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（未来实验班）

组长：刘冰昱

组员：刘昱昊

报告日期：2025.01.05

目录

一、小组成员工作量划分

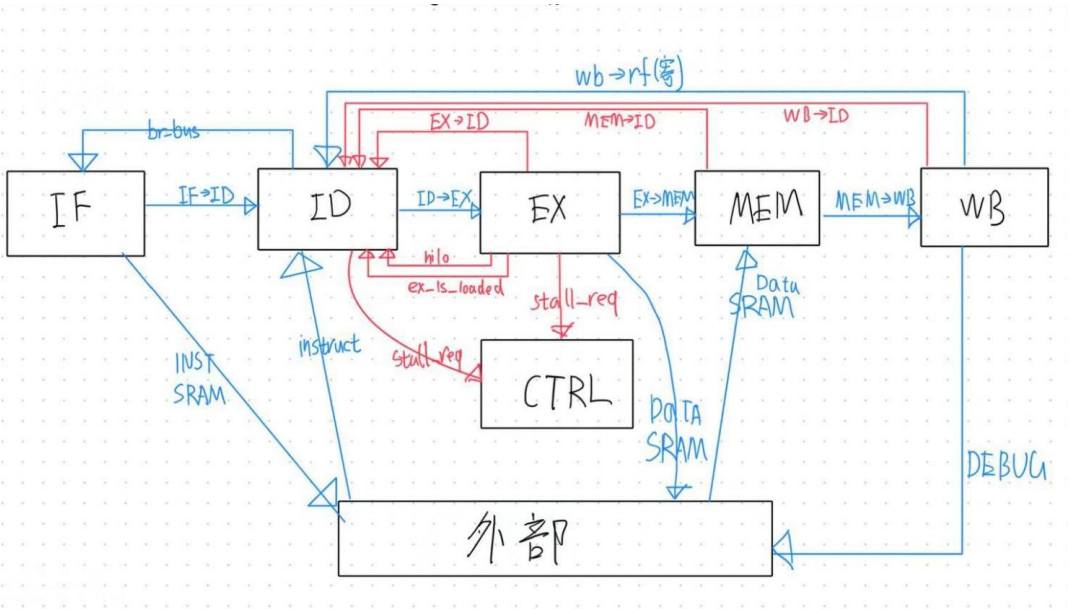
姓名	完成任务点	总任务量占比
刘冰昱	原始代码分析，流水线相关的问题的处理，stall 信号设计，ID 段代码补充	60%
刘昱昊	原始代码分析，EX 段代码补充，自制乘法器的实现	40%

二、总体设计

我们的流水线 CPU 设计包含七个核心模块：**IF 模块**、**ID 模块**、**EX 模块**、**MEM 模块**、**WB 模块**、**CTRL 模块**以及 cpu_core 的整体模块。该 CPU 具有五级流水，并合理设计了**数据前递（旁路）**和 stall 以防止相关问题。

该 CPU 搭载了 32 个 32 位的整数寄存器，采用大端字节序存储，拥有 32 位数据宽度和地址总线。通过一个自行设计了的 32 周期的 32 位移位乘法器，我们的流水线 CPU 能够顺利通过实验课程要求的 **64 个测试点**。

我们的流水线 CPU 的模块间的连接，如下图所示。其中，蓝色的为原始代码具备的连接，红色部分是我们自行添加的连接，用于解决相关问题：



图片 1 整体线路连接

如上图所示：

在 IF 模块（取指令阶段），主要任务是获取指令，并管理指令延迟槽。同时，它接受来自 ID 的 br_bus，解析其中可能蕴含的**跳转指令**，并负责确定下一条指令的地址。

ID 模块（译码阶段）负责指令的解码。我将其分为五部分：**指令译码、功能解析、访问寄存器与数据前递选择、分支处理、stall 请求。**在功能解析部分，将在此模块完成 EX、MEM、WB 的控制信息。在访问寄存器部分，将能调用 regfile 模块从通用寄存器中读取所需值，同时，也控制 hilo 寄存器的读取，并将回写阶段返回的寄存器地址及数据写入寄存器。分支处理，将会生成正确的分支使能信号和地址，传给 IF 段。

