

**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 11 月 22 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： Pipelined CPU supporting multi-cycle operations

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 11 月 22 日

1. 实验目的和要求

实验目的：

1. Understand the principle of piplines that support multicycle operations.
2. Master the design methods of piplines that support multicycle operations.
3. Master verification methods of pipelined CPU supporting multicycle operations.

实验任务：

1. Redesign the pipelines with IF/ID/FU/WB stages and FU stage supporting multicycle operations.
2. Redesign of CPU Controller.
3. Verify the Pipelined CPU with program and observe the execution of program.
4. 实验内容和原理

在本实验中，我们需要借助已有的框架实现多周期操作的流水线CPU。多周期流水线CPU的逻辑电路设计示意图见下。

对于单周期流水线CPU与多周期流水线CPU而言，其最大的区别在于EXE与MEM阶段。在单周期流水线CPU中，我们在EXE阶段进行ALU等操作的运算，可以得到一系列指令要求的计算结果，如乘法除法等；另一方面，我们也可以在此阶段计算获得需要跳转的PC地址以及需要访存的内存地址，介导PC的branch跳转或MEM阶段的内存读取操作。



但在多周期流水线CPU中，我们并没有了明显区域界限的EXE阶段与MEM阶段。在ID阶段与WB阶段之间，我们根据指令的需求，分化出五个不同的处理模块，来介导不同指令要求的操作；分别管理除法，乘法，跳转，ALU计算以及memory存取操作的进行。因为这些操作对映的时钟周期开销不同，因而在ID阶段与MEM阶段之间，我们需要使用的时钟周期数将根据指令的不同而发生不同，且在本实验中，我们对于不同的指令操作，均采用stall的方法，等待指令的逐一执行。

多周期流水线CPU具体的实现表示如下：

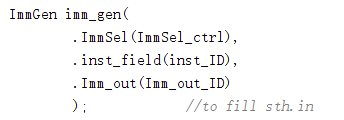
1. 实验过程和数据记录

在本实验中，我们仅需借助已有的实验框架，完成六个文件的填充即可。

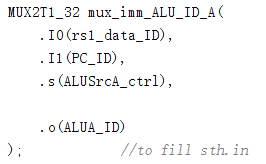
* 1. RV32core模块

在本模块中，我们需要完成三个mux选择器与imm\_gen立即数生成模块的数据来源、选择线路来源及输出目标的连接即可。

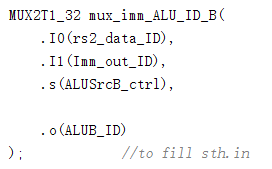
首先是立即数生成模块，immsel表示立即数如何扩展的控制来源，inst\_file记录着当前拆解的具体指令，imm\_out记录经扩展后实际获得的立即数。



接着是两个输出计算数据的mux选择器。对于第一个选择器而言，数据来源于寄存器组的数据获得以及当前ID阶段的PC数值（对映于auipc的指令操作），且根据选择信号，将此输出连接于ALUA\_ID线路。

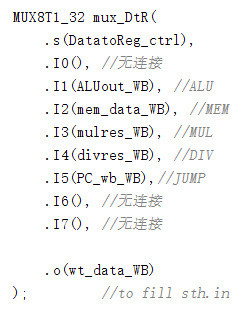


对于第二个mux选择器而言，数据来源于寄存器组的数据获得以及立即数的扩展生成，经过控制信号的选择，结果将连接至ALUB\_ID线路。

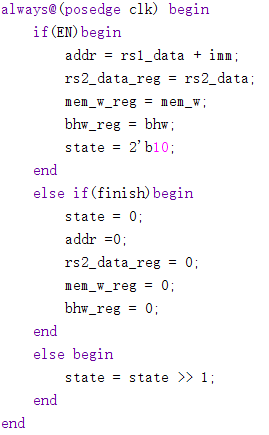


因为在多周期的流水线CPU中，我们使用了五个模块进行不同功能的信息计算；因而在WB阶段，我们需要通过一个mux选择器选择正确的指令要求计算来返回具体的操作结果。

对于这一mux选择器而言，需要使用8 - 1的配置，在五个功能单元的选择之外，我们可以使另外三个接口不接入任何数据。选择器的1 - 5接口分别对映ALU计算、mem数据获得、乘法运算、除法运算以及jumpPC位次运算的结果。

