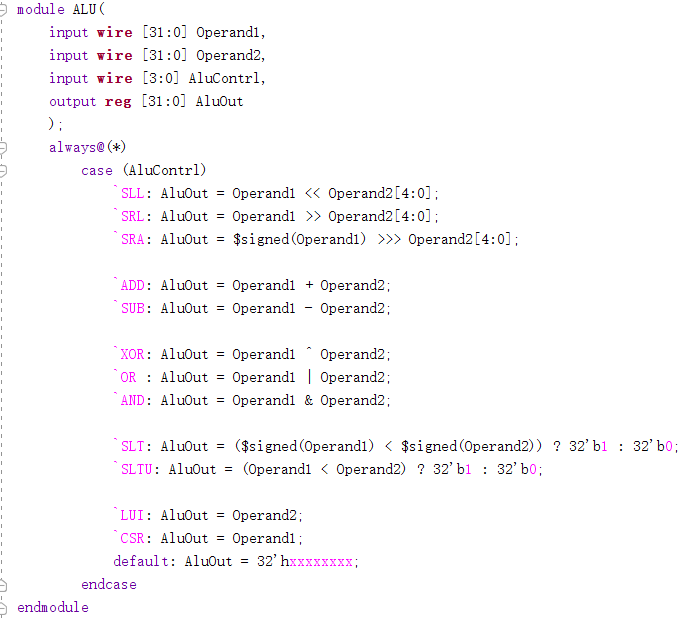
计算机体系结构实验报告LAB02

题目： RISC-V 32I 的实现

姓名：魏钊  
学号： PB18111699

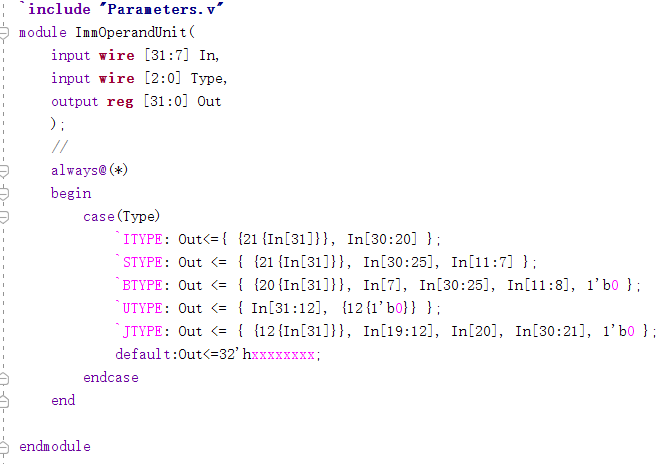
实验目的：  
设计一个 RISC-V 32I 指令集的五段流水线 CPU，实现转发、冒险检测和CSR指令。实验环境和工具：  
Vivado 实验设计与过程：

