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 2201 班

组长：刘奕扬

组员：刘 策 于济航

报告日期：2025.1.3

目录

1. 实验设计

1.1 小组成员工作量划分

姓名	任务分工	任务量占比
刘奕扬	添加算术运算、数据移动、逻辑、跳转、访存指令，参与实现 hilo 寄存器、参与实现 stall	40%
刘策	主要负责在流水线中添加 stall 相关指令，参与 hilo 的相关指令，编写实验报告。	30%
于济航	主要负责在流水线中添加 hilo 相关指令，参与实现 stall 的相关指令，参与实验报告的编写	30%

1.3 运行环境及工具

运行环境：装有 Vivado 的 win11 操作系统。FPGA 的 Family 为 Artix 7，Package 为 fbg676，型号为 xc7a200tfbg676-2

编程工具：使用 VSCode 编写代码，使用 Vivado 模拟仿真，使用 git 进行版本管理，使用 GitHub 搭建项目仓库。

1.2 总体设计

项目包括 IF.v，ID.v，EX.v，MEM.v，WB.v，CTRL.v， mycpu_core.v，mycpu_top.v，构成流水线的基本框架；及位于/lib 目录下的 alu.v，decoder_5_32.v，decoder_6_64.v，defines.vh，div.v，mmu.v，regfile.v，包括了 ALU 和寄存器的构建，还有总线宽度等信息在内的头文件；位于/lib/mul 目录下的 add.v，fa.v，mul.v，这部分实现了乘法的运算。

