

实验 2 报告

学号：2019K8009929039

姓名：周鹏宇

箱子号：50

一、实验任务（10%）

本次报告所涵盖的三个实验：lab3（为已经给出的流水线 CPUdebug），lab4（处理阻塞）和 lab5（处理前递），大致目标就是帮助我们了解编写 rtl 代码时 bug 的常见种类和 debug 的常用手段，同时帮助我们对流水线 CPU 有一个大致的了解，清楚其数据流向和控制信号。此外在原有 CPU 的基础上考虑到写后读产生的流水级冲突，进行阻塞和前递，进行简单的 rtl 代码编写实战，为之后的实验做基础。同时，后两项任务其实也是有一定区分度的，如果针对每个指令的具体情况对阻塞或者前递进行调控，会带来流水级效率的提升，而在前递的最后一个章节，提到了对主频优化的设计，也是具有挑战性的。

二、实验设计（40%）

（一）总体设计思路

本人书写的结构图如下（由于是在 notability 软件上手画的，故仅能在此展示截屏，源文件我已提交至 github，链接：https://github.com/zpy2001/2021CA_Assignment）：

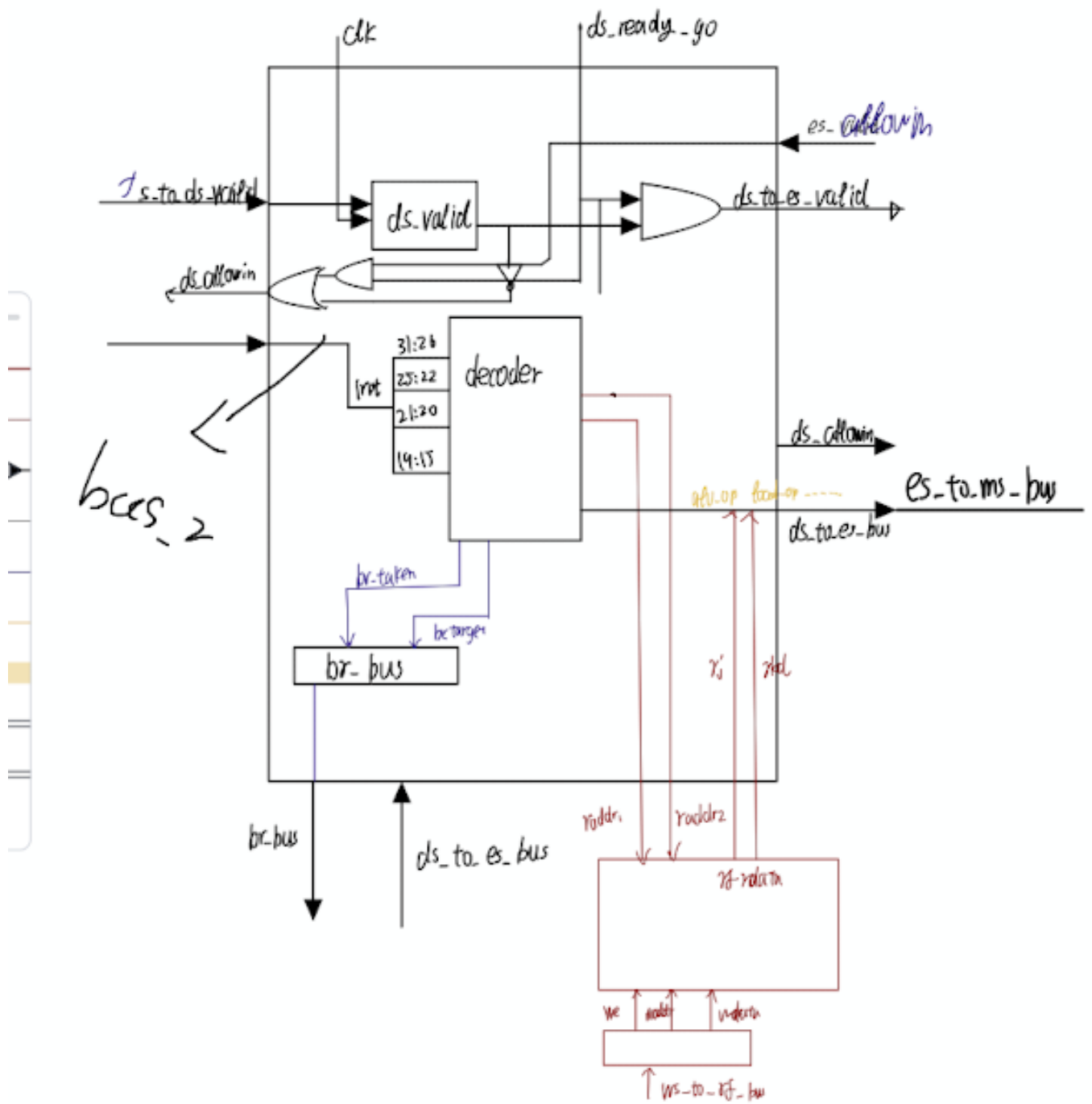


Figure 1 用于说明控制信号的 ds 结构大图

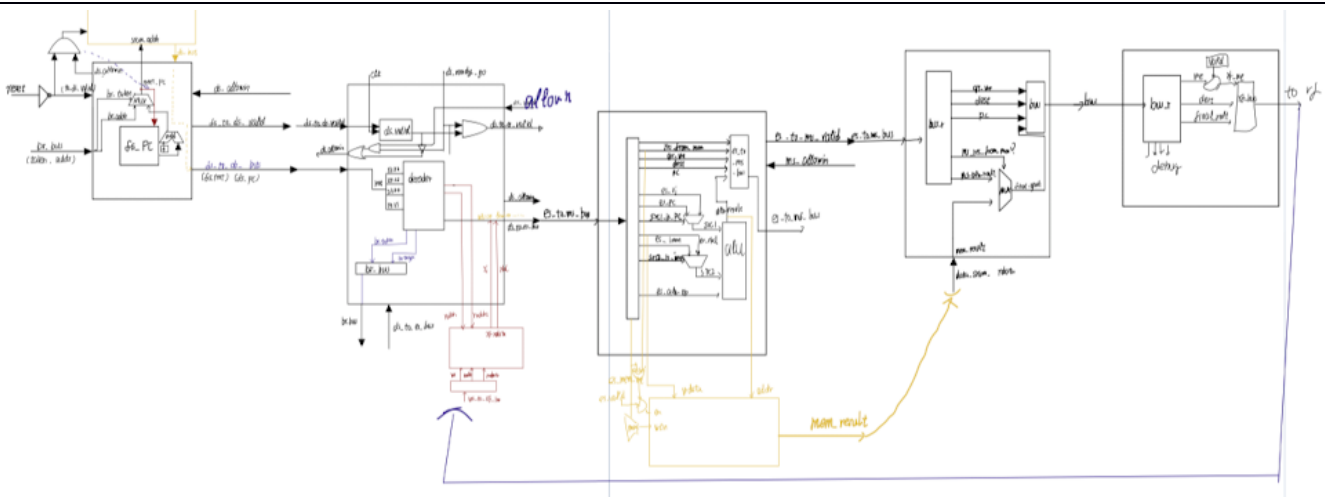


Figure 2 结构全图

本结构图由于制作匆忙，存在一些简略和错误，其中需要指出的错误是部分流水级之间的数据传递并非仅仅一个组合逻辑就可以完成的，而是需要有一个暂存器进行延迟和刷新的，这个错误在 ds 流水级大图中就可以体现出来。此外，由于各个流水级的控制信号是大同小异的，即：

ds_valid 信号拉高表示 ds 流水级中存在有效数据，此时若下一级流水级数据准入（es_allowin 拉高）且本级流水级的数据允许流出（ds_ready_go 信号拉高），则允许上一级流水级（FS）的数据进入，否则不可，若 ds_valid 低位，则也允许上一级流水级数据进入，即 ds_allowin 信号拉高。