**ALU模块：**

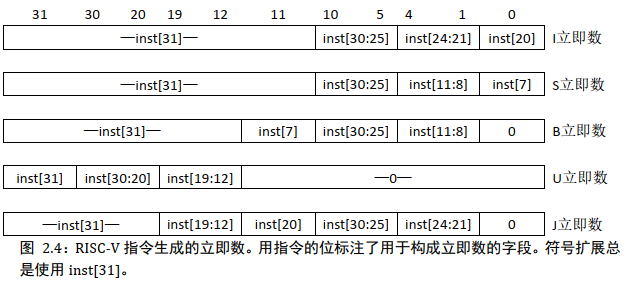


ALU默认将所有操作数视为无符号数，但实际上部分运算是有符号数，这里我们使用$signed函数将其转换为有符号数。CSR指令只需将操作数1输出即可。

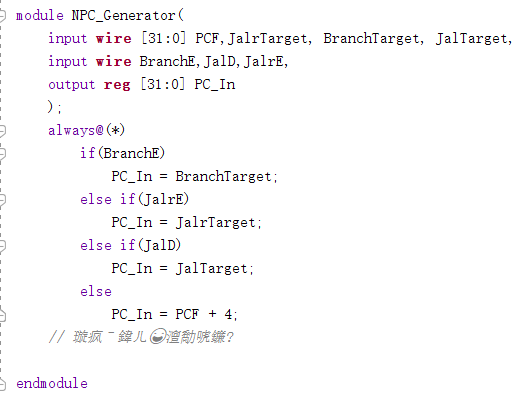
**立即数模块：**



**按照手册中格式拼接即可：**

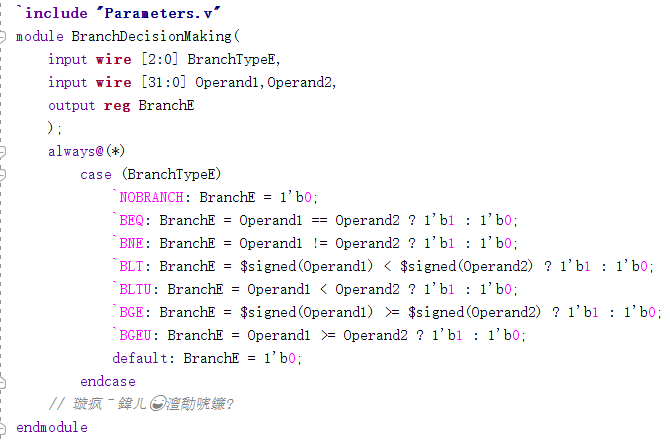


**NPC模块：**



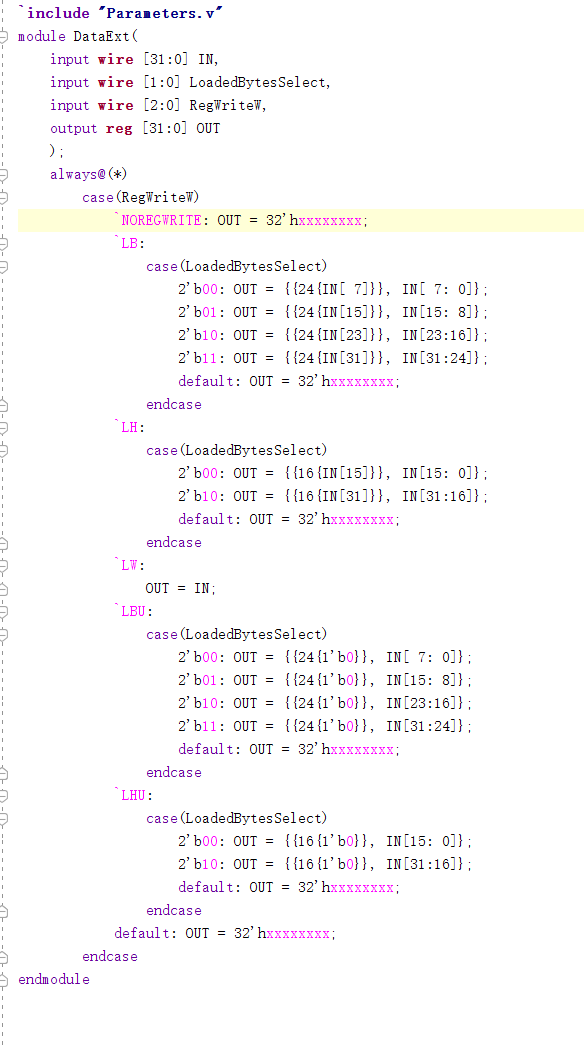
因为Branch和Jarl指令在EX段跳转，而Jal指令在ID段跳转，所以前两者优先度大于后者。

**跳转决定单元：**



使用上文中提到的$signed函数完成涉及有符号数的运算。

**DataExt模块：**



这个模块处理 Load 类型指令及访存地址非字对齐的情况。

由于 Data Memory 是按字访问的，内存地址的低 2 位未使用，要在访存后根据寄存器写入模式（已定义在 Parameters.v 中）及低 2 位地址进行进一步的选择：

对于 LW 模式，无需处理， OUT = IN.

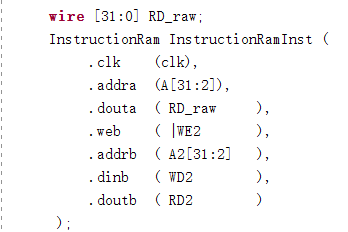
对于 LH 模式，可以按半字读取， 低 2 位地址可能是 00 或者 10，则从访存结果截取低 2 字节或者高 2 字节作为寄存器写入值的低 2 字节，高 2 字节进行符号拓展.

对于 LB 模式，可以按字节读取，低 2 位地址可能是 00/01/10/11，则从访存结果截取第 0/1/2/3 字节作为寄存器写入值的第 0 字节，高 3 字节进行符号拓展.

对于 LHU 模式，与 LH 模式类似，只是高 2 字节位无符号拓展

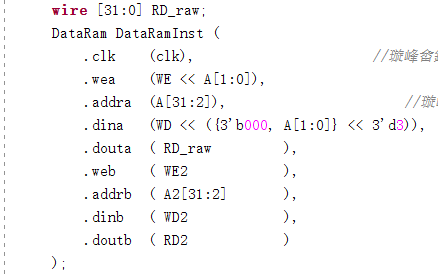
对于 LBU 模式，与 LB 模式类似，只是高 3 字节位无符号拓展

**IDSegReg模块：**



和时钟上升沿同步。另外，指令存储器的读地址应该传入 A[31:2]，按字节读取。

**WBSegReg模块：**



SW、 SH、 SB 指令分别将从 rs2 寄存器低位开始的 32 位、 16 位、 8 位数值保存到存储器中，则 ControlUnit 中 MemWriteD 信号分别置为 4’b1111、 4’b0011、4’b0001（让存储器按字节写入， 1bit 对应写入一字节）。

在例化 DataMem 模块时，并不能将 WB 段寄存器传入的 WE 和 WD 信号直接接入，因为写入地址的低 2 位不一定是 2’b00，应该需要进行移位，这样就可以实现非字对齐的写入。

