**计算机系统结构实验实验报告 Lab05**

姓名：叶增渝

学号：519030910168

日期：2021年04月14日

**目录**

[1.实验概述 3](#_Toc70973266)

[1.1 实验名称 3](#_Toc70973267)

[1.2实验目的 3](#_Toc70973268)

[2.实验描述 3](#_Toc70973269)

[2.1 InstMemory 3](#_Toc70973270)

[2.1.1InstMemory模块描述 3](#_Toc70973271)

[2.2 Mux单元模块 4](#_Toc70973272)

[2.2.1 Mux单元模块描述 4](#_Toc70973273)

[2.2.2 32位Mux单元模块代码 5](#_Toc70973274)

[2.2.3 5位Mux单元模块代码 5](#_Toc70973275)

[2.3 dataMemory模块修改后代码 5](#_Toc70973276)

[2.2.5 ALUAdd模块修改后代码 6](#_Toc70973277)

[2.3单周期处理器整体模块 7](#_Toc70973278)

[2.3.1定义信号线 7](#_Toc70973279)

[2.3.2程序计数器PC 8](#_Toc70973280)

[2.3.3 RESET 8](#_Toc70973281)

[2.3.4 Top模块代码 8](#_Toc70973282)

[2.3.5 Top模块仿真激励 12](#_Toc70973283)

[2.3.6 外部文件 13](#_Toc70973284)

[2.3.7 仿真波形 15](#_Toc70973285)

[2.3.8 实验结论 16](#_Toc70973286)

[3.实验心得与总结 16](#_Toc70973287)

[4.参考资料 17](#_Toc70973288)

1.实验概述

1.1 实验名称

类 MIPS 单周期处理器的设计与实现

1.2实验目的

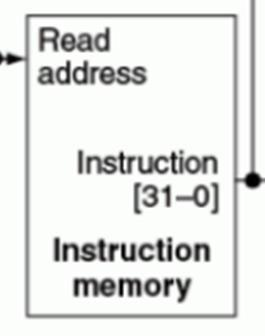
完成单周期的类 MIPS 处理器  
设计支持 16 条 MIPS 指令（add,sub,and,or,addi,andi,ori,slt,lw,sw,beq, j,jal,jr,sll,srl）的单周期 CPU

