**计算机系统结构实验实验报告 Lab06**

姓名：叶增渝

学号：519030910168

日期：2021年05月03日

**目录**

[1.实验概述 3](#_Toc71018027)

[1.1 实验名称 3](#_Toc71018028)

[1.2 实验目的 3](#_Toc71018029)

[2.实验描述 3](#_Toc71018030)

[2.1 流水线处理器的寄存器设计 3](#_Toc71018031)

[2.1.1 PC寄存器 4](#_Toc71018032)

[2.1.2 IF/ID段寄存器 4](#_Toc71018033)

[2.1.3 ID/EX段寄存器 4](#_Toc71018034)

[2.1.4 EX/MEM段寄存器 5](#_Toc71018035)

[2.2 流水线处理器的各阶段设计 6](#_Toc71018036)

[2.2.1 IF阶段实现 7](#_Toc71018037)

[2.2.2 ID阶段实现 7](#_Toc71018038)

[2.2.3 EX阶段实现 9](#_Toc71018039)

[2.2.4 MEM阶段实现 9](#_Toc71018040)

[2.2.5 WB阶段实现 10](#_Toc71018041)

[2.3 流水线处理器设计改进 10](#_Toc71018042)

[2.3.1 Stall机制实现 10](#_Toc71018043)

[2.3.2 Forward机制实现 11](#_Toc71018044)

[2.3.3 predict-not-taken策略实现 11](#_Toc71018045)

[2.4 流水线处理器的整体模块 12](#_Toc71018046)

[2.4.1 Top模块源文件 12](#_Toc71018047)

[2.4.2 Top模块仿真激励 17](#_Toc71018048)

[2.3.6 外部文件 18](#_Toc71018049)

[2.3.7 仿真波形 21](#_Toc71018050)

[2.3.8 实验结论 22](#_Toc71018051)

[3.实验心得与总结 22](#_Toc71018052)

[4.参考资料 23](#_Toc71018053)

1.实验概述

# 1.1 实验名称

简单的类 MIPS 多周期流水线处理器设计与实现

# 1.2 实验目的

1．理解CPU Pipeline，了解流水线冒险(hazard)及相关性，设计基础流水线CPU

2．设计支持Stall的流水线CPU。通过检测竞争并插入停顿（Stall）机制解决数据冒险、控制竞争和结构冒险

3．在2.的基础上，增加 Forwarding机制解决数据竞争，减少因数据竞争带来的流水线停顿延时，提高流水线处理器性能

PS：也允许考虑将Stall与Forwarding结合起来实现

4．在3.的基础上，通过predict-not-taken或延时转移策略解决控制冒险/竞争，减少控制竞争带来的流水线停顿延时，进一步提高处理器性能

PS：也允许考虑将2、3和4结合起来设计

5．在4.的基础上，将CPU 支持的指令数量从16条扩充为31条，使处理器功能更加丰富（选做）

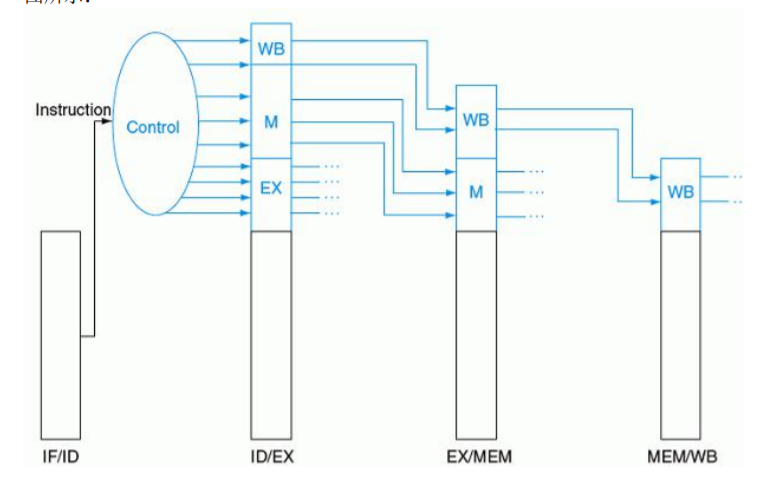
6．中断、异常处理（选做）

7．Cache的设计（选做）

2.实验描述

# 2.1 流水线处理器的寄存器设计

Control的输出需要被加入流水线寄存器保存下来，以供后续每级流水使用，结构如下图所示：

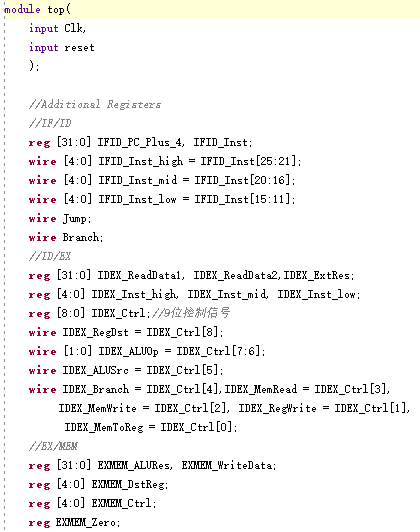


## 2.1.1 PC寄存器

PC用于存放指令的地址（32 位）。

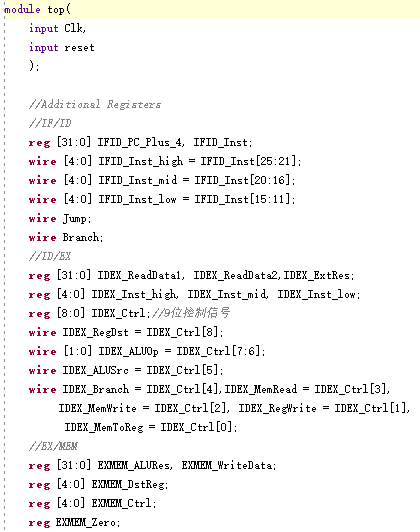
## 2.1.2 IF/ID段寄存器

存放指令本身、以及下一条指令的地址PC+4（其中Inst在这个阶段有3部分需要用到，分别以线连接，同时要得到从后方传递的jump与Branch控制信号）



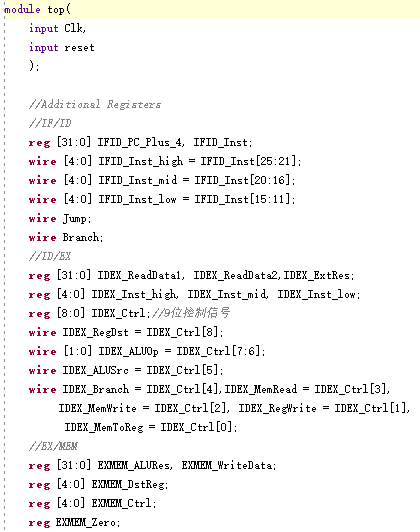
## 2.1.3 ID/EX段寄存器

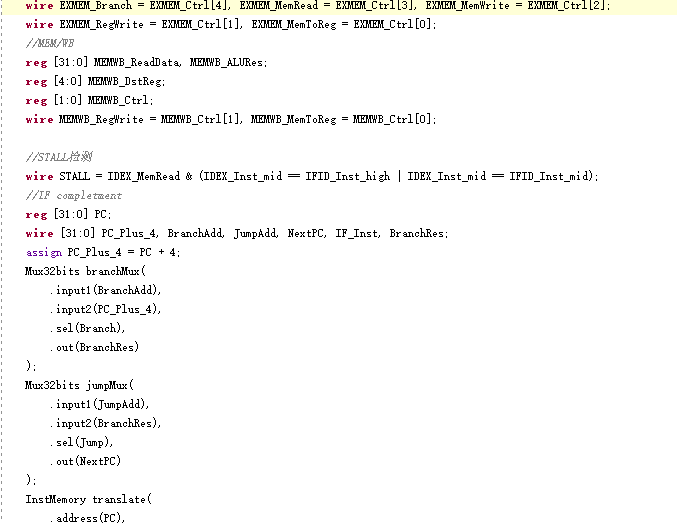
存放9位控制信号、两个读取的数据（32位）, 立即数 (32位),还有分别接受IF/ID段的3个5位数据



