

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机体系结构 |
| 姓 名： |  |
| 学 院： | 计算机科学与技术学院 |
| 系： | 计算机科学与技术 |
| 专 业： | 计算机科学与技术 |
| 学 号： |  |
| 指导教师： |  |

2023年 12 月 6 日

**浙江大学实验报告**

课程名称： 计算机体系结构 实验类型： 综合

实验项目名称： 基于计分板实现的动态流水线

学生姓名： 专业： 计算机科学与技术 学号：

同组学生姓名： 指导老师：

实验地点： 曹西301 实验日期： 2023 年 12 月 6 日

1. 实验目的和要求

实验目的：

1. Understand the principle of piplines that support multicycle operations.

2. Understand the principle of Dynamic Scheduling With a Scoreboard.

3. Master the design methods of piplines that support multicycle operations.

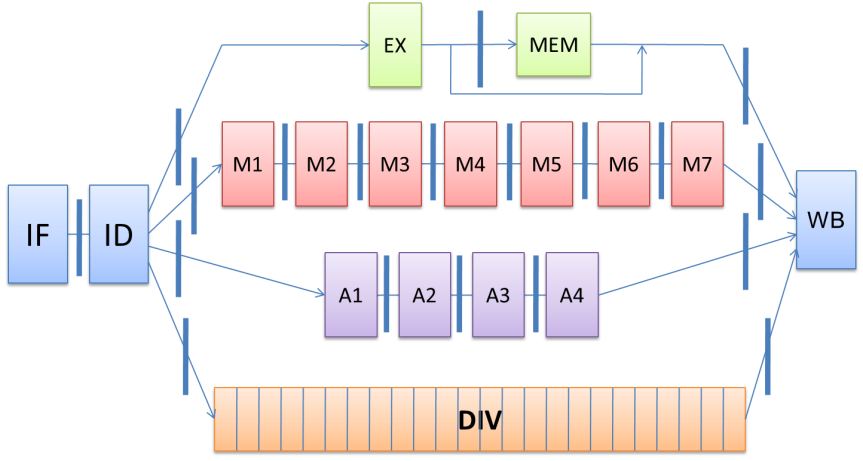
4. Master the design methods of Dynamically Scheduled Pipelines using Scoreboarding.

5. Master verification methods of Dynamically Scheduled Pipelines using Scoreboarding.

实验任务：

1. Redesign the pipelines with IF/IS/RO/FU/WB stages and supporting multicycle operations.
2. Design of a scoreboard and integrate it to CPU.
3. Verify the Pipelined CPU with program and observe the execution of program.
4. 实验内容和原理

计分板的设计思想图如下，首先，我们引入多个不同的functional unit用于区分alu，mlu，div，jump，mem不同功能操作；使要求不同功能计算的指令可以在一定程度上同时运行。

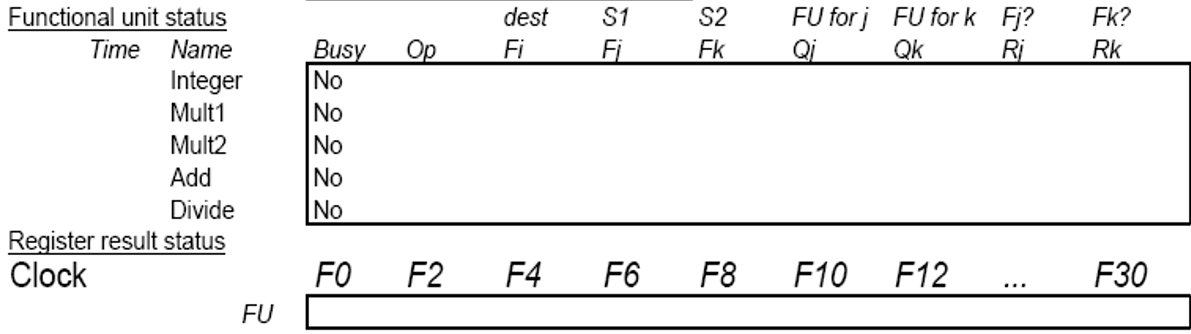


接着，我们引入functional unit status，register result status，instruction status三个状态表，用于记录流水线运行时各block中指令，各寄存器以及各功能单元的运行状态。

其中，instruction status用于记录当前在流水线中运行的指令以及它们正在执行的阶段；此表多作为辅助记录，因而在本实验中并不多加关注。

Functional unit status用于记录每个功能单元的使用状态，执行的具体指令类型以及目的寄存器源寄存器等信息。其中Rj，Rk分别用于记录第一/第二个源数据是否准备就绪，若数据已准备就绪，则该指令可以进入EX阶段开始计算；若没有准备就绪，则可以查看Qj，Qk来索引数据的计算获得相关来源。

最后是register result status，其用于记录当前流水线执行时，各寄存器将要被哪个functional unit的计算结果修改，这有利于判断write after write的hazard情况。

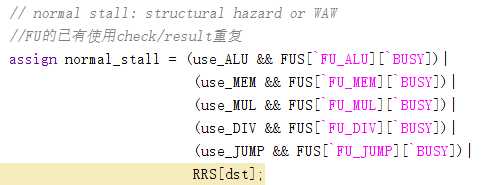


1. 实验过程和数据记录

在本实验中，我们可以忽略instruction status的设计，而只关注functional unit status和register result status的表现。

