# Lab1-RV31I实验指导和实验报告要求

* 用verilog实现RV32I **单周期CPU**。
* **实验工具：Vivado**
* **实验方式：Vivado自带的波形仿真**

### 设计要求：

1. 参考4\_ProjectDesignFiles 文件夹中提供的CPU模块图和0\_SingleCycleLab文件夹中对应的Verilog代码，思考每条指令的数据通路，详细写出每个待完成模块的设计思路，并思考如何用verilog简洁高效的表达这些逻辑电路。
2. 待实现指令：
   1. RISC-V 32bit 整型指令集（除去FENCE,FENCE.I,CSR,ECALL和EBREAK指令）
   2. 可参考官方提供的RISCV用户指令集手册（中文版或者英文版），本文件夹内提供了
   3. 也可以参考<https://www.cnblogs.com/mikewolf2002/p/9864652.html> （注意本网站有些错误，对于branch指令的功能描述不是很准确，auipc指令描述有问题，jalr是I型指令不是UJ型）
3. 设计原则：可以完全按照我们提供的CPU模块图，补全模块内部逻辑完成CPU设计；也可以根据你的个人理解对CPU模块图做出相应修改并在报告中提出你自己的改进方案并给出修改原因。

### 测试要求：

**PART1:**

1. 自己手写合适的测试用汇编代码，通过提供的工具生成.inst和.data文件，用于初始化指令和数据的Block Memory，或者直接手写二进制测试代码
2. 测试用的指令流中需要包含的指令包括SLLI、SRLI、SRAI、ADD、SUB、SLL、SLT、SLTU、XOR、SRL、SRA、OR、AND、ADDI、SLTI、SLTIU、XORI、ORI、ANDI、LUI、AUIPC
3. 测试例（汇编和对应的.inst .data）可以用其他同学提供的，但是需要自己知道对应的指令逻辑，需要在报告中能清楚表达这个测试例如何验证CPU功能正确，即正确运行后寄存器值应该是多少
4. CPU执行后，各寄存器值符合预期

**PART2:**

1. 自己手写合适的测试用汇编代码，通过提供的工具生成.inst和.data文件，用于初始化指令和数据的Block Memory，或者直接手写二进制测试代码
2. 测试用的指令流中需要包含的指令包括**JALR、LB、LH、LW、LBU、LHU、SB、SH、SW、BEQ、BNE、BLT、BLTU、BGE、BGEU、JAL**
3. 测试例（汇编和对应的.inst .data）可以用其他同学提供的，但是需要自己知道对应的指令逻辑，需要在报告中能清楚表达这个测试例如何验证CPU功能正确，即正确运行后寄存器值应该是多少
4. CPU执行后，各寄存器值符合预期

**PART3：**

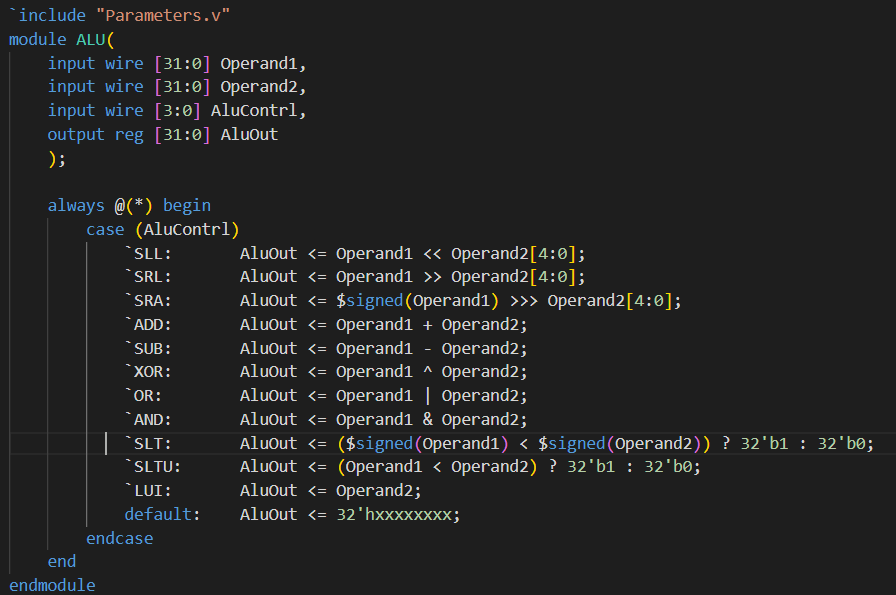
1. 利用我们提供的修改过得RISCV test官方测试汇编代码，通过提供的工具生成.inst和.data文件，用于初始化指令和数据的Block Memory
2. CPU开始执行后3号寄存器的值会从2一直累增，该数字正在进行第多少项测试，执行结束后3号寄存器值变为1
3. CPU执行后，各寄存器值符合预期

