实验五报告

学号 2017K8009929044
姓名 李昊宸
箱子号 33

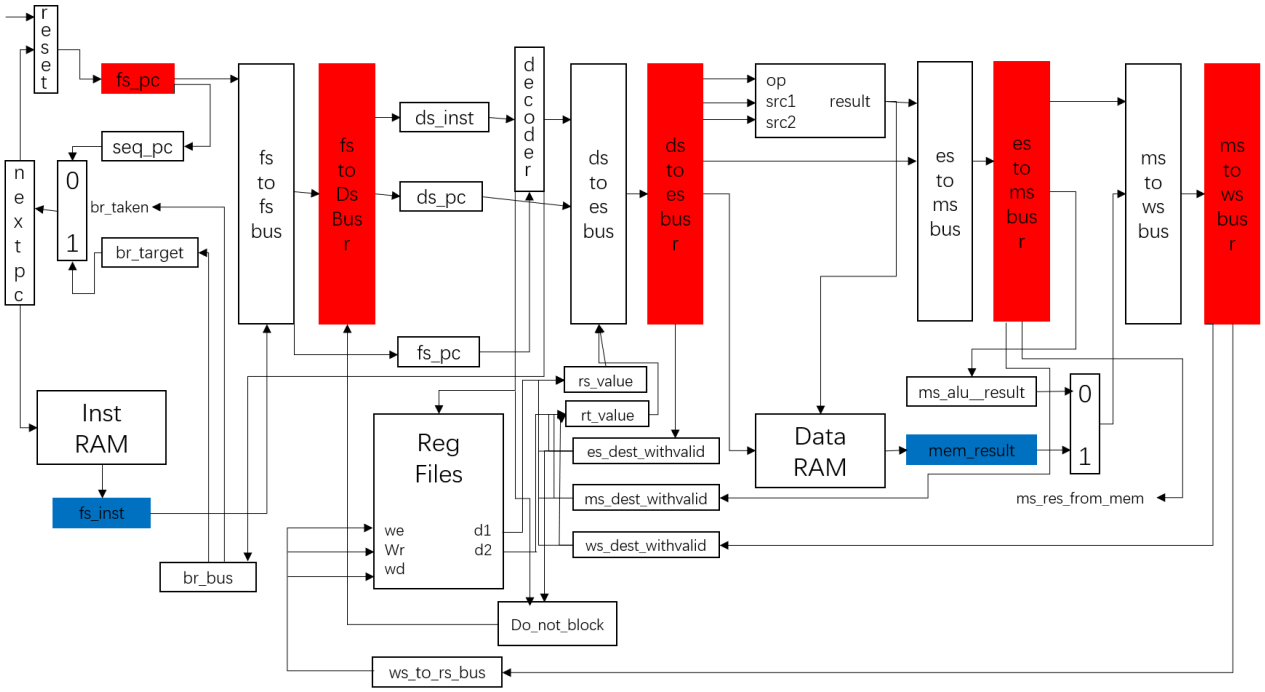
一、实验任务（10%）

1.. 加深对流水线结构的理解。

2. 学会使用阻塞的方式解决相关引发的流水线冲突的方法。

二、实验设计（40%）

（一）总体设计思路



图一 mycpu 模块结构图

上周实验所实现的部分为使用阻塞的办法解决写后读相关的问题。当 EXE,MEM,WB 阶段出现向寄存器堆写入数据的目的寄存器号与 ID 阶段从寄存器堆中读出数据的源寄存器号相同时，部分情况下我们需要考虑将 ID 取值模块阻塞在这里，直到后续模块将数据写入之后再进行读取操作，否则会出现数据还未写入就已经被读取的错误。在这里，我们采取的方式是分别从 EXE,MEM,WB 模块中向 ID 模块引入三条组合逻辑总线，将当前的目标操作寄存器号送回 ID 模块，与 ID 模块源操作数进行比较，若符合阻塞条件就将 ID 模块的 ds_ready_go 修改为 0。

这一周的实验是在阻塞的基础上进行补充，采用前递的方法减少不必要的阻塞。思路就是将执行级、访存级、写回级出口端的将写回寄存器堆的数据传回译码级的寄存器堆读出口，在此处使用四选一数据选择器选出运行指