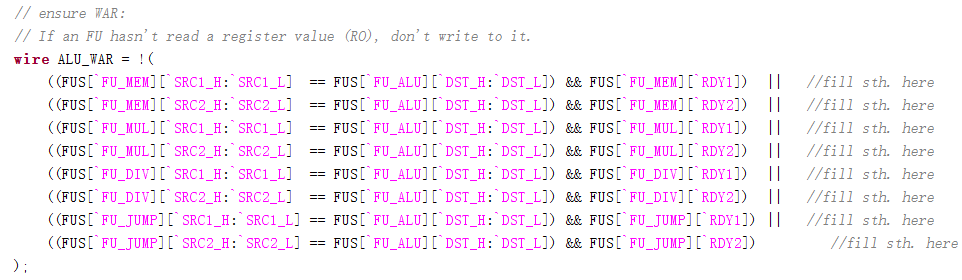
在设计的过程中，我们需要特别考虑WAW，WAR以及structural hazard产生的情况判断并及时的暂停流水线。在此，我们主要实现的模块为CtrlUnit模块。

首先，CtrlUnit需要对当前instruction进行解析，获得指令中的操作数、目的/源寄存器以及具体的指令操作；并基于这些信息判断需要使用的functional unit。

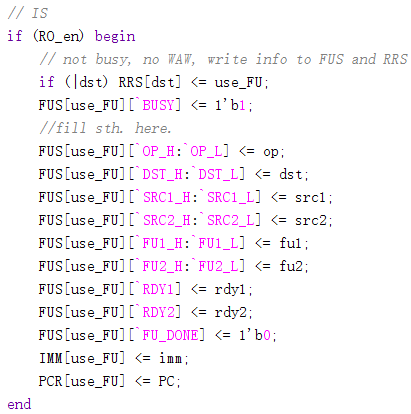
接着，我们需要在IS阶段对structural hazard以及write after write进行甄别；只要功能单元预期重复使用，或register result status相同位置预期重复写入时，我们就需要stall流水线的运行。

接着，我们需要判断IS阶段以及RO阶段进入的使能信号；并当JUMP类指令计算完成时，我们需要根据结果对IS阶段进行flush操作并在过程中控制stall的产生。进而，我们根据直接从instruction分离获得的可能的rd、rs1、rs2，对真实的dst、src1、src2进行验证（可能一些指令不存在相关的目的/源寄存器），同时记录当前源寄存器是否已经准备就绪或获得来源为何种functional unit。

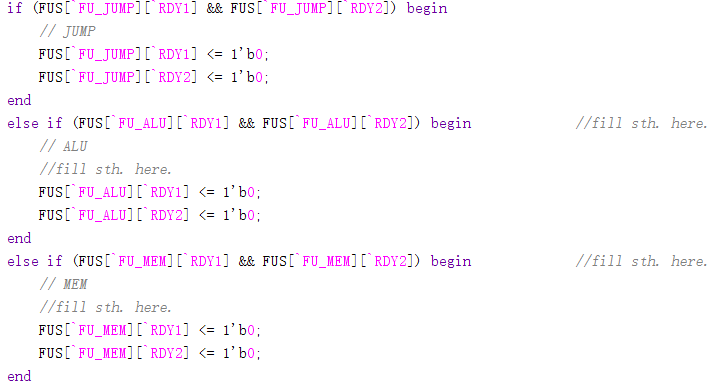
基于以上的信息，我们可以判断当前指令是否会发生write after read的问题。对于某一个functional unit而言，若其目标寄存器与其他functional unit的源寄存器相同且其他functional unit的源寄存器就绪状态为yes时，说明write after read冲突产生。在此仅以ALU计算单元为例给出代码实现细节。



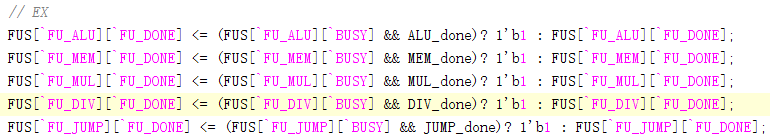
另外，在CtrlUnit模块中，我们还要完成scoreboard的记录工作。首先，当我们可以从IS阶段进入RO阶段时，我们需要把对应的functional unit busy状态置位为1，同时写入需要运算的具体指令，并把destination寄存器、source寄存器及其对应准备状态或数据来源的信息均写入scoreboard。具体的代码实现见下：



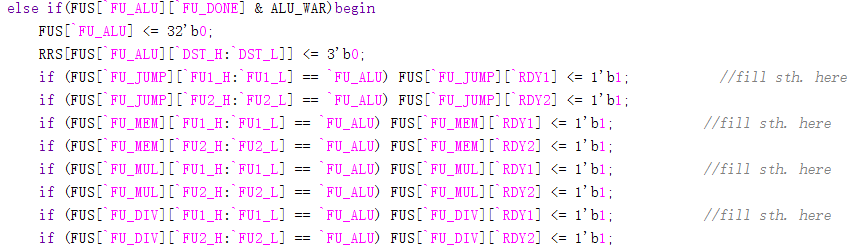
而在RO阶段运行时，我们需要先check每个functional unit的运算数据是否已经准备完全；若准备完全，则可以进入EX阶段，此时需要再次将数据的ready状态置为NO，减少write after read现象的产生。部分代码实现如下：