## 2.1.4 EX/MEM段寄存器

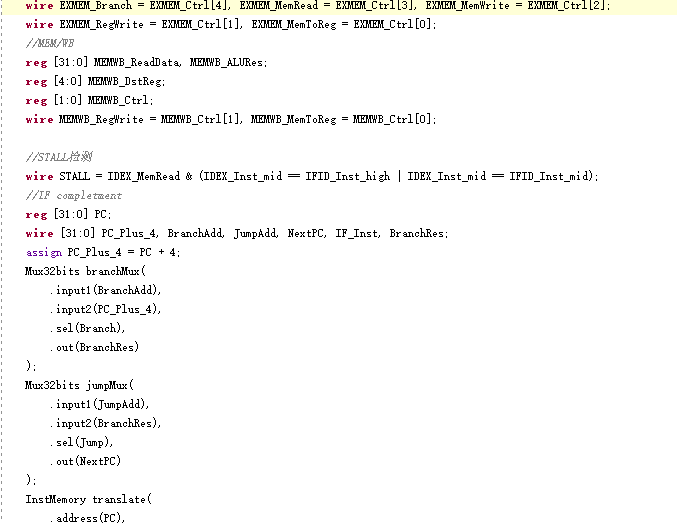
存放5位控制信号，zero信号、ALU的计算结果和可能的需要写入到目标寄存器序号



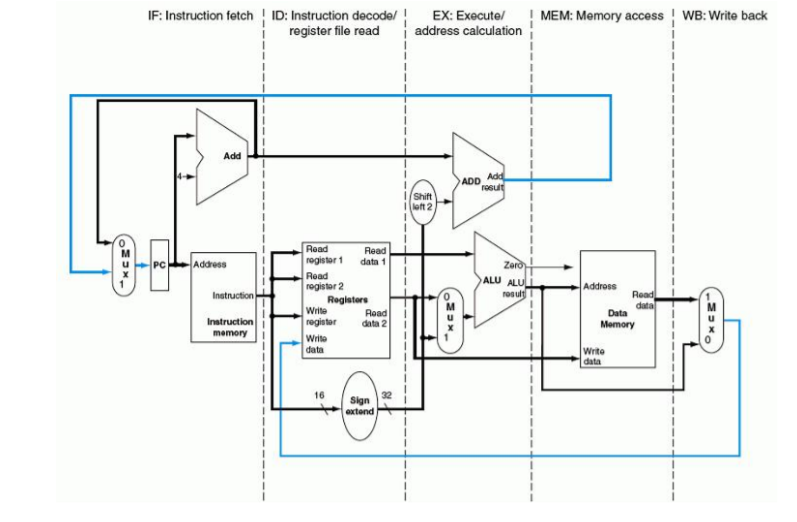


**2.1.5 MEM/WB段寄存器**

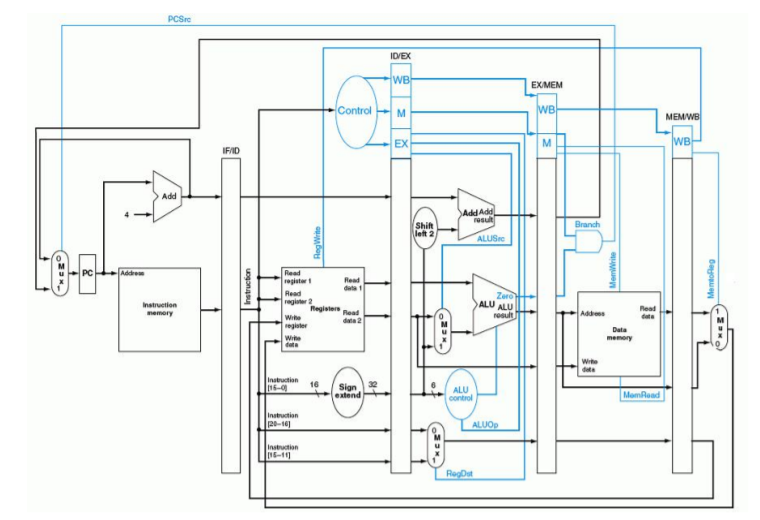
存放2 位控制信号，在数据DataMemory读到的数据，ALU传递的计算结果，写回的寄存器序号



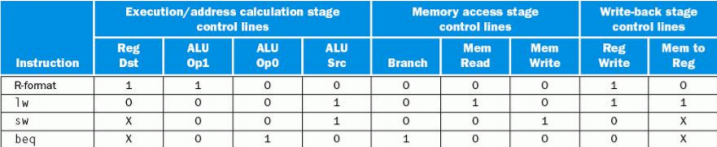
# 2.2 流水线处理器的各阶段设计



将单周期CPU进行分割，插入4级寄存器，将其分割为IF，ID，EX，MEM，WB 五大部分。各部分具体实现如下图所示：

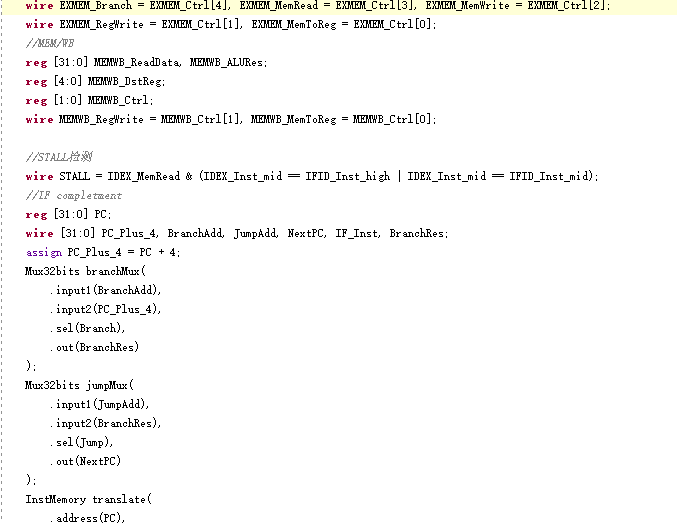


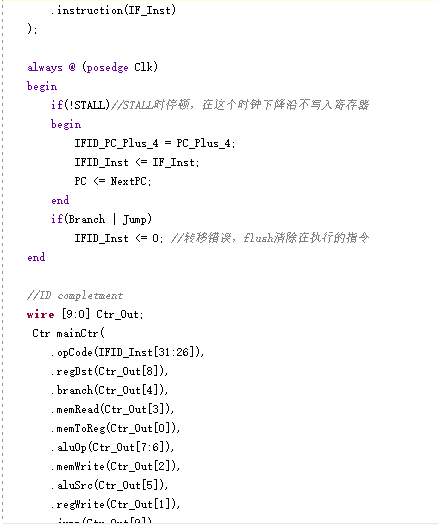
其中Control产生的控制信号如下图所示，其实现早在lab03中的Ctr中已经模块化实现



