实验报告成绩:	成绩评定日期:

2022 ~ 2023 学年秋季学期

《计算机系统》必修课

课程实验报告

班级：人工智能 2001 班

组长：蒋湘

组员：季晔

报告日期：2022.1.2

目录

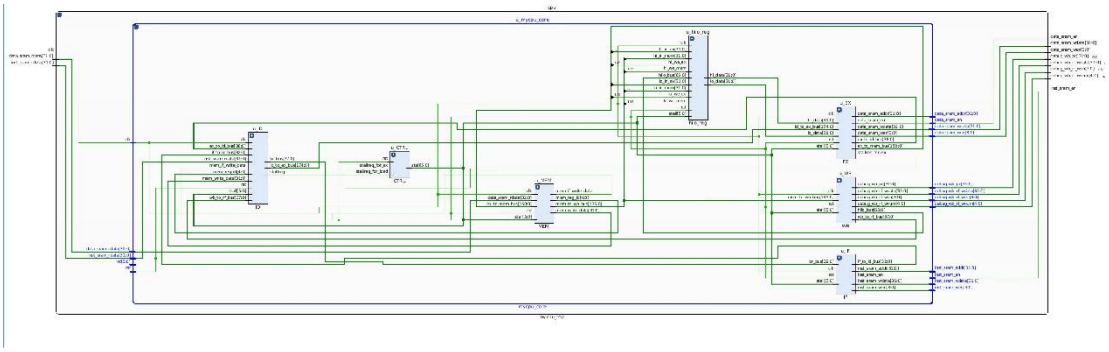
小组分工

蒋湘：完成大部分指令、数据相关添加，通过了 1-43 号测试点，参与完成了 59-64 号测试点。工作量约占 60%

季晔：完成了乘除法相关指令、部分跳转指令、访存指令的添加。通过了 44-58 号测试点，参与完成了 59-64 号测试点。工作量约占 40%

总体设计

连线图如下



完成了共计 51 条指令

```
wire    inst_ori    ,inst_lui   , inst_addiu,inst_beq ,
        inst_subu   ,inst_jr    , inst_jal  ,inst_addu,
        inst_bne    ,inst_sll   , inst_sltu ,inst_slt ,
        inst_slti   ,inst_sltiu, inst_j    ,inst_add ,
        inst_addi   ,inst_sub   , inst_and  ,inst_andi,
        inst_nor    ,inst_xori , inst_sllv ,inst_sra ,
        inst_srav   ,inst_srl   , inst_srlv ,inst_bgez,
        inst_bgtz   ,inst_blez , inst_bltz ,inst_bltzal,
        inst_bgezal ,inst_jalr , inst_or    ,
        inst_lb     ,inst_lbu   , inst_lh   ,inst_lhu ,
        inst_lw     ,inst_sb    , inst_sh   ,inst_sw  ,
        inst_div    ,inst_divu , inst_mult ,inst_multu ,
        inst_mfhi   ,inst_mflo , inst_mthi ,inst_mtlo ;
```

运行环境为 Linux 下的 vivado，使用 vscode 和 vivado 进行编写。

单个流水段具体说明

IF 段

整体功能说明

控制 pc 值，并根据 pc 值从 sram 中读取指令传到 ID 段中

端口介绍

```
module IF(
    input wire clk,
    input wire rst,
    input wire [`StallBus-1:0] stall,
    input wire [`BR_WD-1:0] br_bus,

    output wire [`IF_TO_ID_WD-1:0] if_to_id_bus,

    output wire inst_sram_en,
    output wire [3:0] inst_sram_wen,
    output wire [31:0] inst_sram_addr,
    output wire [31:0] inst_sram_wdata
);
```

stall 为控制流水线暂停的线路，从 ctrl 段接出。

br_bus 为跳转指令实现的接口，通过该接口控制要跳转到的 pc 地址

信号介绍

```
reg [31:0] pc_reg;
reg ce_reg;
```

pc_reg 储存了当前的 pc 值

ce_reg 控制是否从 sram 中读取指令

功能模块

暂无

ID 段

整体功能说明

解析当前指令，并将当前指令执行需要的资源，选择器或者数据进行预处理，从 regfile 中读出 rs,rt 对应寄存器的值，一并传入 ex 段中，并进行跳转指令的跳转部分的执行