2.实验描述

# 2.1 InstMemory

## 2.1.1InstMemory模块描述

InstMemory模块读入一个ReadAddress将其右移两位得到对应的内存地址用于取出对应的32位指令

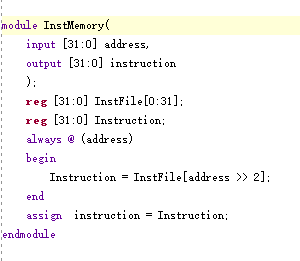


InstMemory模块

**2.1.2 InstMemory模块代码**

由于PC每次以+4为单位，所以Address右移两位即为地址，输出为地址对应的32位数据。

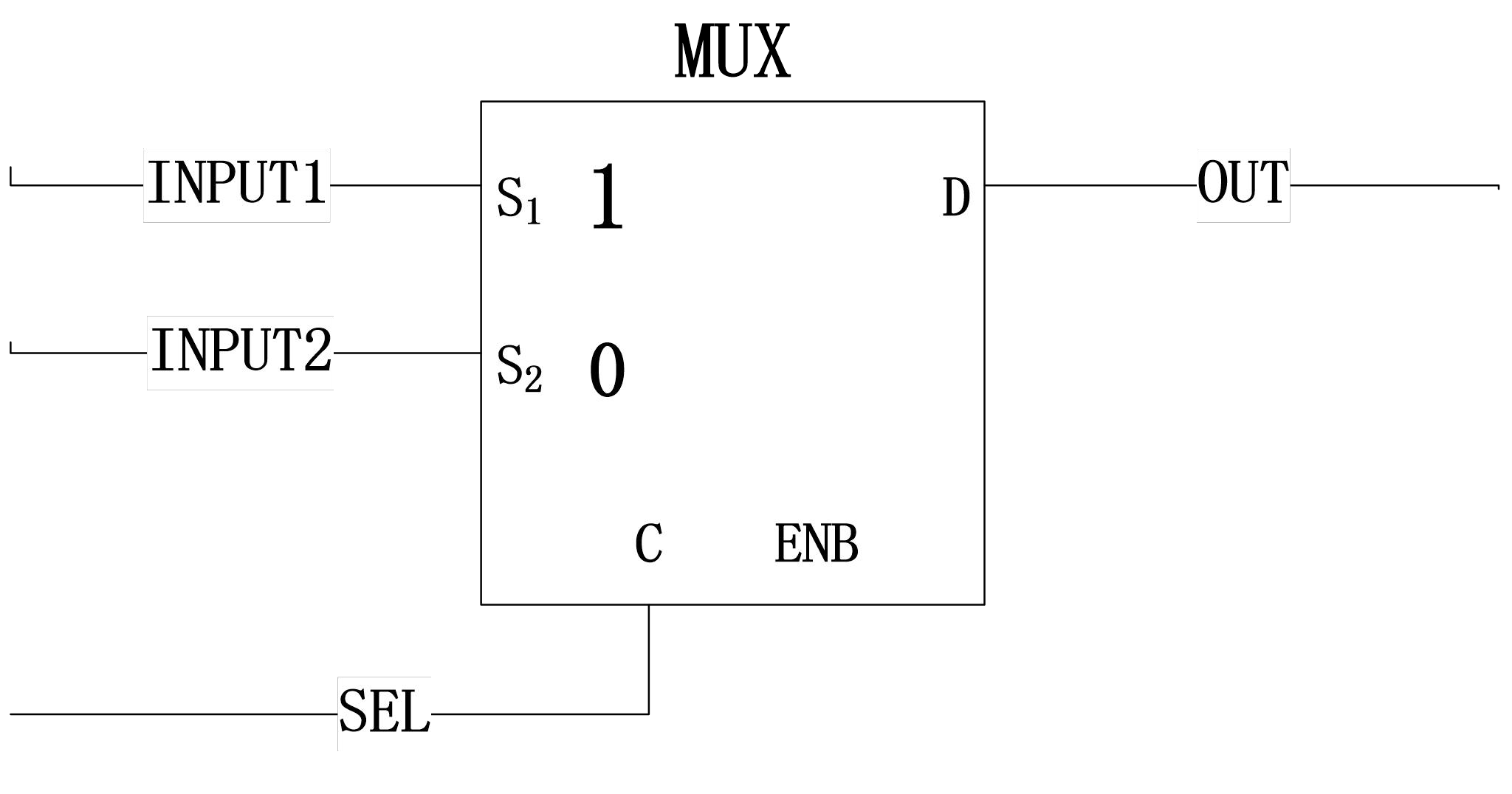
（对dataMemory同样的操作）



# 2.2 Mux单元模块

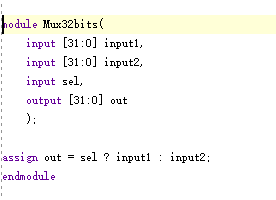
## 2.2.1 Mux单元模块描述

Mux 实现很简单，使用一个三目运算符Assign OUT = SEL ? INPUT1 : INPUT2;  
OUT, SEL, INPUT1, INPUT2 都是预先定义好的信号

