**实验四 单周期CPU实验**

1. **实验目的**

1. 理解 MIPS 指令结构，理解 MIPS 指令集中常用指令的功能和编码，学会对这些指令进行归纳分类。

2. 了解熟悉 MIPS 体系的处理器结构，如延迟槽，哈佛结构的概念。

3. 熟悉并掌握单周期 CPU 的原理和设计。

4. 进一步加强运用 verilog 语言进行电路设计的能力。

5. 为后续设计多周期 cpu 的实验打下基础。

1. **实验设备**

1、装有 Xilinx Vivado的计算机一台

2、 LS-CPU-EXB-002教学系统实验箱一套

1. **实验内容**

根据提供的 32 位 MIPS 框架，至少调试出 5 条指令（运算类、传送类、访存类、 控制转移类等）的运行波形。

单周期 CPU 就是在 1 个 CPU 周期内执行完一条指令，执行指令所需的微操作由控 制器产生，结构简单，容易设计。指令系统可包括如下指令类型：

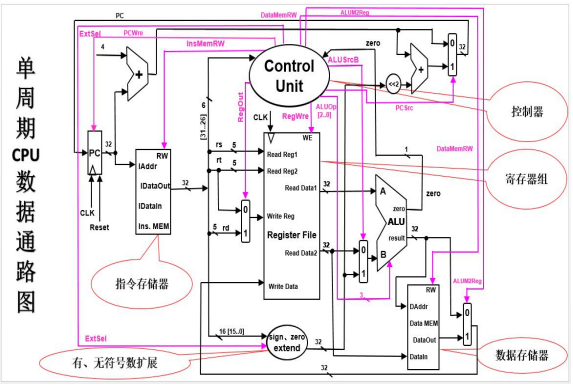
运算类指令：算术运算、逻辑运算、移位运算，如 ADD 、SUB、AND、NOT 、 SLL、SRL等等。

传送类指令：寄存器之间的传送、立即数传送，如 MOVZ、MOVN、MFHI 等等。

访存类指令：读存储器指令、写存储器指令，如 LB、LW、SB、SW 等等。

控制类指令：无条件转移、有条件转移，如 J、JR、JAL、BEQ 等等。

设计中所有寄存器和存储器都是异步读同步写的，即读出数据不需要时钟控制，但写入数据需时钟控 制。 故单周期 CPU 的运作即：在一个时钟周期内，根据 PC 值从指令 ROM 中读出相应的 指令，将指令译码后从寄存器堆中读出需要的操作数，送往 ALU 模块，ALU 模块运算 得到结果。 如果是 store 指令，则 ALU 运算结果为数据存储的地址，就向数据 RAM 发出写请 求，在下一个时钟上升沿真正写入到数据存储器。 如果是 load 指令，则 ALU 运算结果为数据存储的地址，根据该值从数据存 RAM 中 读出数据，送往寄存器堆，根据目的寄存器发出写请求，在下一个时钟上升沿真正写入 到寄存器堆中。 如果非 load/store 操作，若有写寄存器堆的操作，则直接将 ALU 运算结果送往寄存器堆，根据目的寄存器发出写请求，在下一个时钟上升沿真正写入到寄存器堆中。 如果是分支跳转指令，则是需要将结果写入到 pc 寄存器中的。