设地址低 2 位为 A[1:0]，则写使能应该接入 WE << A[1:0]，写数据应为WD << ({3'b000, A[1:0]} << 3'd3)。

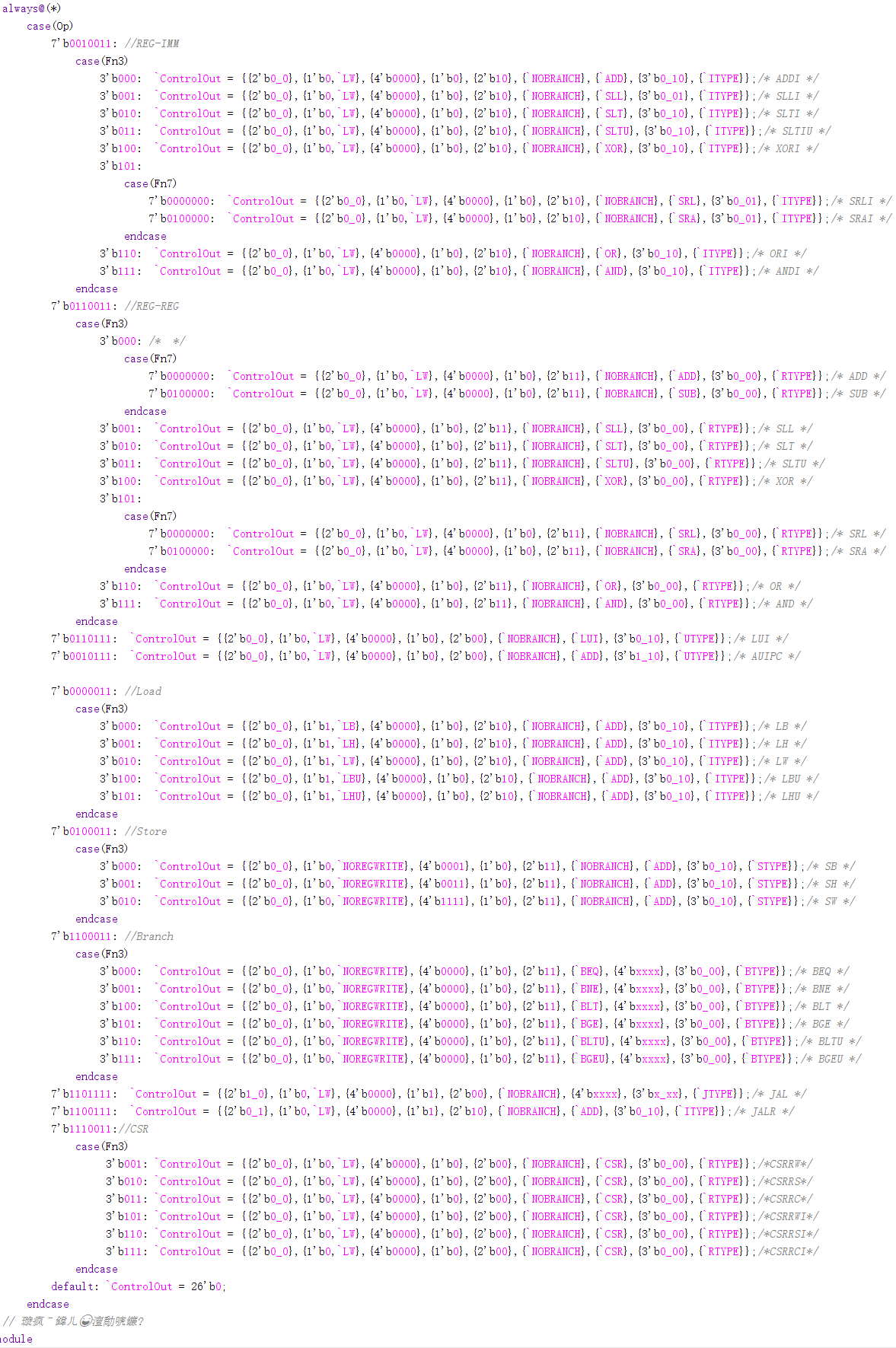
**控制单元模块：**

首先定义：

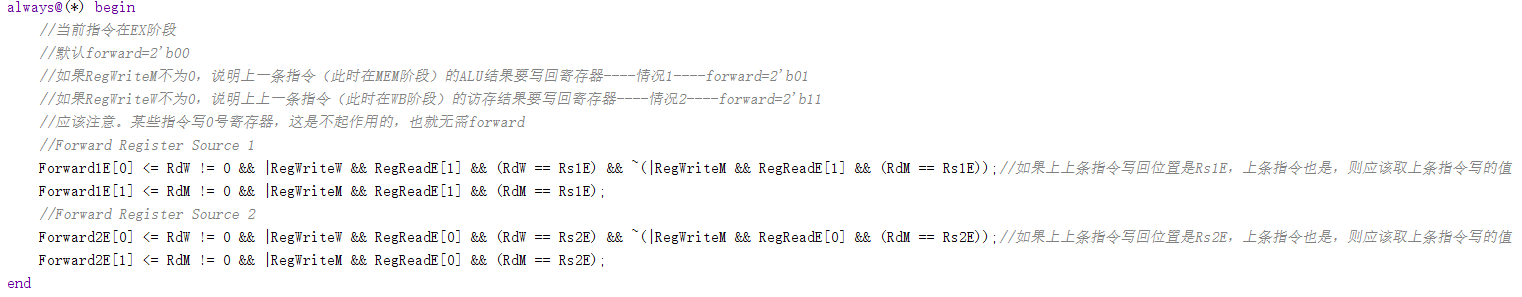
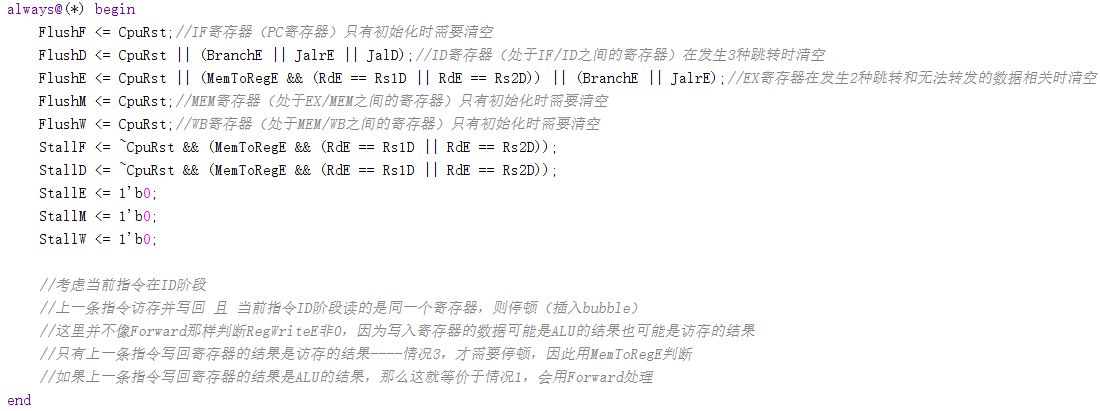


