Lab03

2021K8009929033　　马迪峰　5 号箱

实验任务

- 任务要求:

1. 结合本章内容中讲述的设计方案，阅读并理解 mycpu_env/myCPU/目录下提供的代码。
2. 完成代码调试。我们提供的代码中加入了若干错误，请大家通过仿真波形的调试修复这些错误，使设计可以通过仿真和上版验证

- 实验理解:

本次实验相比上次的调试而言难度提升了许多，一方面是多了许多模块，另一方面则是调试手段也变得更加高级。完成这个实验最重要的在于理解 Soc_lite 实验环境的工作原理和设计。本次实验新增了基于 trace 比对的调试辅助手段，用一个功能完备的参考处理器核所生产的 gloden_trace 来作为监视，再从所设计 CPU 中抓取比对信号搭建 testbench 激励环境，可以快速定位出错源头。当然测试 CPU 少不了测试程序集的支持，所以 func 文件夹就可以提供基于 LOOGARC 指令集的测试程序来验证 CPU 各指令实现是否正确。所以明白 CPU 的设计思路和具体实现，就可以快速找到错误原因。

实验设计

实验中基本模块均已实现，具体单周期设计过程在指南中的介绍也十分详细，不需要再额外强调。需要注意到的就是实验环境的配置，如果不是通过压缩包解压的文件则需要额外完成一些操作，这些过程能帮助理解实验流程。

错误记录

（一） 实验流程

本次实验比较简单，只需要不到一天的时间即可完成，但大体还是分为两步，组内成员均完成仿真测试通过后，再一起进行上版测试，最后完成实验任务。

（二） 错误记录

错误 1：变量未声明

（1） 错误现象

第一次运行时，发现写数据一直为 0，与此同时 PC 为不定态，debug_wb_rf_we 信号恒为高阻态

（2） 修正过程

debug_wb_rf_we 的错误较容易发现，因为高阻态的成因往往都是因为悬空未连接信号线，检查后发现 cpu 中错写为了 debug_wb_rf_wen，导致仿真时连接不到 debug_wb_rf_we 信号，呈现高阻态。对于 PC 的问题，观察写数据的生成过程，控制信号和两个选择信号的生成均正常，此时控制信号为 0，理应选择

alu_result 的结果，但是实际上写数据却是 0，这个问题不太容易发现。最终发现 final_result 信号并未声明，在运行时 verilog 会将其隐式声明为 1bit 变量，导致赋值时不为完整结果出现错误。

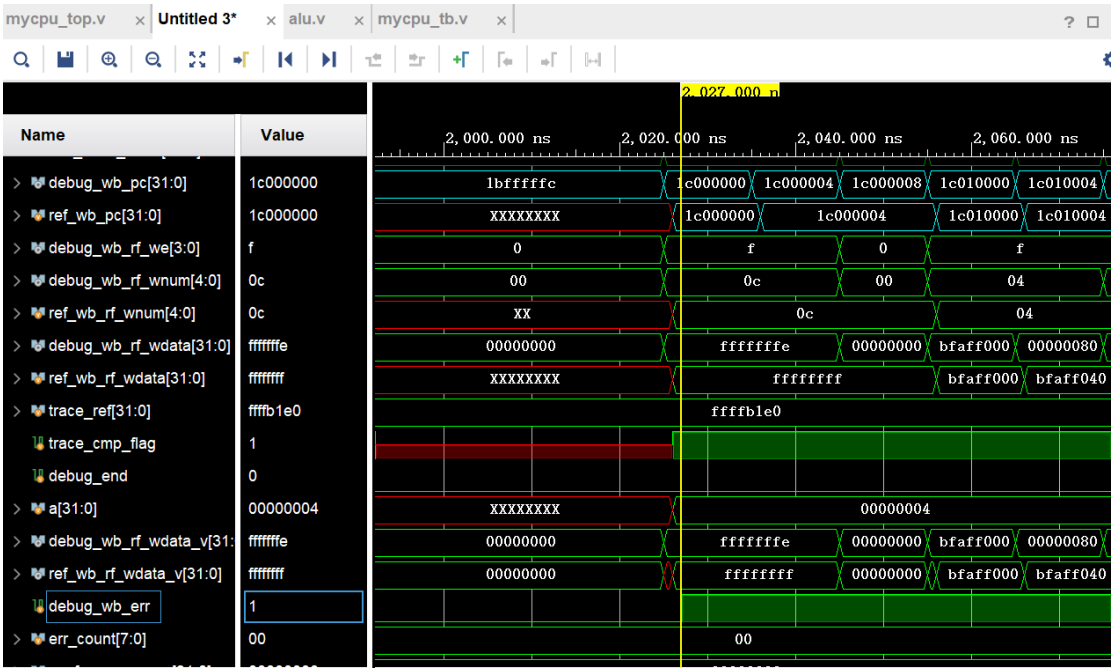
```
assign mem_result   = data_sram_rdata;
assign final_result = res_from_mem ? mem_result : alu_result;
```

