Lab 4 报告

学号 2021K8009929033
姓名 马迪峰
箱子号 5

一、 实验任务（10%）

本次任务分为三个阶段，完成 20 条指令的单周期 CPU 之后，将其改造为单发射五级流水。

- 在单周期 CPU 基础上引入流水线，但不考虑处理各类相关所引发的冲突。
- 了解流水线冲突原因及冲突类型，并利用阻塞的方式解决冲突。
- 了解流水线冲突的解决方式，并利用流水前递技术改造 CPU 减少阻塞带来的损失。

二、实验设计（40%）

（一）总体设计思路

1. 实验验证框架

只有单独的一个 CPU，没有完整的验证模拟环境，是无法实现 CPU 的功能的。本实验提供了 CPU_CDE 开发环境，其内包含金标准及运行结果 golden_trace，测试程序 func 和需要进行开发的 mycpu_verify。实验中，CPU 的验证过程就是将其与已知功能正确的 CPU 的运行结果逐步对比，观察是否一致。。

在硬件方面，CPU 被封装在了片上系统 SoC 上。该 SoC 系统包括一个 CPU 核、一个指令 ram、一个数据 ram、一个配置寄存器和一个 bridge。其中，CPU 与指令 ram 直接相连，CPU 通过 bridge 与数据 ram、confreg 相连；指令 ram 存放指令，数据 ram 存放数据；confreg 实现与 FPGA 引脚的相连，进而实现与实验箱开发板上的数码管、LED 灯、按键等 I/O 设备相连。

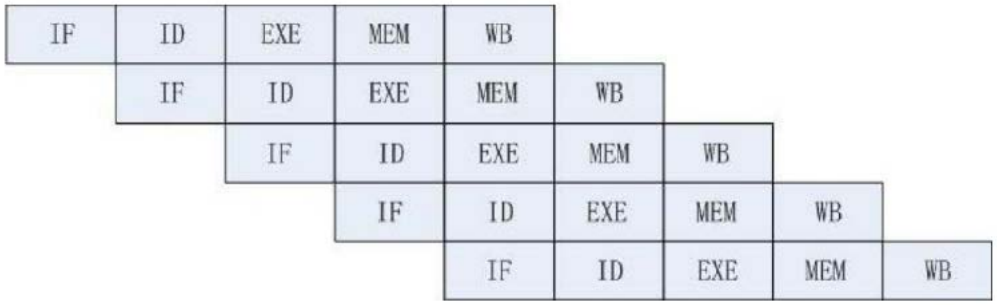
2. 硬件设计概览

在 Lab3 实现的简单 CPU 基础上，本次实验要引入流水线，并分别采用阻塞技术和前递技术处理指令相关冲突。设计思路依然是先确定数据通路和控制逻辑，即明确流水线的划分方式（5 个阶段）、不同阶段间的数据传递方式、阶段转换的控制方式和相关冲突的检测及解决方式，实现一个时钟频率较高、在尽可能少地牺牲性能的前提下能解决各种指令冲突的简单流水线 CPU。

值得注意的是，指令存储器和数据存储器模块需从 Lab3 的异步 RAM 改为同步 RAM，以实现流水线的功能。指令 RAM 采取在 PC 更新阶段开始读的方案，即把 PC 分为两部分：生成 nextPC 和把 nextPC 送到对应的寄存器中，第一步完成后就开始指令 RAM 读操作；后者同理，采取在执行阶段发出读写请求的方案。对应地修改电路结构中的数据通路即可。

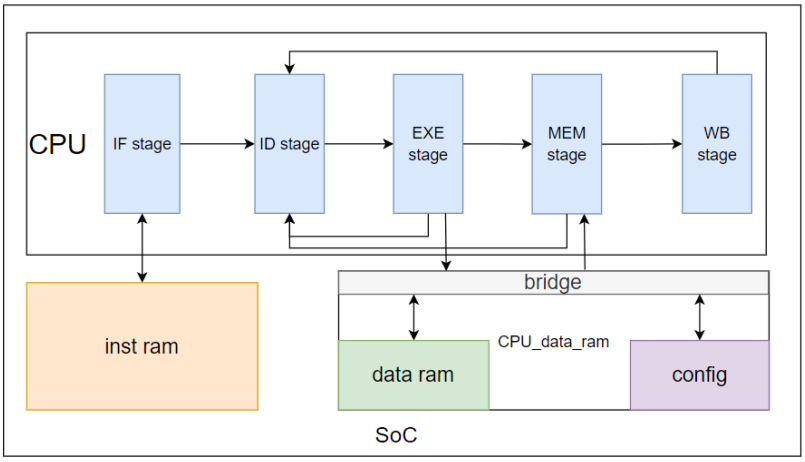
流水线的核心思想是把多条指令的不同执行阶段重叠起来，使得 CPU 能同时处理多条指令，处于不同流水级的指令使用不同的物理器件，借此可以缩短时序器件之间组合逻辑关键路径的时延，以求更高效地利用硬件资源。 在流水线 CPU 中，每条指令的执行过程被分成若干个执行阶段，本实验中将之切分为五个流水级，分别为 IF、ID、EXE、MEM、WB 阶段。