方便后续对信号的操作。

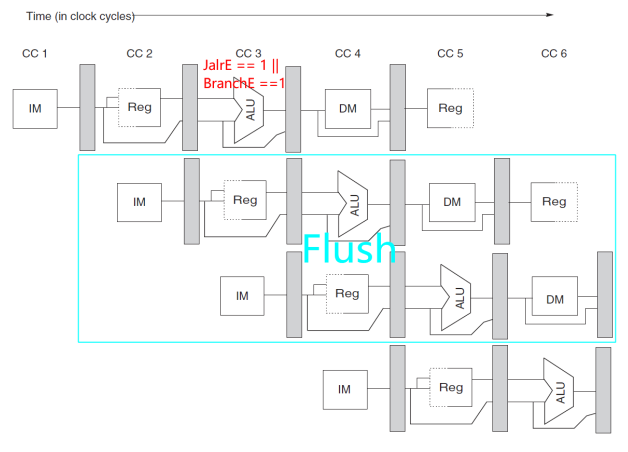
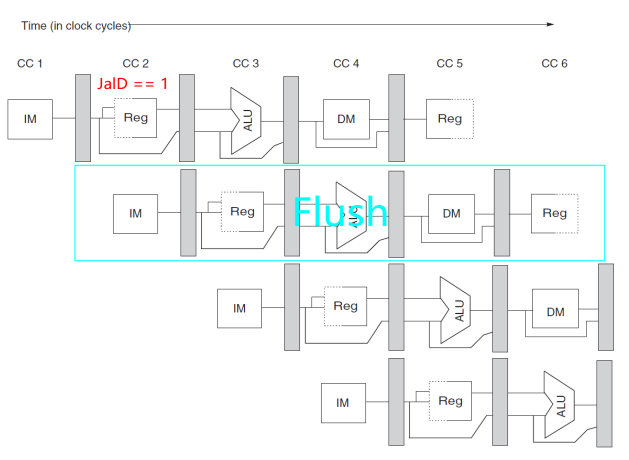
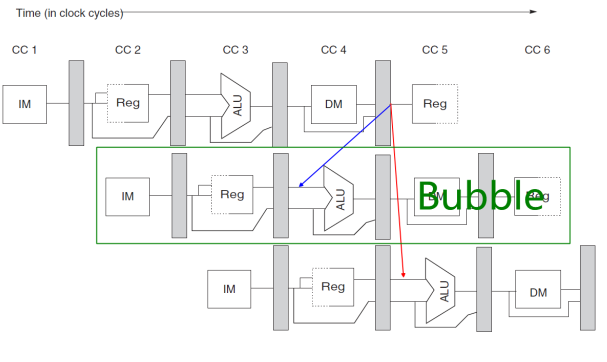
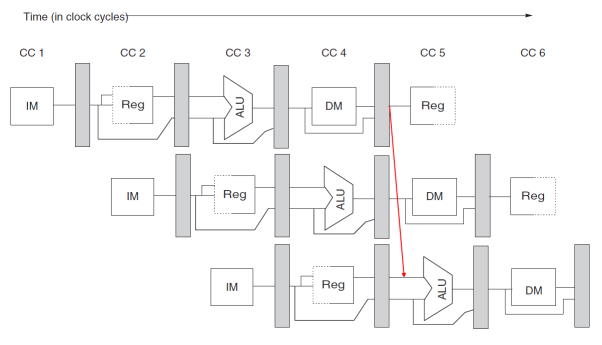
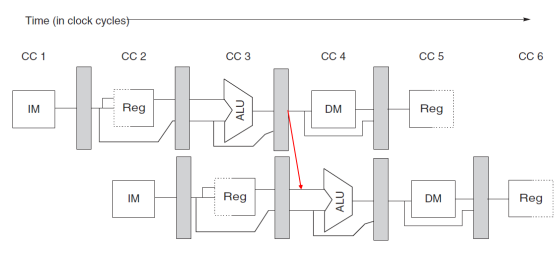
然后根据相应的Op和Fn3设置对应信号即可。



**冒险模块：**



由于某些指令会对 0 号寄存器进行写入（比如不需要 JAL 的链接结果时，可以选择 0 号寄存器作为目标寄存器），但是实际是不会进行写入的。这时如果下一条指令用到了 0 号寄存器作为常量，转发会导致当前指令写入的无用数据被转发给下一条指令。 因此，再默认情况下或者要使用的寄存器是 0 号寄存器，Forward 为 2'b00，表示不使用转发。

