Lab03实验报告

FPGA基础实验：简单的类MIPS单周期处理器部件实现–控制器，ALU

518021910489 陈沛宇

一、实验目的

~理解 **CPU** 控制器，**ALU** 的原理。

~主控制器 **Ctr** 的实现。

~运算单元控制器 **ALUCtr** 的实现。

~**ALU** 的实现。

~使用功能仿真。

二、实现步骤

1.主控制模块

模块描述

主控制器单元（**Ctr**）的输入为指令的 **opCode** 字段，操作码经过 **Ctr** 的译码，给**ALUCtr**，**Data Memory**，**Registers**，**Muxs** 等部件输出正确的控制信号。

实现方法

使用**case**语句，**case**语句与C语言语法类似，其中**begin**和**end**字段可以理解为C语言中的**{}**，根据实验指导书上的真值表对case语句进行书写即可实现译码功能。

ctr.v

```
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://lw
        begin
            regDst=0;
            aluSrc=1;
```

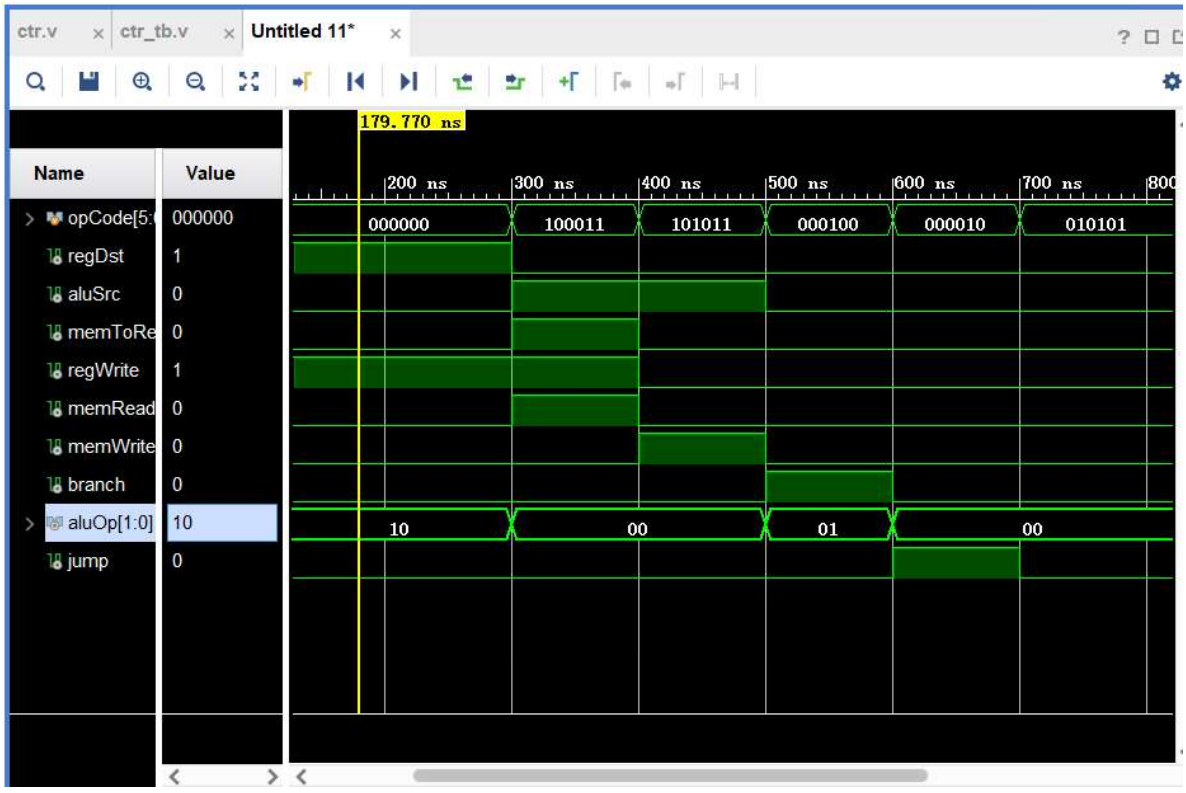
```
            memToReg=1;
            regWrite=1;
            memRead=1;
            memWrite=0;
            branch=0;
            aluOp=2'b00;
            jump=0;
        end
        6'b101011://sw
        begin
            regDst=0;
            aluSrc=1;
            memToReg=0;
            regWrite=0;
            memRead=0;
            memWrite=1;
            branch=0;
            aluOp=2'b00;
            jump=0;
        end
        6'b000100://beq
        begin
            regDst=0;
            aluSrc=0;
            memToReg=0;
            regWrite=0;
            memRead=0;
            memWrite=0;
            branch=1;
            aluOp=2'b01;
            jump=0;
        end
        6'b000010://jump
        begin
            regDst=0;
            aluSrc=0;
            memToReg=0;
            regWrite=0;
            memRead=0;
            memWrite=0;
            branch=0;
            aluOp=2'b00;
            jump=1;
        end
        default:
        begin
            regDst=0;
            aluSrc=0;
            memToReg=0;
            regWrite=0;
            memRead=0;
            memWrite=0;
            branch=0;
            aluOp=2'b00;
            jump=0;
        end
        endcase
    end
```