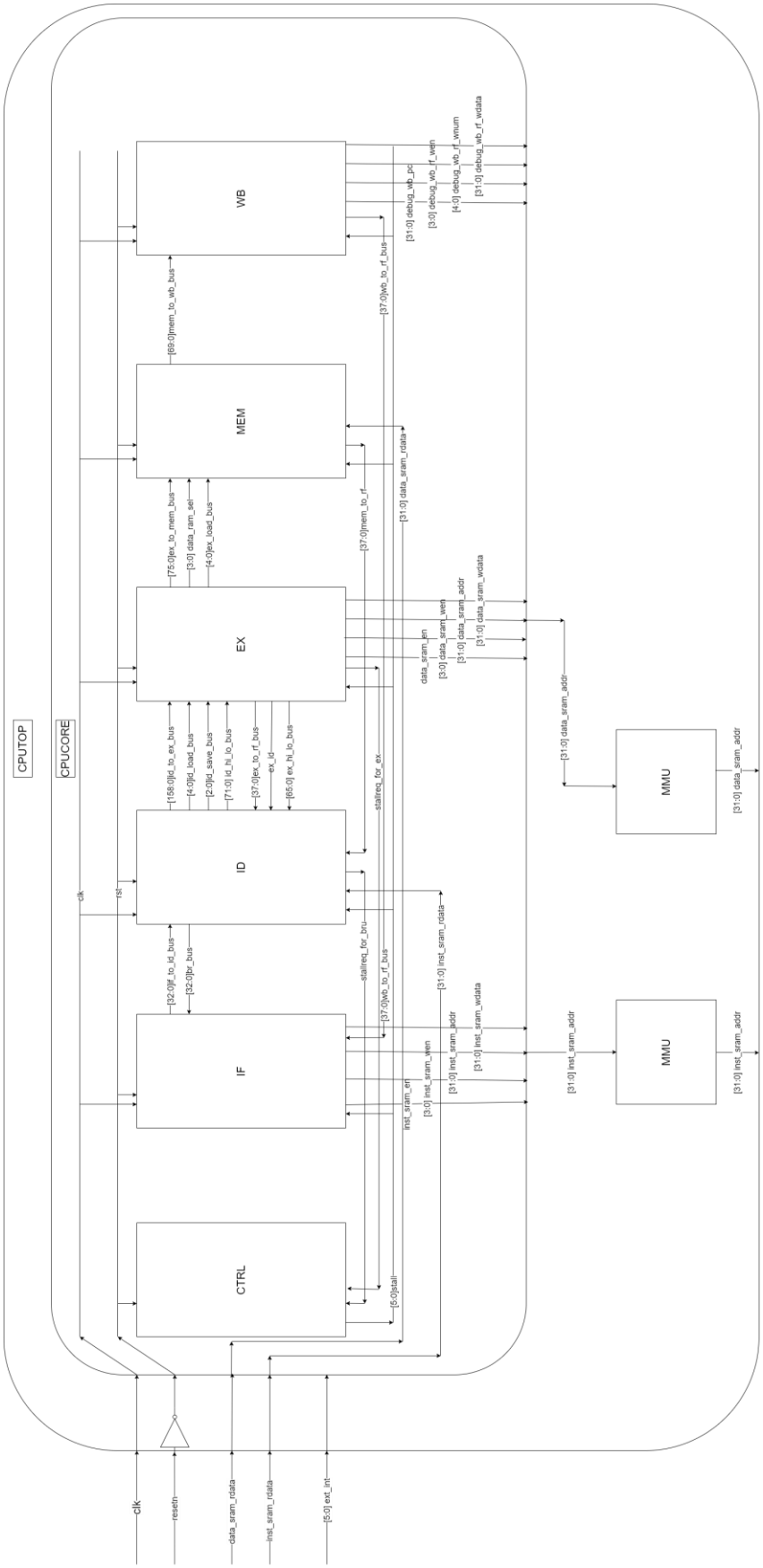


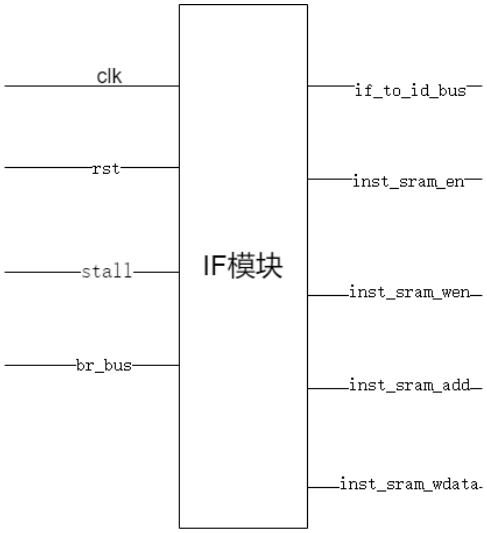
图 1 CPU 流水线示意图

2. 流水线各个阶段的说明

2.1 IF 模块

整体说明：取指令，控制指令延迟槽和跳转指令。接口如图所示。

功能说明：



首先，IF 段会输入时钟信号和复位信号，如果复位信号为真，则复位 pc 值。然后，再判断暂停信号 stall，如果 stall 的值为 1，则暂停延迟槽，即让下一条指令的 pc 值等于当前的 pc 值。再然后，判断 br_bus 的值，若需要跳转，则取出 br_bus 中的地址值赋给 next_pc，然后再把 next_pc 赋给 pc_reg，否则，pc_reg 值为当前 next_pc 的值，next_pc 值还需要加 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。

IF 会段接收时钟信号（clk）和复位信号（rst）。如果复位信号为真（rst == 1），pc 值会被复位，同时指令存储使能信号 ce_reg 也被置为 0，这样来确保 CPU 在复位后，从固定的起始地址开始执行指令，并且在复位期间暂停从指令存储中取指。如果暂停信号 stall[0] == 1，IF 段暂停更新 pc 值，即下一条指令的地址与当前地址相同。确保延迟槽或其他控制信号的处理不会导致流水线出错。PC 值更新逻辑：判断 br_bus 的值，若需要跳转，则取出 br_bus 中的地址值赋给 next_pc，然后再把 next_pc 赋给 pc_reg，否则，pc_reg 值为当前 next_pc 的值，next_pc 值还需要加 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。

表 1 IF 模块输入输出

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	暂停信号，控制指令是否暂停
4	br_bus	33	输入	分支跳转信号，控制延迟槽是否跳转
5	if_to_id_bus	33	输出	IF 段到 ID 段的数据总线
6	inst_sram_en	1	输出	读写使能信号
7	inst_sram_wen	4	输出	写使能信号
8	inst_sram_addr	32	输出	存放指令寄存器的地址
9	inst_sram_wdata	32	输出	存放指令寄存器的数据

