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 2002

组长：刘愉鑫

组员：张杰凯，张玉龙

报告日期：2022.12.21

目录

1.简单介绍

1.1 各人工作量

姓名	完成任务	工作量
刘愉鑫	load 相关，暂停，数据前递，一部分指令实现	45%
张杰凯	指令的实现，乘除法部分实现	35%
张玉龙	Hilo 寄存器实现，乘除法部分实现	20%

1.2 总体设计

CPU 主要由七个模块组成，分别为 IF 模块、ID 模块、EX 模块、MEM 模块、WB 模块、CTRL 模块，以及 regfile 模块。总体具有 32 个 32 位的整数寄存器，并且采用大端模式储存，具有 32bit 数据，地址总线宽度。CPU 可以通过 64 测试点。整体的 cpu 的模块连线图如下图图所示：

流水段	功能
IF	取指令，控制跳转
ID	译码，操作寄存器
EX	运算，计算内存地址
MEM	判断写回寄存器的类型并将结果传给 WB
WB	写回寄存器

1.3 实验环境

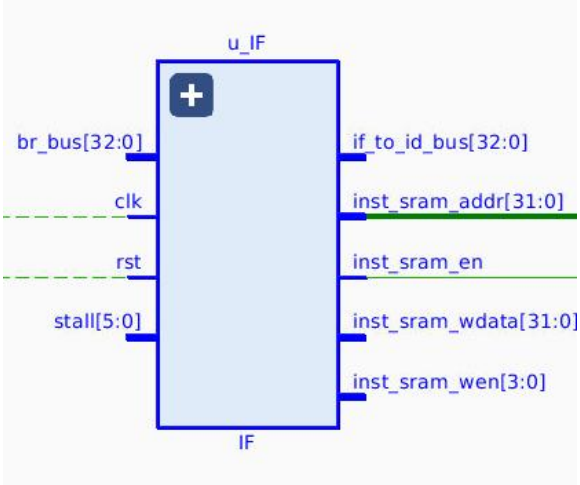
使用 Vivado2019 作为仿真运行的工具。同时使用 VS Code 协助代码编辑。使用 github 完成小组内部的协同开发和代码同步。

2. 各个流水线段说明

2.1 IF 段

整体功能：取指令，并控制指令延迟槽和跳转指令

端口与信号介绍：



接口	长度	输入/输出	作用
br_bus	33	输入	ID 段发出的控制分支跳转指令的信号
clk	1	输入	时钟信号
rst	1	输入	复位信号
stall	6	输入	暂停信号
lf_to_id_bus	33	输出	if 段发给 id 段数据
Inst_sram_addr	32	输出	指令寄存器的地址，用来寻找指令的存放的位置
inst_sram_en	1	输出	指令寄存器的读写使能信号
Inst_sram_wdata	32	输出	指令寄存器的数据
Inst_sram_wen	4	输出	指令寄存器的写使能信

详细说明：

IF 段输入时钟信号、复位信号，如果复位信号为真，就将 pc 的值置为复位的值。输入的 stall 暂停信号是由 CTRL 段发出的，其主要目的就是在流水线暂停时，IF 段通过识别 stall 来暂停指令延迟槽，使得下一条的 pc 值仍等于当前的 PC 值，使得 IF 段暂停一个时钟周期。br_bus 为 ID 段发出的跳转指令的信号，为是否跳转和要跳转的

地址的值。在 IF 段若判断到需要跳转，则把 next_pc 值调成为需要跳转的地址值。在没有暂停和跳转发生的正常的情况下，reg_pc 值为当前的 next_pc 的值，同时 next_pc 值加上 4。最后将 reg_pc 的地址发给指令内存，从指令内存中得到相应的 pc 地址对应的值并发给 ID 段。

IF 段输入时钟与复位信号，若复位信号为 1，则将 pc 复位。IF 段根据输入的 stall 信号来控制指令延迟槽，使得下一条指令的 pc 值仍等于当前的 PC 值，且使得 IF 段暂停一个时钟周期。根据 br_bus 的值，若判断需要跳转，则 next_pc = br_addr，否则 pc+4。最后将 pc_reg 的地址发给指令内存，从指令内存中得到相应的 pc 地址对应的值并发给 ID 段,以 if_to_id_bus。