此外，ds_allowin 的信号拉高时意味着 ds 流水级工作已经完成，允许上一级流水级数据进入（已经写了一遍了），而 ds_to_es_valid 信号拉高则是通知下一流水级，数据已经处理好，可以向该级传递；这一信号和 es_allowin 信号构成了一对握手信号，当且仅当二者同时拉高时，装载在 ds_to_es_bus 中的数据方可传递到 es 流水级中。

各个流水级之间的控制信号交互几乎都与此完全一致，故仅以 ds 流水级为例，详细说明，而结构图中也只详细画出了 ds 流水级中的控制信号线路图。

各个流水级是已经被给出的，并非自主设计，而控制相关的处理也已在讲义中给出，故仅在重要模块设计部分介绍各个流水级，并在错误处理阶段简单介绍 br_taken_cancel 信号的设计。

Lab4 和 lab5 中的处理写后读的方案则是自主设计的，将予以简述

用阻塞的手段处理指令相关：在 es, ms 和 ws 三个流水级各自设计一个 bus，装载该流水级的 rf_wen 信号和 rf_dest（即需要写入的寄存器号）送至 ds 流水级（全为组合逻辑），而在 ds 流水级中，对 ds_ready_go 信号的拉高条件做一些处理：当前三个流水级中任何一个和 ds 流水级同时将 rf_wen 信号拉高，且 ds 级读的寄存器号和前者写的寄存器号相同时，阻塞住 ds，ds_ready_go 信号拉低。其中各个流水级的 rf_we 信号都是 valid 信号和 gr_we 信号相与得到。

用前递的手段处理指令相关：只要完成了 lab4，那么 lab5 的修改过程其实相当简单的，只不过需要 es 和 ms 流水级向 ds 流水级前递的数据装载入 bus 中，同时，ready_go 的信号也需要进行适当的修改（主要取决于 es 流水级）。之后根据传递进来的控制信号（或者说判断是否发生冲突的信号）在 es 流水级返回的数据，ms 流水级反回

的数据和寄存器堆的数据中选择源操作数，并将之传递给下一个流水级。同时，由于本次试验的所使用的 data RAM 是同步 RAM，因此如果 ld.w 指令处在 es 流水级，并与在 ds 级的指令产生了写后读的冲突，那么必须阻塞一拍等待 ld 指令取完以后再数据前递，这也是 lab5 中唯一需要阻塞的地方。

