计算机系统结构实验报告

Lab03

简单的类 MIPS 单周期处理器部件实现

——Ctr、ALU 的实现

姓　　名：　徐薪

学　　号：　519021910726

日　　期：　2021 年 6 月 09 日

摘 要

本实验要求在 Vivado 集成设计环境下实现简单的类 MIPS 单周期处理器部件主控制器（Ctr）、运算单元控制器（ALUCtr）以及算术逻辑单元（ALU）。他们分别用于产生处理器所需的控制信号，以及根据控制信号计算出运行结果。本实验通过软件仿真的方式进行结果验证。

目录

1. 实验概述

1.1 实验内容

在集成设计环境 Vivado 2018.3 下，实现简单的类 MIPS 单周期处理器部件：主控制器（Ctr）、运算单元控制器（ALUCtr）以及算术逻辑单元（ALU），并通过软件仿真的方式进行结果验证。

1.2 实验目的

（1）理解 CPU 主控制器，ALU 的原理；

（2）主控制器 Ctr 的实现；

（3）运算单元控制器 ALUCtr 的实现；

（4）ALU 的实现；

（5）使用功能仿真。

2. 原理分析

2.1 主控制器 Ctr 的设计

2.1.1 Ctr 模块描述

主控制器单元（Ctr）的输入为 MIPS 指令的 opCode 字段。Ctr 的作用是将操作码译码，并向运算单元控制器（ALUCtr）、数据内存（Data Memory）、寄存器（Register）、数据选择器（MUX）等部件输出正确的控制信号。

