中国科学技术大学计算机学院
《计算机体系结构实验报告》

实验题目：RV32I Core设计

学生姓名：柯志伟

学生学号：PB20061338

完成时间：2022年4月21日

【实验题目】

RV32I Core设计

【实验目的】

> 1.掌握五级流水线CPU的设计方法
> 2.熟悉RISC-V的指令集和数据通路，了解其设计背后的思想
> 3.进一步提高使用verilog编码和调试的能力，学会使用仿真（Simulation）调试

【实验环境】

> vivado

【实验过程】

- 阶段一&阶段二

  > 本次实验阶段一与阶段二同时完成，在阶段一主要补充各指令数据通路中缺失的部分：包含NPCGenerator,ControllerDecoder,ImmExtend,ALU,BranchDecision,在阶段二主要涉及数据相关,控制冲突的处理，具体涉及数据前递,流水线停顿，冲洗,完成Hazard模块,这两部分完成具体如下：

  - NPCGenerator

    > 该模块主要作为多路选择器确定IF段下个取指的PC,需要注意的是,各种数据源的选择有优先度,具体表现在:jalr_target,br_target>jal_target>pc+4

```
`timescale 1ns / 1ps

module NPC_Generator(
    input wire [31:0] PC, jal_target, jalr_target, br_target,
    input wire jal, jalr, br,
    output reg [31:0] NPC
    );

    always@(*)begin
        NPC = br ? br_target :
                        jalr ? jalr_target :
                                jal ? jal_target :
                                    PC;
    end

endmodule
```