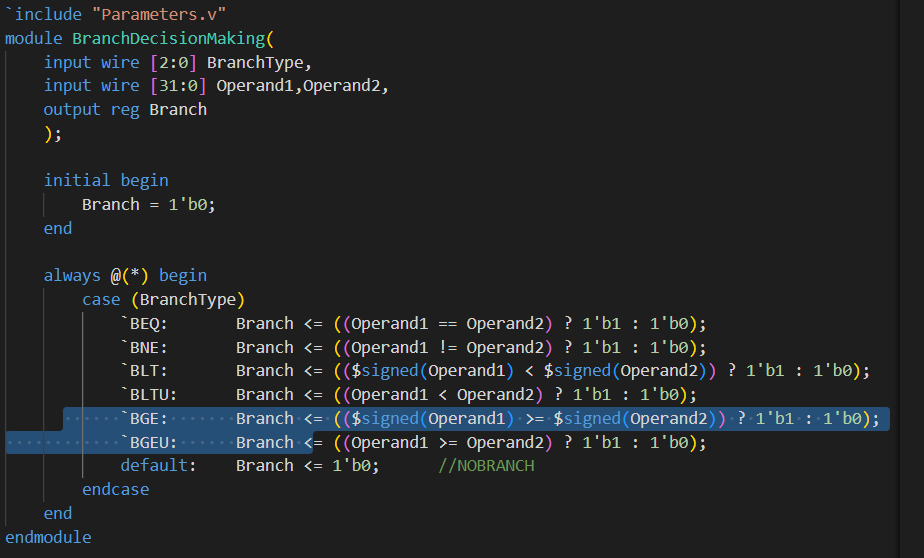
注意：三部分为递进关系，建议大家按照顺序进行测试

### 实验报告：

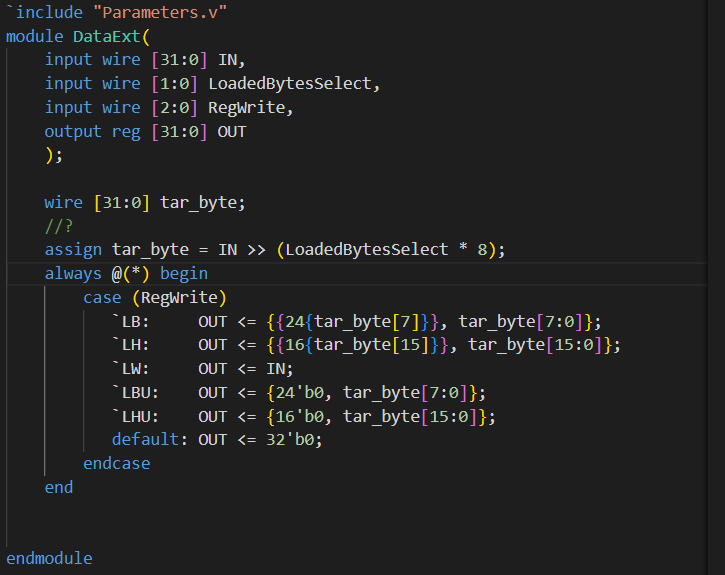
* 实验目标：用verilog实现RV32I **单周期CPU**。
* 实验环境和工具：Vivado及其自带的波形仿真
* 实验内容和过程（总结自己所做的三个阶段工作）：
  + 先在给定的框架下完成各个模块的单周期CPU代码：

1. 完成ALU,和BranchDecisionMaking 模块，这两个模块较为简单，只需要根据所输入的指令输出单一的输出结果，只需要分支语句就能完成主要的处理。

