b0010
    // ALU_AND   4'b0011
    // ALU_OR    4'b0100
    // ALU_XOR   4'b0101
    // ALU_SL    4'b0110
    // ALU_SRL   4'b0111
    // ALU_SRA   4'b1000
    // ALU_LT    4'b1001
    // ALU_LTU   4'b1010
     // ALU_B     4'b1011
    assign ALUOp[0] = i_add | i_lw | i_lh |i_lb |i_lbu | i_lhu | i_sw |
 i_addi | i_and |i_slt|i_srl|i_slti|i_xor|i_xori|i_andi|i_srli | i_jalr
 |stype|i_blt |i_bge|i_lui|i_auipc;
    assign ALUOp[1] = i_sub | i_beq | i_and | i_sll| i_sltu|i_srl|i_slt
iu|i_andi|i_slli|i_srli|i_bne|i_bltu|i_bgeu|i_lui|i_auipc;
    assign ALUOp[2] = i_or | i_ori | i_sll|i_srl|i_xor|i_xori|i_ori|i_s
lli|i_srli;
    assign ALUOp[3] = i_slt | i_sltu |i_sra||i_slti|i_sltiu|i_srai|i_bl
t |i_bge|i_bltu|i_bgeu|i_lui|i_auipc;
```

```
    //lstype
    //w             4'b0000
    //h             4'b1000
    //b             4'b0100
    //hu            4'b0010
    //bu            4'b0001
    assign ls[3] = i_sh|i_lh;
    assign ls[2] = i_sb|i_lb;
    assign ls[1] = i_lhu;
    assign ls[0] = i_lbu;
endmodule
```

7.7 hazard_detection（阻塞检测单元）

```
module hazard_detection (
    input ID_EX_MemRead,
    input [4:0] ID_EX_RegisterRd,
    input [4:0] IF_ID_RegisterRs1,
    input [4:0] IF_ID_RegisterRs2,
    output reg data_nstall
);
    always @(*) begin
        if(ID_EX_MemRead &((ID_EX_RegisterRd == IF_ID_RegisterRs1)|(ID_
EX_RegisterRd == IF_ID_RegisterRs2)))begin
            data_nstall<=0;
        end
        else begin
            data_nstall<=1;
        end
    end
endmodule
```

7.8 mux_from_rs2_EXT_to_alu（ALU 输入 B 多选器）

```
module mux_from_rs2_EXT_to_alu(input ALUSrc,
                               input [31:0] RD2,
                               input [31:0] imm,
                               output reg [31:0] B);
```

```
    always @(*) begin
        case (ALUSrc)
            1'b0:begin
                B <= RD2;
            end
            1'b1:begin
                B <= imm;
            end
        endcase
    end
endmodule
```

7.9 forwardingunit（前推控制信号生成）

```
module forwardingunit (
    input EX_MEM_RegWrite,
    input [4:0] EX_MEM_RegisterRd,
    input [4:0] ID_EX_RegisterRs1,
    input [4:0] ID_EX_RegisterRs2,
    input MEM_WB_RegWrite,
    input [4:0] MEM_WB_RegisterRd,
    output reg [1:0] ForwardA,
    output reg [1:0] ForwardB
);
    always @(*) begin
        if (EX_MEM_RegWrite& (EX_MEM_RegisterRd != 0)& (EX_MEM_Register
Rd == ID_EX_RegisterRs1))
            ForwardA <= 2'b10;
        else if (MEM_WB_RegWrite& (MEM_WB_RegisterRd != 0)& ~(EX_MEM_Re
gWrite & (EX_MEM_RegisterRd != 0)& (EX_MEM_RegisterRd == ID_EX_Register
Rs1))& (MEM_WB_RegisterRd == ID_EX_RegisterRs1))
            ForwardA <= 2'b01;
        else
            ForwardA <= 2'b00;
        if (EX_MEM_RegWrite& (EX_MEM_RegisterRd != 0)& (EX_MEM_Register
Rd == ID_EX_RegisterRs2))
            ForwardB <= 2'b10;
        else if (MEM_WB_RegWrite& (MEM_WB_RegisterRd != 0)& ~(EX_MEM_Re
gWrite & (EX_MEM_RegisterRd != 0)& (EX_MEM_RegisterRd == ID_EX_Register
Rs2))& (MEM_WB_RegisterRd == ID_EX_RegisterRs2))
            ForwardB <= 2'b01;
        else
            ForwardB <= 2'b00;
```