1. **实验步骤**

1、启动Vivado，选择File->New Project，输入工程名称，选择工程文件的路径

2、选择RTL Project，勾选Do not specify sources at this time

3、在筛选器中进行如下选择：

family->Artix7 package->fbg676 最后选择型号xc7a200tfbg676-2

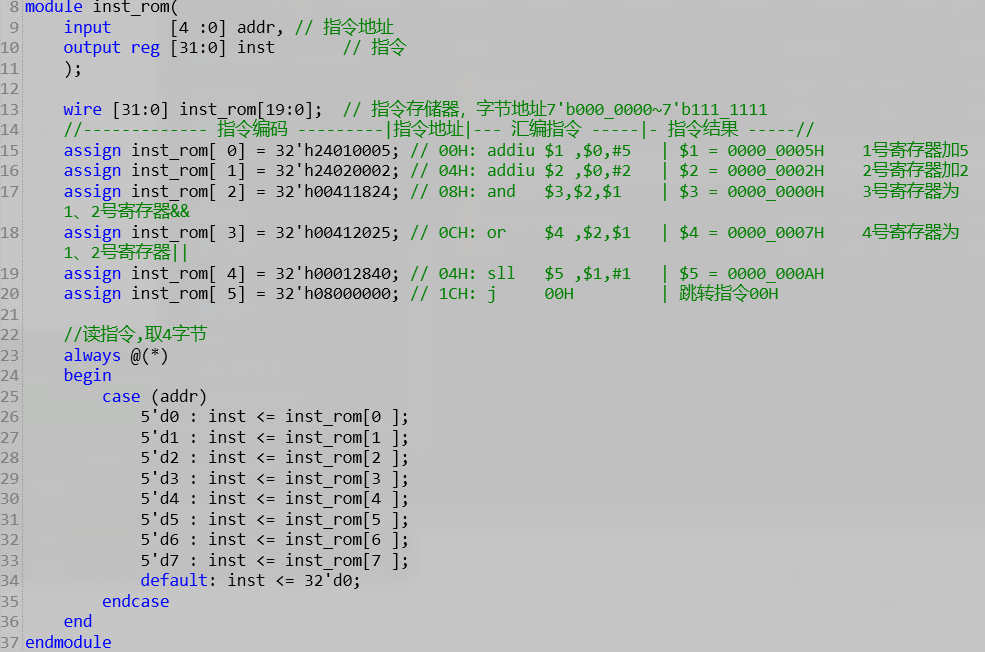
4、添加源文件Project Manager->Add sources->Add or create design sources