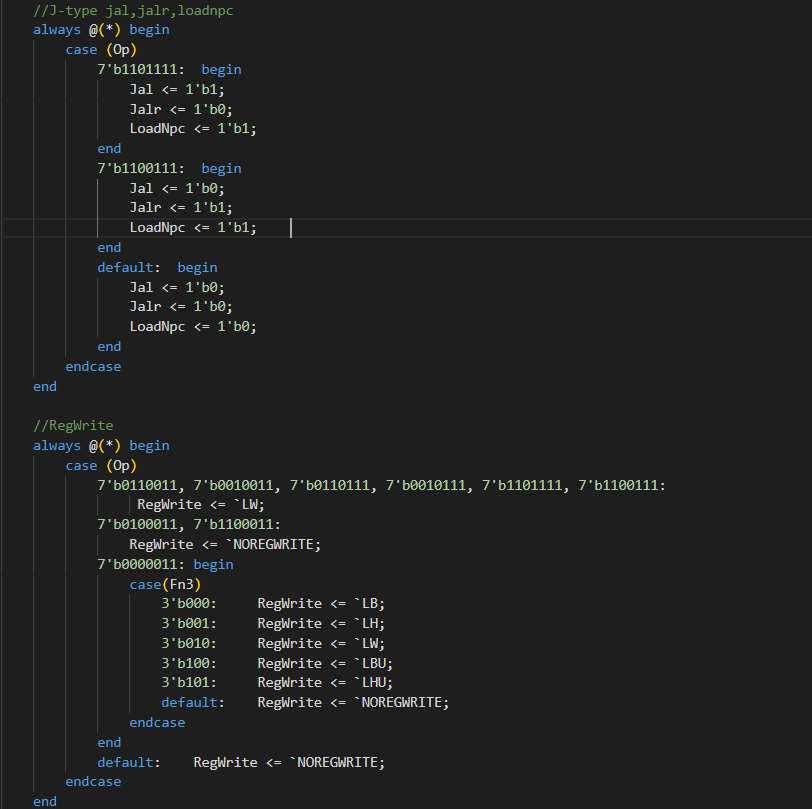


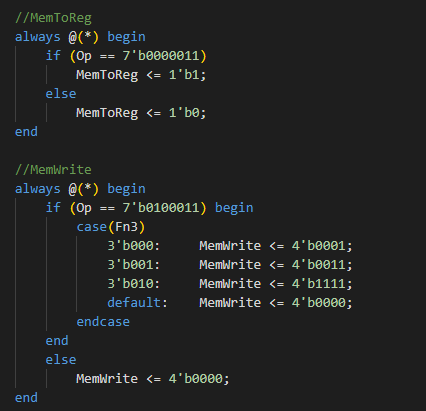


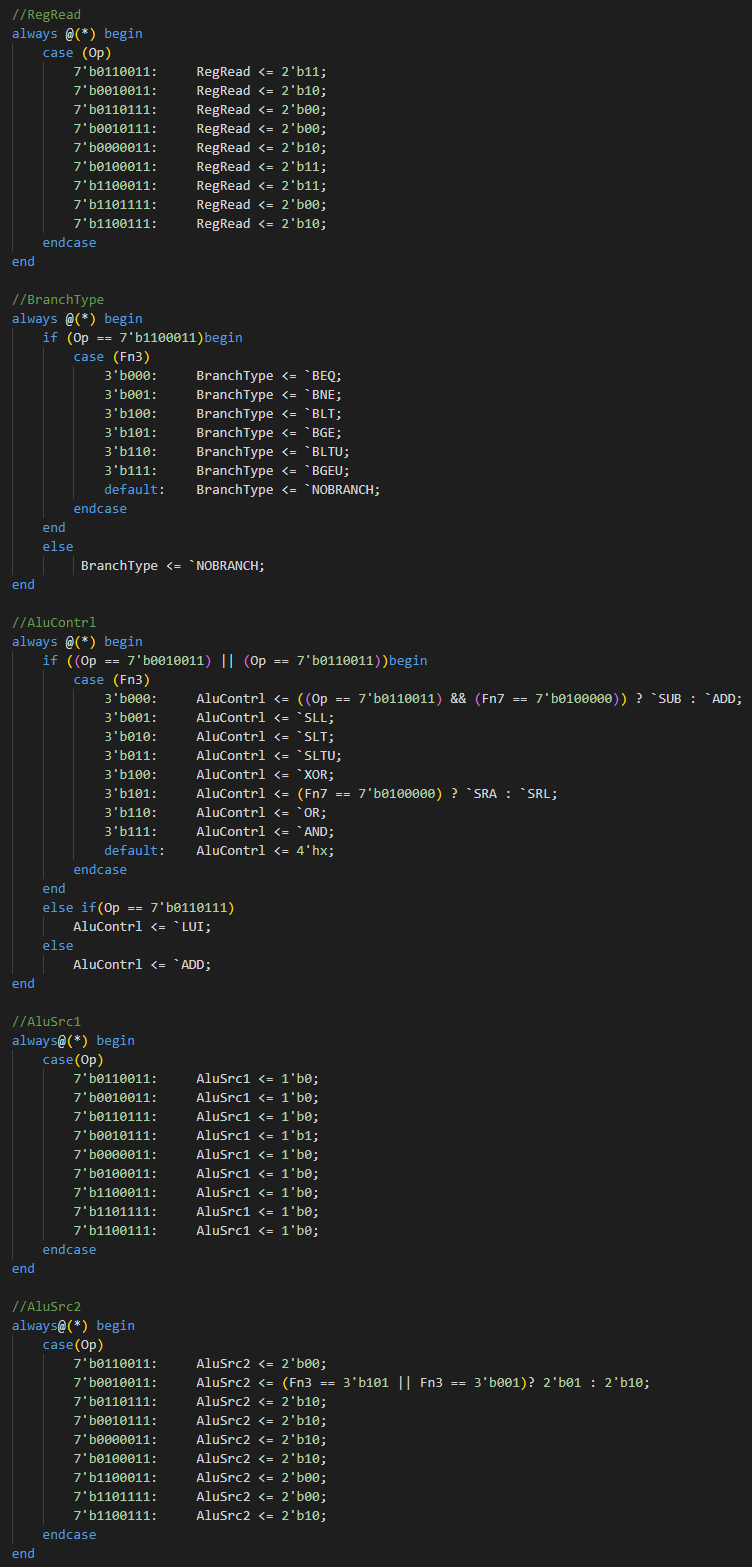
同样的，DataExt模块也是根据LD命令给出的RegWrite给出不同的寄存器加载形式，这里位置选择的方式是通过将输入的读取的整个字，移位代表一个字中要取的段落所代表的长度LoadByteSelect\*8（一个Byte长度），这样最低位即为所要读取的内容，并由此在进行位扩展：

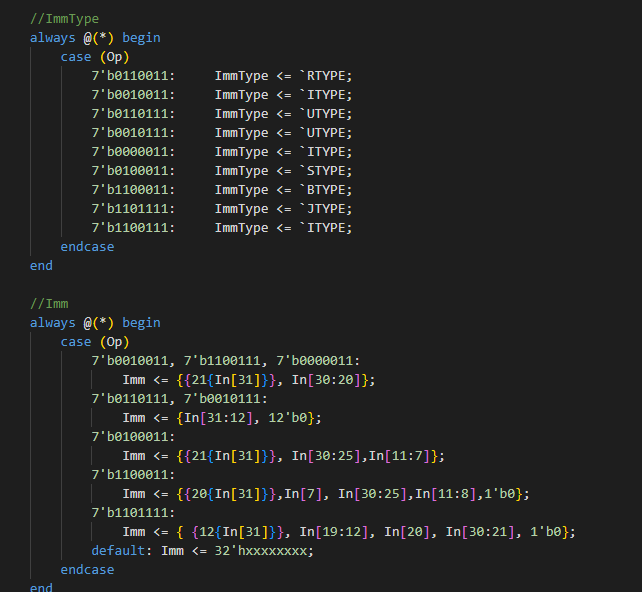


1. 完成Control Unit 模块，和上述模块不同，由于输出较为复杂，对每个输出分别处理会相较于使用一个分支语句一次处理完更简洁，

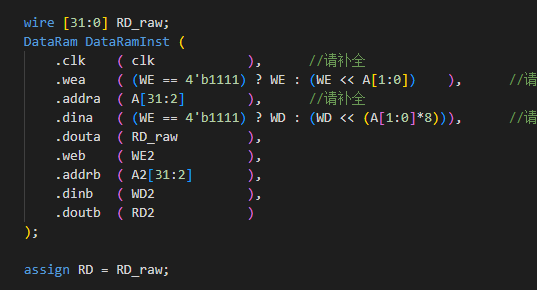


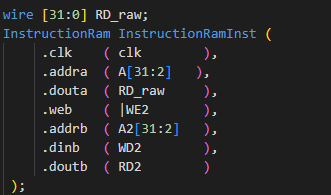




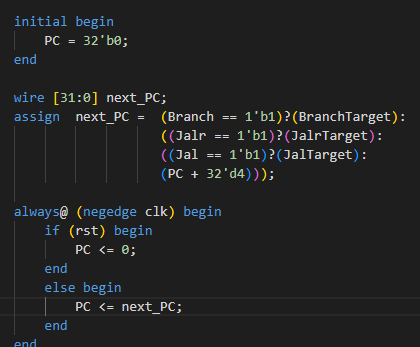


1. 补全IMEM和DMEM相关的，InstrSeg 和DataSeg模块的参数传递部分代码，需要注意的是，这些模块和上述组合逻辑模块不同，由于涉及到内存和寄存器的处理，需要使用时序逻辑的形式编写。在DataSeg中的参数传递会涉及store不同指令的处理，因此非sw的情况需要根据所得地址后两位做移位。