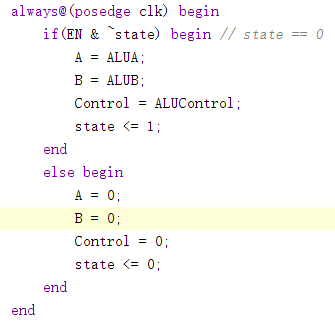


* 1. FU\_mem模块

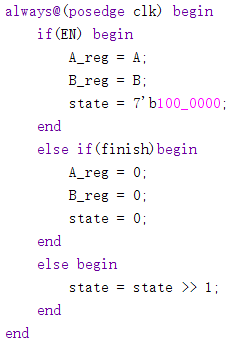
在该模块中，我们实现了mem数据存取的功能。一般而言，我们对memory数据的存取，都是基于address实现的；在一般的指令中，我们需要借助ALUA的结果以及立即数偏移，来获得address的具体数值。接着，我们根据state变量的周期数暗示，利用三个时钟周期，来完成memory读取的操作。

* 1. FU\_ALU模块

在本模块中，我们主要实现了除乘法除法外的计算单元功能，包含加法、减法、或运算、与运算等操作。根据给定的控制信号以及输入数据，我们使用两个时钟周期来完成这一操作。首先进行运算操作的选择与具体寄存器的赋值，再进行相应的运算操作。

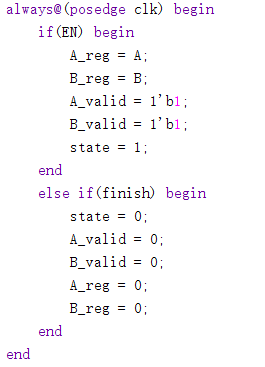


* 1. FU\_mul模块

在本模块我们主要实现乘法运算功能。根据state信号量的定义，我们需要7个时钟周期，来完成对应的乘法操作。在此，我们通过不断的右移state，来充当clock cycle的计数功能。同样，我们使用了finish信号来表示乘法的计算完成，且在乘法计算完成后，我们将计算输入以及state信号均置位为0。

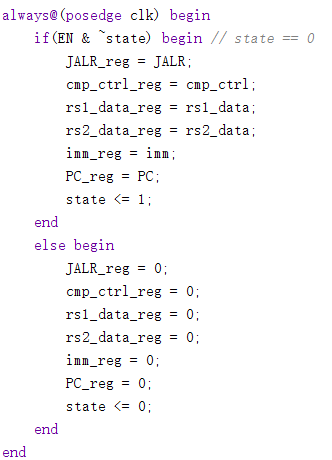
* 1. FU\_div模块

在本模块中，我们主要实现除法的计算功能。因而使用的vivado除法ip核自带了除法完成信号，因而对于将近30+的clock cycle，我们不需要再使用state信号量进行一一计数。因而在具体的实现中，我们只需要在除法使能被置位时，简单的将计算输入参数赋值，并且将其对映的valid信号置1即可进行除法运算。在除法运算结束后，我们也同样需要将这些信号重置为0。