IF 阶段负责取指，ID 负责译码和读写寄存器，EXE 负责运算执行和向数据 ram 发送读写信号，MEM 阶段负责接收来自数据 ram 的数据，WB 阶段负责写回寄存器堆并提交 debug 的相关信号。在各级流水线之间既有控制信号的传输，又存在数据通路来传递数据，每个流水级内部都设置寄存器来临时存放当前执行所需的数据和信号。

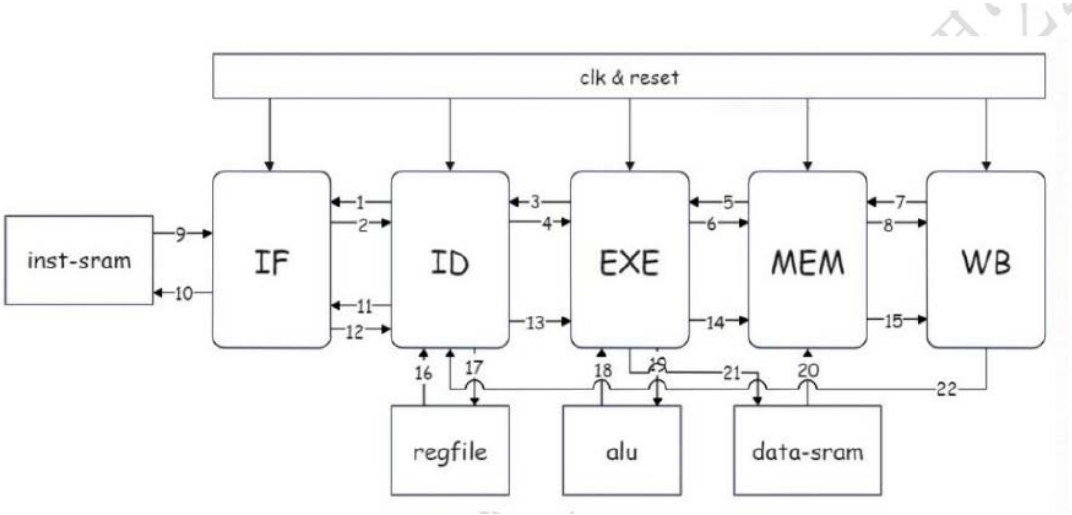


图表 1 五级流水线时空图

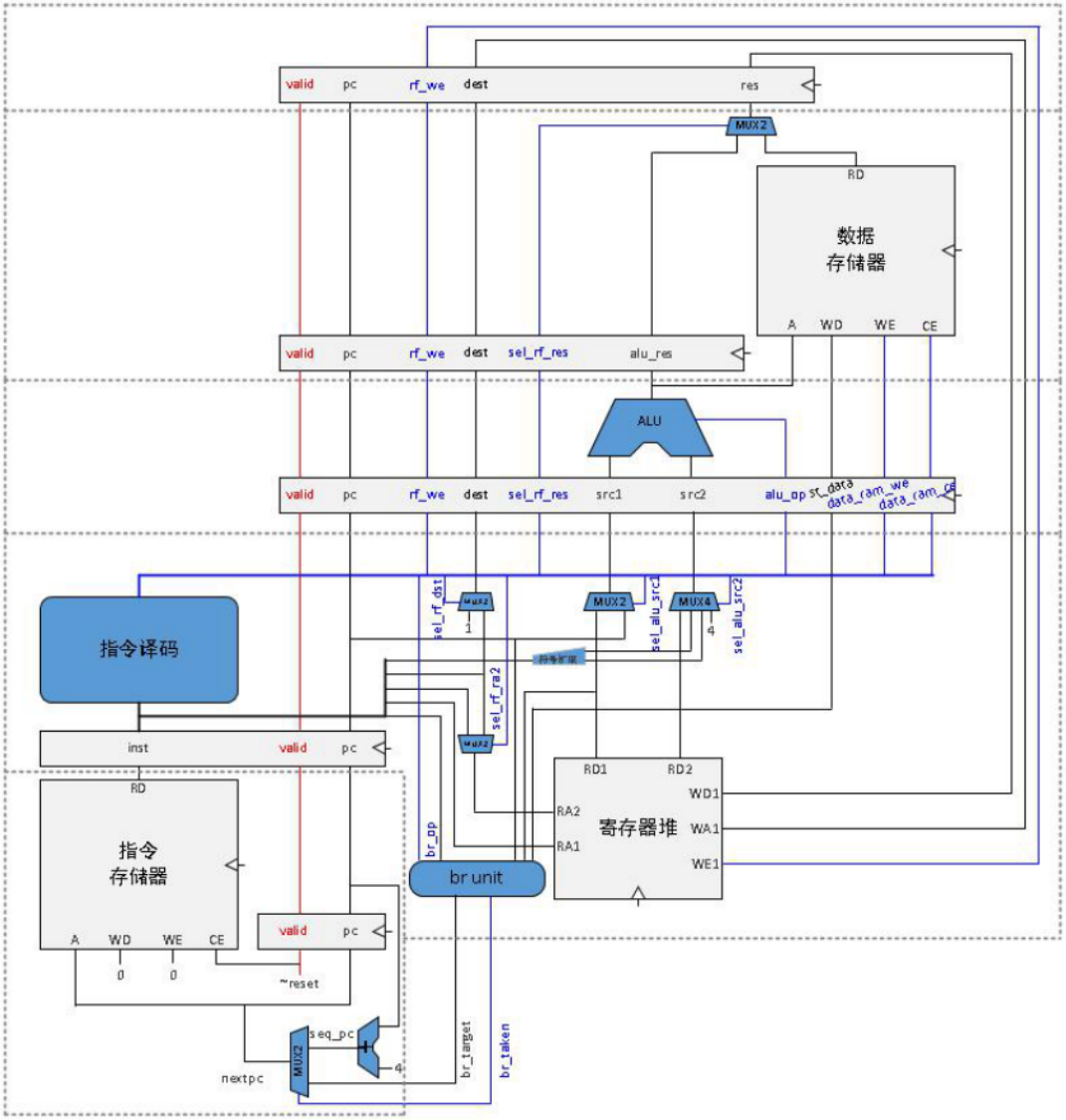
本实验所设计的单发射五级流水 CPU 的总体框架如下：



图表 2 SoC 片上系统示意图



图表 3 流水线 CPU 各模块间数据传递的示意图



图表 4 my_CPU 总框架

3. PC 的数据通路

在完成原本数据通路的拆分、实现五级流水间数据内容的传递后，还需要调整更新 PC 的数据通路，包括转移指令更新 PC 的通路和复位更新 PC 的通路。

因为转移指令的跳转目标在译码阶段就可以算出（目标信息来自指令码和寄存器堆），所以通过 br_control 传递来的信号，在 IF 阶段就可以完成 nextpc 的修改（同时还要修改控制信号），以此来尽可能减少控制相关造成的流水线阻塞。

除此之外，还要调整复位更新 PC 的通路，把在复位信号有效期间赋给 PC 寄存器的值改为 0x1BFFFFFC，使复位撤销后第一个发出的取指请求的地址是 0x1C000000。

```
always @(posedge clk) begin
    if(~resetn)
        IF_pc <= 32'h1BFF_FFFC;
