



**计算机组成原理课程设计**

个人实验报告

|  |  |
| --- | --- |
| 学 院 | 计算机学院 |
| 专 业 | 计算机科学与技术 |
| 指导老师 | 王娟 |
| 姓 名 | 周一鸣 |
| 联系方式 | 2633866263@qq.com |

二O二二 年 八 月

目 录

[第一章 项目简述 1](#_项目简述)

[第二章 设计目的 1](#_设计目的)

[第三章 设计环境 1](#_设计环境)

[第四章 设计原理及内容 1](#_设计原理及内容)

[4.1 数据通路 2](#_Toc84763292)

[4.2 控制逻辑 2](#_Toc84763293)

[第五章 设计与实现 4](#_Toc84763294)

[第六章 测试 5](#_Toc84763295)

[第七章 问题及解决方法 7](#_Toc84763296)

[第八章 心得体会及总结 7](#_Toc84763297)

[第九章 参考文献有价值的资源推荐 7](#_参考文献有价值的资源推荐)

# 项目简述

利用Verilog语言，设计并实现一个单周期CPU，并支持以下MIPS指令子集：Add, Addi, Addiu, Beq, J, Lui, Lw, Sw, Addu（随机抽取的指令）。

# 设计目的

利用Verilog语言，设计并实现一个支持MIPS指令子集：Lui，Addiu，Add，Lw，Sw，Beq，J，以及一条随机抽取的指令的单周期CPU，给出详细的测试程序，完成实验报告以及提交物

# 设计环境

|  |  |
| --- | --- |
| 操作系统 | Windows10 |
| 编程语言 | Verilog HDL |
| EDA工具 | Vivado2019.2 |
| 汇编语言 | MIPS |
| 汇编程序编辑器 | mars4\_5 |

# 设计原理及内容

实验覆盖了如下8条指令，其中ADDU是随机挑选的指令。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 指令 | opcode | function | 功能 |
| 1 | ADD | 000000 | 100000 | rd=rs+rt |
| 2 | ADDIU | 001001 | / | rd=rs+im（无符号数） |
| 3 | ADDU | 000000 | 100001 | rd=rs+rt (无符号数) |
| 4 | BEQ | 000100 | / | PC=(rs==rt)? PC+im<<2 : PC |
| 5 | LUI | 001111 | / | rt=im\*65536 |
| 6 | J | 000010 | / | PC={(PC+4)[31,28],addr,00} |
| 7 | LW | 100011 | / | LW rt, offset(base) |
| 8 | SW | 101011 | / | SW rt, offset(base) |