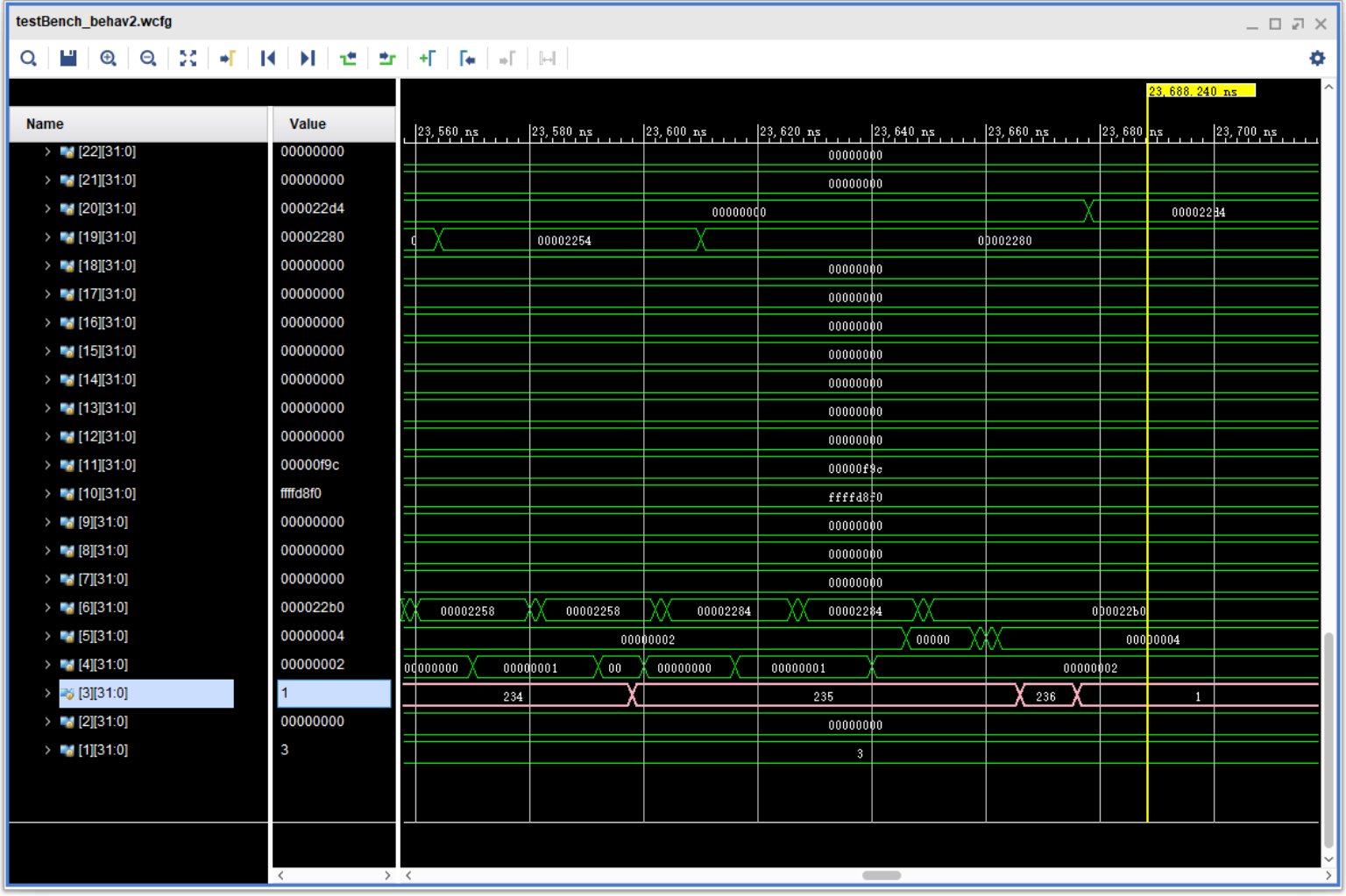


以上为需要转发和Stall以及Flush的情况。

**前两阶段的实验结果：**

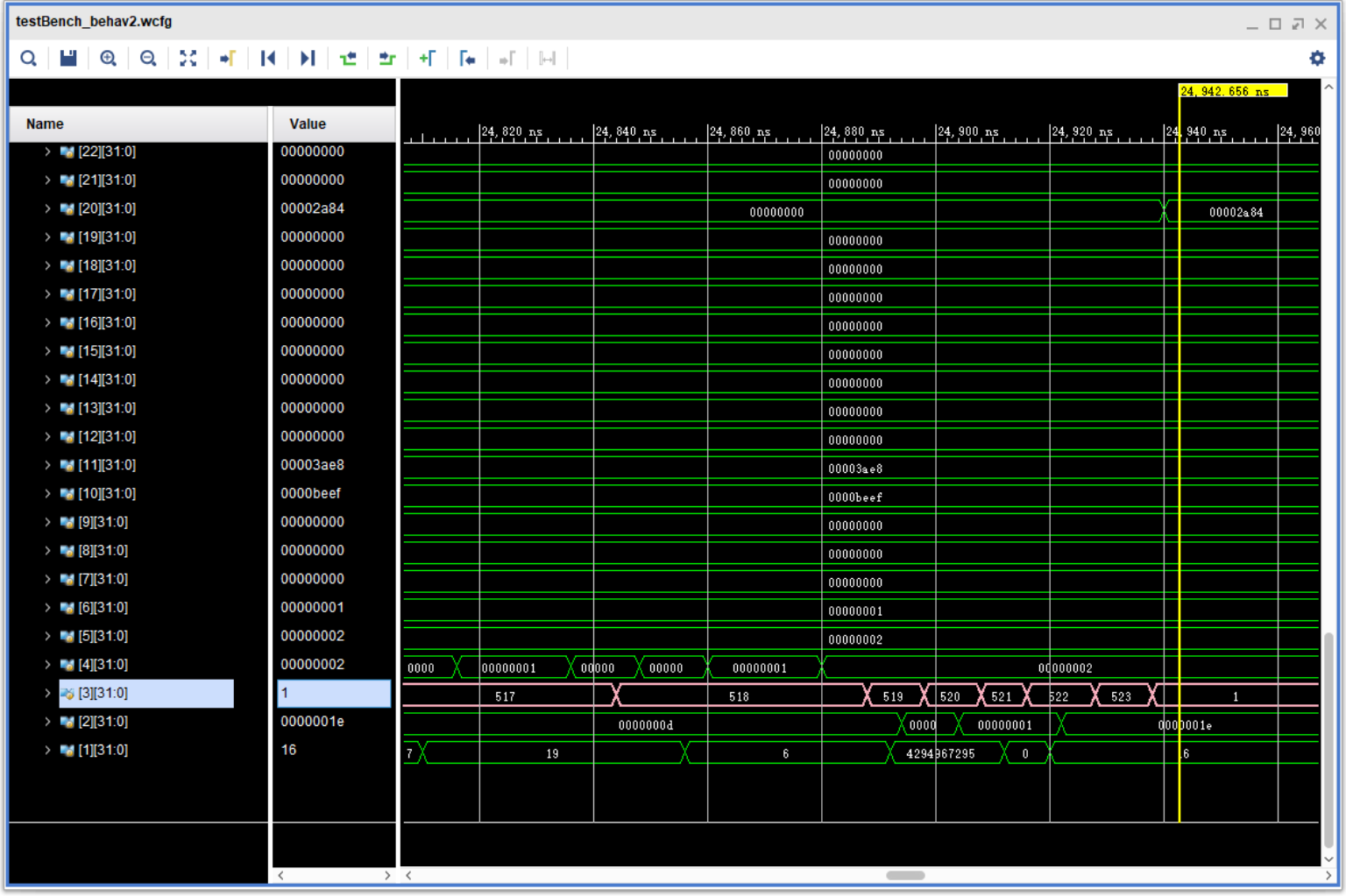
1testAll.inst

仿真结果， 236 号测试后 3 号寄存器的值变为 1， 且不再改变



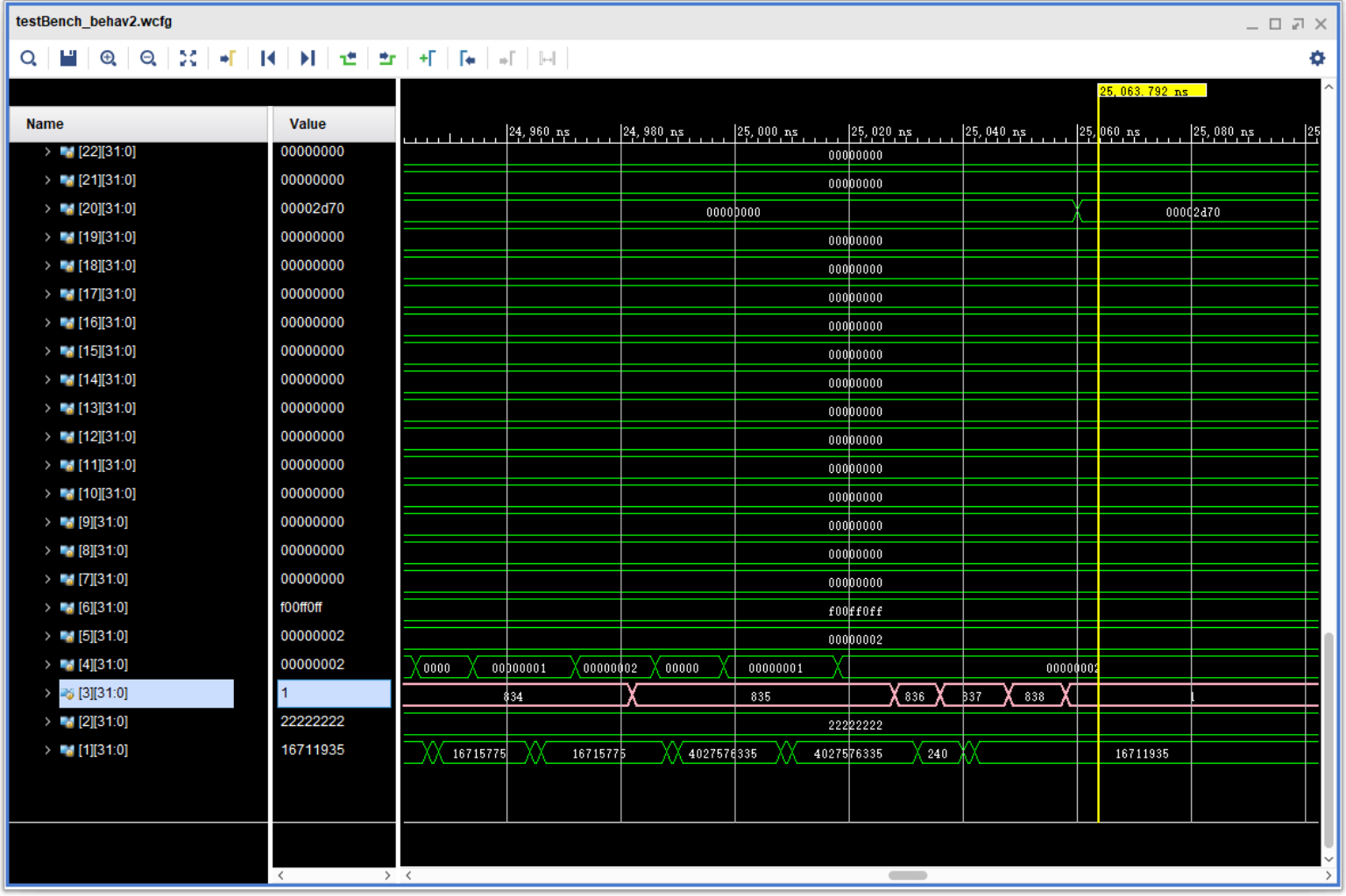
2testAll.inst

仿真结果， 523 号测试后 3 号寄存器的值变为 1， 且不再改变。