```
    end
endmodule
```

7.10 alu（算术逻辑单元）

```
`include "ctrl_encode_def.v"

module alu(A,
           B,
           ALUOp,
           C,
           Zero);

    input  signed [31:0] A, B;
    input         [3:0]  ALUOp;
    output signed [31:0] C;
    output Zero;

    reg [31:0] C;
    integer i;

    always @(*)
    begin
        case (ALUOp)
            `ALU_NOP:
            C <= A;
            `ALU_ADD:
            C <= A+B;
            `ALU_SUB:
            C <= A-B;
            `ALU_AND:
            C <= A&B;
            `ALU_OR:
            C <= A|B;
            `ALU_XOR:
            C <= A^B;
            `ALU_SL:
            C <= A<<B;
            `ALU_SRL:
            C <= A>>B;
            `ALU_SRA:
            C <= A>>>B;
            `ALU_LT:
```

```
            begin
                if (A[31]<B[31])
                    C <= 0;
                else if (A[31]>B[31])
                    C <= 1;
                else
                begin
                    if (A[30:0]<B[30:0])
                        C <= 1;
                    else
                        C <= 0;
                end
            end
            `ALU_LTU:
            begin
                if (A[31:0]<B[31:0])
                    C <= 1;
                else
                    C <= 0;
            end
                `ALU_B:
            begin
                C <= B;
            end
            default:
            C <= A;
        endcase
    end // end always

    assign Zero = (C == 32'b0);

endmodule
```

7.11 NPC（下条指令地址生成）

```
`include "ctrl_encode_def.v"

module NPC(PC,
           NPCOp,
           IMM,
           C,
```

```
        branch,
        NPC,
        jump);  // next pc module

input  [31:0] PC;        // pc
input  [1:0]  NPCOp;     // next pc operation
input  [31:0] IMM;       // immediate
input  [31:0] C;         //C from ALU for jalr
input branch;
output reg [31:0] NPC;   // next pc
output reg jump;

wire [31:0] PCPLUS4;

assign PCPLUS4 = PC + 4; // pc + 4

always @(*) begin
    case (NPCOp)
        `NPC_PLUS4:begin
            NPC <= PCPLUS4;
            jump<=0;
        end
        `NPC_BRANCH:begin
            if(branch) begin
                NPC <= PC + {{19{IMM[12]}}, IMM[12:2], 2'b00};
                jump <=1;
            end
            else begin
                NPC <=PCPLUS4;
                jump<=0;
            end
        end
        `NPC_JUMP:begin
            NPC <= PC + {{11{IMM[20]}}, IMM[20:2], 2'b00};
            jump<=1;
        end
        `NPC_JALR:begin
            NPC <= C;
            jump<=1;
        end
        default:begin
            NPC <= PCPLUS4;
            jump<=0;
        end
```

```
        endcase
    end // end always

endmodule
```

7.12 mux_from_dm_alu_to_RF（寄存器堆写回数据选择多选器）

```
module mux_from_dm_alu_to_RF(input [1:0] WDSel,
                             input [31:0] C,
                             input [31:0] readdata,
                             input [31:0] PC,
                             output reg [31:0] WD);
    always @(*) begin
        case (WDSel)
            2'b00:begin
                WD <= C;
            end
            2'b01:begin
                WD <= readdata;
            end
            2'b10:begin
                WD <= PC+4;
            end
            2'b11:begin
                WD <= PC+C;
            end
            default:begin
                WD <=C;
            end
        endcase
    end
endmodule
```

7.13 GRE_array（通用流水线寄存器）

```
module GRE_array #(
    parameter WIDTH=300
) (
    input Clk,Rst,write_enable,flush,
```

```
    input [WIDTH-1:0] in,
    output reg [WIDTH-1:0] out
);
    always @(posedge Clk) begin
        if(write_enable)begin
            if(flush)
                out<=0;
            else
                out<=in;
        end
          if(Rst)begin
              out<=0;
          end
    end
endmodule
```

7.14 pipeif（if 阶段封装）

```
module pipeif (
    input clk,
    input rst,
    input PCOP,
    input write_enable,
    input [31:0] NPC,
    output [31:0] PC
);
    wire [31:0] muxNPC;
    mux_from_exe_pcplus4_to_pipeif U_mux_mux_from_exe_pcplus4_to_pipeif
(
        .PCOP(PCOP),
        .NPC(NPC),
        .PC(PC),
        .muxNPC(muxNPC)
    );
    PC U_PC(
        .clk(clk),
        .rst(rst),
        .write_enable(write_enable),
        .NPC(muxNPC),
        .PC(PC)
    );
endmodule
```

7.15 pipeid（id 阶段封装）

