上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

计算机系统结构实验报告

实验名称：简单的类 MIPS 单周期处理器部件实现-控制器，ALU

姓　　名：洪瑄锐

学　　号：517030910227

班　　级：F1703302

手　　机：15248246044

邮　　箱：1204378645@qq.com

目录

1. 实验目的

1) 理解 CPU 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. 实验原理与代码实现

2.1 主控制器 Ctr 的实现

2.1.1 主要思想

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

各控制信号所代表的含义：

1) RegDst：

R 型指令和 I 型指令目标寄存器在指令编码中的位置不同，前者为 Inst[15-11]，后者为 Inst[20-16]，RegDst 信号用于根据指令类型选择目标寄存器。

2) Branch：

用于判断是否为跳转指令，在 beq 指令中如果两个寄存器中的值相等则跳转到与当前指令的下一条指令距离为指定立即数的指令，Branch 信号与 zero 信号相与，如果结果为 1 则进行 pc=pc+4+(immediate<<2)运算，否则 pc=pc+4。

3) MemRead：

判断当前指令是否需要读内存。

4) MemtoReg：

判断当前指令是否需要将内存中的数据读入寄存器，一般用于 lw 指令。

5) ALUOp：

根据指令类型判断算术逻辑运算单元 ALU 需要执行什么运算，如 add、sub、or 等等。

6) Memwrite：

判断当前指令是否需要写内存，一般用于 sw 指令。

7) ALUSrc：

算术逻辑运算单元 ALU 的操作数可能来自于寄存器，也可能是立即数，根据指令类型确定操作数来源。

8) RegWrite：

判断当前指令是否需要写寄存器。

因此，根据实验指导书所给真值表编写代码即可。

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

主控制模块真值表

注：Jump 指令编码是 000010，Jump 信号输出 1，其余输出 0

2.1.2 Ctr 模块代码

```
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
        if(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
```

```
    else if(opCode==6'b000100)//beq
    begin
       RegDst = 1'bx;
       ALUSrc = 0;
       MemToReg = 1'bx;
       RegWrite = 0;
       MemRead = 0;
       MemWrite = 0;
       Branch = 1;
       ALUOp = 2'b01;
       Jump = 0;
    end
```

```
else if(opCode==6'b100011)//lw
begin
   RegDst = 0;
   ALUSrc = 1;
   MemToReg = 1;
   RegWrite = 1;
   MemRead = 1;
   MemWrite = 0;
   Branch = 0;
   ALUOp = 2'b00;
   Jump = 0;
end
```

```
else if(opCode==6'b101011)//sw
begin
   RegDst = 1'bx;
   ALUSrc = 1;
   MemToReg = 1'bx;
   RegWrite = 0;
   MemRead = 0;
   MemWrite = 1;
   Branch = 0;
   ALUOp = 2'b00;
   Jump = 0;
end
```

```
else if(opCode==6'b000010)//Jump
begin
   RegDst = 0;
   ALUSrc = 0;
   MemToReg = 0;
   RegWrite = 0;
   MemRead = 0;
   MemWrite = 0;
   Branch = 0;
   ALUOp = 2'b00;
   Jump = 1;
end
```