EX 模块（执行阶段）根据译码阶段提供的操作数和选定的 ALU 计算方式，计算出运算结果。对于类似 load 和 store 的指令，还会计算出相应的**内存地址**，并设定内存数据交互方式。

MEM 模块（访存阶段）负责判断即将写回寄存器的值是来自 ALU 的计算结果还是内存读取的结果，并将结果传递至回写阶段。

WB 模块（回写阶段）理论上是将运算结果存储到 CPU 的 32 个 32 位寄存器中。为了提升效率，将要写回的值和地址提前发送至译码阶段，在译码阶段读取寄存器值的同时，直接将值写入相应寄存器。

CTRL 模块负责控制流水线的暂停和清除等操作。

Cpu_core 模块，负责将内存读写的控制信号传出，将返回的结果传入。

实验环境：采用龙芯杯指定 XILINX 公司的 Vivado2019.2 为 仿真运行的工具和 FPGA 综合工具，同时使用 VS Code 作为 Vivado2019.2 的默认代码编辑器，使用 git 完成小组内部的协同开发和代码同步。

三、流水线各个阶段的说明:

1、IF 模块

IF 模块功能：取指令，并生成 next_pc，处理跳转指令

IF 模块接口：

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	分支跳转指令的信号，控制是否跳转
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_addr	33	输出	指令寄存器的地址，用来寻找指令的存放的位置
9	inst_sram_wdata	33	输出	指令寄存器的写数据，本题中不被使用