    else if(IF_readygo & ID_allowin)
        IF_pc <= nextpc;
end
```

图表 5 PC 通路的设计

4. 阻塞与前递的实现

在 exp8 中，需要解决写后读（RAW）数据相关时的流水线阻塞。在具体实现上，EXE、MEM、WB 阶段都要向 ID 阶段提供自己当前的寄存器堆写使能信号 wen、写地址信号 waddr 和有效位 valid，然后在 ID 阶段判断是否发生数据相关，如果发生，则将 ID 阶段的 readygo 置 0，此时后面的流水线级每一拍从 ID 接收到的数据都是无效的。当无效信号传递到 WB 级后，ID 的阻塞会被解除，可以从异步读的 regfile 中读出刚被写入的数据。

在 exp9 中，添加前递技术，缓解了流水线阻塞所导致的性能损失。在具体实现上，首先需要扩充 EXE 和 MEM 阶段向 ID 前递数据的接口，分别将 EXE 阶段的 alu_result 和 MEM 阶段的 final_result 送至 ID 级。ID 级在判断对应的数据相关发生后，不再统一处理为阻塞，而是根据阻塞的来源，检查是否可以将 EXE、MEM 和 WB 送来的数据中的某一个作为 regfile 的读出值，而不用等待 WB 阶段将所需数据写入 regfile 后再在下一拍读出。考虑到指令的先后顺序，ID 级在接收到来自 EXE、MEM 和 WB 阶段的前递数据时，接收的优先级是 EXE > MEM > WB。因为当前指令一旦执行到 ID 阶段，说明上面三条在 EXE、MEM 和 WB 阶段的指令都经过了 ID 阶段并获得了正确的操作数，假设它们依次结束执行，最终寄存器中写入的值便是按照 EXE、MEM、WB 的顺序依次覆盖的。

