实验 2-3 报告

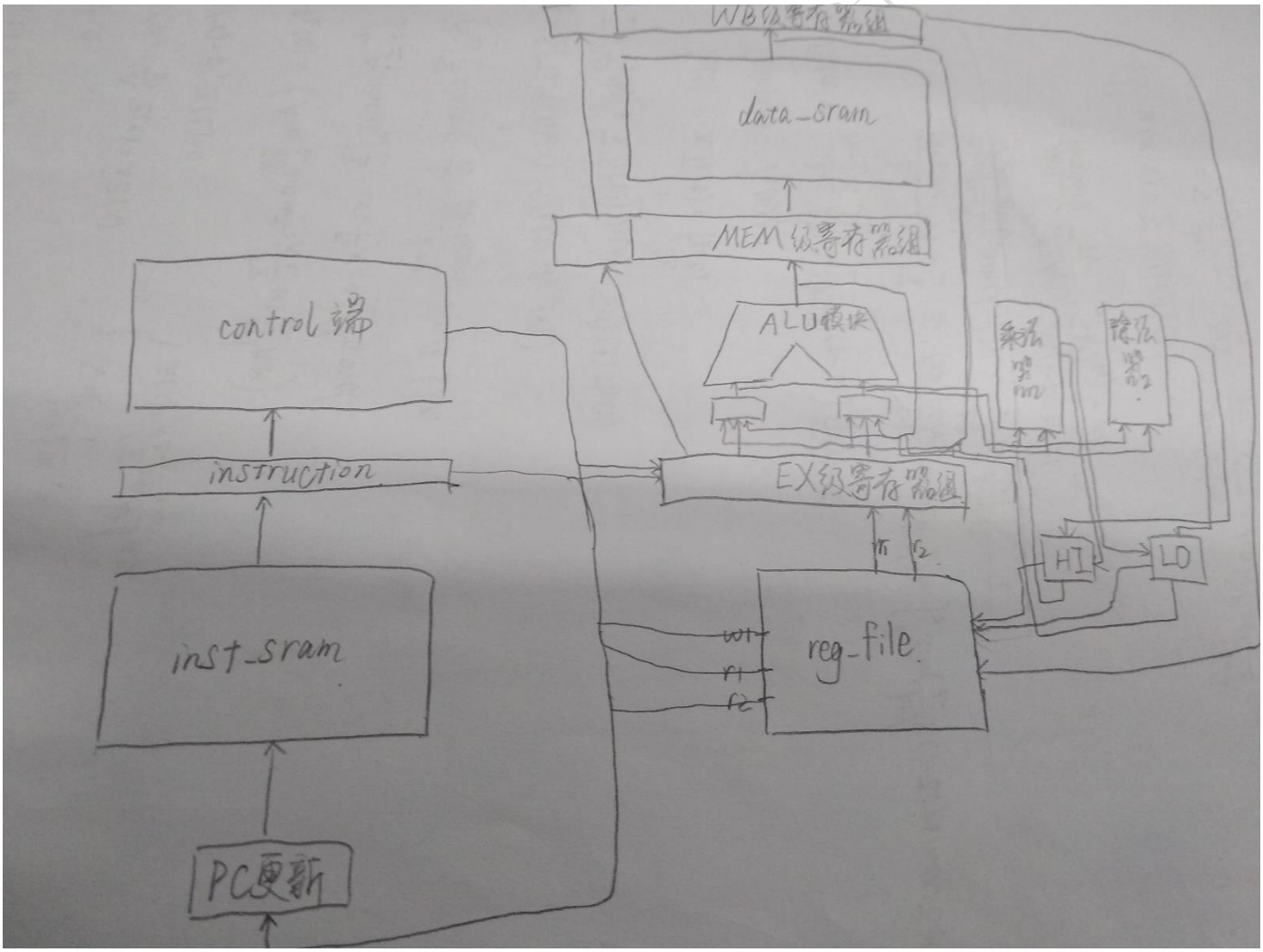
学号：2016K8009929009
姓名：秦宏

一、 实验任务（10%）

第三阶段实验任务是在实验 2-2 的 CPU 基础上添加 18 条指令 J、BGEZ、BGTZ、BLEZ、BLTZ、BLTZAL、BGEZAL、JALR、LB、LBU、LH、LHU、LWL、LWR、SB、SH、SWL、SWR，根据之前的数据相关和阻塞实现完成添加指令，完成上板操作，并且测试程序 coremark 和 dhrystone 通过。通过两个测试程序的 log 计算自实现 CPU 的性能计算。