在激励文件中，设置好不同类型的指令的opCode进行测试即可。

ctr_tb.v

```
initial begin
opCode = 0;
#100
#100 opCode = 6'b000000;
#100 opCode = 6'b100011;
#100 opCode = 6'b101011;
#100 opCode = 6'b000100;
#100 opCode = 6'b000010;
#100 opCode = 6'b010101;
#100;
end
```

测试结果



测试结果逻辑正确，符合预期。

算术逻辑单元（ALU）控制器模块

模块描述

ALU 的控制器模块 （**ALUCtr**）是根据主控制器的 **ALUOp** 来判断指令类型。根据指令的后6 位区分 R 型指令。综合这两种输入，控制 **ALU** 做正确的操作。

实现方法

与主控制模块的实现方法类似，判断**aluOp**和**Funct**结合起来的值，但采用的是**casex**语句，**casex**可以对某几位不确定的情况进行**case**划分。即**casex**表达式与**case**语句的差异是**casex**的语句中可以含有无关项，无关项会带来的问题是需要严格检查**casex**不同情况的顺序，以防止因无关项而带来的错误。

aluctr.v

```
begin
        casex({aluOp,Funct})
            8'b00xxxxxx:aluCtrOut = 4'b0010;
            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;
            8'bx1xxxxxx:aluCtrOut = 4'b0110;
        default:aluCtrOut = 4'b0000;
        endcase
end
```

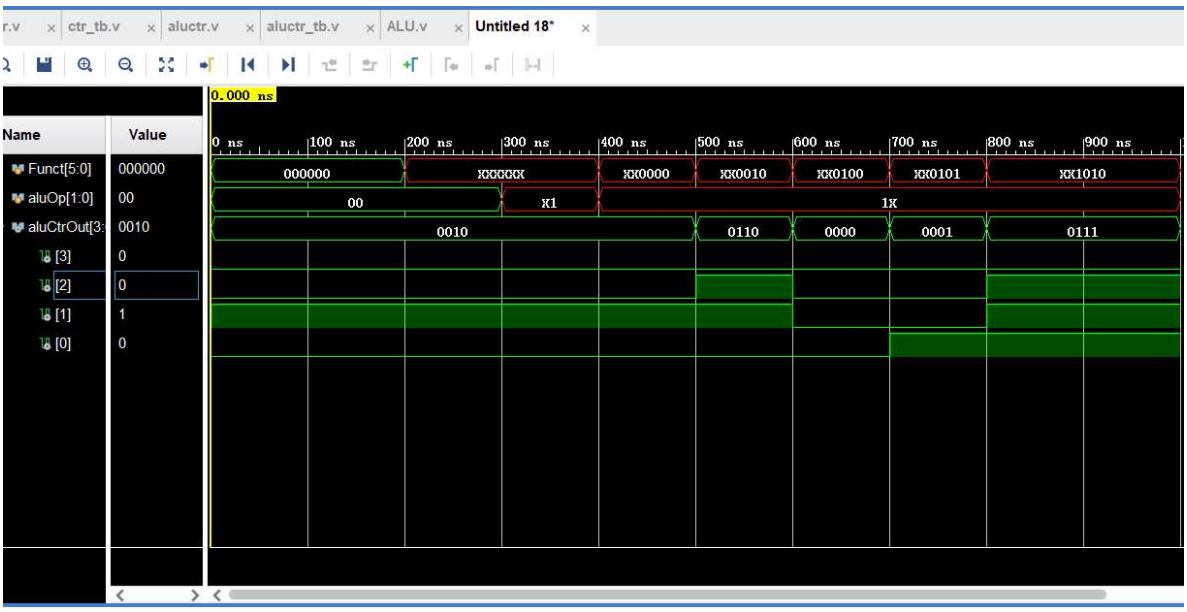
可以看到代码里最后一个**case**放在最后的原因就是为了避免影响前面的判断。

