实验报告成绩：	成绩评定日期：

2024～2025 学年秋季学期

《计算机系统》必修课

课程实验报告

班级：人工智能未来班，人工智能 2201

组长：王宇菲

组员：高向博

报告日期：2024.12.21

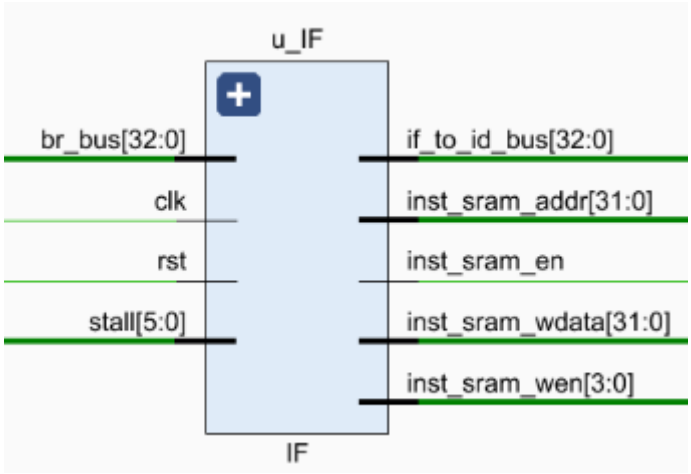
一、小组成员工作量划分

姓名	完成任务量	占比
王宇菲	设计实现 ID、EX 段	60%
高向博	完成其他段，自制乘法器	40%

二、五段流水实现

1、IF

接口：



序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	控制暂停信号
4	br_bus	33	输入	ID 段发出的分支跳转指令的信号，控制指令延迟槽是否跳转
5	if_to_id_bus	33	输出	IF 段发给 ID 段的数据
6	inst_sram_en	1	输出	指令寄存器的读写使能信号
7	inst_sram_wen	4	输出	指令寄存器的写使能信号
8	inst_sram_add	32	输出	指令寄存器的

	r			地址，用来寻找指令的存放的位置
9	inst_sram_wdata	32	输出	指令寄存器的数据，用来存放数据

通过两个寄存器 pc_reg 和 ce_reg 来存储当前的 PC 值和使能信号。pc_reg 用于跟踪当前指令位置，而 ce_reg 则用于控制指令存储器的使能信号，确保只有在流水线没有 stalled 时才执行指令读取操作。

模块的核心逻辑首先在时钟的上升沿检查复位信号 rst，如果复位信号有效，则将 pc_reg 初始化为一个预定义的值（32'hbfbf_fffc），这是开始执行程序的初始 PC 值。如果复位信号无效且流水线没有停顿（通过检查 stall[0]是否为 NoStop），则根据是否接收到分支信号 br_e 来决定下一个 PC 值。如果接收到分支信号，下一个 PC 值将是分支目标地址 br_addr，否则，PC 值将递增 4。

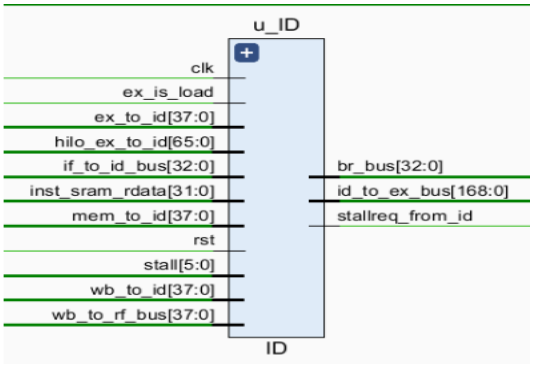
使能信号 ce_reg 的处理逻辑与 PC 相同，也在复位时被清零，并且在流水线没有停顿时被置 1，表示允许从指令存储器中取指令。