```
module pipeid (
    input clk,
    input rst,
    input [31:0] instr,
    input RegWrite,
    input [4:0] rd,
    input [31:0] WD,
    output [31:0] RD12exe,RD22exe,
    output [4:0] rd2wb,
    output RegWrite2wb,
    output MemWrite2mem,
    output [3:0] ALUOp2exe,
    output [1:0] NPCOp2exe,
    output ALUSrc2exe,
    output [3:0] ls2mem,
    output [1:0] WDSel2wb,
    output [31:0] Imm322exe,
    output Zero_12exe,
    output Memread
);
    wire [4:0]  rs1;          // rs
    wire [4:0]  rs2;          // rt
    wire [6:0]  Op;          // opcode
    wire [6:0]  Funct7;       // funct7
    wire [2:0]  Funct3;       // funct3
    wire [11:0] Imm12;       // 12-bit immediate
    wire [19:0] IMM;         // 20-bit immediate (address)

    wire [4:0] EXTOp;

    assign Op     = instr[6:0];  // instruction
    assign Funct7 = instr[31:25]; // funct7
    assign Funct3 = instr[14:12]; // funct3
    assign rs1    = instr[19:15];  // rs1
    assign rs2    = instr[24:20];  // rs2
    assign rd2wb  = instr[11:7];  // rd for wb
    assign Imm12  = instr[31:20];// 12-bit immediate
    assign IMM    = instr[31:12];  // 20-bit immediate


    RF U_RF(
       .clk(clk),
```

```
        .rst(rst),
        .RFWr(RegWrite),
        .A1(rs1),.A2(rs2),
        .A3(rd),
        .WD(WD),
        .RD1(RD12exe),
        .RD2(RD22exe),
        .reg_sel(),
        .reg_data()
        );
    ctrl U_ctrl(
         .Op(Op),
         .Funct7(Funct7),
         .Funct3(Funct3),
         .RegWrite(RegWrite2wb),
         .MemWrite(MemWrite2mem),
         .EXTOp(EXTOp),
         .ALUOp(ALUOp2exe),
         .NPCOp(NPCOp2exe),
         .ALUSrc(ALUSrc2exe),
         .ls(ls2mem),
         .WDSel(WDSel2wb),
         .Zero_1(Zero_12exe),
         .Memread(Memread)
          );
    EXT U_EXT(
         .iimm(Imm12),
         .simm({Funct7,rd2wb}),
         .bimm({Funct7[6],rd2wb[0],Funct7[5:0],rd2wb[4:1]}),
         .uimm(IMM),.jimm(IMM),
         .EXTOp(EXTOp),
         .immout(Imm322exe)
    );
endmodule
```

7.16 pipeexe（exe 阶段封装）

```
module pipeexe (
    input ALUSrc,
    input [31:0] RD1,
```

```
    input [31:0] RD2,
    input [31:0] Imm32,
    input [3:0] ALUOp,
    input [31:0] PC,
    input [1:0] NPCOp,
    input Zero_1,
    input [1:0] ForwardA,
    input [1:0] ForwardB,
    input [31:0] EX_MEM_Register,
    input [31:0] MEM_WB_Register,
    output [31:0] aluout,
    output [31:0] NPC2if,
    output jump2if,
    output [31:0] Bs
);
    wire [31:0] A,B;
    wire Zero;
    mux_from_rs2_EXT_to_alu U_mux_from_EXT_im_to_alu(
        .ALUSrc(ALUSrc),
        .RD2(Bs),
        .imm(Imm32),
        .B(B)
        );
    mux_from_forward_to_AB U_mux_from_forward_to_A(
        .Forward(ForwardA),
        .RD(RD1),
        .EX_MEM_Register(EX_MEM_Register),
        .MEM_WB_Register(MEM_WB_Register),
        .AB(A)
    );
    mux_from_forward_to_AB U_mux_from_forward_to_B(
        .Forward(ForwardB),
        .RD(RD2),
        .EX_MEM_Register(EX_MEM_Register),
        .MEM_WB_Register(MEM_WB_Register),
        .AB(Bs)
    );
    alu U_alu(
        .A(A),
        .B(B),
        .ALUOp(ALUOp),
        .C(aluout),
        .Zero(Zero)
    );
```