## 2.2.1 IF阶段实现

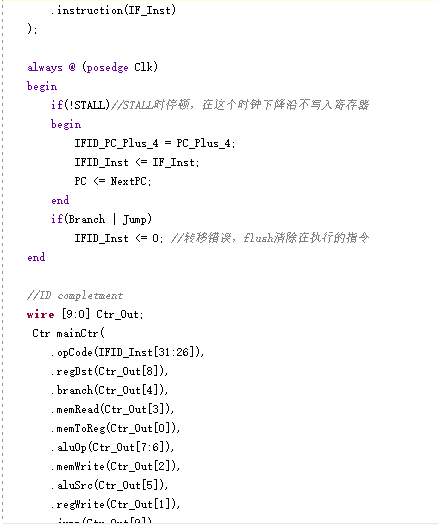
根据后续阶段传输出的控制信号与相应地址，更改PC的值，同时将之前PC的对应的Inst取出来

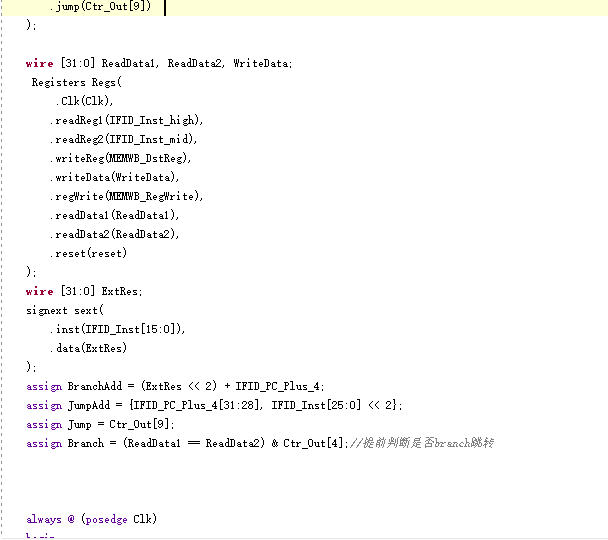


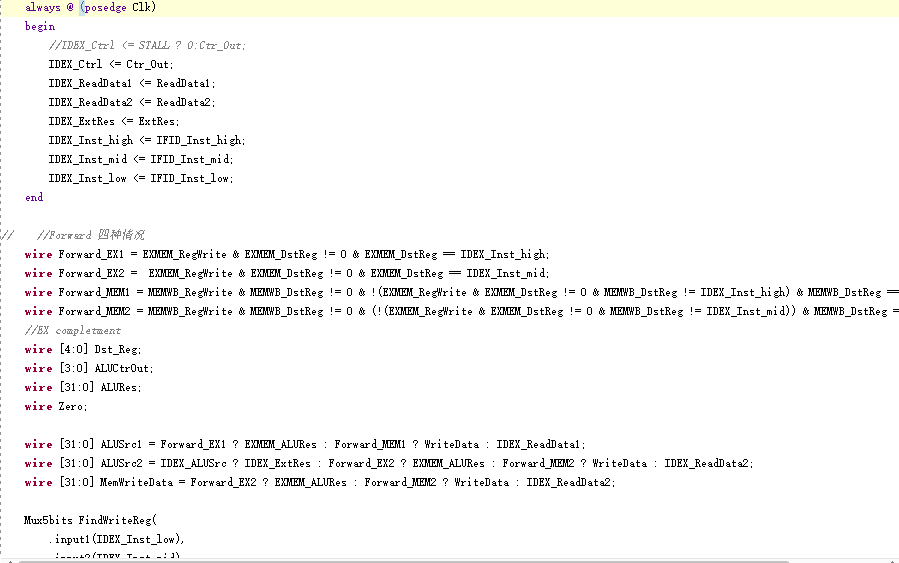


## 2.2.2 ID阶段实现

产生所有9位控制信号，同时译码，将对应数据从寄存器组中取出来，计算并且完成16位立即数的32位拓展，同时在此处提前计算可能产生的Jump跳转地址与Branch跳转地址