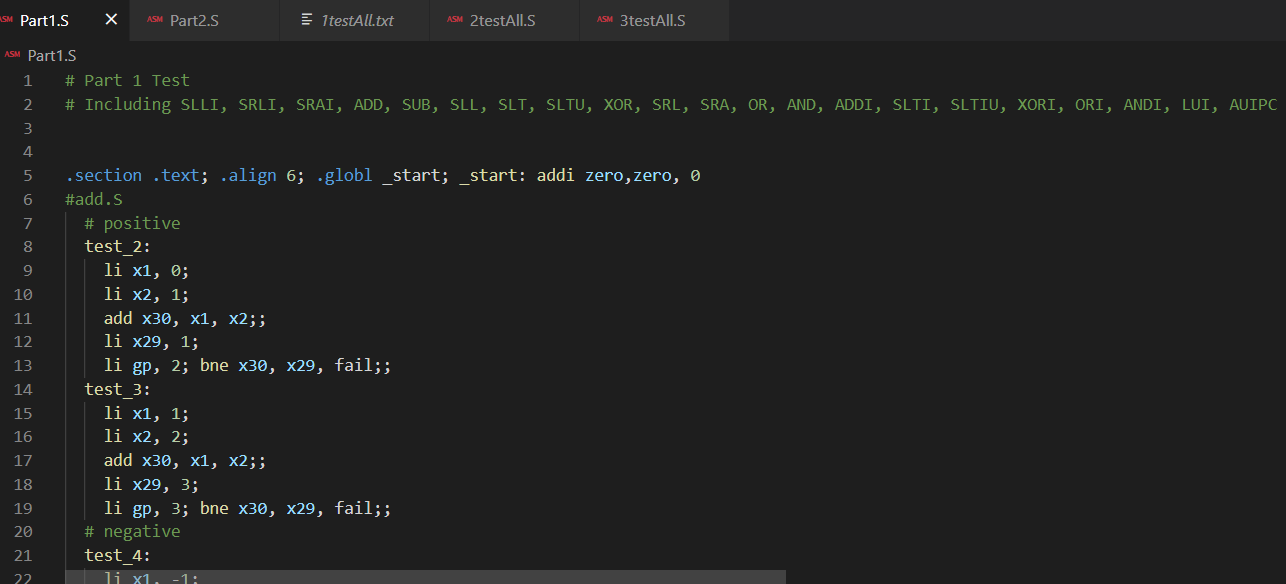
1. 特别的，对于PC相关的指令，由于所以模块都在相同的上升沿处理，如果简单的用NPC模块一个来解决并不做出较大的修改，比较难实现PC对应的正确的Instr值读取，因为仿真总体框架没有进行对每个模块时延的设计。这个问题在后期debug时是主要解决的问题，将在第三部分重点讨论。而我们的解决方法则是将NPC模块的触发时钟反置，即为negedge，即可在PC赋值后读取正确指令。



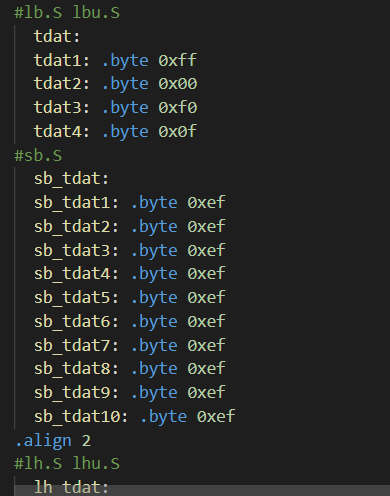
* + 再编写测试用的除官方测试集之外的两部分测试汇编代码：

主要是基于提供的part3测试文件做改写，去除了复杂的测试，只针对每种指令的基础实现进行测试。主要是提前发现BUG并做修改。

Part1的测试不涉及load和store指令，主要是做逻辑指令和立即数生成的测试



Part2的测试会着重于load、store和跳转指令，但事实上part1测试需要保证跳转指令的正确性，所以与part1相比，part2主要是测试load和store指令，但还是保留了跳转指令的测试。为此需要在汇编代码中设置数据区以供测试。这里直接借鉴了part3的数据区设置方式。