IF 模块细述：

其内容的总体设计图，如下图 2 所示：

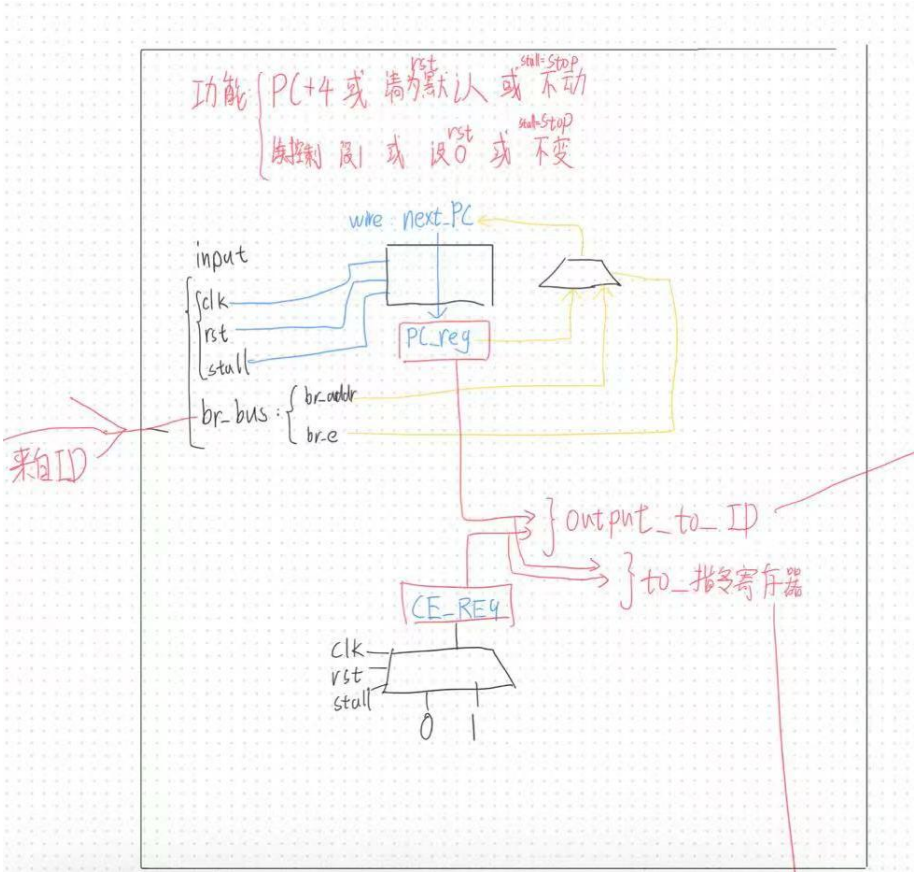


图 2 IF 段整体设计图

在 IF 段，主要功能由两部分：**设置当前指令 PC，根据分支设置 next_PC**。

当接收到时钟信号和复位信号时，若复位信号有效，则将程序计数器 pc 的值设置为复位值。由 CTRL 段发出的输入信号 stall，其主要功能是在流水线暂停期间，如果 IF 段识别到该信号，就不更新 PC 寄存器的值，确保下一条指令的 pc 值与当前 pc 值相同，实现 IF 段的时钟周期暂停。在无暂停的正常运行情况下，reg_pc 的值被设置为当前的 next_pc 值，

ID 段发出的 br_bus 信号指示是否需要跳转以及跳转的目标地址。如图中黄色线条（连接到 mux 的线条），在 IF 段，一旦检测到跳转需求，next_pc 的值将被调整为跳转目标地址，否则，则设置为原值加 4。

最终，将 reg_pc 的地址通过 cpu_core 发送至指令内存，获取相应的指令值并传递给 ID 段。

2、ID 模块

ID 模块功能：对指令进行译码，形成 ALU、DATA_SRAM、REG 的控制和信号选择，将结果传给 EX 段。与寄存器进行交互，实现寄存器的读写。处理数据回传，处理可能的跳转语句，并将跳转传给 IF 段。

