# Lab2-RV31I实验指导和实验报告要求

* 用verilog实现RV32I 四级流水线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1\_PipelineLab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：参考我们提供的CPU模块图，修改补全模块内部逻辑完成CPU设计；也可以根据你的个人理解进行设计，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
5. **此时不需要处理数据相关。可以令Harzard模块始终输出stall、flush恒为0，forward恒为不转发，每两条指令之间间隔四条空指令。**

**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
5. **此时不需要处理数据相关。可以令Harzard模块始终输出stall、flush恒为0，forward恒为不转发，每两条指令之间间隔四条空指令。**

**PART3：**

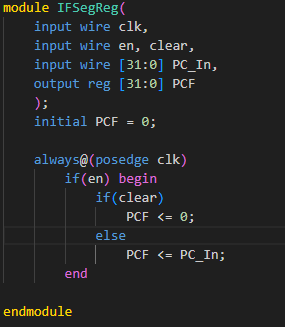
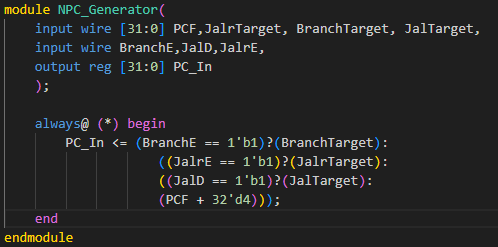
1. 利用我们提供的修改过得RISCV test官方测试汇编代码，通过提供的工具生成.inst和.data文件，用于初始化指令和数据的Block Memory
2. CPU开始执行后3号寄存器的值会从2一直累增，该数字正在进行第多少项测试，执行结束后3号寄存器值变为1
3. CPU执行后，各寄存器值符合预期
4. **此时需要处理数据相关，实现Harzard模块内部逻辑。**

注意：三部分为递进关系，建议大家按照顺序进行测试

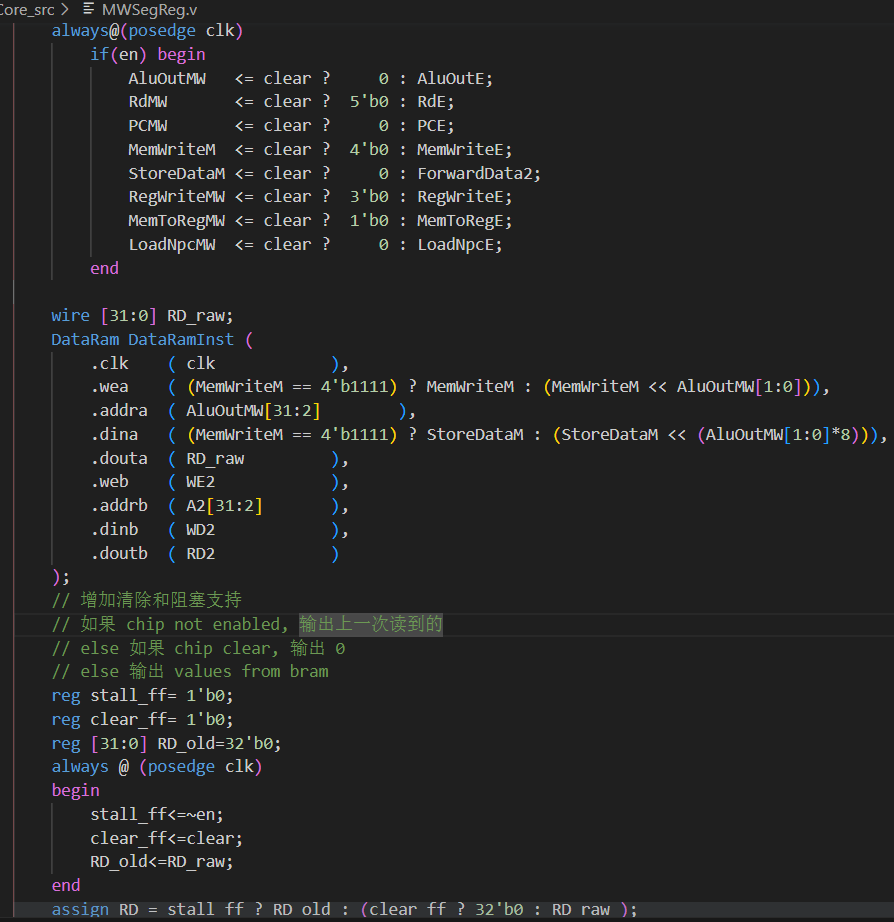
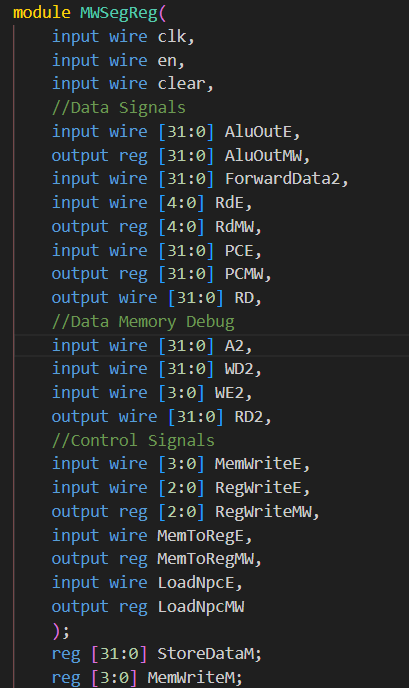
### 实验报告：