2.2 ID 模块

整体说明：

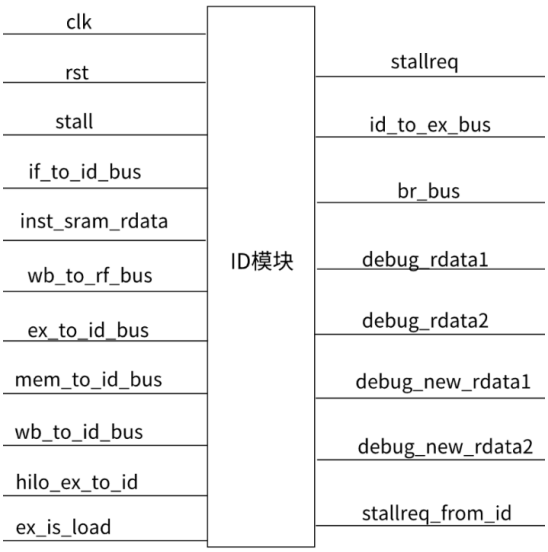
ID 段是解码模块，从 IF 阶段传来的指令数据中提取操作码、寄存器编号和立即数等信息，并进行相应的解码操作。这些信息将用于生成控制信号并传递给后续 EX 段。同时还处理数据相关和跳转相关。

表 2 ID 模块输入输出

序号	接口名	宽度	输入/输	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	暂停信号，控制指令是否暂停
4	stallreq	1	输出	流水线暂停信号
5	if_to_id_bus	33	输入	IF 段到 ID 段的数据总线
6	inst_sram_rdata	32	输入	读写使能信号
7	wb_to_rf_bus	38	输入	WB 段存放进寄存器的数据
8	id_to_ex_bus	163	输出	ID 到 EX 段数据总线
9	br_bus	33	输入	EX 段存放进寄存器的数据
10	ex_to_id_bus	38	输入	EX 返回 ID 段数据总线
11	mem_to_id_bus	38	输入	MEN 返回 ID 段数据总线
12	wb_to_id_bus	38	输入	WB 返回 ID 段数据总线
13	debug_rdata1	32	输出	用于调试的输出信号
14	debug_rdata2	32	输出	用于调试的输出信号
15	debug_new_rdata1	32	输出	用于调试的输出信号
16	debug_new_rdata2	32	输出	用于调试的输出信号
17	Stallreq_from_id	1	输出	来自 ID 段的暂停请求信号
18	hilo_ex_to_id	66	输入	用于 HI/LO 寄存器的转发信号
19	ex_is_load	1	输入	EX 段是否正在加载数据

功能说明：

实现暂停



当 ID 段检测到 stall 信号为`Stop 时，说明发生了访存冲突，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，且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，当前指令需要等待后续操作完成后才能继续执行。将 IF 段传入的总线信号 if_to_id_bus 清零，暂停流水线，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，当前周期的指令值 inst 需暂存，以保证暂停恢复后指令的执行与其对应的 PC 值一致，所以我们引入了一个一位寄存器，若当前判断为暂停了，那么就将寄存器值置 1，那么下个周期就不从内存中读取指令而是从当前 inst 值里读取就行。

而如何传递需要暂停的信号呢，在 ID 段我们接受了从 EX 段传来的 ex_is_load，了解了上一条指令是否为 lw 指令，而在这里我们通过判断，若当前指令需要对上一条指令的目的寄存器进行读操作，且上一条指令是 lw，那么我们就将 stallreq_from_id 置为 1，这样就可以传给 CTRL 段并且将 stall 信号设置为 000111，也就是暂停 ex 及其之后的段落。