ID 模块接口：

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 段发出的暂停信号

ID 模块详述：

该部分的整体规划如下图所示：

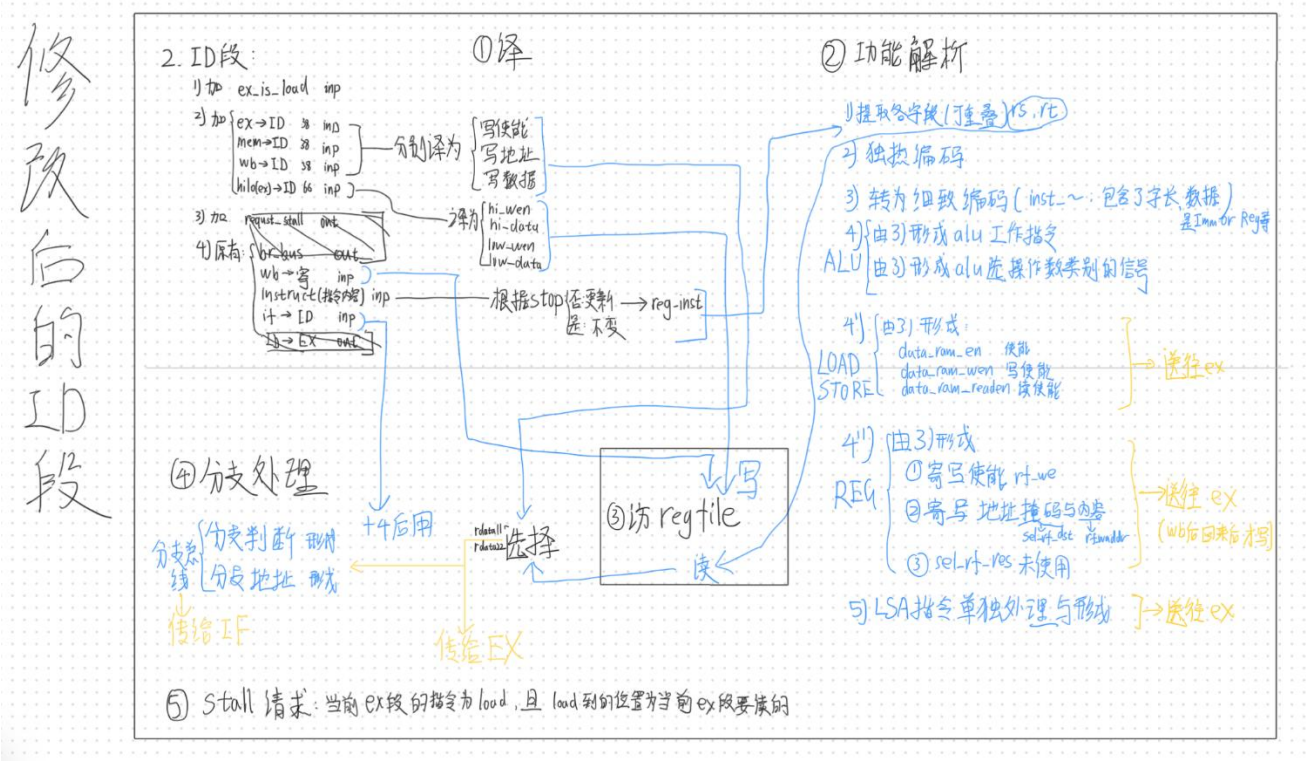


图 3 ID 模块整体规划

ID 模块（译码阶段）负责指令的解码。我将其分为五部分：**指令译码、功能解析、访问寄存器与数据前递选择、分支处理、stall 请求。**

指令译码部分，主要功能是，将得到的指令码解析出**各个可能的字段**，这些字段甚至有重叠，然后，将部分字段进行**独热编码**。此外，还对回传数据解析为**使能、寄存器序号、存入内容**。

在功能解析部分，主要使用上一步译码出来的指令部分。在这部分，将会 1. 根据指令内容，解析出 **alu 指令**所需要的具体的 op 算符以及其两个操作数分别使用的内容选择信号，如使用 PC、立即数、或寄存器内容。 2. 根据指令内容可能存在的**访存操作**，例如 load 或 store 等，将解析出使能、写使能、读使能三个信号。 3. 将根据指令内容判断，此条指令是否有**寄存器写操作**，将生成寄存器写使能信号以及写地址的信号来源。

在访问寄存器和数据前递选择部分。分为两个部分，第一，**访问寄存器**，第二，**处理数据前递**。将调用 regfile 模块从通用寄存器中读取 rs、rt 值，并将 wb 阶段返回的数据写入寄存器；同时，也控制 hilo 寄存器的读取。在处理数据前递部分，将会根据 **EX、MEM、WB 的从前到后的顺序**，判断是否与 rs、rt 冲突，选择数据信号。

分支处理部分，将会判断当前指令是否是分支指令，如果是的话，将会直接判断分支结果是否为 true。并将**分支使能以及计算出来的分支地址**返回到 IF 段。

Stall 请求部分，是为了**处理数据相关**的问题。如果 ex 段当前的指令为 load，并且 load 到的地址为当前 ID 段所取的内容，那么说明发生了**读后写相关**。此时 ID 段将向 ctrl 发出一个 stall 请求，从而暂停流水线。

3、EX 模块

EX 模块功能：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，执行乘除法。

EX 模块接口：

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

EX 模块细述：

设定以下变量：

在EX段，主要完成的工作为：**1.计算 ALU 的结果；2.根据当前指令，确定即将要内存的读写；3.执行乘除法以及相应stall。**

第一部分，计算ALU结果。这个部分相对简单，其操作为根据ID段传入的ALU操作符以及ALU操作数选择信号，首先进行ALU两个**操作数信号的选择**，以及操作数的零扩展或者符号扩展。接着将操作符以及选择到的内容传给alu模块即可得到ALU结果。

第二部分，根据当前指令确定即将要进行的内存读写。在这部分将根据传入的数据读使能或写使能信号，首先判断读取的字节数。由于CPU的内存采用了**四个字节的对齐。**因此将根据alu运算的结果后2位，选择四个字节中，第几号字节被读或者写，从而形成相应的写使能以及写数据的设置。此外，将**ALU的运算结果**作为内存的写入地址。

第三部分，执行乘除法以及设置对应的stall。在这部分为了在这部分为了实现节省流水线的运行时间我们采用了自己写的乘法器。乘法和除法分别将占用多个时钟周期，因此，在乘法和除法执行的过程中，将向ctrl发出stall请求信号，以暂停流水线的运行。