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

采用 if else 语句实现真值表。

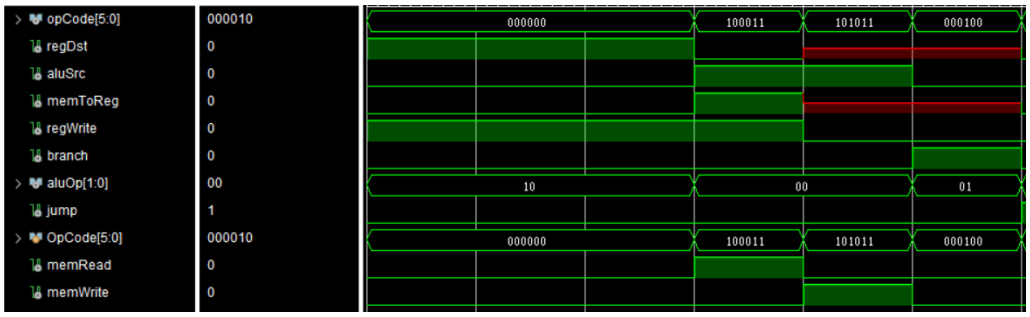
2.1.3 仿真测试

通过给 opcode 赋值观察各信号是否为对应的值来判断试验是否成功。

部分仿真代码如下所示：

```
initial begin
    //Initialize Inputs
    OpCode = 0;

    //Wait 100ns for global reset to finish
    #100;

    #100
        OpCode = 6'b000000;//R-type
    #100
        OpCode = 6'b100011;//lw
    #100
        OpCode = 6'b101011;//sw
    #100
        OpCode = 6'b000100;//beq
    #100
        OpCode = 6'b000010;//Jump
end
```

仿真结果：



真值表：

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

主控制模块真值表

注：Jump 指令编码是 000010，Jump 信号输出 1，其余输出 0

经过与真值表对比，仿真结果正确。

2.2 运算单元控制器 ALUCtr 的实现

2.2.1 主要思想

ALU 的控制单元模块（ALUCtr）是根据主控制器的 ALUOp 来判断指令类型。根据指令的后 6 位即功能码区分 R 型指令，如 add、sub、or 等，综合操作码 opcode 和功能码 funct 控制 ALU 做正确的操作。因此，要想实现 ALUCtr 只要综合 aluop 和 funct 输出 aluCtrOut 即可。

实验指导书所示真值表：

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
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

2.2.2 ALUCtr 模块代码

根据实验指导书中真值表得：

```
module ALUCtr(
    input [1:0] aluOp,
    input [5:0] funct,
    output [3:0] aluCtrOut
    );

    reg [3:0] ALUCtrOut;
   assign aluCtrOut = ALUCtrOut;

    always @ (aluOp or funct)
    begin
        casex ({aluOp,funct})
            8'b00xxxxxx:
                ALUCtrOut = 4'b0010;

            8'bx1xxxxxx:
                ALUCtrOut = 4'b0110;

            8'b10xx0000:
                ALUCtrOut = 4'b0010;

            8'b10xx0010:
                ALUCtrOut = 4'b0110;

            8'b10xx0100:
                ALUCtrOut = 4'b0000;


            8'b10xx0101:
                ALUCtrOut = 4'b0001;

            8'b10xx1010:
                ALUCtrOut = 4'b0111;
            endcase
        end

endmodule
```

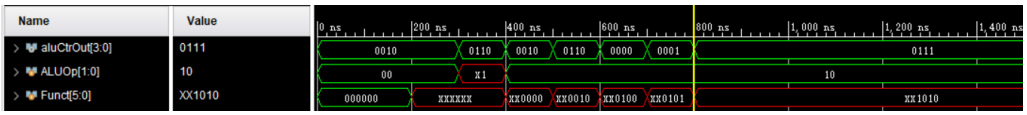
2.2.3 仿真测试

通过给 ALUOp 和 Funct 赋值检查 aluCtrOut 的值是否与真值表相符即可。

部分代码：

```
initial begin
    {ALUOp, Funct} = 8'b00000000;

    #100;

    #100
        {ALUOp, Funct} = 8'b00xxxxxx;
    #100
        {ALUOp, Funct} = 8'bx1xxxxxx;
    #100
        {ALUOp, Funct} = 8'b10xx0000;
    #100
        {ALUOp, Funct} = 8'b10xx0010;
    #100
        {ALUOp, Funct} = 8'b10xx0100;
    #100
        {ALUOp, Funct} = 8'b10xx0101;
    #100
        {ALUOp, Funct} = 8'b10xx1010;
end
```

仿真结果：



与真值表对应后可知仿真结果正确。

2.3 ALU 的实现

2.3.1 主要思想

根据 ALUCtr，ALU 对两个输入执行对应的操作，ALURes 为输出结果。若减法操作 ALURes 的结果为 0 时，则 Zero 的输出置为 1。编写模块代码时，输入为 ALUCtr，两个操作数，输出为 ALURes，Zero，根据 ALUCtr 判断对两个操作数执行何种运算。