  - ControllerDecoder

此模块具体根据不同指令确定各阶段的控制信号

```
`timescale 1ns / 1ps
`include "Parameters.v"
module ControllerDecoder(
    input wire [31:0] inst,
    output reg jal,
    output reg jalr,
    output reg op1_src, op2_src,
    output reg [3:0] ALU_func,
    output reg [2:0] br_type,
    output reg load_npc,
    output reg wb_select,
    output reg [2:0] load_type,
    output reg reg_write_en,
    output reg [3:0] cache_write_en,
    output reg [2:0] imm_type,
    // CSR signals
    output reg CSR_write_en,
    output reg CSR_zimm_or_reg
    );

    // TODO: Complete this module

    wire [6:0] opcode, funct7;
    wire [2:0] funct3;

    assign opcode = inst[6:0];
    assign funct7 = inst[31:25];
    assign funct3 = inst[14:12];

    always @ (*)
    begin
        if (opcode == `U_LUI)
        begin
            jal = 0;
            jalr = 0;
            op1_src = 0;
            op2_src = 1;
            ALU_func = `LUI;
            br_type = 0;
            load_npc = 0;
            wb_select = 0;
            load_type = 0;
            reg_write_en = 1;
            cache_write_en = 0;
            imm_type = `UTYPE;
            CSR_write_en = 0;
            CSR_zimm_or_reg = 0;
```

```
        end
        else if(opcode == `U_AUIPC)
        begin
            jal = 0;
            jalr = 0;
            op1_src = 1;
            op2_src = 1;
            ALU_func = `ADD;
            br_type = 0;
            load_npc = 0;
            wb_select = 0;
            load_type = 0;
            reg_write_en = 1;
            cache_write_en = 0;
            imm_type = `UTYPE;
            CSR_write_en = 0;
            CSR_zimm_or_reg = 0;
        end
        else if (opcode == `J_JAL)
        begin
            jal = 1;
            jalr = 0;
            op1_src = 0;
            op2_src = 0;
            ALU_func = 0;
            br_type = 0;
            load_npc = 1;
            wb_select = 0;
            load_type = 0;
            reg_write_en = 1;
            cache_write_en = 0;
            imm_type = `JTYPE;
            CSR_write_en = 0;
            CSR_zimm_or_reg = 0;
        end
        else if(opcode == `J_JALR)
        begin
            jal = 0;
            jalr = 1;
            op1_src = 0;
            op2_src = 1;
            ALU_func = `ADD;
            br_type = 0;
            load_npc = 1;
            wb_select = 0;
            load_type = 0;
            reg_write_en = 1;
            cache_write_en = 0;
            imm_type = `ITYPE;
```

```
            CSR_write_en = 0;
            CSR_zimm_or_reg = 0;
        end
        else if(opcode == `B_TYPE)
        begin
            jal = 0;
            jalr = 0;
            op1_src = 0;
            op2_src = 0;
            ALU_func = `ADD;
            load_npc = 0;
            wb_select = 0;
            load_type = 0;
            reg_write_en = 0;
            cache_write_en = 0;
            imm_type = `BTYPE;
            CSR_write_en = 0;
            CSR_zimm_or_reg = 0;
            if(funct3 == `B_BEQ)
            begin
                br_type = `BEQ;
            end
            else if(funct3 == `B_BNE )
            begin
                br_type = `BNE;
            end
            else if(funct3 == `B_BLT)
            begin
                br_type = `BLT;
            end
            else if(funct3 == `B_BLTU)
            begin
                br_type = `BLTU;
            end
            else if(funct3 == `B_BGE)
            begin
                br_type = `BGE;
            end
            else if(funct3 == `B_BGEU)
            begin
                br_type = `BGEU;
            end
            else begin
                reg_write_en = 0;
                load_type = `NOREGWRITE;
            end
        end
        else if(opcode == `I_LOAD)
        begin
```

```
            jal = 0;
            jalr = 0;
            op1_src = 0;
            op2_src = 1;
            ALU_func = `ADD;
            br_type = 0;
            load_npc = 0;
            wb_select = 1;
            reg_write_en = 1;
            cache_write_en = 0;
            imm_type = `ITYPE;
            CSR_write_en = 0;
            CSR_zimm_or_reg = 0;
            if(funct3 == `I_LB)
            begin
                load_type = `LB;
            end
            else if(funct3 == `I_LH)
            begin
                load_type = `LH;
            end
            else if(funct3 == `I_LW)
            begin
                load_type = `LW;
            end
            else if(funct3 == `I_LBU)
            begin
                load_type = `LBU;
            end
            else if(funct3 == `I_LHU)
            begin
                load_type = `LHU;
            end
            else
            begin
                reg_write_en = 0;
                load_type = `NOREGWRITE;
            end

        end
        else if(opcode == `I_ARI)
        begin
            jal = 0;
            jalr = 0;
            op1_src = 0;
            op2_src = 1;
            br_type = 0;
            load_npc = 0;
            wb_select = 0;
```

```
            load_type = 0;
            reg_write_en = 1;
            cache_write_en = 0;
            imm_type = `ITYPE;
            CSR_write_en = 0;
            CSR_zimm_or_reg = 0;
            if(funct3 == `I_ADDI)
            begin
                ALU_func = `ADD;
            end