Ctr 的工作模式如下所示：

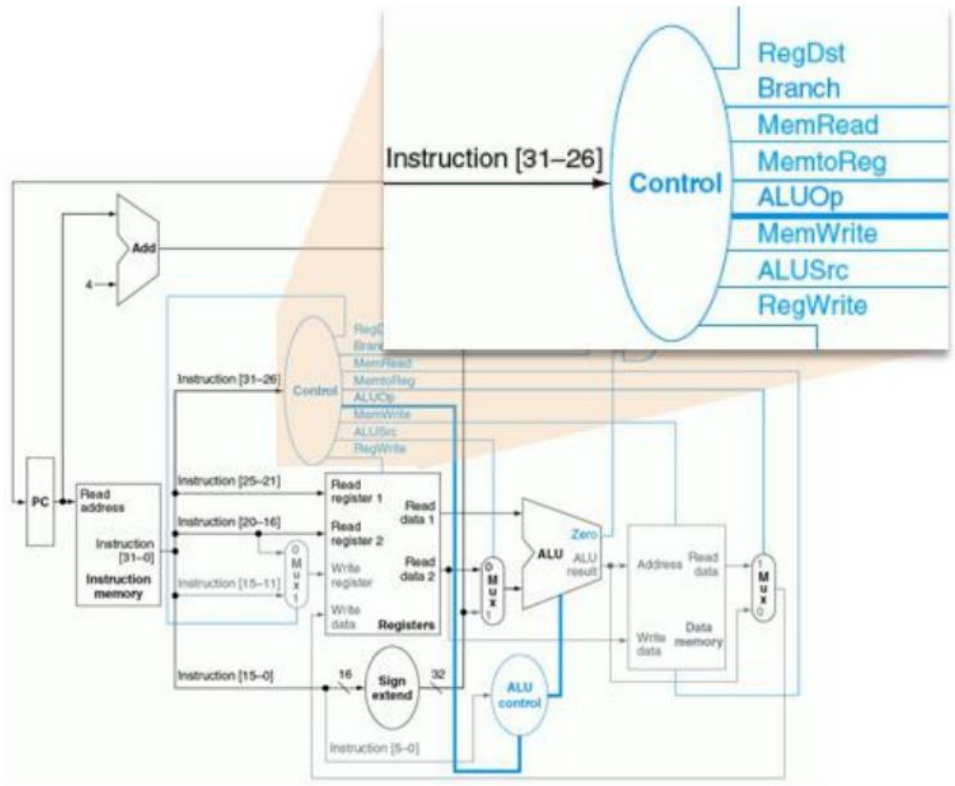


图 1. 主控制模块的 IO 定义

其中，根据输入的指令 instruction[31:26]，可以将 MIPS 指令做如下分类：

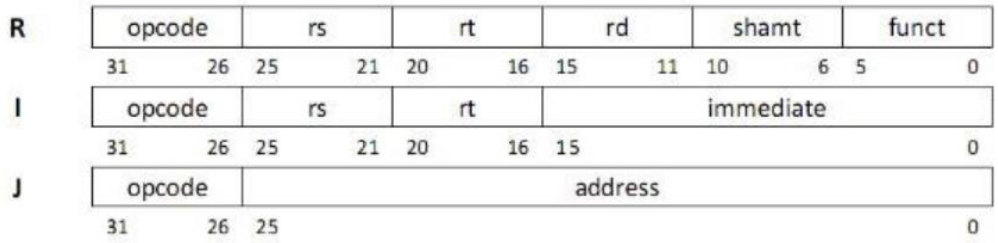


图 2. MIPS 基本指令格式

Ctr 输出的控制信号的含义为：

信号名称	具体含义
RegDst	目标寄存器选择（0：rt，1：rd）
Branch	跳转使能信号（高电平有效）

MemRead	内存读使能（高电平有效）
MemtoReg	内存读取内容写回寄存器信号（高电平有效）
ALUOp	ALU 需执行的操作类型
MemWrite	内存写使能（高电平有效）
ALUSrc	ALU 第二操作数选择信号（0：rt，1：立即数）
RegWrite	寄存器写使能（高电平有效）

表 1. 主控制模块输出的控制信号类型

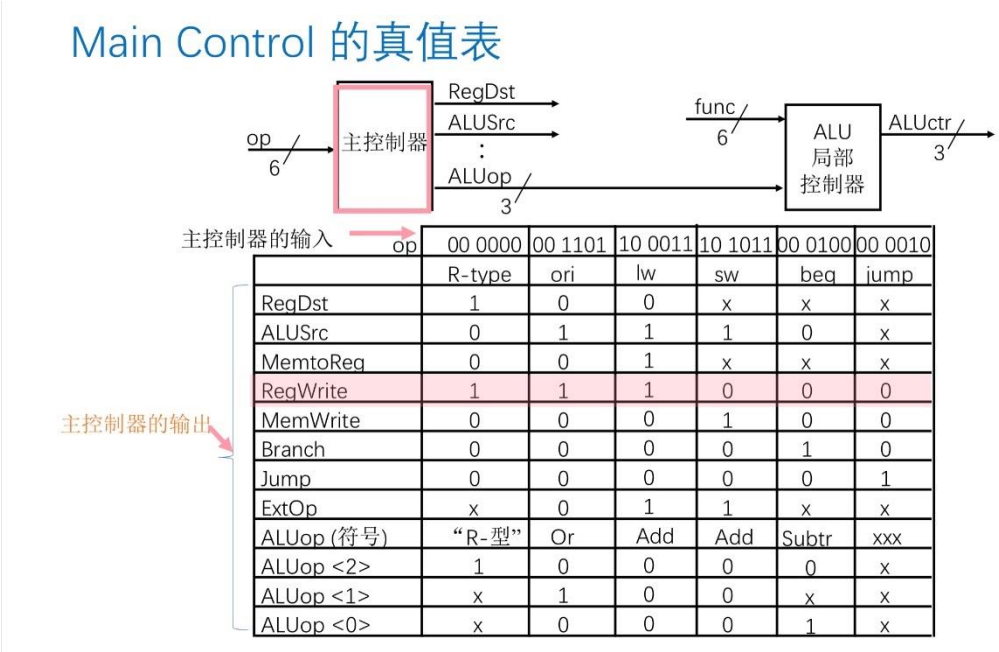
2.1.2 Ctr 模块编码与译码

MIPS 指令主要有 R 型、I 型与 J 型之分，可通过 opCode 进行判别。

指令类型	opCode
R 型：add，sub，and，or，slt	000000
I 型：lw	100011
I 型：sw	101011
I 型：beq	000100
J 型：j	000010