乘除法指令的实现：

乘法器的具体实现见下文的“6 MUL模块”。

除法指令的实现：

除法指令的实现与乘法类似，先判断是符号除法还是无符号除法，再判断除法器是否空闲，若可用，则传入被除数和除数，并将 stallreq_for_div 设为 `Stop 以暂停流水线。除法结束后，结果存于 div_result 中，并恢复流水线运行。乘除法结束后，将结果存入 hilo 寄存器。首先将 result 的高32位和低32位分别存入 hi 和 lo，然后将要写入的值和写使能信号直接发送给 ID 段。由于 MEM 和 WB 段不涉及 hilo 寄存器的读写，直接将 hilo 写入信号发送给 ID 段，不同于通用寄存器那样传递到 WB 段再发给 ID 段。至此，乘法和除法指令已能正常执行。

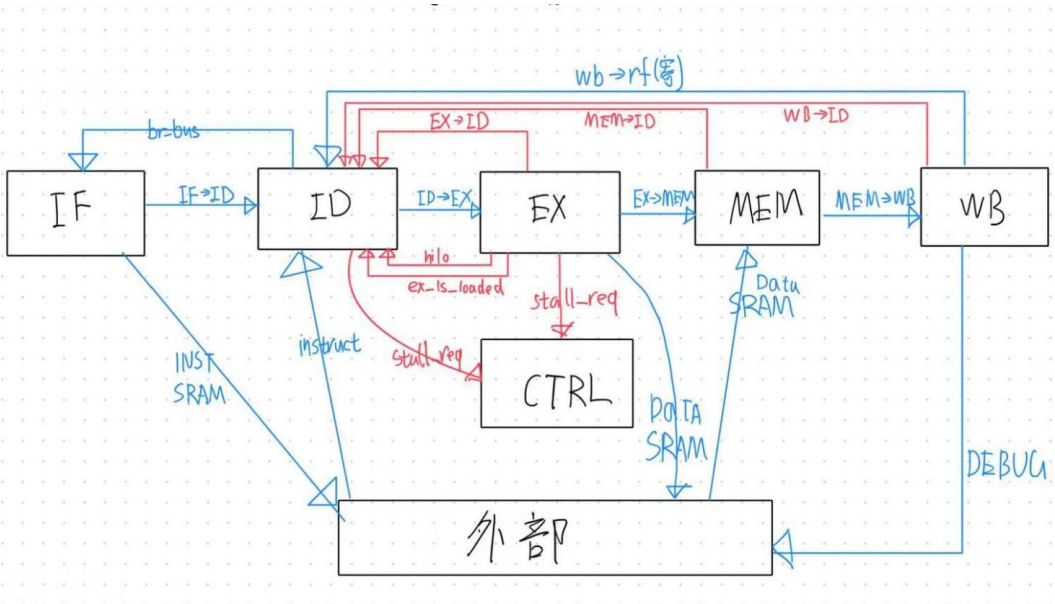
4、MEM 模块

MEM 模块功能：读取内存中相应地址的值（其实是在 EX 到 MEM 的过程中读取的），根据当前指令，确定要写入寄存器的值。

MEM 模块接口：

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

MEM 模块细述：



如上图所示。MEM 段接收到的输入一共有两个：第一，是由 ex 段传入的，其主要使用内容为内存读取的使能信号；第二，是由 ex 段向内存请求数据，并从 cpu_core 中向 MEM 段传入的具体的内存内容。

该部分所做的主要工作是，根据 ex 段传入的内存读取信号，**选择将写回寄存器的写入内容**。其主要代码如下图所示：

```
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;
```