在此简单画一下 lab4 的结构图

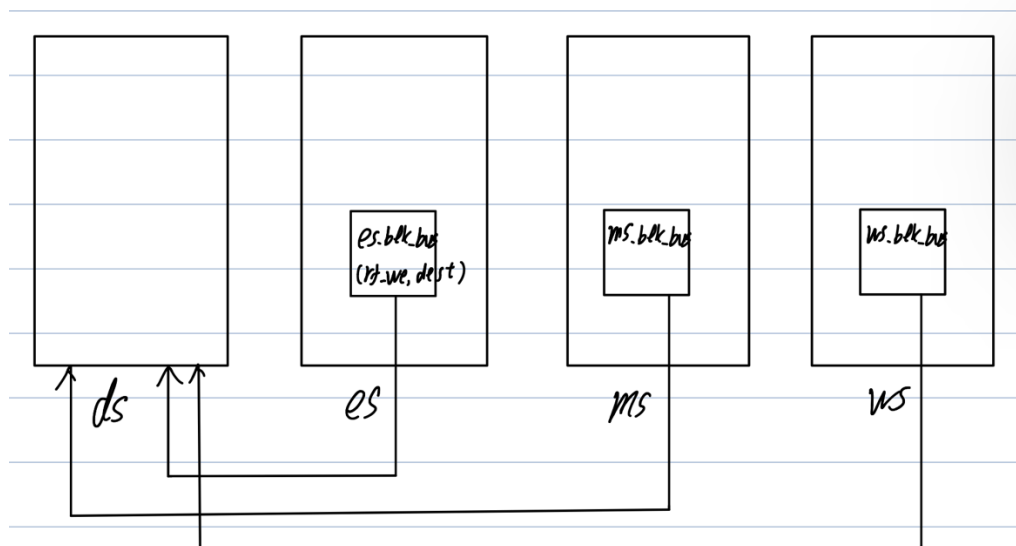


Figure 3lab4 的结构图

（二）重要模块 1 设计：DS 流水级

该流水级负责分析 FS 阶段取进来的指令的功能，同时将译码出的信息传递给下一个流水级或者 fs 流水级。其中，部分信息需要从寄存器堆中取出。

1、工作原理

通过译码器对读进来的指令进行解析，并将解析出的信息装载入 ds_to_es_bus 或者直接从寄存器中读出信息后装入 ds_to_es_bus，此外，若译码信息是分支跳转指令，则将此信息传递回 fs 流水级。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟周期信号
reset	IN	1	复位信号
es_allowin	IN	1	当此信号拉高时，表示 es 流水级允许数据传入
ds_allowin	OUT	1	当此信号拉高时，表示 ds 流水级允许数据传入
fs_to_ds_valid	IN	1	当此信号拉高时，表示 fs 流水级已经准备好发送信号
fs_to_ds_bus	IN	[FS_TO_DS_BUS_WD-1:0]	装载 fs 流水级向 ds 流水级传递的信息
ds_to_es_valid	OUT	1	当此信号拉高时，表示 ds 流水级已经准备好发送信号
ds_to_es_bus	OUT	[DS_TO_ES_BUS_WD-1:0]	装载 ds 流水级向 es 流水级传递的信息
br_bus	OUT	[BR_BUS_WD :0]	装载需要传递给 fs 流水级的译码出的分支跳转信息

名称	方向	位宽	功能描述
ws_to_rf_bus	IN	[`WS_TO_RF_BUS_WD - 1:0]	写回需要的
es_blk_bus	IN	[`ES_FWD_BUS - 1:0]	从 es 流水级传递来的前递信息
ms_blk_bus	IN	[`MS_FWD_BUS - 1:0]	从 ms 流水级传递来的前递信息

3、功能描述

其译码阶段主要是通过逐条译码和逐项译码，将多数信息直接得出，部分通过译码中的信息在寄存器中读出需要向下一级流水级传递的信息，还有部分则是通过从寄存器中读出的信息进行运算后得到。译出的信息中跳转部分则要装载入另一个 bus 传递给 fs 流水级。

（三）重要模块 2 设计：ES 流水级

1、工作原理

（1）、运算的实现：由 alu 完成绝大多数运算操作，alu 获取两个操作数和 alu_op 后，对 alu_op 进行译码，或许进行何种运算的信息后，对两个操作数进行运算，并在 alu_result 接口输出，整个过程为组合逻辑。Alu 的操作数则在 ID 译码后的结果进行进一步的选取后获得。

（2）、前递：用流水级 valid 信号屏蔽流水级无效时的前递数据，至于在 es 流水级可能出现的阻塞，已经在前文中介绍了，阻塞使能信号即 es_res_from_mem（是否涉及到访存）。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟周期信号
reset	IN	1	复位信号
ms_allowin	IN	1	当此信号拉高时，表示 ms 流水级允许数据传入
es_allowin	OUT	1	当此信号拉高时，表示 es 流水级允许数据传入
ds_to_es_valid	IN	1	当此信号拉高时，表示 ds 流水级已经准备好发送信号
ds_to_es_bus	IN	[`DS_TO_ES_BUS_WD - 1:0]	装载 ds 流水级向 es 流水级传递的信息
es_to_ms_valid	OUT	1	当此信号拉高时，表示 es 流水级已经准备好发送信号
es_to_ms_bus	OUT	[`ES_TO_MS_BUS_WD - 1:0]	装载 es 流水级向 ms 流水级传递的信息
data_sram_en	OUT	1	数据 RAM 使能信号
data_sram_wen	OUT	[3:0]	数据 RAM 写使能信号，store 指令置为全 1，其他全 0
data_sram_addr	OUT	[31:0]	数据 RAM 访存地址
data_sram_wdata	OUT	[31:0]	数据 RAM 写入数据
es_blk_bus	OUT	[`ES_FWD_BUS - 1:0]	es 流水级传递的前递信息

3、功能描述

负责对 ID 生成的各种数据进行运算，在本级流水中完成绝大多数运算指令的操作、并生成访存指令的地址，而以上任务主要通过 ALU 完成。

（四）重要模块 3 设计：MS 流水级

1、工作原理

通过由 es 传递来的信息（其实控制信号是 ID 阶段译出的）,对 es 流水级算出的结果和 ms 从 RAM 中读出的结果进行筛选。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟周期信号
reset	IN	1	复位信号
es_allowin	IN	1	当此信号拉高时，表示 ms 流水级允许数据传入
ms_allowin	OUT	1	当此信号拉高时，表示 es 流水级允许数据传入
es_to_ms_valid	IN	1	当此信号拉高时，表示 ds 流水级已经准备好发送信号
es_to_ms_bus	IN	[DS_TO_ES_BUS_WD-1:0]	装载 ds 流水级向 es 流水级传递的信息
ms_to_ws_valid	OUT	1	当此信号拉高时，表示 ms 流水级已经准备好发送信号
ms_to_ws_bus	OUT	[ES_TO_MS_BUS_WD-1:0]	装载 ms 流水级向 ws 流水级传递的信息
data_sram_rdata	OUT	[31:0]	数据 RAM 读出数据
ms_blk_bus	OUT	[MS_FWD_BUS-1:0]	ms 流水级传递的前递信息