指令译码

指令译码阶段主要根据指令操作码字段解析指令类型，激活相应控制信号。通过寄存器文件读取源操作数（rs 和 rt）获得 rdata1 和 rdata2，检测是否产生了数据相关问题并采用 forwarding 或暂停机制解决，确保数据一致性。生成 ALU 控制信号（alu_op），编码为 12 位以表示 16 种操作类型，并传递给 EX 段。同时，根据指令类型生成写使能信号（rf_we）和目标寄存器选择信号（sel_rf_dst），确定写入地址（rf_waddr）。同时在已有的内存访问控制信号读写使能信号 data_ram_en 和写使能信号 data_ram_wen 的基础上再加入了读使能信号 data_ram_readen 以区分读储存器操作，一起通过数据总线传给 ex 和 mem 段，决定不同的写入方式和读取方式。对于跳转指令，由一位的 br_e 和 32 位宽的 br_addr 控制并传给 IF 段计算 PC 值，其中 br_e 的 0 和 1 表示该条指令是否是跳转指令，

这五个变量根据当前的 rdata1 和 rdata2 的大小关系填写，控制 br_e 的值

```
assign rs_ge_z  = (new_rdata1[31] == 1'b0); //大于等于0, begz, begzal
assign rs_gt_z  = (new_rdata1[31] == 1'b0 & new_rdata1 != 32'b0  );  //大于0, bgtz
assign rs_le_z  = (new_rdata1[31] == 1'b1 | new_rdata1 == 32'b0  );  //小于等于0, blez
assign rs_lt_z  = (new_rdata1[31] == 1'b1);  //小于0, bltz, bltzal
assign rs_eq_rt = (new_rdata1 == new_rdata2);
```

之后再根据当前操作的类型来填写 br_addr 也就是跳转地址的值，查阅 AO3 文件，根据他们的定义完成这一部分。

处理数据相关

在 ID 段我们还接受了从 EX,MEM,WB 段 forwarding 来的数据来处理数据相关，也就是说，我们在这一段还需要根据当前寄存器和上一条指令所使用的寄存器来判断是否接收前递来的数据作为新的 rdata1 或 rdata2 的值。

判断操作数来源

sel_alu_src1：控制第一个操作数的来源，共三种情况：（1）第一个操作数的值为 rs 寄存器的值，（2）当前 的 PC 值，（3）立即数零扩展。
sel_alu_src2：控制第二个操作数的来源，共四种情况：（1）：第二个操作数的值为 rs 寄存器的值，（2） rs 的值为立即数符号扩展，（3）将第二个操作数复制为 32'b8，只有个别指令可以用的到，（4）以第二个操作数的值为立即数零扩展。

传值给 ex 阶段：在这里我们拓宽了传值给 ex 段总线的长度，

```
inst_mthi,      // 168
inst_mtlo,      // 167
inst_multu,     // 166
inst_mult,      // 165
inst_divu,      // 164
inst_div,       // 163
data_ram_readen,// 162:159
```

其中六个操作指令是在 EX 段使用，判断是否需要使用乘法器或除法器。

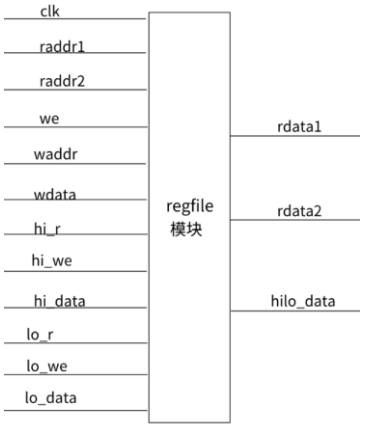
而 data_ram_readen 则是区分访存操作的，在 ex 段主要是区分 sb、sh、sw 操作，来区分当前写内存的方式，之后还得继续传给 MEM 段，在 MEM 段我们再做介绍。

regfile 模块接口如图所示。

regfile 模块说明：

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	we	1	输入	寄存器的写使能信号
3	raddr1	5	输入	读取的第一个数的地址
4	raddr2	5	输入	读取的第二个数的地址
5	waddr	32	输入	将要写入数据的寄存器的地址
6	wdata	32	输入	要写入寄存器的数据
7	hi_r	1	输入	是否读取 hi 寄存器的信号
8	hi_we	1	输入	是否写入 hi 寄存器的信号
9	hi_data	32	输入	写入 hi 寄存器的数据
10	lo_r	1	输入	是否读取 lo 寄存器的信号
11	lo_we	1	输入	是否写入 lo 寄存器的信号
12	lo_data	32	输入	写入 lo 寄存器的数据
13	rdata1	32	输出	地址一读取的数据
14	rdata2	32	输出	地址二读取的数据
15	hilo_data	32	输出	hilo 寄存器中读取出来的值