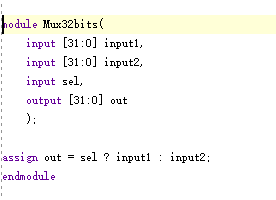


Mux单元

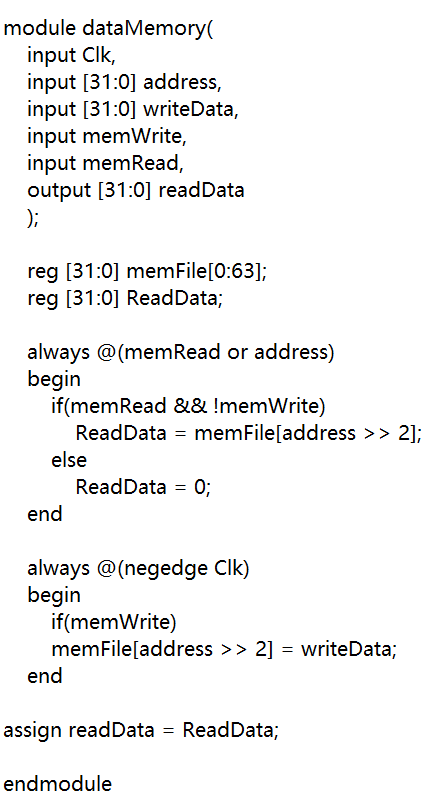
## 2.2.2 32位Mux单元模块代码



## 2.2.3 5位Mux单元模块代码

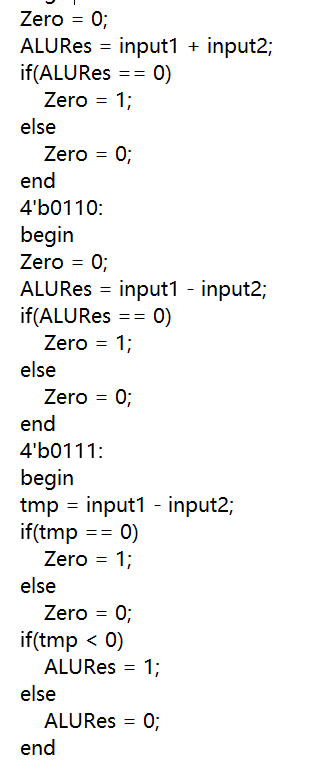
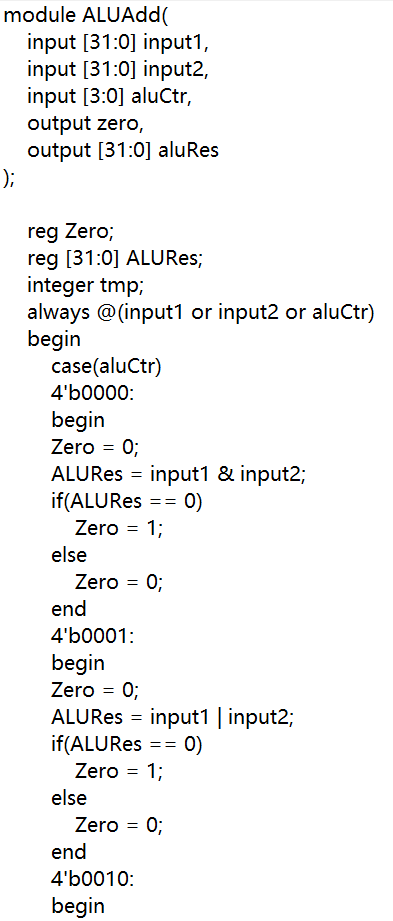