3、功能描述

根据上游传递来的控制信号在 EXE 级计算结果与数据 RAM 读取结果间选择其一作为写回结果传递给下游写回级。

可以注意到，s 指令的使能，地址和数据的发送是在上一个流水级（ES 级）完成的，这同样是因为同步 RAM 的特性，在第一拍发送使能信号后，第二拍才能返回数据，因此当使能信号在 ES 级发送后，在下一拍进入本级流水时，本级流水恰好能够从数据 RAM 中获取结果，可以提高流水线效率。

（五）重要模块 4 设计：WS 流水级

1、接口定义

名称	方向	位宽	功能描述
debug_wb_pc	OUT	[31:0]	输出 pc debug 信息
debug_wb_rf_wen	OUT	[3:0]	输出 rf_wen debug 信息
debug_wb_rf_wnum	OUT	[4:0]	输出 rf_wnum debug 信息（写地址，即寄存器号）
debug_wb_rf_wdata	OUT	[31:0]	输出 rf_wdata 信息

2、功能描述

输出 debug 数据同时向寄存器堆写入数据。

三、实验过程（50%）

（一）实验流水账

日期	工作	结果
----	----	----

日期	工作	结果
9.8.2021 20:30-22:00	阅读研讨课讲义相关章节	基本读完第四章 lab1 相关部分
9.9.2021 14:30-7:00	开始 debug	修改出 5 个致命性 bug
9.10.2021 10:00-11:00	继续 debug	修改出 3 个致命性错误，并上板验证
9.20.2021 10:00-16:00	阅读讲义并完成 lab4	Lab4 完成
9.21.2021 10:00-10:30	完成 lab5	Lab5 完成

（二）错误记录

重点记录调试过程和机理分析。请以**图文结合**的方式进行描述，如有波形图应当**分组（Group）分明、分割（Divider）清晰、有标志线（Marker）指示关键时刻**。

1、错误 1：信号未赋值

（1）错误现象

多个信号处于不定态，且 run all 后长时间不停止（此处未能保留截图，故仅给出文字描述）

```
reg      ds_valid ;
wire     ds_ready_go;
```

Figure 4 声明 ds_ready 处

而后续整个 if_stage 文件中并未出现对其的赋值。

（2）分析定位过程

由于 mycpu_top 中的 debug_wb_pc 信号一直是 x 态，故以此为引子，逐步向上一层流水线追溯，直至 if_stage 文件中的 ds_valid 信号，发现其并未被赋值。

（3）错误原因

信号被声明后却未能赋值。

（4）修正效果

该信号的意义在于判断 if_stage 这一层流水级是否已经储存有效数据。

（5）归纳总结（可选）

单纯声明一个变量而不予以赋值，这类错误其实很难犯，而且即使是出现了也不会很难排除。

2、错误 2：信号未赋值

（1）错误现象

```
[ 2067 ns] Error!!!
reference: PC = 0x1c000000, wb_rf_wnum = 0x0c, wb_rf_wdata = 0xffffffff
mycpu      : PC = 0x1c000000, wb_rf_wnum = 0x0c, wb_rf_wdata = 0xxxxxxxxX
```

Figure 5 lab3 的第二个错误

PC 的值归于正常，但写回的数据仍然处于 x 态

（2）分析定位过程

基于与上一个 bug 类似的 debug 方式，从 top 层开始追索，直至发现其 ds_to_es_bus 信号存在一位 Z，换言之，该位信号（load_op）被声明了确未能赋值。

(3) 错误原因

信号被声明后却未能赋值。

(4) 修正效果

按照 loooooonarc 指令集中 ld.w 指令给其赋值,其后进入下一个错误。



```
assign load_op = inst_ld_w;
```

Figure 6 第二个错误的修正方式

(5) 归纳总结 (可选)

3、错误 3 加法出问题 (写入 alu 的地方有问题)

(1) 错误现象

依然是返回值报错,其中金标准为 ffff 但实际返回值为 fffe。(此 bug 忘记截图)

(2) 分析定位过程

在定位到 MEM 流水级时,发现写回的是 alu 运算的结果,遂去检查 alu 模块,起初发现 alu_src1 和 alu_src2 都是 ffff,且 alu_op 为 001,换言之是加法,所以实际写回的并没有错误。但后来意识到,可能是 alu_src1 和 alu_src2 出错,故发现错误在 alu 的输入中。

(3) 错误原因

向 alu 模块 src1 和 src2 接口输入的都是 src2。

(4) 修正效果

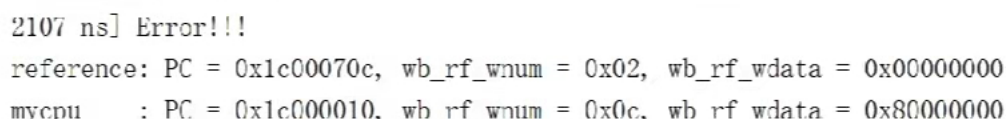
把接口重连就好了,此后进入下一个 bug。

(5) 归纳总结 (可选)

说说你觉得这个错误是哪种类型的,今后如何提前规避。

4、错误 4 跳转出错 (br-bus 位宽问题)

(1) 错误现象



```
2107 ns] Error!!!
reference: PC = 0x1c00070c, wb_rf_wnum = 0x02, wb_rf_wdata = 0x00000000
mycpu    : PC = 0x1c000010, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x80000000
```