            else if(funct3 == `I_SLTI)
            begin
                ALU_func = `SLT;
            end
            else if(funct3 == `I_SLTIU)
            begin
                ALU_func = `SLTU;
            end
            else if(funct3 == `I_XORI)
            begin
                ALU_func = `XOR;
            end
            else if(funct3 == `I_ORI)
            begin
                ALU_func = `OR;
            end
            else if(funct3 == `I_ANDI)
            begin
                ALU_func = `AND;
            end
            else if(funct3 == `I_SLLI)
            begin
                ALU_func = `SLL;
            end
            else if(funct3 == `I_SR && funct7 == `I_SRAI)
            begin
                ALU_func = `SRA;
            end
            else if(funct3 == `I_SR && funct7 == `I_SRLI)
            begin
                ALU_func = `SRL;
            end
            else
            begin
                reg_write_en = 0; // to look for
                load_type = `NOREGWRITE;
            end
        end
        else if(opcode == `S_TYPE)
```

```
        begin
            jal = 0;
            jalr = 0;
            op1_src = 0;
            op2_src = 1;
            ALU_func = `ADD;
            br_type = 0;
            load_npc = 0;
            wb_select = 0;
            load_type = 0;
            reg_write_en = 0;
            imm_type = `STYPE;
            CSR_write_en = 0;
            CSR_zimm_or_reg = 0;
            if(funct3 == `S_SB)
            begin
                cache_write_en = 4'b0001;
            end
            else if(funct3 == `S_SH)
            begin
                cache_write_en = 4'b0011;
            end
            else if(funct3 == `S_SW)
            begin
                cache_write_en = 4'b1111;
            end
            else
            begin
                cache_write_en = 0;
            end
        end
        else if(opcode == `R_TYPE)
        begin
            jal = 0;
            jalr = 0;
            op1_src = 0;
            op2_src = 0;
            br_type = 0;
            load_npc = 0;
            wb_select = 0;
            load_type = 0;
            reg_write_en = 1;
            cache_write_en = 0;
            imm_type = `RTYPE;
            CSR_write_en = 0;
            CSR_zimm_or_reg = 0;
            if(funct3 == `R_AS && funct7 == `R_ADD)
            begin
                ALU_func = `ADD;
```

```
            end
            else if(funct3 == `R_AS && funct7 == `R_SUB)
            begin
                ALU_func = `SUB;
            end
            else if(funct3 == `R_SLL)
            begin
                ALU_func = `SLL;
            end
            else if(funct3 == `R_SLT)
            begin
                ALU_func = `SLT;
            end
            else if(funct3 == `R_SLTU)
            begin
                ALU_func = `SLTU;
            end
            else if(funct3 == `R_XOR)
            begin
                ALU_func = `XOR;
            end
            else if(funct3 == `R_SR && funct7 == `R_SRL)
            begin
                ALU_func = `SRL;
            end
            else if(funct3 == `R_SR && funct7 == `R_SRA)
            begin
                ALU_func = `SRA;
            end
            else if(funct3 == `R_OR)
            begin
                ALU_func = `OR;
            end
            else if(funct3 == `R_AND)
            begin
                ALU_func = `AND;
            end
            else
            begin
                reg_write_en = 0;
                load_type = `NOREGWRITE;
            end
        end
        else if(opcode == `I_CSR)
        begin
            jal = 0;
            jalr = 0;
            op1_src = 0;
            op2_src = 0;
```

```
            load_npc = 0;
            wb_select = 0;
            CSR_write_en = 1;
            load_type = 0;
            reg_write_en = 1;
            cache_write_en = 0;
            CSR_write_en = 1;
            br_type = 0;
            imm_type = `RTYPE;
            if(funct3 == `I_CSRRC)begin
                ALU_func = `NAND;
                CSR_zimm_or_reg = 0;
            end
            else if(funct3 == `I_CSRRCI)begin
                ALU_func = `NAND;
                CSR_zimm_or_reg = 1;
            end
            else if(funct3 == `I_CSRRS)begin
                ALU_func = `OR;
                CSR_zimm_or_reg = 0;
            end
            else if(funct3 == `I_CSRRSI)begin
                ALU_func = `OR;
                CSR_zimm_or_reg = 1;
            end
            else if(funct3 == `I_CSRRW)begin
                ALU_func = `OP1;
                CSR_zimm_or_reg = 0;
            end
            else if(funct3 == `I_CSRRWI)begin
                ALU_func = `OP1;
                CSR_zimm_or_reg = 1;
            end
            else
            begin
                reg_write_en = 0;
                load_type = `NOREGWRITE;
            end
        end
        else
        begin
            jal = 0;
            jalr = 0;
            op1_src = 0;
            op2_src = 0;
            ALU_func = 0;
            br_type = 0;
            load_npc = 0;
            wb_select = 0;
```

```
            load_type = 0;
            reg_write_en = 0;
            cache_write_en = 0;
            imm_type = 0;
            CSR_write_en = 0;
            CSR_zimm_or_reg = 0;

        end

    end

endmodule
```