->Create File,开始输入程序代码

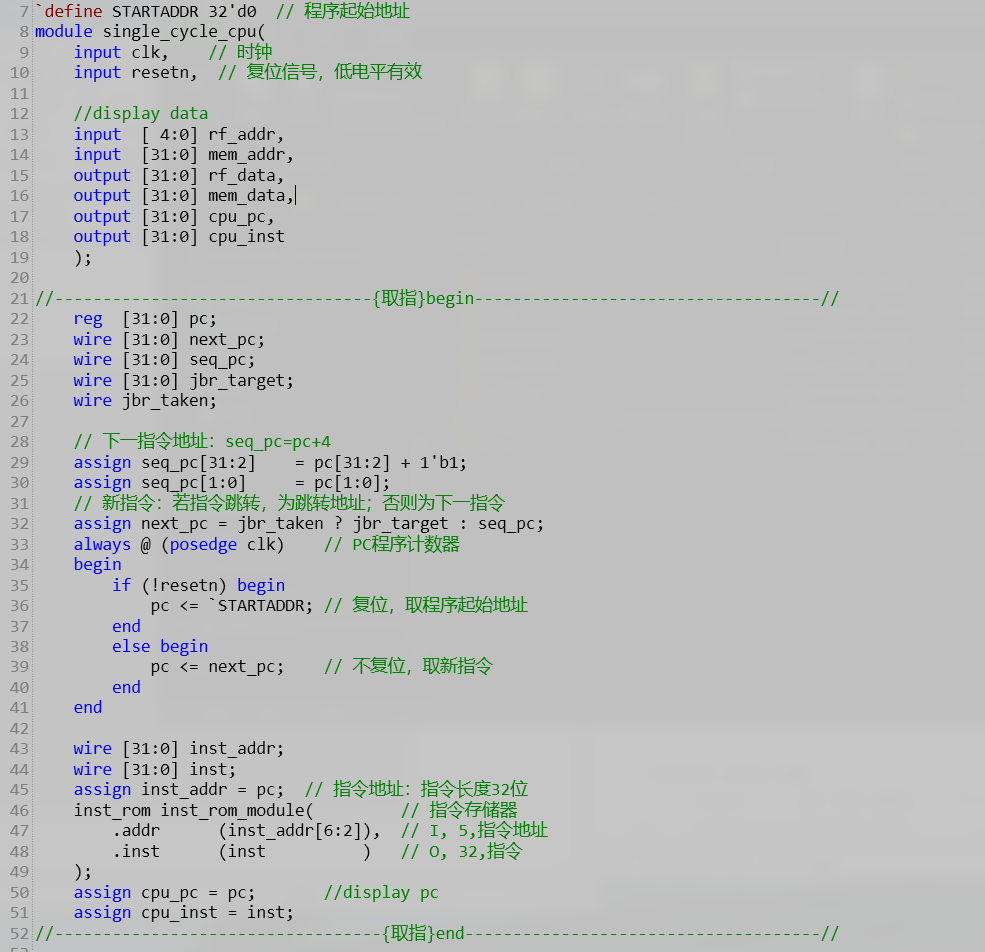
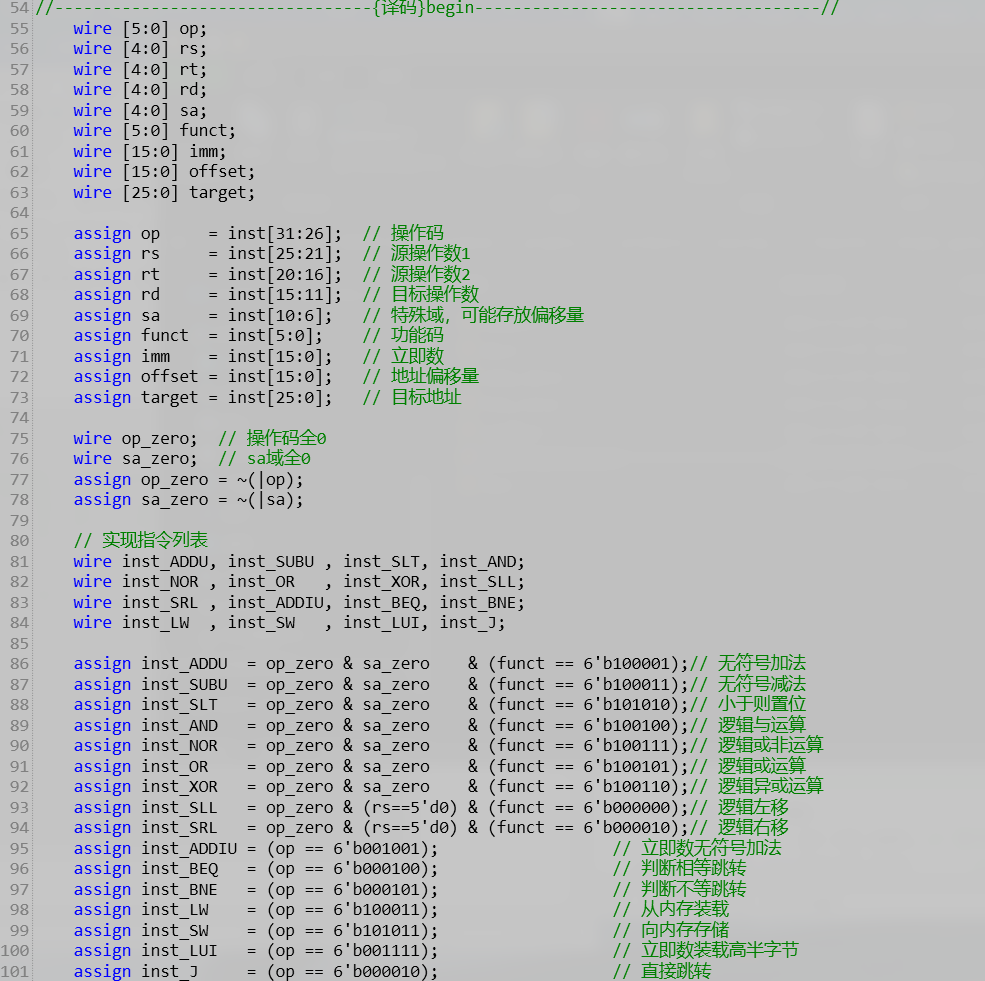