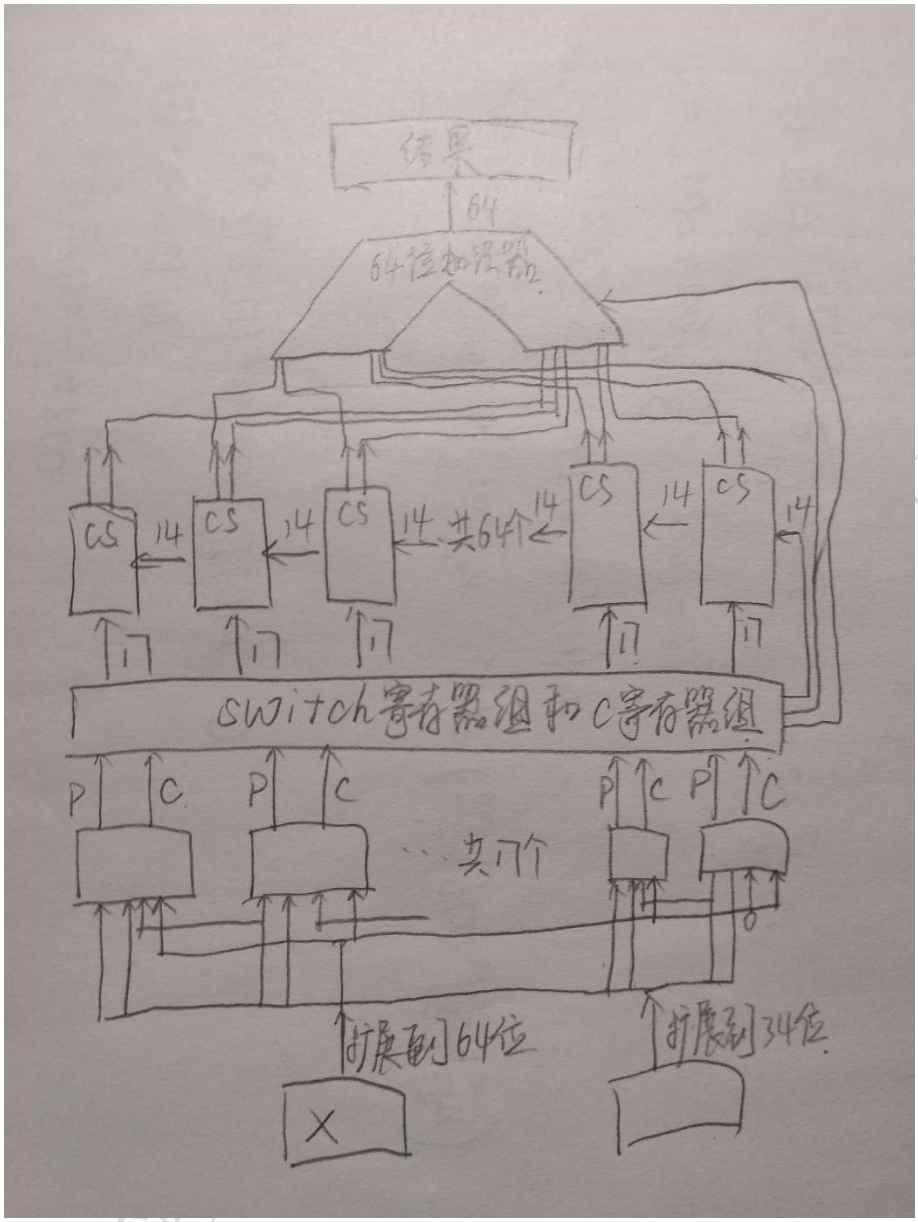
二、实验设计（30%）

（一）实验 2-3 的 CPU 整体结构基于实验 2-2，与实验 2-2 的流水线流程不同的是，实验 2-3 的指令中包含了字节读写操作和不对齐读写操作，因此需要在 data_sram 的端口判断具体访存指令的要求，整体结构与实验 2-2 一致，结构图如下，