2.2 ID 段

整体功能：对指令进行译码，并将译码结果传给 EX 段，同时与寄存器进行交互，实现寄存器的读写，处理数据相关。

端口与信号介绍：

接口	长度	输入/输出	作用
clk	1	输入	时钟信号
rst	1	输入	复位信号
stall	6	输入	暂停信号
ex_is_load	1	输入	判断上一条指令是否是加载指令
if_to_id_bus	33	输入	IF 段发给 ID 段的数据
inst_sram_rdata	32	输入	当前指令地址中储存的值
wb_to_rf_bus	38	输入	WB 段传给 ID 段要写入寄存器的值
ex_to_id	38	输入	EX 段传给 ID 段的数据，用来判断数据相关
mem_to_id	38	输入	MEM 段传给 ID 段的数据，用来判断数据相关
wb_to_id	38	输入	WB 段传给 ID 段的数据，用来判断数据相关
hilo_ex_to_id	66	输入	EX 传给 ID 段要写入乘除法寄存器的值
id_to_ex_bus	169	输出	ID 段传给 EX 段的数据
br_bus	33	输出	ID 段传给 IF 段的数据，用来判断下一条指令的地址
stallreq_from_id	1	输出	从 ID 段发出的暂停信号

详细说明:

正常指令的译码：

根据译码器结果激活对应指令，regfile 模块读取地址为 rs（inst[25:21]）以及地址 rt(inst[20:16]的通用寄存器,得到 rdata1 以及rdata2，并且通过判断是否发生数据相关，从而更改 rdata1 以及rdata2 的值。同时分析要执行的运算，给对应的 ALU 标识符赋值，同时将所有的 ALU 标识符组合起来成为 alu_op，并且传入 EX 。rf_we 代表写使能信号，表示该条指令是否用写入通用寄存器，sel_rf_dst[0] 为一位宽，表示该指令要将计算结果写入 rd 通用寄存器，sel_rf_dst[1] 为一位宽，表示该指令要将计算结果写入 rt 通用寄存器，sel_rf_dst[2] 为一位宽，表示该指令要将计算结果写入 31 号通用寄存器。rf_waddr 表示要该条指令的计算结果要写入的通用寄存器的地址，如果 sel_rf_dst[0]等于 1，那么 rf_waddr 将被赋值为 rd 寄存器的地址,如果 sel_rf_dst[1]等于 1,那么 rf_waddr 将被赋值为 rs 寄存器的地址,如果 sel_rf_dst[2]等于 1，那么 rf_waddr 将被赋值为 31 号寄存器的地址。
data_ram_en 为一位宽,表示该条指令是否要与内存中取值或者写入值，如果该条指令要从内存中取值或者写入值，那么它将被赋值为 1’b1,data_ram_wen 为四位宽，表示该条指令是否要写入寄存器，如果该条指令要将计算结果的第几个字节写入寄存器，那么对应位置的值设为 1。

跳转指令的译码：

br_e 为一位宽，表示该条指令是否是跳转指令，如果该指令需要跳转，那么 br_e 被赋值为 1’b1，如果该指令不需要跳转，那么 br_e 被赋值为 1’b0。rs_ge_z 为一位宽，表示是否满足 rdata1 的值大于等于 0， 如果 rdata1 的值大于等于 0，那么他被赋值为 1’b1,如果不满足，那么它将被赋值为 1’b0 ；rs_gt_z 为一位宽，表示是否满足 rdata1 的值大于 0，如果 rdata1 的值大于 0，那么他被赋值为 1’b1，如果 rdata1 的值不满足大于 0， 那么他被赋值为 1’b0；rs_le_z 为一位宽，表示是否满足 rdata1 的值小于 0，如果 rdata1 的值小于 0，那么他被赋值为 1’b1，如果 rdata1 的值不满足小于0，那么他被赋值为 1’b0；rs_lt_z 为一位宽，表示是否满足 rdata1 的值小于0,如果 rdata1 的值小于 0,那么他被赋值为 1’b1，如果 rdata1 的值不满足小于0,那么他被赋值为 1’b0;rs_eq_rt 为一位宽,表示是否满足 rdata1 是否等于radta2 的值，如果 rdata1 是否等于 radta2 的值，那么他被赋值为1’b1，如果 rdata1 的值不满足 rdata1 是否等于 radta2 的值,那么他被赋值为1’b0。br_addr 为三十二位宽，表示跳转后的地址，如果是 beq 指令，就将该分支指令对应的延迟槽指令的 pc 加上立即数 offset 左移两位并进行有符号扩展的值赋值给 bre_addr。如果是 bne 指令，就将立即数 offset 左