在赋值信号（assign）上，inst_sram_en 用于命令存储器使能，其值取决于 ce_reg；inst_sram_wen 被设置为 4 个位宽的 0 值，因为在这个阶段我们只读取指令不写入任何东西；inst_sram_addr 用于指定读取指令的地址，即 pc_reg 的值；inst_sram_wdata 设置为零，因为它在这个阶段不起作用；if_to_id_bus 是一个输出总线，用于把当前的使能状态和 PC 值传输到下一个流水线阶段。

---

2. ID

接口：



序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	控制暂停信号
4	ex_is_load	1	输入	判断上一条指令是否是加载指令
5	stallreq	1	输出	控制暂停
6	if_to_id_bus	33	输入	IF 段发给 ID 段的数据
7	inst_sram_rdata	32	输入	当前指令地址中储存的值
8	wb_to_rf_bus	38	输入	WB 段传给 ID 段要写入寄存器的值
9	ex_to_id	38	输入	EX 段传给 ID 段的数据，用来判断数据相关
10	mem_to_id	38	输入	MEM 段传给 ID 段的数据，用来判断数据相关
11	wb_to_id	38	输入	WB 段传给 ID 段的数据，用来判断数据相关
12	hilo_ex_to_id	66	输入	EX 传给 ID 段要写入乘除法寄存器的值
13	id_to_ex_bus	169	输出	ID 段传给 EX 段的数据
14	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
15	stallreq_from_id	1	输出	从 ID 段发出的暂停信

ID 段工作：

第一段：流水线是否暂停的判断和加气泡的实现。

```
always @ (posedge clk) begin
    if (rst) begin
        if_to_id_bus_r <= `IF_TO_ID_WD'b0;
    end
    // else if (flush) begin
    //     ic_to_id_bus <= `IC_TO_ID_WD'b0;
    // end
    else if (stall[1]==`Stop && stall[2]==`NoStop) begin
        if_to_id_bus_r <= `IF_TO_ID_WD'b0;
    end
    else if (stall[1]==`NoStop) begin
        if_to_id_bus_r <= if_to_id_bus;
    end
end
```

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。

```
always @(posedge clk) begin
    if (stall[1]==`Stop) begin
        q <= 1'b1;
    end
    else begin
        q <= 1'b0;
    end
end
assign inst = (q) ?inst: inst_sram_rdata;
```

但是还存在的另外一个问题就是 if_to_id_bus 是 IF 段发给 ID 段的内容,而这部分的内容是不包含指令值的，指令的值即 inst 值是在 ID 段时根据上个周期的 IF 段中的 pc 值从内存中读取到的,是直接从内存获取的。在暂停发生后 ID 段虽然对 if_to_id_bus 置为 0 但是 inst 值并没有被置为 0，因此还需要另外的一个操作就是需要判断暂停发生后把当前时刻的 inst 值保存一个周期，下一个周期再使用当前周期的 inst 值，只有这样才可以保证 ID 段和之后所有部分的指令的 pc 值和 inst 值是相互匹配的。

第二段：添加指令：实现对指令的译码，依据指令中的特征字段区分指令，同时定义相应的指令对应的 inst_**变量为每条指令的操作码（op_d）和功能码（func_d），表示是哪一条指令。

根据译码结果，读取通过 regfile 模块读取地址为 rs（inst[25:21]）以及地址为 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，0 表示该条指令不采用该 ALU，1 表示该条指令采用该 ALU，同时将所有的 ALU 标识符组合起来成为 alu_op，alu_op 为十二位宽，代表 16 种不同的 ALU，并且作为传入 EX 段的一部分。

要写入的目的寄存器。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，就比如 sw 指令要将整个寄存器的值写入内存，那么 data_ram_wen 就设置为 4’b1111。data_ram_readen 为四位宽，表示该条指令是否要从寄存器中读取内容，并设置不同的读取方式。

通过逻辑运算生成了分支使能信号（br_e）和分支地址（br_addr）。根据不同的分支指令类型。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