表 2. MIPS 指令的 opCode 编码规则

主控制器 Ctr 再将 opCode 转化为控制信号的译码规则如下图所示：



	00 0000 R-type	00 1101 ori	10 0011 lw	10 1011 sw	00 0100 beq	00 0010 jump
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop (符号)	“R-型”	Or	Add	Add	Subtr	xxx
ALUop <2>	1	0	0	0	0	x
ALUop <1>	x	1	0	0	x	x
ALUop <0>	x	0	0	0	1	x

图 3. 主控制模块的译码规则

2.2 运算单元控制器模块 ALUCtr

2.2.1 ALUCtr 模块描述

ALU 的控制器模块(ALUCtr）是根据主控制器输出的控制信号 ALUOp 来判断指令类型的。

特别地，对于 R 型指令，由于一个 opCode 可能对应多种运算指令，单凭 opCode 译码而获得 的 ALUOp 尚不足以确定 ALU 操作类型。因此 R 型指令还需根据指令的后 6 位（funct）进一步区分。

ALUCtr 需综合这两处输入，以得到正确的控制信号来控制 ALU 执行正确的操作。

其中 ALUCtr 的工作原理如下图所示：

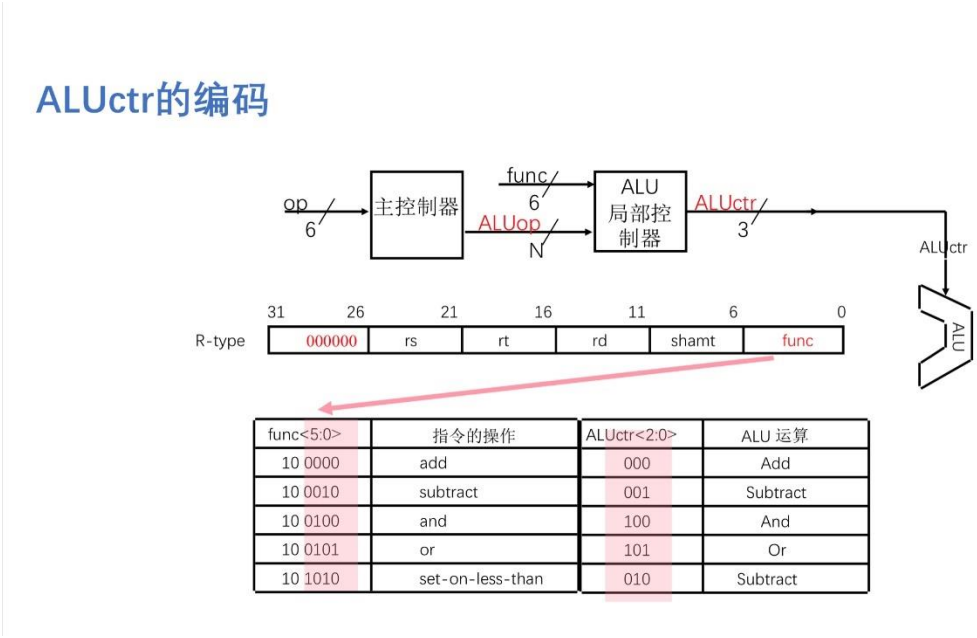


图 4. ALUCtr 的编码

2.2.2 ALUCtr 模块编码与译码

ALUCtr 将 ALUOp 以及 funct 翻译为 ALUctrOut 信号：

图 5. ALUCtr 的 译码规则

2.3 算术逻辑单元 ALU

2.3.1 ALU 模块描述

算术逻辑单元 ALU 根据 ALUCtr 信号将两个输入执行对应的操作，ALURes 为输出结果。若做减法操作，当 ALURes 结果为 0 时，则 Zero 输出置为 1。