- ImmExtend

根据输入的控制信号分别产生RV32I的RTYPE，ITYPE，STYPE，BTYPE，UTYPE，JTYPE中具体一种立即数作为输出

```
`include "Parameters.v"
module ImmExtend(
    input wire [31:7] inst,
    input wire [2:0] imm_type,
    output reg [31:0] imm
    );

    always@(*)
    begin
        case(imm_type)
            `ITYPE: imm = {{21{inst[31]}}, inst[30:20]};
            // TODO: complete left part
            // Parameters.v defines all immediate type

            /* FIXME: Write your code here... */
            `RTYPE: imm = 32'b0;
            `STYPE: imm = {{21{inst[31]}},
inst[30:25],inst[11:8],inst[7]};
            `BTYPE: imm = {{20{inst[31]}},
inst[7],inst[30:25],inst[11:8],1'b0};
            `UTYPE: imm = {inst[31],
inst[30:20],inst[19:12],12'b0};
            `JTYPE: imm = {{12{inst[31]}},
inst[19:12],inst[20],inst[30:25],inst[24:21],1'b0};
            default: imm = 32'b0;
        endcase
    end
endmodule
```

- ALU

> 算术逻辑单元,实现指令需要的各种运算,根据输入的控制信号对输入的两个源操作数进行运算

```
`include "Parameters.v"
module ALU(
    input wire [31:0] op1,
    input wire [31:0] op2,
    input wire [3:0] ALU_func,
    output reg [31:0] ALU_out
    );

    // TODO: Complete this module


    always @ (*)
    begin
        case(ALU_func)
            `ADD: ALU_out = op1 + op2;
            `SLL: ALU_out = op1 << op2[4:0];
            `SLTU: ALU_out = (op1 < op2) ? 32'd1 : 32'd0;
            `LUI: ALU_out = op2;
            `SRL: ALU_out = op1 >> op2[4:0];
            `SRA: ALU_out = $signed(op1) >>> op2[4:0];
            `SUB: ALU_out = op1 - op2;
            `XOR: ALU_out = op1 ^ op2;
            `OR:  ALU_out = op1 | op2;
            `AND: ALU_out = op1 & op2;
            `SLT: ALU_out = ($signed(op1) < $signed(op2)) ?
32'd1 : 32'd0;
            `OP1: ALU_out = op1;
            `OP2: ALU_out = op2;
            `NAND: ALU_out = (~op1 & op2);
            /* FIXME: Write your code here... */
            default: ALU_out = 32'b0;
        endcase
    end
endmodule
```

- BranchDecision

> 根据输入的两个源操作数的大小关系以及控制信号确定跳转信号是否有效

```
`include "Parameters.v"
```

```
module BranchDecision(
    input wire [31:0] reg1, reg2,
    input wire [2:0] br_type,
    output reg br
    );

    // TODO: Complete this module


    always @ (*)
    begin
        case(br_type)
            `NOBRANCH: br = 0;
            `BEQ: br = (reg1 == reg2) ? 1 : 0;
            `BLTU: br = (reg1 < reg2) ? 1 : 0;
            `BNE: br = (reg1 != reg2) ? 1 : 0;
            `BLT: br = ($signed(reg1) < $signed(reg2)) ? 1 : 0;
            `BLTU:  br = (reg1 < reg2) ? 1 : 0;
            `BGE: br = ($signed(reg1) >= $signed(reg2)) ? 1 :
0;
            `BGEU: br = (reg1 >= reg2) ? 1 : 0;

            /* FIXME: Write your code here... */
            default: br = 0;
        endcase
    end

endmodule
```