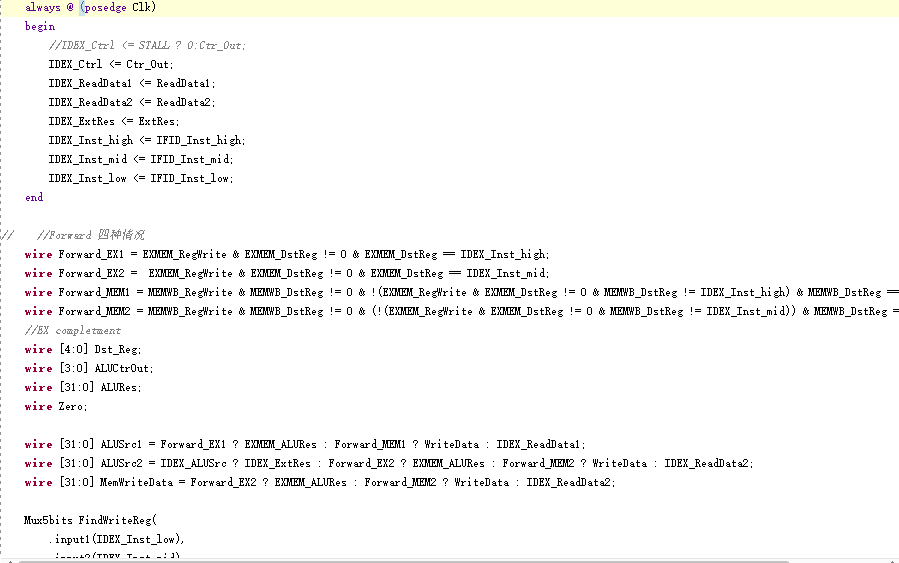


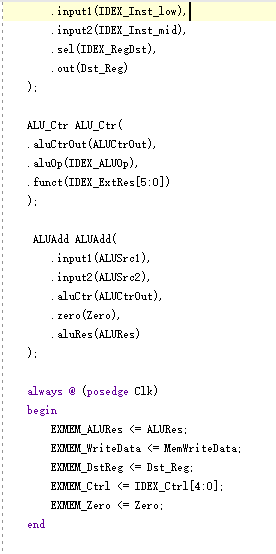




## 2.2.3 EX阶段实现

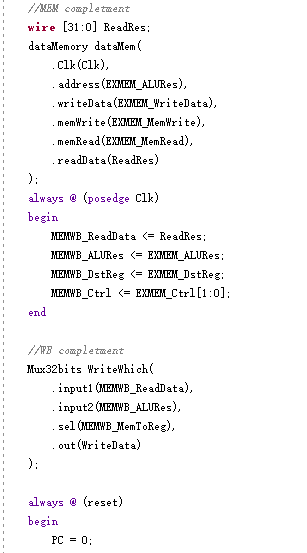
完成ALUOp的计算，就算ALU模块的输入，进行ALU的计算





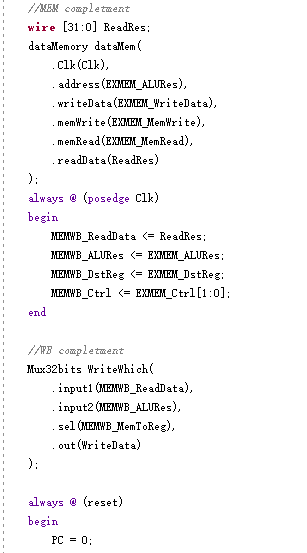
## 2.2.4 MEM阶段实现

完成dataMemory的访问，同时将部分结果重新传递回之前的阶段



## 2.2.5 WB阶段实现

只需完成对WriteData的选择，并不需要时钟下降沿的限制

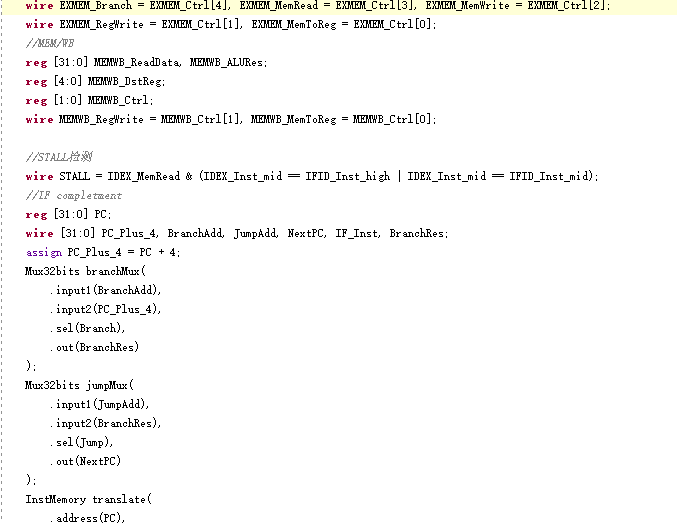


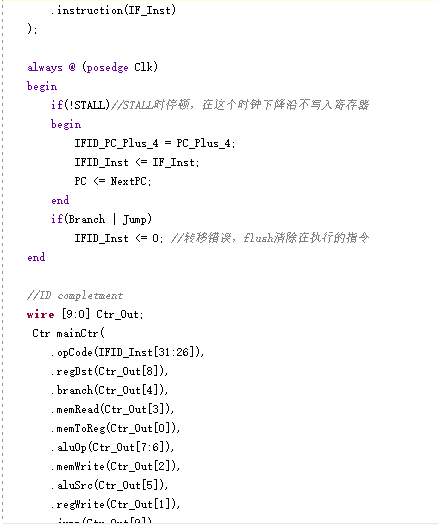
# 2.3 流水线处理器设计改进

对于一个流水线处理器，可能存在各种冒险，由于前方指令没有完成，改变对应寄存器或数据存储器中的值，导致后面的指令取到了错误的数据（数据冒险）；也有可能因为跳转指令导致该指令后面正在执行的指令无效需要清空并跳转PC（控制冒险），导致效率低下。为了避免数据的错误取值与控制冒险，我们需要对普通的流水线处理器进行改进。

## 2.3.1 Stall机制实现

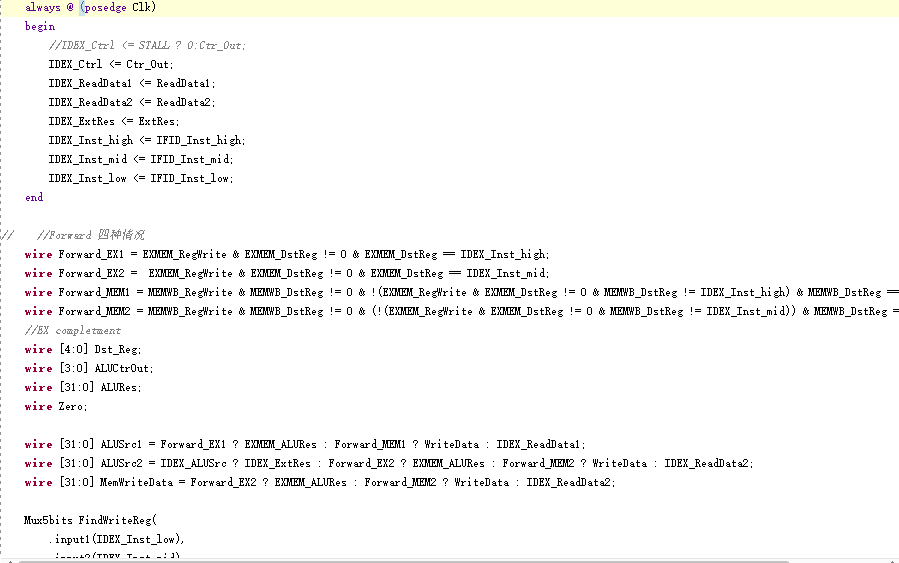
对于一个流水线处理器，有可能产生数据冒险（即一条指令需要等待前面仍在处理器中的指令完成写入后再进行下去），对于这种情况可以采取流水线停顿的方式实现，如果单纯使用Stall，可能需要多次停顿来实现。





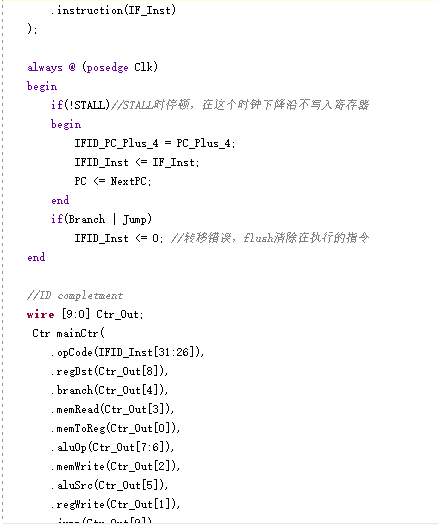
## 2.3.2 Forward机制实现

如果等结果重新写回寄存器，可能要经过几个cycle后才能得到数据，但如果我们在计算得到结果时就传回需要数据的输入端，便可以减少Stall的发生与数量。



## 2.3.3 predict-not-taken策略实现

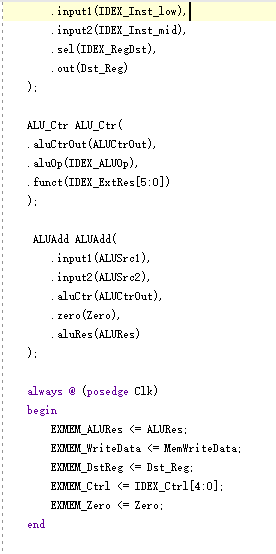
我们的预测策略为当碰到Branch指令时，预测不会转移，所以在没有得到结果时，后一条PC+4的指令继续执行如果发现跳转，则进行一次flush，将Branch后面载入的指令全部清空（若采用了Forward与Stall，那么在ID段便可得到判断结果，因此只需要将IF段清空即可）