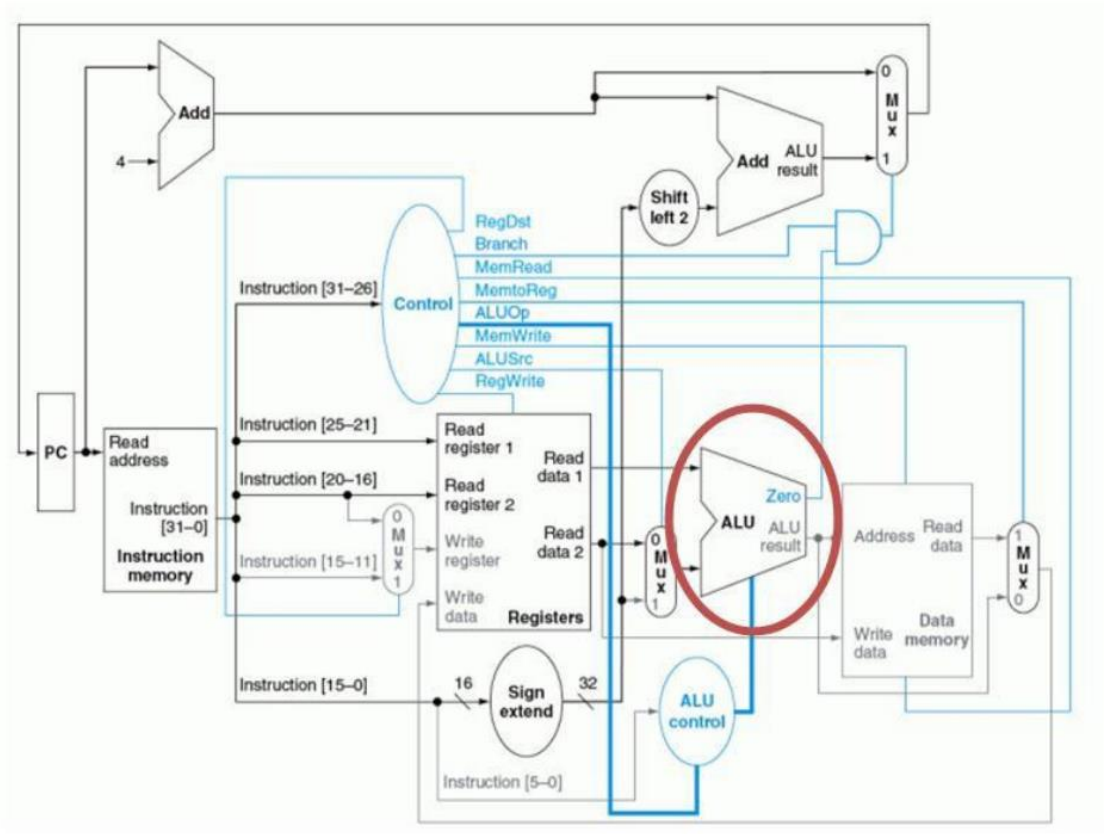


图 6. ALU 模块

2.3.2 ALU 模块编码与译码

ALUctrOut 对应不同的 ALU 功能：

ALUctrOut[3:0]	ALU 功能
0000	与运算
0001	或运算
0010	加法运算

0110	减法运算
0111	小于时置位

表 3.ALUctrOut 编码规则

3. 功能实现

3.1 主控制器模块 Ctr

3.1.1 Ctr 模块实现代码

```
module Ctr(

    input [5:0] opCode,
    output regDst,
    output aluSrc,
    output memToReg,
    output regWrite,
    output memRead,
    output memWrite,
    output branch,
    output [1:0] aluOp,
    output jump
    );

reg RegDst;
reg ALUSrc;
reg MemToReg;
reg RegWrite;
reg MemRead;
reg MemWrite;
reg Branch;
reg [1:0] ALUOp;
reg Jump;

always @(opCode)
begin
    case(opCode)
    6'b000000: //R type
    begin
        RegDst = 1;
        ALUSrc = 0;
        MemToReg = 0;
        RegWrite = 1;
        MemRead = 0;
        MemWrite = 0;
        Branch = 0;
        ALUOp = 2'b10;
        Jump = 0;
    end

    6'b100011: //I type:lw
    begin
        RegDst = 0;
        ALUSrc = 1;
        MemToReg = 1;
        RegWrite = 1;
        MemRead = 1;
        MemWrite = 0;
        Branch = 0;
        ALUOp = 2'b00;
        Jump = 0;
    end
```

```
6'b101011: //I type:sw
begin
    RegDst = 0;
    ALUSrc = 1;
    MemToReg = 1;
    RegWrite = 0;
    MemRead = 0;
    MemWrite = 1;
    Branch = 0;
    ALUOp = 2'b00;
    Jump = 0;
end
  6'b000010: //J type:jump
  begin
      RegDst = 0;
      ALUSrc = 0;
      MemToReg = 0;
      RegWrite = 0;
      MemRead = 0;
      MemWrite = 0;
      Branch = 0;
      ALUOp = 2'b00;
      Jump = 1;
  end

  end
  assign regDst = RegDst;
  assign aluSrc = ALUSrc;
  assign memToReg = MemToReg;
  assign regWrite = RegWrite;
  assign memRead = MemRead;
  assign memWrite = MemWrite;
  assign branch = Branch;
  assign aluOp = ALUOp;
  assign jump = Jump;
endmodule
```

```
6'b000100: //I type:beq
begin
    RegDst = 0;
    ALUSrc = 0;
    MemToReg = 1;
    RegWrite = 0;
    MemRead = 0;
    MemWrite = 0;
    Branch = 1;
    ALUOp = 2'b01;
    Jump = 0;
end

  default:
  begin
      RegDst = 0;
      ALUSrc = 0;
      MemToReg = 0;
      RegWrite = 0;
      MemRead = 0;
      MemWrite = 0;
      Branch = 0;
      ALUOp = 2'b00;
      Jump = 0;
  end
```

3.2 运算单元控制器模块 ALUCtr

3.2.1 ALUCtr 模块实现代码

```
module ALUCtr(

    input [1:0] aluOp,
    input [5:0] funct,
    output [3:0] aluCtrOut

    );

    reg [1:0] ALUOp;
    reg [5:0] Funct;
    reg [3:0] ALUCtrOut;

always @ (aluOp or funct)
begin
    casex ({aluOp,funct})
        8'b00xxxxxx : ALUCtrOut = 4'b0010;
        8'b01xxxxxx : ALUCtrOut = 4'b0110;
        8'b1xxx0000 : ALUCtrOut = 4'b0010;
        8'b1xxx0010 : ALUCtrOut = 4'b0110;
        8'b1xxx0100 : ALUCtrOut = 4'b0000;
        8'b1xxx0101 : ALUCtrOut = 4'b0001;
        8'b1xxx1010 : ALUCtrOut = 4'b0111;
        default:
            begin
                ALUOp = 0;
                Funct = 0;
                ALUCtrOut = 0;
            end
    endcase
end
assign aluCtrOut = ALUCtrOut;
endmodule
```

3.3 算术逻辑单元 ALU

3.3.1 ALU 模块实现代码

```
module ALU(

    input [31:0] input1,
    input [31:0] input2,
    input [3:0] aluCtr,
    output zero,
    output [31:0] aluRes

    );
```

```
reg Zero;
reg [31:0] ALURes;

always @ (input1 or input2 or aluCtr)
begin
    if(aluCtr == 4'b0010) //add
        ALURes = input1 + input2;
    else if (aluCtr == 4'b0110) //sub
    begin
        ALURes = input1 - input2;
        if (ALURes == 0)
            Zero = 1;
        else
            Zero = 0;
    end

else if (aluCtr == 4'b0000) //and
    ALURes = input1 & input2;
else if (aluCtr == 4'b0001) //or
    ALURes = input1 | input2;
else if (aluCtr == 4'b0111)// set on less than
begin
    if (input1 < input2)
        ALURes = 1;
    else
        ALURes = 0;
end
else if (aluCtr == 4'b1100) //nor
begin
    ALURes = input1 | input2;
    ALURes = ~ALURes;
end

        if (ALURes == 0)
            Zero = 1;
        else
            Zero = 0;
    end
    assign zero = Zero;
    assign aluRes = ALURes;
endmodule
```