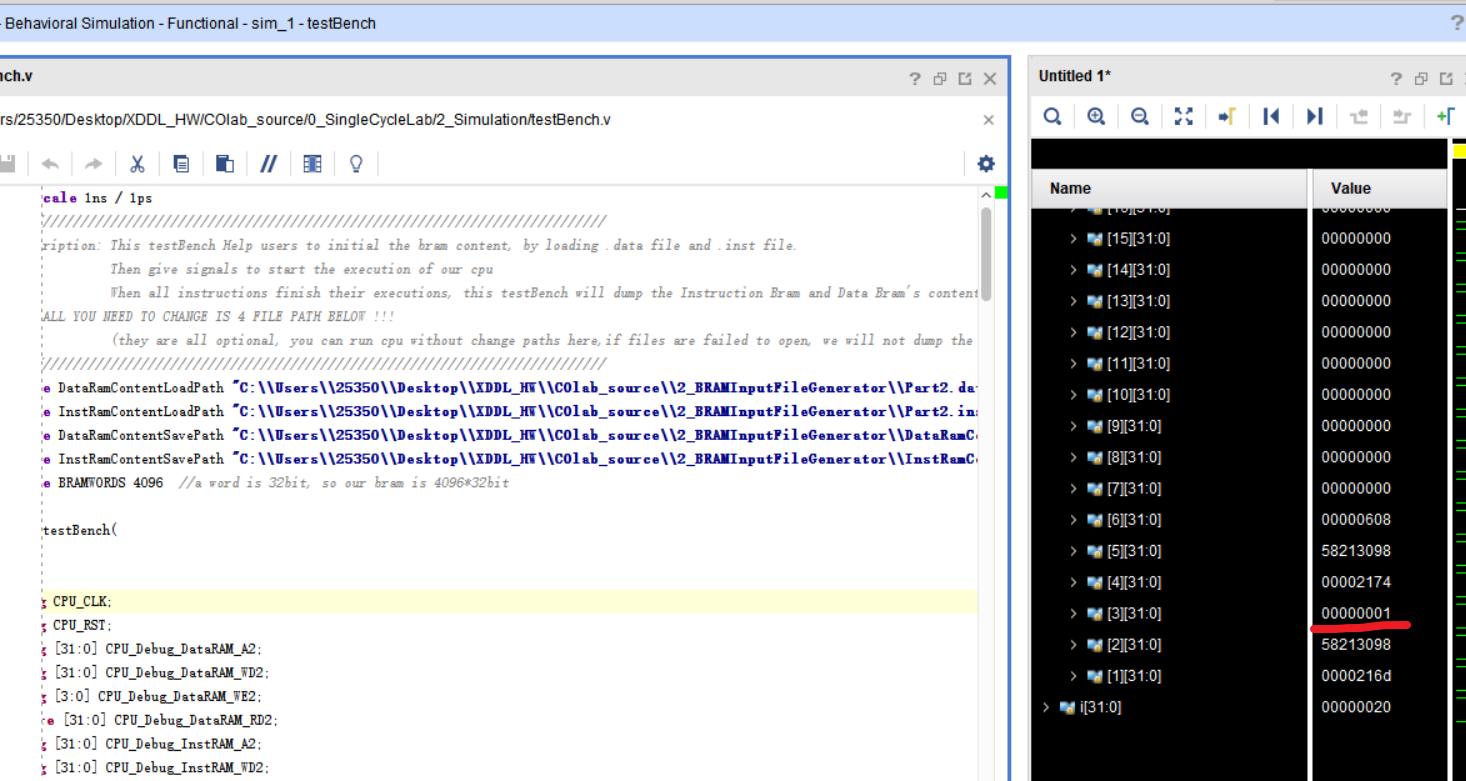
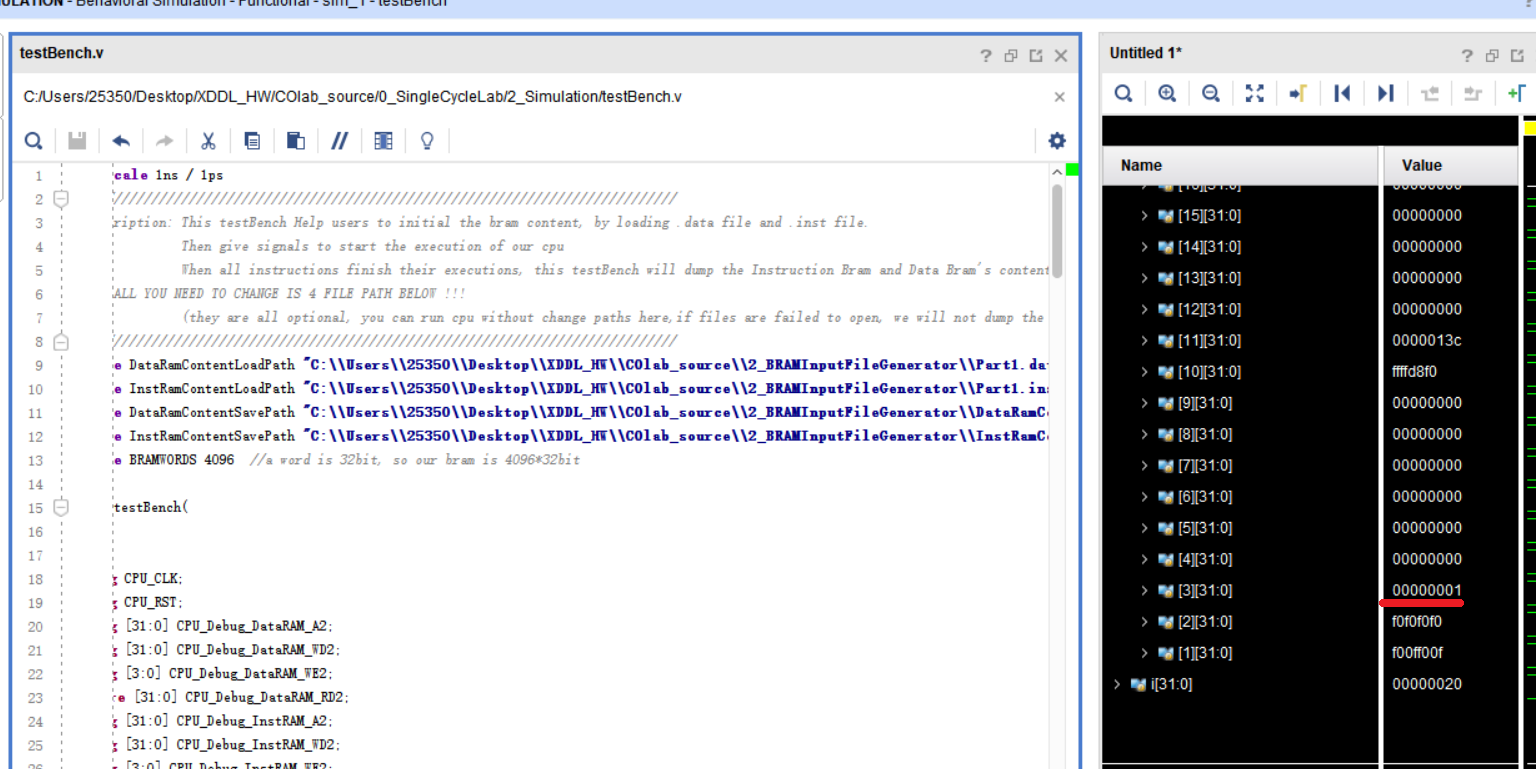
在测试结果判断上，part1、part2使用了和part3一样的逻辑（用寄存器x3记录测试指令的位置），当测试出错时放弃执行后续测试，直接跳入死循环中，保留出错时的寄存器信息。如果x3最后保留的值不是1，说明在x3值对应的测试指令处出现了错误，方便定位和排除。若测试用例全部通过则会在x3寄存器处显示1。