表 3 regfile 模块输入输出



原始的 regfile 模块的作用是确定 rs 寄存器以及 rt 寄存器的值，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 rdata2。

而为了增加 hi 和 lo 寄存器，我们新定义了两个 32 位寄存器，一个用于乘除法的高位 hi 寄存，一个用于乘除法的低位 lo 寄存。由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。

2.3 EX 模块

整体说明：

EX 段负责执行具体的算术运算、逻辑运算、内存访问地址计算，以及对 HI/LO 寄存器的操作。它接收来自 ID 段的指令信息、操作数和控制信号，输出结果并传递到 MEM 段。

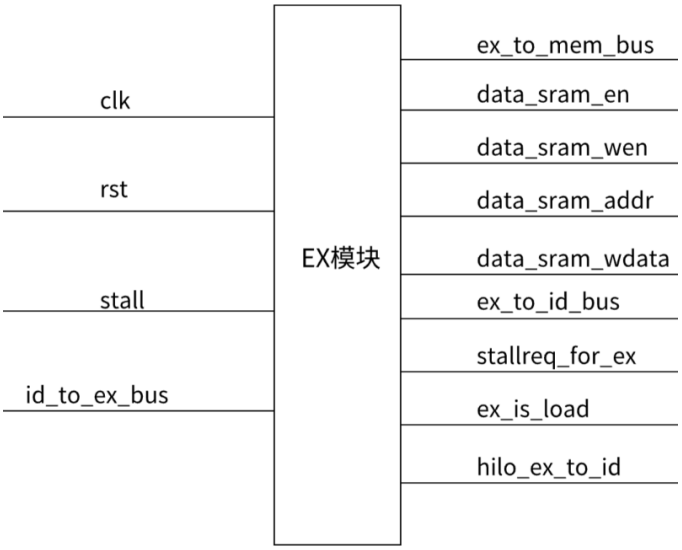


表 4 EX 模块输入输出

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	163	输入	ID 段传给 EX 段的数据
5	ex_to_mem_bus	5	输出	EX 阶段到 MEM 阶段的数据总线
6	data_sram_en	1	输出	数据存储器的使能信号
7	data_sram_wen	4	输出	数据存储器写使能信号
8	data_sram_addr	32	输出	数据存储器的地址信号
9	data_sram_wdata	32	输出	数据存储器的写数据信号
10	ex_to_id_bus	38	输出	返回给 ID 阶段的总线
11	stallreq_for_ex	1	输出	自 EX 阶段的流水线暂停请求信号
12	ex_is_load	1	输出	当前指令是否为加载指令
13	hilo_ex_to_id	66	输出	hi lo 寄存器的 forwarding

工作原理：

EX 段（执行阶段）是指令执行的核心，负责完成算术逻辑运算、访存地址计算以及特殊寄存器（HI/LO）的操作。通过 ALU 执行算术和逻辑操作，计算访存地址并生成内存控制信号，同时处理乘法、除法等复杂指令，通过 hi_we 和 lo_we 更新 HI/LO 寄存器。EX 段与流水线其他阶段协作，通过控制信号管理数

据冒险和控制冒险，最终将执行结果传递给 MEM 段并回传部分信息供后续使用。

在 ex 段解码 id 段传来的信息之后，我们根据 data_sram_en 判断是否能写内存，再根据 data_ram_readen 确定写内存操作的类型，也就是区分 sb、sh、sw 操作，再根据这三个操作来具体确定要写入内存的位置，其中我们通过 ex_result[1:0]来确定要写入的是第几个字节或哪几个字节。通过这步操作我们修改 data_ram_wen 的值，获得七种写内存的不同方式，它是一个四位宽的写使能信号，之后我们就可以根据它来修改 32 位宽的 data_sram_wdata 以控制写入内存中的数据。