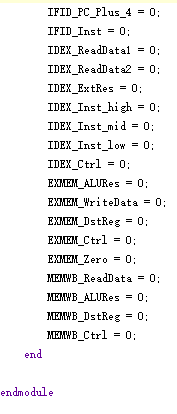
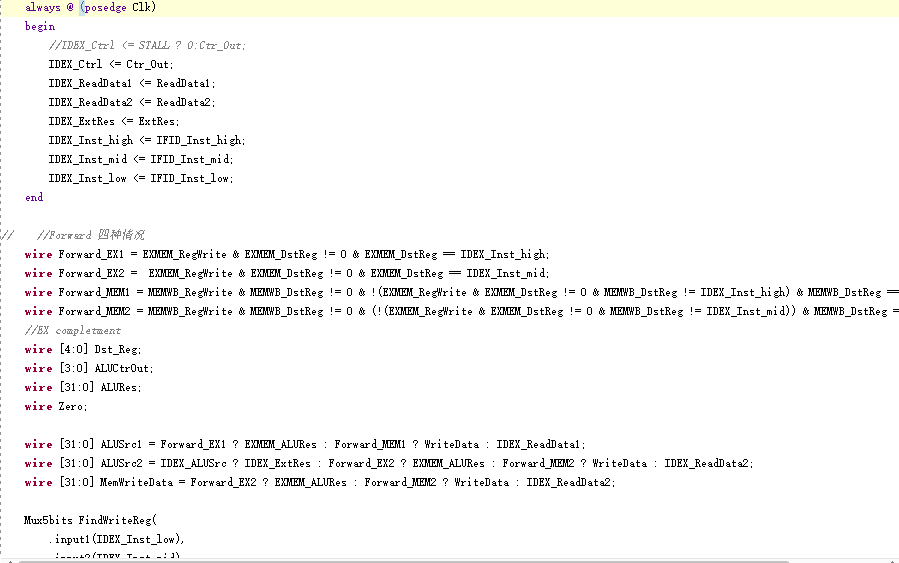
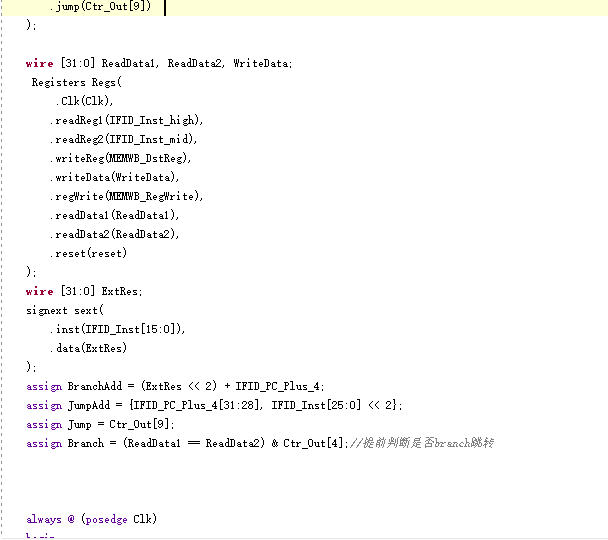
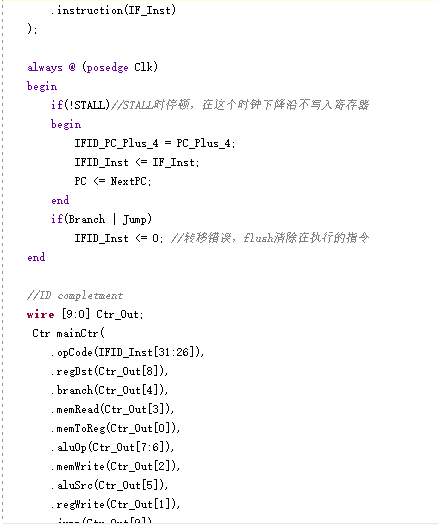
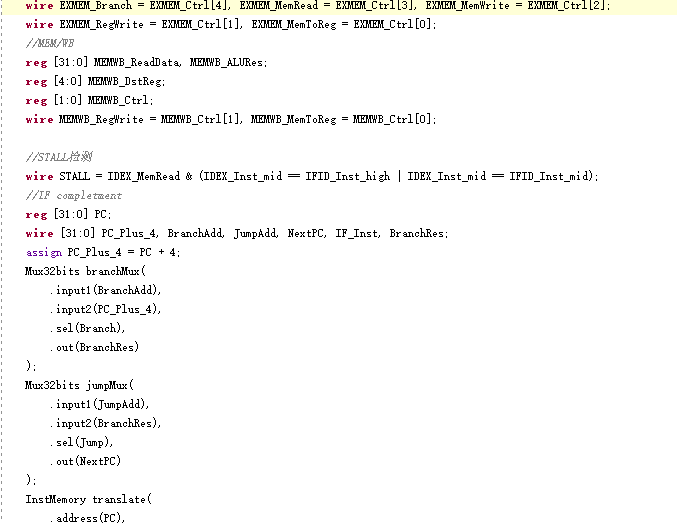
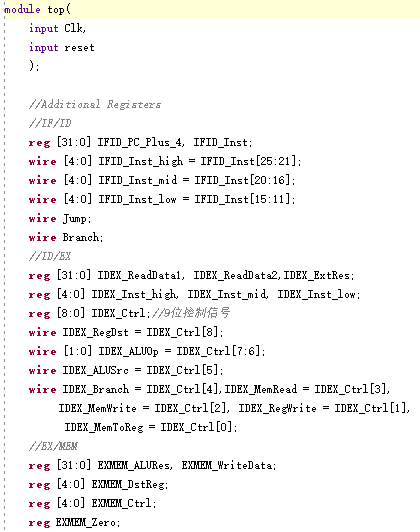


# 2.4 流水线处理器的整体模块

其中工作的每一模块已经在lab05中完成定义，在top文件中只需完成连接与时钟下降沿的工作

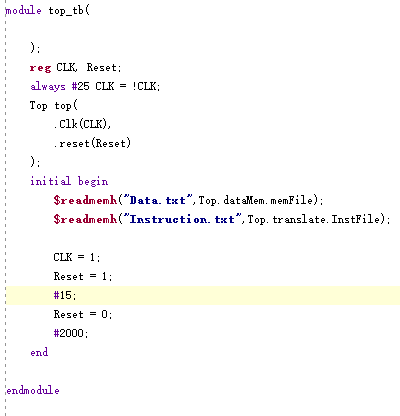
## 2.4.1 Top模块源文件





## 2.4.2 Top模块仿真激励

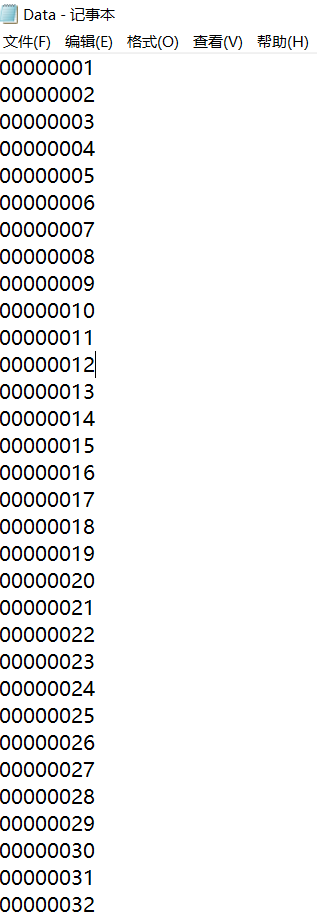
从外部读入Data.txt到memFile存储器中，读入Instruction.txt到InstFile存储器中，开始先将Reset置1，使得所有reg清零，然后读入。