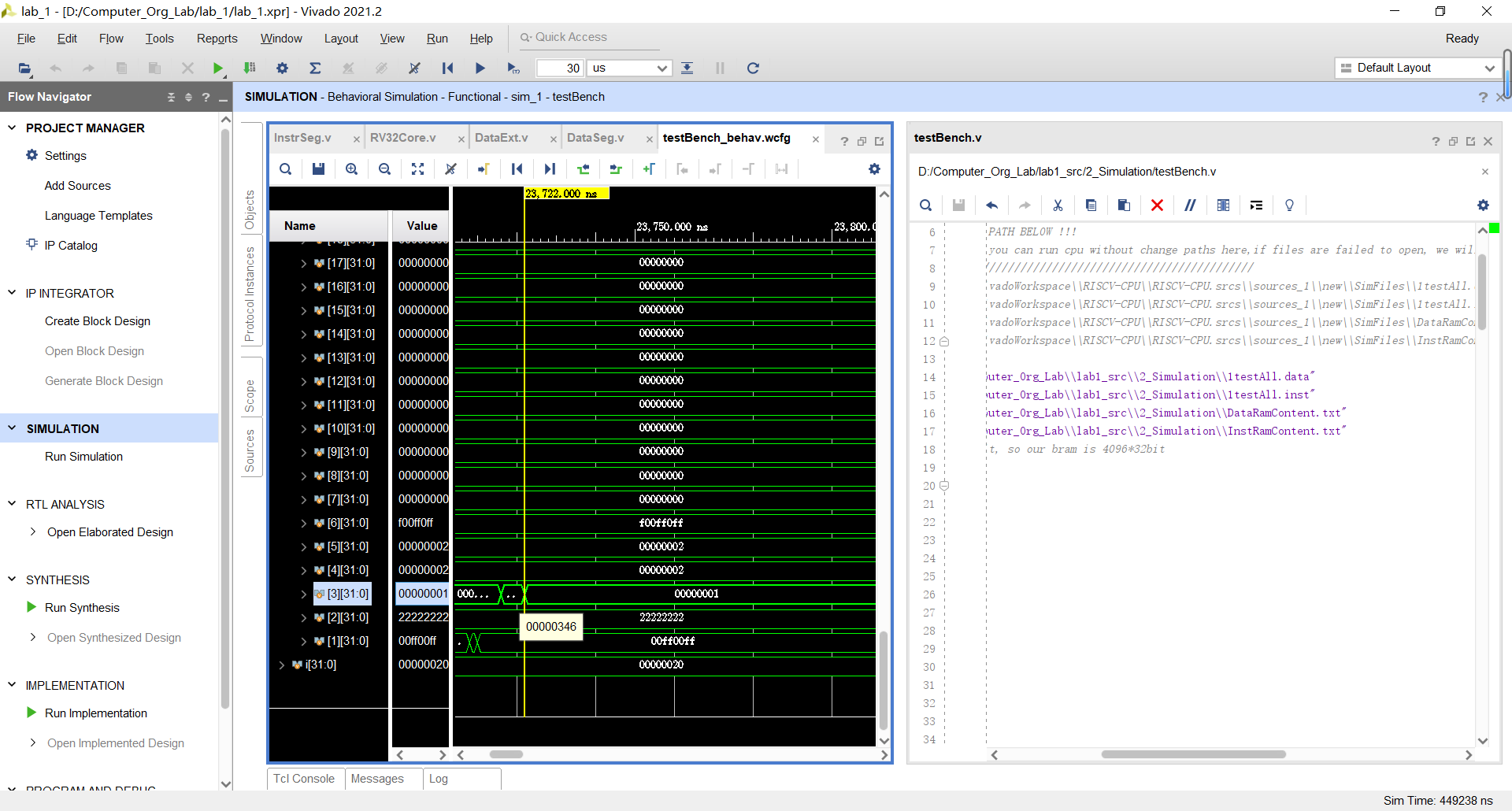
事实上这样的测试并不能实现指令分开测试，但是在结果保证上是合理的（可以方便确定指令是不是都正确执行），对一开始的debug能力会有所要求。