# 2.3 dataMemory模块修改后代码

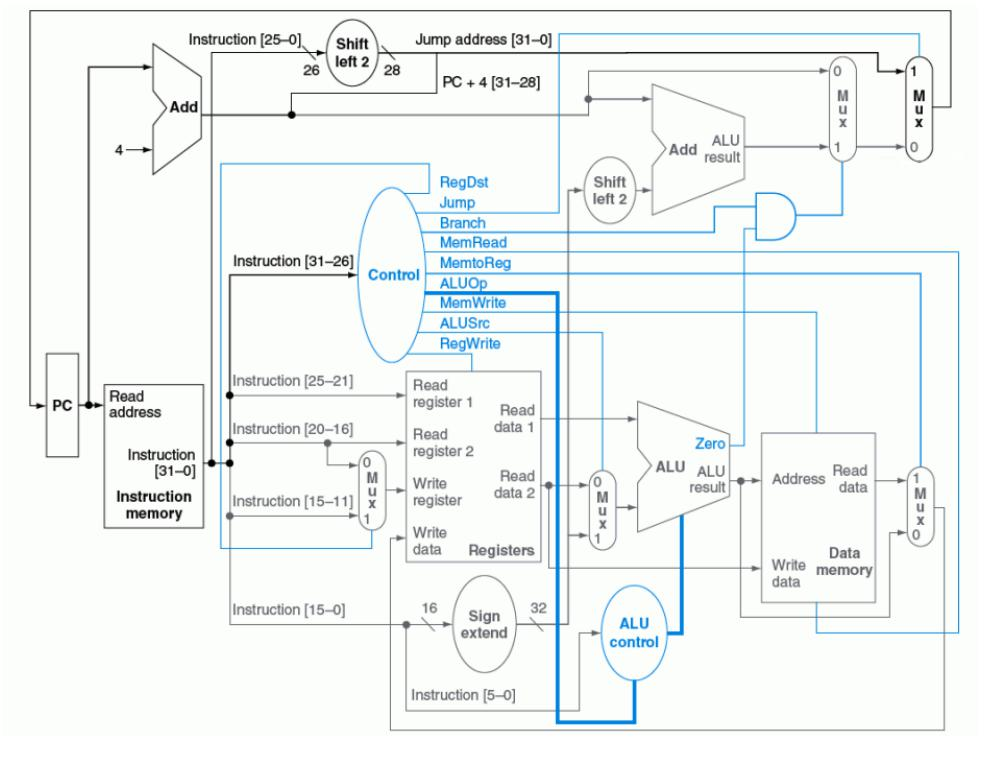


# 2.2.5 ALUAdd模块修改后代码

此处仅展示到slt的改变，增加了一个暂时的变量tmp，用于slt的数字暂时存储



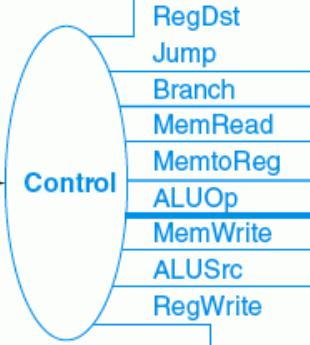
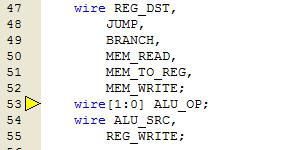
# 2.3单周期处理器整体模块



类MIPS单周期处理器原理图

## 2.3.1定义信号线

为 Top 模块内的每一根连接的信号线命名， 并在 Top 模块中定义它们



主控制模块输出端口上的连线的举例

## 2.3.2程序计数器PC

程序计数器是这个简单 CPU 能够跑起来的关键。定义一个 32 位reg类型的程序计数器PC，在时钟上升沿(下降沿已经被我们用作寄存器的写了)做 PC<=PC+4。  
PS：在组合逻辑中用阻塞赋值“=”，时序逻辑中用非阻塞赋值“<=”。

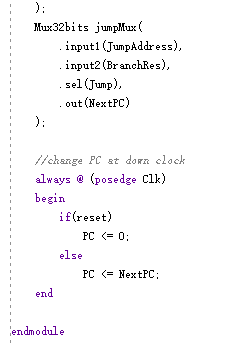
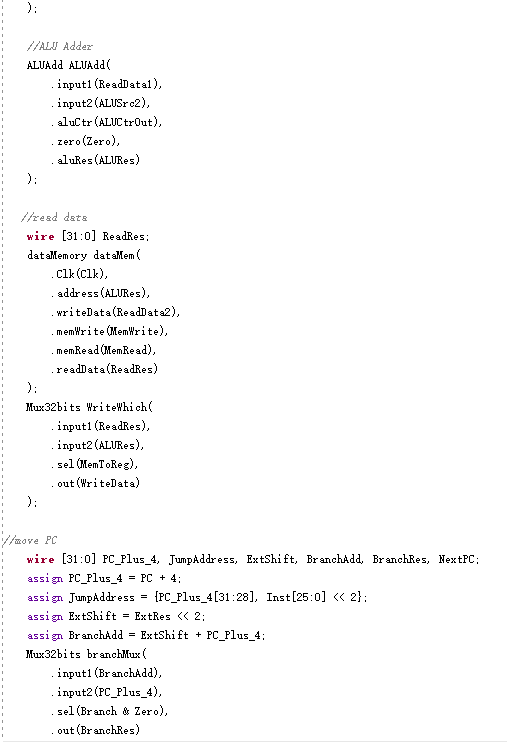
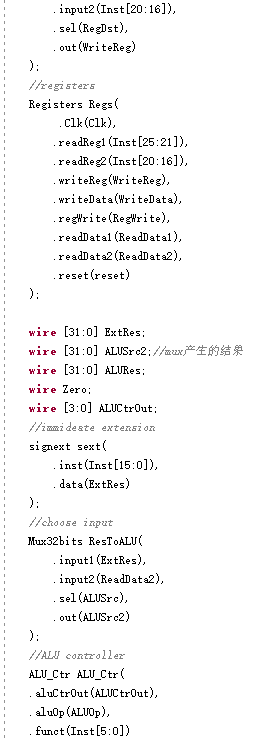
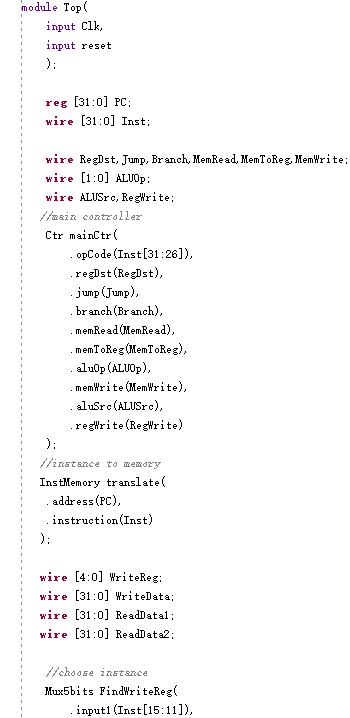
## 2.3.3 RESET