* 实验目标：用verilog实现RV32I 四级流水线CPU
* 实验环境和工具：Vivado
* 实验内容和过程（总结自己所做的三个阶段工作）：

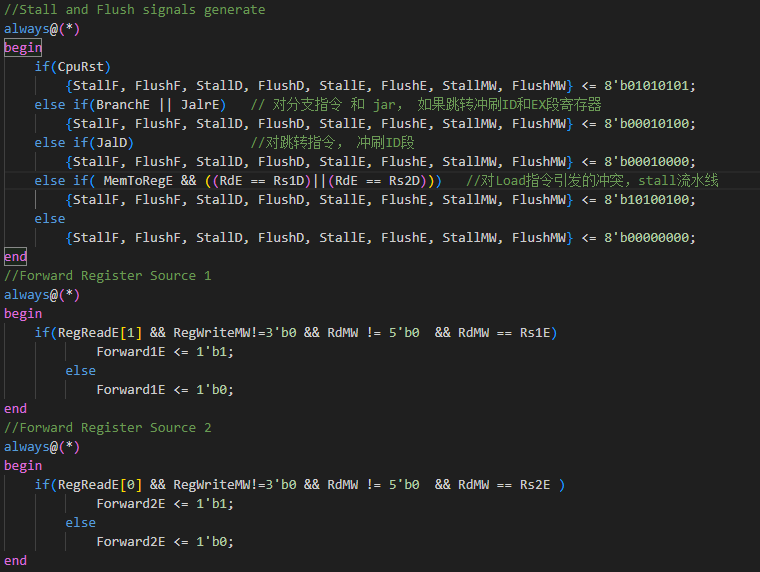
1. 根据单周期CPU，完成各个阶段的运算相关指令模块，这些模块不太需要针对流水线做太多的修改，这里不做过多赘述，比较大的变化分别是ImmOperandUnit即根据指令值生成立即数操作数的模块成为独立模块，但是与lab1本质上相同；以及NPCGenerator和IFSegReg，在lab1中，因为要一个周期内同时生成PCF和PCNext的值并且读取正确值，并且在一个模块内完成，实现上较为困难且不变，当时只能采用negedge的方式让pc相关运算提前半周期，而在流水线CPU设计中，有了寄存器并且共四个周期一条指令的执行时间使得这些值能够得到正确的储存，这些模块就能比较好的同步在同一个clk上升沿运算：



1. 主要是做好MWRegReg模块的代码编写，这部分耗时较多。其中主要要完成寄存器内容的转发和基于DataRam实现store指令，store的实现和lab1中的DataSeg基本一致，区别只在于这次由于是在一个模块中完成。因此先去理解了设计图中五级流水线的设计，MWSegSeg就是对M和W部分的合并，但是要注意其中数据所在的时钟周期。
   1. 首先要做好数据的转发，因此仿照EXSegReg编写了寄存器转发的部分（用？操作符简化了代码）。
   2. 其次基于DataRam做store指令的支持，主要仿照lab1的DataSeg部分。要特别注意其中DataRam应该使用的是转发后的信号，否则使用的信号会超前一个周期。这里发现原本的A就对应输出的寄存器AluOutMW，但WE、WD应该对应ForwardData2和MemWriteE，可是这两个是输入信号，周期不对。因此在MW中增设了StoreDataM和MemWriteM（对应五级流水线设计图中的变量）两个寄存器，在数据的转发中同时也做了ForwardData2和MemWriteE信号的存储以实现周期对齐。然后将A、WE、WD信号替换成AluOutMW、MemWriteM、StoreDataM即可。
   3. 最后实现了clear和stall的支持，处理了RD的输出。

