实验二报告（阶段二）

2016K8009929049
万之凡

一、实验任务

增加新的 19 条机器指令丰富原有 CPU 功能；用前递处理实现解决数据相关的问题；实现乘法器和除法器。

在设计阶段需要将 lab2_1 的 CPU 考虑如何添加新信号并且添加乘法器和除法器，步骤分别为：

（1）在原有 CPU 基础上考虑数据相关；

（2）实现乘法器和除法器并且关联到 CPU 上；

（3）添加新的 19 条机器指令（详细指令内容见研讨课讲义）。

实现阶段需要实现乘法器、除法器、新指令，步骤分别为：

（1）采用 booth 算法和华莱士树实现乘法器；

（2）用迭代算法实现除法器；

（3）根据给出的乘法器和除法器实现新的指令。

验证过程中根据 SoC_Lite 实验环境的工作原理，分别检查 Vivado 实现的波形、inst_ram 的仿真编译结果，在得到 pass 后上板验证。

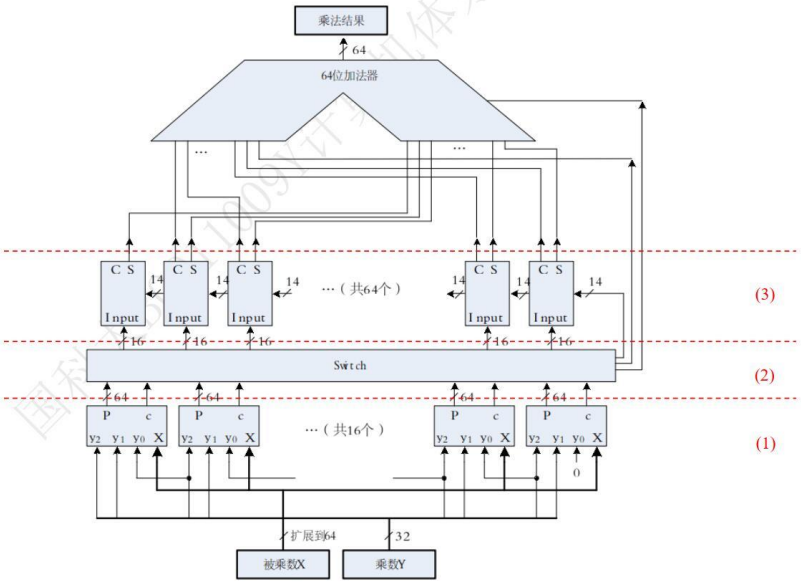
（1）更换 soft 文件夹内原有的仿真验证文件与 golden_trace 文件，Vivado 仿真运行 mycpu；

（2）在波形正确后查看仿真结果，pass 后生成 bit 文件上板实验；

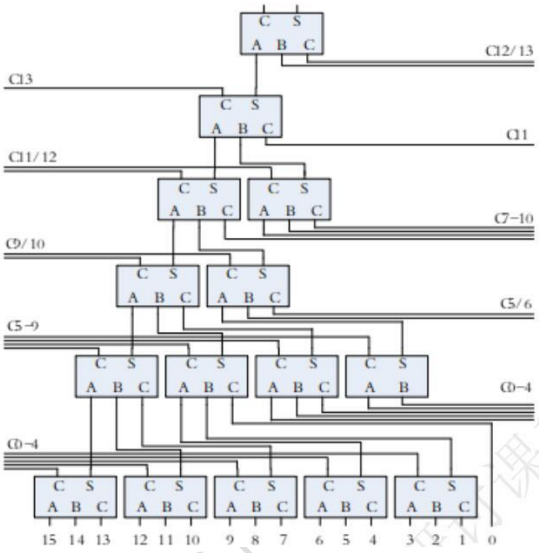
（3）若上板显示结果正确，mycpu 验证成功。

二、实验设计

乘法器的实现原理基于 booth 乘法，使用 2 位 booth 乘法实现能够显著减少加法的次数，使用 15 次加法进行 32 位的乘法，并且在这个过程中使用保留进位加法器节省进位延时，最大化优化乘法器效果。华莱士树在使用过程中首先需要注意设计的连线问题，传送至更高层的数据在想清楚之后还是挺简单的，但最初还是有些难度彻底划清楚。最终实现的是能够计算无符号数的 33 位乘法器。大体的思路与实现 32 位补码乘法器相同，因此给出 32 位乘法器的结构图如下：



图一 32 位补码乘法器结构



图二 16 个数相加的 1 位华莱士树

除法器通过迭代除法进行实现，额外需要注意的是对于余数值的调整，无论采用循环加法补码除法器还是使用加减交替补码除法器，都需要对生成的余数进行调整。本次实验实现的是绝对值迭代除法器，一共需要分三步完成需要的运算：首先根据被除数和除数确定商和余数的符号，计算被除数和除数的绝对值，这里使用加法器进行计算（无符号数采用补码进行计算即可）；然后依次迭代运算得到商和余数的绝对值，将结果不断在高位补 0，连续计算 32 次求得商值和未调整余数值；最后调整商和余数，之前计算都是使用的绝对值进行计算，这一步根据符号位判断正负并进行转换，生成最终结果并输出。