PC置0x00000000，各寄存器清零，这是 reset 要做的工作。同步或异步，边沿或电平，需修改register模块，添加reset信号

## 2.3.4 Top模块代码

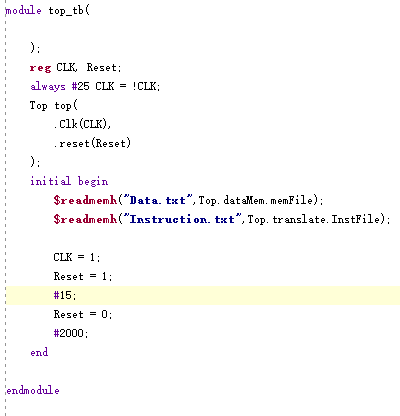
dataMemory模块相比lab04，下标从address更改为address >> 2（对dataMempry与InstMemory而言）；对register模块，增加readReg1与readReg2指向0号寄存器时的判断；ALUAdd增加对slt的结果生成，比较后更改ALURes为0或1。

根据原理图在top源文件中将各个模块连接起来即可。需要注意的是在时钟下跳沿根据reset的情况将PC设置为0或者nextPC。



## 2.3.5 Top模块仿真激励

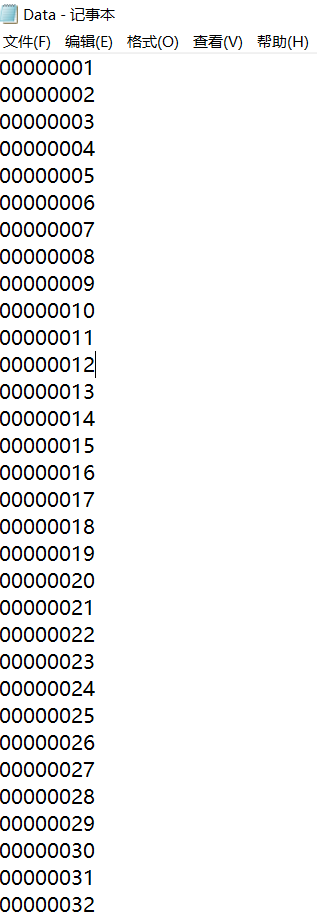
从外部读入Data.txt到memFile存储器中，读入Instruction.txt到InstFile存储器中，开始先将Reset置1，使得所有reg清零，然后读入。



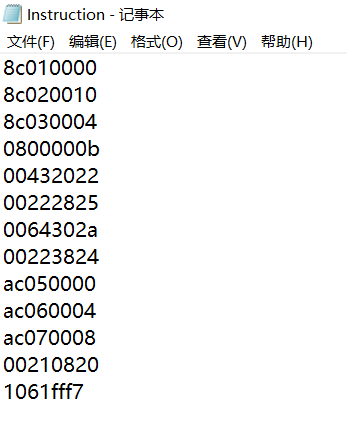
## 2.3.6 外部文件

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, 0($0) (0)

lw $2, 16($0) (1)

lw $3, 4($0) (2)

j begin1 (3)

sub $4, $3, $2 (4) begin2

or $5, $1, $2 (5)

slt $6, $3, $4 (6)

and $7, $1, $2 (7)

sw $5, 0($0) (8)

sw $6, 4($0) (9)