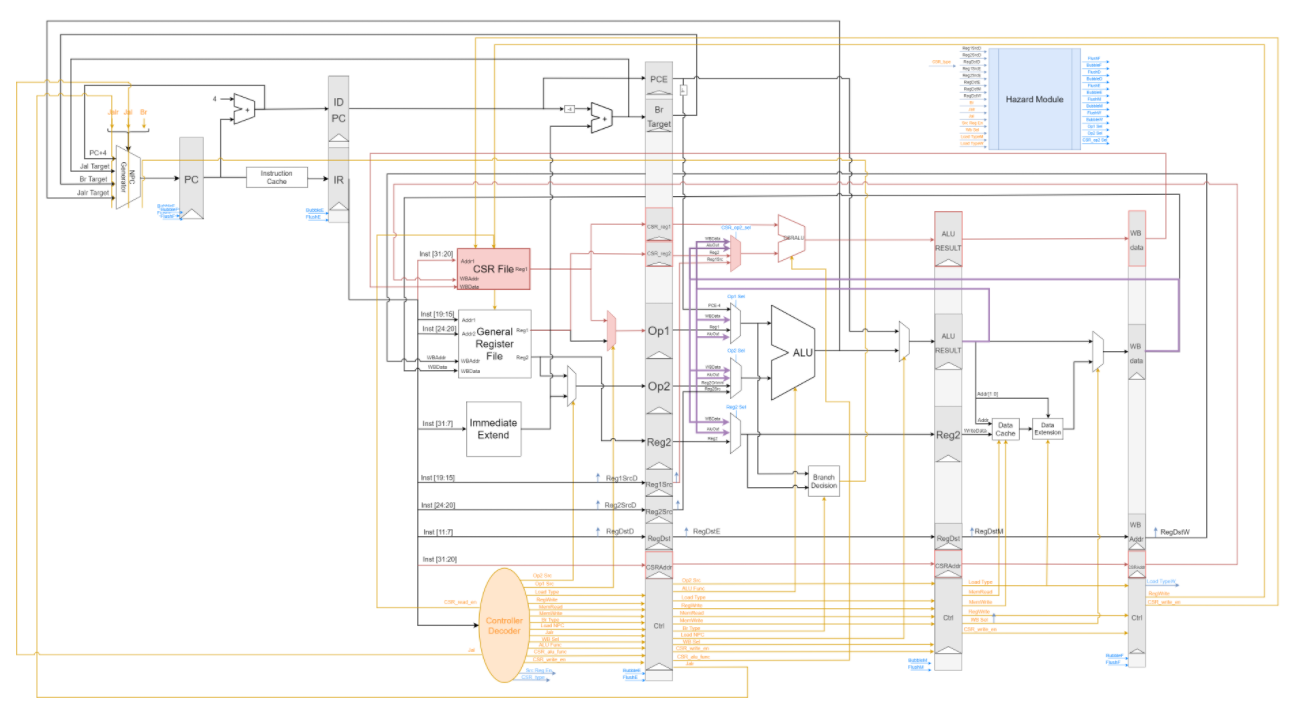


3testAll.inst

仿真结果， 838 号测试后 3 号寄存器的值变为 1， 且不再改变。

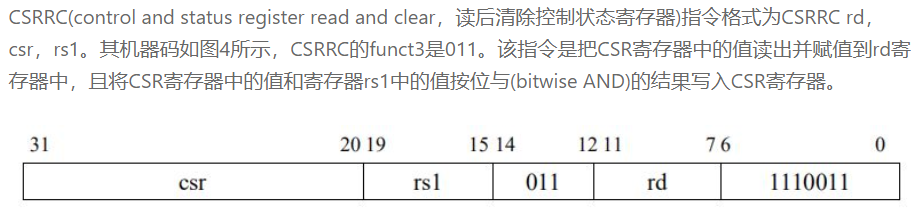
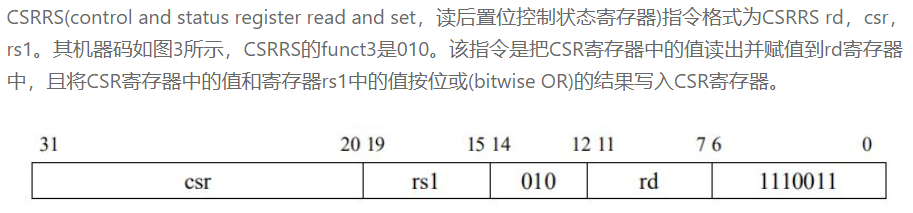


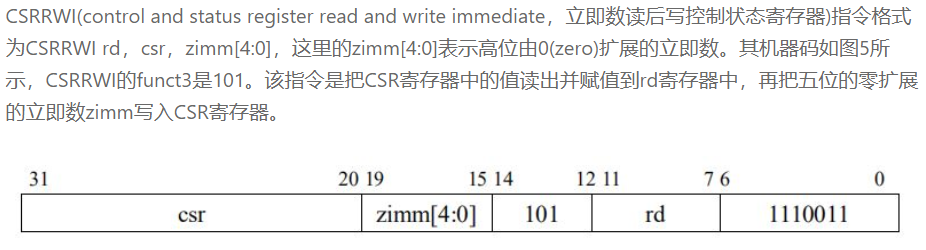
**阶段三：**

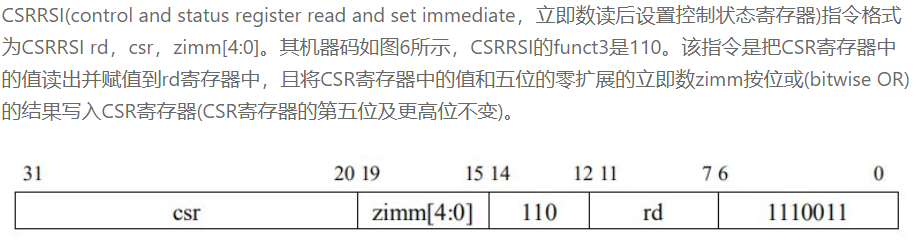