表4.1 8条指令

## 数据通路

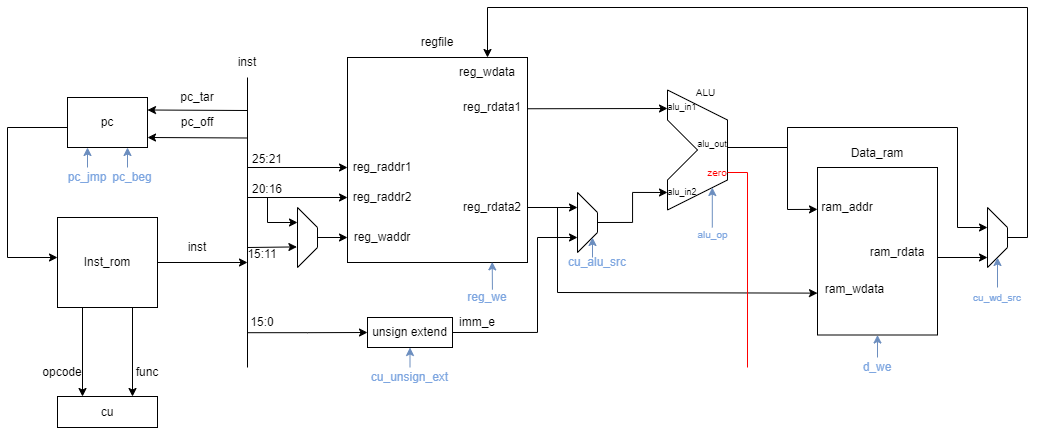


图4.1 单周期CPU数据通路

其中蓝色表示大部分由CU模块控制的控制信号，红色表示ALU计算是否为0信号，其余信号说明见下一小节。

## 控制逻辑

控制信号：

* **pc\_jmp**：表示pc是否执行JMP跳转，0表示不执行，1表示执行。
* **pc\_beq**：表示pc是否执行BEQ跳转，0表示不执行，1表示执行；根据之后的cu\_br和alu\_zero信号的与值判断是否执行BEQ跳转。
* **reg\_we**：通用寄存器写使能，0表示不能写，1表示能写。
* **alu\_op**：ALU执行的操作，0表示add，1表示sub，2表示lui，3表示null。
* **d\_we**：数据存储器写使能，0表示不能写，1表示能写。
* **cu\_alu\_src**：ALU第一个操作数来源，0-寄存器，1-立即数；当指令为add、addu、beq、jmp时为0。
* **cu\_wd\_dst**：写入的目的寄存器地址，0-rt对应寄存器，1-rd段对应寄存器；当指令为add和addu时为1。
* **cu\_unsign\_ext**：表示是否进行无符号扩展，0表示带符号扩展，1表示无符号扩展。
* **cu\_wd\_src**：写入寄存器的数据的来源，0表示ALU的运算结果，1表示从数据存储器读取的数据。
* **cu\_br**：表示指令是否为BEQ，0表示否，1表示是。
* **alu\_zero**：表示ALU运算结果是否为0，0表示否，1表示是。

数据信号：

* **pc\_val**：当前指令地址。
* **pc\_off**：JMP指令的目标地址。
* **pc\_tar**：BEQ指令的偏移地址。
* **reg\_ra1**：寄存器组选择寄存器信号1。
* **reg\_ra2**：寄存器组选择寄存器信号2。
* **reg\_wa**：写寄存器的选择寄存器信号 。
* **reg\_wd**：写寄存器的数据。
* **reg\_rd1**：选择信号1对应的寄存器的值。
* **reg\_rd2**：选择信号2对应的寄存器的值。
* **alu\_in1**：ALU输入数据1。
* **alu\_in2**：ALU输入数据2。
* **alu\_out**：ALU计算结果。

# 设计与实现

实验中设计并实现了以下5个模块：

1. cpu：顶层模块，负责连接各个下层模块。
2. regfile：寄存器模块，用于处理对通用寄存器的读写。
3. control：控制模块，根据指令的opcode和func字段，确定指令的类型及功能，并产生相应的控制信号。其中部分指令和信号的关系采用掩码的方式实现。
4. alu：计算模块，进行算术运算，并根据计算结果产生可能的控制信号。
5. pc：存储当前指令地址，根据控制信号进行JMP跳转、BEQ跳转等操作。

此外，部分指令所需的符号扩展在CPU模块中使用组合电路进行实现。

各模块接口定义如下：

cpu(

    input       rstn,

    input       clk,

    output [31:0] inst\_rom\_addr,

    input  [31:0] inst\_rom\_rdata,

    output [31:0] data\_ram\_addr,

    output [31:0] data\_ram\_wdata,

    output        data\_ram\_wen,

    input  [31:0] data\_ram\_rdata

);

pc(

    input       clk,

    input       rst,

    input       pc\_jmp,

    input       pc\_beq,

    //beq

    input   [31:0] pc\_off,

    //jmp

    input   [31:0] pc\_tar,

    output  [31:0] pc\_val

);

regfile(

    input  wire       clk,

    input  wire       rst,

    input  wire[4:0]  reg\_raddr1,

    input  wire[4:0]  reg\_raddr2,

    input  wire       reg\_wen,

    input  wire[4:0]  reg\_waddr,

    input  wire[31:0] reg\_wdata,

    output wire[31:0] reg\_rdata1,

    output wire[31:0] reg\_rdata2

);