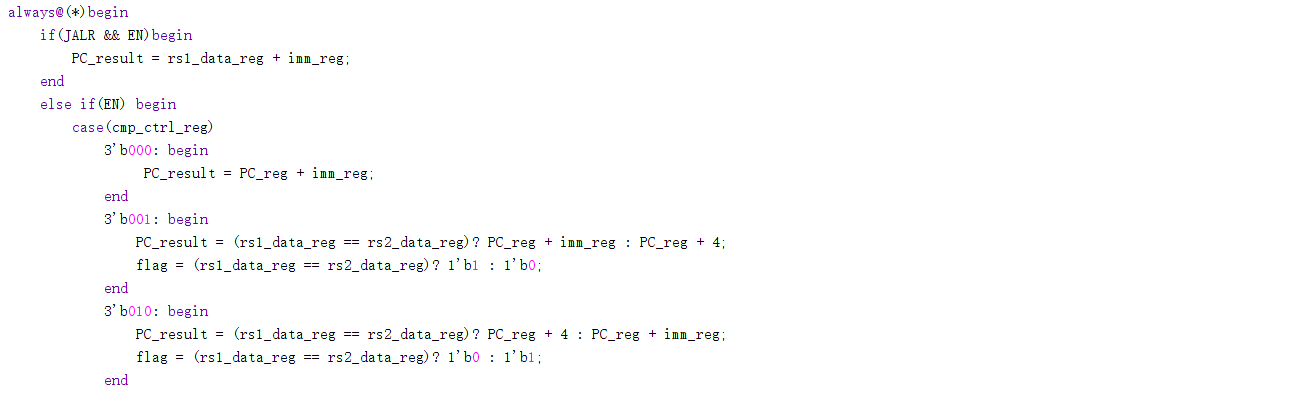
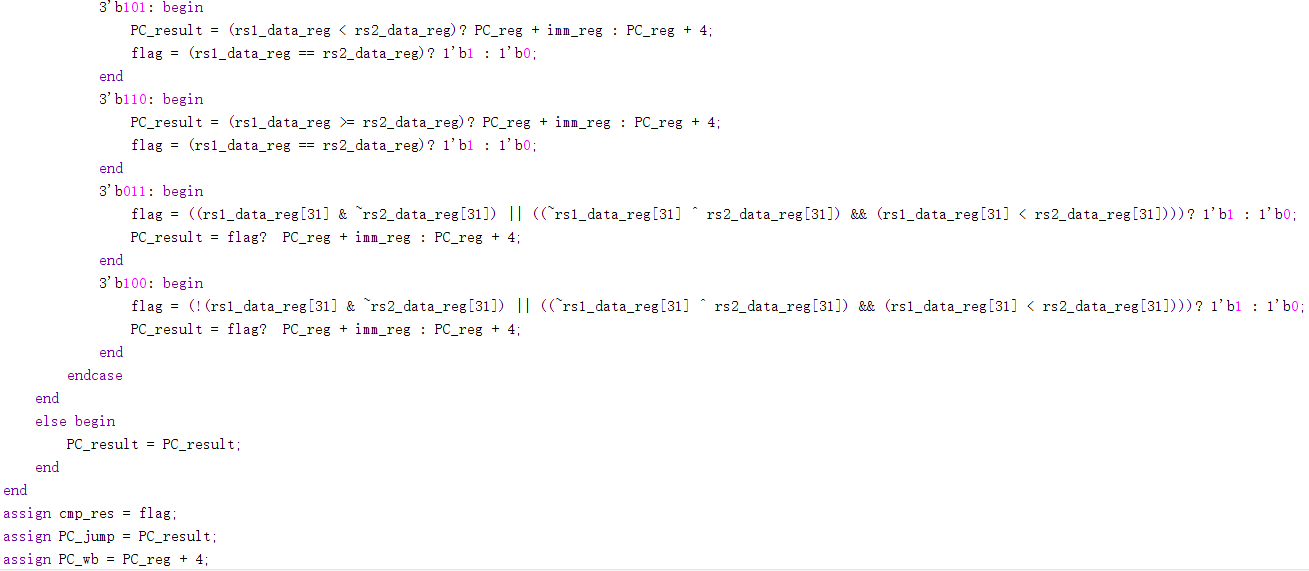
* 1. FU\_jump模块

在最后一个模块的设计中，我们需要计算决定下一条PC指令的读取位置，以及在需要的时候将当前PC + 4的数值写回到寄存器组之中。



首先，一如既往的，我们需要为各个寄存器赋予初值。在本模块中，我们使用了六个寄存器，分别用于记录输入参数A、输入参数B和立即数扩展后的数值，以及当前PC数值，控制信号量与JALR控制信号量。

当JALR控制信号被置位时，表示我们需要处理的指令为JALR；在其余情况下，我们需要根据3位控制信号量的具体数值，来判断我们需要实现跳转的指令为beq,bne,blt,bge,bltu还是bgeu. 最后，我们需要根据输入数值的具体比较结果，来判断是否需要进行PC的跳转以及是否置位PC branch信号。