```
    wire branch=~(Zero_1^Zero);//是否分支
    NPC U_NPC(
        .PC(PC),
        .NPCOp(NPCOp),
        .IMM(Imm32),
        .C(aluout),
        .branch(branch),
        .NPC(NPC2if),
        .jump(jump2if)
    );
endmodule
```

8 流水线处理器测试及结果分析

8.1 仿真代码及分析

流水线仿真代码重点分析数据冒险和控制冒险中的前推、阻塞和清除。正常指令运行结果同前面相同。

```
# Test the RISC-V processor in simulation
#  已经能正确执行：addi, add, lw, sw, beq, jalr
#  待验证：能否正确处理前推：MEM-->EX, WB-->EX, WB-->MEM, MEM-->ID

main: addi x5, x0, 1
addi x6, x0, 2
add x7, x5, x6 #EX rs1 from WB, rs2 from MEM, x7 = 3
add x8, x7, x6 #EX rs1 from MEM, rs2 from WB, x8 = 5
sw x8, 0(x0) #MEM write data from WB's arith op, mem[0] = 5
lw x9, 0(x0)
sw x9, 4(x0) #MEM write data from WB's load, mem[4] = 5
```

```
addi x5, x0, 3
addi x6, x0, 3
addi x0, x0, 0
beq x5, x6, br1 #ID rs1 from MEM
addi x10, x0, 10 #should not run
br1ret: addi x11, x0, 1
jalr x0, x0, main
```

```
br1: addi x11, x0, 0x30
addi x0, x0, 0
jalr x0, x11, main #jalr x0, x0, br1ret
```

8.2 仿真测试结果

