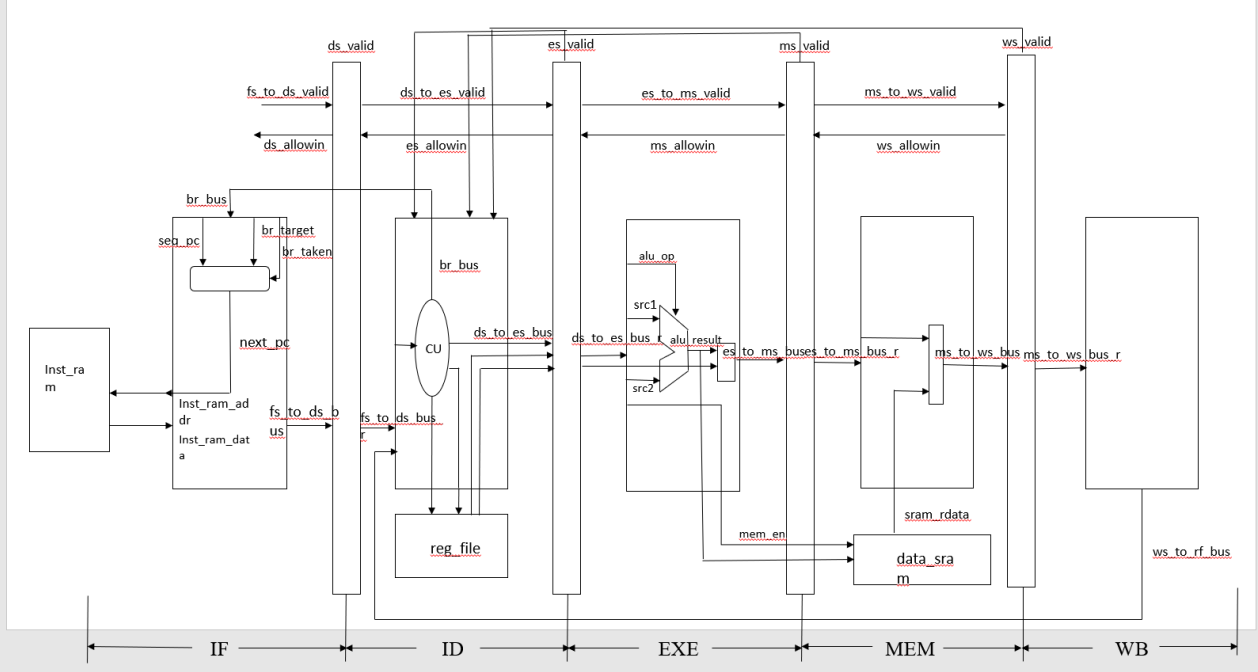
实验 4 报告

学号：2017K8009922026
姓名：康齐瀚
箱子号：63

一、实验任务（10%）

通过在流水线中增加阻塞的方法消除数据相关，主要是读后写相关。最终能成功通过仿真并完成上板

二、实验设计（40%）



（一）总体设计思路

在本次实验中，需要通过增加阻塞的方式来消除数据相关，通过分析可以知道，只需要在译码阶段将译码出的读寄存器号与后面几个阶段的写寄存器号相比较就能判断出是否需要阻塞。如果此时读寄存器号与后面任何一个阶段的写回寄存器号相同，就需要把该指令阻塞在译码阶段。

在判断是否需要阻塞时不仅需要看寄存器号是否相同，还需要看当前正在译码的指令是否需要读取寄存器。对于有立即数的指令，以及无条件跳转指令等，并不需要等待寄存器结果写回就能继续运行，因此不需要阻塞。（尽管阻塞不会产生错误，但强行阻塞会影响性能）；此外，需要考虑后面几级流水级中的指令是否有效，这主要由 EXE，MEM，WB 级的 valid 信号给出，valid 有效时进行寄存器比较才有意义，否则会出现阻塞无法恢复的问题。

在本次实验中采用了将两个操作数 src1 和 src2 分开考虑的策略。在 EXE，MEM，WB 模块中增加新的输出端口 dest, wen, valid 分别表示要写的寄存器号，写使能信号以及流水线有效信号。

```
output [4:0]  es_waddr   ,
output        es_wen,
output        es_is_valid

output [4:0]                    ms_waddr       ,
output                          ms_wen         ,
output                          ms_is_valid

output        ws_is_valid
```