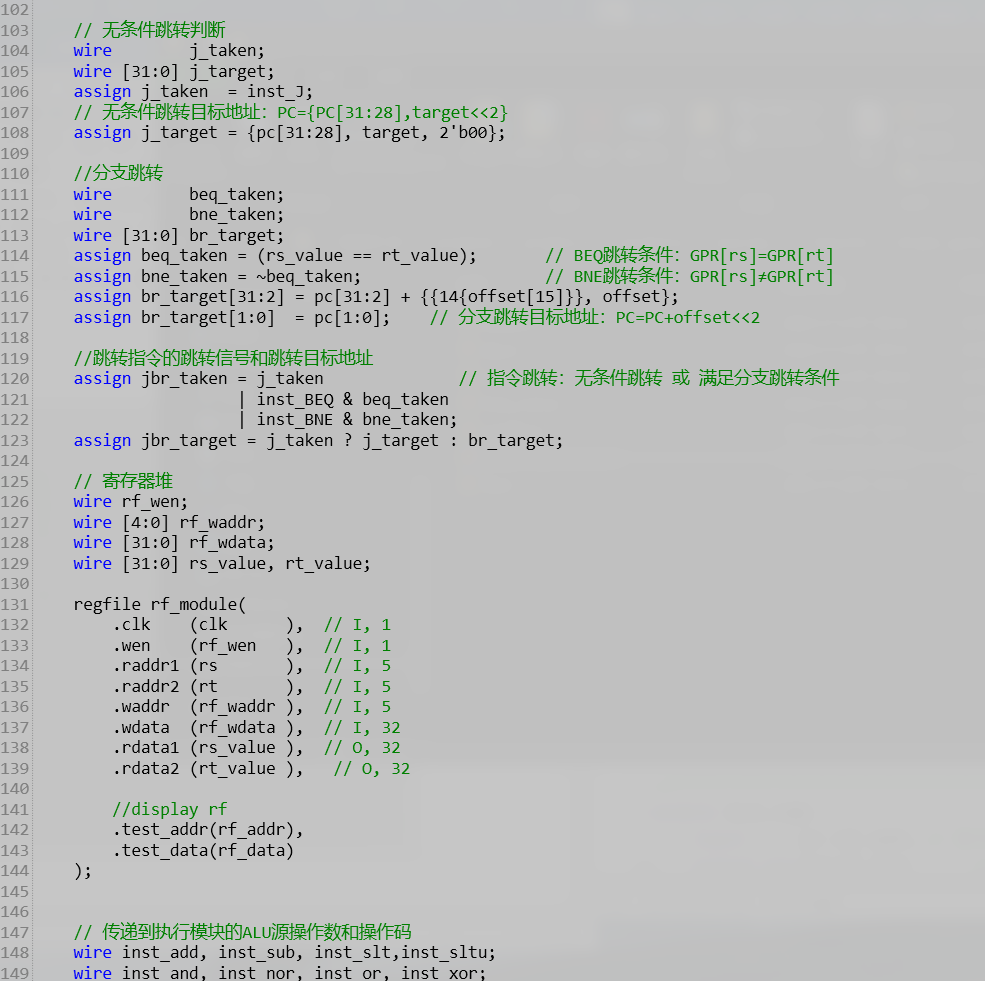
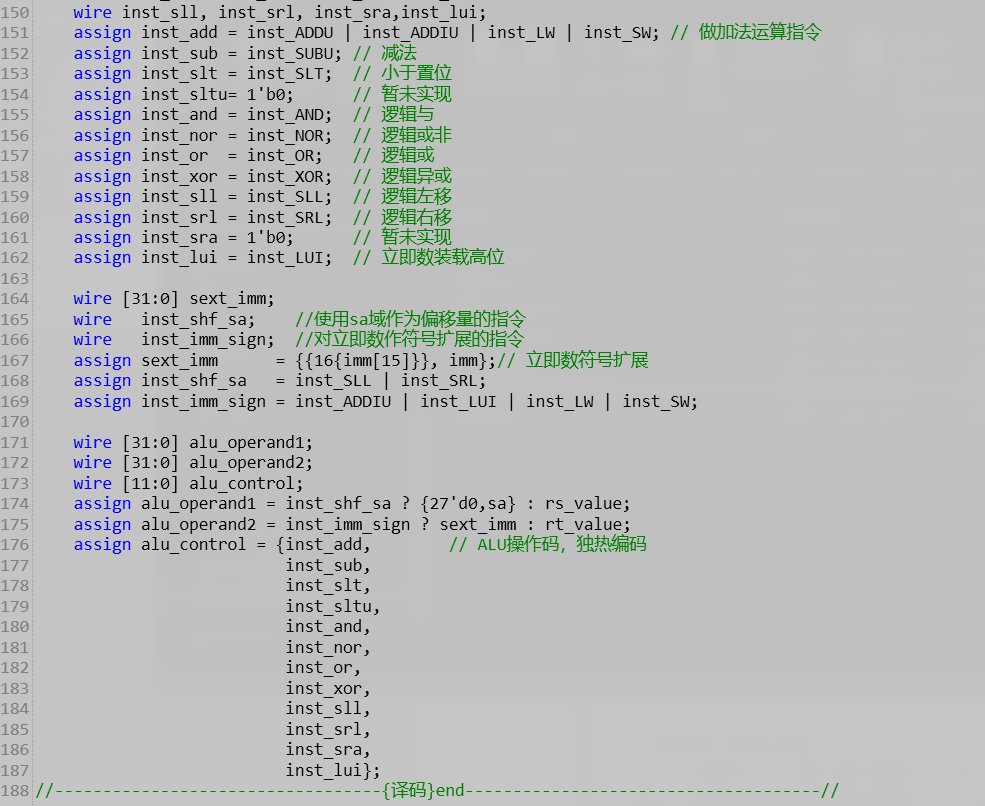
