实验 3 报告 3

第 32 小组
施璠、袁峥

一、实验任务（10%）

1. 设计一款静态 5 级流水简单 MIPS CPU。

2. 本次实验要求延续 lab2 实验中的以下要求：

(1) CPU 复位从虚拟地址 0xbfc00000 处取指。

(2) CPU 虚实地址转换采用：虚即是实。

(3) CPU 对外访存接口为取指、数据访问分开的同步 SRAM 接口。

(4) CPU 只实现一个操作模式：核心模式，不要求实现其他操作模式。

(5) 不要求支持例外和中断。

(6) CPU 顶层连出写回级的 debug 信号，以供验证平台使用。

3. 整个实验中，最后要求实现 MIPS I 指令集，除了 ERET（非 MIPS I）、MTC0、MFC0、BREAK、SYSCALL 指令，其余指令均要求实现，共 56 条指令。

(1) 要求实现 MIPS 架构的延迟槽技术，延迟槽不再设定为 NOP 指令，可能是任意指令。

(2) 控制相关由分支指令造成，通过延迟槽技术可以完美解决。

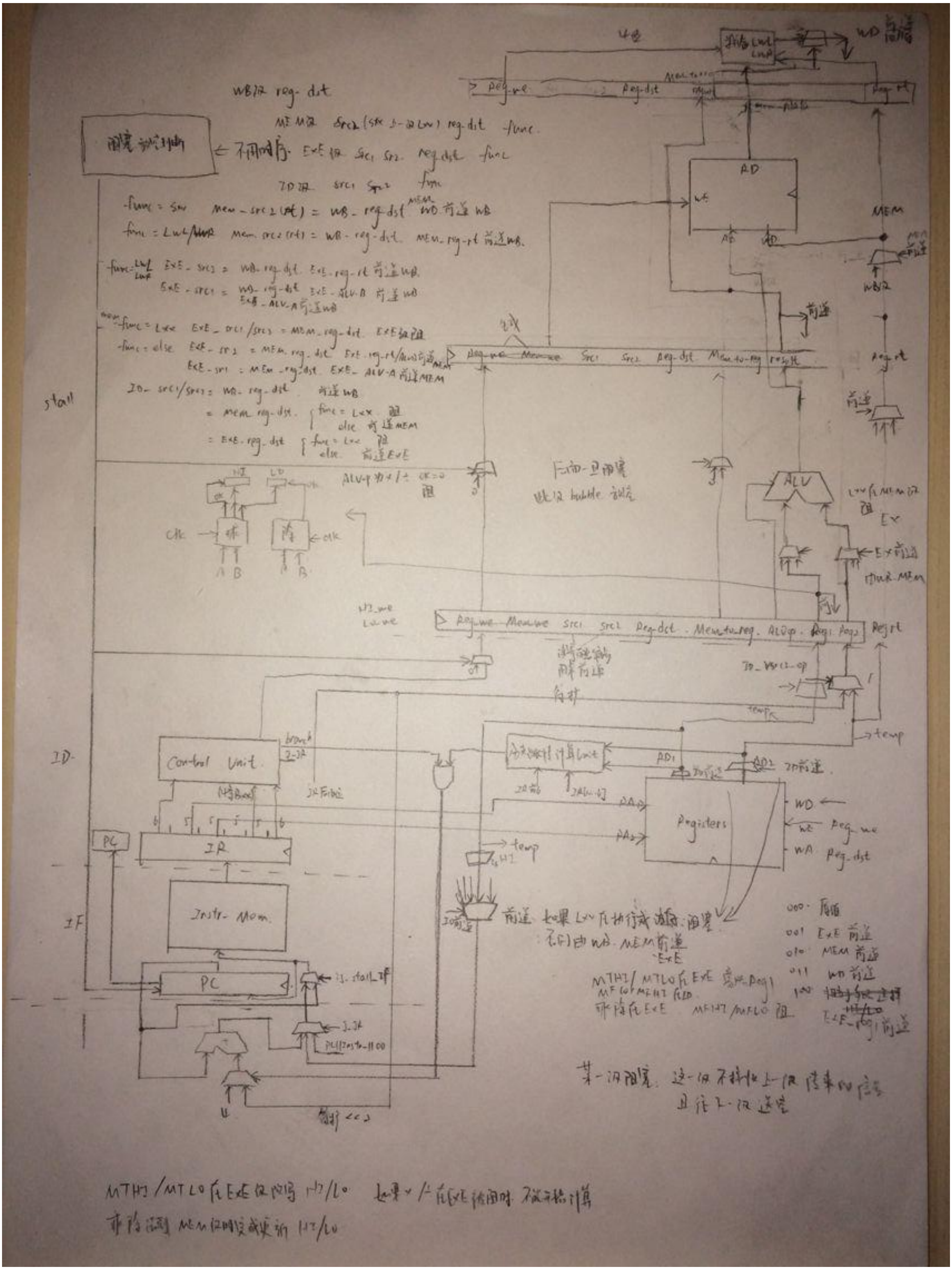
(3) 结构相关即某一级流水停顿了，会阻塞上游的流水级。

(4) 要求数据相关采用前递处理。

(5) 乘除法指令实现可以调用 Xilinx 的乘除法 IP，推荐能力有余的同学自行编写乘除法器，乘法采用 booth 算法+华莱士、除法采用迭代算法。

二、实验设计（30%）