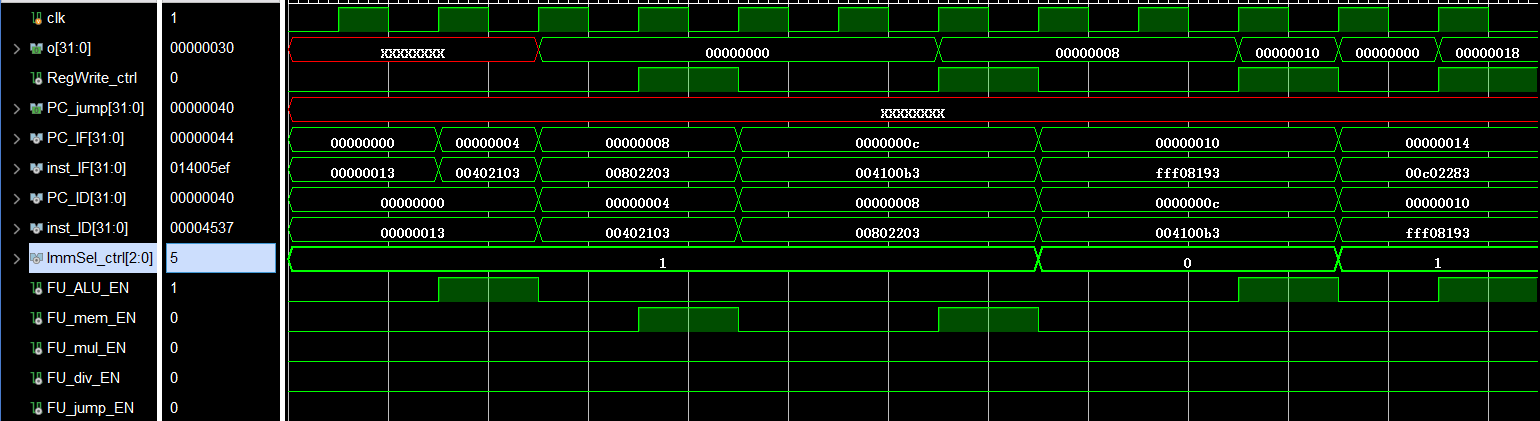


1. 实验结果分析

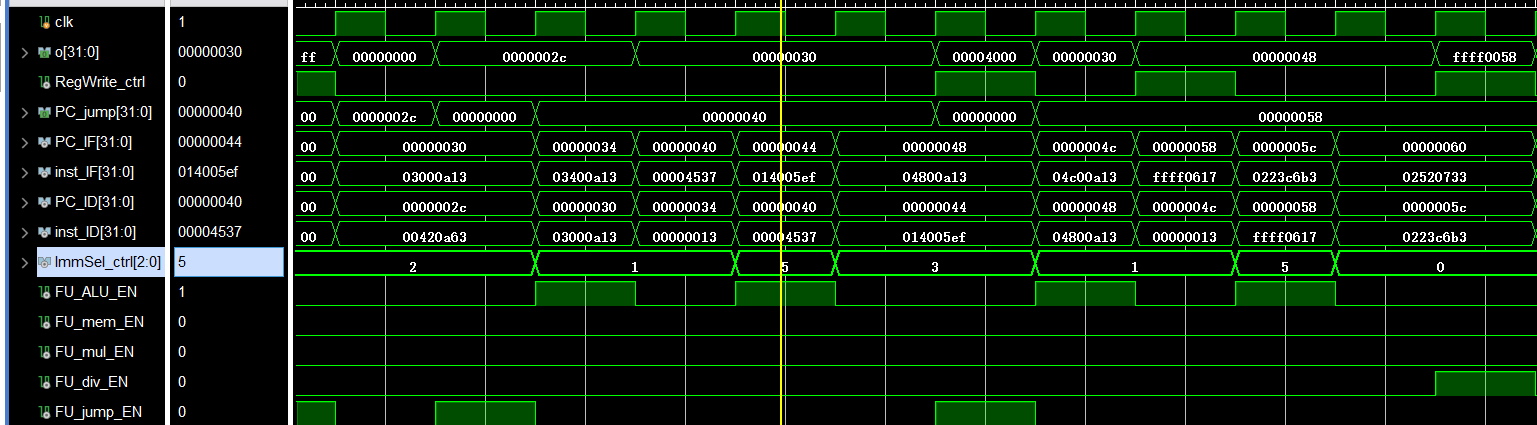
4.1 仿真验证

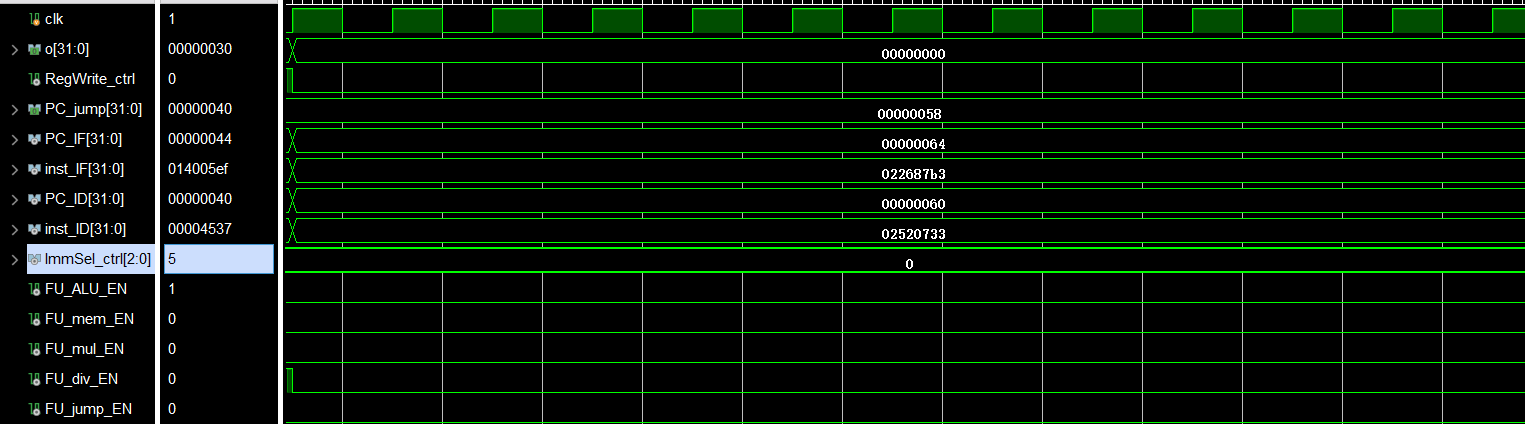
实现完该架构后，我们进行了仿真验证。在此我们仅取部分仿真结果说明程序设计的正确性。

通过下图我们可以看到，程序可以正常的执行ALU计算操作以及memory的存取操作。其中ALU计算操作需要使用2个时钟周期，而memory的存取操作需要使用三个时钟周期。另外，在进行ALU计算与memory存取操作时，不同功能模块的使能信号均会被分别置1。且随着write\_back信号使能的置位，我们写回的信号也与要求保持一致。