alu(

    input  wire       clk,

    input  wire       rst,

    input  wire[3:0]  alu\_op,

    input  wire[31:0] alu\_in1,

    input  wire[31:0] alu\_in2,

    output reg [31:0] alu\_out,

    output wire       alu\_zero

);

control(

    input   [5:0]   opcode,

    input   [5:0]   func,

    output  [3:0]   cu\_cA,

    output          cu\_jmp,

    output          cu\_br,

    output          d\_we,

    output          reg\_we,

    output          cu\_alu\_src,

    output          cu\_wd\_dst,

    output          cu\_unsign\_ext,

    output          cu\_wd\_src

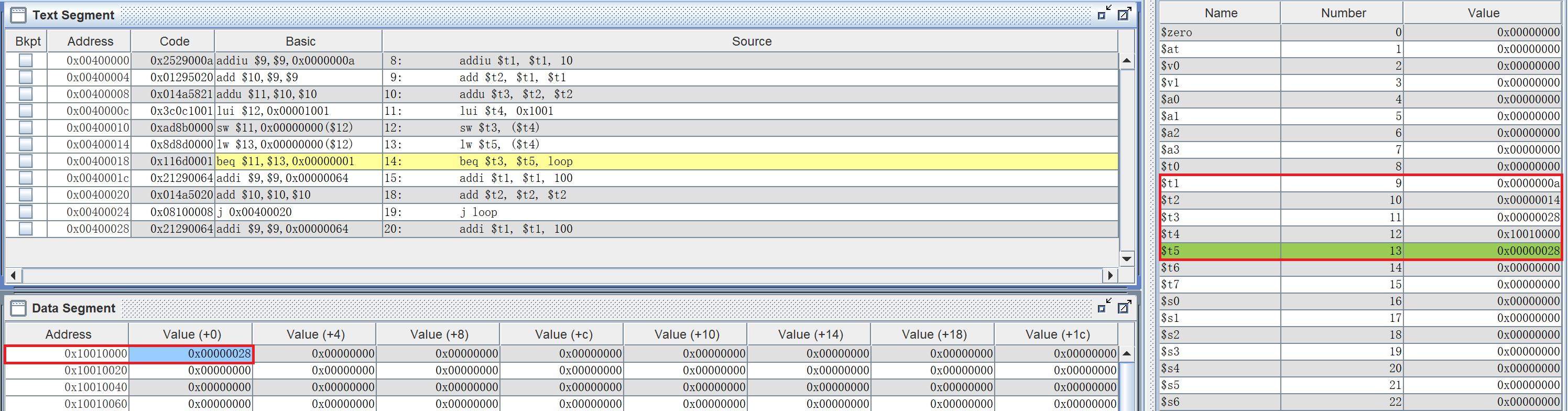
);

# 测试

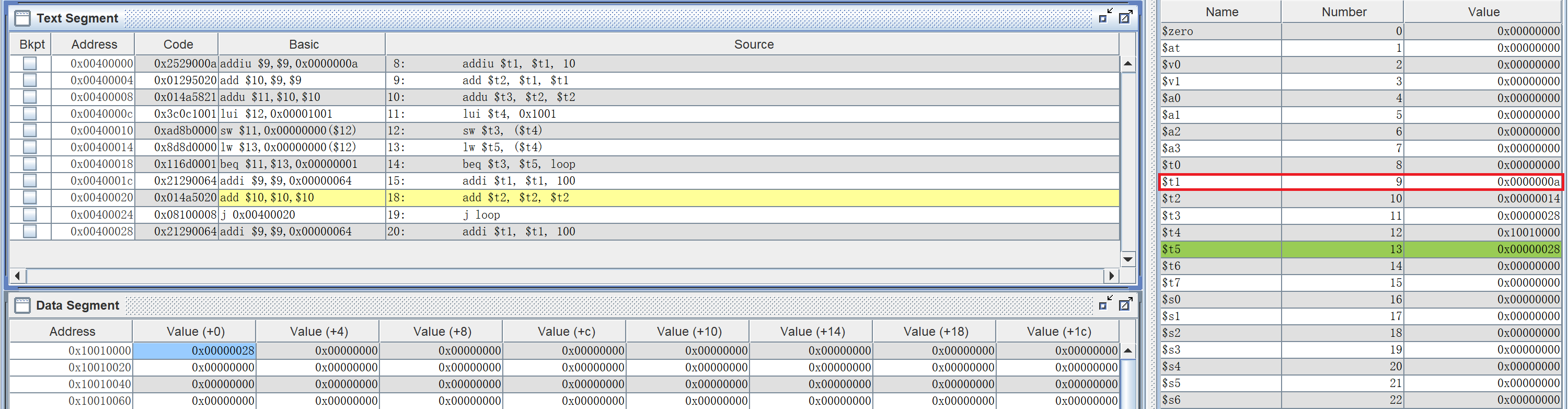
编写的汇编程序使用了MIPS汇编设计的测试代码，具体的实现和验证方法均已在MIPS汇编设计报告中说明，此处仅展示模拟验证与CPU仿真测试的对比结果。