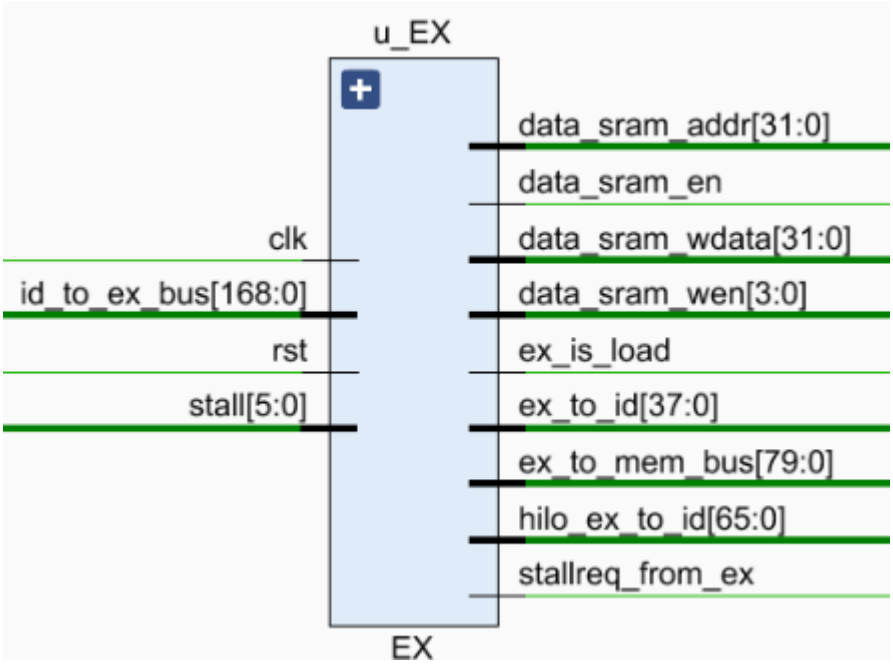
PC 计算得到的值赋值给 bre_addr。如果是 bltzal 指令，就将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 bre_addr。如果是 j 指令，就将该分支指令对应的延迟槽指令的 PC 的最高四位与立即数 inst_index(inst[26:0])左移两位后的值拼接之后赋值给 bre_addr。如果是 jal 指令，就将该分支指令对应的延迟槽指令的 PC 的最高四位与立即数 inst_index(inst[26:0])左移两位后的值拼接之后赋值给 bre_addr。

第三阶段：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，只有个别指令可以用的到，第四种，第二个操作数的值为立即数零扩展。

第四阶段：传值给其他阶段。传值给 ex 阶段，将 其中 data_ram_readen 表示是否需要从内存中读取数据，inst_mthi,inst_mtlo,inst_multu, inst_divu，inst_div, inst_mult, 分别表示是否是 mthi,mtlo,multu,mult,divu,div,指令，在 EX 会用到，判断是否调用乘法器以及除法器，alu_op 传到 EX 段，告诉 EX 段将要调用哪一个 ALU，sel_alu_src1 告诉 EX 段操作数一的来源，sel_alu_src2 告诉 EX 段操作数二的来源，data_ram_en，data_ram_wen 告诉 EX 段是否要与内存进行交互，rf_we，rf_waddr 告诉 EX 段是否要将结果写入寄存器，以及要写入寄存器的地址。 传给 IF 段，告诉 IF 段目前指令是否为跳转指令，以及要跳转的地址。

3. EX

接口：



序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	控制暂停信号
4	id_to_ex_bus	169	输入	ID 段传给 EX 段的数据
5	ex_to_mem_bus	80	输出	EX 段传给 MEM 短的数据
6	data_sram_en	1	输出	内存数据的读写使能信号
7	data_sram_wen	4	输出	内存数据的写使能信号
8	data_sram_addr	32	输出	内存数据存放的地址
9	ex_to_id	38	输出	EX 段传给 ID 段的数据
10	data_sram_wdata	32	输出	要写入内存的数据
11	stallreq_from_ex	1	输出	EX 发出的是否暂停的信号
12	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
13	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