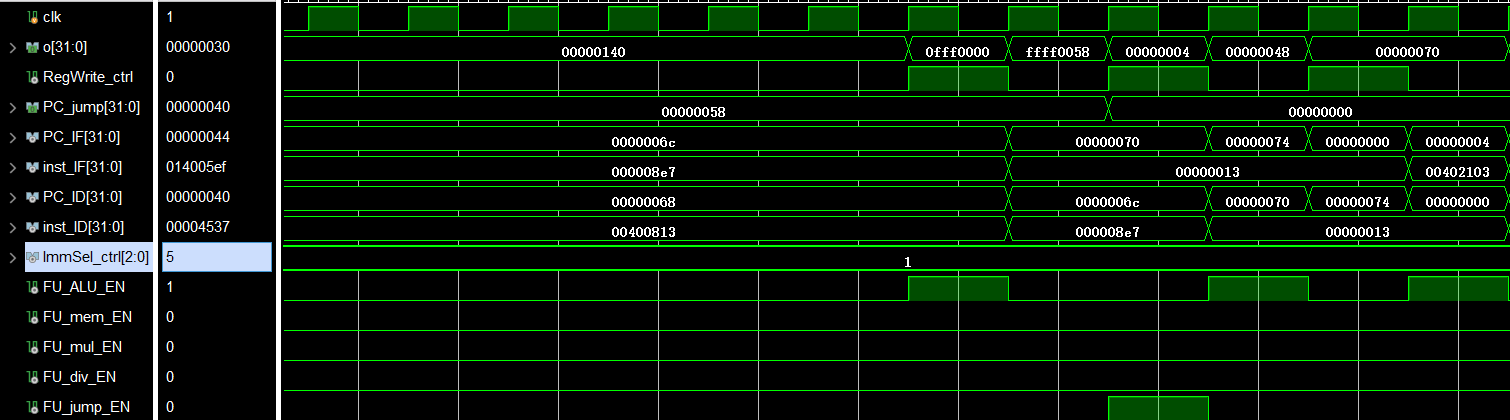


对于下图的实验结果我们可以看出，程序在正常的实行，且在处理第10条指令时，jump使能被置位，但PC没有进行相应的跳转。这是因为，跳转判定结果不满足跳转需求，因而没有跳转的发生。但在处理第11条语句时，跳转信号被置位且跳转条件满足，因而介导了正常的跳转操作。



通过仿真实验结果我们也可以看到，除法操作消耗了大量的clock cycle，且在运行除法时，整条多周期流水线CPU处于阻塞的状态；直至除法运算得出结果后，才可以正常的继续运行。

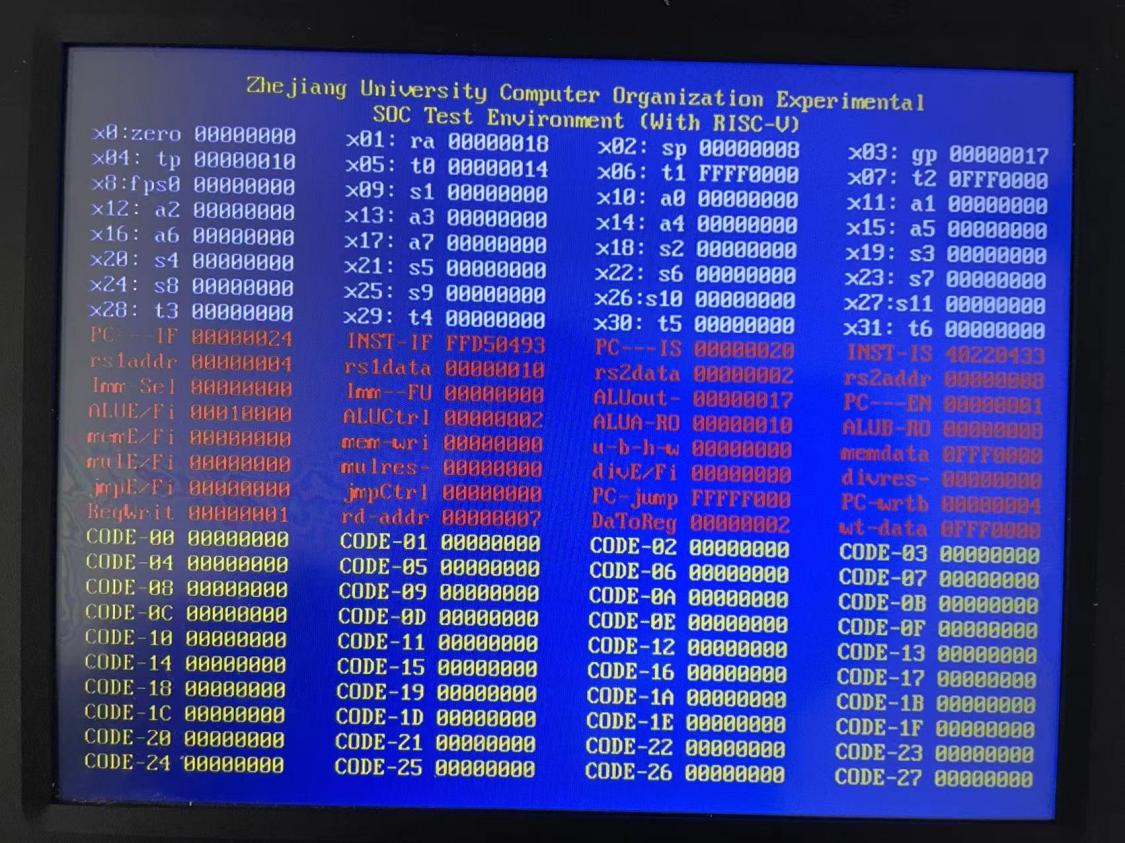
最后，我们观察到了JALR指令的正常运行。一方面jump信号被置位为1；另一方面，在write\_back信号使能时，我们成功的将当前的PC + 4数值写回了寄存器组中。另外，我们的PC也成功fetch了0地址处的指令并正常运行。