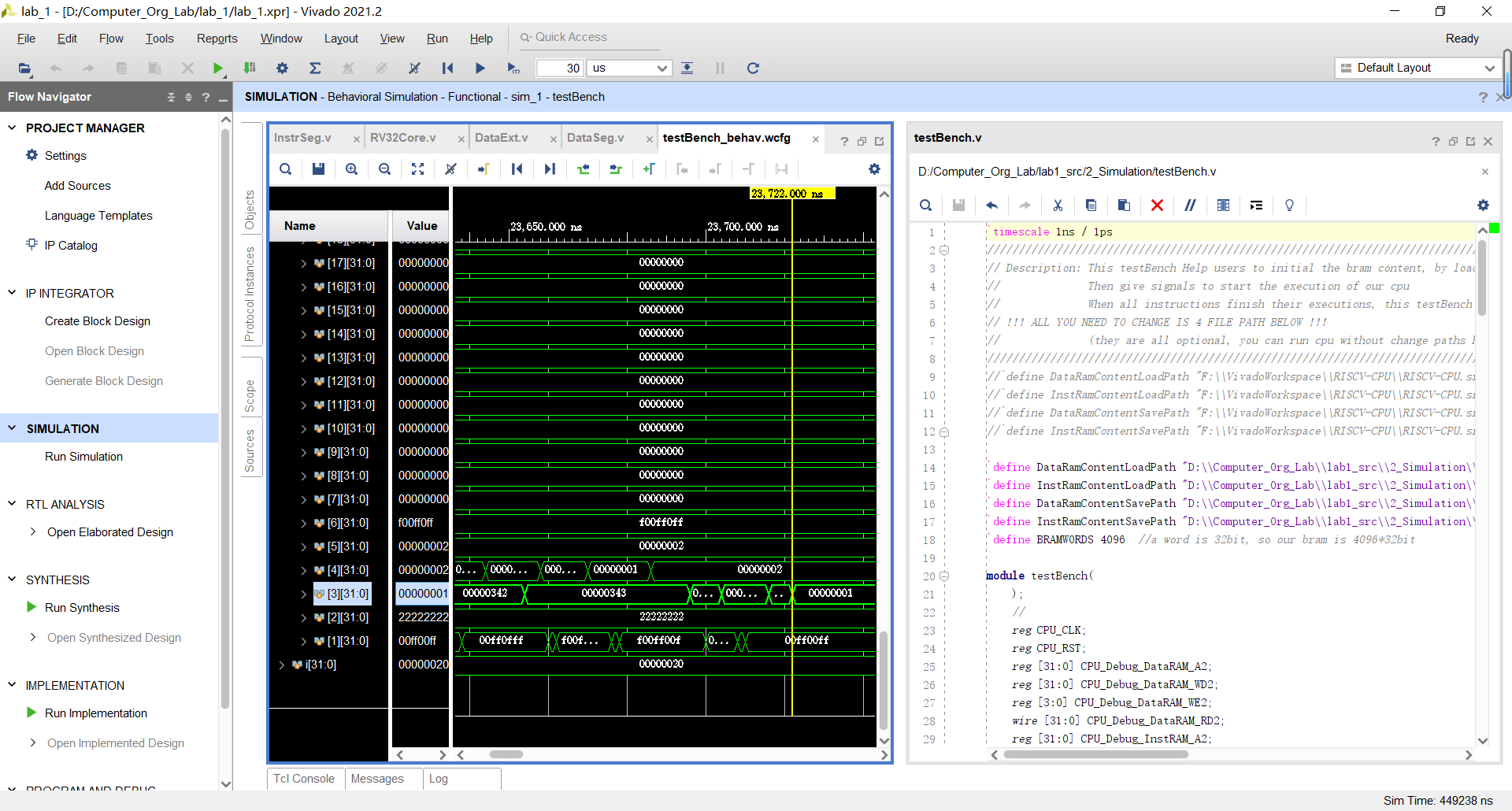
* + 根据三部分测试计测试CPU仿真，修改了一些错误之外，针对系统性的问题做了一些调整。

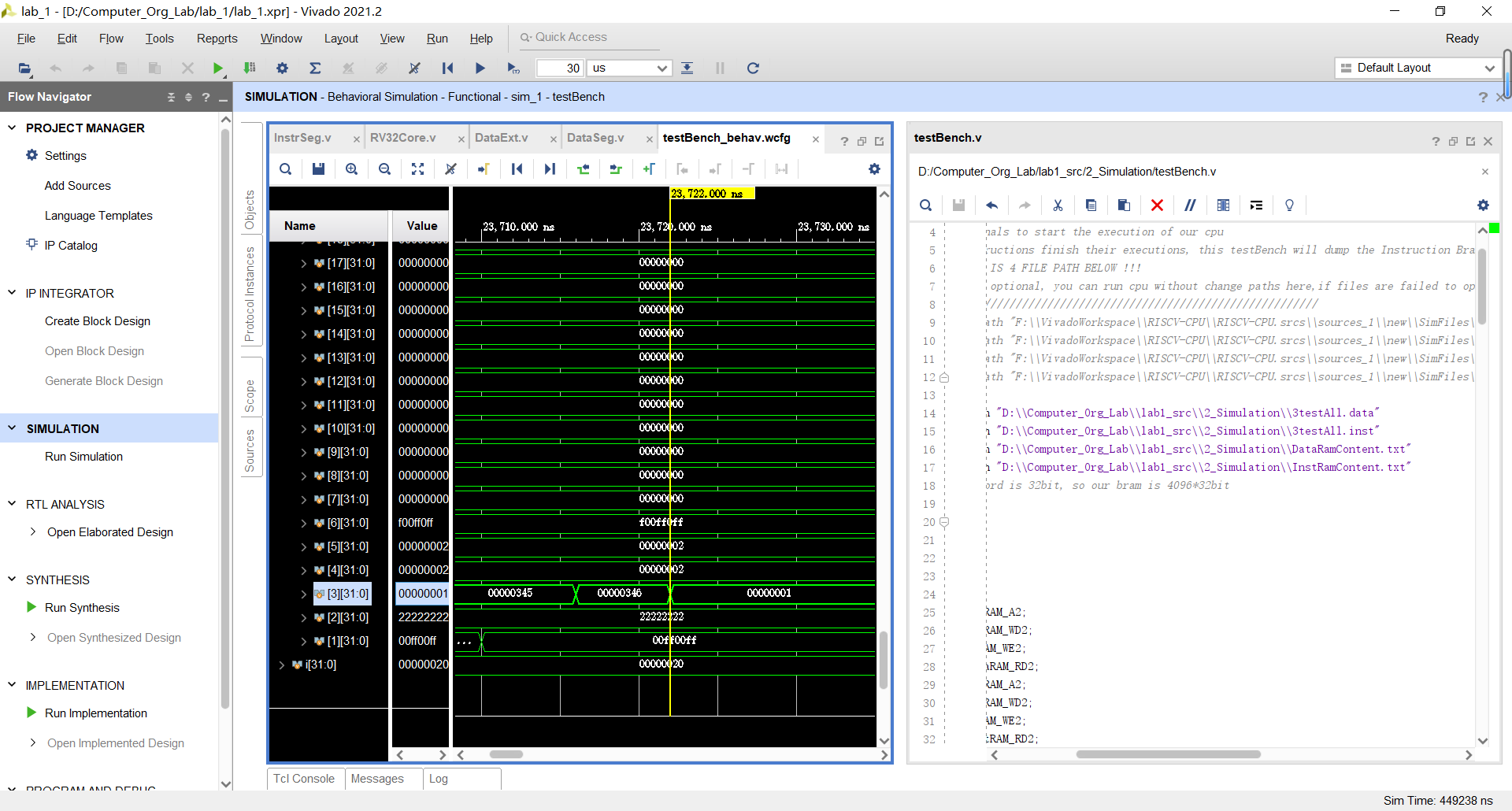
对PC的时序读取问题，由于InstrSeg和NPCGenerator模块都在同一个上升沿读取数据，所以原本编写的模块实际上存在结构冒险。