在这一阶段我们还需判断当前指令是否为 lw 指令，修改 ex_is_load 的值，传给 ID 段从而判断下一个指令是否触发了无法用前递技术解决的数据相关问题。之后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送给 MEM 段。将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传回给 ID 段。这样让 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。

在这一段我们还接入了乘除法器，使用的是其自带的乘除法器，其中乘法器的实现，将乘法两个数根据当前是否为乘法指令设置为三十二位全零或当前 rdata1 和 rdata2，乘法符号标记就根据是 mult 还是 multu 来设置为 0 或 1 就行。而在 ex 段的暂停请求 stallreq_from_ex 则根据已定义 stallreq_for_div 来控制，若其为 1，则暂停下一条指令 ID 及其之后段落。Hi 和 lo 寄存器中的 data 直接根据乘除法的结果来设置就行，若是乘法则将 hi_data 设置为乘法结果的高 32 位，将 lo_data 设置为乘法结果的低 32 位，若只是移位指令，就将 rdata1 传入。

2.4 MEM 模块

整体说明：

MEM 段主要负责与数据存储器的交互，从 EX/MEM 流水线寄存器中得到地址读取数据寄存器，并将数据存入 MEM/WB 流水线寄存器。同时，它还将必要的数据回传到 ID 段。

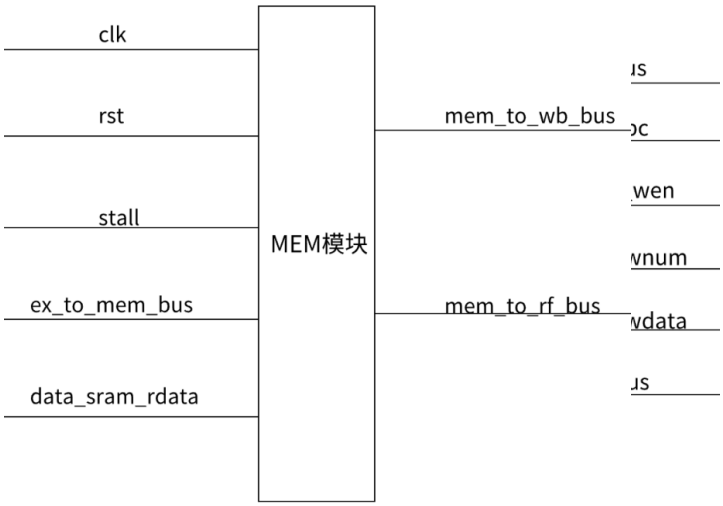


表 5 MEM 模块输入输出

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
6	mem_to_wb_bus	70	输出	MEM 传给 WB 段的数据
7	mem_to_rf_bus	38	输出	MEM 段传给 regfile 段的数据

功能说明：

在 MEM 段解码 EX 段传来的信息之后，我们根据 data_ram_readen 来确定操作是 lw,lb,lbu,lh,lhu 之中的哪个，再根据 ex_result 的前两位来确定读取内存中的第几个字节或哪几个字节，之后将其读取到 rf_wdata 传给 WB 段就大功告成了。

之后也将写使能信号还有寄存器地址和内容前递给 ID 段用于处理数据相关，这样我们的整体功能就差不多实现了。

2.5 WB 模块

整体说明：

WB 段是处理器流水线的最终阶段，其作用是将 MEM 段计算或访存得到的数据写回寄存器文件（rf），以完成指令的执行。同时，WB 段将写回数据提供给 ID 段，用于解决流水线中的数据相关问题。

表 6 WB 模块输入输出

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	debug_wb_pc	32	输出	用来 debug 的 pc 值
7	debug_wb_rf_wen	4	输出	用来 debug 的写使能信号
8	debug_wb_rf_wnum	5	输出	用来 debug 的写寄存器地址
9	debug_wb_rf_wdata	32	输出	用来 debug 的写寄存器数据
10	wb_to_id_bus	38	输出	用来返回给 ID 段

功能说明：

WB 段从 MEM 段接收数据，通过寄存器暂存后，根据流水线控制信号判断是否更新数据。随后，解析接收到的写回信号、目标寄存器地址和写回数据，将其打包输出给寄存器文件以完成写回操作。同时，WB 段生成调试信息（包括写回地址、数据和当前指令的 PC 值）输出供调试使用，并将写回信息回传到 ID 段，支持后续指令的正确执行。