令所需要的正确的数据，从而减少阻塞的时长。

（二）重要模块 1 设计：ID 译码模块

1、工作原理

从 IF 传入的信号，经过译码逻辑后，被分解成每个部分的信号后，和 PC 有关的信号被返回 ID，其余和执行有关的信号被传递到 EX 模块。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟
reset	IN	1	复位信号，拉高为复位
es_allowin	IN	1	允许数据流水进入 EX
ds_allowin	OUT	1	允许数据流水进入 DS
fs_to_ds_valid	IN	1	从 IF 向 ID 的数据传输是否可以进行
fs_to_ds_bus	IN	64	IF 向 ID 的总线
ds_to_es_valid	OUT	1	从 ID 向 EX 的数据传输是否可以进行
ds_to_es_bus	OUT	136	ID 向 ES 的总线
br_bus	OUT	33	从 ID 模块返回的下一步 PC 跳转信号
ws_to_rf_bus	IN	38	从 WB 模块返回的存入 regfile 的数据
es_dest_withvalid	IN	39	从 ES 模块返回的目的寄存器号和阻塞信号和数据和选择信号
ms_dest_withvalid	IN	39	从 MS 模块返回的目的寄存器号和阻塞信号和数据和选择信号
ws_dest_withvalid	IN	38	从 WS 模块返回的目的寄存器号和阻塞信号和数据和选择信号

表一 ID 模块接口定义

3、功能描述

因为存在写后读相关的取值问题，在 lab4 中使用阻塞方法解决相关问题。引入三个目的寄存器号传递信号：es_dest_withvalid、ms_dest_withvalid 和 ws_dest_withvalid，作用为传递 EXE,MEM 和 WB 阶段正在运行中的寄存器号，随后与 ID 阶段的源寄存器号作比较，决定是否将其阻塞。具体实现方式见实验实现部分。

在 lab5 中，为了减少不必要的阻塞消耗，我们可以考虑把发生写后读冲突的寄存器的数据直接在流水中前递到译码级模块，也就是说，采用旁路（bypass）的方式，以取代堵，不必傻傻的等到数据存到寄存器内之后再取出来，而是将已经计算好的数据在还没存入之前就允许其他指令使用。具体的实现见实现部分。

（三）重要模块 2 设计：EX 执行模块

1、工作原理

从 ID 传入的信号，被一一取出，一部分进如 alu 中进行计算，另一部分作为数据 RAM 的使能和控制信号被输出到数据 RAM 处，等待下一拍上升沿被更新到 MEM 阶段。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟
reset	IN	1	复位信号，拉高为复位
ms_allowin	IN	1	允许数据流水进入 MEM
es_allowin	OUT	1	允许数据流水进入 EX

名称	方向	位宽	功能描述
ds_to_es_valid	IN	1	从 ID 向 EX 的数据传输是否可以进行
ds_to_es_bus	IN	136	ID 向 ES 的总线
es_to_ms_valid	OUT	1	从 EX 向 MEM 的数据传输是否可以进行
es_to_ms_bus	OUT	71	EX 向 MEM 的总线
data_sram_en	OUT	1	ram 使能信号，高电平有效
data_sram_wen	OUT	4	ram 字节写使能信号，高电平有效
data_sram_addr	OUT	32	ram 读写地址，字节寻址
data_sram_wdata	OUT	32	ram 写数据
es_dest_withvalid	OUT	39	从 ES 模块返回的目的寄存器号和阻塞信号和数据和选择信号

表二 EX 模块接口定义

3、功能描述

因为存在写后读相关的取值问题，在 lab4 中使用阻塞方法解决相关问题。引入 EXE 模块目的寄存器号传递信号：es_dest_withvalid，作用为传递 EXE 阶段正在运行中的寄存器号，随后与 ID 阶段的源寄存器号作比较，决定是否将其阻塞。具体实现方式见实验实现部分。