第一阶段：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；如果不是，就将 ex_is_load 赋值为 0。计算 ALU 来个操作数的值：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。

第二阶段：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 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。

第三阶段：乘除法指令的实现：在 ex 段我们加入了 MUL 和 DIV 模块，借此完成乘法和除法的指令。在收到来自 ID 段的 id_to_ex_bus 后，可以判别是否为乘数法，并调用相应的乘法或除法模块进行运算。 如果是乘法，则需要声明几个变量，分别为：stallreq_for_mul、mul_ready_i、signed_mul_o、mul_opdata1_o、mul_opdata2_o、mul_start_o。stallreq_for_mul 表明是否因为多周期的 mul 进行流水线的暂停，mul_ready_i 表明乘法是否已经结束，signed_mul_o 表示是否为有符号的乘法，mul_opdata1_o、mul_opdata2_o 是 ID 段传过来的两个操作数，mul_start_o 表明是否开始乘法运算。具体的乘法器模块的解释说明在下文中给出，这里需要说明的是对传入乘法器 mymul 的相应的值的赋值。

当检测到乘法的指令 inst_mult 或者 inst_multu 乘法器就会开始运作，开始前，会先判断当前乘法器的状态，若当前乘法器为空闲的状态，则将操作数传入乘法器开始运算，同时因为该乘法器是 32 周期的，因此需要对流水线进行暂停，将 stallreq_for_mul 的值改为`Stop 并传给 CTRL 模块进行流水线暂停。当乘法指令运行结束后，stallreq_for_mul 的值改为`NoStop，流水线继续运行。乘法的运行结果存放到了 mul_result 中。

除法指令的实现：

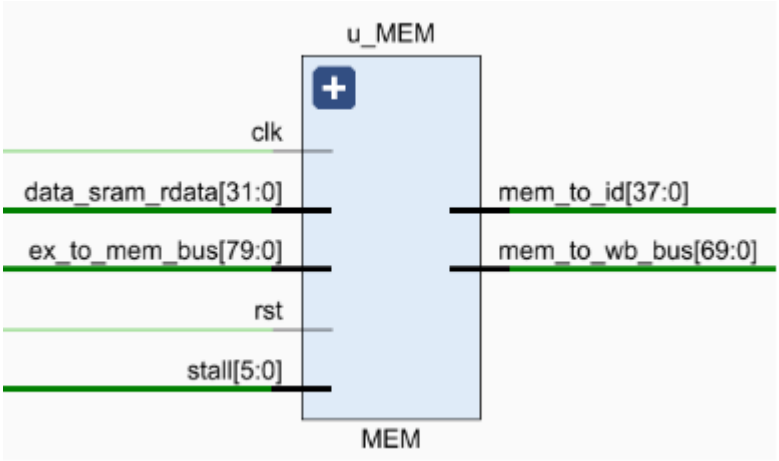
除法指令的实现与乘法类似，都是先判断是否为有符号除法或无符号除法，然后判断除法器的状态是否为空闲，除法器可用时，就把被除数和除数传入除法器，并把 stallreq_for_div 值改为`Stop 来暂停流水线。在除法结束后结果放在 div_result 中并恢复流水线继续运行。