上面代码的选择逻辑是，根据**内存访问的类型**（如字节、半字、字等）和**地址对齐方式**，从内存读取的数据“data_sram_rdata“中选择正确的部分，并进行符号扩展或无符号扩展，最终生成 32 位写回数据 rf_wdata。正如前文所说，因为内存的**对齐方式是四个字节**，所以一次必定读取四个字节的内存数据，也因此才需要这部分处理。具体来说，通过 data_ram_readen 判断访问类型（如 4’b1111 表示字，4’b0001 表示字节等），结合 ex_result[1:0]确定地址对齐方式（如 2’b00 表示对齐到字节 0），从 data_sram_rdata 中提取相应的字节或半字，并根据访问类型进行符号扩展或无符号扩展，生成最终的 32 位数据；如果当前不是内存访问操作，则直接将执行阶段的结果 ex_result 写回寄存器文件。

5、WB 模块

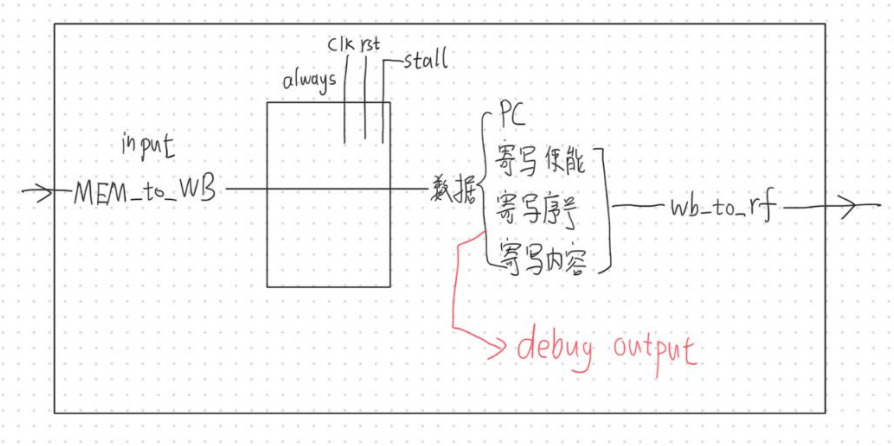
WB 模块功能：将结果写入寄存器（实际是传给 ID 段调用 regfile 模块），设置 debug 信号。

WB 模块接口：

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	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

WB 模块细述：



如上图所示，在每个模块都存在一个流水控制器，其接收重置信号或者暂停信号，从而实现暂停或机器重启。

由于我们前面在 ID 端实现了寄存器的写入，使得 WB 段功能比较简单。其主要功能仅为提取出从 MEM 段传入的数据，并将其中的寄存器写使能信号、写的寄存器序号、写寄存器的内容，提取出来并打包发给 ID 段，ID 段将会完成对寄存器的写入。

此外，该部分提供调试信息：输出 WB 阶段的 PC 值、写使能信号、写地址和写数据，用于调试和监控流水线状态。

6、MUL 模块

MUL 模块功能：实现两个 32 位有符号或者无符号数的乘法，并返回。

MUL 模块原理：先把被乘数扩展成 64 位，再左移 32 位，每一次左移之前都要判断乘数最低位是否为 1，为 1 则把那一步的被乘数加到结果里（result），为 0 则不做处理，于此同时将乘数右移一位。