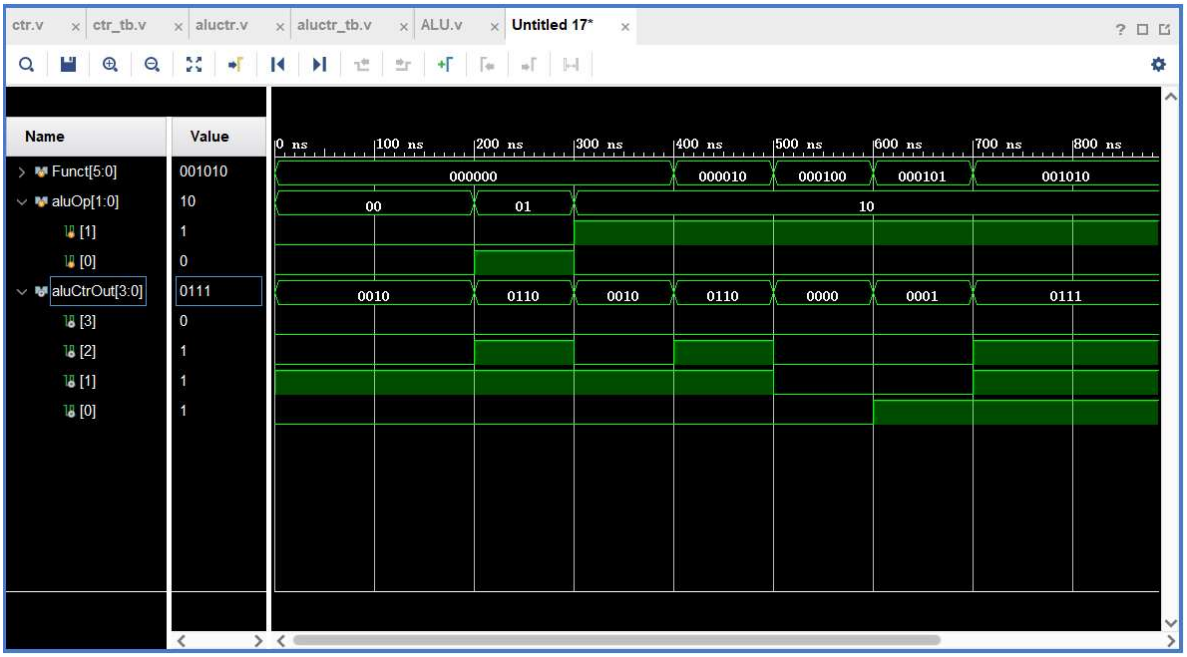
激励文件中对含无关项和不含无关项的数据都进行了测试。

aluctr_tb.v

```
initial begin
    Funct=0;
    aluOp=0;
//含无关项
    #100 Funct=6'b000000;aluOp=2'b00;
    #100 Funct=6'bxxxxxx;
    #100 aluOp=2'bx1;
    #100 Funct=6'bxx0000;aluOp=2'b1x;
    #100 Funct=6'bxx0010;
    #100 Funct=6'bxx0100;
    #100 Funct=6'bxx0101;
    #100 Funct=6'bxx1010;
//不含无关项
    #100 Funct=6'b000000;aluOp=2'b00;
    #100 aluOp=2'b01;
    #100 aluOp=2'b10;
    #100 Funct=6'b000010;
    #100 Funct=6'b000100;
    #100 Funct=6'b000101;
    #100 Funct=6'b001010;
end
```

测试结果





测试结果逻辑正确，符合预期。

ALU模块

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。

实现方法

同样使用**case**语句，可以直观简便的完成该模块的设计，通过判断**aluCtr**信号，进行不同类型的运算并根据运算结果设置**zero**的值即可。

ALU.v

```
always @(input1 or input2 or aluCtr)
    begin
        zero = 0;
        case (aluCtr)
            4'b0010: // add
                begin
```

```
                    aluRes = input1 + input2;
                    if (aluRes == 0)
                        zero = 1;
                end
                4'b0110: // sub
                begin
                    aluRes = input1 - input2;
                    if (aluRes == 0)
                        zero = 1;
                end
                4'b0000: // and
                begin
                    aluRes = input1 & input2;
                    if (aluRes == 0)
                        zero = 1;
                end
                4'b0001: // or
                begin
                    aluRes = input1 | input2;
                    if (aluRes == 0)
                        zero = 1;
                end
                4'b1100: // nor
                begin
                    aluRes = ~(input1 | input2);
                    if (aluRes == 0)
                    zero = 1;
                end
                4'b0111: // set on less than
                begin
                    if (input1 < input2)
                        begin
                            aluRes = 1;
                            zero = 0;
                        end
                    else
                        begin
                            aluRes = 0;
                            zero = 1;
                        end
                end
            endcase
        end
```