在进入EX阶段并完成相应的计算后，我们需要将functional unit status中的done状态置位为1，表示该功能单元的计算已完成。这里需要特别注意，因为functional unit的完成信号并不是在完成后一直被置位为1，所以我们要在根据完成信号置位done状态信号的前提下，保留done状态的已有信息。具体实现如下：



而在EX阶段之后，我们进入了WB阶段，需要把计算的结果写回register file中去。此时，因为我们已经完全使用完毕了functional unit，所以我们需要把functional unit status中对映的状态记录信息全部置0，并将register result status中相应的目标寄存器写入来源状态清空；最重要的，是清除write after read的情况；当该functional unit的目标寄存器与其他functional unit的源寄存器发生冲突时，我们需要将其他functional unit的源寄存器的对映ready位置位为1，这表示它依赖的数据已经可以通过计算获得。在此以ALU functional unit的判断作为实现代码示例：

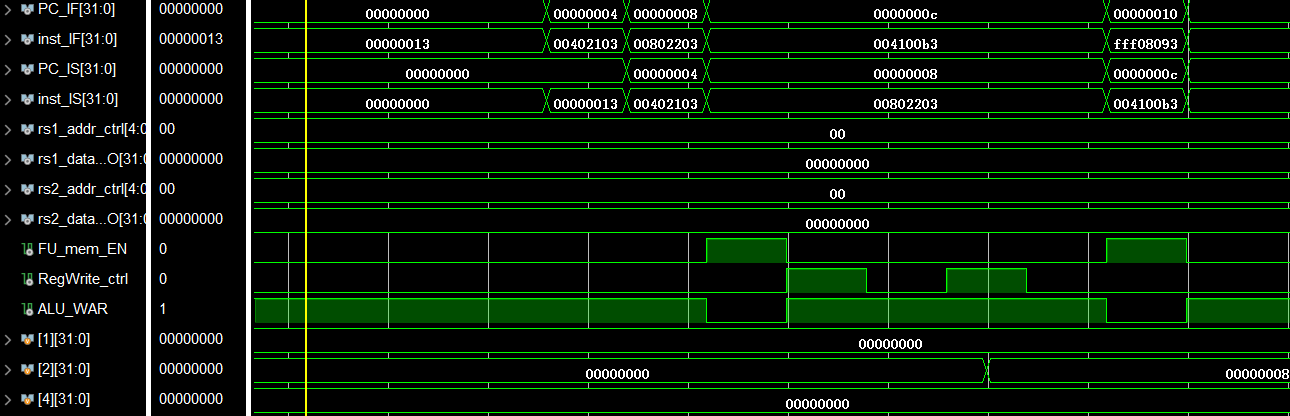


在此实现部分之外，我们还需要在确定functional unit运算单元时，为对映的模块选择相映且唯一的使能信号，并把对映的计算数据来源（数值）以及计算指令（具体）要求传递给对映的模块，以介导它们的正确计算。在write back阶段，也需要根据完成计算的functional unit设置相应的write back选择信号，并将register write信号置为1。至此，则完成了CtrlUnit模块的全部功能设计。