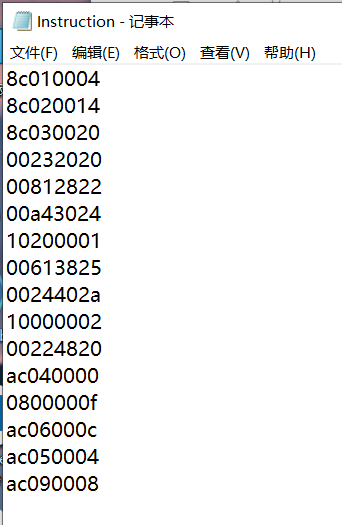
## 2.3.6 外部文件

DataMemory中寄存器的下标+1即为初始所存寄存器的值

Data.txt:



Instruction.txt：



上述8位16进制机器码对应mips指令为：

lw $1, 4($0) (0)

lw $2, 20($0) (1)

lw $3, 32($0) (2)

add $4, $1, $3 (3)

sub $5, $4, $1 (4)

and $6, $5, $4 (5)

beq $1, $0, begin1 (6)

or $7, $3, $1 (7)

slt $8, $3, $1 (8) begin1

beq $0, $0, begin2 (9)

add $9, $1, $2 (10)

sw $4, 0($0) (11)

j begin3 (12) begin2

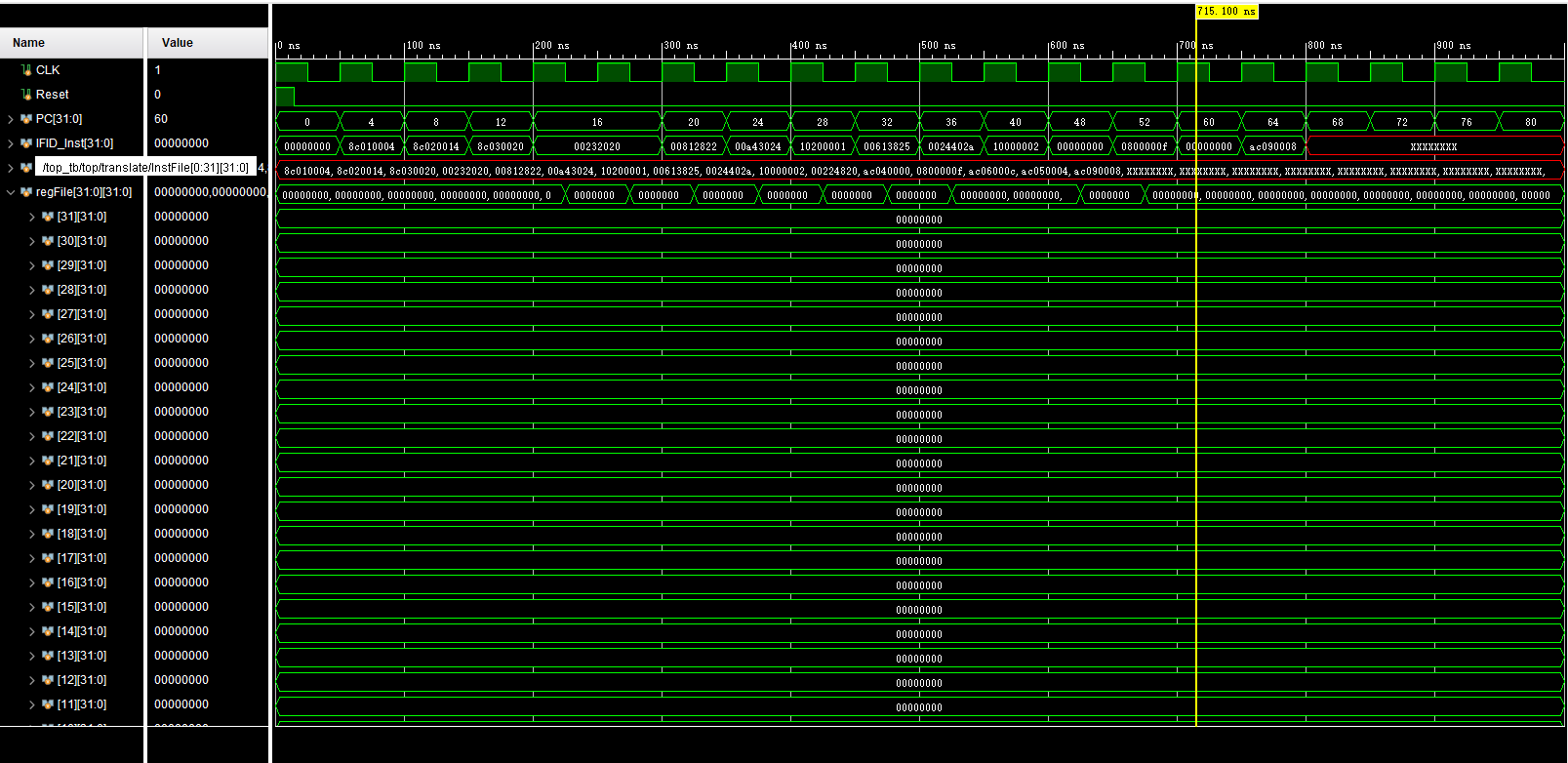
sw $6, 12($0) (13)