4. 仿真测试

4.1 主控制器模块 Ctr

4.1.1 Ctr 模块仿真结果

在 Ctr 的测试文件中依次测试了 R type、I type（lw）、I type（sw）、I type（beq）、J type（jump），测试结果如下：

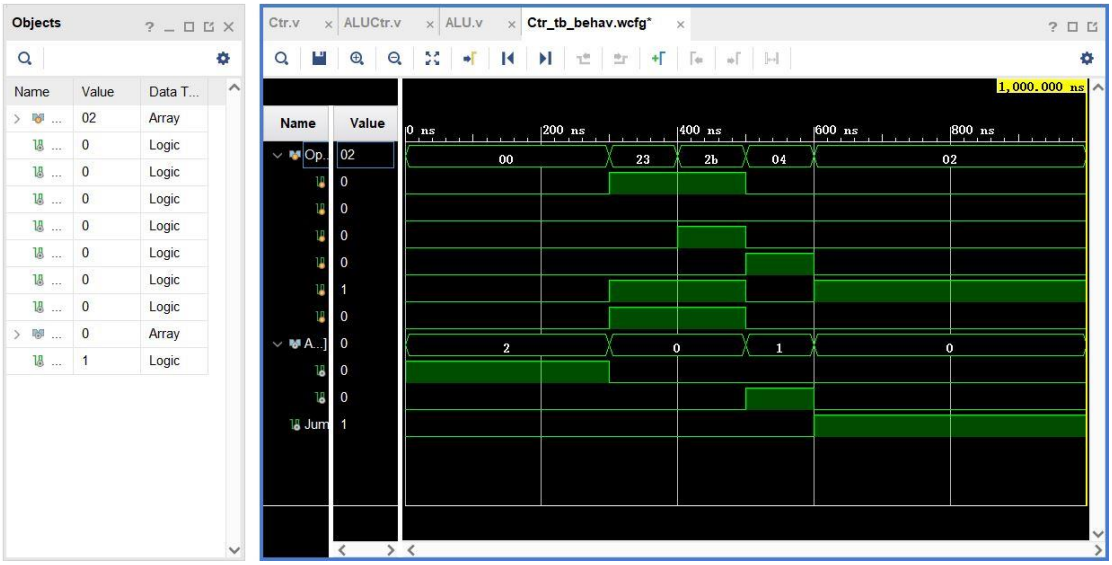


图 7. Ctr 模块仿真结果

检验测试结果，正确。

4.2 运算单元控制器模块 ALUCtr

4.2.1 ALUCtr 模块仿真结果

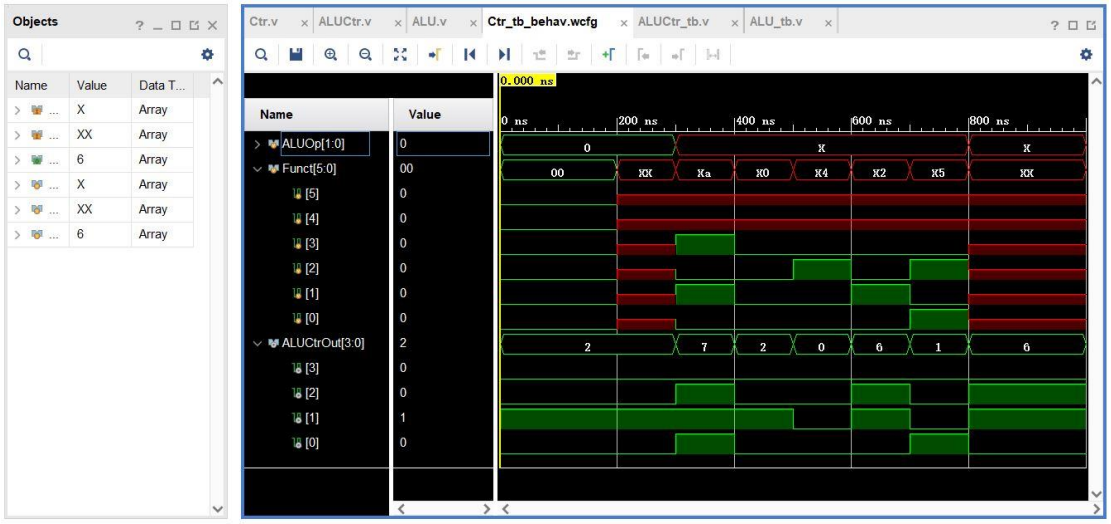


图 8. ALUCtr 模块仿真结果

注：这里的红线表示任意值，即控制信号在该位可以取任意值

检验测试结果，正确。

4.3 算术逻辑单元 ALU

4.3.1 ALU 模块仿真结果

在 ALU 的测试文件中依次测试了 and、or、add、sub、slt、nor 指令，测试结果如下：

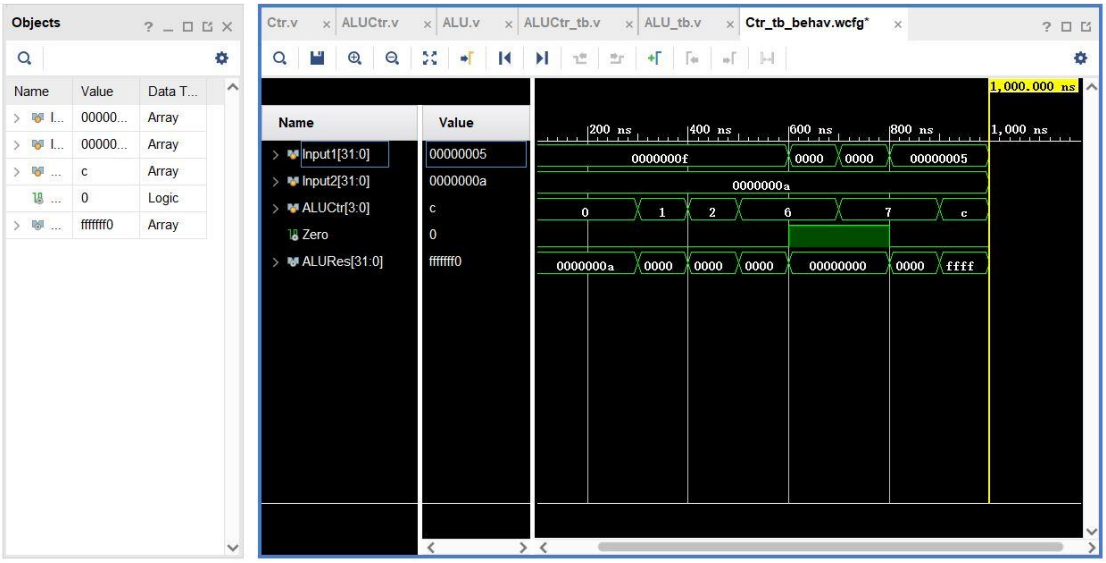


图 9. ALU 模块仿真结果

检验测试结果，正确。

5. 实验总结

5.1 实验评价

综合以上模拟仿真结果，可以验证 Ctr、ALUCtr 以及 ALU 模块均已成功实现：

Ctr 能够有效将不同类型的 MIPS 指令的 opCode 转化为控制信号输出；