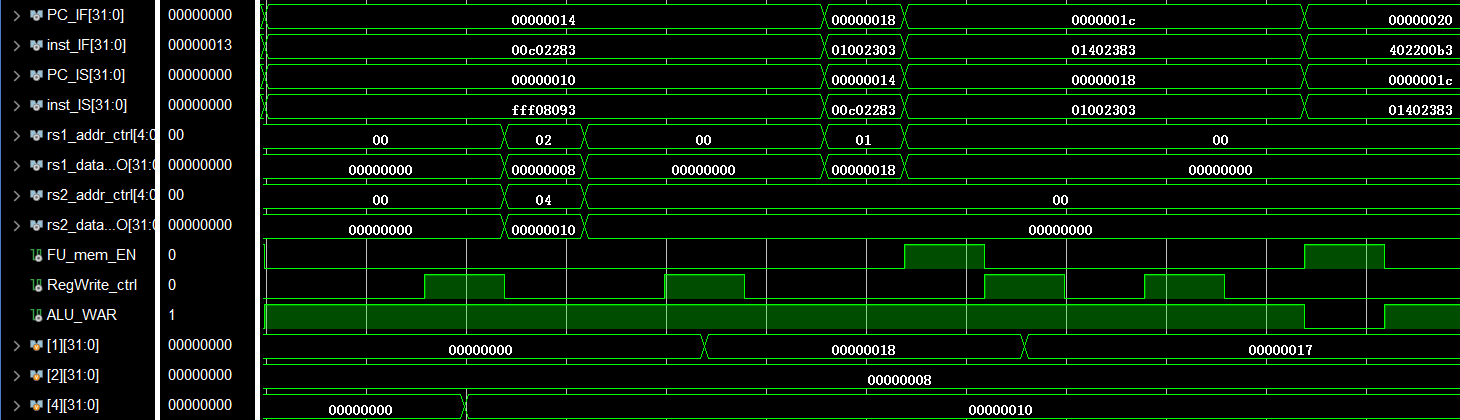
1. 实验结果分析

仿真验证：

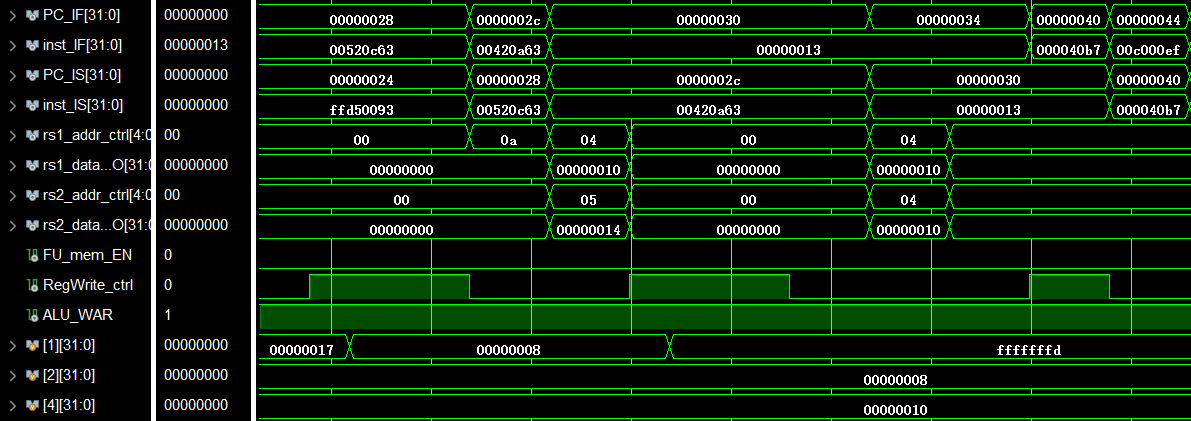
在此，我们仅选择部分仿真结果来说明仿真试验的正确性。



由上图可见，在rst清零信号变为0后，流水线开始正常的运行。PC开始不断递增，并将相应的指令填入动态流水线中。我们可以看到，最开始三条指令正常运行，因为有lw指令的存在，FU\_mem\_EN使能信号也被正常置位，表示memory存取操作的存在。但对于第四条指令而言，两个lw指令发生了structural hazard，因而流水线产生了normal\_stall冲突，使流水线需要暂停片刻。而随着reg\_write\_ctrl信号的置位，alu计算以及第一个lw运算的结果，也被成功放入了相应的寄存器中。

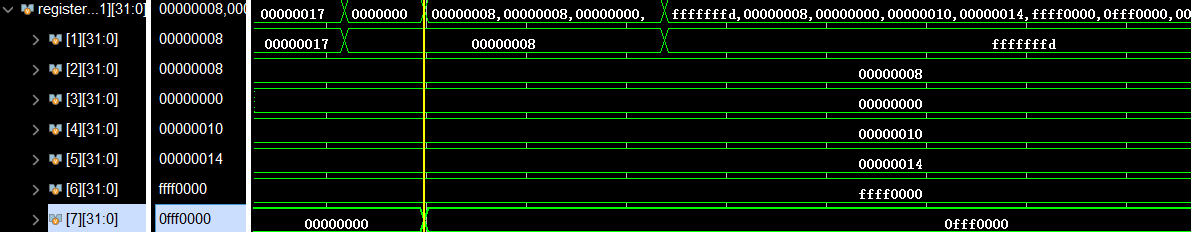


流水线继续运行，因为第三条add指令与第四条addi指令的目标寄存器均为x1，因而发生了write after write的冲突情况，根据流水线设计，normal\_stall信号又被置位为1，流水线需要陷入stall状态，因而位于 PC = 14的指令被stall进入。之后，从FU\_mem\_ctrl信号的置位可以看出，第二个以及第三个lw指令也开始被正常的执行，且随着RegWrite\_ctrl信号的置位x1，x2，x4寄存器也被成功写回了正确的数值。

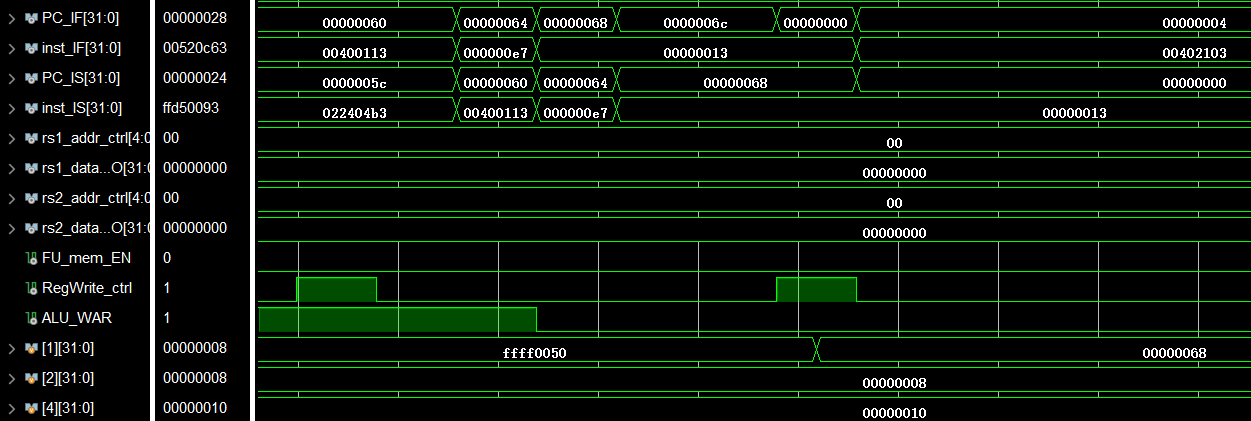


接着执行后序指令，我们首先可以看到，动态流水线完成了一个branch跳转操作，成功跳转至PC = 40 的label0标签处。第11/12/13条指令虽然被读入流水线中，但并未有效执行，从结果中我们可以看到，虽然这些指令均被解析且进行了一部分的传递，但最终均被跳转指令的flush信号全部重置为0。另外，我们也可以从结果得知，不同寄存器被先前计算的指令写入数据。