端口介绍

```
input wire clk,
input wire rst,
// input wire flush,
input wire [`StallBus-1:0] stall,

output wire stallreq,

input wire [`IF_TO_ID_WD-1:0] if_to_id_bus,

input wire [31:0] inst_sram_rdata,

input wire [`WB_TO_RF_WD-1:0] wb_to_rf_bus,

output wire [`ID_TO_EX_WD-1:0] id_to_ex_bus,

output wire [`BR_WD-1:0] br_bus,

    //new
input wire [`EX_TO_ID_WD-1:0]  ex_to_id_bus,


input wire mem_if_write_data,
input wire [4:0] mem_reg_id,
input wire [31:0] mem_write_data
```

stallreq 接往 crtl 段，给出暂停请求

ex_to_id_bus 为 ex 段的数据回流，防止读取还未写入数据的寄存器时出错

mem_开头的三条线同 ex_to_id_bus，为 mem 段的数据回流

信号介绍

```
wire [2:0] sel_alu_src1;
wire [3:0] sel_alu_src2;
wire [11:0] alu_op;
wire [7:0] ls_op;

wire sel_rf_res;
wire [2:0] sel_rf_dst;
```

```
wire br_e;
wire [31:0] br_addr;
wire [31:0] br_addr_temp;
wire [31:0] beq_addr;
wire [31:0] jr_addr;
wire [31:0] jal_addr;
wire rs_eq_rt;
wire [31:0] pc_plus_4;
wire rs_greater_eq_zero;
wire rs_greater_zero;
```

sel_alu_src1 控制 ex 段计算所需的 1 号资源选择，sel_alu_src2 则控制 2 号资源选择

alu_op 为控制 ex 段进行哪种计算的选择器

ls_op 为控制 ex 段访存时执行哪种指令的选择器

sel_rf_res 为控制 mem 段选择哪个结果作为待写入数据的选择器

sel_rf_dst 为控制待写入数据写入哪个寄存器的选择器

br_e，br_addr 分别控制是否需要跳转以及跳转的地址

re_开头的几个信号为判断某些跳转指令是否执行的条件

功能模块

数据回流

```
assign rdata1 =          (ex_if_write_data   &   (rs == ex_reg_id))
                         ?  ex_write_data    :
                         ((mem_if_write_data &   (rs == mem_reg_id))
                         ?  mem_write_data   :
                         (wb_rf_we           &   (rs == wb_rf_waddr)
                         ?   wb_rf_wdata     :
                             temprdata1));

assign rdata2 =          (ex_if_write_data   &   (rt == ex_reg_id))
                         ?  ex_write_data    :
                         ((mem_if_write_data &   (rt == mem_reg_id))
                         ?  mem_write_data   :
                         (wb_rf_we           &   (rt == wb_rf_waddr)
                         ?   wb_rf_wdata     :
                             temprdata2));
```

当需要读取的寄存器地址和从 ex 或 mem 段回流的未写入数据要写入的地址相同时，并且此时写入控制为能，则直接将该数据赋给 rdata1 或 rdata2，避免读取还未来及写入寄存器的数据时产生错误。优先选择 ex 段的进行赋值，因为 ex 段的数据是上一个指令回流的数据，而 mem 段的数据来自上上个指令