MUL 模块总述：

1. MulFree（乘法器空闲状态）

功能：这是乘法器的初始状态，表示乘法器处于空闲状态，等待新的乘法运算。

操作：

如果接收到 start_i 为 MulStart，表示需要开始一次新的乘法运算。进入 MulOn 状态，初始化相关的寄存器：

根据 signed_mul_i 判断是否是有符号乘法,如果是则对输入的 a_o 和 b_o 做符号扩展处理（如果是负数，则取补码处理）。

将操作数 a_o 和 b_o 分别存入临时变量 temp_opa 和 temp_opb，然后将 ap（扩展后的被乘数）和 pv（乘法的中间结果）初始化。

清除结果寄存器 result_o，并设置 ready_o 为 MulResultNotReady，表示运算尚未完成。

状态转换：当 start_i == MulStart 时，进入 MulOn 状态。

2. MulOn（乘法运算进行中状态）

功能：这是进行乘法运算的状态，表示乘法器正在进行计算。

操作：

每个时钟周期，乘法器根据 i 的值（控制乘法的位数）逐位执行乘法操作。

如果乘数 temp_opb 的最低位为 1，则将乘积加到累加器 pv 上；然后将 ap 左移一位，temp_opb 右移一位。

i 从 0 开始，每次增 1，直到达到 32 位（即 i==6'b100000）。

如果乘法完成，即 i == 6'b100000，检查乘法是否为有符号运算，如果是并且符号不一致（异号），则对结果 pv 取反加 1（补码操作）表示负数的结果。

进入 MulEnd 状态，表示乘法运算已经完成。

状态转换：在每次时钟周期，i 自增，逐步执行乘法操作，直到所有的位都处理完为止。当 i 达到 32 位（i == 6'b100000）时，进入 MulEnd 状态。

3. MulEnd（乘法运算结束状态）

功能：这是乘法器的运算结束状态，表示乘法运算已经完成，可以输出结果。

操作：

将最终的乘法结果 pv 输出到 result_o 寄存器。

设置 ready_o 为 MulResultReady，表示运算结果已经准备好。

如果 start_i == MulStop,表示乘法器完成后的停止信号,此时会回到 MulFree 状态，清除结果和准备状态。

状态转换：一旦乘法完成，result_o 输出运算结果，ready_o 设置为 MulResultReady。如果接收到 start_i == MulStop，则进入 MulFree 状态，准备接收下一个乘法运算。

4. **整体状态转移图：**

MulFree → 如果接收到 start_i == MulStart，转到 MulOn。

MulOn → 一旦完成乘法操作（i == 6'b100000），转到 MulEnd。

MulEnd → 如果接收到 start_i == MulStop，转回 MulFree。

四、组员感受以及改进意见

刘冰昱感受以及改进意见：

这次实验，是我第一次进行的计算机系统的相关实验，也是我第一次接触 verilog 和 vivado 软件，以进行硬件级的程序编写。在这次实验中，我将课堂上所学的五级流水的知识运用到了实践当中，现在完成了对五级流水 cpu 的代码的修改。使我对 CPU 和现代计算机知识有了更深刻的认识。

在这次实验中我遇到了很多困难。例如：不知道如何进行仿真、不会 verilog 语言语法、不会配置 verilog 语言的环境、不知道程序运行的结果如何输出等等很多很多问题。通过自身不懈的学习，以及向同学学长进行求助、参考网上资料，最终解决了这些问题，完成了第一个 CPU 的搭建。这个经历对我而言十分宝贵，使我有信心在短期时间内接触并完成一个新的领域的工作，并且增强了我团队协作的能力。

事实上，在我看来。他对该项目的代码内容并多，代码函数也比较少，但是其代码的结构异常复杂、困难。这启迪我:代码的设计，比代码的编写更要重要，也要困难很多。这教会了我，在研究代码时，先研究其结构以及设计。在此之上，才能对一个代码进行很好的改写、添加内容。

最后，对于这门课，我认为其可能需要改进的点是：课时时间太短，不够完成这项工作;在实验课开始前没有入门引导,对于没接触过硬件的我而言,github 中给的文档内容比较概括，相较于教程，更类似于说明性文档，看完仍然不知道做什么。

刘昱昊感受以及改进意见：

在本次实验中，我将课堂所学知识应用于实践，对流水线的运作有了更深入的认识。

在构建流水线 CPU 的过程中，由于经验不足且需自行设计许多环节，一些看似简单的问题因思维不够周密，在执行时遭遇了不少难题。这次实验显著提升了我对流水线的理解，并让我更深刻地认识到团队合作和整体架构的重要性。要在短时间内完成大量任务，团队合作是必不可少的。为了确保团队合作的顺利进行，采用模块化设计方法和合理安排软件结构是关键。总体而言，这次课程设计加深了我对理论知识的理解，提高了我的编程技能，并让我在团队合作方面获得了丰富的经验，收获颇丰。

五、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、（美）DavidA.Patterson、John L.Hennessy 著 《计算机组成与设计：硬件、软件接口（原书第 4 版）》

4、Yale N.Patt 著 《计算机系统概论（原书第 2 版）》

4、助教给出的龙芯杯官方的参考文档