移2 位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算结果赋值给 bre_addr。如果是 bgez 指令，就将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 bre_addr。如果是 bgtz 指令，就将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 bre_addr。如果是 blez 指令，就将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 bre_addr。如果是 bltz 指令，就将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 bre_addr。如果是 bgezal 指令，就将立即数offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的PC 计算得到的值赋值给 bre_addr。如果是 bltzal 指令，就将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 bre_addr。如果是 j 指令，就将该分支指令对应的延迟槽指令的PC 的最高四位与立即数 inst_index(inst[26:0])左移两位后的值拼接之后赋值给 bre_addr。如果是 jal 指令，就将该分支指令对应的延迟槽指令的 PC 的最高四位与立即数 inst_index(inst[26:0])左移两位后的值拼接之后赋值给bre_addr。

设置操作数来源：

sel_alu_src1 为三位宽，表示对于第一个操作数有三种来源，第一种：第一个操作数的值为 rs 寄存器的值，第二种：当前的PC 值，第三种：立即数零扩展。sel_alu_src2 为四位宽，表示对于第二个操作数有四种来源，第一种：第二个操作数的值为 rs 寄存器的值，第二种 rs 的值为立即数符号扩展，第三种，将第二个操作数复制为 32'b8，第四种，第二个操作数的值为立即数零扩展。

暂停和气泡：

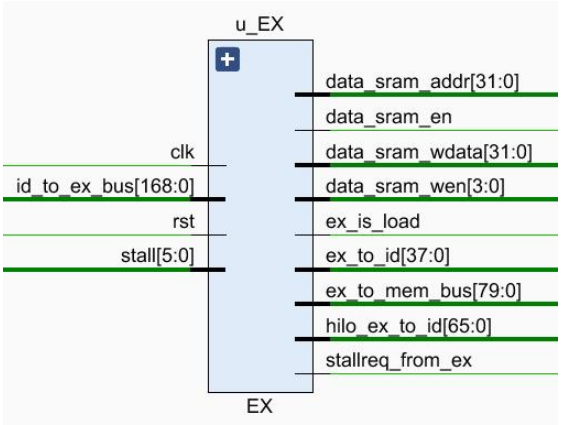
ID 段会收到来自 CTRL 模块的stall 值，而stall 的值就是用来控制流水线的暂停的。当 ID 段判断到 stall 的值的对应 ID 段的部分为`NoStop 时，即意味着没有流水线暂停，则在此部分将 IF 段传给 ID 段的if_to_id_bus 正常地赋值给 if_to_id_bus_r，然后就可以进行接下来的正常的译码、取操作数的部分，流水线正常运行。但是如果判断到 stall 的值的对应的 ID 段的部分是

`Stop，即此刻发生了访存冲突，现在需要读取的寄存器中的值还没有获得，还需要在下一个周期才可以从内存中读取出来，无法通过数据前递解决，则现在需要对流水线的 ID 段进行暂停一个周期，在下个周期获得到需要读取的值后再发给 ID 段。当判断到暂停后就将 if_to_id_bus_r 置为0，本周期停止，下个周期再恢复正常。

2.3 EX 段

整体功能：计算 ALU 的结果，根据当前指令，确定即将要写入内存的数据以及地址，或者下一步是否要从内存中读取值。

端口与信号介绍：



接口	长度	输入/输出	作用
clk	1	输入	时钟信号
rst	1	输入	复位信号
stal	6	输入	暂停信号

id_to_ex_bus	169	输入	ID 段传给 EX 段的数据
ex_to_mem_bus	80	输出	EX 段传给 MEM 短的数据
data_sram_en	1	输出	内存数据的读写使能信号
data_sram_wen	4	输出	内存数据的写使能信号
data_sram_addr	32	输出	内存数据存放的地址
ex_to_id	38	输出	EX 段传给 ID 段的数据
data_sram_wdata	32	输出	要写入内存的数据
stallreq_from_ex	1	输出	EX 发出的是否暂停的信号
ex_is_load	1	输出	EX 段发给 ID 段的数据，用来判断上一条指令是否是储存指令
hilo_ex_to_id	66	输出	EX 段将乘除法器的结果发给 ID 段的 regfile 模块

详细说明：

```
assign {
    data_ram_readen,//168:165
    inst_mthi,      //164
    inst_mtlo,      //163
    inst_multu,     //162
    inst_mult,      //161
    inst_divu,      //160
    inst_div,       //159
    ex_pc,          // 148:117
    inst,           // 116:85
    alu_op,         // 84:83
    sel_alu_src1,   // 82:80
    sel_alu_src2,   // 79:76
    data_ram_en,    // 75
    data_ram_wen,   // 74:71
    rf_we,          // 70
    rf_waddr,       // 69:65
    sel_rf_res,     // 64
    rf_rdata1,         // 63:32
    rf_rdata2          // 31:0
} = id_to_ex_bus_r;
```