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。至此，乘法和除法的指令已经可以正常运行。

4. MEM

接口：

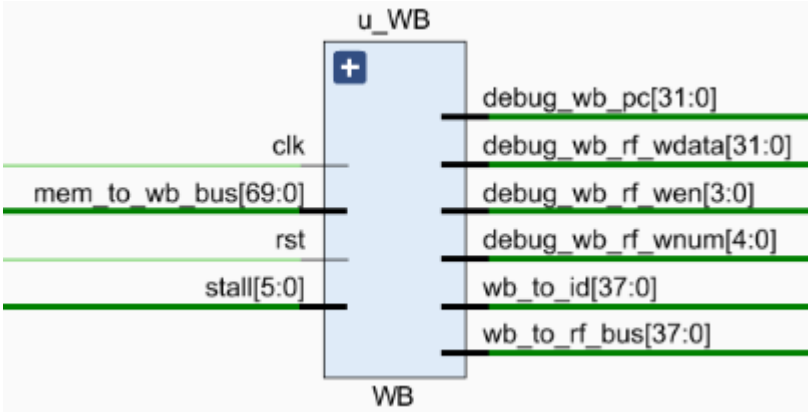


序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	控制暂停信号
4	ex_to_mem_bus	80	输入	EX 传给 MEM 段的数据
5	data_sram_rdata	32	输入	从内存中读出来要写入寄存器的值
6	mem_to_id	38	输出	MEM 传给 ID 段的数据
7	mem_to_wb_bus	70	输出	MEM 传给 WB 段的数据

确定要写入寄存器的值：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。此处可能需要插入图片。判断是否是 lw 指令(lw 指令是将从内存中的值全部写入相应寄存器)，如果是，就将要写入寄存器的值，更新为从内存中读出来的值；如果不是 lw 指令，判断是否是 lb 指令（lb 指令是根据要写入寄存器的地址的最后两位判断要将从内存中读出来的数据的哪一个字节写入寄存器的最低一个字节，其他字节采用符号扩展的方式），如果是，判断要写入寄存器

的地址的后两位的值，如果为 2’b00,就将从内存中读出来的值的第一个字节写入寄存器的最低一个字节其他部分用第 7bit 的值进行符号扩展，如果为 2’b01,就将从内存中读出来的值的第二个字节写入寄存器的最低一个字节其他部分用第 15bit 的值进行符号扩展，如果为 2’b10,就将从内存中读出来的值的第三个字节写入寄存器的最低一个字节其他部分用第 23bit 的值进行符号扩展，如果为 2’b11,就将从内存中读出来的值的第四个字节写入寄存器的最低一个字节其他部分用第 31bit 的值进行符号扩展；如果也不是 lb 指令，判断是否是 lbu 指令（lbu 指令是根据要写入寄存器的地址的最后两位判断要将从内存中读出来的数据的哪一个字节写入寄存器的最低一个字节，其他字节采用零扩展的方式），如果是，判断要写入寄存器的地址的后两位的值，如果为 2’b00,就将从内存中读出来的值的第一个字节写入寄存器的最低一个字节其他部分进行零扩展，如果为 2’b01,就将从内存中读出来的值的第二个字节写入寄存器的最低一个字节其他部分进行零扩展，如果为 2’b10,就将从内存中读出来的值的第三个字节写入寄存器的最低一个字节其他部分进行零扩展，如果为 2’b11,就将从内存中读出来的值的第四个字节写入寄存器的最低一个字节其他部分进行零扩展；如果也不是 lbu 指令，判断是否是 lh 指令（lh 指令是根据要写入寄存器的地址的最后两位判断要将从内存中读出来的数据的哪两个字节写入寄存器的最低两个字节，其他字节采用符号扩展的方式），如果是，判断要写入寄存器的地址的后两位的值，如果为 2’b00,就将从内存中读出来的值的第一，第二字节写入写入寄存器的最低两个字节其他部分用第 15bit 的值进行符号扩展，如果为 2’b00,就将从内存中读出来的值的第三，第四字节写入写入寄存器的最低两个字节其他部分用第 15bit 的值进行符号扩展；如果也不是 lh 指令，判断是否是 lhu 指令（lhu 指令是根据要写入寄存器的地址的最后两位判断要将从内存中读出来的数据的哪两个字节写入寄存器的最低两个字节，其他字节采用零扩展的方式），如果是，判断要写入寄存器的地址的后两位的值，如果为 2’b00,就将从内存中读出来的值的第一，第二字节写入写入寄存器的最低两个字节其他部分进行零扩展，如果为 2’b10,就将从内存中读出来的值的第三，第四字节写入写入寄存器的最低两个字节其他部分进行零扩展，如果都不是，那么将要写入寄存器的值赋值为指令阶段计算的结果。