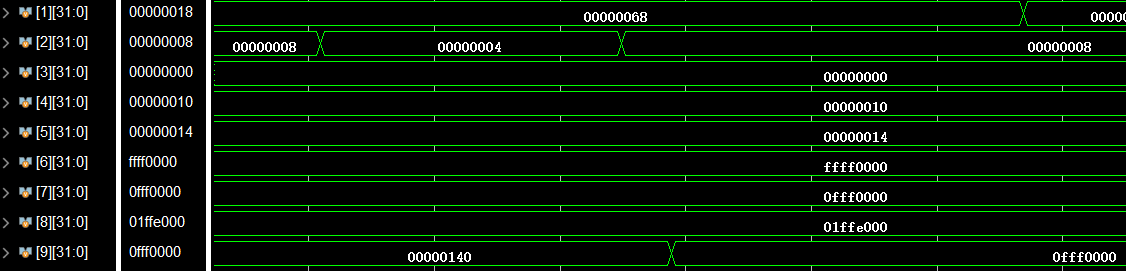
这里需要特别注意，因为第7条lw指令与第8条subi指令同时计算完成，得到可以写入的结果；但因为一次clock周期只能提供一个写入选择信号，为保证写入信号的不重叠与正常进行，我们在设计时默认先写回MEM functional unit计算获得的数据，再写回ALU functional unit计算获得的数据。



接着，在第一轮指令运算结束后，我们看到jalr指令被正确的解析与运行，PC重新跳回了第一条指令。因为第22与第23条指令均使用了mul乘法运算单元，因而我们可以从结果中看见，第23条指令（PC = 5C）因为structural hazard被阻塞。另外，因为第23条mul乘法指令与第24条addi指令分别将x2寄存器作为源寄存器与目的寄存器，因而产生了write after read冲突，这将会延后addi指令的写回时间（addi指令往往比mul指令更快得到计算结果）。



最后，我们来查看运行完全部程序后寄存器获得的实际结果。因为程序循坏执行，会对寄存器的数值判断产生一定的影响。

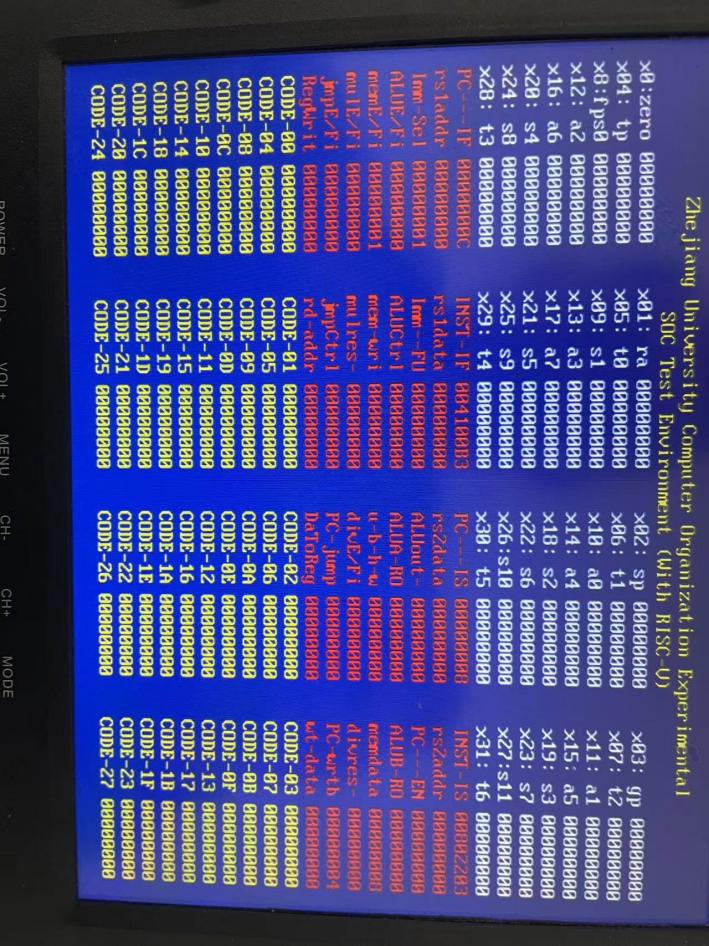
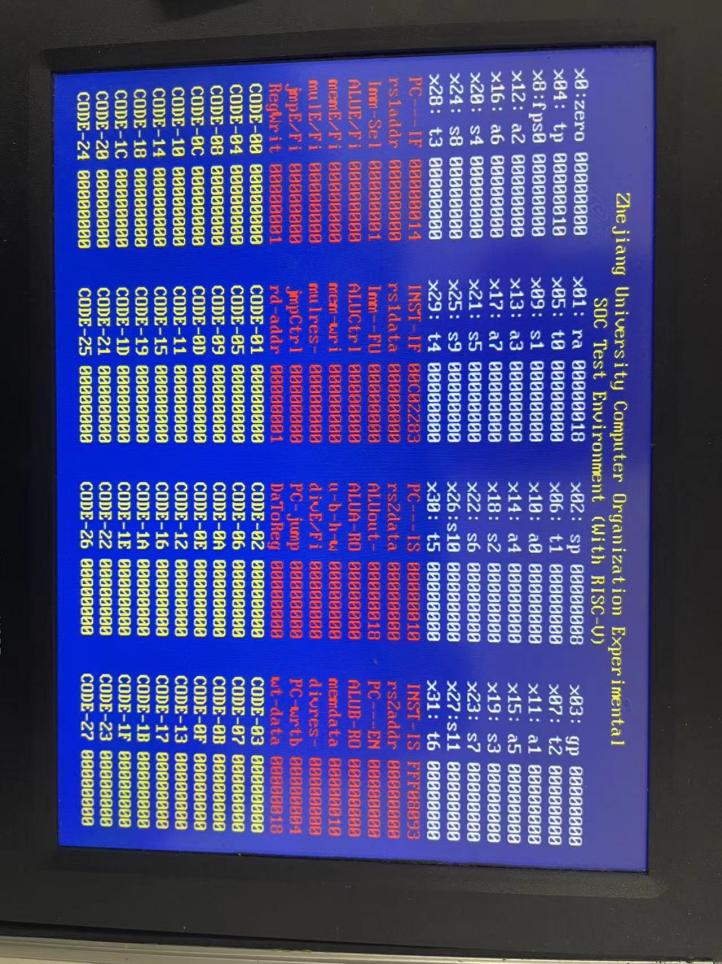