5、控制逻辑

（1）握手信号

用于控制流水线各阶段的“流动”，主要包括各阶段的 valid 信号、readygo 信号、XXReg_valid 信号和 allowin 信号。valid 信号用来判断当前阶段是否有效；readygo 信号用来判断当前阶段进行的操作是否能在一拍内完成；若当前阶段有效且 readygo 信号拉高，则该阶段的 XXReg_valid 信号拉高；若当前阶段的 readygo 信号拉高且下一阶段的 allowin 信号拉高，或当前阶段的 valid 信号为 0，则当前阶段的 allowin 信号拉高。

只有在当前阶段的 allowin 信号和上一阶段的 XXReg_valid 信号都拉高时，当前阶段才能接收上一阶段传来的控制内容和数据内容。

（2） 冲突检测信号

判断处于 XX 阶段的指令的寄存器写地址与其下一条指令的寄存器读地址 1 是否一致
判断处于 XX 阶段的指令的寄存器写地址与其下一条指令的寄存器读地址 2 是否一致

```
assign block_r1_exe = (|rf_raddr1) & (rf_raddr1 == EXE_rf_waddr) & EXE_rf_we;
assign block_r2_exe = (|rf_raddr2) & (rf_raddr2 == EXE_rf_waddr) & EXE_rf_we;
assign src1_addr          = ~src1_is_pc & (|alu_op);
assign src2_addr         = ~src2_is_imm &(|alu_op);

assign rj_value  = (|rf_raddr1) & (rf_raddr1 == EXE_rf_waddr) & EXE_rf_we? EXE_rf_wdata:
                   (|rf_raddr1) & (rf_raddr1 == MEM_rf_waddr) & MEM_rf_we? MEM_rf_wdata:
                   (|rf_raddr1) & (rf_raddr1 == WB_rf_waddr) & WB_rf_we ? WB_rf_wdata : rf_rdata1;

assign rkd_value =  (|rf_raddr2) & (rf_raddr2 == EXE_rf_waddr) & EXE_rf_we? EXE_rf_wdata:
                   (|rf_raddr2) & (rf_raddr2 == MEM_rf_waddr) & MEM_rf_we? MEM_rf_wdata:
                    (|rf_raddr2) & (rf_raddr2 == WB_rf_waddr) & WB_rf_we ? WB_rf_wdata : rf_rdata2;
```

值得注意的是，还要添加控制信号 src_addr1 和 src_addr2，用来判断哪些指令用到了寄存器读地址 1 中的数据，哪些指令又用到了寄存器读地址 2 中的数据，并将其作为其他冲突检测信号的产生因子。否则，处理器会在无需阻塞和前递的时候进行阻塞或前递，造成性能的下降。

（二）重要模块 1 设计：IF 模块

1、工作原理

IF 模块需完成取指和更新 PC 的作用。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
resetn	IN	1	复位信号
ID_allowin	IN	1	ID 模块允许 IF 模块传入数据
br_control	IN	33	branch 相关数据，包含跳转信号第 1 位和跳转地址后 32 位
IDReg_valid	OUT	1	标记 IF 模块向 ID 模块传递的数据是否有效
IDReg_bus	OUT	64	IF 模块向 ID 模块传递的数据
inst_sram_en	OUT	1	指令 ram 片选信号
inst_sram_we	OUT	4	指令 ram 写使能
inst_sram_addr	OUT	32	指令 ram 地址信号
inst_sram_wdata	OUT	32	指令 ram 写数据
inst_sram_rdata	OUT	32	指令 ram 读数据

3、功能描述

在 pre-IF 阶段完成 pc 的更新，在 IF 阶段接收从指令 RAM 中读出的指令 inst，并通过 IDReg_bus 把 pc 和 inst 传到 ID 阶段。更新方式是在每个时钟上升沿，根据 IF 阶段是否可以读进值(IF_allowin)和跳转使能 br_taken 是否拉高来确定 next_pc 的值，并更新 pc 为 next_pc。

三）重要模块 2 设计：ID 模块

1、 工作原理

利用 IF 传来的指令，进行译码并生成各种控制信号；将译码信号传递给 EXE 模块；根据译码结果进行 branch 的判断并将有效信号、跳转地址传递给 IF 模块；处理 regfile 的写回。ID 阶段对 IF 送来的指令进行译码操作，确定当前指令类型并生成后续流水级所需控制信号，如 ALU 的操作码、操作数等。同时，ID 级在判断 ALU 操作数时还需注意是否存在数据相关的情况，根据后续流水级传递来的寄存器写地址、写有效信号、写数据判断最终有效的数据；需要注意的是，遇到访存指令+寄存器指令的数据相关时，需要阻塞一周期。阻塞是通过控制 ID_readygo 的信号实现的，其与 EXEReg_valid 和 ID_allowin 相关，其拉低可确保流水线被阻塞。