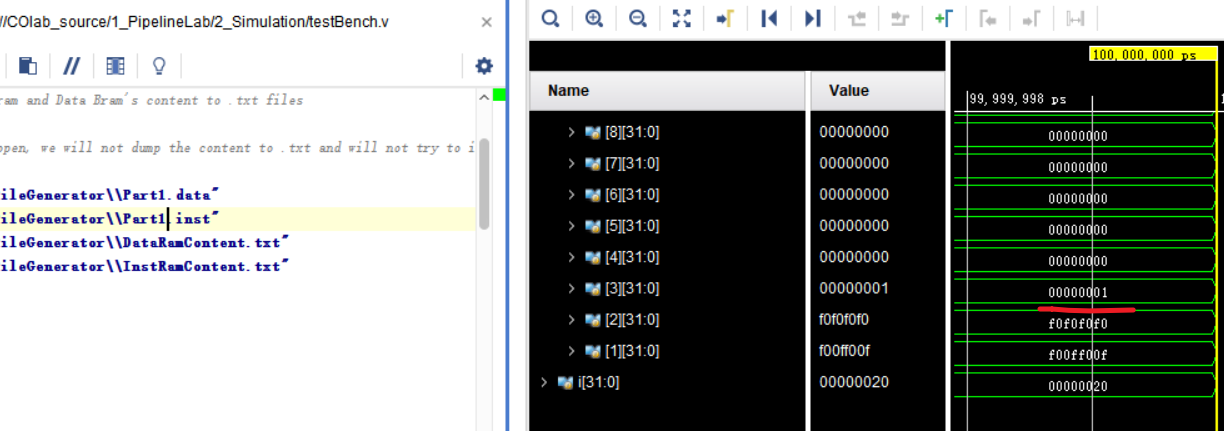


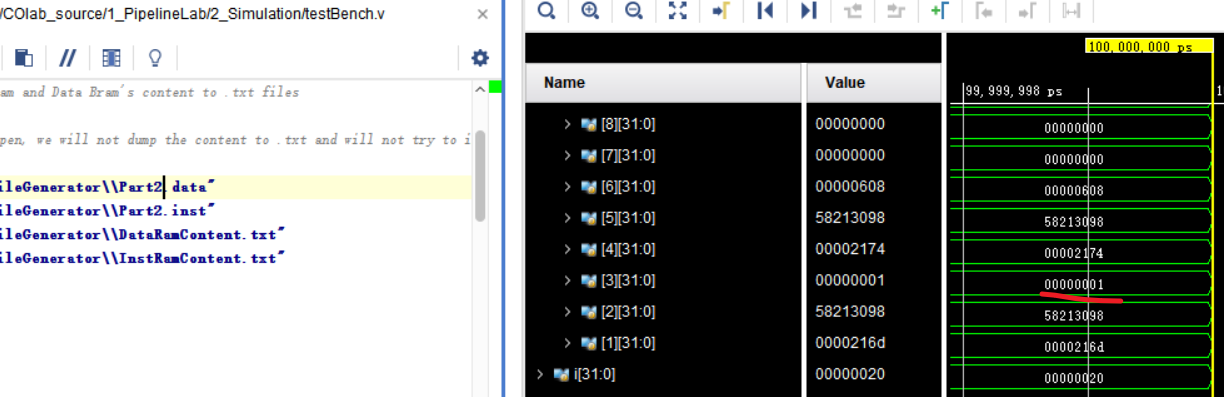
1. 然后完成HazardUnit模块，作为处理流水线各种冒险的核心模块，根据提示完成代码如下：