接收 id 段发送的值，并赋值给相应变量。

```
assign ex_is_load = (inst[31:26] == 6'b10_0011) ? 1'b1 : 1'b0;
```

根据 inst 的值，修改 ex_is_load。例如判断当前指令是否为 LW 指令，即 inst[31:26] 是否 6'b10_0011，如果是那么就将 ex_is_load 赋值为 1；

计算 ALU 结果：

定义立即数符号扩展的变量并将其赋值为{{16{inst[15]}},inst[15:0]}，定义立即数零扩展的变量并将其赋值为{16'b0, inst[15:0]}，定义 s a 零扩展的变量并将其赋值为{27'b0, inst

[10:6]}计算参与ＡＬＵ运算的操作数，根据 ID 段，传过来的操作数来源的方式，也就是 sel_alu_src1，sel_alu_src2 中的值，然后给 alu_src1，alu_src2 赋值成对应的值。调用 ALU 模块，将参与 ALU 计算的两个操作数，已经 ALU 计算的方式传给 ALU 的接口，然后从 ALU 模块中得到 ALU 计算的结果，将其赋值给 ex_result。

读写内存：

将内存读写使能设置为相应的值。判断当前指令是否是 sb 指令，即 data_ram_readen 是否为 4'b0101，如果是，判断要写入内存的值，即判断要写入内存地址的后两位的值为多少，如果 ex_result[1:0] == 2'b00，那么就将内存写使能赋值为 4'b0001，表示要写入的是第一个字节，如果 ex_result[1:0] == 2'b01，那么就将内存写使能赋值为 4'b0010，表示要写入的是第二个字节，如果 ex_result[1:0] == 2'b10，那么就将内存写使能赋值为 4'b0100，表示要写入的是第三个字节，如果 ex_result[1:0] == 2'b11，那么就将内存写使能赋值为 4'b1000，表示要写入的是第四个字节；如果不是 sb 指令，那么判断是否是 sh 指令，即 data_ram_readen 是否 4'b0111，如果是，判断要写入内存的值，即判断要写入内存地址的后两位的值为多少如果 ex_result[1:0] == 2'b00，那么就将内存写使能赋值为 4'b0011，表示要写入的是第一、第二字节，如果 ex_result[1:0] == 2'b10，那么就将内存写使能赋值为 4'b1100，表示要写入的是第三、第四字节。将写内存的值赋值为当前 ALU 计算的结果。这是由于在当前情况下，写内存的值都是由参与 ALU 计算的两个操作数执行相应运算的结果。

要写入内存的数据：

判断要写的数据是什么，即判断 data_sram_wen 为多少，如果为 4'b1111，表示要将写入数据来源的四个字节全部写入内存，则把 data_sram_wdata 赋值为 rf_rdata2；如果为 4'b1111，表示要将写入数据来源的四个字节全部写入内

存，则把 data_sram_wdata 赋值为 rf_rdata2；如果为 4'b0001，暗示是 sb 指令，而 sb 指令是只写入最低字节，只是放置位置不同，表示要将写入数据来源的第一个字节放到第一个字节的位置，其他位填充 0 写入内存，则把 data_sram_wdata 赋值为{24'b0,rf_rdata2[7:0]}；如果为 4'b0010，暗示是 sb 指令，表示要将写入数据来源的第一个字节放到第二个字节的位置，其他位填充 0 写入内存，则把 data_sram_wdata 赋值为{16'b0,rf_rdata2[7:0],8'b0}；如果为 4'b0100，暗示是 sb 指令，表示要将写入数据来源的第一个字节放到第三个字节的位置，其他位填充 0 写入内存，则把 data_sram_wdata 赋值为{8'b0,rf_rdata2[7:0],16'b0}；如果为 4'b1000，暗示是 sb 指令，表示要将写入数据来源的第一个字节放到第二个字节的位置，其他位填充 0 写入内存，则把 data_sram_wdata 赋值为 rf_rdata2；如果为 4'b0011，，暗示该条指令是 sh 指令，而 sh 指令是值写入最低两个字节，只是最低来个字节放置的位置不同，暗示是 sb 指令，表示要将写入数据来源的第一、第二字节放到第一、第二个字节的位置，其他位填充 0 写入内存，则把 data_sram_wdata 赋值为{16'b0,rf_rdata2[15:0]}，如果为 4'b1100，，暗示该条指令是 sh 指令，暗示是 sb 指令，表示要将写入数据来源的第一、第二字节放到第三、第四个字节的位置，其他位填充 0 写入内存，则把 data_sram_wdata 赋值为{rf_rdata2[15:0],16'b0}。调用乘除法器，并且处理乘除法器返回的值：通过。