在测试阶段，我们完成了PC指令的正确更新，和不同store指令的字节对齐。最后顺利通过了前两段测试。随后直接通过了part3的测试，对测试集最后x3的值在累加之后返回1，说明所有测试样例都通过，且经过观察寄存器也符合预期值。









* 实验总结（说说踩的坑，总结收获，分析下自己花了多少时间，都用来做什么事情）

许珂钒：本次实验最大的挑战就是正确的实现跳转的指令，而这一部分的核心就是在PC部分要在跳转或者分支信号为真时，准确让下一次被读取的PC值为目标地址，在这个过程中我踩了两个坑，第一次我只输出了NPC的值而没有PC值，导致Instr指令永远比当前PC落后一位，而在想办法修改后，不管是增加一个PC模块寄存器还是同时输出PC和NPC，都会因为同步的上升延无法读取到正确的值。因此此处改变触发上升沿是较为合理的处理方式。

本次实验也收获了很多，通过对代码的补全和理解通读，已经基本掌握单周期CPU的底层运行原理，也在debug过程中复习了数字电路的基本运行原理比如时序电路的逻辑，为整个课程的基础知识和下一个lab关于流水线CPU的架构打下基础。

项奕玮：本次实验最大的挑战在于研究到底是哪里出了问题，当CPU没有正确运转时，根据变量的值进行错误推断是非常艰辛的工作。并且由于大三上没有使用vivado的课程，重新捡起vivado基础也是很大的难点。同时，实际实验的CPU架构和课上介绍的有不小的出入，这导致花了大量时间来做CPU框架的理解，也为Debug带来了不小的难度。

本次实验也收获了很多，主要是重新熟悉了vivado，和掌握了调试电路的技能，完成了错误的排查和修正。并且阅读part3的测试文件让我对如何做指令集的测试有了一个基本的概念。同时也对整个课程的基础知识和单周期CPU的底层运行原理有了更深的了解。

* 提出改进实验的意见

整体上，将单周期的注释说明和变量、模块定义进行适当改善尽量减少误导；实验中的无关变量过多。通过时延或者改变敏感列表等等手段模拟真实五个模块之间的延迟触发，而不是完全理想的仿真导致每个模块间出现结构冒险。

### 思考题：

请在报告中回答下述问题：

1. 为什么将DataMemory和InstructionMemory嵌入在段寄存器中？

*因为同步读 memory 相当于 异步读 memory 的输出外接D触发器，需要时钟上升沿才能读取数据。此时如果再通过段寄存器缓存，那么需要两个时钟上升沿才能将数据传递到下一阶段，因此在段寄存器模块中嵌入 memory，直接将输出传递到组合逻辑*

1. DataMemory和InstructionMemory输入地址是字（32bit）地址，如何将访存地址转化为字地址输入进去？

*因为地址间隔为8，最后两位必定为0，只取前30位（丢弃掉最后两位），虽然逻辑上应该是右移两位，但是实验里直接用了前30位。*

1. 如何实现DataMemory的非字对齐的Load？

*读 Data Memory 地址的低两位，按照地址后两位的值对读取到的32位字做右移，以确定实际读取的数据，然后根据所选的Load类型，对右移后的值做裁剪和符号扩展。在dataext中完成。*

1. 如何实现DataMemory的非字对齐的Store？

*取 ALUout的低两位（所计算地址模4的结果），按照地址后两位的值对要存储寄存器的32位字做右移，并根据所选的Store类型设置要改写的内存位置（传递修改过的MemWrite）。在dataseg中完成。*

1. 为什么RegFile的时钟要取反？

*这里实际上是为了解决在做流水线处理时的数据冒险，防止出现当一条指令的执行结果还没有写回寄存器，后面的指令就要使用前者目标寄存器的值。所以将RegFile的触发改为了下降沿触发。*

1. NPC\_Generator中对于不同跳转target的选择有没有优先级？

*有的，Jalr和Br的跳转地址在EX阶段被计算出来，Jal的地址在ID阶段被计算出来。*

*当有流水线时，相当于NPC模块收到的Jalr或Br在EX段，而Jal在ID段时，会先响应Jalr或Br，不再响应Jal。因为这相当于代码中Jalr或Br指令在Jal之前。*

*因此，优先级Jalr=Br>Jal*

1. ALU模块中，默认wire变量是有符号数还是无符号数？

*是无符号数*

1. AluSrc1E执行哪些指令时等于1’b1？

*在本实验中，只有执行AUIPC时会使ALuSrc1为1，其他为0，因为虽然分支和跳转命令也需要对PC进行加计算，但是在实际实现的框架中在顶层框架中直接与立即数相加。因此对真实的CPU，J，B-type的指令也应该经过ALU模块计算且取1.*

1. AluSrc2E执行哪些指令时等于2‘b01？

*执行SLLI、SRLI、SRAI时AluSrc2 == 2‘b01,因为这些立即数移位指令指令只使用立即值的低5位作为移位量(只能移位0-31位位置)，这五位的shamt码相当于R-type指令的rs2寄存器编号所在的位置，因此AluSrc2置为2‘b1选择shamt码的位宽为5的位置*

1. 哪条指令执行过程中会使得LoadNpcD==1？

*Jalr和Jal会使Load NPC==1。*

1. DataExt模块中，LoadedBytesSelect的意义是什么？

*LoadedBytesSelec等价于AluOutM[1:0]，是读Data Memory地址的低两位，因为DataMemory是按字（32bit）进行访问的，所以需要把字节地址转化为字地址传给DataMem。 DataMem一次返回一个字，低两位地址用来从32bit字中，定位LB、LH在这个字中的位置，本实验中通过移位的方式，将所需（双）字节移位到最低位，从而挑选出我们需要的字节。*

* 分数分配

两人工作量基本相近，平均分配