跳转实现

```
wire br_e;
wire [31:0] br_addr;
wire [31:0] br_addr_temp;
wire [31:0] beq_addr;
wire [31:0] jr_addr;
wire [31:0] jal_addr;
wire rs_eq_rt;
wire [31:0] pc_plus_4;
wire rs_greater_eq_zero;
wire rs_greater_zero;
assign pc_plus_4 = id_pc + 32'h4;
//re是否等于rt
assign rs_eq_rt = (rdata1 == rdata2);
assign rs_greater_eq_zero = (rdata1[31] == 1'b0);
assign rs_greater_zero = ((rdata1[31] == 1'b0)&(rdata1 != 32'b0));

assign br_e = (inst_beq & rs_eq_rt)
            | inst_jr | inst_jal | inst_j | inst_jalr
            | (inst_bne     & ~rs_eq_rt)
            | (inst_bgez    & rs_greater_eq_zero)
            | (inst_bgezal  & rs_greater_eq_zero)
            | (inst_bgtz    & rs_greater_zero)
            | (inst_blez    & ~rs_greater_zero)
            | (inst_bltz    & ~rs_greater_eq_zero)
            | (inst_bltzal  & ~rs_greater_eq_zero);
assign beq_addr = pc_plus_4 + {{14{inst[15]}},inst[15:0],2'b0};
assign jr_addr = rdata1;
assign jal_addr = {pc_plus_4[31:28],instr_index,2'b0};

assign br_addr_temp = ({32{inst_beq & rs_eq_rt }} & beq_addr)
                    | ({32{inst_jr             }} & jr_addr )
                    | ({32{inst_jalr           }} & jr_addr )
                    | ({32{inst_jal |inst_j    }} & jal_addr)
                    | ({32{inst_bne     & ~rs_eq_rt }}          & beq_addr)
                    | ({32{inst_bgez    &  rs_greater_eq_zero}} & beq_addr)
                    | ({32{inst_bgezal  &  rs_greater_eq_zero}} & beq_addr)
                    | ({32{inst_bgtz    &  rs_greater_zero}}    & beq_addr)
                    | ({32{inst_blez    & ~rs_greater_zero}}    & beq_addr)
                    | ({32{inst_bltz    & ~rs_greater_eq_zero}} & beq_addr)
                    | ({32{inst_bltzal  & ~rs_greater_eq_zero}} & beq_addr);


assign br_addr = br_e ? br_addr_temp :32'b0;
assign br_bus = {
    br_e,
    br_addr
};
```

先提前计算好每个跳转指令可能需要跳往的地址，然后在跳转达成条件后，将 br_e 赋为使能，并将当前跳转指令需要跳往的地址赋值给 br_addr

EX 段

整体功能说明

进行运算操作，访存操作，将结果传入 mem 段

端口介绍

```
module EX(
  input wire clk,
  input wire rst,
  // input wire flush,
  input wire [`StallBus-1:0] stall,

  input wire [`ID_TO_EX_WD-1:0] id_to_ex_bus,

  //new
  input wire [31:0] hi_data,
  input wire [31:0] lo_data,

  output wire [`EX_TO_MEM_WD-1:0] ex_to_mem_bus,

  output wire data_sram_en,
  output wire [3:0] data_sram_wen,
  output wire [31:0] data_sram_addr,
  output wire [31:0] data_sram_wdata,

  output wire [`EX_TO_ID_WD-1:0]  ex_to_id_bus,

  //new
  output wire stallreq_for_ex
```