整个 CPU 设计共分为 12 个模块，分别为 ALU、IF_stage、ID_stage、EXE_stage、MEM_stage、WB_stage、next_pc、regfile、stall、mycpu、divider 和 multiplier。其中 IF_stage、ID_stage、EXE_stage、MEM_stage 和 WB_stage 分别为 CPU 五级流水的阶段，ALU 模块在 EXE_stage 级进行算逻运算，next_pc 产生下条指令地址，regfile 为寄存器堆，stall 为流水线的阻塞总调度，mycpu 是整个设计的顶层，负责其他各个模块的整体调度，divider 和 multiplier 为自行设计的乘除法器。整体设计图如下：


Control Unit
IR
PC
Instr. Mem.
Registers
ALU
AD
MEM
stall
ID
IF

（一）除法器模块

（1）基本概述

本周更新了原来的除法器设计，原来的是将有符号除法和无符号除法分开完成的，现在将两个模块整合到了一起，实现了一个可以同时完成有符号除法和无符号除法的除法器。基本原理是采用原码加减交替法。具体步骤如下：

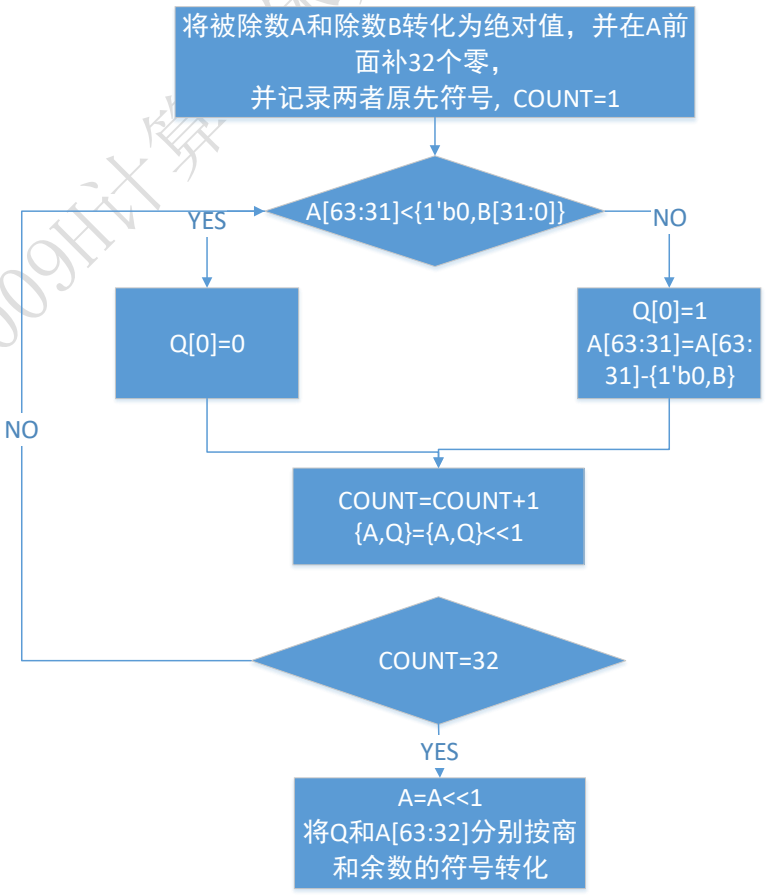
1、 首先将读入的被除数和除数转化为绝对值，并记录符号，用以确定商和余数的符号。

2、 迭代运算得到商和余数的绝对值。首先在被除数前补充 32 个零，并每次用最高 33 位与除数进行比较，如果大于除数则上商 1，并将被除数高 33 位减去除数，否则上商 0。然后将被除数和商共同左移 1 位，并进行下一次计算。