三、实验过程

（一）实验流水账

2018/09/27，阅读相关资料并开始编写试验报告设计部分；

2018/09/29，补全 lab1 中没有彻底完成的部分；

2018/09/31，重新编写 CPU；

2018/10/02，给编写的 CPU 添加流水结构；

2018/10/03，根据研讨课上的实例代码修改 CPU；

2018/10/05，lab2_1 上板失败，继续 debug；

2018/10/06，设计乘法器和除法器；

2018/10/07，继续编写修改乘法器和除法器；

2018/10/08，添加新指令并上板测试；

2018/10/09，编写实验报告。

（二）错误记录

1、乘法器（连线）逻辑错误

（1）错误现象

在进行设计编写的时候计算错误，用例子测试结果并不能完全正确。

（2）分析定位过程

画出华莱士树的草稿，连线过程较复杂，逐步比较 booth 乘法结果和华莱士树种测得的结果。

（3）错误原因

连线有误，起初并未注意讲义中的接线方式，导致 C0 在计算时错位。

（4）修正效果

设计出正确的设计图，计算结果和手算结果相同。

（5）归纳总结

认真阅读讲义，不断回滚原有知识进行复习。

2、除法器迭代逻辑错误

（1）错误现象

除法测试结果完全不正确。

（2）分析定位过程

除法器的原理无非是判断移位后剩余下来的除数被除数的大小，确定商，减法得出余数，更新商和被除数。因此将一个完整的过程连续实现 32 次直至全部计算完成为止。

（3）错误原因

对于除法的实现并不熟悉。

（4）修正效果

模拟计算结果正确，写出正确的代码，testbench 测试通过。

（5）归纳总结

从原理出发进行 CPU 的设计。

3、循环变量的定义和使用

（1）错误现象

起初想在 Verilog 中尝试使用 for 循环（因为乘法除法都有类似需要循环解决的计算过程），但是由于无法像 c 语言一样定义 int 类型的循环变量，一度使用手动复制粘贴每一次循环，以此实现多次迭代。

（2）分析定位过程

迭代过程中有路径过长综合失败的情况，因此需要简化循环过程，尽可能使用 for 或类似循环解决问题。

（3）错误原因

Verilog 语法不熟悉。

（4）修正效果

使用 generate 和 genvar 语句，节省空间，能够成功通过乘法的 testbench。在此记录 generate 的使用方式：（代码中实现的 generate 语句如右图）

```
generate
    genvar i;
    for (i = 1; i < 17; i = i + 1)
    begin:BOOTH
        booth booth_i(
            .x    ( x                  ),
            .y    ( y[i*2+1:i*2-1]     ),
            .p    ( booth_sum[66*i+:66] ),
            .c    ( c[i]               )
        );
        assign part_sum[66*i+:66] = {booth_sum[66*i+:66-2*i],{i*2{c[i]}}};
    end
endgenerate
```

图三 multiplier 中的 generate 语句块

genvar 循环变量名；
generate
  // generate 循环语句
  // generate 条件语句
  // generate 分支语句
  // 嵌套的 generate 语句
endgenerate。

（5）归纳总结

从原理出发进行 CPU 的设计。

*以下是两个在补交 lab2_1 的过程中发现并解决的问题。

4、五级流水线未解决的设计问题补全

（1）错误现象

Lab2_1 中未能解决的问题，五级流水线设计过于粗糙。

（2）分析定位过程

状态机的思维还未消除，没有彻底理解流水思想。

（3）错误原因

将流水按照 clk 信号和 resetn 信号强行推进，每一拍固定长度，导致 bubble 过多，cpu 内部逻辑混乱。

（4）修正效果

使用握手信号，将不同模块分为不同的 stage，添加 valid、ready_to_go、allowin 信号，实现任意长度的延迟，有效提升 cpu 性能，同样简化代码风格，结构更加清晰。

（5）归纳总结

这也是在讲义最初提到过的想法，常读常新。

5、vivado 软件操作问题

（1）错误现象

在进行仿真模拟的时候生成的 PC 总与 ref_PC 相差 3 位，起始值为 38c，但我的生成值为 390。

（2）分析定位过程

原本认为是流水过程没有做到，PC 没有一直传到写回级导致在写回级写了错误的数据。但是根据波形和 test.S 中的汇编代码检查发现并没有 PC 错误，后发觉是 goldentrace 文件没有更新。

（3）错误原因

前几次实验中给出的环境已经进行了交叉编译和 trace_ref.txt 文件的生成导致这次惯性思维没有提前准备相关的比对文件。

（4）修正效果

按照讲义内容，运行 gettrace 生成比对文件、并且在虚拟机环境中进行交叉编译，准备好所有文件之后进行仿真，成功 PASS。

（5）归纳总结

值得一提的是在交叉编译的过程中也有 make 报错的可能，但是在删除 inst 文件夹后能够成功 make，一个可能的解释是 makefile 文件在反汇编的时候和原有的 inst 文件夹中的内容冲突，删除后可以正常 make。

四、实验总结

最终成功实现的只有乘法器和除法器。新的指令在添加时仍有很多问题，数据通路的问题也没有彻底解决。