2、 接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
reset	IN	1	复位信号
EXE_allowin	IN	1	EXE 模块允许 ID 模块传入数据
ID_allowin	OUT	1	ID 模块允许 IF 模块传入数据
IDReg_valid	IN	1	标记 IF 模块传入 ID 模块的数据是否有效
IDReg_bus	IN	65	IF 模块传入 EXE 模块的数据
EXEReg_valid	OUT	1	标记 ID 模块传入 EXE 模块的数据是否有效
EXEReg_data	OUT	149	ID 模块传入 EXE 模块的数据
br_control	OUT	33	branch 相关数据，包含跳转信号和跳转地址
WB_reg_control	IN	38	WB 模块向 ID 模块传递的 regfile 写回信息和前递数据
EXE_reg_control	IN	39	EXE 模块前递到 ID 模块的数据
MEM_reg_control	IN	38	MEM 模块前递到 ID 模块的数据

3、 功能描述

ID 阶段首先将 IF 传来的指令通过译码部件，得到指令操作码、立即数、寄存器号等信息。然后根据指令操作码，生成各种控制信号，之后再根据控制信号选择数据通路。这些控制信号和数据会根据指令功能由不同的部件生成进一步的信号，比如控制 IF 级 PC 跳转的 br_control，控制流水线是否阻塞的 hold 信号，以及传递给 EXE 级的信号

EXEReg_bus。

在 exp7 中设计的 WB_reg_control 中已经包含 WB 阶段的写地址和写数据，因此可以直接进行修改和使用。实现阻塞的方式是：当 hold 信号为 1 时，ID 阶段的 readygo 置 0，因此 ID_allowin 也会被置 0，阻止前面的流水级继续传递信号。同时 EXEReg_valid 也会变为 0，防止后续流水级执行 ID 阶段的指令。

exp9 中实现了数据前递技术，将后续流水级中传递的数据通过旁路直接传送到 ID 阶段，因此降低阻塞的时长。但需要注意的是，如果上一条指令是 load 指令，那么发生数据相关的下一条指令需要从内存中获得读数据，即需要该指令在 MEM 阶段前递的数据，因此仍需阻塞一拍。此外，当 ID 与前面多条指令发生数据相关时，应该按照优先级 EXE > MEM > WB 的顺序互斥地选择数据来源。因为当前指令一旦执行到 ID 阶段，说明上面三条在 EXE、MEM 和 WB 阶段的指令都经过了 ID 阶段并获得了正确的操作数，假设它们依次结束执行，最终寄存器中写入的值便是按照 EXE、MEM、WB 的顺序依次覆盖的。

（四）重要模块 3 设计：EXE 模块

1、 工作原理

根据传来的控制信号和数据进行运算；向数据 ram 发送读写请求；执行各条指令的具体任务。

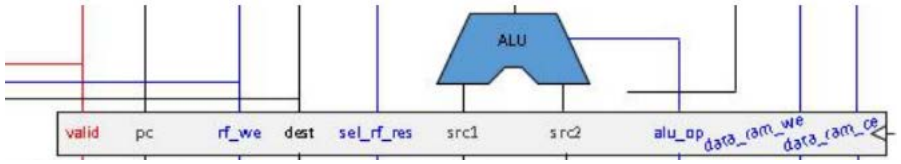
2、 接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
reset	IN	1	复位信号
EXE_pc	OUT	32	EXE 阶段所执行指令的 PC 值
MEM_allowin	IN	1	MEM 模块允许 EXE 模块传入数据
EXE_allowin	OUT	1	EXE 模块允许 ID 模块传入数据
EXEReg_valid	IN	1	标记 ID 模块传入 EXE 模块的数据是否有效
EXEReg_bus	IN	149	ID 模块传入 EXE 模块的数据
MEMReg_valid	OUT	1	标记 EXE 模块传入 MEM 模块的数据是否有效
EXE_reg_control	OUT	39	EXE 模块前递到 ID 模块的数据
data_sram_en	OUT	1	数据 ram 片选信号
data_sram_we	OUT	4	数据 ram 写使能
data_sram_addr	OUT	32	数据 ram 地址信号
data_sram_wdata	OUT	32	数据 ram 写数据

3、 功能描述

结合控制信号、数据进行指令的执行。EXE 阶段从 ID 阶段取得有效操作数和操作

码后，进行算术逻辑运算。同时 EXE 需传给 ID ,res_from_mem 和 mem_we 信号判断是否需要拉高数据寄存器的使能信号和写使能信号，其中读写地址有 alu 计算结果给出，写数据由 rkd_value 给出。其读使能信号的拉高是为后续 MEM 阶段做准备。



