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

组长：常东阳 20206402

组员：赵亮 20206399 王月 20204057

报告日期：2023.1.2

目 录

一、总体设计

1.1 成员分工

常东阳：1 号点、59-64 号点以及最后的整合

赵亮：2-36 号点，44-58 号点一部分

王月：37-43 号点，44-58 号点一部分

1.2 总体设计

IF 段：

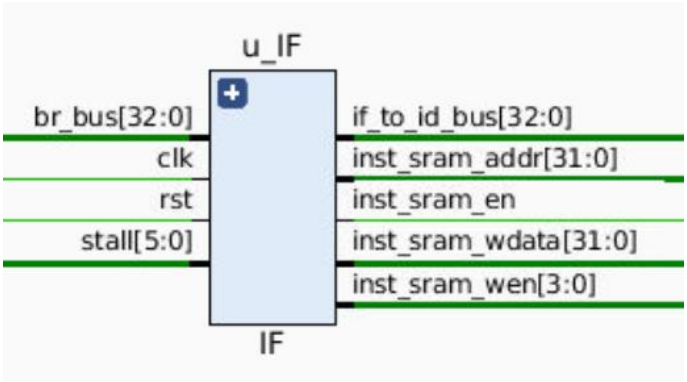

图 1.1 IF 段设计

ID 段：

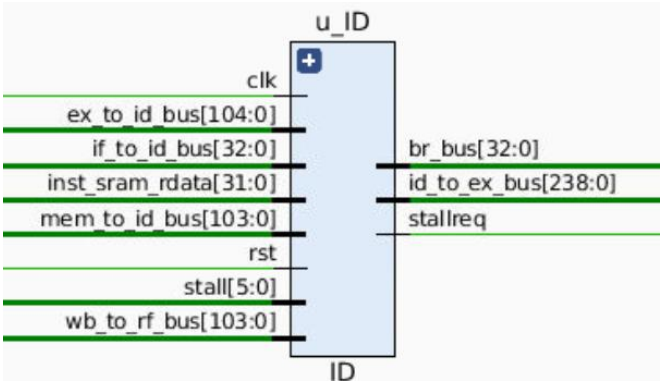

图 1.2 ID 段设计

EX 段：

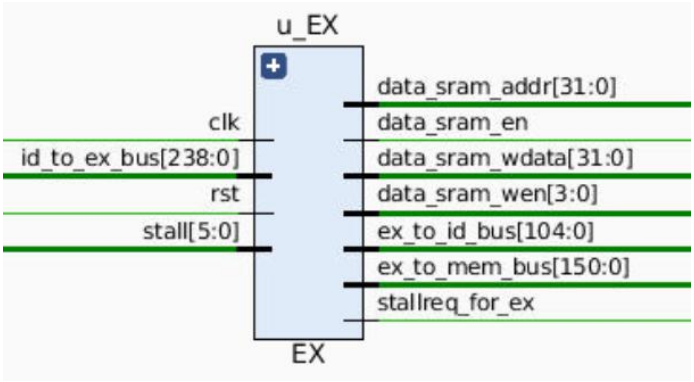

图 1.3 EX 端设计

MEM 段：

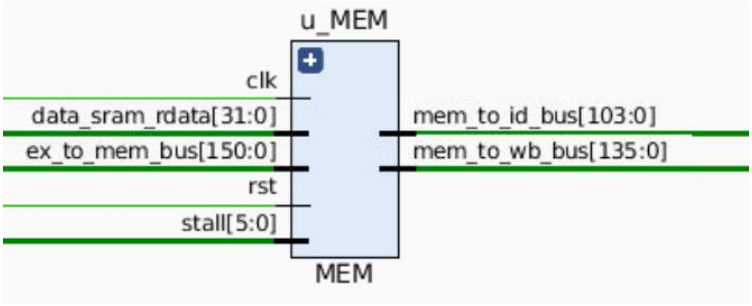

图 1.4 MEM 段设计

WB 段：

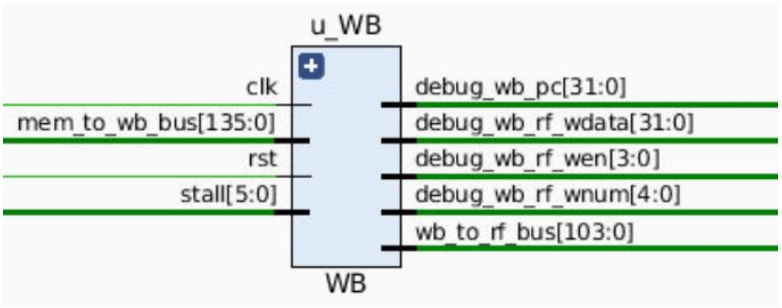

图 1.5 WB 段设计

CTRL 段：

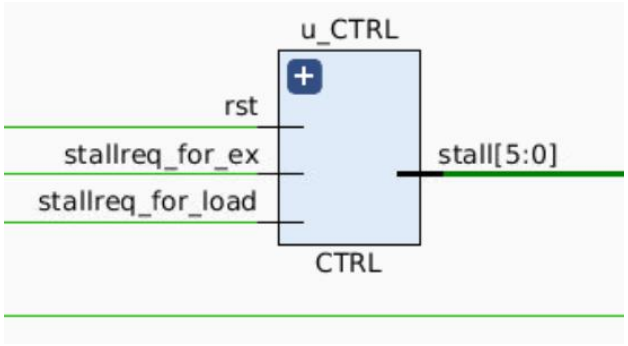

图 1.6 CTRL 段设计

总共完成了 64 条指令。

```
        [1342000 ns] Test is running, debug_wb_pc = 0xbfc4c7e8
----[1342985 ns] Number 8'd62 Functional Test Point PASS!!!
        [1352000 ns] Test is running, debug_wb_pc = 0xbfc44da4
        [1362000 ns] Test is running, debug_wb_pc = 0xbfc45cb4
----[1371175 ns] Number 8'd63 Functional Test Point PASS!!!
        [1372000 ns] Test is running, debug_wb_pc = 0xbfc2ff28
        [1382000 ns] Test is running, debug_wb_pc = 0xbfc30dd0
        [1392000 ns] Test is running, debug_wb_pc = 0xbfc31cf8
----[1397715 ns] Number 8'd64 Functional Test Point PASS!!!
--------------------------------------------------------------
```

图 1.7 测试结果

1.3 程序运行环境及使用工具

Vivado2019.2、VS code

二、详细设计

2.1 IF 段

取值阶段，取出并保存指令地址，并在下一个时钟周期传递到译码阶段。

输入：

Stall：控制流水段暂停

br_bus：控制跳转指令以及跳转目标

关键代码及解释：

```
