计算机系统结构实验报告 实验3

惠宇龙 518030910059

1 实验概述

1.1 实验描述

简单的类MIPS单周期处理器部件的实现。在本实验中先后实现了控制器、ALU-Ctr 和 ALU 三个基本的部件，也为后面深入设计复杂CPU的实验做好了铺垫。

1.2 实验目的

- 理解 CPU 控制、ALU 的原理
- 实现主控制器、ALU 控制器和 ALU
- 使用功能仿真

2 主控制器Ctr

2.1 主控制器Ctr的设计

主控制单元为一个译码器，其接受指令的 [31:26] 位字段（opCode）作为输入，给 ALU 控制器、数据内存、寄存器和数据选择器输出正确的控制信号。

不同类型的指令会产生不同的控制信号，其控制模块的真值表如下：

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

表1.主控制器真值表

我们需要解析指令类型，并按照真值表的数据，生成正确的控制信号

2.2 主控制器Ctr的实现

将 opCode 作为输入信息，将其他控制信号作为输出信息，定义模块：

```
module ctr(
input [5:0] opcode,
output reg [1:0] ALUOp,
output reg Branch,
output reg Jump,
output reg ALUSrc,
output reg MemWrite,
output reg RegWrite,
output reg MemToReg,
output reg MemRead,
output reg RegDst
);
```

在 Verilog 中，译码器可以使用 case 语句来实现。这里将 `opCode` 的各种可能作为各种 case，在每个 case 的代码块内按照上表所示的控制信号正确输出。

```
always@(opcode)
    begin
        case(opcode)
        6'b000000://R
        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;
```

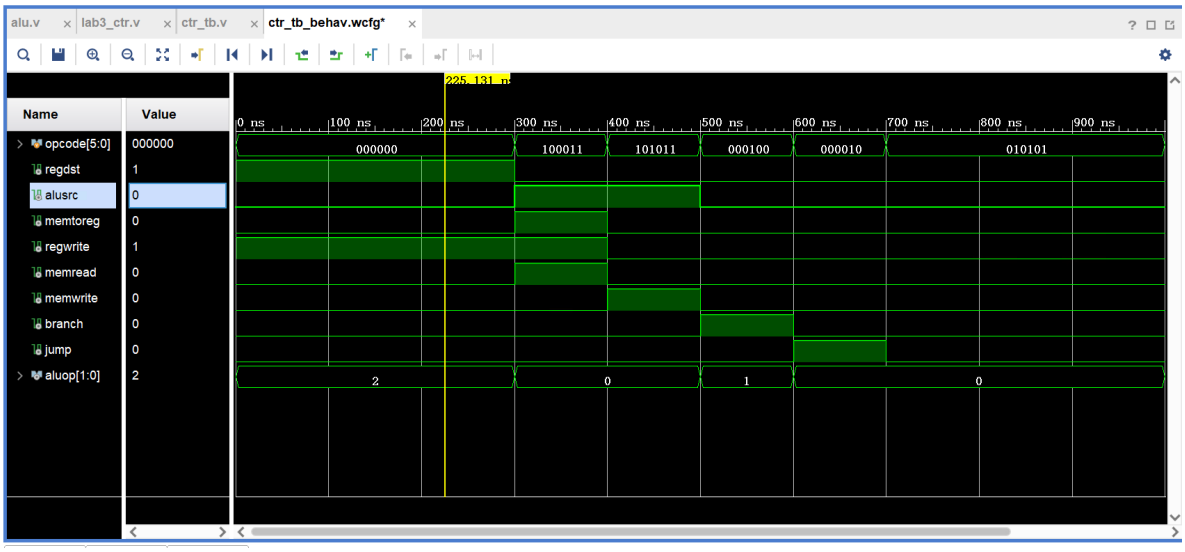
```
                RegWrite=0;
                MemRead=0;
                MemWrite=0;
                Branch=0;
                ALUOp=2'b00;
                Jump=0;
            end
          endcase
        end
```