5. WB

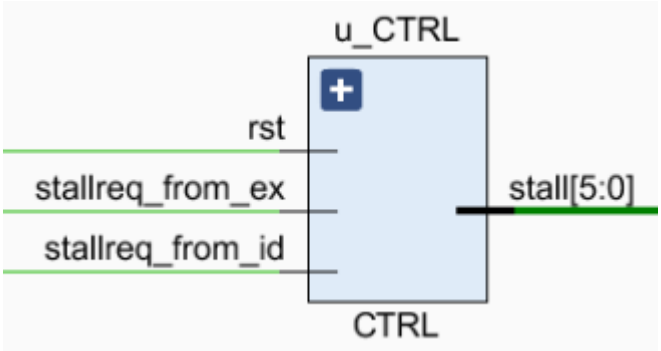


序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	控制暂停信号
4	mem_to_wb_bus	70	输入	MEM 传给 WB 的数据
5	wb_to_rf_bus	38	输出	WB 传给 rf 的数据
6	wb_to_id	38	输出	WB 传给 ID 的数据
7	debug_wb_pc	32	输出	用来 debug 的 pc 值
8	debug_wb_rf_wen	4	输出	用来 debug 的写使能信号
9	debug_wb_rf_wnum	5	输出	用来 debug 的写寄存器地址
10	debug_wb_rf_wdata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

当没有停顿信号时，mem_to_wb_bus_r 捕获来自 MEM 阶段的 mem_to_wb_bus 数据，这些数据包括程序计数器 wb_pc、寄存器写使能信号 rf_we、写地址 rf_waddr 以及写数据 rf_wdata。通过这种方式，WB 阶段持有最新的写回信息，准备将其传递给寄存器文件。

Wb_to_rf_bus 和 wb_to_id 均打包了写使能、写地址和写数据信号，分别用于不同的模块接口，确保写回信息能够正确传递和使用。此外，调试信号如 debug_wb_pc、debug_wb_rf_wen、debug_wb_rf_wnum 和 debug_wb_rf_wdata 被赋值，以便在调试过程中监控 WB 阶段的具体操作。这些调试信号提供了程序计数器的值、寄存器写使能的状态、写地址以及写入的数据，帮助开发者分析和验证流水线的行为。

6. CTRL

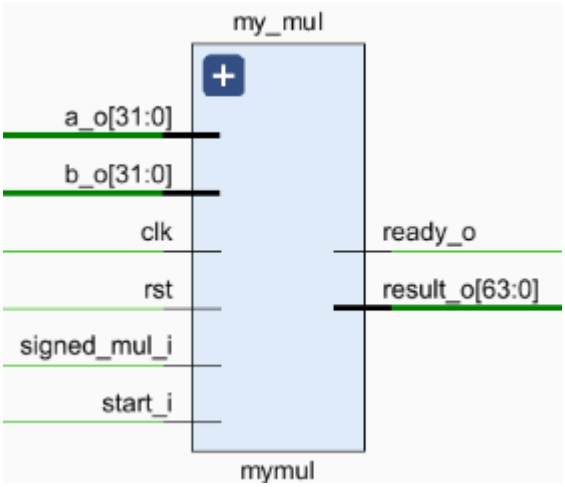


序号	接口名	宽度	输入/输出	作用
1	rst	1	输入	复位信号
2	stallreq_from_ex	1	输入	处于译码阶段的指令是否请求流水线暂停
3	stallreq_from_id	1	输入	处于执行阶段的指令是否请求流水线暂停
4	stall	6	输出	暂停流水线控制信号

stall 用于表示流水线各个阶段的暂停状态，其中 stall[0]表示整体流水线是否暂停（1 表示不暂停），而 stall[1]到 stall[5]分别对应 IF（取指）、ID（指令译码）、EX（执行）、MEM（存储访问）和 WB（写回）阶段的暂停状态。