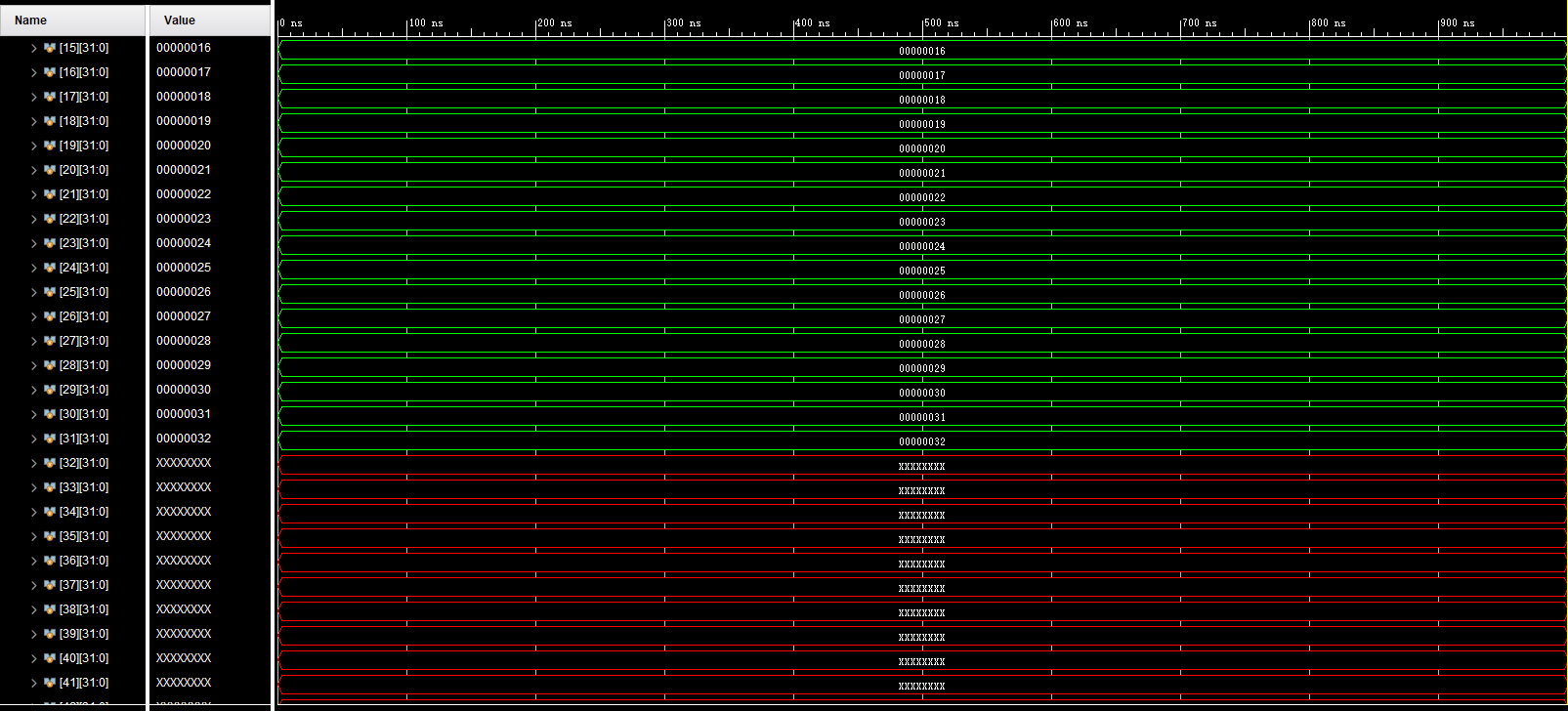
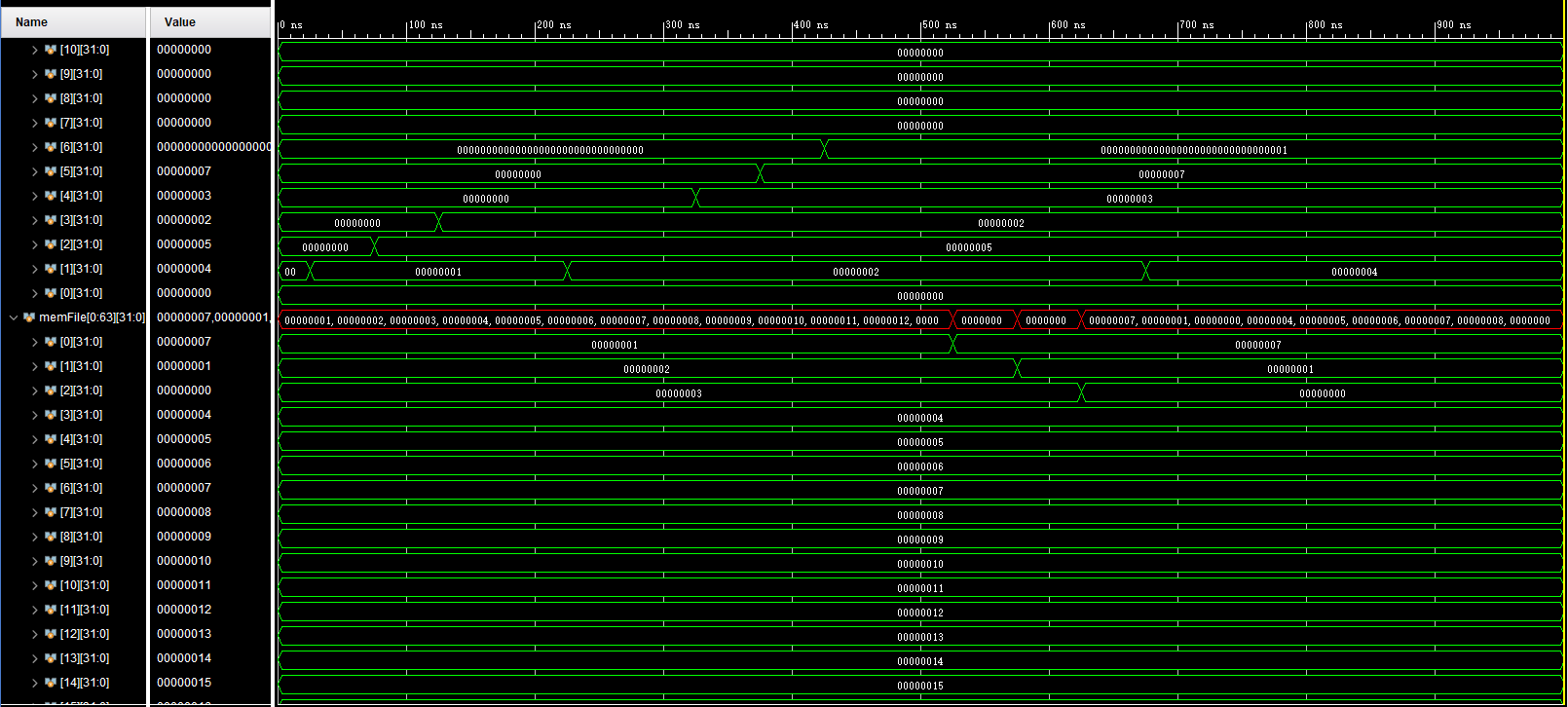