4.2 上板验证

在此仅选取部分上板实验结果作为程序设计正确性的验证。

首先是ld指令，在单步执行三个clock cycle后，ld指令获得的memory信息成功写回（输出）；我们可以看到，此时x1,x2,x3等寄存器被成功赋予正确的数值，说明先前执行的指令均获得了正确的结果。且ALU计算与memory内存读取，不同的功能实现也消耗了不同的clock cycle数量。



我们继续观察跳转指令的正确执行与否。可以看到，在执行第一个beq语句时，虽然jump\_ctrl信号被置位为1，但程序并未进行跳转，因为此时beq指令的跳转条件并不满足。在运行到第二个beq语句时，程序成功跳转至40地址处指令，并正常 + 4 运行后续指令。通过上板结果可以看出，先前的各部分计算输出均得到了正确的寄存器写入。

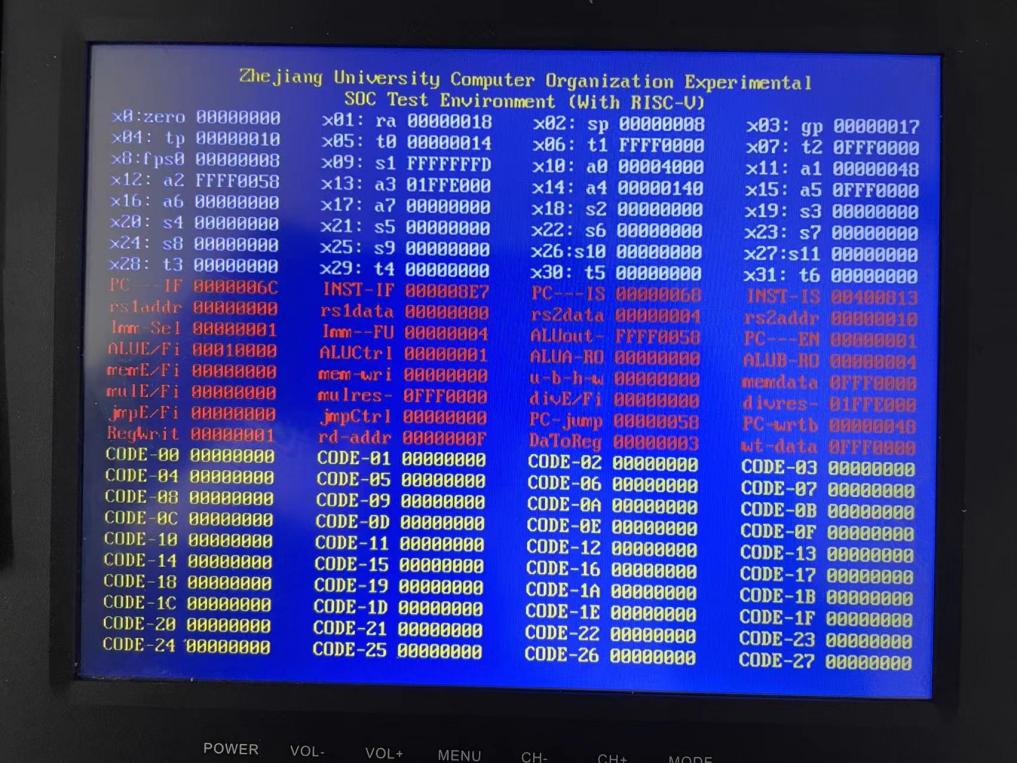




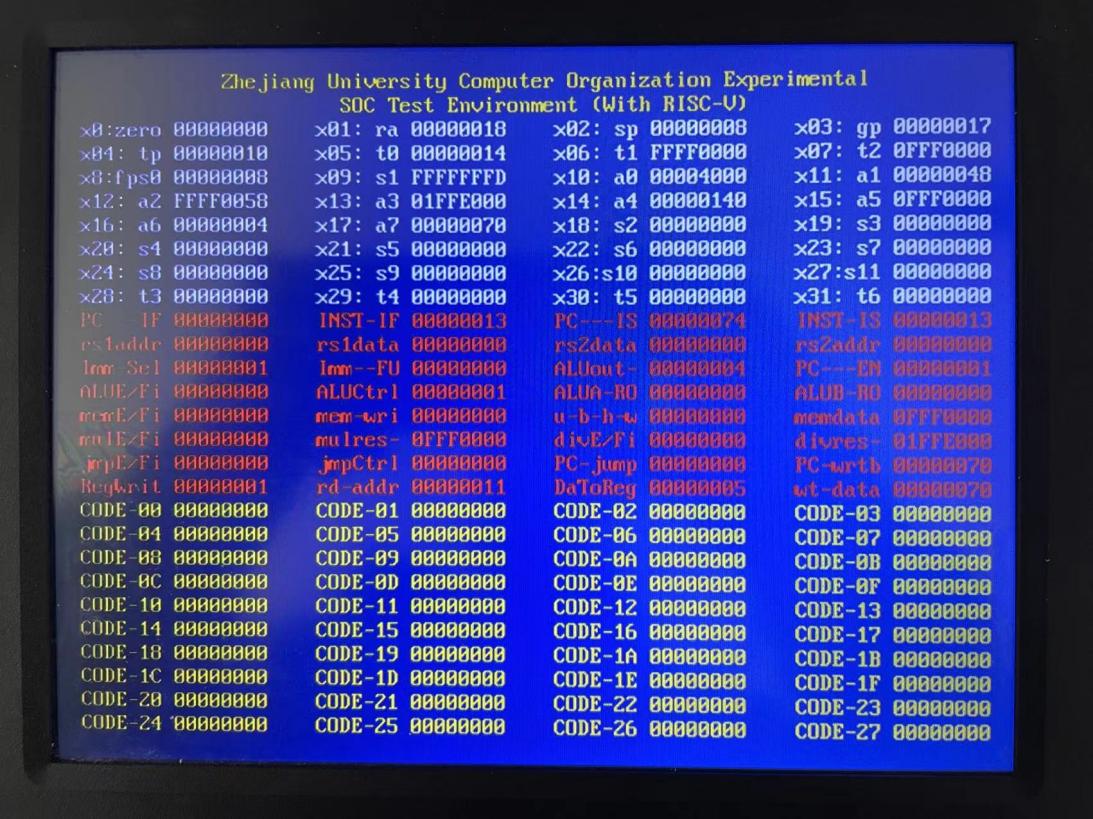
接着我们进行了div除法指令的探究。本次实验除法的实现，我们借助了vivado自带的除法IP核；因而在上板验证时，我们需要单步执行38个clock cycle，才可以得到除法运算的正确结果。



另外，我们也探究了mul乘法的IP核运算clock cycle消耗。相比于除法而言，在进行乘法运算时我们可以更快的得到结果。