2.6 CTRL 模块

整体说明：

CTRL 段是处理器流水线控制的核心模块，其主要作用是根据各流水线阶段的暂停请求生成控制信号，用于协调流水线的运行状态，防止数据冒险或其他异常导致的错误。CTRL 的暂停机制能够保证流水线的正确性和稳定性。

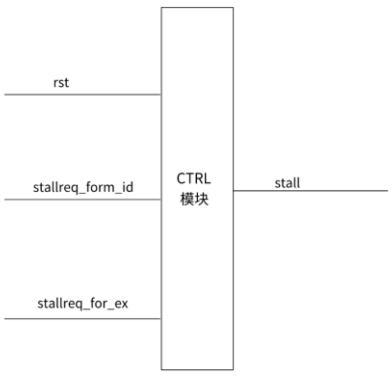


表 7 CTRL 模块输入输出

序号	接口名	宽	输入/输出	作用
1	rst	1	输入	复位信号
2	stallreq_from_id	1	输入	ID 阶段是否请求流水线暂停
3	stallreq_from_ex	1	输入	EX 阶段是否请求流水线暂停
4	stall	6	输出	暂停信号

功能说明：

stallreq_from_ex 判断 EX 段除法操作时是否请求暂停，如果请求暂停，那么就把 流水线暂停控制信号赋值为 6'b001111，表示将 ID 及其之后的流水线段落暂停。stallreq_from_id 判断 ID 段是否发生了无法通过前递技术解决的数据相关问题，一般是 lw 操作相关，如果请求暂停，那么就把流水线暂停控制信号赋值为 6'b000111，表示将 EX 及其之后的流水线段暂停。

3. 实验感受及收获

3.1 刘奕扬部分

在实验中我熟练的掌握了 GitHub 的使用，能够用它搭建仓库、审阅代码、管理版本，极大的提高了我们的工作效率。

在 debug 时，根据可能出错的位置，在波形图添加可能有问题的数值，查看各个段接收和传递的信息是否有错误，查看 stall 指令是否生效，一般通过对错误数据的追踪和朔源都能找到错误原因，但不得不说波形图看着头晕。

在代码编写的过程中，最常出现的问题就是在处理数据相关问题时，某条总线在上一段已经扩充并且加入了新数据，但是下一段解包时却没有将其接收，或是在 define 中没有将位宽扩展，导致 debug 时摸不着头脑。指令的添加根据 A03 文件中的指令规范来写还是很容易的，但最后的几个访存操作还是通过查阅网上的相关资料进行学习，采用了引入 data_ram_readen 这个写使能操作，从而在 EX 段\MEM 段进行写\读操作时能够区分操作并以不同方式写\读内存。

这次的实验使我的对流水线的理解有了很大提高，而且通过与团队成员的配合，我才能更好地完成这个任务，总的来说，这次实验让我受益匪浅。

3.2 刘策部分

本次实验我了解了五级流水线技术的整体架构，学到了每一段的功能即运行机制。初步掌握了 vivado 仿真模拟软件的使用方法，并且对于 Verilog 语言有了一定程度的熟悉。

本次实验我的主要负责暂停相关功能的实现，明白了流水线中的暂停机制。 还有乘除法器的接入。我负责的任务不多，但是还是加深了对于课堂学到知识的掌握。本次实验我查阅了大量网络相关资料，学习了许多类似的项目，对于我的信息检索能力，以及实践能力都有很大提升。

3.3 于济航部分

在实验中，我学会使用 github 来进行文件存储，这极大提升了代码的管理效率。

通过这次实验，我了解了流水线的工作流程，学会了一些关于 CPU 相关的知识，同时也学会了之前没接触过的内容，这次实验给我带来很多收获。

本次实验中我负责的任务不是很多，但是我也积极承担自己的任务，查阅了一些相关资料，并且在组员们的帮助下顺利解决了一些问题，这让我深刻体会到团队合作的重要性

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、龙芯杯官方的参考文档