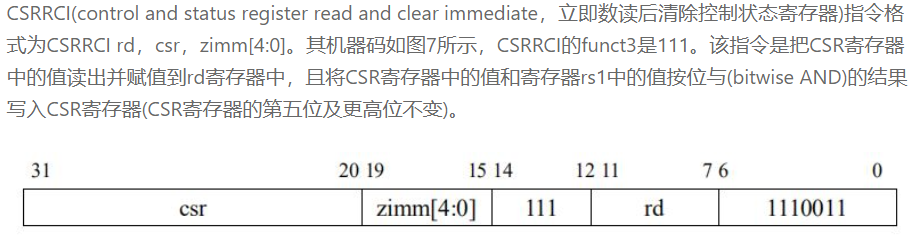
**主要增加了CSR寄存器，CSR\_ALU以及各个段间寄存器的修改。**

**首先介绍各个CSR指令，及其作用：**

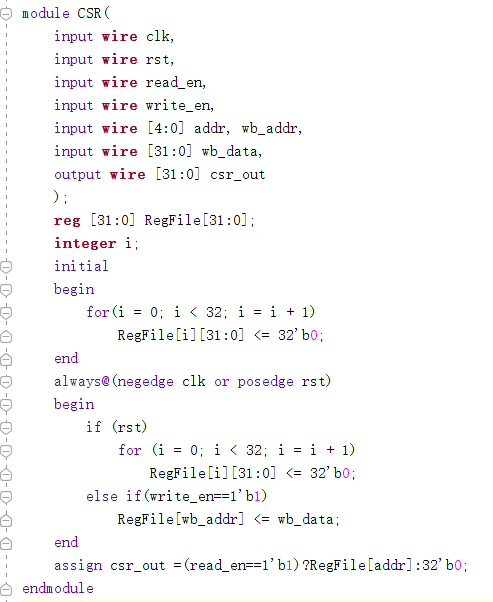
![C:\Users\Lucifer.dark\Documents\Tencent Files\709990727\Image\C2C\Image1\1HSM~%ZPZ[I2](5$3G`~15P.png](data:image/png;base64,)