```
# r[ 5] = 0x00000001,
# r[ 6] = 0x00000002,
# r[ 7] = 0x00000003,
# r[ 8] = 0x00000005,
# dmem[0x00000000] = 0x00000005,
# r[ 9] = 0x00000005,
# dmem[0x00000004] = 0x00000005,
# r[ 5] = 0x00000003,
# r[ 6] = 0x00000003,
# r[ 0] = 0x00000000,
# r[11] = 0x00000030,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000044,
# r[11] = 0x00000001,
# r[ 0] = 0x00000038,
# r[ 5] = 0x00000001,
# r[ 6] = 0x00000002,
# r[ 7] = 0x00000003,
# r[ 8] = 0x00000005,
# dmem[0x00000000] = 0x00000005,
# r[ 9] = 0x00000005,
# dmem[0x00000004] = 0x00000005,
# r[ 5] = 0x00000003,
# r[ 6] = 0x00000003,
# r[ 0] = 0x00000000,
# r[11] = 0x00000030,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000044,
# r[11] = 0x00000001,
# r[ 0] = 0x00000038,
# r[ 5] = 0x00000001,
# r[ 6] = 0x00000002,
# r[ 7] = 0x00000003,
# r[ 8] = 0x00000005,
# dmem[0x00000000] = 0x00000005,
# r[ 9] = 0x00000005,
# dmem[0x00000004] = 0x00000005,
# r[ 5] = 0x00000003,
# r[ 6] = 0x00000003,
# r[ 0] = 0x00000000,
# r[11] = 0x00000030,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000044,
# r[11] = 0x00000001,
```

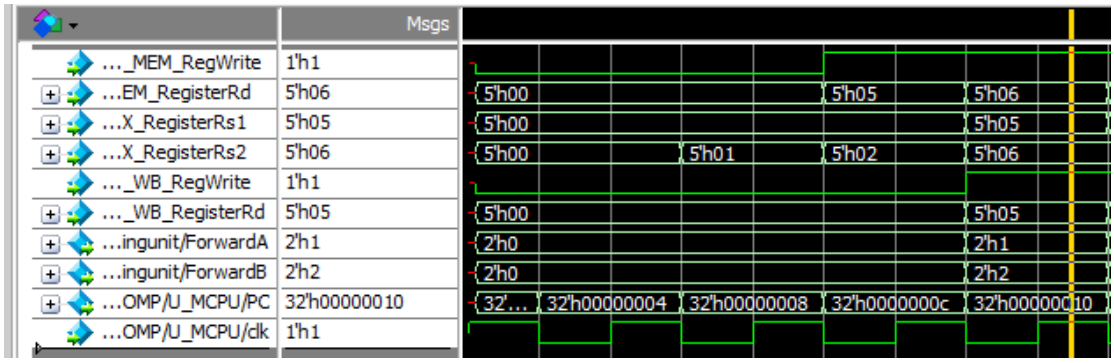
```
# r[ 6] = 0x00000002,
# r[ 7] = 0x00000003,
# r[ 8] = 0x00000005,
# dmem[0x00000000] = 0x00000005,
# r[ 9] = 0x00000005,
# dmem[0x00000004] = 0x00000005,
# r[ 5] = 0x00000003,
# r[ 6] = 0x00000003,
# r[ 0] = 0x00000000,
# r[11] = 0x00000030,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000044,
# r[11] = 0x00000001,
# r[ 0] = 0x00000038,
# r[ 5] = 0x00000001,
# r[ 6] = 0x00000002,
# r[ 7] = 0x00000003,
# r[ 8] = 0x00000005,