Figure 7 错误 4 现象

(2) 分析定位过程

直接找到 IF 流水级,发现此处的跳转信号异常。

(3) 错误原因

由于 br_bus 高位是决定是否发生跳转的控制信号,而由于其位宽被少赋值了一位,故该信号处于异常态,进而导致 nextpc 对跳转的判断出现问题。

(4) 修正效果

修正方式为在 mycpu.h 文件中 BR_BUS_WD 直接修改大小,其后进入下一个 bug。

(5) 归纳总结 (可选)

在报告二中已经详细阐述了。

5、错误 5 还是跳转（b 指令后多执行了一个，如果是 mips 就没问题）

（1）错误现象

```
[ 2117 ns] Error!!!  
reference: PC = 0x1c000710, wb_rf_wnum = 0x03, wb_rf_wdata = 0x00000000  
mycpu    : PC = 0x1c000388, wb_rf_wnum = 0x04, wb_rf_wdata = 0xbfaaff000
```

Figure 8 错误 5 现象

（2）分析定位过程

本错误为 lab3 中调试难度相对最高的一个（当然，如果掌握反汇编的话应当是能立刻反应过来的）。我实际调试的过程中发现其应当是一个跳转指令，但 nextpc 并没有进行相应的调整（虽然有一拍的确存入了正确的跳转地址，但并未能及时跳转，而是执行了一条莫名其妙的指令），故联想到这不是 mips 指令集，因而成功 de 出。

实际上，如果掌握反汇编调试的办法，应当可以一眼看出在 bl 指令后其并未正确执行之后的 add 指令，甚至可以在不拉波形的情况下作出正确的判断。

（3）错误原因

错误原因为执行了紧随着跳转指令后一拍的指令（即 PC+4），而其本应取消该指令，而紧接着去执行目标指令。

（4）修正效果

由于只需要在 ID 阶段判断出 br_taken 拉高时无效化此时 IF 阶段取进来的指令即可，故直接修改 fs_to_ds_valid，在 br_taken 拉高时不让 IF 取进来的指令进入 ID 即可。修改后进入下一个 bug。

```
assign fs_to_ds_valid = ~br_taken & fs_valid & fs_ready_go
```

Figure 9 错误 5 修正

（5）归纳总结（可选）

本错误属于 loongarch 和 risc-v 等指令集的特点与 mips 指令集特点的冲突产生的错误之一。由于 mips 指令集在设计的时候有分支延迟槽，这虽然在上学期计组实验课程中给我带来了一点麻烦，但在应对流水线的时候，则可不必要费心考虑跳转后指令的处理问题了。由于本实验报告的写作流程为边实验边写大纲，全部实验完成后进行细节补充，故在此可以继续说明，在随后的阻塞和前递中，关于分支跳转的处理会更加复杂（虽然已经在讲义中给出了）。同时，mips 和 loongarch 的不同将在后续给我带来更有趣的 bug。

6、错误 6 移位问题（alu 移位移反了）

（1）错误现象

```
[ 30077 ns] Error!!!  
reference: PC = 0x1c063344, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x01000000  
mycpu    : PC = 0x1c063344, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x00000070
```

Figure 10 错误 6 现象

（2）分析定位过程

该 bug 又回到了 alu 的运算上，实际上，阅读完汇编代码，并用二进制表示 alu_src1 和 alu_src2 后几乎一眼就

能看出问题：移位移反了。

(3) 错误原因

Alu 运算错误，移位移反了。

(4) 修正效果

直接修正方向，其后进入下一个 bug

```
// SLL result
assign sll_result = alu_src2 << alu_src1[4:0];

// SRL, SRA result
assign sr64_result = {{32{op_sra & alu_src2[31]}}, alu_src2[31:0]} >> alu_src1[4:0];
```