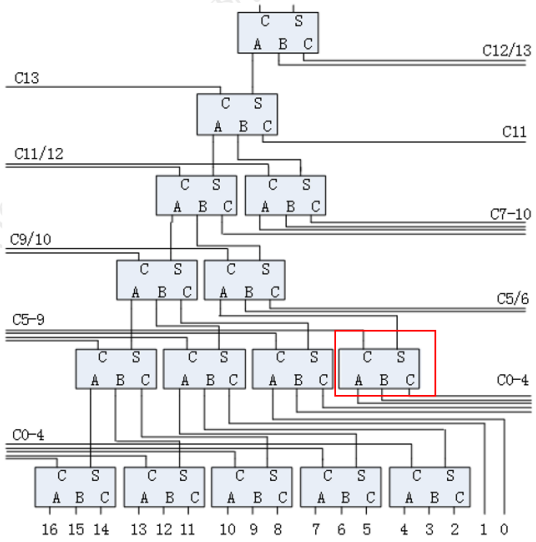


上图中 ALU 接口添加了一个三选一端口用来解决数据相关的问题，同时乘法器和除法器的输入也连接到三选一的端口后直接解决数据相关问题，HI 和 LO 作为单独的寄存器保存乘除法的结果。在测试程序中产生了关于 HI 和 LO 寄存器的数据相关问题，因此也建立了 HI 和 LO 寄存器连接到 ALU 端口的数据相关。

（二）1.实验中的乘法器采用 booth 算法将乘数转化为 17 个部分积，通过 switch 部分转化为 64 个华莱士树求和部分，生成部分积过程中产生的进位信号分别传递到第一个华莱士树的输入信号和最后加法器的补位信号和进位信号，结构图如下所示，



2.实验中的华莱士树为 1 位的 17 个数相加，因此华莱士树的结构如右图



对应的乘法器的端口如下

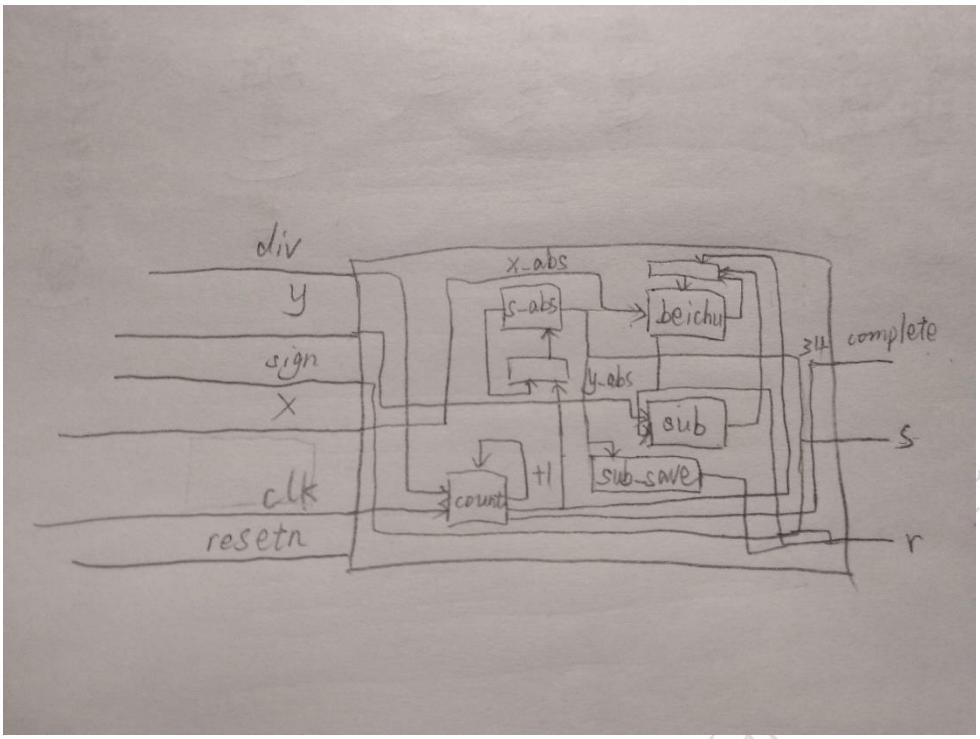
信号	位宽	方向	功能
mul_clk	1	input	乘法器模块时钟信号
resetn	1	input	复位信号，低电平有效
mul_signed	1	input	控制有符号乘法和无符号乘法
x	[31:0]	input	被乘数
y	[31:0]	input	乘数
result	[63:0]	output	乘法结果，高 32 写入 HI，低 32 位写入 LO

乘法器中对华莱士树进行了模块调用，即 64 次华莱士树模块调用，模块名 wtree 端口如下

信号	位宽	方向	功能
num_bit	17	input	输入 switch 转化后的 17 个 1bit 信号
c_in	14	input	输入来自前一华莱士树或者部分积的进位信号
c	1	output	华莱士树输出的 c 信号
s	1	output	华莱士树输出的 s 信号
c_out	14	output	向下一个华莱士树的输入 c_in 传递的进位信号

为了使乘法器的运行实现到两拍上，我在 switch 部分采用了寄存器 temp_0 至 temp_63 对部分积转化为的 64 个华莱士树输入信号进行了保存，到下一拍进行后续操作。

