实验 2-2 报告

学号：2016K8009929011
姓名：段江飞

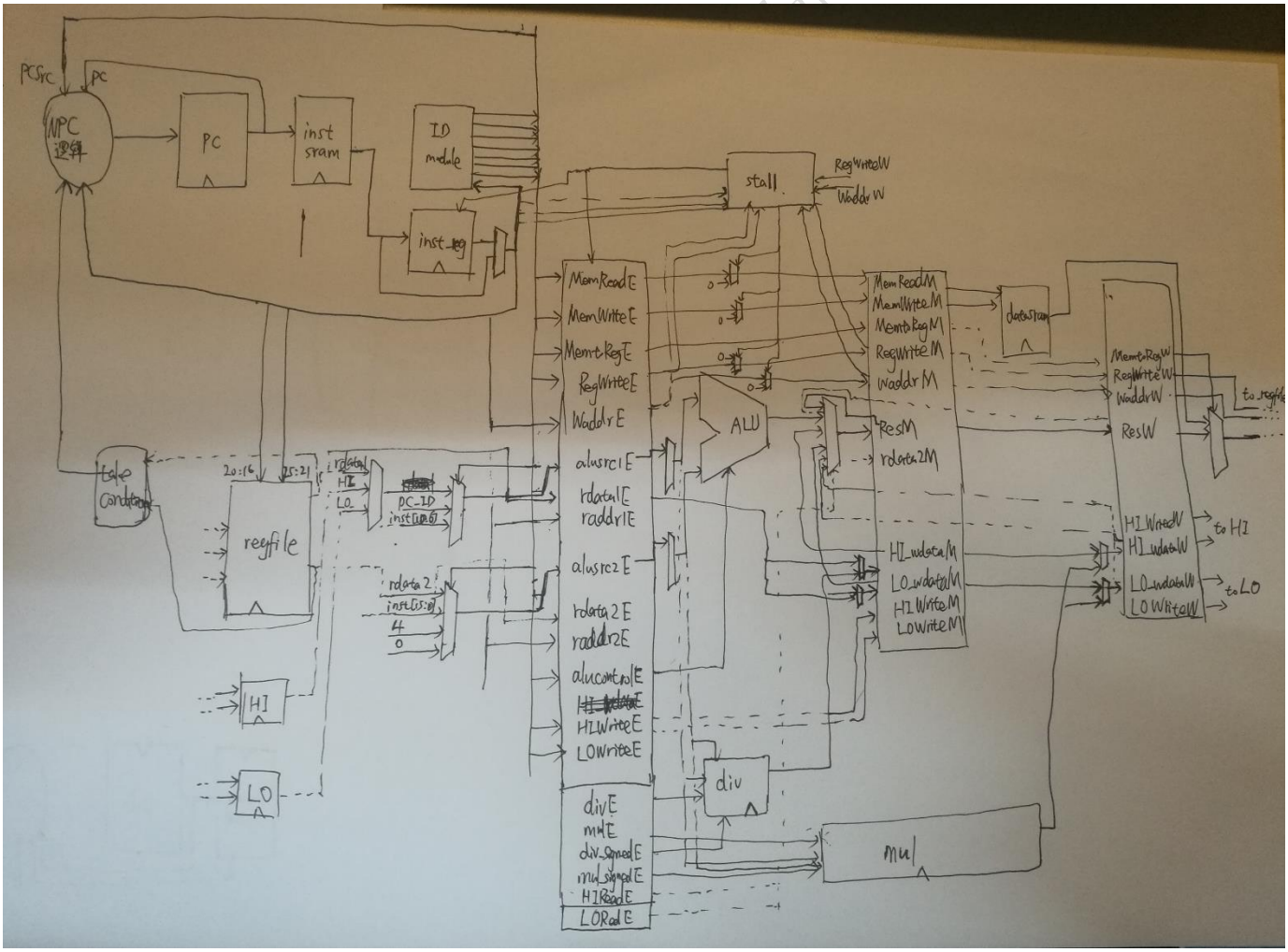
一、实验任务（10%）

本实验要求完善之前的五级流水 CPU，新添加 19 条机器指令，要能够在有数据相关时正常运行，并且要实现乘法和除法运算，乘法利用华莱士树实现乘法器，除法利用迭代算法实现。

设计的 CPU 要通过 42 个功能点的测试，仿真验证在控制台打印 42 个功能点 PASS 和最终的测试结束的 PASS，同时在上板验证时，数码管的值要从 0 累加到 42，高低 8 位要一直相等，LED 灯的变化要和讲义中一致。