2.3 主控制器Ctr的验证

我们需要实例化一个Ctr的对象`c0`，并同时将其端口实例化。随后给`opCode`赋多个值，使其模拟不同类的指令下输出的控制信号，激励文件代码如下所示（端口实例化的代码较简单，故在报告中略去）：

```
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;
    end
```

观察最后的输出波形，可以得到下图结果：



经比较可知，该波形与参考书所给波形一致，且与激励文件中opcode理应产生的控制信号一致。

故本实验的主控制器模块实现成功。

3 运算单元控制器ALUCtr

3.1 运算单元控制器ALU-Ctr的设计

ALU 控制单元模块（ALUCtr）根据主控制器的 ALUOp 判断指令类型，从而向 ALU 输出正确的运算控制信号。如果是R 类型指令，则需要根据指令的低 6 位（funct）来进行判断。即ALUCtr 会综合 ALUOp 和 funct 来进行译码，该模块的输入输出真值表如下 : (Operation 一栏即为输出结果)

ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

表2.运算单元控制器真值表

3.2 运算单元控制器ALUCtr的实现

将 ALUOp和 Funct 一起作为输入。为了减少代码书写，将二者合并为一串数字，进行 `casex` 检测。其中， `casex` 是 case 语句的一种变体，可以在 case 表达式中包含无关项。对任意一个输入，是按照模块编写者指定的顺序，寻找第一个满足的 case 输出。因此，一定注意将所有 case 从高位开始，要按照先写不含无关项的 case 再写含无关项的 case 的顺序排列。具体代码如下：

```
module aluctr(
    input [1:0] ALUOp,
    input [5:0] Funct,
    output [3:0] Oper
    );
    reg [3:0]OPER;
    assign Oper=OPER;

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

3.3 运算单元控制器ALUCtr的验证

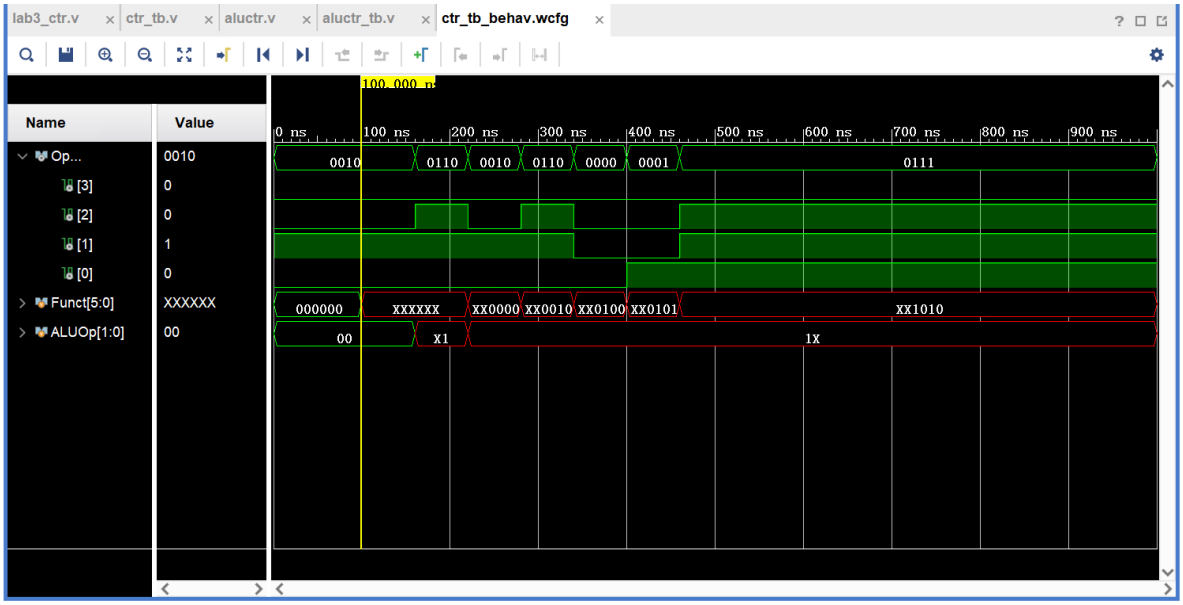
将 ALUOp和 Funct 一起作为输入，并观察波形，其中激励文件的两套代码分别如下所示：