3、 步骤 2 共迭代 32 次后，最终得到商和余数的绝对值。根据步骤 1 记录的符号将商和余数进行转化。

以上步骤为有符号除法的步骤，无符号除法只需完成步骤 2。按照以上步骤，完成一次除法运算需要 32 拍，我们将步骤 1 放在第一拍一起完成，将步骤 3 放在第 32 拍一起完成。

（2）流程图



（3）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，低电平有效
sign	IN	1	有无符号运算信号，0 为无符号运算，1 为有符号运算
dividend	IN	32	被除数输入
divisor	IN	32	除数输入
div_A_valid	IN	1	被除数输入有效信号
div_B_valid	IN	1	除数输入有效信号
div_out	OUT	64	除法运算结果输出
div_validout	OUT	1	除法运算结果输出有效信号

三、实验过程（60%）

（一）实验流水账

本周主要进行了乘除法器的更新，两个人分别完成乘法器和除法器的实现。其中乘法器暂时还未完成，仍用乘号替代，除法器基本功能完成，后续还可以进行优化。（两人大约各自花费 10 小时）

本周还进行了阻塞的优化，将原先一些不需要的阻塞去除，使 CPU 的效率提高了大约百分之 10。(花费大约 5 小时)

（二）错误记录

1、错误 1

（1）错误现象

在仿真时发现阻塞过多。

（2）分析定位过程

到测试数据中查看具体阻塞的指令，发现在进行算逻运算时，如果操作数在前一拍中更新，则会被阻塞。

（3）错误原因

由于设计中已经完成前递的实现，因此此类状况本不应该出现，查看具体阻塞信号的赋值时，发现了问题主要在取址和译码级的阻塞信号。

```
assign ID_stall = (MEM_l_type && (EXE_src1 == MEM_reg_dst && EXE_src1 != 5'd0 || EXE_src2 == MEM_reg_dst && EXE_src2 != 5'd0)) ||
                  (MEM_l_type && (ID_src1  == MEM_reg_dst && ID_src1  != 5'd0 || ID_src2  == MEM_reg_dst && ID_src2  != 5'd0)) ||
                  (EXE_l_type && (ID_src1  == EXE_reg_dst && ID_src1  != 5'd0 || ID_src2  == EXE_reg_dst && ID_src2  != 5'd0)) ||
                  (EXE_mul_div_type && !EXE_mul_div_validout) ||
                  (EXE_mul_div_type && (ID_inst[31:26] == `MFHI && ID_inst[5:0] == `MFHI_2 || ID_inst[31:26] == `MFLO && ID_inst[5:0] == `MFLO_2)) ||
                  ((ID_src1 == EXE_reg_dst && ID_src1 != 5'd0 || ID_src2 == EXE_reg_dst && ID_src2 != 5'd0)) ||
                  (EXE_l_type && MEM_s_type);
```

注意到倒数第二行，这里将所有执行级与译码级产生数据相关的情况都进行了阻塞，但实际上不需要。

（4）修正效果

根据实际情况，如果当前译码级为分支跳转指令且与执行级数据相关，那么进行阻塞，其他情况取消了阻塞。将分支跳转指令阻塞主要是考虑到如果直接从执行级前递可能该路径太长，会不满足时序要求，后序也可以进行尝试。修改后的阻塞信号如下：

```
assign ID_stall = (MEM_l_type && (EXE_src1 == MEM_reg_dst && EXE_src1 != 5'd0 || EXE_src2 == MEM_reg_dst && EXE_src2 != 5'd0)) ||
                  (MEM_l_type && (ID_src1  == MEM_reg_dst && ID_src1  != 5'd0 || ID_src2  == MEM_reg_dst && ID_src2  != 5'd0)) ||
                  (EXE_l_type && (ID_src1  == EXE_reg_dst && ID_src1  != 5'd0 || ID_src2  == EXE_reg_dst && ID_src2  != 5'd0)) ||
                  (EXE_mul_div_type && !EXE_mul_div_validout) ||
                  (EXE_mul_div_type && (ID_inst[31:26] == `MFHI && ID_inst[5:0] == `MFHI_2 || ID_inst[31:26] == `MFLO && ID_inst[5:0] == `MFLO_2)) ||
                  (ID_br_type && (ID_src1 == EXE_reg_dst && ID_src1 != 5'd0 || ID_src2 == EXE_reg_dst && ID_src2 != 5'd0)) ||
                  (EXE_l_type && MEM_s_type);
```

