实验 3 报告 4

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。

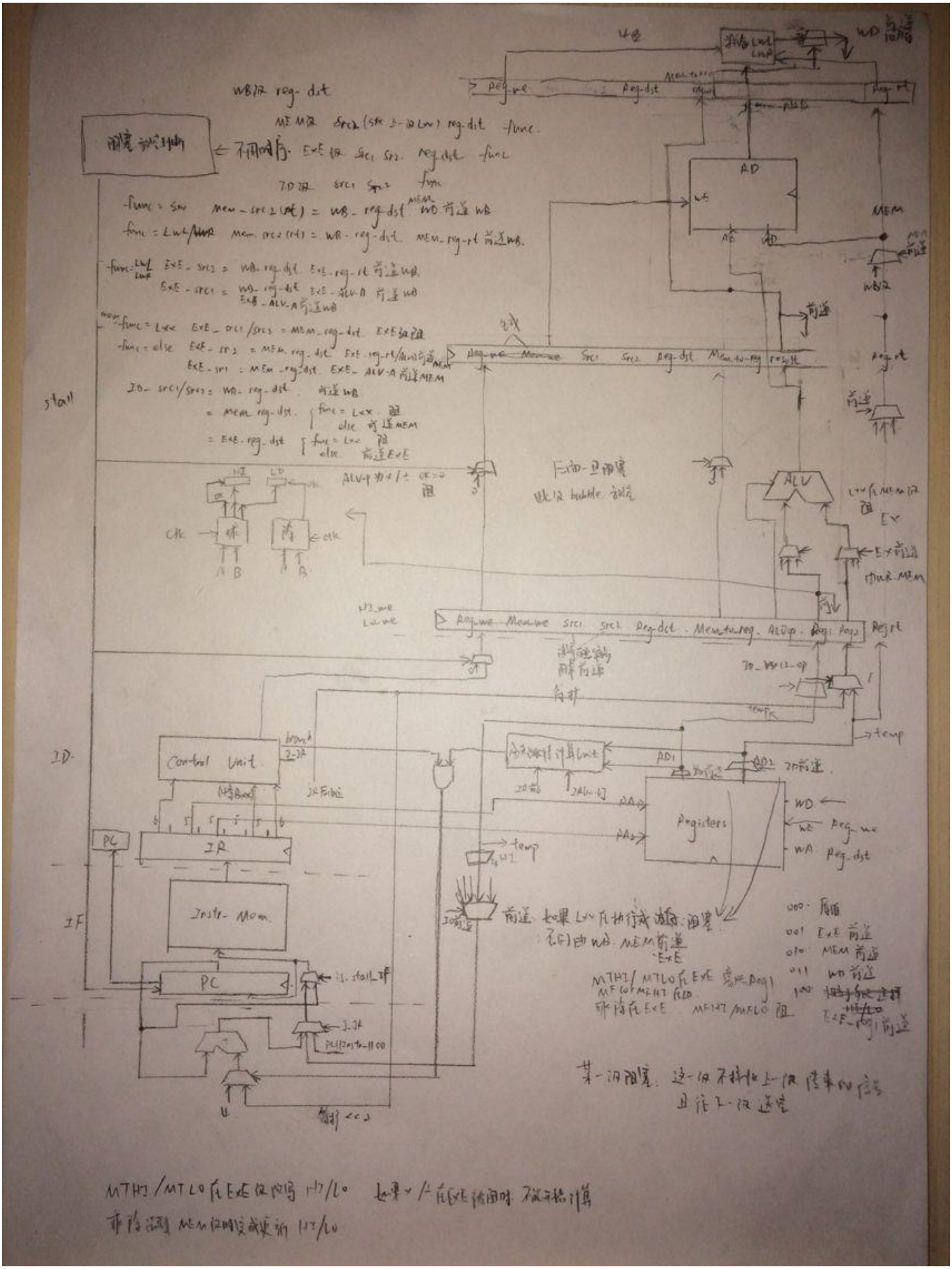
(3) 结构相关即某一级流水停顿了，会阻塞上游的流水级。

(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：


WB段 reg_dst
MEM段 src2 (sw) reg_dst func
EXE段 src1 src2 reg_dst func
ID段 src1 src2 func
func = sw mem_src2(rt) = WB_reg_dst 前递 WB
func = lw mem_src2(rt) = WB_reg_dst MEM_reg_rt 前递 WB
EXE_src1 = WB_reg_dst EXE_reg_rt 前递 WB
EXE_src1 = WB_reg_dst EXE_ALU_B 前递 WB
EXE_ALU_A 前递 WB
func = lw EXE_src1/src2 = MEM_reg_dst EXE阻塞
func = else EXE_src2 = MEM_reg_dst EXE_reg_rt/mem 前递 MEM
EXE_src1 = MEM_reg_dst EXE_ALU_A 前递 MEM
ID_src1/src2 = WB_reg_dst 前递 WB
= mem_reg_dst func = lw 阻 else 前递 MEM
= EXE_reg_dst func = lw 阻 else 前递 EXE
stall
ALU_op 为 x/÷ clk=0 阻
下面一旦阻塞 此处 bubble 就完
Control Unit
IR
PC
Instr_Mem
Registers
ALU
MEM
WB
EX
ID
IF
前递 如果 lw 在执行或 访存 阶段 则是 由 WB MEM 前递 EXE
000 原值
001 EXE 前递
010 MEM 前递
011 WB 前递
MTHI/MTLO 在 EXE 段 reg1
MFHI/MFLO 在 EXE 阻
第一阶阻塞 这一阶不指向上阶 指令的信号 且往下一阶送空
MTHI/MTLO 在 EXE 段写 HI/LO 如果 x/÷ 在 EXE 段用时 不能计算
并阻塞到 MEM 段时设成功的 HI/LO