```
initial begin
    ALUOp=0;
    Funct=0;

    #100 Funct=6'bxxxxxx;
    #60
    ALUOp=2'bx1;
    #60
    ALUOp=2'b1x;
    Funct=6'bxx0000;
    #60 Funct=6'bxx0010;
```

```
    #60 Funct=6'bxx0100;
    #60 Funct=6'bxx0101;
    #60 Funct=6'bxx1010;
end
```

该激励文件对应的波形图为：



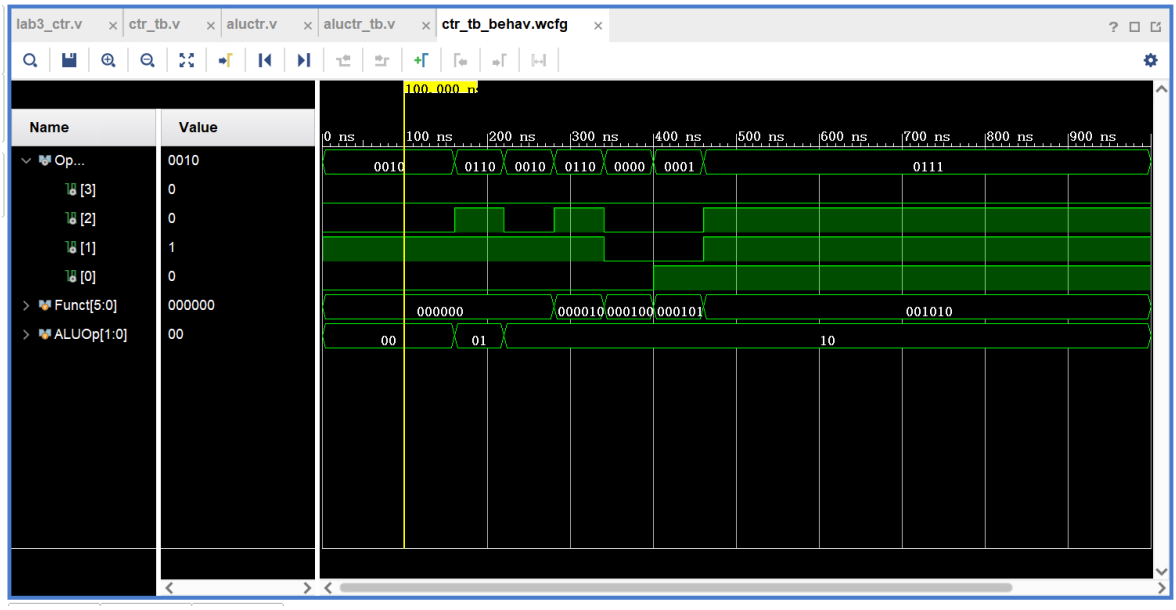
另一组激励文件的代码，如下所示：

```
initial begin
  ALUOp=0;
  Funct=0;

  #160 ALUOp=2'b01;
  #60 ALUOp=2'b10;
  #60 Funct=6'b000010;
  #60 Funct=6'b000100;
  #60 Funct=6'b000101;
  #60 Funct=6'b001010;

end
```

该激励文件对应的波形图为：



综上，两种激励文件都得到了与参考书一致且符合逻辑的波形图，故该模块成功实现。

4 运算单元ALU

4.1 运算单元ALU的设计

根据 ALUCtr 的控制信号，对两个操作数进行某种逻辑或算术运算，将结果输出到 ALURes 中。如果结果为零，将 Zero 设为真。其中，ALUCtr的值与对应的ALU操作满足如下的关系：

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

4.2 运算单元ALU的实现

用case语句解析ALUCtr 信号，并对操作数进行相应的运算。特别的，结果运算完之后，要根据 ALURes 是否为零来设置 Zero，具体代码如下：