（5）归纳总结

在完成新的设计（如前递）时，要考虑这与之前的设计在哪些地方有重复和冲突，要细致的进行修改，保证新的设计可以完全按照预想实现。

2、错误 2

（1）错误现象

除法器运算结果商的答案正确，但是余数的结果错误。

（2）分析定位过程

根据仿真的报错结果找到了相应的测试数据，具体是 PC=bfc14180 时发生错误。

```
bfc14170:   3c092083    lui t1,0x2083
bfc14174:   35291400    ori t1,t1,0x1400
bfc14178:   0109001a    div zero,t0,t1
bfc1417c:   0000a812    mflo    s5
bfc14180:   0000b010    mfhi    s6
```

（3）错误原因

进行手算除法后发现，我所设计的除法器的余数与正确答案相差了一位，具体来说是少左移了一位。然后回到代码重新分析了整个除法器的计算过程，发现由于第一拍没有进行溢出判断，而是直接取 33 位进行运算，因此整个除法在 32 拍可以完成，但是最后一拍上商后余数没有移位，因此导致结果错误。

（4）修正效果

在最后一拍的处理上加上了将余数左移一位，然后重新测试发现结果正确。

（5）归纳总结

在根据参考文档进行自己的设计时，如果相应修改了一些实现方法，要具体想清楚还有哪些细节也同时需要修改，不能遗漏细节。

3、错误 3

（1）错误现象

assign 赋值信号结果和预想中不一样。

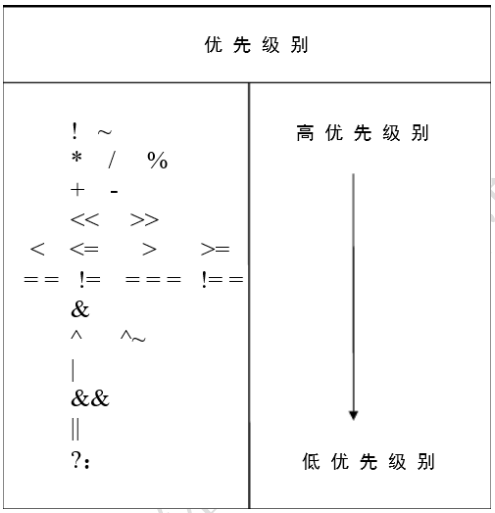
（2）分析定位过程

在发现乘法器结果错误后，我跟踪了每一根线，在进行手算后发现其中有一些和预想中的不一样。

```
(!yy[13] && yy[12]&yy[11]) ? xb_2 :
(yy[13] && yy[12]|yy[11]==0) ?neg_xb_2 :
```

（3）错误原因

后来反复检查，觉得可能是运算符优先级的问题，于是查了运算符的优先级。



发现是==符号的优先级比|的优先级高，导致了错误。

（4）修正效果

于是在 yy[12]|yy[11]的外面加了括号，成功解决了这个错误。

（5）归纳总结

在编写代码的时候不能想当然，不太确定的时候要勤于查资料，此类问题虽然小但是费时。

四、实验总结

（一）组员：袁峥

这周主要干了两件事，第一件事是在仿真时发现了阻塞信号有效过于密集，然后查看了一些阻塞的指令，经过研究发现在本来的设计中本不应该被阻塞，然后找到了原因并进行了修改，经测试发现原来多加了大约十分之一的阻塞，经过改进后性能的提升还是比较可观的。后续可以尝试一些其他的前递，可以更少的阻塞指令，但是这可能也会给时序的满足带来影响，因此暂时还没有进行尝试。

第二件事是将原先分开了有符号除法器和无符号除法器进行了合并，写了一个有无符号除法可以通过的除法

器。这个过程还算顺利，由于之前已经分开编写了两者，因此对整个流程还算清楚，查看了相关的文档发现共同点后稍加修改和调试后，成功了更新了该功能。

后续剩余的事情是，除法器还可以提前开始和结束，如开始时可以先将被除数左移到第一个大于除数的位，这样可以省去前面连续上商 0 的步骤。结束时如果发现被除数已经为 0，可以提前结束。

（二）组员：施璠

这周我承担了编写乘法器的任务，在反复阅读了文档 LEC05_硬件乘除法器实现后，初步对华莱士树+booth 编码的乘法器有了概念。我先画了华莱士树，具体理清了每个运算部件的连线后，编写了代码。接着在 testbench 中验证功能，但在这里碰到了一些困难，调试了很久仍然还有问题，其中碰到过运算符的优先级的错误使用导致运算结果和想象中不一样等问题。目前乘法器仍在调试过程中，争取下周可以将该模块完成。