assign next_pc = br_e ? br_addr
                 :  pc_reg + 32'h4;
```

其中，br_e 用以判断是否有跳转指令，br_addr 为跳转目标。

输出：

If_to_id_bus：从 IF 段传入 ID 段的总线，传输信号与 pc 值

Inst_sram_en：RAM 信号

Inst_sram_wen：RAM 写信号

inst_sram_addr：写进 RAM 的地址

inst_sram_wdata：写进 RAM 的数据

2.2 ID 段

译码阶段，将取到的指令进行译码，给出运算类型，子类型，源操作数 1，源操作数 2，要写入的目的寄存器地址等信息，其中运算类型指的是逻辑运算、移位运算、算术运算等，子类型指的是更加详细的运算类型，并把这些信息在下一个时钟传递到执行阶段。

输入：

Stall：控制流水段暂停

If_to_id_bus：从 IF 段传来的总线，传输信号与 pc 值

Inst_sram_rdata：从 RAM 传来的数据

Wb_to_rf_bus：从 wb 段传来的寄存器地址及数据，用来实现数据前推解决数据相关

Ex_to_id_bus：从 EX 段传来的寄存器地址及数据，用来实现数据前推解决数据相关

Mem_to_id_bus：从 MEM 段传来的寄存器地址及数据，用来实现数据前推解决数据相关

关键代码及解释：

```
assign rdata1_sum1 = (ex_rf_we&&(rs==ex_rf_waddr)) ? ex_ex_result
                   :  (mem_rf_we&&(rs==mem_rf_waddr)) ?   mem_rf_wdata
                   :  (wb_rf_we&&(rs==wb_rf_waddr)) ? wb_rf_wdata
                   :  rdata1;
assign rdata1_sum2 = (ex_rf_we&&(rt==ex_rf_waddr)) ? ex_ex_result
                   :  (mem_rf_we&&(rt==mem_rf_waddr)) ? mem_rf_wdata
                   :  (wb_rf_we&&(rt==wb_rf_waddr)) ? wb_rf_wdata
                   :  rdata2;
```

若在 ID 段读取的寄存器与 EX 段或 MEM 段或 WB 段传来的寄存器是同一个，则采用传回来的值，从而解决数据相关。

```
assign hi_o = ex_hi_we ? ex_hi
            :  mem_hi_we ? mem_hi
            :  wb_rf_hi_we ? wb_rf_hi
            :  hi_data;
```

```
assign lo_o = ex_lo_we ? ex_lo
            :  mem_lo_we ? mem_lo
            :  wb_rf_lo_we ? wb_rf_lo
            :  lo_data;
```