添加声明 final_result 为 32 位向量后，恢复正常。

错误 2 ：addi 指令出错

（1） 错误现象

寄存器写数据发生错误，debug_err 信号拉高，观察发现写数据与参考值不同。



检查写数据的生成信号，因为 res_from_mem 为 0，实际赋值为 alu_result，所以只需要检查 alu_result。

（2） 纠正思路

alu_result 的生成跟很多因素有关：

（a） alu_op：检查后发现与指令码相符，确实为 addi 加操作

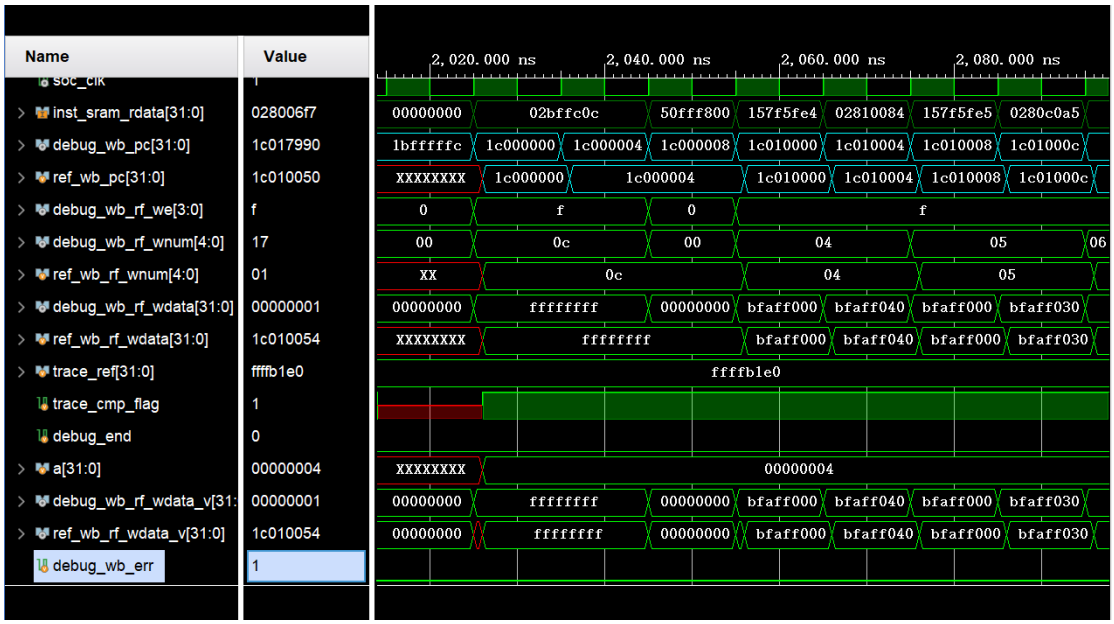
（b） 源数据：源数据又分为操作数 1 和操作数 2，操作数 1 不需要考虑，它恒为 rf_raddr1 即 rj 地址码对应寄存器中的数据，所以检查 alu_src2，该数据可能为立即数，也可能为 rk 寄存器所读出的值。因为为 addi 操作类型，控制信号 src2_is_imm 也符合预期，所以理应为 imm 值，检查后发现均正确。

（c） 例化：以上检查都没问题，只能是例化出现错误，检查后确实发现例化时连接端口出现错误。

```
alu u_alu(
    .alu_op     (alu_op    ),
    .alu_src1   (alu_src2  ),
    .alu_src2   (alu_src2  ),
    .alu_result (alu_result)
    );
```