# dmem[0x00000000] = 0x00000005,
# r[ 9] = 0x00000005,
# dmem[0x00000004] = 0x00000005,
# r[ 5] = 0x00000003,
# r[ 6] = 0x00000003,
# r[ 0] = 0x00000000,
# r[11] = 0x00000030,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000044,
# r[11] = 0x00000001,
# r[ 0] = 0x00000038,
# r[ 5] = 0x00000001,
# r[ 6] = 0x00000002,
# r[ 7] = 0x00000003,
# r[ 8] = 0x00000005,
# dmem[0x00000000] = 0x00000005,
# r[ 9] = 0x00000005,
# dmem[0x00000004] = 0x00000005,
# r[ 5] = 0x00000003,
# r[ 6] = 0x00000003,
# r[ 0] = 0x00000000,
# r[11] = 0x00000030,
# r[ 0] = 0x00000000,
# r[ 0] = 0x00000044,
# r[11] = 0x00000001,
# r[ 0] = 0x00000038,
# r[ 5] = 0x00000001,
# r[ 6] = 0x00000002,
# r[ 7] = 0x00000003,
```

循环

数据冒险：

instr3:**add** x7, x5, x6 *#EX rs1 from WB, rs2 from MEM, x7 = 3*

EX rs1 from WB,rs2 from MEM

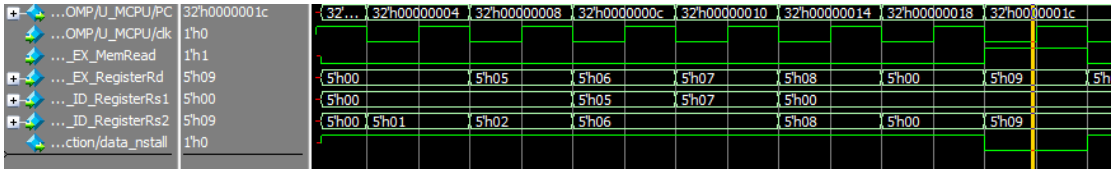


ForwardA=01 EX 阶段 rs1 来自 MEM/WB 寄存器前推

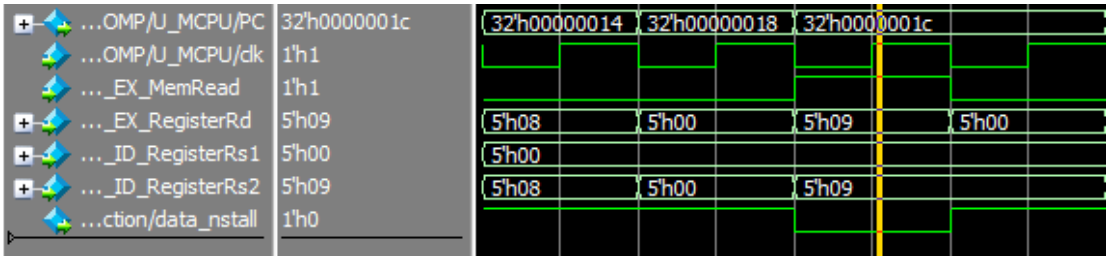
ForwardB=10 EX 阶段 rs2 来自 EX/MEM 寄存器前推

instr7:**sw** x9, 4(x0) *#MEM write data from WB's load, mem[4] = 5*

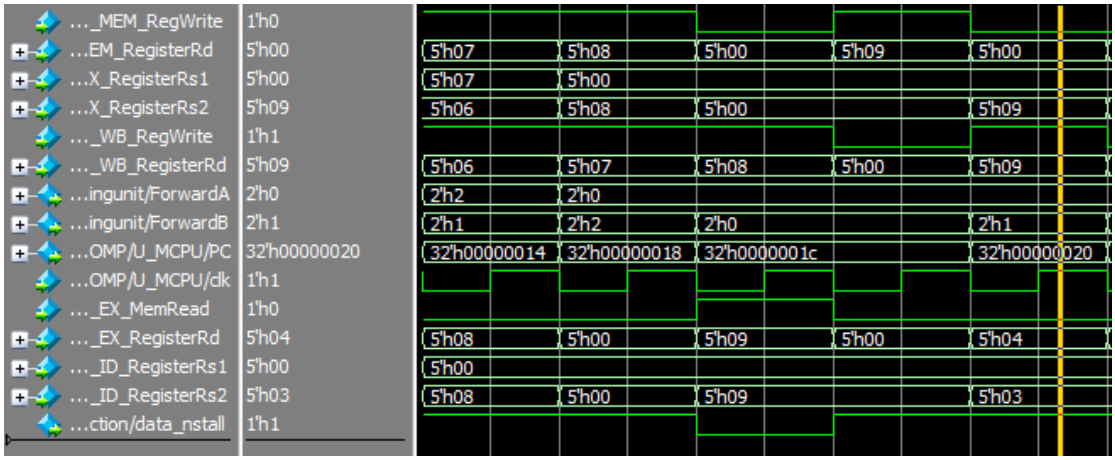
需要阻塞一个周期



instr7 id 阶段发出阻塞信号，接下来一个周期 PC 保持不变



instr exe 阶段 rs2 来自 MEM/WB 寄存器前推，ForwadB=01

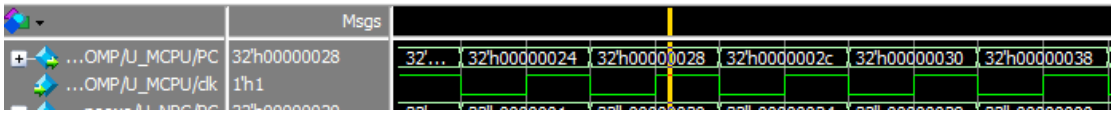


控制冒险：

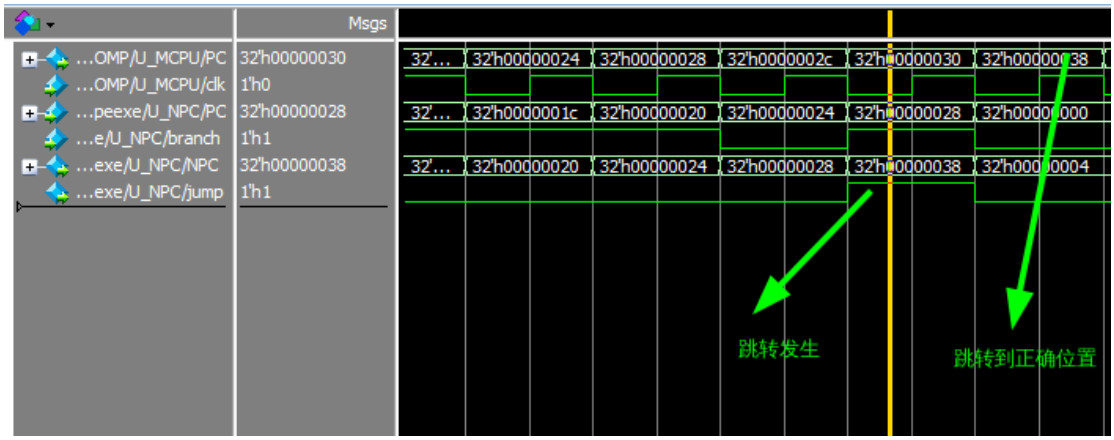
instr11:**beq** x5, x6, br1 *#ID rs1 from MEM*

静态预测，默认指令不发生，继续向下取指

instr11 在 IF 阶段时 PC 为 0x28，ID 阶段，EXE 阶段默认指令不发生向后取指 0x2c,0x30



在 exe 阶段时检测到跳转发出 jump 信号，在下一周期使 PC 进行跳转并清除 IF/ID,ID/EXE



教师评语评分

评语：_____

_____

_____

_____

_____

_____

_____

_____

评分：_____

评阅人：

年　　月　　日

（备注：对该实验报告给予优点和不足的评价，并给出百分之评分。）