最后是jalr语句的成功返回，从上板验证结果可以看到，经过jalr指令的执行，（下一clock cycle时）PC成功跳转到了0地址并获取了该地址的指令；且对于x17寄存器而言，当前PC + 4 的数值也被成功写入。



1. 讨论与心得

本次实验的实践实现还算简洁，但在mul、div等模块中，state的设置还是会让人捉摸不透，不知其确切的用法。主要还是mul模块中的state设置，[6:0]的定义与[0]位判断，总使人怪异为什么不直接定义state的数值，每次递减再与零比较，来check 计算的clock cycle是否已经足够。另外，state[6:0]的右移计数意味着6 - 7个计算周期，但乘法的具体实现好像只需要更少的2个周期就够哩。

另外，在RV32/core模块的“连线”部分，只有ALU计算模块使用了经过mux选择器选择的ALUA以及ALUB输出，其他模块还是使用着直接从寄存器组获得的数据；且在各自的模块接口，附加了PC等额外的信息输入。感觉这样的设计是不是有所缺陷，一方面可以直接取消两个mux选择器的存在，在功能模块处直接增加需要数据来源的接口；另一方面，或者应该简化功能模块的接口，使其只接受ALUA与ALUB的输入，而ALUA与ALUB的输出选择，则可以通过CtrlUnit控制模块的处理来实现，这样可能可以使程序更具规范性。