hilo 寄存器同理。

输出：

Id_to_ex_bus：从 ID 段传入 EX 段的总线，包括 pc 值，源操作数等重要数据

Br_bus：从 ID 段到 IF 段的总线，用来判断跳转指令及跳转目标

这一阶段还需要对跳转指令进行处理，下面给出跳转指令的实现的相关设计。

关键代码及解释：

```
assign br_e = (inst_beq & rs_eq_rt) | inst_jr | inst_jal |
              (inst_bne & (!rs_eq_rt)) | inst_j |(inst_bgez & rs_ge_z)|
              (inst_bgtz & rs_gt_z)| (inst_blez & rs_le_z)|
              (inst_bltz & rs_lt_z)     |(inst_bltzal & rs_lt_z)|
              (inst_bgezal & rs_ge_z)| inst_jalr;
assign br_addr = inst_beq ? (pc_plus_4 + {{14{inst[15]}},inst[15: 0],2'b0})
               :  inst_jr ? rdata1_sum1
               :  inst_jal ? {pc_plus_4[31: 28], inst[25: 0], 2'b00}
               :  inst_bne ? (pc_plus_4 + {{14{inst[15]}},inst[15: 0],2'b0})
               :  inst_j ? {pc_plus_4[31: 28], inst[25: 0], 2'b00}
               :  inst_bgez ? (pc_plus_4 + {{14{inst[15]}},inst[15: 0],2'b0})
               :  inst_bgtz ? (pc_plus_4 + {{14{inst[15]}},inst[15: 0],2'b0})
               :  inst_blez ? (pc_plus_4 + {{14{inst[15]}},inst[15: 0],2'b0})
               :  inst_bltz ? (pc_plus_4 + {{14{inst[15]}},inst[15: 0],2'b0})
               : inst_bltzal ? (pc_plus_4 + {{14{inst[15]}},inst[15:0],2'b0})
               : inst_bgezal ? (pc_plus_4 + {{14{inst[15]}},inst[15:0],2'b0})
               :  inst_jalr ? rdata1_sum1
               : 32'b0;
```

其中，Br_e 用来判断跳转指令，br_addr 用来判断跳转目标。

forwarding 技术并不能解决全部相关问题，一些指令导致的相关问题需要利用暂停来解决，比如 load 指令（lw，lb，lbu，lh，lhu 指令），这些指令的相关问题通过设计一个 Stallreq 来解决，下面给出其具体设计。

关键代码及解释：

```
assign pre_inst_is_load = (pre_load_op == 1'b1) ? 1'b1 :  1'b0;
assign stallreq_for_reg1 = (pre_inst_is_load == 1'b1 && (rs==ex_rf_waddr)) ?
                           1'b1 :  1'b0;
assign stallreq_for_reg2 = (pre_inst_is_load == 1'b1 && (rt==ex_rf_waddr)) ?
                           1'b1 :  1'b0;
assign stallreq = stallreq_for_reg1 | stallreq_for_reg2;
```

其中，pre_inst_is_load 是从 EX 段传来的指令，若 EX 段指令是 lw，lb，lbu，lh，lhu 之一，则为其赋 1。

stallreq_for_reg1 是判断 load 指令是否与第一个源寄存器存在数据相关。

stallreq_for_reg2 是判断 load 指令是否与第二个源寄存器存在数据相关。

stallreq 是从 ID 段传到 CTRL 段的信号，若 load 指令至少与一个源寄存器存在数据相关，则为其赋 1，再在 CTRL 段实现暂停，解决 load 相关。

2.3 EX 段

这一阶段需要根据译码阶段得来的信息进行运算，并把信息传递到访存阶段。

输入：

Stall：控制流水段暂停

Id_to_ex_bus：从 ID 段到 EX 段的总线，传输 pc 值，源操作数等重要数据

输出：

ex_to_mem_bus：从 EX 段到 MEM 段的总线，传输 pc 值，计算结果等重要数据

data_sram_en：RAM 信号

data_sram_wen：RAM 写信号

data_sram_addr：写进 RAM 的地址

data_sram_wdata：写进 RAM 的数据

关键代码及解释：

```
assign data_ram_sel = inst_sb |inst_lb |inst_lbu ? byte_sel
                    :  inst_sh |inst_lh |inst_lhu ?
                        {{2{byte_sel[2]}},{2{byte_sel[0]}}}
                    :  inst_sw |inst_lw ? 4'b1111
                    :  4'b0000;
assign data_sram_en = data_ram_en;
assign data_sram_wen = data_ram_wen & data_ram_sel;
assign data_sram_addr = ex_result;
assign data_sram_wdata = inst_sb ? {4{rf_rdata2[7：0]}}
                       :  inst_sh ? {2{rf_rdata2[15：0]}}
                       :  rf_rdata2;
```