读内存：

将 data_sram_en 赋值为对应的值后，1 表示要可以读内存，0 表示不可以读取内存，此外将 data_sram_addr 赋值为要读内存的地址，在 EX 会获取到想要读取内存地址的数据。

发送 EX 段结果给其他段：发给 WB 段：

将内存的读使能信号，当前的 Pc 值，内存的读写使能，以及内存写使能信号，以及寄存器的写使能信号，以及寄存器要写的地址与数据。**发给 ID 段：**寄存器的写使能信号，以及寄存器要写的地址与数据，用来让 ID 段判断是否会出现相关的情况发生。还有乘除法器高位寄存器以及低位寄存器的写使能信号，以及乘除法器高位和低位要写入的数据，让 ID 段在调用 regfile 的同时，将乘除法器高位和低位的值也一并写入寄存器中，提高了 CPU 效率。

除法指令实现：

先判断是否为有符号除法或无符号除法，然后判断除法器的状态是否为空闲，除法器可用时，就把被除数和除数传入除法器，并把 stallreq_for_div 值改为`Stop 来暂停流水线。在除法结束后结果放在 div_result 中并恢复流水线

继续运行。在乘除法结束后，需要将结果存到 hilo 寄存器中。首先把 result 中的高 32 位的值和低 32 位的值放在对应的 hi、lo 的值中，然后把要写入的值和写使能信号直接发送给 ID 段。因为在 MEM 和 WB 段没有涉及到 hilo 寄存器的读写的问题，因此我们直接把 hilo 的写入的线发给了 ID 段，并没有像通用寄存器一样向后传到 WB 段再发给 ID 段。

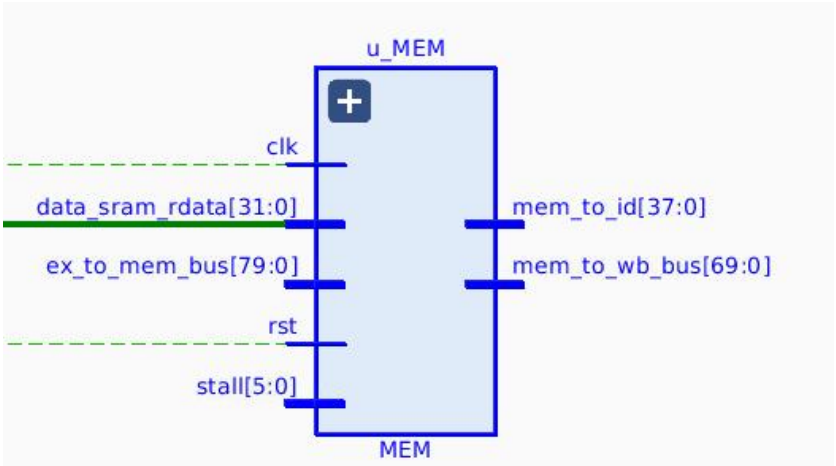
乘法指令实现：

与除法指令实现相同，先判断是否为有符号除法或无符号除法，然后判断乘法器的状态是否为空闲，乘法器可用时，就把被除数和除数传入乘法器，并把 stallreq_for_mul 值改为`Stop 来暂停流水线。在乘法结束后结果放在 div_result 中并恢复流水线继续运行。在乘除法结束后，需要将结果存到 hilo 寄存器中。首先把 result 中的高 32 位的值和低 32 位的值放在对应的 hi、lo 的值中，然后把要写入的值和写使能信号直接发送给 ID 段。因为在 MEM 和 WB 段没有涉及到 hilo 寄存器的读写的问题，因此我们直接把 hilo 的写入的线发给了 ID 段，并没有像通用寄存器一样向后传到 WB 段再发给 ID 段。

2.4 MEM 段

整体功能：读取内存中相应地址的值，根据当前指令，确定要写入寄存器的值。

端口与信号介绍：



接口	长度	输入/输出	作用
clk	1	输入	时钟信号
rst	1	输入	复位信号
stall	6	输入	暂停信号
data_sram_rdata	32	输入	从内存中读出来要写入寄存器的值
ex_to_mem_bus	80	输入	EX 传给 MEM 段的数据
mem_to_id	38	输出	MEM 传给 ID 段的数据
mem_to_wb_bus	70	输出	MEM 传给 WB 的数据

详细说明：

```
assign {
    data_ram_readen,//79:76
    mem_pc,          // 75:44
    data_ram_en,     // 43
    data_ram_wen,    // 42:39
    sel_rf_res,      // 38
    rf_we,           // 37
    rf_waddr,        // 36:32
    ex_result        // 31:0
} =  ex_to_mem_bus_r;
```