二、实验设计（30%）

如下静态五级流水 CPU 结构图



图一 CPU 结构图

（一）数据相关

1、数据旁路

ID 阶段取数据时，正好 WB 要写数据，若出现相关，构造数据旁路，可以解决寄存器堆和 HI/LO 寄存器的一部分相关。ID 阶段取指令，但是再 EX 和 MA 阶段会写寄存器或 HI\LO 寄存器，若不是 ID 和上一条的 EX 是 load 指令相关，则可以建立数据旁路解决。

2、阻塞

ID 阶段和紧邻上一条 load 指令出现数据相关时，需要进行阻塞一拍，然后 load 指令能取出数据，在 EX 的时候利用数据旁路解决。分支跳转指令需要判断取出的值想否相等，若与 load 相关，也需要阻塞解决。

阻塞的设计为利用 ID 阶段译码产生的 raddr 信号和 EX、MA 和 WB 阶段的 waddr 和 RegWrite 信号判断，是否相关和需要阻塞，若需要阻塞，则将相应的 readygo 设为 0，并且给相应阶段的寄存器添加空泡，以免造成持续阻塞。

3、HI 和 LO 的处理

HI 和 LO 是新添加的寄存器，处理是类似寄存器堆，有 Read 和 Write 使能信号，rdata 和 wdata 信号。

（二）乘法器

1、基本思想

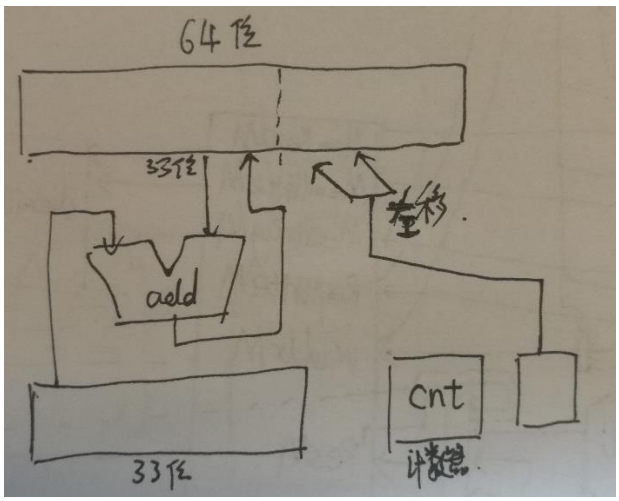
乘法计算利用 Booth 两位乘，考虑到有符号无符号数能同样计算，用 33 位数相乘，产生 17 个部分积，17 个部分积的产生可利用 generate 生成 17 个 Booth 编码器模块，部分积由于-[x]补的原因会出现取反加 1，把加的 1 拿出来共有 17 个，但是通过分析可以发现，最高的三位通过扩展，是不会出现取反加 1 的，因此还是 16 个加 1 的信号，可以利用第一个华莱士树的 14 个 cin 和 64 位加法器的 cin 和华莱士树的进位 C 移位后的低位，可以满足计算。

2、流水线切分

流水线切分是从 64 位全加器的前面切分。

（三）除法器

1、结构



图二 除法器

除法器的设计颇为巧妙，但是计算所需要的时钟周期过长。除法器的被除数放在一个 64 位的寄存器里，除数放在 33 位寄存器里，开始除法时，对计数器赋 33，然后降到 0 结束除法。除法进行过程中，取被除数高 33 位与除数相减，若结果大于 0，在末位上 1，将高 33 位替换为减法结果，整体左移移位，否则，末位上 0，整体左移一位。

2、阻塞

除法计算要 34 拍，因此在 EX 阶段会发生阻塞，对 EX_readygo 置为 0，并向后面阶段塞空泡，直到除法完成，停止阻塞。

（四）实现功能

静态五级流水能正常运行，通过 42 个指令功能点测试，解决数据相关，实现华莱士树乘法器和迭代除法器。

（五）验证

1、仿真验证

在 vivado 上仿真，首先编译提供的测试代码，然后给龙芯 CPU 重新定制 instram，跑测试文件生成比对文件。之后，在自己的 CPU 上进行仿真验证，知道打印 42 个功能点 PASS，通过验证。

2、上板验证

将实验箱连接到 vivado，编程后看实验箱的数码管和 LED 灯的变化，数码管的高低 8 位一直同时累加到 0x2a，LED 灯的颜色变化和讲义一致，则通过上板验证。