首先在mars中进行验证。

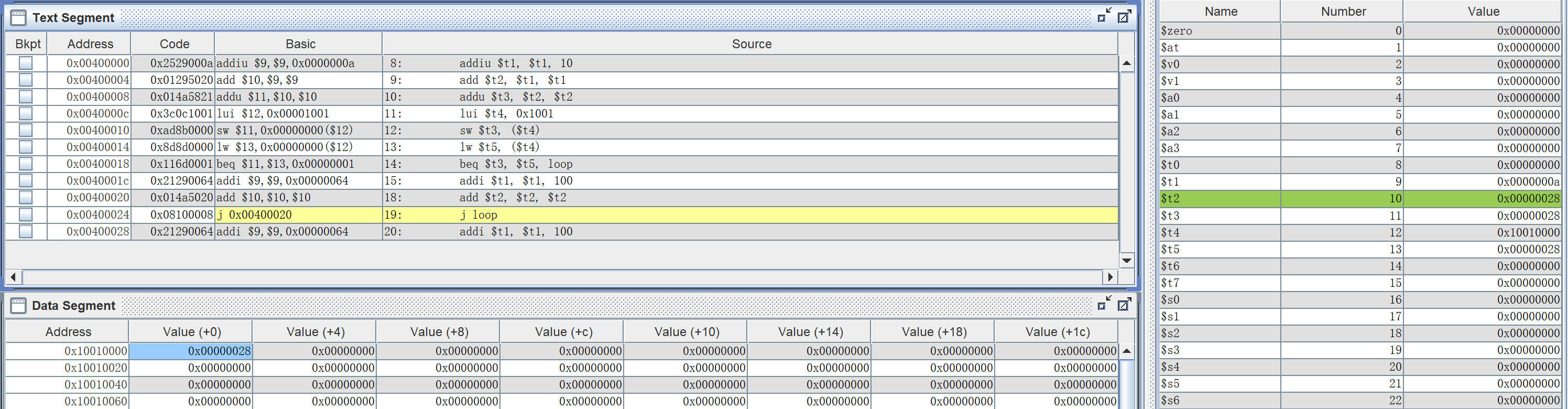
1. 前6条指令的执行结果正确，寄存器和内存数据段内容如下。



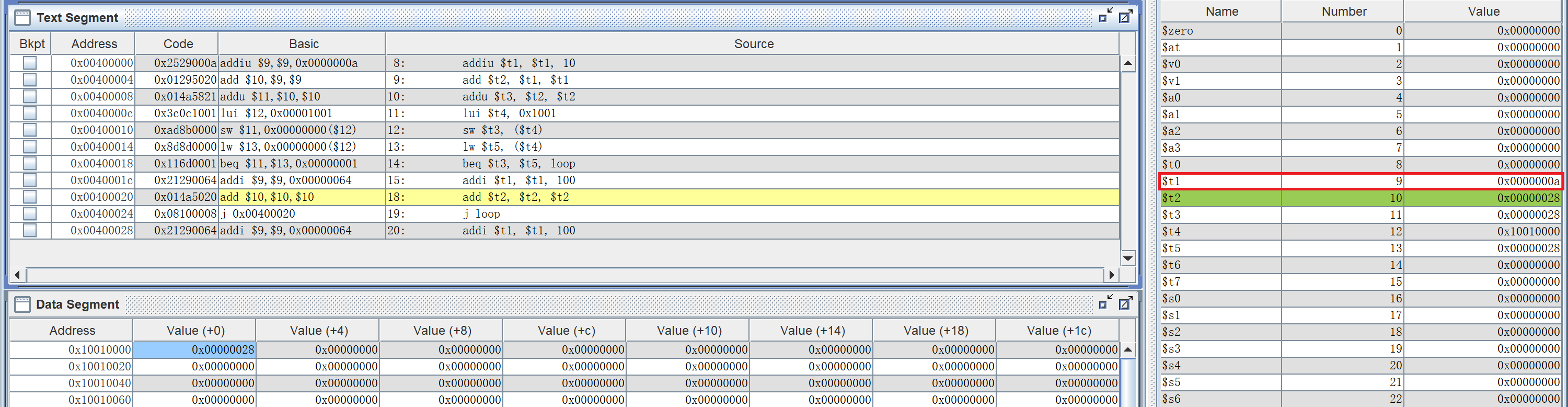
1. 成功跳过了addi $t1, $t1, 100，寄存器t1的值没有改变。



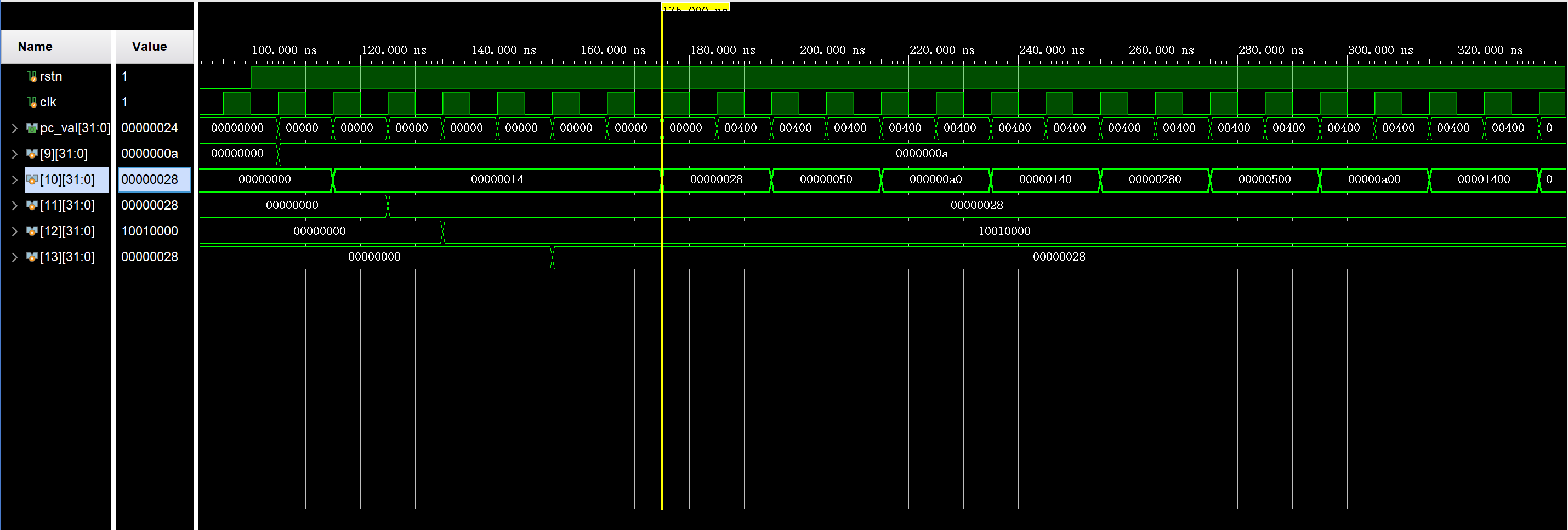
1. 寄存器t2的值倍加。