接收从 ex 段发送的数据。其中 data_ram_readen 代表指令类别，data_ram_en 以及 data_ram_wen 告诉访存段是否要进行从内存读取值。mem_pc 是当前指令地址。rf_we，rf_waddr，ex_result 告诉访存段该指令是否要写入寄存器，以及要写入寄存器的地址，以及要写入寄存器的数据。

由于要写入寄存器的值不止由 ALU 计算的结果，也可能是在执行段到访存段访问内存之后得到的值。所以在访存阶段要进行分析。根据 data_ram_readen 的值，确定指令类型，以此确定 rf_wdata 的值。

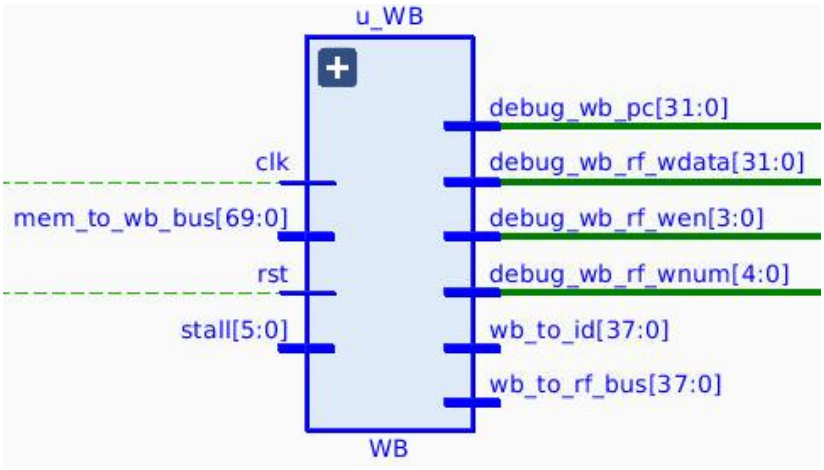
```
assign rf_wdata =  (data_ram_readen==4'b1111 && data_ram_en==1'b1) ? data_sram_rdata
                 : (data_ram_readen==4'b0001 && data_ram_en==1'b1 && ex_result[1:0]==2'b00) ?({{24{data_sram_rdata[7]}},data_sram_rdata[7:0]})
                 : (data_ram_readen==4'b0001 && data_ram_en==1'b1 && ex_result[1:0]==2'b01) ?({{24{data_sram_rdata[15]}},data_sram_rdata[15:8]})
                 : (data_ram_readen==4'b0001 && data_ram_en==1'b1 && ex_result[1:0]==2'b10) ?({{24{data_sram_rdata[23]}},data_sram_rdata[23:16]})
                 : (data_ram_readen==4'b0001 && data_ram_en==1'b1 && ex_result[1:0]==2'b11) ?({{24{data_sram_rdata[31]}},data_sram_rdata[31:24]})
                 : (data_ram_readen==4'b0010 && data_ram_en==1'b1 && ex_result[1:0]==2'b00) ?({24'b0,data_sram_rdata[7:0]})
                 : (data_ram_readen==4'b0010 && data_ram_en==1'b1 && ex_result[1:0]==2'b01) ?({24'b0,data_sram_rdata[15:8]})
                 : (data_ram_readen==4'b0010 && data_ram_en==1'b1 && ex_result[1:0]==2'b10) ?({24'b0,data_sram_rdata[23:16]})
                 : (data_ram_readen==4'b0010 && data_ram_en==1'b1 && ex_result[1:0]==2'b11) ?({24'b0,data_sram_rdata[31:24]})
                 : (data_ram_readen==4'b0011 && data_ram_en==1'b1 && ex_result[1:0]==2'b00) ?({{16{data_sram_rdata[15]}},data_sram_rdata[15:0]})
                 : (data_ram_readen==4'b0011 && data_ram_en==1'b1 && ex_result[1:0]==2'b10) ?({{16{data_sram_rdata[31]}},data_sram_rdata[31:16]})
                 : (data_ram_readen==4'b0100 && data_ram_en==1'b1 && ex_result[1:0]==2'b00) ?({16'b0,data_sram_rdata[15:0]})
                 : (data_ram_readen==4'b0100 && data_ram_en==1'b1 && ex_result[1:0]==2'b10) ?({16'b0,data_sram_rdata[31:16]})
                 : ex_result;
```

例如判断是否是 lw 指令(data_ram_readen=4’b1111)如果是，就将要写入寄存器的值，更新为从内存中读出来 rf_wdata = data_sram_rdata