三、实验过程（60%）

（一）实验流水账

1、9 月 29 日晚上 7 点到 12 点

理解阅读华莱士树实现方法，并实现华莱士树。

2、9 月 30 日下午 1 点到 4 点

找华莱士树的 bug。

3、10 月 1 日晚上 6 点到次日凌晨 1 点 40

写迭代除法器，并找 bug，然后检查数据旁路，修正了 ID 与前一条 load 指令相关引起阻塞。

4、10 月 2 号晚上 7 点到 9 点

遇到分支跳转和 lw 指令的数据数据相关，添加阻塞失败。

5、10 月 5 号晚上 7 点到凌晨 3 点 40

继续 load to use 阻塞问题，遇到组合环，并找出造成组合环的原因。

6、10 月 6 号上午 10 点到晚上 12 点

解决组合环问题之后，开始真正进行测试和找 bug，并通过 2-2，最后完善了实验报告。

（二）错误记录

1、错误 1

（1）错误现象

测试华莱士树，发现乘法计算结果不对。

（2）分析定位过程

由于 random 的乘数与被乘数不易观察，我就将 x 和 y 分别设为 9 和 8，然后利用手算 booth 两位乘，和波形上的部分积进行比较。

（3）错误原因

部分积计算过程中忘记移位，导致错误，第一次修改方法是直接将计算出的部分积移位（图二），发现结果还是错误，后来又仔细计算比对，发现在加-x 和-2x 的补码的时候，直接移位最后补的是 0，但是实际应该是 1，于是将代码修改为传入 booth 编码器时就是被乘数移位后的结果（图三）。

```
generate
    genvar i;
    for(i = 0; i < 17; i = i + 1)
    begin : encoder
        BoothDe u_BoothDe(
            .y2 (mul_y[2*(i+1)] ),
            .y1 (mul_y[2*i+1]   ),
            .y0 (mul_y[2*i]     ),
            .x  (mul_x          ),
            .c  (c[i]           ),
            .p  (ps[i]          )
        );
        assign p[i] = ps[i] << 2*i;
    end
endgenerate
```

图三

```
generate
    genvar i;
    for(i = 0; i < 17; i = i + 1)
    begin : encoder
        BoothDe u_BoothDe(
            .y2 (mul_y[2*(i+1)] ),
            .y1 (mul_y[2*i+1]   ),
            .y0 (mul_y[2*i]     ),
            .x  (mul_x << 2*i   ),
            .c  (c[i]           ),
            .p  (p[i]          )
        );
    end
endgenerate
```

图四

（4）修正效果

成功通过了乘法的测试。

2、错误 2

（1）错误现象

除法器测试，结果不正确。

（2）分析定位过程

根据除法的结果和比对结果，发现商是对的，但余数不对，余数恰好是正确结果的二倍。

（3）错误原因

多进行了一轮迭代，计数器设为 32 来记录做除法的次数，这样计数器从 32 减到 0，除法器共迭代了 33 次，多进行了一次，导致多进行一次左移，余数扩大了一倍，应该把计数器修改为 31。

（4）修正效果

结果正确。

3、错误 3

（1）错误现象

在验证除法器模块时，修改了上述的错误 2 之后，又出现了一个结果错误的例子，余数和商都不正确。

（2）分析定位过程

根据计算的结果，商结果为 1，我的商为 0，到波形图查看，发现商 1 是最后一次应该上的商，我的试商的结果是正确的，但是除法提前结束了，没有上商，出现了错误，之前的都通过测试是由于商的最后一位恰好为 0 的巧合。

（3）错误原因

仔细分析我的结构，我把商和余数放在了一起，是一个 65 位的寄存器，然后每次试商的时候是拿高 33 位减除数，上的商放在了第 0 位，一轮上商结束之后，将 65 位的数左移，继续这个过程，这样做 32 次，就会使得余数实际上是扩大了两倍，造成了错误 2 中的 bug，我将迭代次数减少一次之后，能结果上正确几个测试，但实际上迭代的次数少了一次。修正之后，我把 65 位的寄存器改为了 64 位，其他的计算过程没有变化，这样迭代进行 32 次，是正确的结果。