Figure 11 错误 6 的修正方式

(5) 归纳总结（可选）

这个错误其实很难犯，因为毕竟上学期已经写过了。

7、错误 7 组合逻辑环（alu 的 or）

(1) 错误现象

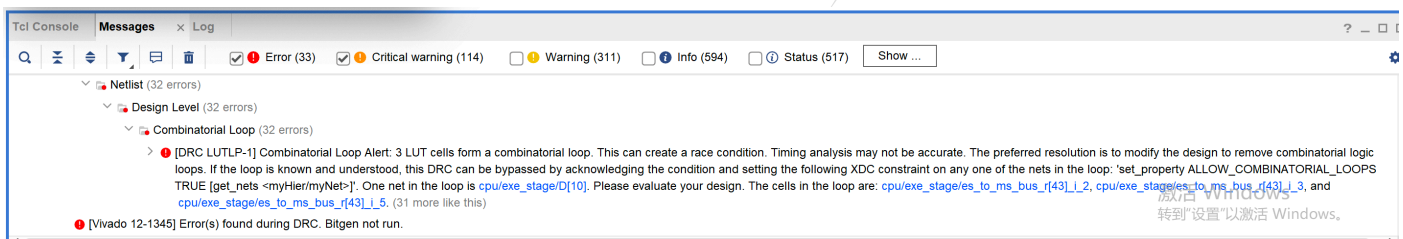


Figure 12 错误 7 的现象 i

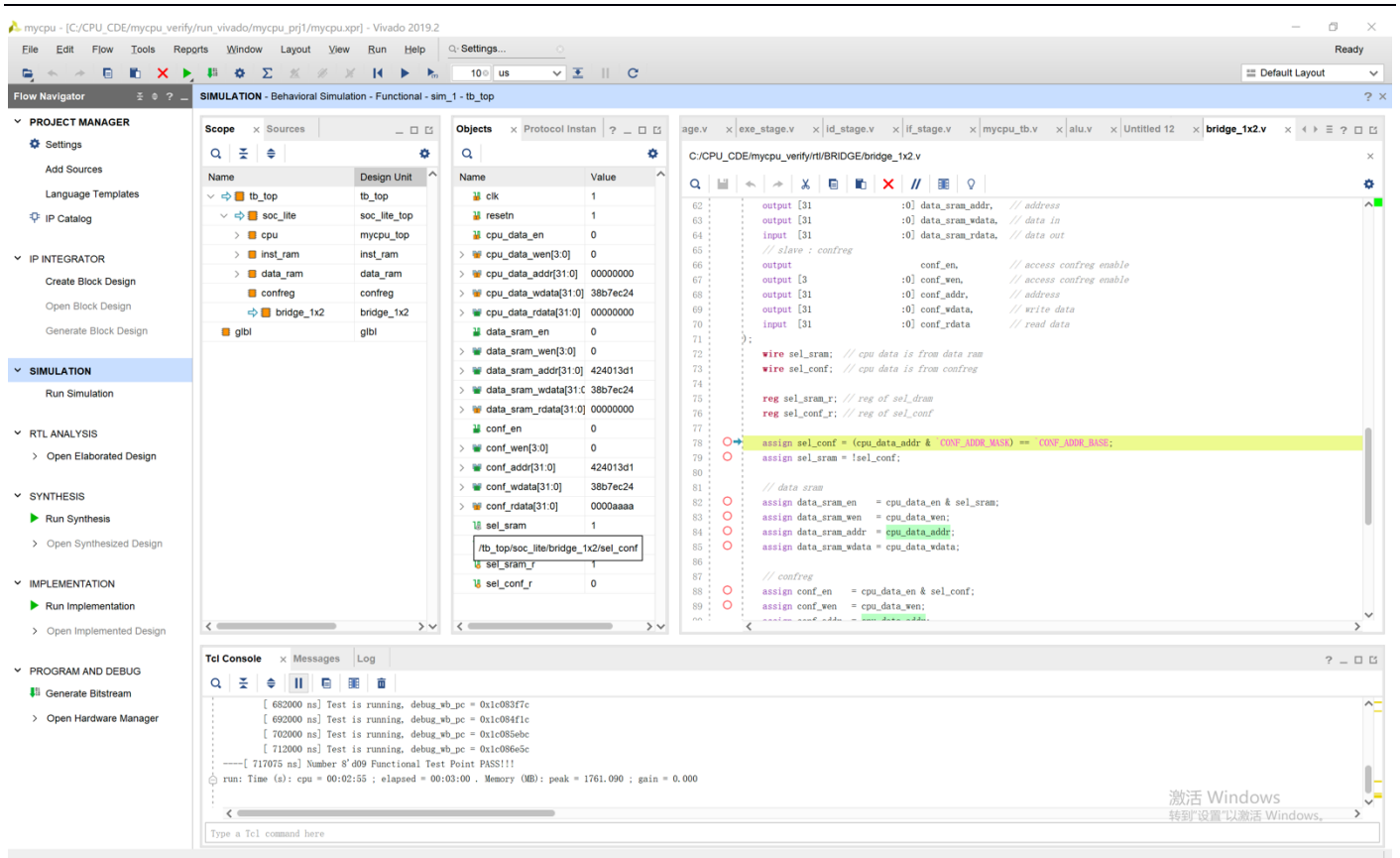


Figure 13 错误 7 的现象 ii

(2) 分析定位过程

这个和初次实验中遇到组合逻辑环时的现象几乎一样，卡在一个时刻不再往前走，根据报错可以发现是在 `es_to_ms_bus` 中，但这个范围太模糊了，需要检查的信号也很多，但直觉告诉我大概率又是出在 `alu` 中了，于是直接追索各类 `result`，其后发现是进行 `or` 运算时存在组合逻辑环。

(3) 错误原因

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

Figure 14 错误 7 原因

(4) 修正效果

修正后可以继续运行，并到下一个 `bug`。

(5) 归纳总结（可选）

组合逻辑环的问题在 `lab2` 中已经提及了。

8、错误 8 写回的数据有问题

(1) 错误现象

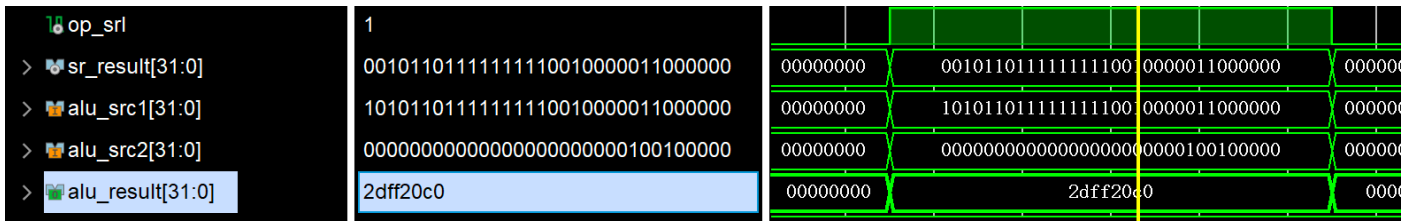


Figure 15 错误 8 出现时的波形

(2) 分析定位过程

写回的数据出现问题，于是我直接去找 alu，发现的确是 alu 运算的结果作为写回的结果提交，故开始看 alu 的运算，发现该运算的结果在高位存在问题，自然而然发现其位宽不对。

(3) 错误原因

位宽错误。

```
// SRL, SRA result
assign sr64_result = {{32{op_sra & alu_src1[31]}} , alu_src1[31:0]} >> alu_src2[4:0]; //rj >> i5
assign sr_result = sr64_result[30:0];
```