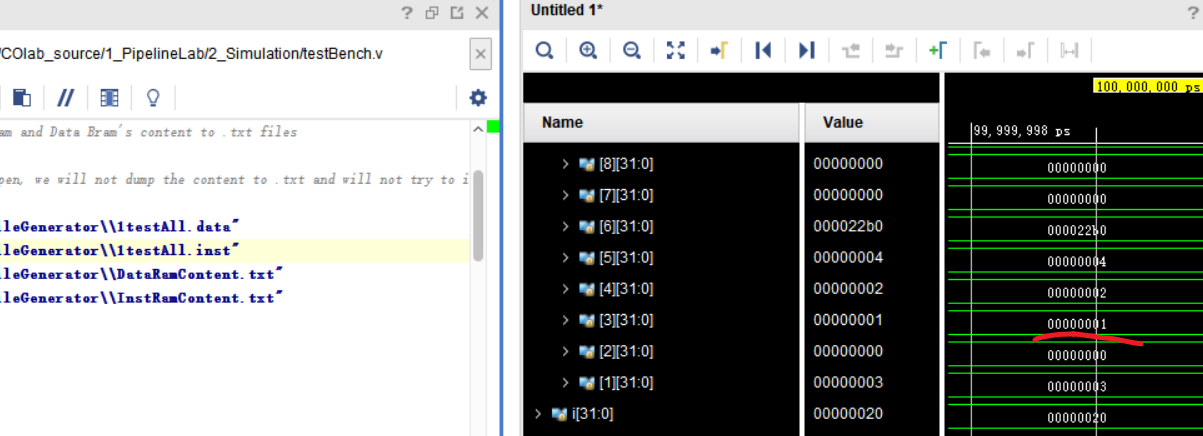


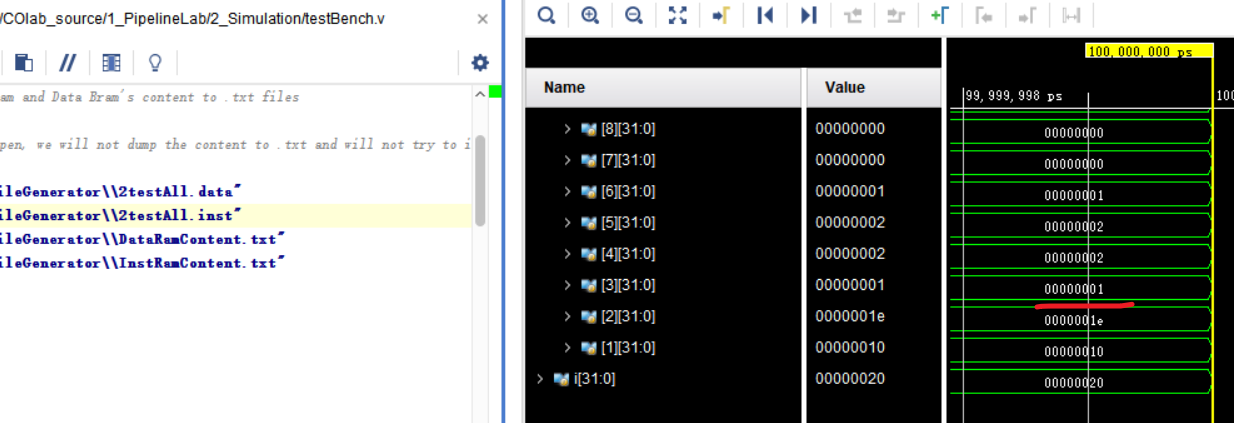
1. 测试部分：

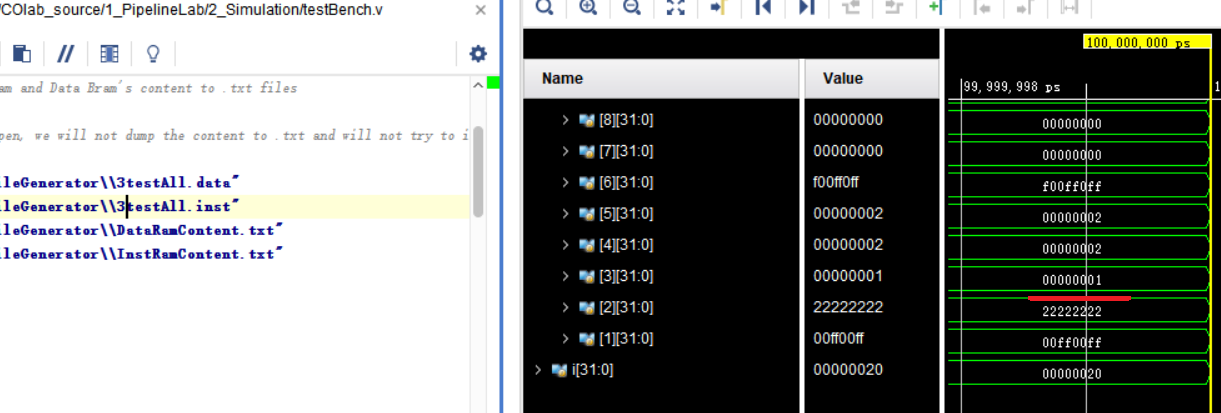
由于分工，我们直接完成了HU模块（很幸运，它测试良好），因此就没有必要再做空指令的插入，可以直接沿用lab1的测试数据做测试，一开始测试的时候卡在了store指令出错，后来发现了是MW模块中DataRam的指令周期不一，没有对齐，在加入两个转发寄存器后顺利解决。接下来展示P1，P2，1testall，2testall，3testall的最终测试结果：