（4）修正效果

通过了除法模块的验证。

4、错误 4

（1）错误现象

波形停止。

（2）分析定位过程

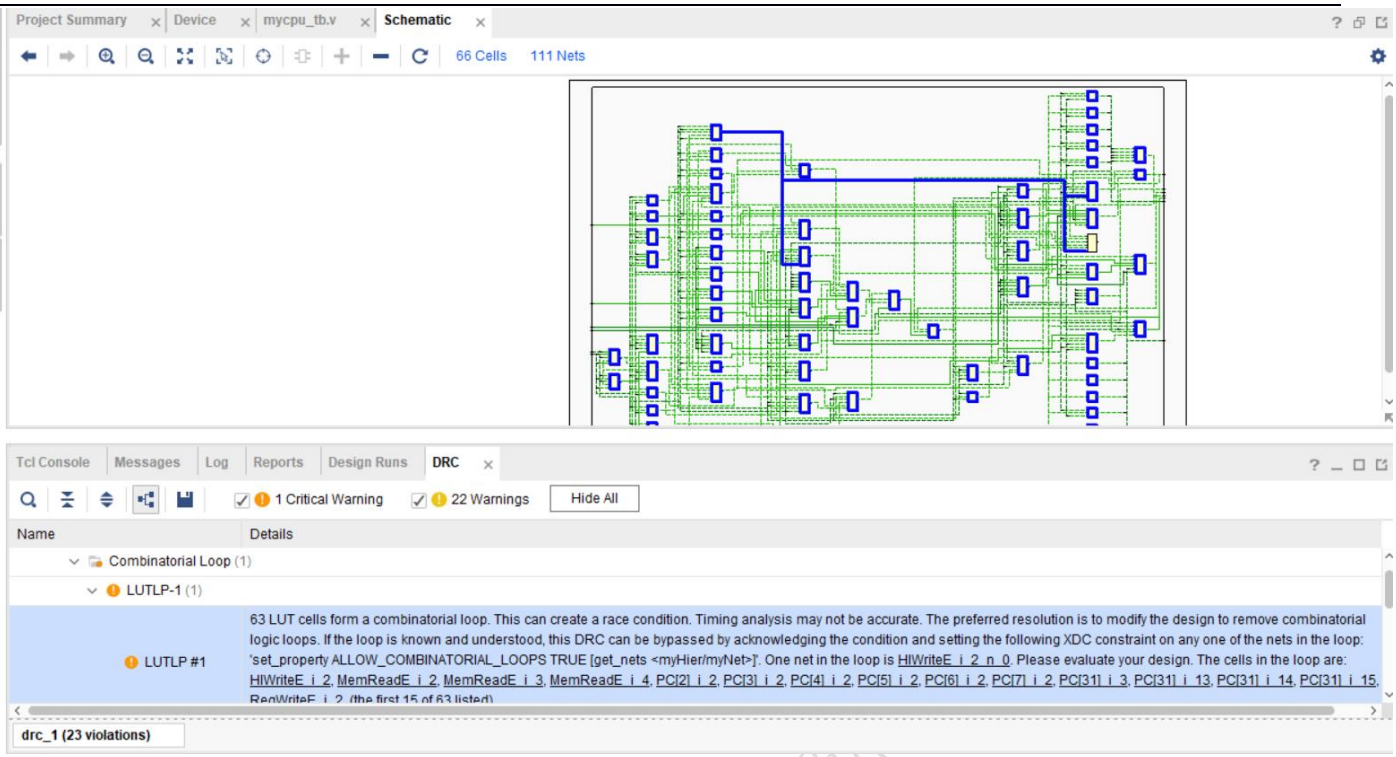
综合，进行时序分析，发现出现了组合环。出现错误是因为我添加了指令寄存器，为了避免阻塞时 ID 的指令发生变化，添加了指令寄存器，并且根据阻塞信号对指令是来自刚读出的指令还是指令寄存器进行选择，但是，我发现，如果不添加就没有组合环，添加之后会有组合环，通过查阅资料，发现组合环（见图五）。

（3）错误原因

组合环出现在对指令进行选择的逻辑上，MUX 的选择信号是有指令组合逻辑产生的，用指令产生的信号去选择指令，这个组合逻辑形成了环。

（4）修正效果

因为组合环的原因，MUX 的选择信号不能来自后来的阻塞信号，通过和同学交流，选择对 IF_allowin 存一拍，发生阻塞时，IF_allowin 会拉低，通过这个信号能确定是否发生阻塞，根据此消除了组合环。



图五

5、错误 5

（1）错误现象

阻塞不正确，load to use 没法通过测试。

（2）分析定位过程

根据波形图找错，发现是阻塞之后，前面的指令继续执行，阻塞之后的值会保持，阻塞判断条件会一直成立。

（3）错误原因

没有送空拍，而且阻塞信号赋值也有问题。之前总是给 allowin 赋为 0 来阻塞，后来发现 allowin 的阻塞时不正确的，应该给 readygo 赋为 0，因为 allowin 也影响了后面的信号，导致了逻辑的混乱，于是改为 readygo 进行阻塞。后来又根据该级对下一级的 valid 信号添加空拍。

（4）修正效果

Load to use 的数据相关通过测试，阻塞也正确了。

四、实验总结

本次实验真的好难啊！尤其是阻塞，写出了一个组合环，完全不知道问题在哪，进度一度停滞了好久，花了好久的时间才找到问题，然后完善旁路，才通过了测试。