（五）重要模块 4 设计：MEM 模块

1、 工作原理

接收数据 ram 返回的读出数据，根据控制信号生成最终的写回数据。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
reset	IN	1	复位信号
EXE_pc	IN	32	EXE 模块所执行指令的 pc
WB_allowin	IN	1	WB 模块允许 MEM 模块传入数据
MEM_allowin	OUT	1	MEM 模块允许 EXE 模块传入数据
MEMReg_valid	IN	1	标记 EXE 模块传入 MEM 模块的数据是否有效
MEMReg_bus	IN	71	EXE 模块传入 MEM 模块的数据
WBReg_valid	OUT	1	标记 MEM 模块传入 WB 模块的数据是否有效
MEM_reg_control	OUT	38	MEM 模块前递到 ID 模块的数据
data_sram_rdata	IN	32	数据 ram 读数据
MEM_pc	OUT	32	MEM 模块所执行指令的 pc

3、功能描述

根据 EXE 模块传递来的数据执行相应的访存操作。并将访存指令结果、写回控制信号、PC 等信息在下一个时钟上升沿传递给 WB 模块。

（六）重要模块 4 设计：WB 模块

1、 工作原理

将寄存器堆写信号和数据传回 ID 阶段执行写寄存器的操作。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
reset	IN	1	复位信号
WB_allowin	OUT	1	WB 模块允许 MEM 模块传入数据
WBReg_valid	IN	1	标记 MEM 模块传入 WB 模块的数据是否有效
MEM_pc	IN	32	MEM 模块传入 WB 模块的 pc
WB_reg_control	OUT	38	WB 模块向 ID 模块传递的 regfile 写回信息和前递数据

名称	方向	位宽	功能描述
debug_wb_pc	OUT	32	写回指令 PC 值（用于 debug）
debug_wb_rf_we	OUT	4	写回指令写使能（用于 debug）
debug_wb_rf_wnum	OUT	5	写回指令写地址（用于 debug）
debug_wb_rf_wdata	OUT	32	写回指令写数据（用于 debug）

3、功能描述

根据 MEM 模块传递的与写回相关的控制信号进行写回操作，即将有关信息传递到 ID 模块中的寄存器堆中。同时，将写回操作的 PC、使能、地址、数据等信息赋值给相应 debug 信号，用于调试 CPU。

三、实验过程（50%）

（一）实验流水账

9 月 19 日发布 exp7 任务，周内阅读讲义材料，了解了本次实验的背景知识和具体内容。

9 月 24 日 10:00 - 22:00 实现 exp7 要求的任务，并通过上板测试。

9 月 28 日 15:00 - 23:00 实现 exp8-exp9 要求的任务，并通过上板测试。

10 月 4-8 日 11:00 - 10 月 8 日撰写实验报告。

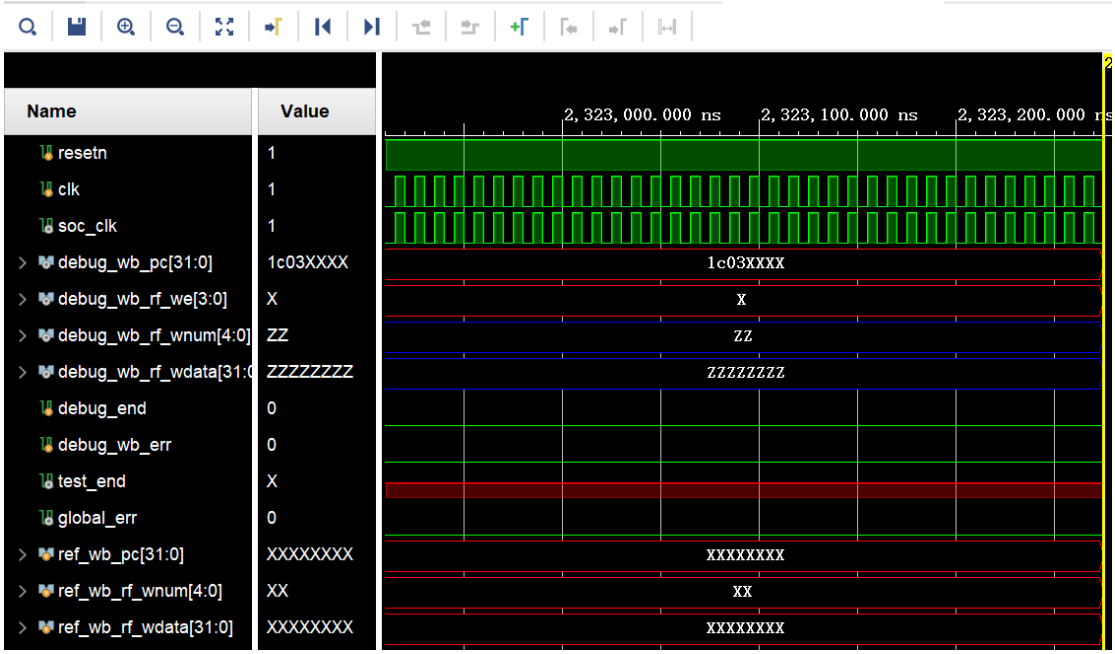
（二）错误记录

1、错误 1：exp7 代码中写使能信号例化错误

（1）错误现象

Debug 信号始终为同一个值，global_err 不会拉高，

```
[1942000 ns] Test is running, debug_wb_pc = 0x1c03XXXX
[1952000 ns] Test is running, debug_wb_pc = 0x1c03XXXX
[1962000 ns] Test is running, debug_wb_pc = 0x1c03XXXX
[1972000 ns] Test is running, debug_wb_pc = 0x1c03XXXX
[1982000 ns] Test is running, debug_wb_pc = 0x1c03XXXX
[1992000 ns] Test is running, debug_wb_pc = 0x1c03XXXX
```