2.5 WB 段

整体功能：将结果写入寄存器，这一段实际是在 ID 段调用 regfile 模块实现的

端口与信号介绍：



接口	长度	输入/输出	作用
clk	1	输入	时钟信号
rst	1	输入	复位信号
stall	6	输入	控制暂停信号
mem_to_wb_bus	70	输入	MEM 传给 WB 的数据
wb_to_rf_bus	38	输出	WB 传给 rf 的数据
wb_to_id	38	输出	WB 传给 ID 的数据
debug_wb_pc	32	输出	用来 debug 的 pc 值
debug_wb_rf_wen	4	输出	用来 debug 的写使能信号
debug_wb_rf_wnum	5	输出	用来 debug 的写寄存器地址
debug_wb_rf_wdata	32	输出	用来 debug 的写寄存器数据

详细说明：

```
assign {
    wb_pc,
    rf_we,
    rf_waddr,
    rf_wdata
} = mem_to_wb_bus_r;
```

接收 mem 发送数据，得到 PC 值，是否要写入寄存器，以及要写入寄存器的值还有数据。

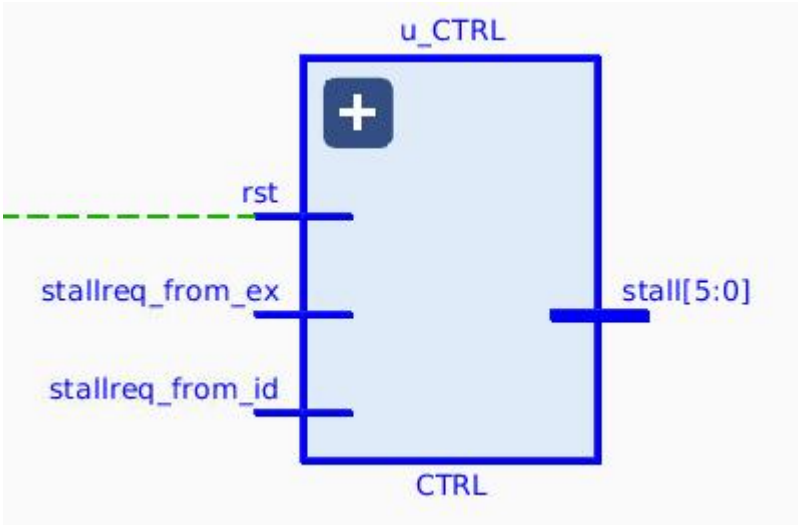
```
assign wb_to_rf_bus = {
    rf_we,
    rf_waddr,
    rf_wdata
};
```

将数据发送给 ID 模块，在 ID 模块将数据写入寄存器

2.6 CTRL 模块

整体功能：接收各段传递过来的流水线请求信号，从而控制流水线各阶段的运行。

端口与信号介绍：



接口	长度	输入/输出	作用
rst	1	输入	复位信号
stallreq_from_ex	1	输入	处于译码阶段的指令是否请求流水线暂停
stallreq_from_id	1	输入	处于执行阶段的指令是否请求流水线暂停
stall	6	输出	暂停信号

详细说明:

根据 id 和 ex 模块发送的请求暂停信号输出 stall，控制流水线的暂停。stall 第 0 到第 5 位分别代表不暂停，if 段暂停，id 段暂停等。

stall[0]为 1 表示没有暂停

stall[1]为 1 if 段暂停

stall[2]为 1 id 段暂停

stall[3]为 1 ex 段暂停

stall[4]为 1 mem 段暂停

stall[5]为 1 wb 段暂停

```
else if(stallreq_from_ex == 1'b1) begin
    stall <= 6'b001111;
end
else if(stallreq_from_id == 1'b1) begin
    stall <= 6'b000111;
end else begin
    stall <= 6'b000000;
end
```

如果 id 段请求暂停，那么就把暂停控制信号赋值为 6'b000111，表示取值阶段暂停，译码阶段暂停，执行阶段不暂停，访存阶段不暂停，回写阶段不暂停。

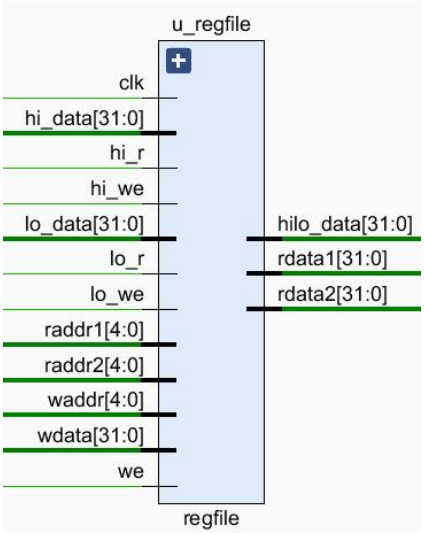
如果 ex 段请求暂停，那么就把暂停控制信号赋值为 6'b000111，表示取值阶段暂停，译码阶段暂停，执行阶段暂停，访存阶段不暂停，回写阶段不暂停。