（一）乘法器模块

（1）基本概述

本周用 booth 编码和华莱士树实现了乘法器具体步骤如下：

1、 为了统一有符号乘法和无符号乘法，根据是否是有符号乘法，把乘数扩展为 65 位，有符号就补符号位，无符号就补 0；

2、 计算[x]补, -[x]补, 2[x]补, -2[x]补；

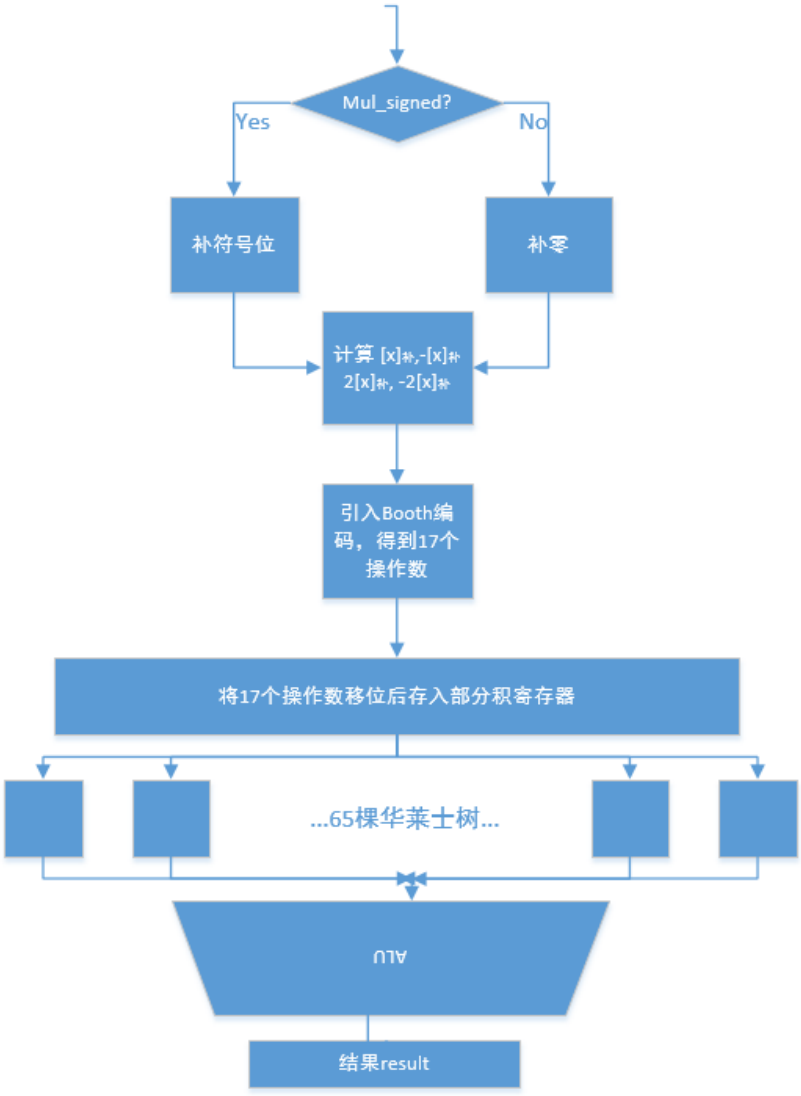
3、 引入 Booth 编码，得到 17 个操作数；

4、 将 17 个操作数移位后存入部分积寄存器；

5、 得到 65 棵输入为 17 位的华莱士树；

6、 通过 ALU 计算得到结果。

（2）流程图



（3）接口定义

名称	方向	位宽	功能描述
mul_clk	IN	1	时钟信号
resetn	IN	1	复位信号，低电平有效
mul_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
x	IN	32	乘数 x
y	IN	32	乘数 y
valid_in_x	IN	1	乘数 x 输入有效信号
valid_in_y	IN	1	乘数 y 输入有效信号
result	OUT	64	乘法运算结果输出
valid_out	OUT	1	乘法运算结果输出有效信号

三、实验过程（60%）

（一）实验流水账

本周主要进行了乘法器的设计和性能的优化（两人大约各自花费 10 小时）

（二）错误记录

1、错误 1

（1）错误现象

在初次尝试仿真中计算得到的结果与应该出现的结果不一样

（2）分析定位过程

考虑到 testbench 给出的是两组随机数的乘法，得到的数和结果都很复杂，不利于发现错误，就从 2*3 出发，观察每一步的波形。

（3）错误原因

对$[x]_{补}$, $-[x]_{补}$, $2[x]_{补}$, $-2[x]_{补}$ 理解错误，输入到乘法器的接口时，x 已经转换为补码形式，而在计算$-[x]_{补}$时需要再计算一次补码。

（4）修正效果

在计算补码中得到了正确的加数，但还未出现正确结果，还存在别的问题。

（5）归纳总结

计算机组成原理是基础，对于不太清楚的要多查。

2、错误 2

（1）错误现象

计算得到的结果和实际的结果一致，但是与 testbench 提供的结果不一致