由结果可以看到，x1 - x9寄存器最终分别获得了0x68、0x4（第二轮开始改变为0x8）、0x0、0x10、0x14、0xffff0000、0xfff0000、0x1ffe000、0xfff0000的数值，与程序运算的数据预期保持一致，说明了我们仿真结果的正确性。

物理上板验证：

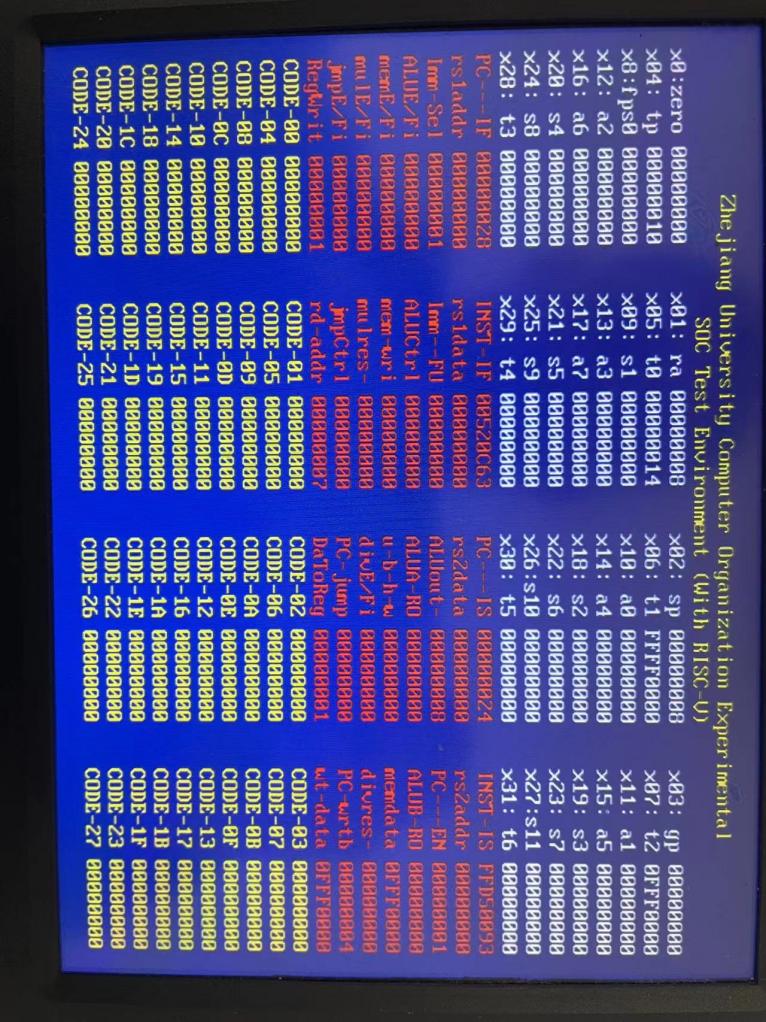
因为本次实验实现的动态流水线虽然顺序发射指令，但乱序执行指令，且数据计算提交情况也顺序不一，因而在此使用寄存器的写入结果来判断上板结果的正确性。