7. MUL

接口：



序号	接口名	宽度	输入输出	作用
1	rst	1	输入	复位信号，高电平有效
2	clk	1	输入	时钟信号
3	signed_mul_i	1	输入	是否为有符号乘法，为 1 表示有符号乘法
4	a_o	32	输入	被乘数
5	b_o	32	输入	乘数
6	start_i	1	输入	是否开始乘法运算
7	result_o	64	输出	乘法运算结果
8	ready_o	1	输出	乘法运算是否结束

通过时钟信号 clk 和复位信号 rst 进行同步操作。当复位信号被激活时，模块进入空闲状态，初始化结果寄存器 result_o 和就绪信号 ready_o。在空闲状态（MulFree）下，模块等待乘法运算的启动信号 start_i。一旦检测到启动信号，模块进入乘法进行状态（MulOn），并开始逐位处理乘数 b_o 和被乘数 a_o。

为了支持有符号乘法运算，通过 signed_mul_i 信号判断操作数是否为有符号数。如果是有符号数且负数，会对被乘数和乘数进行二补码处理，以确保乘法结果的正确性。处理后的被乘数存储在 temp_opa，乘数存储在 temp_opb，并将被乘数扩展到 64 位存储在 ap 寄存器中，准备进行逐位累加。

在 MulOn 状态下，模块通过移位和加法操作逐位计算乘积。具体来说，每一时钟周期检查乘数的最低位，如果为 1，则将当前的部分积 ap 加到累加寄存器 pv 中。随后，ap 左移一位，乘数 temp_opb 右移一位，准备处理下一个有效位。这个过程持续进行，直到处理完所有 32 位。

当所有位处理完成后，如果是有符号乘法且操作数符号不同，模块会对累加结果 pv 进

行二补码处理，以得到正确的负数结果。随后，模块进入结束状态（MulEnd），将最终结果赋值给 result_o，并将 ready_o 信号置为就绪状态，表示乘法运算完成。

在结束状态下，模块等待 start_i 信号的停止指示（MulStop），然后返回空闲状态，准备接受下一个乘法运算请求。整个过程中，状态机的设计确保了乘法运算的有序进行，并通过寄存器和信号的合理配置，实现了数据的正确传递和处理。

三、组员感受

王宇菲：

设计和实现 EX（执行）段与 ID（指令译码）段的代码既具挑战性又充满学习机会。在 EX 段中，核心任务是实现指令执行逻辑，包括数据通路控制、ALU 操作、内存访问和乘除法处理。通过精确控制信号，确保数据流正确，特别是在处理 LW 时，需进行符号或零扩展，并更新 ALU 控制信号。内存访问需计算正确地址并设读写使能，处理不同指令格式如 sb、sh 等。乘除法操作复杂耗时，需引入 stall 机制避免流水线冲突，并高效管理结果如使用 hilo 寄存器。

ID 段则侧重指令解码和控制信号生成，将机器码转为具体执行动作。通过解析指令类型和操作码，为后续段提供精确操作数和控制信息，处理数据前向和冒险，减少流水线停滞。团队合作也至关重要，从方案讨论到代码审查和测试验证，均需紧密配合。

通过设计和实现 EX/ID 段，我不仅深化了对硬件设计的理解，还积累了实际操作经验，这些都将为未来工作奠定基础。

高向博：

在完成这个实验的过程中，我深入理解了流水线控制的核心机制，包括暂停信号的处理和各阶段之间的数据传递。通过设计 CTRL 模块有效地管理不同阶段的暂停请求，确保了指令执行的有序性和数据的一致性。在 IF、MEM 和 WB 模块中，实现了程序计数器的更新、数据存储器的访问以及写回阶段的信号传递，进一步巩固了我对 Verilog 模块化设计的掌握。同时，处理各种数据读取方式和写回逻辑。这个实验不仅提升了我的硬件描述语言编程能力，也加深了我对计算机体系结构中流水线优化的理解。