* 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：因为实验大部分内容都只需要在lab1的基础上稍作修改和增加，总体上的代码量不是特别多。在debug过程中，踩得坑主要是在s指令相关的方面，因为MW指令是不同于五阶流水线的最后两个阶段有两组寄存器，导致MW模块指令内部其实存在addra和写入数据由于都是从ALUout接入但是由于没有考虑好寄存器的时延问题，又产生了和第一个lab中类似的某个寄存器取值慢了一个clk。大部分时间除了花在debug分析每个模块分析的值之外，也对课程上理论上的hazard与实际情况下hazard的实现进行了复习和进一步的理解，发现了它们本质上的一致性。本次实验在实际实现上体会了流水线的巧妙思想与底层实现，对课程内容有了更深的理解。

项奕玮：这次实验做起来感觉确实比lab1轻松不少，很多模块lab1已经写好了，不需要多加调试。主要集中在MW和HU两个核心模块上，我们也做了解耦，我们主要碰到的问题就是MW模块由于合并导致的数据所在周期没有对齐的问题。在碰到问题后我们主要是先去重新理解了一下五级流水线的设计图，并基于此重新思考合并时的数据情况。最后确定给MW中增加两个寄存器以实现其中DataRam输入数据的周期对齐，解决了store指令的报错问题。同时对hazard的实现进行了复习和进一步的理解。本次实验在实际实现上体会了流水线的巧妙思想与底层实现，对课程内容有了更深的理解。体会到了流水线的加速作用。

* 提出改进实验的意见：

在实验设计上，经常需要debug修改一些已搭建完成的框架内的错误代码，扩大了纠错范围，延长时间；同时，需要补全的代码模块中，也存在多余输入输出端口，注释里也存在提示不正确，模糊等小问题。

**实验总结是每位同学写自己的，其他部分同一个小组写一份，合并在一个报告里面即可，最后整个小组上交一份报告，报告最后写上分数分配。**

### 思考题：

请在报告中回答下述问题：

1. Harzard模块中，有哪几类冲突需要插入气泡？

L指令：L指令后的L，R，S指令存在数据相关，插入一个气泡。

跳转指令：分支插入两个气泡，跳转插入一个气泡。

1. Harzard模块中采用默认不跳转的策略，遇到branch指令时，如何控制flush和stall信号？

Branch指令在EX段如果发生跳转，则Flush IF/ID和ID/EX段寄存器。否则默认不flush & stall。

1. Harzard模块中，RegReadE信号有什么用？

RegReadE用来表示当前E阶段的指令中的Rs1、Rs2,是否有在之前的d阶段作为Rd出现过，如果有的话，可以采用数据前递消除数据冒险： RegReadE[1]==1 表示前一阶段Rd对应的寄存器值被当前阶段Rs1使用到了，RegReadE[0]==1表示Rd对应的寄存器值被Rs2使用到了，用于forward的处理

1. 0号寄存器值始终为0，是否会对forward的处理产生影响？

有可能产生影响，比如在通过将Rd寄存器写为x0而用作丢弃数据时，如果前递了AluOut则会得出错误的值，应当在转发时对源寄存器是0的情况进行特殊判断：如果某条指令写的是x0寄存器则不转发或转发0值。

### 分数分配：

HU、MW模块分别由许珂钒、项奕玮完成，其余部分二人共同修改调试。工作量视作均等。