将连接 alu_scr1 的端口改为 aku_src2 即可。

（3） 修正效果



错误 3：BL 指令出错

（1）错误现象

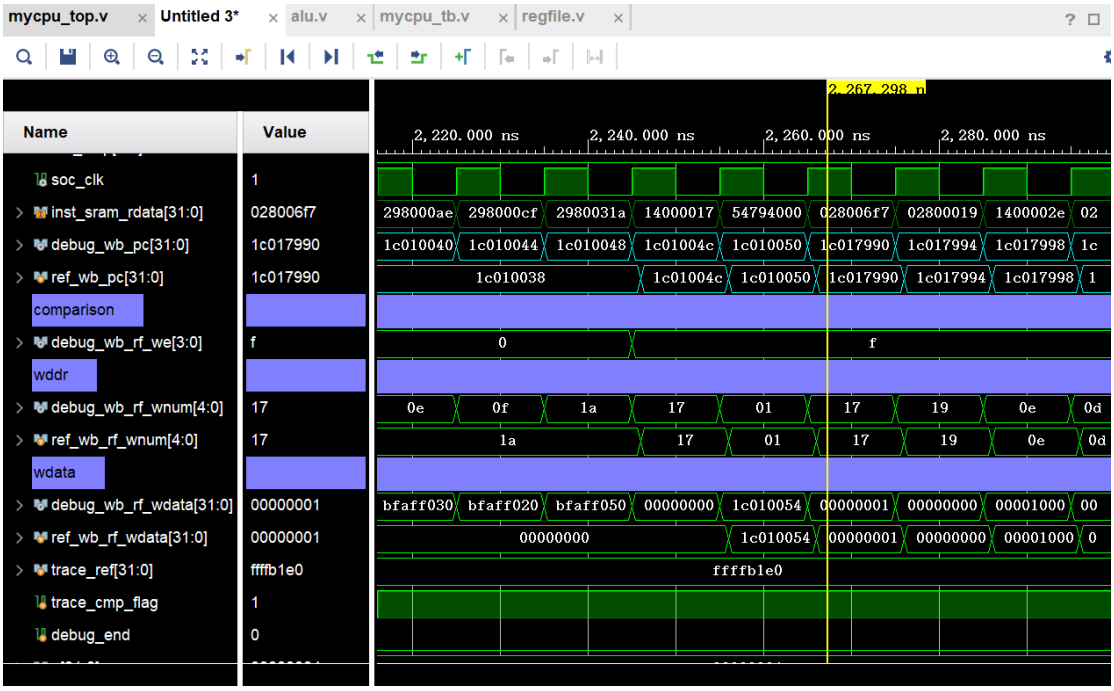
修正上述错误后再次进行仿真，2267ns 时发现参考的 pc 值和实际上的 pc 值不同，同时写地址和写数据均不相同，功能正确的 CPU B 指令并未进行跳转，而出错的 CPU 进行跳转，出现错误。同时多处写数据和写地址并不对应。

（2）修正过程

此时仍为 addi 指令，但是之前已修正过该指令，往前观察发现上一个周期为条件分支指令，猜测是因为未正确跳转。上一周期为 BL 指令，该指令需要跳转并将更新后的 PC 值写回 1 号寄存器，因此写使能信号应当为真，但实际上并为如此。更改后如下：

```
assign gr_we         = ~inst_st_w & ~inst_beq & ~inst_bne & ~inst_b;//inst_bl need to wirte
```

（4） 修正效果



错误 4、SLLI.W 指令出错：

（1） 错误现象

4785ns 时写数据出错，猜测可能是因为 ALU 计算出现错误

（2） 修正过程

ALU 中左移操作操作数有误，因为 alu_src1，修改后计算结果正确。

错误代码：

```
// SLL result
assign sll_result = alu_src2 << alu_src1[4:0];   //rj << i5
```