ALUCtr 能够结合 Ctr 输出信号与 MIPS 指令 funct 字段输出 ALUctrOut;

ALU 能够根据 ALUctrOut 信号对操作数执行对应的运算操作。

5.2 实验心得

总体来说，这次实验比较简单，只要弄清楚了各个模块的输入信号与输出信号之间的依赖关系，并且用数字逻辑电路正确地表示出来，实验就成功了一大半。

但是，在进行实验的过程中，我还是遇到了一些问题，尤其是在定义 ALUCtr 模块的时候。刚开始我只是简单地把讲义上的 ALUop、funct 与 ALUCtrOut 的关系誊写上去,从而忽略了对情况重叠的讨论，导致仿真结果一直不对。后来在老师的提醒下，才注意到其中一种 ALUop 与 funct 的组合包含了另一组 ALUop 与 funct 的组合。例如在 PDF 实验指导文档里面，ALU 的 SUB 操作对应的 ALUop 与 ALUCtrOut 的组合 x1xxxxxx 就重复包含了 ADD 操作对应的 ALUop 与 ALUCtrOut 的组合 1xxx0000，所以在仿真程序运行时，ALU 不能选择出正确的操作。这个问题的解决方法是减少不确定的 x 的值。因为 x 表示任意值，所以我们可以通过对 x 取不同的特定值来分离这些相互包含的组合。修改程序后再进行模拟仿真，仿真结果正确。

这次的实验锻炼了我的分类思想能力，让人受益匪浅。