hi_data,lo_data 为从 hilo 寄存器读入的数据

data_开头的四条线控制写入内存和从内存读取的数据

stallreq_for_ex 为 ex 段的暂停请求，接往 crtl

信号介绍

```
wire [3:0] byte_sel;
wire [3:0] data_ram_sel;
```

byte_sel 通过访存地址的后两位确定要写入或者要读取的字符

data_ram_sel 根据 byte_sel 以及当前执行指令确定字符选择器，赋给接往 ram 的选择器，以及 mem 段

功能模块

计算

```
alu u_alu(
    .alu_control (alu_op ),
    .alu_src1    (alu_src1    ),
    .alu_src2    (alu_src2    ),
    .alu_result  (alu_result  )
);
```

```
assign and_result = alu_src1 & alu_src2;
assign or_result = alu_src1 | alu_src2;
assign nor_result = ~or_result;
assign xor_result = alu_src1 ^ alu_src2;
assign lui_result = {alu_src2[15:0], 16'b0};

wire [31:0] adder_a;
wire [31:0] adder_b;
wire        adder_cin;
wire [31:0] adder_result;
wire        adder_cout;

assign adder_a = alu_src1;
assign adder_b = (op_sub | op_slt | op_sltu) ? ~alu_src2 : alu_src2;
assign adder_cin = (op_sub | op_slt | op_sltu) ? 1'b1 : 1'b0;
assign {adder_cout, adder_result} = adder_a + adder_b + adder_cin;

assign add_sub_result = adder_result;

assign slt_result[31:1] = 31'b0;
assign slt_result[0] = (alu_src1[31] & ~alu_src2[31])
                     | (~(alu_src1[31]^alu_src2[31]) & adder_result[31]);

assign sltu_result[31:1] = 31'b0;
assign sltu_result[0] = ~adder_cout;

assign sll_result = alu_src2 << alu_src1[4:0];
assign srl_result = alu_src2 >> alu_src1[4:0];
assign sra_result = ($signed(alu_src2)) >>> alu_src1[4:0];

assign alu_result = ({32{op_add|op_sub  }} & add_sub_result)
                  | ({32{op_slt         }} & slt_result)
                  | ({32{op_sltu        }} & sltu_result)
                  | ({32{op_and         }} & and_result)
                  | ({32{op_nor         }} & nor_result)
                  | ({32{op_or          }} & or_result)
                  | ({32{op_xor         }} & xor_result)
                  | ({32{op_sll         }} & sll_result)
                  | ({32{op_srl         }} & srl_result)
                  | ({32{op_sra         }} & sra_result)
                  | ({32{op_lui         }} & lui_result);
```

将 ID 段确定的 alu_op，以及计算需要的两个资源传入 alu.v 中进行计算，在 alu.v 中则将不同的计算结果都计算出来，再根据 alu_op 选择需要的计算结果

MEM 段

整体功能说明

整理 ex 段的结果以及访存结果，将写入请求传到 WB 段

端口介绍

```
module MEM(
    input wire clk,
    input wire rst,
    // input wire flush,
    input wire [`StallBus-1:0] stall,

    input wire [`EX_TO_MEM_WD-1:0] ex_to_mem_bus,
    input wire [31:0] data_sram_rdata,

    output wire [`MEM_TO_WB_WD-1:0] mem_to_wb_bus,
    //new
    // mem to id
    output wire mem_if_write_data,
    output wire [4:0] mem_reg_id,
    output wire [31:0] mem_write_data
);
```

data_sram_rdata 为从 ram 中读取出来的数据

信号介绍

无重要信号

功能模块

得到访存结果

```
assign {inst_lw,inst_lh,inst_lhu,inst_lb,inst_lbu} = load_op;
assign b_data = data_ram_sel[3] ? data_sram_rdata[31:24]
              : data_ram_sel[2] ? data_sram_rdata[23:16]
              : data_ram_sel[1] ? data_sram_rdata[15:8]
              : data_ram_sel[0] ? data_sram_rdata[7:0]
              : 8'b0;
assign h_data = data_ram_sel[2] ? data_sram_rdata[31:16]
              : data_ram_sel[0] ? data_sram_rdata[15:0]
              : 16'b0;
assign w_data = data_sram_rdata;

assign mem_result = inst_lw ?                    w_data
                  : inst_lb ? {{24{b_data[7]}}  ,b_data}
                  : inst_lbu? {{24{1'b0}}       ,b_data}
                  : inst_lh ? {{16{h_data[15]}} ,h_data}
                  : inst_lhu? {{16{1'b0}}       ,h_data}
                  : 32'b0;
```