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

输入输出真值表

2.3.2 ALU 模块代码

```
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'b0010://add
        begin
            aluRes = input1 + input2;
            if(aluRes == 0)
                zero = 1;
            else
                zero = 0;
         end
```

```
        4'b0110://sub
        begin
            aluRes = input1 - input2;
            if(aluRes == 0)
                zero = 1;
            else
                zero = 0;
         end

         4'b0000://and
         begin
            aluRes = input1 & input2;
            if(aluRes == 0)
                zero = 1;
            else
                zero = 0;
          end

         4'b0001://or
         begin
            aluRes = input1|input2;
            if(aluRes == 0)
                zero = 1;
             else
                zero = 0;
          end
```

```
            4'b0111://stl
            begin
              if(input1 < input2)
                 begin
                     aluRes = 1;
                     zero = 0;
                  end
              else
                 begin
                     aluRes = 0;
                     zero = 1;
                  end
             end

             4'b1100://nor
             begin
                 aluRes = ~(input1|input2);
                 if(aluRes == 0)
                     zero = 1;
                 else
                     zero = 0;
             end
          endcase
       end
    endmodule
```

2.3.3 仿真测试

通过对操作数 input1、input2 和 aluCtr 赋值得到正确的运算结果 aluRes 和 zero

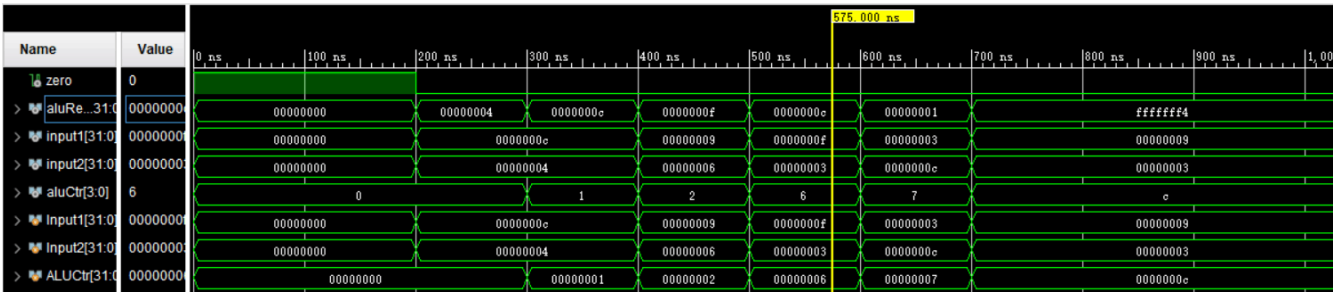
部分代码：

```
initial begin
    Input1 = 0;
    Input2 = 0;
    ALUCtr = 4'b0000;

    #100;

    #100
        Input1 = 4'b1100;
        Input2 = 4'b0100;
        ALUCtr = 4'b0000;//and
    #100
        Input1 = 4'b1100;
        Input2 = 4'b0100;
        ALUCtr = 4'b0001;//or
     #100
        Input1 = 4'b1001;
        Input2 = 4'b0110;
        ALUCtr = 4'b0010;//add
     #100
        Input1 = 4'b1111;
        Input2 = 4'b0011;
        ALUCtr = 4'b0110;//sub
```

```
#100
  Input1 = 4'b0011;
  Input2 = 4'b1100;
  ALUCtr = 4'b0111;//slt;
#100
  Input1 = 4'b1001;
  Input2 = 4'b0011;
  ALUCtr = 4'b1100;//nor

end
```

仿真结果：

1100b&0100b=0100b=**4h**；1100b | 0100b=1100b=**ch**；1001b+0110b=1111b=**fh**；
1111b-0011b=1100b=**ch**；slt 0011b 0110b=**1h**；nor 00000009b 00000003b=0101b=**ffffffff4h**



分析可知仿真结果正确。

3. 感想

由于 lab1 和 lab2 实验指导书给出了完整的代码，所以 lab3 才是真正意义上的第一次实验。通过这次实验我有几点收获：

1) 我学习了 verilog 语言的基本语法，如 module、always、ifelse、case 等，还明白了模块与激励之间的关系，懂得了如何实例化，wire 与 reg 的使用等。

2) 加深了对简单 cpu 中 Ctr 模块，ALUCtr 模块，ALU 模块的理解，了解了其内部实现。

感谢老师和同学们在本次实验中对我的教导和帮助。

4. 参考文献

《cs145 实验指导书 lab03》