其中，data_ram_sel 根据指令类型判断写入的字节的位数。

data_sram_wdata 根据指令类型判断写入的数据。

Ex_to_id_bus：从 EX 段到 ID 段的总线，传输寄存器相关信息，用以解决数据相关。

stallreq_for_ex：从 EX 段到 CTRL 段的线，用于除法指令的暂停。

2.4 MEM 段

访存阶段，处理访存指令。

输入：

stall：控制流水段暂停

ex_to_mem_bus：从 EX 段到 MEM 段的总线，传输 pc 值、计算结果等重要数据

data_sram_rdata：从 RAM 读取的数据

输出：

mem_to_wb_bus：从 MEM 段到 WB 段的总线，传输 pc 值、写入寄存器地址等重要数据

mem_to_id_bus：从 MEM 段到 ID 段的总线，传输寄存器相关信息，用以解决数据相关

关键代码：

```
assign mem_result = 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}
: inst_lw ?  w_data
: 32'b0;
```

其中，mem_result 是根据不同的 load 指令选择不同的数据传入到下一阶段。

2.5 WB 段

回写阶段。

输入：

Stall：控制流水段暂停

mem_to_wb_bus：从 MEM 段到 WB 段的总线，传输 pc 值，写入寄存器地址等重要数据

输出：

wb_to_rf_bus：从 WB 段到 ID 段的总线，传输寄存器等重要数据

Regfile 文件：同时实现 32 个 32 位的通用整数寄存器以及两个 32 位的特殊寄存器 lo 和 hi

输入：

Raddr1：第一个操作数的地址

Raddr2：第二个操作数的地址

We：写入是否判断

Waddr：写入的地址

Wdata：写入的类似

Hi_we：特殊寄存器 hi 写入与否判断

Hi_i：写入的内容

Lo_we：特殊寄存器 lo 写入与否判断

Lo_i：写入的内容

输出：

Rdata1：第一个操作数数据

Rdata2：第二个操作数数据

Hi_o：特殊寄存器 hi 数据

Lo_o：特殊寄存器 lo 数据

关键代码及解释：

```
always @ (posedge clk) begin
      if (rst) begin
            hi_reg <= 32'b0;
            lo_reg <= 32'b0;
      end
      else if (hi_we & lo_we) begin
            hi_reg <= hi_i;
            lo_reg <= lo_i;
      end
      else if (hi_we & ~lo_we) begin
            hi_reg <= hi_i;
      end
      else if(~hi_we & lo_we) begin
```

```
            lo_reg <= lo_i;
        end
    end
```

其中，hi_we 和 lo_we 判断是否写入 hi，lo 寄存器，hi_i 和 lo_i 是写入 hi，lo 寄存器的值。

2.6 CTRL 段

用于控制流水线的暂停。

输入：

stallreq_for_load：解决 load 指令相关的暂停

stallreq_for_ex：用于除法指令的暂停

输出：

stall：控制流水线的暂停

关键代码及解释：

```
    always @ (*) begin
        if (rst) begin
            stall = `StallBus'b0;
        end
        else if (stallreq_for_load == 1'b1)begin
            stall = 6'b000111;
        end
        else if (stallreq_for_ex == 1'b1)begin
            stall = 6'b001111;
        end
        else begin
            stall = `StallBus'b0;
        end
     end
```

根据不同指令的要求为 stall 赋相应的值，用于控制相应流水线结构的暂停。

三、总结与体会

3.1 赵亮的总结与体会

在本次实验中，我主要负责算术运算指令，逻辑运算指令的添加，通过理解 ID 段，EX 段等的结构来添加各种指令。整个过程有点麻烦但很有趣，先了解指令的含义，再想办法加入流水线中，遇到 bug 就通过仿真图查找，最后解决 bug，成功添加指令。仿真实验让我有机会将书本上的知识与实际联系起来，让我对流水线的认识更加彻底，收获很大。

3.2 常东阳的总结与体会

在本次实验中，我主要负责数据移动指令，访存指令的添加，难点在于需要添加 hilo 寄存器以及与访存指令有关的暂停。通过查阅相关书籍，设计 hilo 寄存器，并设计各种访存指令的添加，有点小麻烦但能够顺利解决。同时在实验过程中，理解各段流水线也是个有趣的过程，通过 wire，reg 等理解流水线并添加指令，让我对流水线有了更深的认识，有很大收获。

3.3 王月的总结与体会

在本次实验中，我主要负责移位指令，分支跳转指令的添加，通过理解 IF 段，ID 段，EX 段等的结构来添加各种指令。先是理解流水线的结构，再根据各指令的含义，设计相应的流水线结构从而成功添加指令。在这个过程中，仿真图发挥很大的作用，通过其来 debug，节省了很多时间。这次实验让我自己动手实现了书本上的流水线结构，对流水线结构有了更深的认识，收获颇丰。

四、参考资料

[1]“系统能力培养大赛”MIPS 指令系统规范_v1.01.pdf
[2]《自己动手做 CPU》.雷思磊