当然 WB 模块仅需要 valid 信号，因为 rf_to_ws_bus 中包含了另外两个信号

上述信号的赋值如下：

```
assign es_waddr = es_dest;
assign es_wen   = es_gr_we;
assign es_is_valid = es_valid;

assign ms_waddr = ms_dest;
assign ms_wen   = ms_gr_we;
assign ms_is_valid = ms_valid;

assign ws_is_valid = ws_valid;
```

在 ID 模块中接受上述信号，并根据相关情况对表示 src1 和 src2 是否存在相关的信号 sign1,sign2 进行赋值：

```
assign sign1 = //(rf_raddr1 == 5'b0 || src1_is_pc) ? 1'b0 :
       assign ds_ready_go  =  (sign1 == 1'b1 || sign2 == 1'b1) ? 1'b0 : 1'b1;
               (rf_raddr1 == ms_waddr && ms_wen && ms_is_valid) ? 1'b1 :
               (rf_raddr1 == rf_waddr && rf_we && ws_is_valid) ? 1'b1 : 1'b0;

assign sign2 = //(rf_raddr2 == 5'b0 || src2_is_8 || src2_is_imm) ? 1'b0:
               (rf_raddr2 == 5'b0 || src2_is_8 == 1'b1) ? 1'b0 :
               (rf_raddr2 == es_waddr && es_wen && es_is_valid) ? 1'b1:
               (rf_raddr2 == ms_waddr && ms_wen && ms_is_valid) ? 1'b1:
               (rf_raddr2 == rf_waddr && rf_we && ws_is_valid) ? 1'b1 : 1'b0;
```

sign1 为 1 表示有相关，为 0 表示无相关，最后对 ds_ready_go 进行赋值即可：

本次实验没有任何新增的重要模块。

三、实验过程（50%）

（一）实验流水账

2019/9/19：18：00 ~ 20：00 初次增加阻塞，但未通过仿真

2019/9/20：15：30 ~ 17：00 修改阻塞代码，顺利通过仿真并成功上板

（二）错误记录

初次增加阻塞设计：

1、错误 1：PC 一直不变

（1）错误现象

当仿真到 22000ns 之后，程序计数器 PC 的值一直不变

（2）分析定位过程

观察 PC 的值，发现循环的值一直是 0xbfc41afc 处循环，查看反汇编代码可以发现此处的指令为 or t0,t1,s3, 查看上一条指令发现为 sll t1,s0,0x18，可以看出两条指令存在着数据相关，因此考虑是因为阻塞导致的 bug

进一步观察波形可以发现，由于 sll 和 or 指令之间存在读后写相关，在设计中（如下图所示），判定 ds_ready_go 一直为 0，因此该条指令会一直被阻塞在译码阶段。同时，sll 之前的指令（包括 sll）本身仍然会一直向前顺利执行，在两个周期之后，EXE，MEM，WB 级的流水线寄存器中的写回寄存器号都变为$t1, 而译码级的指令 or 由于一直堵塞在译码级，每当与后面的 EXE，MEM，WB 的流水线寄存器相比较时都会发现有数据相关，从而继续阻塞，这就导致了一个恶性循环，sll 一直阻塞在译码级。

（3）错误原因

代码编写不当，导致该条指令一直阻塞在译码级。在这种代码编写的情况下，只要是一条指令与其前面紧邻的一条指令发生了读后写相关，那么该指令就会一直被阻塞译码级。根本原因是没有考虑到各级的 valid 信号，es_valid 信号表示了该流水级中信号的有效性。在阻塞的情况下，valid 信号会变为 0，表示该级流水级寄存器中的数据无效，此时再判断写回寄存器号和译码时的读取寄存器号已经没有意义了

（4）修正效果

将 EXE，MEM，WB 的 valid 信号加入判断，如果某一级的 valid 有效，而且写回寄存器号与译码的读寄存器号相同，而且该级的写使能有效，那么就阻塞流水线，否则 ds_ready_go 置为 1。

将 valid 纳入判断之后，流水线的阻塞逻辑才算完整，仿真顺利通过

（5）归纳总结（可选）

逻辑考虑不完善，针对这种错误，要么在写代码之前就考虑甚至模拟好运行的实际情况，要么就写出一个粗糙的样品然后等着慢慢调 bug

四、实验总结（可选）

本次实验比较简单，不过第一次由于没有考虑到 valid 信号所以出现了 bug，再认真阅读了任务书并考虑了之后就顺利完成了实验