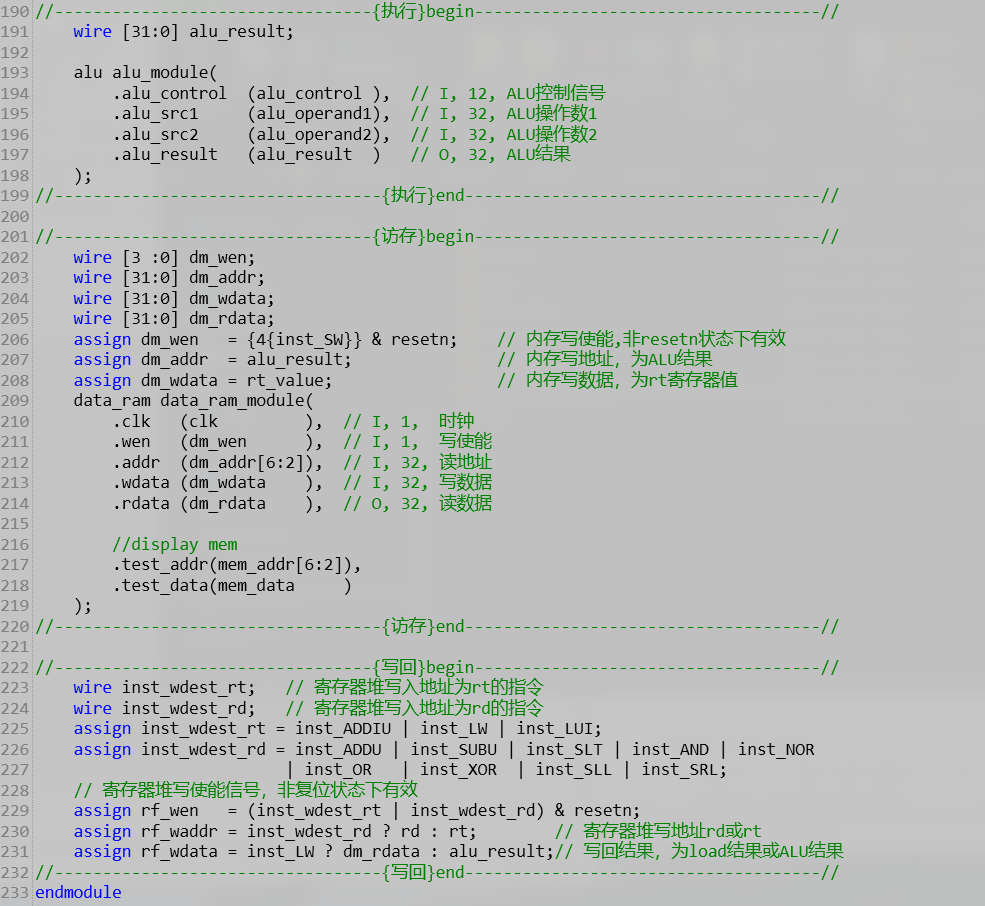
5、对设计文件进行仿真，通过输入激励信号，查看仿真后的输出信号，判断是否符合设计 要求。