- Hazard

该模块主要处理数据相关以及控制冒险
根据ALU的两个源操作数的寄存器号是否与MEM，WB阶段有效的写回寄存器号相同来确定op1_sel,op2_sel两个前递控制信号
根据是否执行跳转，以及load-to-use相关来确定各阶段流水线寄存器是否需要stall或flush

```
`include "Parameters.v"
module HarzardUnit(
    input wire rst,
    input wire [4:0] reg1_srcD, reg2_srcD, reg1_srcE,
reg2_srcE, reg_dstE, reg_dstM, reg_dstW,
    input wire br, jalr, jal,
    input wire wb_select,
    input wire reg_write_en_MEM,
    input wire reg_write_en_WB,
```

```
    output reg flushF, bubbleF, flushD, bubbleD, flushE,
bubbleE, flushM, bubbleM, flushW, bubbleW,
    output reg [1:0] op1_sel, op2_sel
    );

    // TODO: Complete this module


    // generate op1_sel
    always @ (*)
    begin
        if (reg1_srcE == reg_dstM && reg_write_en_MEM == 1 &&
reg1_srcE != 0)
        begin
            // mem to ex forwarding, mem forwarding first
            op1_sel = 2'b01;
        end
        else if (reg1_srcE == reg_dstW && reg_write_en_WB == 1
&& reg1_srcE != 0)
        begin
            // wb to ex forwarding
            op1_sel = 2'b10;
        end
        else
        begin
            op1_sel = 2'b00;
        end
    end

    // generate op2_sel
    always @ (*)
    begin
        if (reg2_srcE == reg_dstM && reg_write_en_MEM == 1 &&
reg2_srcE != 0)
        begin
            // mem to ex forwarding, mem forwarding first
            op2_sel = 2'b01;
        end
        else if (reg2_srcE == reg_dstW && reg_write_en_WB == 1
&& reg2_srcE != 0)
        begin
            // wb to ex forwarding
            op2_sel = 2'b10;
        end
        else
        begin
            op2_sel = 2'b00;
        end
    end
```

```
    // generate bubbleM and flushM
    always @ (*)
    begin
        if (rst)
        begin
            bubbleM = 0;
            flushM = 1;
        end
        else
        begin
            bubbleM = 0;
            flushM = 0;
        end
    end

    // generate bubbleF and flushF
    always @ (*)
    begin
        if (rst)
        begin
            bubbleF = 0;
            flushF = 1;
        end
        else
        begin
            flushF = 0;
            if(((reg1_srcD == reg_dstE) | (reg2_srcD ==
reg_dstE))&& wb_select)begin
                bubbleF = 1;
            end
            else begin
                bubbleF = 0;
            end

        end
    end



    // generate bubbleD and flushD
    always @ (*)
    begin
        if (rst)
        begin
            bubbleD = 0;
            flushD = 1;
        end
        else
```

```
        begin
            if(br| jalr| jal) begin
                flushD = 1;
            end
            else begin
                flushD = 0;
            end
            if(((reg1_srcD == reg_dstE) | (reg2_srcD ==
reg_dstE))&& wb_select )begin
                bubbleD = 1;
            end
            else begin
                bubbleD = 0;
            end

        end
    end


    // generate bubbleE and flushE
    always @ (*)
    begin
        if (rst)
        begin
            bubbleE = 0;
            flushE = 1;
        end
        else if(br |jalr)
        begin
            bubbleE = 0;
            flushE = 1;
        end
        else if((reg1_srcD == reg_dstE || reg2_srcD ==
reg_dstE) && wb_select)
        begin
            bubbleE = 0;
            flushE = 1;
        end
        else
        begin
            bubbleE = 0;
            flushE = 0;
        end


    end
```

```
    // generate bubbleW and flushW
    always @ (*)
    begin
        if (rst)
        begin
            bubbleW = 0;
            flushW = 1;
        end
        else
        begin
            bubbleW = 0;
            flushW = 0;

        end
    end

    /* FIXME: Write your code here... */

endmodule
```

- 阶段三

> 此阶段主要完成CSR相关指令的数据通路,包括:CSR_EX,ControllerDecoder,CSR_Regfile,具体如下:

  - CSR_EX

> CSR指令的流水线寄存器，包含CSR指令的译码信息

```
module CSR_EX(
    input wire clk, bubbleE, flushE,
    input wire [11:0] CSR_addr_ID,
    input wire [31:0] CSR_zimm_ID,
    input wire CSR_zimm_or_reg_ID,
    input wire CSR_write_en_ID,
    output reg [11:0] CSR_addr_EX,
    output reg [31:0] CSR_zimm_EX,
    output reg CSR_zimm_or_reg_EX,
    output reg CSR_write_en_EX
    );

    // TODO: Complete this module
        always@(posedge clk)
        begin
            if(!bubbleE)
            begin
                if(flushE)
```

```
                begin
                    CSR_addr_EX <= 12'h0;
                    CSR_zimm_EX <= 31'h0;
                    CSR_zimm_or_reg_EX <= 0;
                    CSR_write_en_EX <= 0;
                end
                else begin
                    CSR_addr_EX <= CSR_addr_ID;
                    CSR_zimm_EX <= CSR_zimm_ID;
                    CSR_zimm_or_reg_EX <= CSR_zimm_or_reg_ID;
                    CSR_write_en_EX <= CSR_write_en_ID;
                end
            end
        end
    /* FIXME: Write your code here... */

endmodule
```

- ControllerDecoder

> 补充CSR指令的译码信息，与其余指令相较，需额外处理CSR_zimm_or_reg与CSR_write_en

- CSR_Regfile

> 此模块为CSR的寄存器,具体的实现逻辑与通用寄存器相同，但需注意的是此寄存器在时钟上升沿写入,需实现为读优先，通用寄存器在时钟的下降沿写入，实现为写优先

```
`timescale 1ns / 1ps
// 实验要求
    // 补全模块（阶段三）

module CSR_Regfile(
    input wire clk,
    input wire rst,
    input wire CSR_write_en,
    input wire [11:0] CSR_write_addr,
    input wire [11:0] CSR_read_addr,
    input wire [31:0] CSR_data_write,
    output wire [31:0] CSR_data_read
    );

    reg [31:0] csr[4095:0];

    integer i;

    // TODO: Complete this module
```

```
    /* FIXME: Write your code here... */
    always @(posedge clk or posedge rst) begin
        if(rst)
            for(i=1;i<4096;i=i+1)
                csr[i] <= 32'h0;
        else if(CSR_write_en)
            csr[CSR_write_addr] = CSR_data_write;
    end

    assign CSR_data_read = csr[CSR_read_addr];

endmodule
```

- ALU

```
改写NAND指令以完成csrrc指令功能
```

【总结与思考】

- 问题
  - 在实现hazard模块时,在处理load-to-use相关，相关stall，flush信号出错

```
处理load-to-use相关，需将if，id阶段流水段寄存器停顿一个周期，同时exe阶段产生一个bubble
```

  - 在实现CSR寄存器时,采用通用寄存器相同的实现逻辑，在时钟下降沿写入,发现出错

```
csr指令中涉及在exe阶段交换CSR寄存器与通用寄存器的值，这就要求CSR寄存器中的值在exe执行的一个周期内保持稳定,不能在时钟下降沿写入
```

- 收获

  通过本次实验，主要有如下收获：
  - 熟悉了risicv的指令架构，熟悉risicv整数指令的数据通路与控制逻辑，掌握了risicv五段流水线cpu的设计
  - 更加深入的了解了流水线停顿的原因，以及各种相关问题的处理方法，掌握了数据前递，stall，flush技术
  - 更加熟悉利用vivado的仿真机制验证程序的正确性，提高动手能力

【改进建议】

1. 对模块进行合理的划分与整合

在计算机组成原理实验中实现的五段流水线cpu设计，对各流水段间的流水寄存器进行合理的整合，而在本次实验中各阶段的同属一个流水段的寄存器相互独立给阅读理解代码的整体逻辑带来一些麻烦

2. 译码模块太过繁琐

在译码模块中大量采用if-else嵌套,一方面带来不必要的优先级关系，另一方面导致代码模块异常庞大，在阶段一的排错阶段很大一部分出在译码阶段,可采用为指令设定编号，单独设计一个译码模块确定指令，之后根据具体指令有效信号间的与或运算利用assign赋值实现控制信号的生成