Figure 16 错误 8 的具体位置

(4) 修正效果

将位宽+1 即修正完毕，其后 pass。

(5) 归纳总结（可选）

位宽错误的概述在其上已经说过了。

9、杂七杂八的不影响跑通的错误：（decoder 最高位没赋值）

(1) 错误现象

本处的错误为不影响 lab3 测试但显然有问题的地方，如 decoder 中未能给最高位赋值

```
generate for (i=0; i<63; i=i+1) begin : gen_for_dec_6_64
    assign out[i] = (in == i);
end endgenerate
```

Figure 17 错误 9 现象

(2) 分析定位过程

偶然发现。由于调试过程中曾经检查过 decode 部分，发现本处赋值存在问题。

(3) 错误原因

循环少了一次

(4) 修正效果

将<改为<= 即修正完毕。

10、写后读阻塞时 pc 跳转错误：

1) 错误现象

在处理写后读问题时，nextpc 未能保持在正确的跳转位置，而是在阻塞过程中被新取进来的错误 PC 替代掉了。

```
[ 473177 ns] Error!!!
```

```
reference: PC = 0x1c00b624, wb_rf_wnum = 0x0a, wb_rf_wdata = 0x4030e000
```

```
mycpu      : PC = 0x1c00b654, wb_rf_wnum = 0x1c, wb_rf_wdata = 0x4030e000
```

Figure 18lab4 中 br_taken_cancel 赋值错误的复现

(2) 分析定位过程

本错误定位过程是很简单的，直接拉波形到 IF 流水级的 pc 和 nextpc 信号，发现在流水级因为写后读被阻塞时，pc 依然在随着时钟周期更新，正确的 PC 没有取进来，但新 PC 被取进来且留存。

(3) 错误原因

首先注意到 pc 的更新条件。

```
else if (to_fs_valid && fs_allowin) begin
    fs_pc <= next_pc;
end
```

Figure 19nextpc 的更新条件

可以注意到，这个更新条件依赖于 to_fs_valid 信号和 fs_allowin 信号，而在写后读且 br_taken 拉高时，readygo 信号拉低，导致流水线阻塞，进而导致此时的 nextpc（正确的 pc）无法进入，而在随后的周期，当写后读被处理之后，错误的 pc 被读进来，并进入之后的 ID 流水级，导致错误。

实际上，这个错误在讲义上已经提及了：

最后再补充一个关于 br_taken_cancel 信号有效时机的注意事项。由于转移指令可能需要对源操作数进行运算处理后才能得到是否跳转以及跳转目标，所以这类指令即使到达了译码阶段呢，也可能因为存在寄存器的写后读相关而被阻塞住，那么在其获得正确的源操作数之前（也即写后读相关阻塞解除之前），一定不能将 br_taken_cancel 信号置为有效。

Figure 20 讲义中关于此错误的修正方式

在此必须提及，相较于笔者已经完成的计算机组成原理实验课和正在修读的操作系统研讨课，体系结构研讨课的讲义可谓是细致入微了，如果不是对自己将要完成的部件早已有十足的把握或者独立于讲义的想法，在进行试验前先熟悉一遍讲义能有效的提高设计速度和精度。

(4) 修正效果

修正方式即定义：

```
assign br_taken_cancel = br_taken & ds_ready_go;
```

Figure 21 错误 10 的修正方法

(5) 归纳总结（可选）

对于流水级中各个信号在因为不同原因阻塞时的表现应当有充分的了解以后才能有效避免此类错误，不过有

有趣的是，有同学并未使用 `br_taken_cancel` 信号，而是对 `br_taken` 信号后与上了 `ds_ready_go`，也能通过测试，这也不失为一种处理跳转时遭遇写后读问题的方法。