通过 ex 中得到的 data_ram_sel 选择器确定 mem_result

WB 段

整体功能说明

处理 mem 传来的写入请求，将请求传入 ID 段的 regfile 或写入 hilo_reg

端口介绍

```
`include "defines.vh"
module WB(
    input wire clk,
    input wire rst,
    // input wire flush,
    input wire [`StallBus-1:0] stall,

    input wire [`MEM_TO_WB_WD-1:0] mem_to_wb_bus,

    output wire [`WB_TO_RF_WD-1:0] wb_to_rf_bus,
    //new
    output wire [65:0] hilo_bus,

    output wire [31:0] debug_wb_pc,
    output wire [3:0] debug_wb_rf_wen,
    output wire [4:0] debug_wb_rf_wnum,
    output wire [31:0] debug_wb_rf_wdata
);
```

wb_to_rf_bus 为传入 refile 的写入请求合并

hilo_bus 为传入 hilo_reg 的写入请求合并

信号介绍

无重要信号

功能模块

无

实验感受

蒋湘实验感受

本次实验通过实操编写五段流水线，将课堂所学的有关流水线的知识付诸实践，加深了我对流水线知识的记忆与理解。我认为这次实验很有意义，带给了我很多收获。首先，实验巩固并加深了我对课堂所学知识的理解，也锻炼了我的实践能力，在添加指令、添加数据相关，最终通过每一个测试点的过程中，我对每个流水段该干什么、该怎么干熟悉了许多。其次，我认识到了协作的力量，例如，在过一号点时，我添加了数据相关却不能通过，询问了学长后才解决。此外，队友在实验过程中也给予了我许多帮助。最后，我觉得最重要的是我在本次实验中收获了自主学习和反思的能力，正如学长写在 Q&A 中的，先反思再问>不懂立刻问>摆烂挂机，后期遇到问题，我已经能很娴熟地在 vivado 中 debug，发现问题并解决了。实验带给我的这种反思能力，相信今后能伴随我渡过更多难关。

季晔实验感受

感受：

本次实验通过编写流水线，检验了我对计算机体系结构中流水线相关知识的掌握，加深了对相关知识的理解。在整个过程中，我遇到了很多困难，例如实验初期不理解整个代码的架构，跟不上实验课的进度，不熟悉 Vivado 的 debug……编写乘除法相关指令时，对 hilo 寄存器的用途也没有完全理解，导致效率低下等等。这反映了我对课堂内容掌握的不牢靠，以及实践能力薄弱。但在这过程中，我也获益良多。首先，这锻炼了我的自主学习思考能力，卡在某个点过不去，我会积极地翻阅书籍、观看学长的讲解，尽力地去思考、尝试。同时，通过实验我也认识到了团队协作的重要性。我实验初期进度缓慢、

不理解操作时，几乎全靠队友赶进度，并为我讲解相关操作。在我自己编写乘除法指令时，在 hilo 寄存器部分犯了难，也是请教了第六组的裴桐才得以最终完成。最后，整个过程让我对 CPU 的认识也更加全面了。

改进意见：

我认为实验布置的整体安排（11 月初做小实验让我们熟悉实验环境和语言，之后陆续安排实验课为我们答疑）比较合理。但四节实验课太过紧凑了，以我们组为例，最后两节实验课仅相隔两天且我和组长蒋湘恰好在这期间发了高烧，这两节实验课我们几乎没有进度的进展。因此我建议将实验课周期拉长，这样可以给足时间推进实验进度，也更能暴露出实验中遇到的问题，方便在实验课上汇报。

参考资料

《自己动手写 CPU》——雷思磊
《CPU 设计实战》——汪文祥　邢金璋
RUNOOB　Verilog 教程
《A03_“系统能力培养大赛”MIPS 指令系统规范_v1.01》
《A07_vivado 使用说明_v1.00》
《A09_CPU 仿真调试说明_v1.00》
《A11_Trace 比对机制使用说明_v1.00》