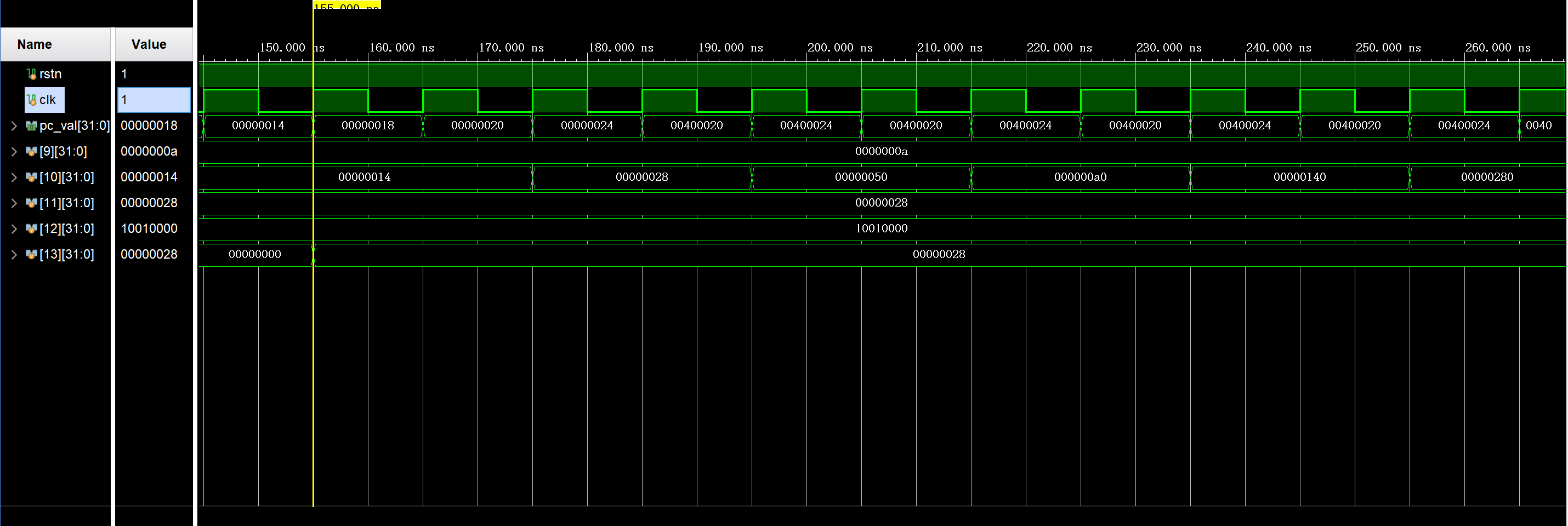
1. 成功跳过最后一条addi指令。



将测试代码转换为机器码，并写入CPU的IP核进行仿真测试。寄存器$t1-$t5和$pc分别对应regfile的regs[9]-regs[13]和pc的pc\_val。



然后开始无限循环，寄存器regs[9]开始无限倍加。



综上，实验结果符合预期，代码实现无误。

# 问题及解决方法

不同指令对操作数的扩展有不同的要求。ADDIU指令和ADDU指令采用无符号扩展，其他指令则采用符号扩展。因此，控制模块中判断指令类型并告知顶层CPU模块扩展方式，由CPU对操作数进行相应扩展。

# 心得体会及总结

相对流水线CPU而言，单周期CPU的结构十分简单，但是由于各模块之间的接口和信号繁多，整体逻辑上略微复杂，在编码初期出现了大量奇奇怪怪的bug。另外Verilog语言的编程想法和常用的高级语言差别较大，以至于刚上手的时候感觉十分别扭。

# 参考文献有价值的资源推荐

无