（2）分析定位过程

考虑 testbench 的结果应该不会出错，所以应该是得到结果的时间的问题，发现应该是在第二拍出结果

（3）错误原因

没有用两拍实现乘法器

（4）修正效果

加入寄存器，使得乘法结果在第二拍给出，从而与 testbench 一致，在初始设定的 2*3 计算中得到了正确的结果。

（5）归纳总结

要仔细阅读任务书中。

3、错误 3

（1）错误现象

有符号乘法结果出错。

（2）分析定位过程

根据波形，把计算结果和实际结果相比较，发现高位丢失。

（3）错误原因

在符号位扩展的时候，只扩展了一位，应该扩展到 65 位

（4）修正效果

有符号乘法结果正确

（5）归纳总结

要正确理解负数在计算机中的存储方式，细致谨慎。

四、实验总结

（一）组员：袁峥

本周完成的主要工作是性能的提升和频率的测试。性能的提升方面，主要是更换了新的乘法器，之后进行测试后发现时序果然比以前短了一些。还有就是删除了一些没有用的信号，这主要是之前在对代码进行修改的时候没有修改干净，留下了一些残余，这样修改之后可能会对布局布线进行一些提升。在进行第四阶段的指令测试时比较顺

利，一遍就完成了测试。接着运行性能测试程序，由于需要尽可能的提高频率，因此此处采用二分法，先测得一个跑不了的频率，然后用二分法不断逼近，测得极限频率，最后发现大约在 130MHz 左右。

（二）组员：施璠

在本阶段中的实验中，我总算改完了上周没改好的乘法器，通过这部分的实验，对补码的定义，booth 编码，以及华莱士树有了更深刻的理解。上周没把乘法器写完的原因之一大概是第一遍写的有点草率，然后 bug 就很多，看着波形就头大，不知从何改起。后来选择从自定义数据开始测起，先测小数据的无符号乘法，再测随机数的无符号乘法，然后测小数据的有符号乘法，最后测所有随机数的乘法。这样分功能调试更有利于逐步跟踪，发现错误。反思 bug 多的原因还是概念不清又不够仔细。

五、实验结果

Dhrystone 运行实际 total_ns	42300
Coremark 运行实际 total_ns	3435750
myCPU 上板最高可运行频率	132MHz
Coremark/MHz	2240/1000
DMIPS/MHz	1043/1000