（三）实验中的除法器模块设计了 34 个时钟周期的运行，运行流程图如下，



每次迭代之后被除数都会左移一位，所以直接取被除数最高的 33 位进行商上 0 或上 1 的判断，每个时钟周期更新一位商的绝对值信号。

三、实验过程（60%）

（一）实验流水账

1.10 月 13 日 17 点至 24 点，在实验 2-2 的 CPU 中添加了实验 2-3 要求的 18 条指令，解决了仿真之前的一些语法问题。

2.10 月 14 日 9 点至 22 点，对实验 2-3 的 CPU 进行了行为仿真测试，并在解决了数据相关等问题后仿真测试和上板均通过。

3.10 月 15 日 8 点 12 点，对实验中的 coremark 和 hdrystone 测试程序进行测试，解决了 HI 和 LO 寄存器的数据相关 后问题解决，上板和行为仿真均在 50MHz 下通过。

4.10 月 15 日 15 点 17 点，测试 CPU 的最大频率，定位最大频率为 65MHz,并且进行了新的频率下的 coremark 和 hdrystone 的程序测试。

（二）错误记录

1、错误 1

（1）现象：LB 和 LH 指令报错

（2）分析定位过程

从 trace 报错的指令找起，发现是 SB 和 SH 指令有问题，写入时就出现了问题。

（3）错误原因

CPU 中设置 SB 和 SH 指令时没有考虑存储的数据形式与存储地址的最后两位有关，代码设计不到位。

（4）修正效果

对于 SB 指令，如果地址后两位是 00，则写入的数据是要写入数据(32 位)的 0~7 位，并且前 24 位补 0，地址后

两位是 01，写入的数据是该数据的 0~7 位，但是这 8 位数位于要写入数据的 8~15 位，前 16 位和后 8 位补 0，其他情况类似。

对于 SH 指令则是根据地址的倒数第二位判断，若为 1，则写入数据的前 16 位是原数据的 0~15 位，后 16 位补 0.

2、错误 2

（1）现象：测试程序运行时指令报错

（2）定位过程

加法指令前一条指令是 mflo 指令，在 2-2 的代码中并没有设计数据相关的处理。

（3）错误原因

Mflo 等指令的返回结果同样要写入寄存器中，但是没有添加数据相关处理导致数据出错。

（4）修正效果

将 mflo 指令要输入寄存器的值也添加到 ALU 的三选一端口就可以实现 mflo 指令有关的数据相关的解决方法。

（5）总结

数据相关问题是由相邻或者只隔一条指令的情况引起的，由于指令条数较多，不能够考虑全面，因此只能够遇见一次处理一下有关的数据相关。

3、错误 3

（1）现象：jr ra 汇编指令跳转出现错误

（2）定位过程：

按照 jr ra 的指令，ra 寄存器的值追溯发现有处指令在执行完 jal 的延迟槽指令后就执行了 jr ra 指令，因此会产生 ra 寄存器的数据相关。

（3）错误原因

Ra 寄存器没有写回时便执行 jr ra 指令返回了错误地址。

四、实验总结（可选）

实验 2-3 的任务难度相对不大，添加指令可以参考上学期的组成原理的内容就可以很简单的实现，debug 阶段也是主要解决添加新指令后可能出现的数据相关问题。整体来讲实验 2-3 的内容较为轻松，如果不考虑代码优化的话。

测试程序的内容主要是跑完 coremark 和 hrdystone 两个程序，根据最后生成的 log 判断测试是否通过，并通过频率和运行时间来计算 CPU 的性能。从提高 CPU 的频率来看，我的 CPU 的频率上限应该是 65MHz，首先是 50MHz 情况下，根据 coremark 的 log 看到 50MHz 下我的 CPU 的一次 coremark 的运行时间为 7473540ns，推算出每秒可以运行 133.81 次,除以频率 50MHz 后得到 coremark 评分为 coremark/MHz = 2.676coremark/MHz。再根据 hdrystone 的 log 来看，hdrystone 的 10 次运行时间为 98520ns，则 hdrystone 每秒可以运行 101502 次程序，计算 hdrystone 的得分为 1.155DMIPS/MHz。

然后是我的 65MHz 时的测试程序得分，根据 hdrystone 的 log 来看，hdrystone 的 10 次运行时间为 75790ns，推算出每秒运行 131943 次，计算 hdrystone 的得分为 1.754DMIPS/MHz。根据 coremark 的 log 来看，一次运行时间为 5748880ns，每秒运行 173.947 次，除以频率得到 coremark 评分为 coremark/MHz = 2.676coremark/MHz。比较看来 coremark 测试更加准确稳定。