在 lab5 中，为了减少不必要的阻塞消耗，我们选择直接把已经计算出的数值直接通过旁路（bypass）传递回译码级模块。于是我们选择将 es_dest_withvalid 的功能扩大化，把待写入数据的值也一并传递，这样可以避免大部分的阻塞。另外，如果处于执行级的 PC 为 lw 或者其他的访存指令的话，需要等到执行到访存模块时才能拿到正确的值，所以如果此时目的寄存器号与译码模块的源寄存器号相同的话，需要将 ID 阻塞一拍。

（四）重要模块 3 设计：MEM 访存模块

1、工作原理

从 EX 传入的信号主要被保存之后，等待下一拍将其传递到下一个 WB 写回阶段。另外，最关键的部分是上个阶段从 EX 提出的访存申请被这一拍上升沿应答之后，返回到该模块中。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟
reset	IN	1	复位信号，拉高为复位
ws_allowin	IN	1	允许数据流水进入 WB
ms_allowin	OUT	1	允许数据流水进入 MEM
es_to_ms_valid	IN	1	从 EX 向 MEM 的数据传输是否可以进行
es_to_ms_bus	IN	64	EX 向 MEM 的总线
ms_to_ws_valid	OUT	1	从 MEM 向 WB 的数据传输是否可以进行
ms_to_ws_bus	OUT	70	MEM 向 WB 的总线
data_sram_rdata	IN	32	ram 读数据
ms_dest_withvalid	OUT	39	从 MS 模块返回的目的寄存器号和阻塞信号和数据和选择信号

表三 MEM 模块接口定义

3、功能描述

因为存在写后读相关的取值问题，在 lab4 中使用阻塞方法解决相关问题。引入 MEM 模块目的寄存器号传递

信号：ms_dest_withvalid，作用为传递 MEM 阶段正在运行中的寄存器号，随后与 ID 阶段的源寄存器号作比较，决定是否将其阻塞。具体实现方式见实验实现部分。

在 lab5 中，为了减少不必要的阻塞消耗，我们选择直接把已经计算出的数值直接通过旁路（bypass）传递回译码级模块。于是我们选择将 ms_dest_withvalid 的功能扩大化，把待写入数据的值也一并传递，这样可以避免目前为止所有在访存级引起的写后读数据相关的阻塞。

（五）重要模块 4 设计：WB 写回模块

1、工作原理

从 MEM 传来的写回寄存器堆的数据在从 ID 一路传递过来的指令控制下，被写入到 regfile 中。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟
reset	IN	1	复位信号，拉高为复位
ws_allowin	OUT	1	允许数据流水进入 WB
ms_to_ws_valid	IN	1	从 MEM 向 WB 的数据传输是否可以进行
ms_to_ws_bus	IN	70	MEM 向 WB 的总线
ws_to_rf_bus	OUT	38	WB 向 ID 中 regfile 的总线
debug_wb_pc	OUT	32	写回级（多周期最后一级）的 PC，需要 mycpu 里将 PC 一路带到 写回级
debug_wb_rf_wen	OUT	4	写回级写寄存器堆(regfiles)的写使能，为字节写使能，如果 mycpu 写 regfiles 为单字节写使能，则将写使能扩展成 4 位即可
debug_wb_rf_wnum	OUT	5	写回级写 regfiles 的目的寄存器号
debug_wb_rf_wdata	OUT	32	写回级写 regfiles 的写数据
ws_dest_withvalid	OUT	38	从 WS 模块返回的目的寄存器号和阻塞信号和数据

表四 WB 模块接口定义

3、功能描述

因为存在写后读相关的取值问题，在 lab4 中使用阻塞方法解决相关问题。引入 WB 模块目的寄存器号传递信号：ws_dest_withvalid，作用为传递 WB 阶段正在运行中的寄存器号，随后与 ID 阶段的源寄存器号作比较，决定是否将其阻塞。具体实现方式见实验实现部分。

在 lab5 中，为了减少不必要的阻塞消耗，我们选择直接把已经计算出的数值直接通过旁路（bypass）传递回译码级模块。于是我们选择将 ws_dest_withvalid 的功能扩大化，把待写入数据的值也一并传递，这样可以避免目前为止所有在写回级引起的写后读数据相关的阻塞。