1. **verilog程序**

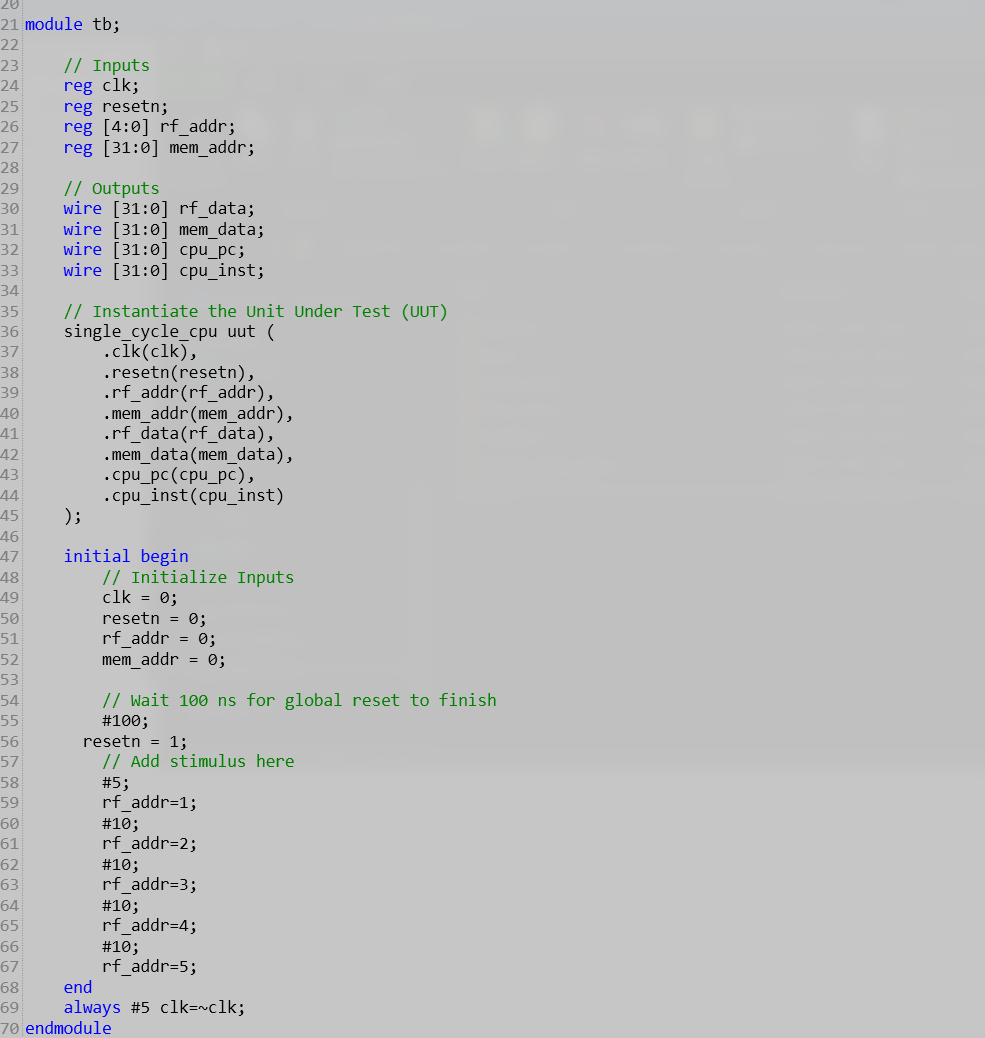
**1.inst\_rom.v**



**2. single\_cycle\_cpu.v**

**3.tb.v**

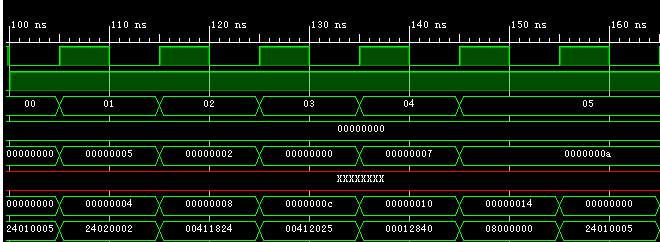


**六、实验结果**

**指令：**

|  |  |
| --- | --- |
| 1. addiu $1 , $0, #5 | (0号寄存器的值加5存储到1号寄存器) |
| 2. addiu $2 , $0, #2 | (1号寄存器的值加5存储到1号寄存器) |
| 3. and $3, $2, $1 | (1号寄存器的值按位与2号寄存器的值存储到3号寄存器) |
| 4. or $4 , $2 , $1 | (1号寄存器的值按位或2号寄存器的值存储到3号寄存器) |
| 5. sll $5 , $1 , #1 | (1号寄存器的值左移一位存储到5号寄存器) |
| 6. j 00H | (跳转到地址00(H)) |

**波形图：**

**七、心得体会**

这次实验一开始我并不知道指令如何设计，看了源代码后一点一点自己修改测试后才明白指令的结构和各种指令命令的编码，但由于时间限制，并没有实现较多的指令，实现的指令单一且简单，即使如此，我也有很多的收获，对单周期CPU指令执行过程有了更加深刻的理解。