```
module alu(
   input [31:0] input1,
   input [31:0] input2,
   input [3:0] aluCtr,
   output reg  zero,
   output reg [31:0] aluRes
    );

 always @(input1 or input2 or aluCtr)
 begin
   case (aluCtr)
    4'b0000: aluRes=input1 & input2;
```

```
    4'b0001: aluRes=input1 | input2;
    4'b0010: aluRes=input1 +input2;
    4'b0110: aluRes=input1-input2;
    4'b0111: aluRes=input1 < input2;
    4'b1100:
      begin
       aluRes= input1|input2;
       aluRes= ~aluRes;
      end
   endcase

   if (aluRes==0) zero=1;
   else zero=0;
 end

endmodule
```

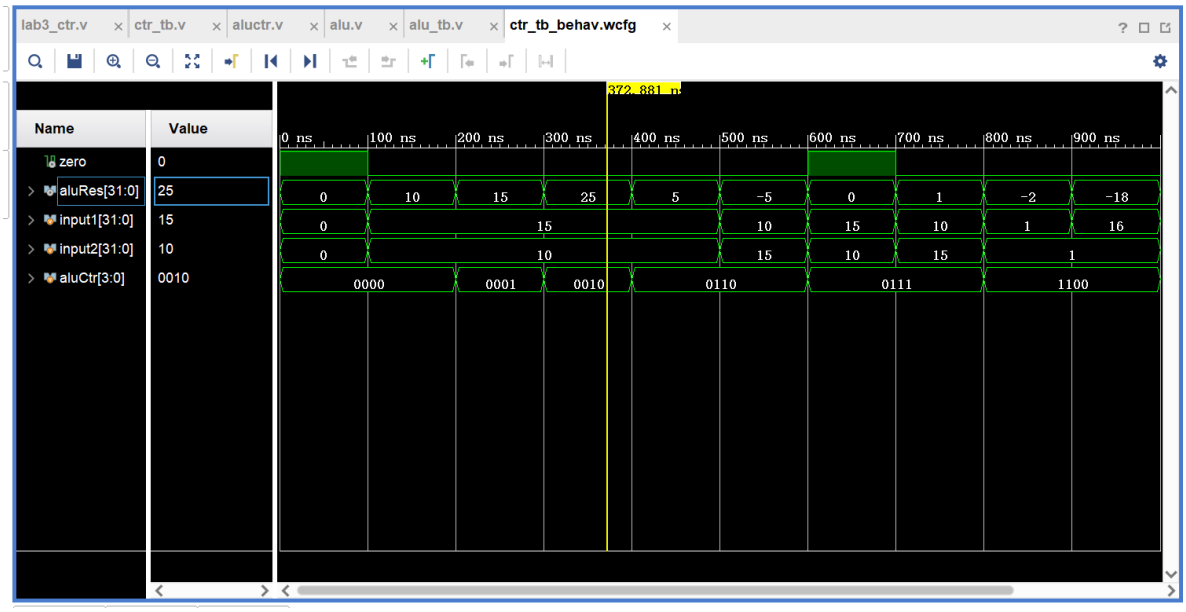
4..3 运算单元ALU的验证

输入两个操作数，并不断改变操作码和操作数来检查模块的正确性。

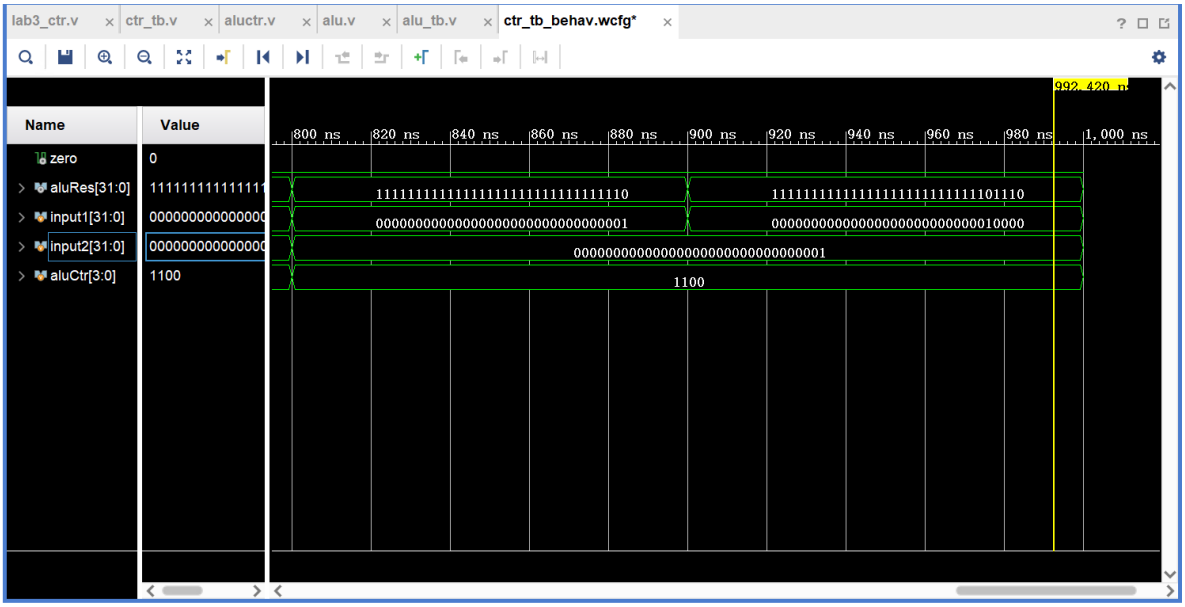
```
 initial begin
    input1=0;
    input2=0;
    aluCtr=0;
    #100;
    input1=15;
    input2=10;