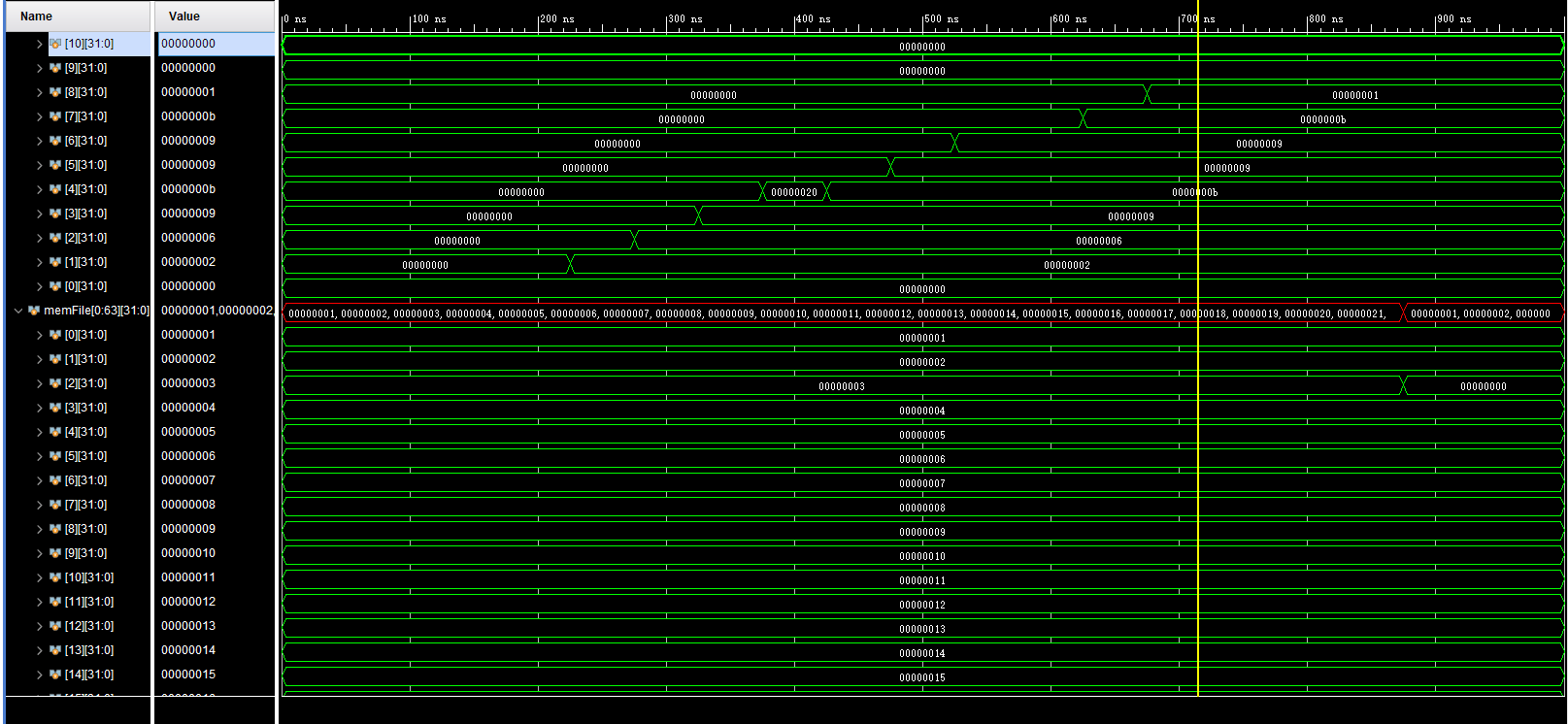
sw $5, 4($0) (14)

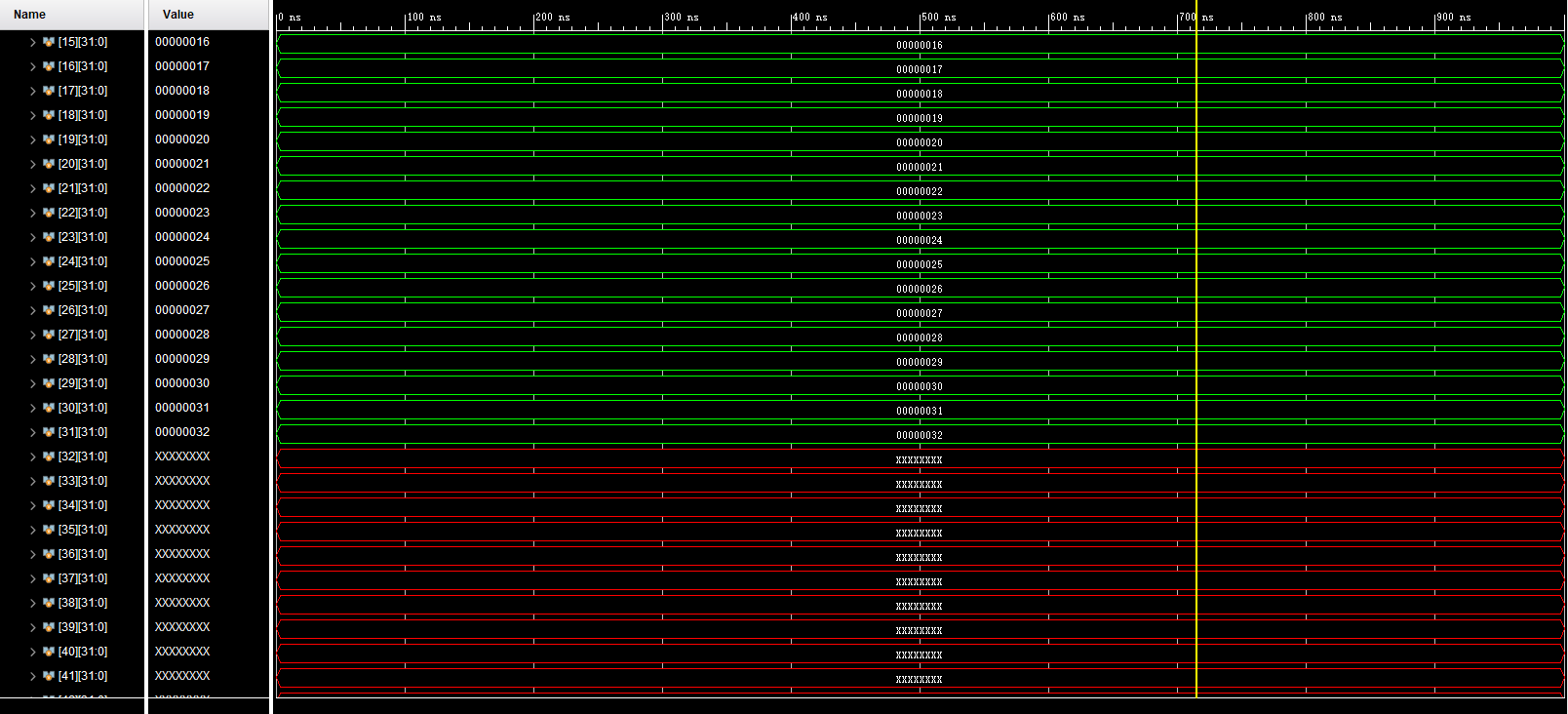
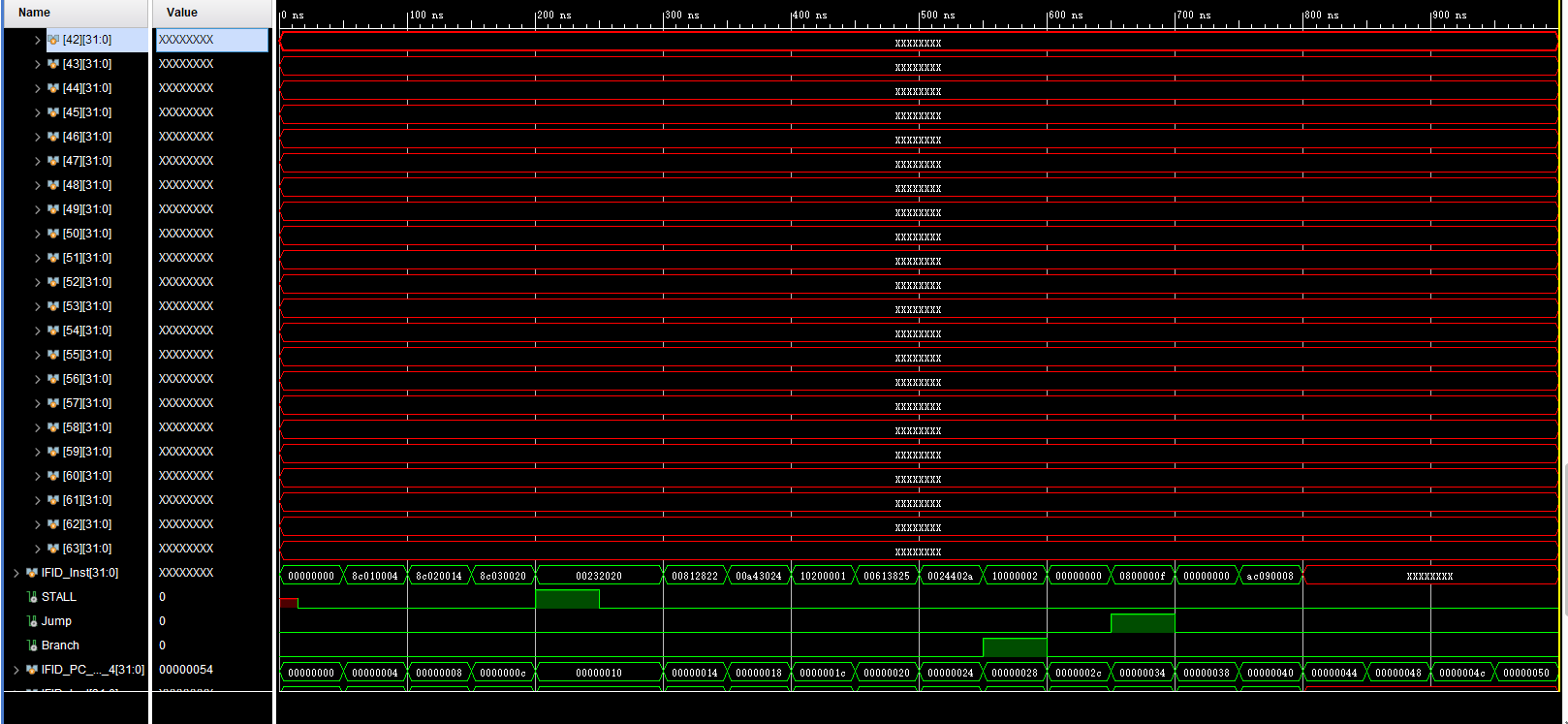
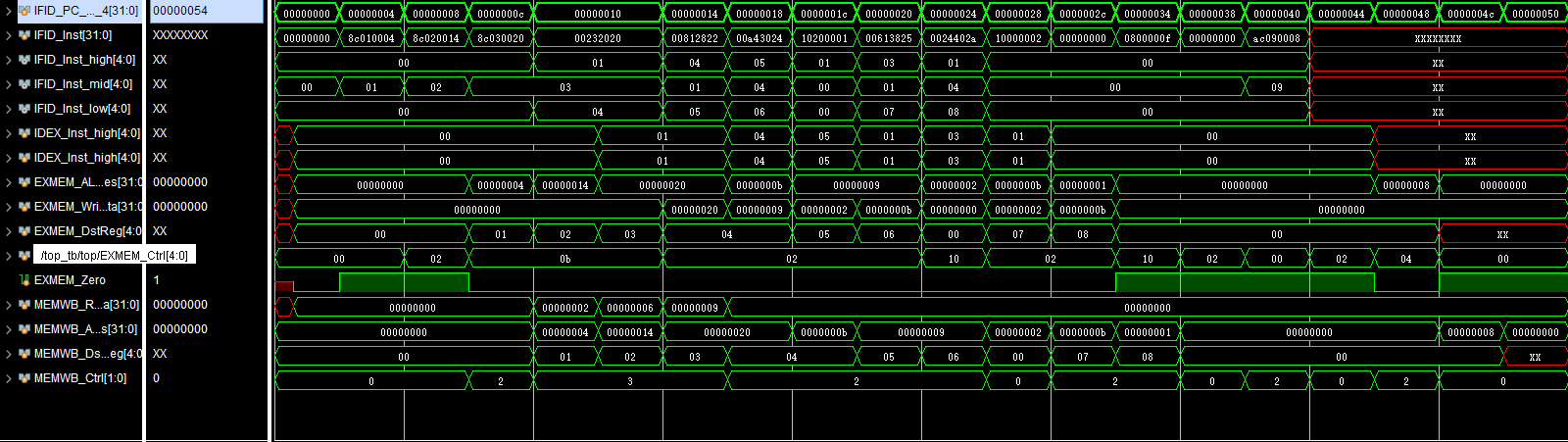
sw $9, 8($0) (15) begin3