如果都不暂停将流水线控制信号赋值为 6'b000000，表示不暂停。

2.7 regfile 模块：

整体功能：提供寄存器模块进行数据操作。

端口与信号介绍：



接口名	长度	输入/输出	作用
clk	1	输入	时钟信号
raddr1	5	输入	请求读取的第一个数的地址
rdata1	32	输出	读取出的第一个数的值
raddr2	5	输入	请求读取的第二个数的地址
rdata2	32	输出	读取出的第二个数的值
we	1	输入	是否要写入寄存器的写使能信号
waddr	5	输入	要写入的地址
wdata	32	输入	要写入寄存器的值

hi_r	1	输入	是否要读取 hi 寄存器的值的读使能信号
hi_we	1	输入	是否要写入 hi 寄存器的写使能信号
hi_data	32	输入	要写入 hi 寄存器的值
lo_r	1	输入	是否要读取 lo 寄存器的值的读使能信号
lo_we	1	输入	是否要写入 lo 寄存器的写使能信号
lo_data	32	输入	要写入 lo 寄存器的值
hilo_data	32	输出	从 hilo 寄存器中读取出来的值

详细说明:

regfile 模块定义了 32 个 32 位的通用寄存器，以及一个乘除法的高位 hi 寄存器，一个乘除法的低位 lo 寄存器。于所有乘除法高位寄存器以及乘除法低位寄存器在调用时，不会同时调用 hi 寄存器和 lo 寄存器的值，即只会读取其中一个寄存器的值,所以我们的 hilo 寄存器在实现的时候只有一个输出的接口。在读取 hilo 寄存器中的值的时候， 先判断 hi_r 是否等于 1’b1，如果等于，输出的 hilo_data 赋值为乘除法高位 hi 寄存器的值；再判断判断 lo_r 是否等于 1’b1,如果是的话那么输出的 hilo_data 赋值为乘除法低位 lo 寄存器的值，如果两个都为 0 即不需要读取 hilo 寄存器的值，输出的 hilo_data 为 0。 判断 raddr1 是否为零，如果为零，就把 32’b0 赋值给 rdata1,如果不为零，就把 raddr1 对应的寄存器的值赋值给 rdata1; 判断 raddr2 是否为零，如果为零，就把 32’b0 赋值给 rdata2,如果不为零，就把 raddr2 对应的寄存器的值赋值给 rdata2。

3. 感受与意见

刘愉鑫:

这次实验确实一开始没有头绪无从下手，但通过查阅资料与同学交流渐渐找到了思路知道了如何操作，并通过阅读完参考资料后知道了需要实现和解决之处，并不断尝试成功解决问题，不仅提升了动手能力也是对课堂知识更深层次的掌握。

张杰凯:

在这次实验中，我深刻体会到什么是“透过现象看本质，勿使表象遮望眼”。在刚知道要实现一个简化 CPU 的我被吓到了，放眼望去，全是看不懂的代码与变量，也不知道该如何处理 bug，如何找到错误，甚至不知道怎么使用 vivado 软件。

但是，我们不应该被处理器的庞大神秘吓到，通过询问同学与助教，我逐渐懂得了如何完成任务。时空与空间定位 bug，解决数据相关，发出访存信号，添加第一条指令。一点一点，一块一块的，蓦然回首，你会发现，任务原来那么简单。总的来说，这一次计算机系统的实验。让我对于 CPU 有了一个更加深入的了解和认识，收获很多。

张玉龙:

首先，我想说，在这次实验中，我感受到了使用 verilog 语言自己动手写 cpu 的乐趣。虽然在开始时，我对这门语言和这项任务有些恐惧，但是当我慢慢掌握了这门语言，并且成功地编写出了我们自己的 cpu，我感到非常的兴奋和成就感。

其次，我觉得这次实验对我的技能提升也有很大的帮助。让我更加熟悉了计算机系统的工作原理，也增强了我的硬件知识。我相信，这些知识和技能在今后的学习和工作中都会大有裨益。

4. 参考资料

1、张晨曦 著《计算机体系结构》（第二版） 高等教育出版社

2、雷思磊 著《自己动手写 CPU》 电子工业出版社

3、助教给出的龙芯杯官方的参考文档