    #100 aluCtr=4'b0001;
    #100 aluCtr=4'b0010;
    #100 aluCtr=4'b0110;
    #100;
    input1=10;
    input2=15;
    #100;
    input1=15;
    input2=10;
    aluCtr=4'b0111;
    #100;
    input1=10;
    input2=15;
    #100;
    aluCtr=4'b1100;
    input1=1;
    input2=1;
    #100 input1=16;

    end
```

由此激励文件得到的波形图如下：



其中`xor`操作的二进制显示为：



综上，我们得到了与参考书一致且符合激励代码逻辑的波形图。故，该模块成功实现。

5 总结反思

- 本次实验中，指导书提供的指导相对比较简略，因此需要我们对前两次的实验代码较为熟悉，而且需要我们掌握一定的verilog基本语法。只有这样才能较顺利的完成实验任务。
- 本次实验中，我主要遇到的困难出现在ALU-Ctr的实现过程中。这是因为我对 `casex` 的使用比较陌生，我不理解无关项`x`如何影响case的判断，这也导致我的波形结果总是出现问题。后来我查阅了相关资料，发现要把明确指明数字的case放置在无关项的case前，这样才能保证：明确指明数字的case不会提前被无关项的case所包含，而因此执行错误的代码。经过这样的调整，我的ALUCtr模块最终才得以成功地实现。
- 通过本次实验，我对于主控制单元等部件的解码过程有了更充分、更深入的理解。此前老师曾在课上多次讲解针对不同类型指令、不同类型操作的不同输出信号，在亲自动手实现之后，我能够更清晰的认识到硬件设计方面的逻辑以及MIPS指令集的底层实现方法。