在初始阶段，我们可以看到流水线的正常运行，读取指令，并进行相应的计算。当IS阶段指令需要访问MEM内存时，相应信号memE/Fi被置位为1。

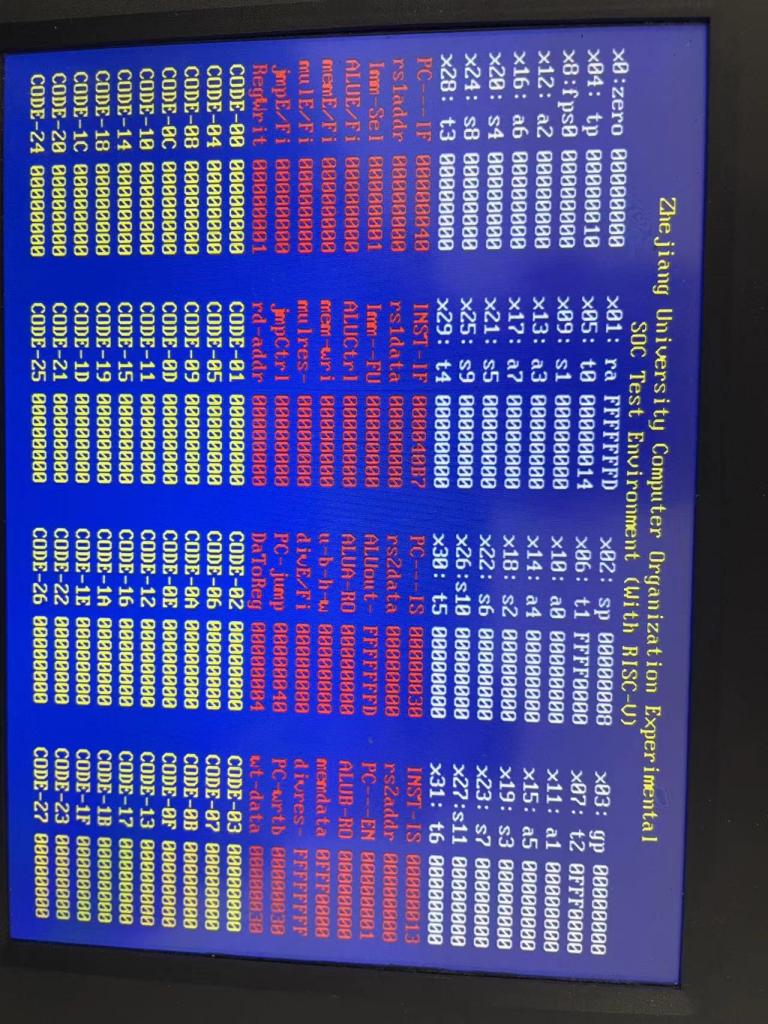


通过上图我们可以看到，x1、x2、x4寄存器均被写入了正确计算数值。其中寄存器的写入顺序为x2、x4、x1，因为lw指令structural hazard与add指令的源寄存器依赖关系，使得这三条指令为正确顺序的输出关系。在需要写回寄存器时，reg\_write信号被置位为1。

由下图可见，x1寄存器中的数值发生变化，x5、x6、x7寄存器被写入计算获得的数值。总体而言，这三条语句也是顺序写入指令计算结果的；但这三条指令并不为相邻指令，这说明了该动态流水线的乱序输出特性。