错误 5、OR 指令出错

同上原因仍在 ALU 中，将 OR 操作更改正确即可

正确代码：

```
assign or_result  = alu_src1 | alu_src2;//mistake :or
```

错误 6、SRLI.W 指令出错

（1） 错误现象

5195ns 时，写数据出现错误，此时指令为逻辑右移指令，猜测错误原因仍可能是在 ALU 中

（2） 修正过程

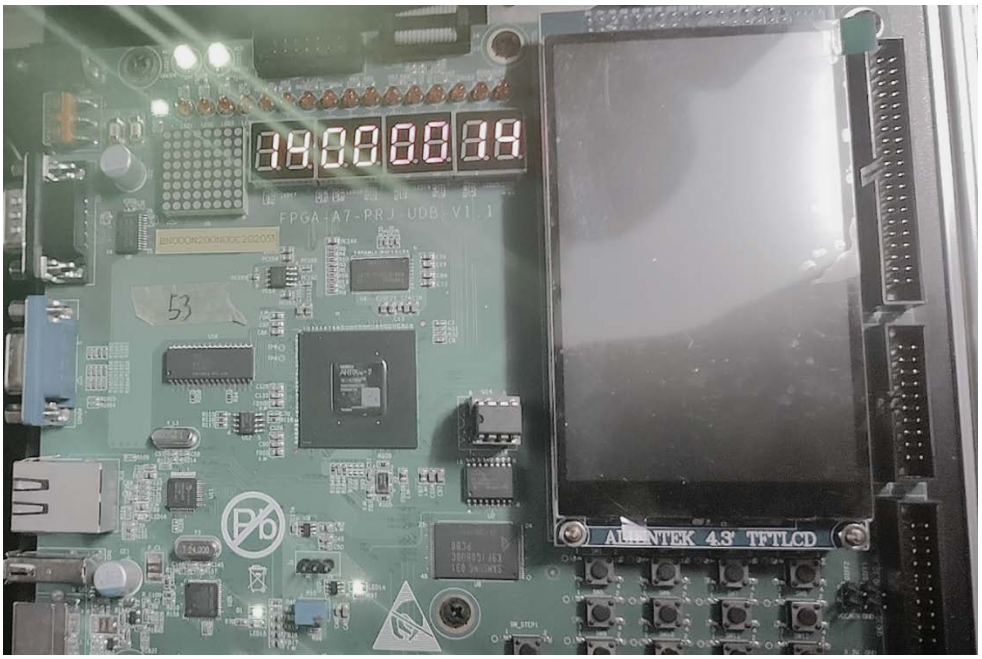
在 ALU 中找到逻辑右移指令的实现过程，发现其操作与 SLLI.W 的错误类型一样源操作数出现错误，调换两者即可。

正确代码：

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

assign sr_result   = sr64_result[31:0];//mistake
```

上板测试：



结果正确，通过调节拨码开关可以使数码管跳动变缓。

实验总结

本次实验的主要任务是熟悉基于 trace 的实验仿真调试环境，20 条单周期处理器核的设计并不复杂，但是其中数据通路的设计和关键部件的定制比较关键，理解这一部分的构造可以扩展实现更多指令。利用 Soc_lite 来搭建功能测试环境，同时利用基于 trace 的调试手段可以快速定位出错位置，极大加快了调试速度。

吐槽环节：

没什么好吐槽的，虽然没有比计组那么自动化，但是至少我明白计组研讨调试的工作原理是什么了，收获还是很大。不过有几处 BUG 设置地比较容易发现，比如 ALU 的设计代码已经在指南书中展示过，很容易就能发现，但是隐式声明和位数错误这种错误比较难发现，确实容易出现错误以及译码错误等。

cpu_design_guidebook 问题总结：

本部分罗列一些比较五、六章有启发的内容

- 设计一个 CPU 就是设计它的“数据通路+控制逻辑”
- 任何时候 CPU 上运行的程序中出现的地址都是虚地址，而 CPU 本身访问内存、I/O 设备所用的地址都是物理地址
- 模块划分思路
- 分支指令的立即数跳转偏移 offs 是左移两位后再与其自身 PC 相加，四字节对齐，左移两位可以获得更大的跳转范围
- 调试思路

  等等……

思考题:

- 思考示例中出现的一条指令“jr $ra”,是不是宏指令？如果是，又是对应哪种具体情况下的机器指令。

答："jr $ra" 的含义是跳转到寄存器 $ra（返回地址寄存器）中存储的地址。在实际的架构中，没有专门的机器指令来执行这个操作。因此 "jr $ra" 是一个伪指令，它由汇编器转换成真正的机器指令，

可能是 ：

"jalr $ra,$ra,$0"，即跳转到$ra+0 的地址，并将其加 4 的结果存入 ra.（下一条指令）

这是跳转并链接寄存器的指令，它将控制流返回到 $ra 寄存器中的地址。

- 为什么判断写回目的寄存器非 0 时才采样?

答：确保采样的数据不受 0 号寄存器的影响。因为 0 号寄存器的值恒为 0，通常不会出现写入 0 号寄存器的操作，并且如果采样数据需要与 0 号寄存器的值进行比较或计算，那么写回 0 号寄存器就会引入难以捉摸的不确定性，以便在测试中更容易分析结果，减少测试过程中潜在的误差来源。