（2）分析定位过程

验证环境不可能会出现问题，而 debug 信号始终未更新，且为高阻态，这表明在 WB 阶段给调试信号赋值时出现了问题。根据激励文件的设计，仅当写使能信号为高且写地址不为 0 时才开始采样，由此推测是某一阶段的写使能等信号的赋值出现了问题。

（3）错误原因

错误原因在于 MEM 传给 WB 阶段的寄存器相关的 bus 赋值时少了一个字母。即给不存在的信号赋值，没有正确赋值。control 误写为了 contrl.

```
assign MEM_reg_contrl  = {MEM_rf_we&MEM_valid, MEM_rf_waddr, MEM_rf_wdata};
```

（4）修正效果

将其改为正确命名即可。

（5）归纳总结（可选）

这个错位类型显然为信号为“X”的情况，需要检查是否正确赋值，还是很容易找到出错原因，但是不好定位，因为是在 WB 阶段写入的调试信号，可能在前面任一阶段出现了赋值错误，这个时候就要看对应阶段是否有正确相关信号。

2、错误 2：alu 操作出错

（1）错误现象

在 36797ns 时，写数据与理论值不符

```
[  36797 ns] Error!!!
    reference: PC = 0x1c03adf4, wb_rf_wnum = 0x19, wb_rf_wdata = 0x00000000
    mycpu    : PC = 0x1c03adf4, wb_rf_wnum = 0x19, wb_rf_wdata = 0x00000003
```

（2）分析定位过程

这个原因很容易找到，因为 pc 和写地址均为出错，但是写数据出现错误，那只可能是在执行阶段发生了错误。找到该条指令 ID 执行阶段对应的类型，发现为逻辑或操作，操作数据和操作码都正确，但是结果错误。

（3）错误原因

说明在 alu 中，逻辑或的实现出现问题，找到 alu 中 or 实现部分，发现是 exp6 修改时

遗漏的错误。

```
assign or_result  = alu_src1 | alu_src2 | alu_result;
```

（4）修正效果

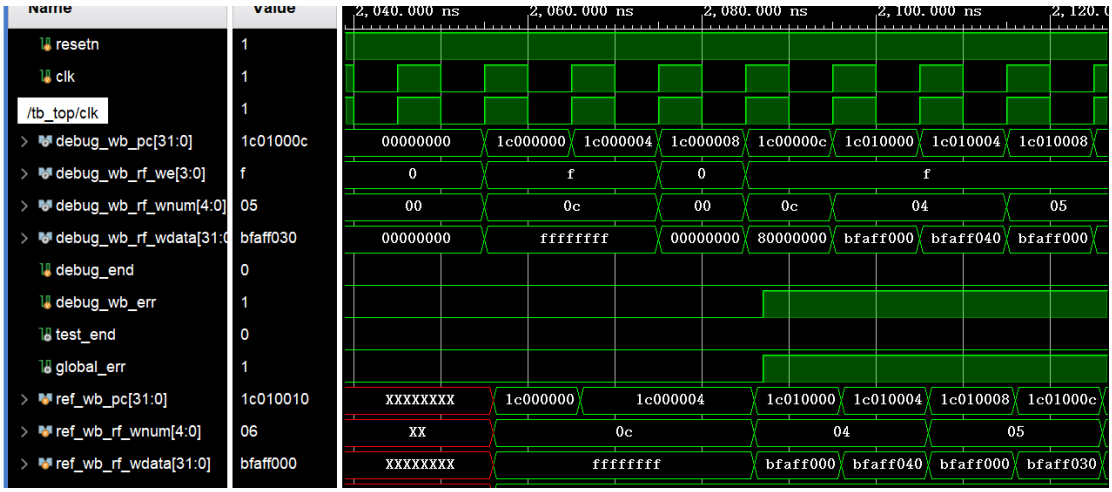
将其后多余的或 alu_result 去掉即可。

错误 3：未处理控制相关（exp7）

（1） 错误现象

如图所示，最开始时的 pc 就与金标准不一致，总是落后一拍

```
[   2087 ns] Error!!!
    reference: PC = 0x1c010000, wb_rf_wnum = 0x04, wb_rf_wdata = 0xbfaff000
    mycpu    : PC = 0x1c00000c, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x80000000
```