三、实验过程（50%）

（一）实验流水账

2019/9/25　10:00 – 10:30 对照代码研究多周期阻塞加前递流水 cpu 的结构，并画出框图

2019/9/25　18:00 –20:00　书写实验报告

（二）错误记录

1、错误 1：多余阻塞时长

（1）错误现象

在实现前递后，非LW指令应该不引起阻塞，但是波形中依然存在该问题，总的用时并没有比仅采用阻塞解决冲突的方式更少。

（2）分析定位过程

决定是否进行阻塞的信号是 ds_ready_go，问题应该出现在 ds_ready_go 信号。

（3）错误原因

在 lab4 的书写中，我引入了 wire 型变量 do_not_block，其作用为当不存在写后读数据相关时为 1，出现写后读数据相关时为 0，并且直接把 do_not_block 的值赋用给 ds_ready_go，这样导致无论数据是否前递成功，译码级模块都被阻塞，于是没有起到优化的效果。

（4）修正效果

对 ID、EXE、MEM 和 WB 全体进行修正：

对 WB：

更改输出信号　*output [37 :0]　ws_dest_withvalid*　，并给他赋值：

```
wire    block_valid,block;
assign block = (ws_dest == 5'd0)?0:ws_gr_we;
assign block_valid = block&ws_valid;
assign ws_dest_withvalid = {ws_final_result,block_valid,ws_dest};
```

block 信号表示处于 WB 模块的指令是否是需要写回非 0 号寄存器的指令。blokc_valid 信号则为将 block 信号与 ws_valid 信号做与操作，从而消除了在指令还未到达 WB 模块时，block 信号为 X，无法给 ID 模块做判断的情况。ws_dest_withvalid 信号则为 38 位宽信号，第 0 至 4 位放置目的寄存器号，第 5 位 ws_dest_withvalid 信号，第 6 至 37 位放置将 alu 的计算结果。

对 MEM：

更改输出信号　*output [38 :0]　ms_dest_withvalid*　，并给他赋值：

```
wire    block_valid,block;
assign block = (ms_dest == 5'd0)?0:ms_gr_we;
assign block_valid = block&ms_valid;
assign ms_dest_withvalid = {ms_res_from_mem,ms_final_result,block_valid,ms_dest};
```

block 信号表示处于 MEM 模块的指令是否是需要写回非 0 号寄存器的指令。blokc_valid 信号则为将 block 信号与 ms_valid 信号做与操作，从而消除了在指令还未到达 MEM 模块时，block 信号为 X，无法给 ID 模块做判断的情况。ms_dest_withvalid 信号则为 39 位宽信号，第 0 至 4 位放置目的寄存器号，第 5 位 ws_dest_withvalid 信号，第 6 至 37 位放置将 alu 的计算结果，最高位放置最终数据来源是 alu 还是内存的信号。（其实最高位信号并没有用上，可以删去）

对 EXE：

更改输出信号　*output [38 :0]　es_dest_withvalid*　，并给他赋值：

```
wire    block_valid,block;
assign block = (es_dest == 5'd0)?0:es_gr_we;
assign block_valid = block&es_valid;
assign es_dest_withvalid = {es_load_op&es_valid,es_alu_result ,block_valid,es_dest};
```

block 信号表示处于 EXE 模块的指令是否是需要写回非 0 号寄存器的指令。blokc_valid 信号则为将 block 信号与 es_valid 信号做与操作，从而消除了在指令还未到达 EXE 模块时，block 信号为 X，无法给 ID 模块做判断的情况。es_dest_withvalid 信号则为 39 位宽信号，第 0 至 4 位放置目的寄存器号，第 5 位 es_dest_withvalid 信号，第 6 至 37 位放置将 alu 的计算结果，最高位放置最终数据来源是 alu 还是内存的信号。

对 ID:

有很多信号可以删除，此处列出的仅为不用删除的部分和修改的部分：

```
wire        rs_es, rs_ms, rs_ws, rt_es, rt_ms, rt_ws;
assign      rs_es = (es_dest_withvalid[5] == 0)?1                    //没有写回操作或者目标寄存器为 0，或者有写回操作但是目标寄存器不是 ID 源寄存器时为 1
              :(rs == es_dest_withvalid[4:0])?0
              :1;
assign      rs_ms = (ms_dest_withvalid[5] == 0)?1
              :(rs == ms_dest_withvalid[4:0])?0
              :1;
assign      rs_ws = (ws_dest_withvalid[5] == 0)?1
              :(rs == ws_dest_withvalid[4:0])?0
              :1;
assign      rt_es = (es_dest_withvalid[5] == 0)?1                    //没有写回操作或者目标寄存器为 0，或者有写回操作但是目标寄存器不是 ID 源寄存器时为 1
              :(rt == es_dest_withvalid[4:0])?0
              :1;
assign      rt_ms = (ms_dest_withvalid[5] == 0)?1
              :(rt == ms_dest_withvalid[4:0])?0
              :1;
assign      rt_ws = (ws_dest_withvalid[5] == 0)?1
              :(rt == ws_dest_withvalid[4:0])?0
              :1;

assign      do_not_block = es_dest_withvalid[38]?0:1;                //如果 exe 阶段的是 lw 指令，那么就阻塞一拍，否则就不阻塞
assign  rs_value = (~rs_es)?es_dest_withvalid[37:6]
              :(~rs_ms)?ms_dest_withvalid[37:6]
              :(~rs_ws)?ws_dest_withvalid[37:6]
              :rf_rdata1;
assign  rt_value = (~rt_es)?es_dest_withvalid[37:6]
              :(~rt_ms)?ms_dest_withvalid[37:6]
              :(~rt_ws)?ws_dest_withvalid[37:6]
              :rf_rdata2;
```

最后，还需要对 ds_ready_go 信号做一点修改：

```
assign ds_ready_go = do_not_block;
```

（5）归纳总结（可选）

这次实验也很顺利，一次性通过。下附使用前递和不使用前递时间长短的对比图：

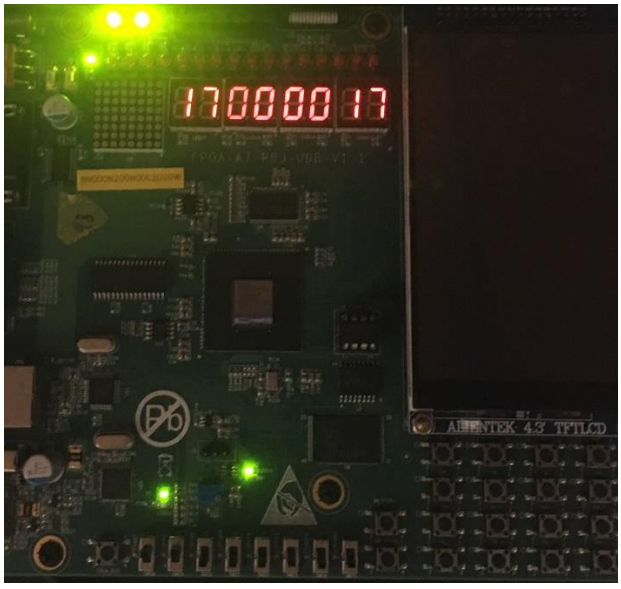
```
----[1148515 ns] Number 8'd21 Functional Test Point PASS!!!
----[1149855 ns] Number 8'd22 Functional Test Point PASS!!!
      [1152000 ns] Test is running, debug_wb_pc = 0xbfc510cc
----[1152545 ns] Number 8'd23 Functional Test Point PASS!!!
==========
Test end!
----PASS!!!
$finish called at time : 1153185 ns : File "E:/Xilinx/Vivado/home
```

图一 使用前递方式的仿真运行时间

```
----[1310005 ns] Number 8'd23 Functional Test Point PASS!!
==========
Test end!
----PASS!!!
$finish called at time : 1310945 ns : File "E:/Xilinx/Vivs
run: Time (s): cpu = 00:01:22 ; elapsed = 00:01:23 . Memor
```

图二 不使用前递方式的仿真运行时间

四、实验总结（可选）



图三 上板成功拍照留念

终于完成了前递方式多周期五级流水 cpu 的设计！对上学期多周期 cpu 的设计进行了更系统的完善。