激励文件可以按照实验报告书的测试结果截图设置测试值。

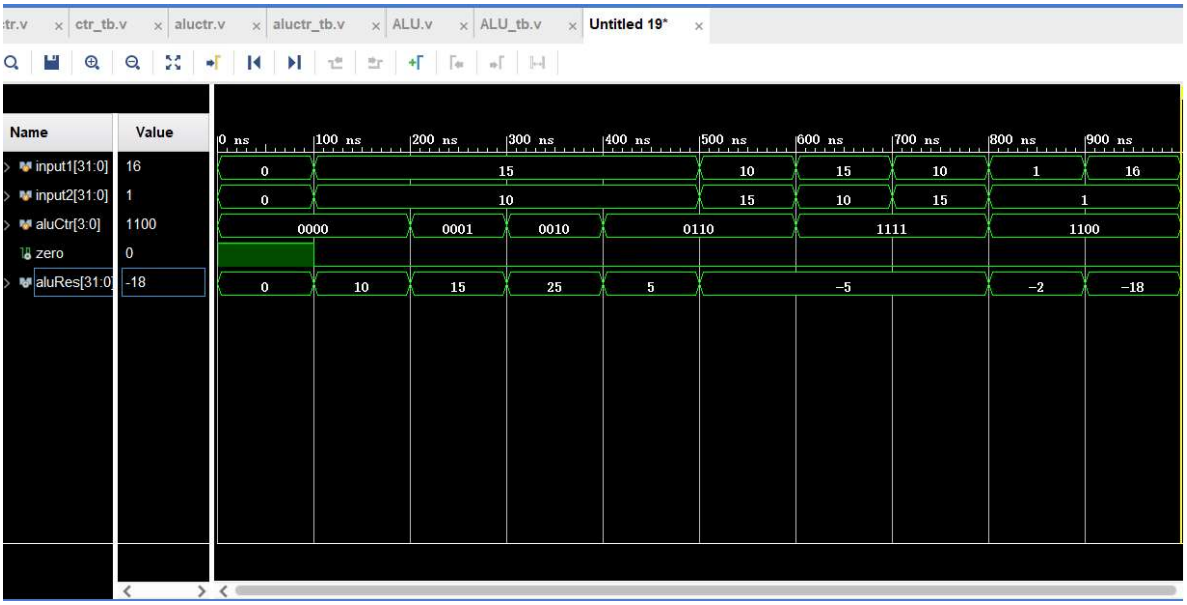
ALU_tb.v

```
initial begin
    input1=0;
    input2=0;
    aluCtr=4'b0000;
    #100 input1=15;input2=10;
    #100 aluCtr=4'b0001;
    #100 aluCtr=4'b0010;
    #100 aluCtr=4'b0110;
    #100 input1=10;input2=15;
```

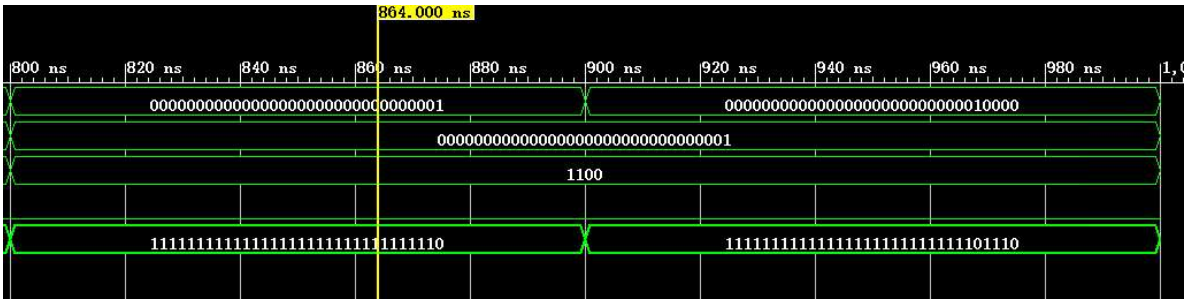
```
    #100 aluCtr=0111;input1=15;input2=10;
    #100 input1=10;input2=15;
    #100 aluCtr=1100;input1=1;input2=1;
    #100 input1=16;

    end
```

测试结果



其中**NOR**运算的结果如图



测试结果逻辑正确，符合预期。

三、总结和心得体会

1.总结

在模块的设计中遇到的最大问题是在进行激励文件测试的时候，出现了诸如input信号的值为X或output信号的值为Z的情况，总结起来的解决方案为：

```
wire output     //output信号在激励文件中要用wire声明
reg input       //input信号在激励文件中要用reg声明
NAME test (     //所有信号都需要实例化
.signal1(signal1),
...
)
```

2.心得体会

本次实验我花费了较多时间在调试上，出现了很多之前两次实验没有遇到的问题。我认为这次花费时间较多的原因是在前两次实验中很多声明性的语句都已经被提前写好，我不求甚解的直接将代码抄到了软件中，对很多声明都没有理解。

这次实验让我对Verilog的语法更加熟悉，在激励文件的编写和测试上也更加熟练，是一次非常有收获的实验，锻炼了我用硬件描述语言的思想来编写模块的能力。同时我也认识到学习一门新的编程语言应该脚踏实地，至少需要了解基本的语法再开始编程，否则会浪费很多时间在走弯路上。