11、错误 11：比较写后读时寄存器号的问题

(1) 错误现象

在下图中第二或第三个分支跳转指令出现的问题，发生意外跳转

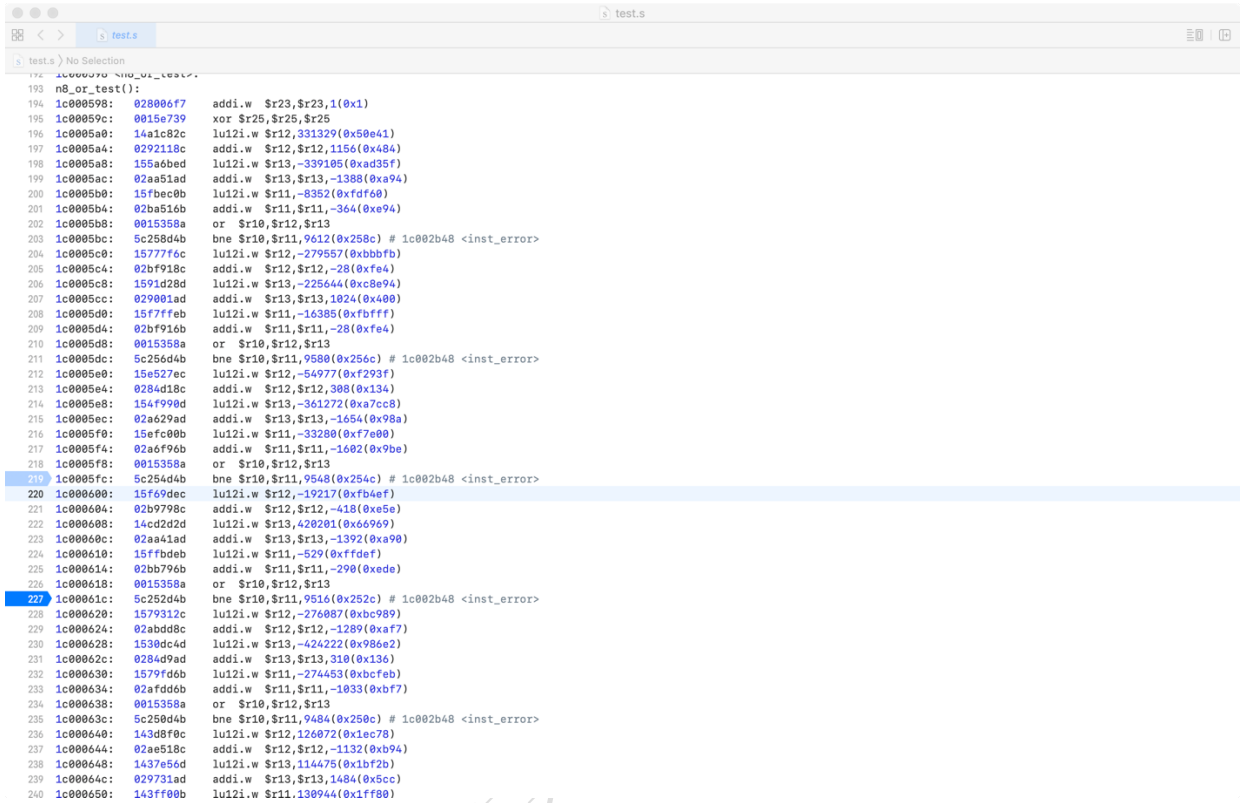


Figure 22 错误 11 在汇编文件中的位置

非常吊诡的是，在 lab4 中，我修改此 bug 后通过测试，而后我在 lab5 中试图复现这种错误，却意外发现即使是修改 bug 前后的区别，也能通过 test，令人感到疑惑。

(2) 分析定位过程

定位过程依然是先找到跳转的问题，而后发现 `ready_go` 信号的表现和理论上的表现存在异常，在应当被阻塞时却照常运行，而最令人感到困惑的是，其在前几个跳转中未出差错，而在本次跳转中却出现问题，且这几个测试指令基本上是完全一致的，更让人难以下手。最后重新翻看 loongarch 的相关指令，仍然不知所以，在同学的建议下将 ID 流水级中 `dest` 和该阶段读寄存器号的比较修改为 `dest == raddr1 | dest == raddr2`，而不是与 `rj` 和 `rk` 做比较后，通过了测试。其后对 lab5 前递的修改中也未再改动此处，也能通过测试。

(3) 错误原因

/*截止本报告提交时刻，我还没能想出其原因，起初我猜测这是因为 mips 和 loongarch 指令集在对于 rd 号寄存器的使用中存在差异，即前者中 rd 单纯作为目标寄存器，而后者中 rd 可以作为源寄存器，但阅读说明书后，我并没有在其中找到支持这一观点的论据。*/

指令格式: add.w rd, rj, rk add.d rd, rj, rk
sub.w rd, rj, rk sub.d rd, rj, rk

Figure 23 loongarch 指令集部分截图 i

指令格式: lu12i.w rd, si20 lu32i.d rd, si20
lu52i.d rd, rj, si12

Figure 24 loongarch 指令集部分截图 ii

ori rd, rj, ui12
xori rd, rj, ui12

Figure 25 loongarch 指令集部分截图 iii

在重新翻了一遍指令集以后，发现我的猜测很可能是正确的，因为

beq rj, rd, ofs16
bne rj, rd, ofs16
blt rj, rd, ofs16
bge rj, rd, ofs16
bltu rj, rd, ofs16
bgeu rj, rd, ofs16

Figure 26 分支跳转指令，可以注意到其需要读 rd 寄存器

或许问题就是出在跳转时需要读 rd 寄存器和 rd 寄存器正在被写入的冲突。

(4) 修正效果

如 (2) 中所写的解决方式实施后，问题得到解决，lab4 完成。其后的 lab5 是一遍过的。

(5) 归纳总结 (可选)

多看一看指令，之后的优化也需要熟悉这些指令。

(三) 提速效果

在 lab3 中，其最终效率为 1402735ns，在第一次做前递时，我对前递中 es 流水级阻塞的理解出现偏差，因此依然保留了对 es 全部写后读情况的阻塞，运行时间为 833435ns，本以为提速已经很明显，但在和同学比较以及重新阅读讲义后发现前递中阻塞几乎只有一种情况，即 ld 指令在 es 级中由于同步 RAM 的工作原理而引发的 1 拍的阻塞，故修改阻塞条件，最终运行时间为 610115 ns

$$\text{加速比为 } \frac{1402735}{610115} = 2.299$$

利用阻塞处理数据冲突显然会导致流水线效率降低，在极端情况下（冲突数量较大）会逼近多周期 CPU 的效率，利用前递技术可以有效的提高其效率。但在前递操作中，原本提高了主频的流水级的切分工作在失去了部分效果，自 ALU 入口出发至最终的两个源操作数构成的长组合逻辑不利于主频的提升。

四、实验总结 (可选)

本次试验显然比 lab2 难度有了显著的提高（虽然还是不难），不过难度远低于上学期的单周期 CPU 设计。但在本次实验中，我对流水线的结构，数据通路和运行都有了更加细致的了解。相较于上学期的课件，本学期的讲义对实验的帮助和对实验内容理解的帮助的确是高到不知道哪里去了。

此外，本次实验也着实让我主动去了解 loongarch 指令集，并且发现了一些 loongarch 和 mips 指令集的不同之

处，但有一点必须吐槽，截止目前，我上过的计算机系专业课中，要求掌握或者至少有所了解且能读懂的汇编语言，已经包括 mips、risc-v，AT&T 和 loongarch 四种，尽管这几者是有相似之处的，但是还是希望之后对课程中使用的语言加以统一，至少组成原理和体系结构尽量做到统一。

国科大 B62009H 计算机体系结构研讨课
17-18 秋季