可以看到在9种指令的基础上，（2）（3）指令存在数据冒险，应当需要一次Stall与Forward来规避错误，（3）（4）（5）存在数据冒险，需要Forward来实现；控制冒险有3次体现，（6）不跳转，判断成功，应当与普通指令相同，完全流水化，在（9）（12）都跳转，predict-not-taken不成功，应当flush清除之后进入的指令。

## 2.3.7 仿真波形





## 2.3.8 实验结论

在对应的实验代码下，我们可以看最下方的各阶段的段寄存器都是在每次时钟下降沿会发生改变（每周期改变，即执行不同指令），满足流水线处理器的特点。

而对于各类冒险，我们看到（2）（3）指令的数据冒险，存在一次Stall，使得200~300nsPC（2个时钟周期）没有发生改变（也可以根据Stall信号在此时变为1判断），然后可以看到（3）指令成功计算了lw后$3的值与$1值相加的结果，但是由于是在一次Stall后在Forward，在刚开始由于Stall，Forward产生的值不正确（这是由于Stall是简单停顿，没有清除所有寄存器的值）但这只是暂时的，在完成时会正确将正确的值存入寄存器。后面（3）（4）（5）存在数据冒险，但我们通过Forward直接前递没有流水线停顿，看各寄存器的值，是能够正确完成指令的。在（6）处没有跳转，预测成功，可以看到流水线如普通指令一般流水执行。在（8）处发生了Branch跳转，预测不成功，我们可以看到经过了flush，将此时在IF阶段的IFID\_Inst清零了，然后跳转到jump指令处，再次跳转，flush，IFID\_Inst再次清零。可以看到（10）（11）（13）（14）并没有被执行，对应的dataMemory与寄存器都没有存入对应的值，仅有最后一条sw指令被执行。符合实验预期。

3.实验心得与总结

实验6对比实验5，由于将低效率的单周期处理器改进为流水线处理器，所以会增加大量的段寄存器与数据传递，对于信号的传递与模块的连接提出了较高的要求，但由于不用改造整体模块，整体普通流水线处理器的工作量相较实验5不大，但是为了改进流水线处理器的性能与规避错误，必须增加诸如Stall、Forward、predict-not-taken等机制来提高性能，Stall与predict-not-taken的实现比较简单，但是对于Forward，我们需要考虑各类情况来决定EX阶段的input1、input2、MemWriteData，较为复杂，在调试这个部分的代码时花费了比较大量的时间。但实际上我只完成了9条指令情况下的流水线处理器，可以预见对于31条指令的流水线处理器将更为复杂，我也同时意识到，虽然我采取了3种策略，但是依然有很多情况下处理器的效率依然不够高，但是相较于实验5的单周期处理器还是有性能上的提升。

这次的实验，让我在理论课的基础上更加巩固了流水线处理器相关的知识，了解了细节，虽然没有上板实验是一个遗憾，但是我依然学到了很多，包括但不限于Verilog的应用、单周期处理器与多周期处理器的设计，收获满满。

4.参考资料

2020计算机系统结构实验指导书-LAB06\_M