（2）分析定位过程

造成 pc 总落后于金标准一拍的原因很可能是跳转指令的处理不当。研究正确的流水线情况后我发现，当 pc 为 1c000008 时，下一拍的 pc 需被转移地址更新掉，在更新后再流入流水线。因此，应该是 ID 阶段的握手信号出现了问题，没有考虑到跳转指令造成的控制相关情况。

（3）错误原因

如图所示，在设计握手信号 ID_valid 时，未考虑控制相关问题，导致 valid 信号在取到错误的 pc 时仍是拉高。

```
always @(posedge clk) begin
    if(~resetn)
        ID_valid <= 1'b0;
    else if(ID_allowin)
        ID_valid <= IDReg_valid;
end
```

（4）修正效果

如图所示，在握手信号 ds_valid 的赋值语句中添加 br_taken 信号拉高的情况即可。再次仿真后得到的波形图中，该时刻的 pc 值已与金标准一致。

```
always @(posedge clk) begin
    if(~resetn)
        ID_valid <= 1'b0;
    else if(br_taken)
        ID_valid <= 1'b0;
    else if(ID_allowin)
        ID_valid <= IDReg_valid;
end
```

（5）归纳总结

4、错误 4：各阶段 allowin 信号出现漏洞

（1）错误现象

严格来说，这并不算是一种错误，但是它会影响流水线的性能。具体来说就是，流水线的 allowin 应该对应两种状况，一种是当前阶段信号处理完毕且下一流水段准备接受数据，另一种是当前流水段的无效。

（2）分析定位过程

这个错误是由助教为我指出的，即使不加这一种情况，仍可以通过仿真测试。我本身并未意识到这点，提供的仿真激励环境中也并未出现性能浪费的情况，也就是下一流水段尚不可接收，但本流水段无效时可以接收到数据，在下一流水段可以接受时传出数据。

（3）错误原因

我在设计时并未考虑到当前流水段无效的情况，这样出现的问题就是当前流水段无效时，可以接收数据而不是等待下一时钟再接收数据，会造成一定程度上的性能浪费。

（4）修正效果

增加流水段 allowin 信号拉高的情况。

```
assign ID_allowin    = ~ID_valid|ID_readygo & EXE_allowin;
```

（5）归纳总结（可选）

这个错误非常细节，极少可能下会注意到这种情况，在此非常感谢助教为我指出这一问题。

四、 实验结果

Exp7 仿真结果如下：

```
          [2022000 ns] Test is running, debug_wb_pc = 0x1c09f804
----[2023145 ns] Number 8'd20 Functional Test Point PASS!!!
==============================================================
Test end!
----PASS!!!
```

Exp9 仿真结果如下：

```
----[ 604005 ns] Number 8'd20 Functional Test Point PASS!!!
==============================================================
Test end!
----PASS!!!
$finish called at time : 604455 ns : File "F:/vivado_test/exp8/mycpu_env/soc_verify/soc_bram/testbench/mycpu_tb.v" Line 270
```

五、实验总结（可选）

本次实验的第一阶段难度较大，从编写代码到让流水线流起来（尚不论对错）的debug过程就非常漫长而曲折，而且往往是一些诸如忘记声明信号、误把某数据定义成一根电线、reg类型和wire类型混用、数据长度算错的低级错误。由于我在计算机组成原理课程中并没有选做过流水线CPU的实验，因此本次实验任务对我来说上手有些困难，尤其是有关各流水级中valid信号的定义和传递，我没有完全理解其中的机理。通过仔细阅读和体会讲义、PPT中的内容，同时询问助教，我对流水线CPU的实现思路逐渐明晰。尽管编写exp7的代码和debug的过程耗费了较多时间，但在一次次的修改代码中，我加强了对流水线CPU工作原理的理解，同时也为我后面能顺利完成exp8和exp9的实验打下了基础。

在完成exp7的基础上，exp8和exp9的修改就相对轻松很多，理解阻塞技术和前递技术的原理并明确结构图后即可顺利完成，在exp8的实验中很自然地想到利用旁路计数增加控制，因此歪打正着地将exp9也完成了，基本没怎么调试就通过了。主要需要注意到的地方就是要判断是否真的需要阻塞或前递，避免处理器性能的无谓牺牲。

吐槽部分：ddl太紧凑了。尤其是exp6到exp7的过渡，我感觉非常不合理，理应首先有一个单周期到多周期的过渡，而后再是多周期到流水线的实现。划分各个模块就需要耗费非常大的时间精力，这一点非常痛苦，尤其是在计组研讨课的核心是单周期的情况下。因此我觉得完全可以前两周为exp7-8，最后一周实现流水前递技术，exp7的任务量远超exp8和exp9。