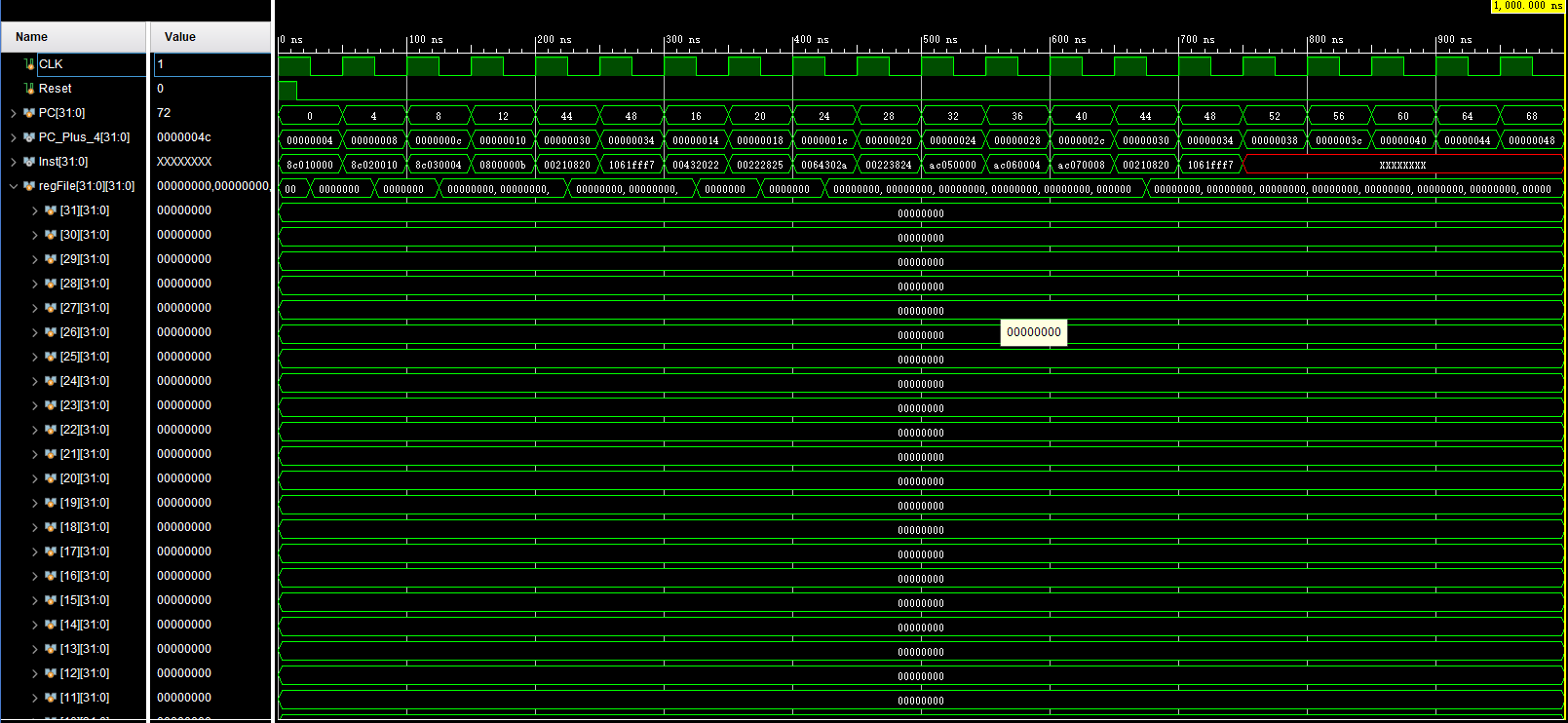
sw $7, 8($0) (10)