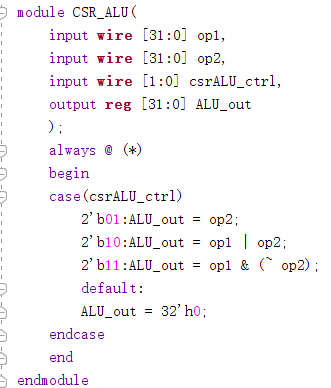


**CSR寄存器模块：**



具体设计类似通用寄存器，不再赘述。

**CSR\_ALU:（在参数模块增加`define CSR 4'd11）**



**RV32Core增加相应的信号、及修改对应的信号：**

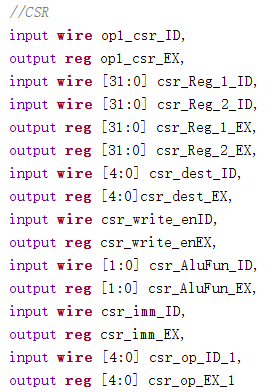


特别注意对通用寄存器中0号寄存器的写一律是无效的。

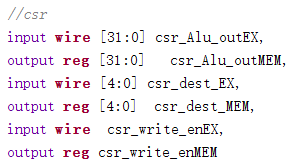
**一般ALU的修改：**



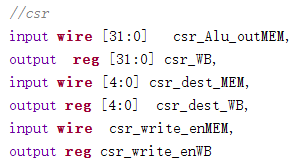
**ExSegReg增加以下相应信号：**



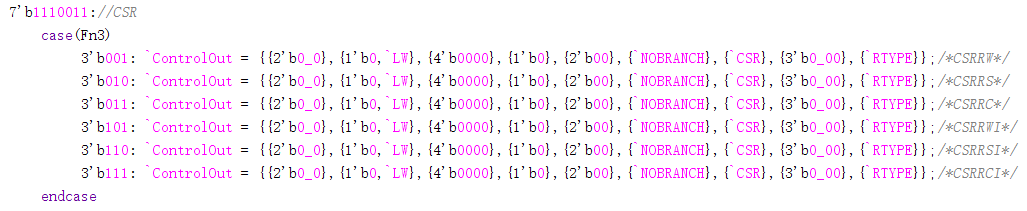
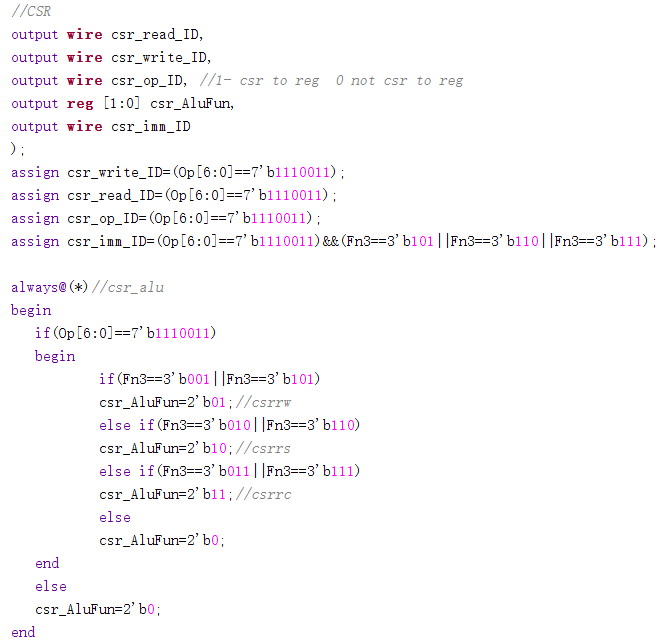
**同理，MemSegReg也增加相应信号：**



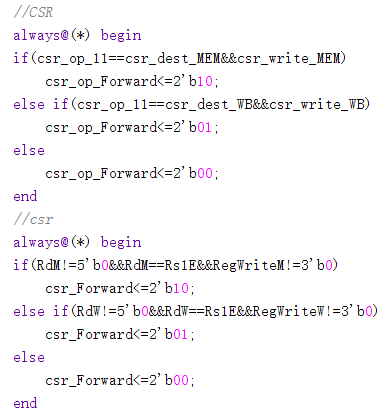
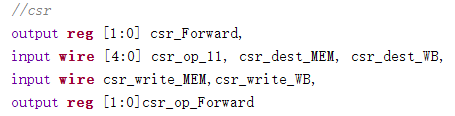
**WBSegReg增加：**



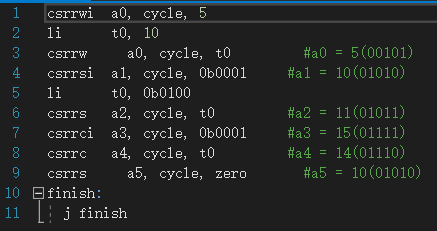
**控制单元：**



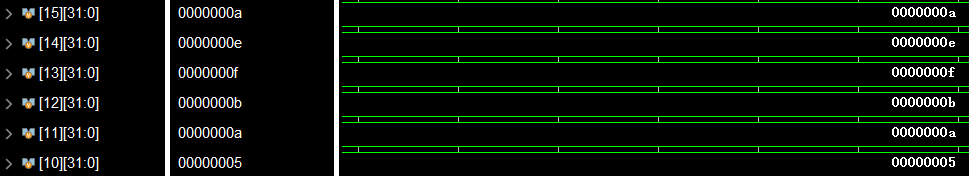
**冒险单元：**



**实验结果：**



**a0到a5寄存器对应10号到15号通用寄存器，他们最终的值应该对应上图：**



**实验成功。**

**实验总结：**

在有了助教提供的框架的前提下，实现所有的数据通路其实不困难。

因为COD中已经实现过流水线，所以第一阶段和第二阶段比较顺利。

困难主要集中在第三阶段CSR指令上，因为不熟悉CSR，前期花了大量时间学习CSR指令以及构思数据通路。

**改进意见：**

希望在实验前对CSR进行讲解，并提供相应的测试代码。