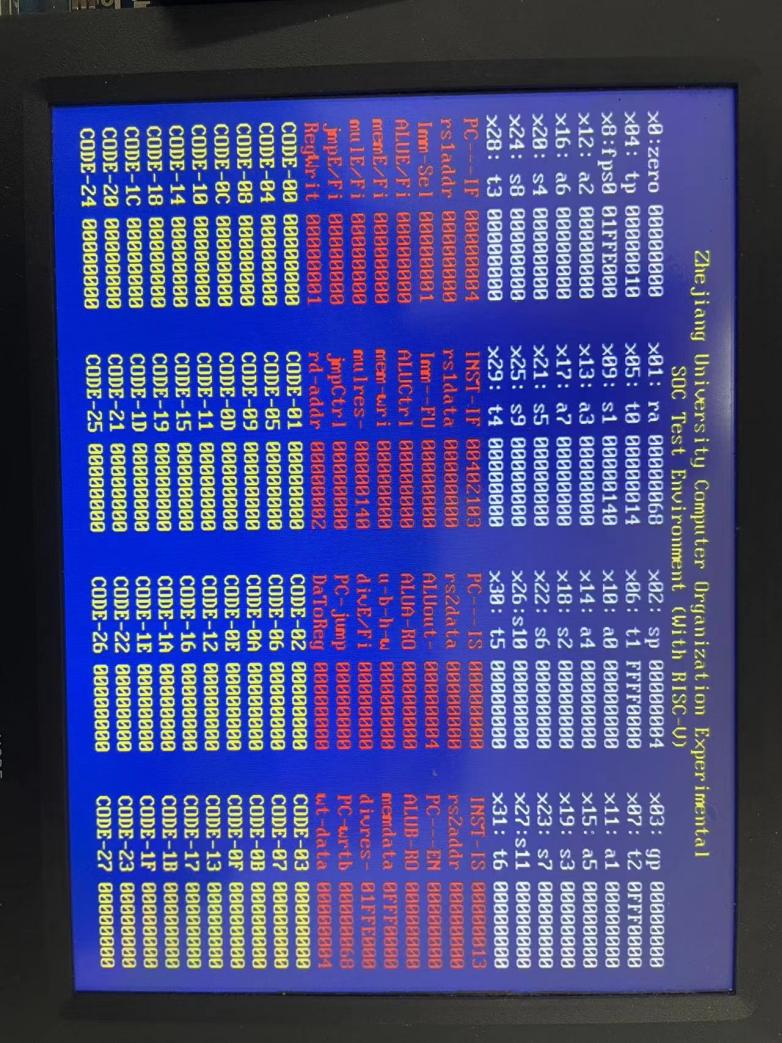


动态流水线，依然保持了正常branch的特性。

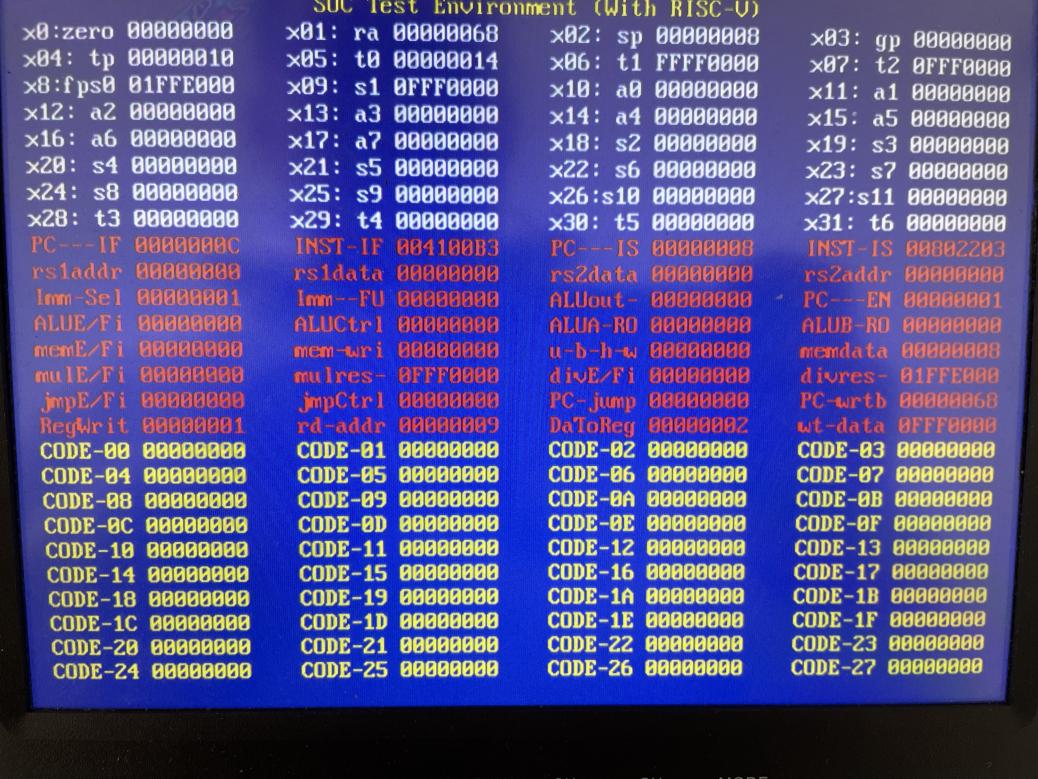


在之后执行的一连串指令中，x1寄存器的数值被一次修改为0x8, 0xFFFFFFFD, 0x4000, 0x48, 0xFFFF0050，因为是对一个目标寄存器的数值修改，因而这些数据的写入顺序一定满足着指令的发出顺序。需要特别注意的是，虽然sub x1, x4, x2指令后于lw x7, 20(x0)指令发射，但因为两者functional unit需要的计算周期数不同（前发射者多于后发射者），导致这两条指令同时计算完成；在本次实验的设计中，我们选择先将MEM操作指令的结果写回，再在下一时钟周期中写回ALU functional unit运算得到的结果。

从下图结果中我们可以看到，x1、x2、x8、x9寄存器中的数值均被修改。而这些寄存器数值修改的顺序却与指令发射顺序不同。这是因为x2寄存器的数值计算依赖于除法的完成，虽然除法的计算没有受到任何hazard的影响，但因为除法本身计算需要消耗大量的时钟周期，从而导致了其更迟计算完毕的情况。另一方面，因为两个mul指令的相邻，导致了structural hazard的发生，且第二个mul更是依赖于除法的计算结果（否则会产生write after read错误），因而至此还没有得到计算结果。



最后，我们第一轮程序仅剩的乘法操作完成，x9寄存器中的数值被成功修改，得到了正确的结果。另一方面，我们看到x2寄存器中的数值又从4转变为8、x1寄存器数值等于0x68的写入（当前执行指令地址）以及PC指令重新跳回0xC的位置（已经过一定clock周期的运行），都说明程序中设置的jalr操作被正常解析与运行，且branch指令跳转并没有干扰divider functional unit等功能单元的计算过程。

****

1. 讨论与心得

终于完成了硬件类课程的最后一个vivado verilog设计实验，体系结构lab完结撒花！

但在本次实验的完成过程中，也还是遇到与发现了不少问题，收获满满。首先是scoreboard的done状态记录问题，在最开始的实现中，我只是通过不同functional unit的finish信号来对done状态进行赋值，且这个赋值仅仅是其与finish信号作等；但在具体的处理过程中，finish信号并不是在functional unit完成后一直被置1，因而done状态仅在finish信号置位时短暂置1。又因为write after read冲突的存在，指令进入done状态后，并不总能及时的写回，这往往会导致done状态的无效设置问题。

另外，在本实验的程序运行时，某一clock周期有两个functional unit同时计算完成，而具体的写回操作只能依次进行；因而在write back阶段，在每一个clock周期内，我们只能判断及处理一个被置位的functional unit done信号，这也意味着在具体实现设计时我们需要使用if-else if的判断语句。

最后也是最离谱的是，vivado对于已有文件的清除操作十分不彻底，会残留已经经过remove操作的文件信息（IP核设定），这导致我的乘法器IP在初次设置错误后，虽然经过改正，但仍然运行不正确，带来了极大的debug阻碍。