add $1, $1, $1 (11)

beq $1, $3, begin2 (12) begin1

在这样的data下，实际执行的mips指令为从0执行至3，通过jump指令跳转至11处，将1寄存器的值加倍，通过12重新跳转至4号指令处，然后顺序执行至12处结束。

## 2.3.7 仿真波形



## 2.3.8 实验结论

在对应的实验代码下，首先成功地将3个数分别写入3个寄存器，然后无条件转移至add操作，将1号寄存器值加倍，与3号寄存器相同，然后从sub开始顺序执行减、按位或、slt、按位与操作，分别存于4、5、6、7号寄存器，其中slt成功，输出结果到6号寄存器值赋1，按位操作两源数据后三位010与101，与为0，或为7，值均符合预期，然后执行3次sw操作，分别将0、1、2号memFile，最后再执行add在将1号寄存器值加倍，beq不再成立跳转，在750ns完成所有代码读取，后续PC对应的指令为空，结束。说明在指令代码正确的情况下，能够正确执行9种指令代码没有出错，符合实验要求。

P.S. memFile为64个32位Reg，Data.txt文件仅设置了前32个值，所以后面的memFile均为x，Inst在750ns后无指令了，所以取到为x，不影响正确性。

3.实验心得与总结

实验5对比实验2、3、4可以说是质的差别，要通过Verilog完整地还原单周期处理器的工作流程，虽然有实验3、4的铺垫，总工作量少了很多，但是依然经历了漫长的debug过程。由于类MIPS处理器线路图非常多，导致在top文件中接线时产生了许多错误；为了对应PC每次+4更改了dataMemory的实现；为了符合slt，还另设了一个临时变量integer tmp，然后根据情况指定ALURes。刚开始甚至把输入的Data.txt与Instruction.txt放错了文件夹，放入了src文件夹，后来才知道应该在sim文件夹向下的位置；而Instruction的设计翻译也是一个难点，既要覆盖9个指令，又要保证32位指令的正确性，也花费了不少时间。好在成功在将近1.5次课的时间内成功完成了类MIPS单周期处理器的实现，感觉对Verilog的接线有了更加深的了解，也更有了完成时的成就感。在经过各类复杂代码的测试后，发现

类MIPS单周期处理器的主要设计思想还是将一个处理器应有的功能模块化，在实现一个个小的模块的基础上，通过顶层的线连接来组成，在时钟下跳沿来临时根据reset来改变PC的值就好了。

但不可否认的是由于时间限制，没有完成16条指令的类MIPS单周期处理器。相比于9条指令，我们依然需要大量的工作，首先增加对应的Ctr信号，增加ALUAdd的可执行的功能，将ALUCtr的ALUOp信号扩展为3位等等，算是lab05的一个遗憾。

4.参考资料

2020计算机系统结构实验指导书-LAB05\_M