计算机系统结构试验

Lab03: 简单的类 MIPS 单周期处理器功能部件的设计与实现（一）

姓名：N/A

摘要

在 Lab03 中，我进行了 MIPS 处理器中 Control、ALU Control 和 ALU 的设计与仿真，包括如何解码指令并生成相应的控制信号、如何根据指令类型生成适当的 ALU 操作码、如何实现 MIPS 处理器中的算术逻辑单元。通过本次实验，我进一步加深了对 Verilog 语言的理解和运用，掌握 MIPS 处理器的关键组成部分的设计和实现方法，并且能够使用仿真工具进行验证和调试，给我带来宝贵的经验和收获。

目录

1. 实验目的

（1）理解主控制部件或单元、ALU 控制器单元、ALU 单元的原理；
（2）熟悉所需的 MIPS 指令集；
（3）使用 Verilog HD 设计与实现主控制器部件(Ctr)；
（4）使用 Verilog 设计与实现 ALU 控制器部件(ALUCtr)；
（5）ALU 功能部件的实现；
（6）使用 Vivado 进行功能模块的行为仿真。

2. 原理分析

2.1 Vivado 工程的基本组成

（1）Crt.v 文件
（2）ALUCtr.v 文件
（3）ALU.v 文件
（4）Ctr_tb.v 激励文件
（5）ALUCtr_tb.v 激励文件
（6）ALU_tb.v 激励文件

2.2 Crt 模块的原理

主控制单元（Ctr）的输入为指令的 opCode 字段，操作码经过 Ctr 模块的译码，输出 ALUOp，RegDst，ALUSrc，MemToReg，RegWrite，MemRead，MemWrite，Branch，Jump 等功能单元输出正确的控制信号。译码逻辑如下：

输入或输出	信号名称	R型	ld	sd	beq
输入	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
输出	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

2.3 ALUCrt 模块的原理

算数逻辑单元 ALU 的控制单元（ALUCtr）根据主控制器的 ALUOp 控制信号来判断指令类型，并依据指令的后 6 位区分 R 型指令。综合这两种输入，以控制 ALU 做正确操作。译码逻辑如下：

ALUOp		funct7字段							funct3字段			操作
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

2.4 ALU 模块的原理

算术逻辑单元 ALU 根据 ALUCtr 的控制信号将两个输入执行与之对应的操作。ALURes 为输出结果。若减法操作 ALURes 的结果为 0 时，则 Zero 输出置为 1。计算逻辑如下：

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

3. 功能实现

3.1 Crt 模块的实现

为实现译码逻辑，使用 always 块来响应 opCode 的变化，根据 opCode 的不同值，使用 case 语句生成相应的控制信号。每个 case 语句对应一个操作码，包括 R-type、lw、sw、beq、jump 和默认操作。根据操作码的不同，分别给输出信号赋予相应的值。代码如下：

```
always @(opCode) begin
  case (opCode)
    6'b000000: begin  // R-type: add, sub, and, or, slt
      regDst = 1'b1;
      aluSrc = 1'b0;
      memToReg = 1'b0;
      regWrite = 1'b1;
      memRead = 1'b0;
      memWrite = 1'b0;
      branch = 1'b0;
      aluOp = 2'b10;
      jump = 1'b0;
    end
    6'b100011: begin  // lw
      regDst = 1'b0;
      aluSrc = 1'b1;
      memToReg = 1'b1;
      regWrite = 1'b1;
      memRead = 1'b1;
      memWrite = 1'b0;
      branch = 1'b0;
      aluOp = 2'b00;
      jump = 1'b0;
    end
    6'b101011: begin  // sw
      regDst = 1'b0;  // don't care
      aluSrc = 1'b1;
      memToReg = 1'b0;  // don't care
      regWrite = 1'b0;
      memRead = 1'b0;
      memWrite = 1'b1;
      branch = 1'b0;
      aluOp = 2'b00;
      jump = 1'b0;
    end
```

```
    6'b000100: begin  // beq
      regDst = 1'b0;  // don't care
      aluSrc = 1'b0;
      memToReg = 1'b0;  // don't care
      regWrite = 1'b0;
      memRead = 1'b0;
      memWrite = 1'b0;
      branch = 1'b1;
      aluOp = 2'b01;
      jump = 1'b0;
    end
    6'b000010: begin  // jump
      regDst = 1'b0;
      aluSrc = 1'b0;
      memToReg = 1'b0;
      regWrite = 1'b0;
      memRead = 1'b0;
      memWrite = 1'b0;
      branch = 1'b0;
      aluOp = 2'b00;
      jump = 1'b1;
    end
    default: begin  // nop
      regDst = 1'b0;
      aluSrc = 1'b0;
      memToReg = 1'b0;
      regWrite = 1'b0;
      memRead = 1'b0;
      memWrite = 1'b0;
      branch = 1'b0;
      aluOp = 2'b00;
      jump = 1'b0;
    end
  endcase
end
```

3.2 ALUCrt 模块的实现

ALUCtr 同样使用 always 块来响应 ALUOp 和 funct 的变化，根据它们的不同值，使用 casex 语句生成相应的 ALU 控制信号。casex 语句根据 ALUOp 和 funct 的组合值进行匹配，每个 casex 语句对应一种组合情况，包括 add、sub、and、or、slt 操作和默认（invalid）情况。根据组合值的不同，分别给 ALUCtrOut 赋予相应的 4 位值。代码如下：

```
always @(ALUOp, funct) begin
  casex ({
    ALUOp, funct
  })
    8'b00_xxxxxx: begin  // add
      ALUCtrOut = 4'b0010;
    end
    8'b01_xxxxxx: begin  // sub
      ALUCtrOut = 4'b0110;
    end
    8'b1x_xx0000: begin  // add
      ALUCtrOut = 4'b0010;
    end
    8'b1x_xx0010: begin  // sub
      ALUCtrOut = 4'b0110;
    end
    8'b1x_xx0100: begin  // and
      ALUCtrOut = 4'b0000;
    end
    8'b1x_xx0101: begin  // or
      ALUCtrOut = 4'b0001;
    end
    8'b1x_xx1010: begin  // slt
      ALUCtrOut = 4'b0111;
    end
    default: begin  // invalid
      ALUCtrOut = 4'bxxxx;
    end
  endcase
end
```

3.3 ALU 模块的实现

算数逻辑单元 ALU 使用 always 块来响应 input1、input2 和 ALUCtr 的变化，根据 ALUCtr 的不同值，使用 case 语句执行相应的算术逻辑运算并给出结果。case 语句根据 ALUCtr 的值进行匹配，每个 case 语句对应一种 ALU 操作，包括 and、or、add、sub、slt、nor 和默认为 0 情况。根据 ALUCtr 的不同，分别执行相应的算术逻辑运算，并将结果赋给 result。在 case 语句之后，根据计算得到的结果判断是否为零，并将结果赋给 zero。如果 result 等于 32 位的 0，则 zero 为 1；否则，zero 为 0。代码如下：

```
always @(input1, input2, ALUCtr) begin
  case (ALUCtr)
    4'b0000: result = input1 & input2;  // and
    4'b0001: result = input1 | input2;  // or
    4'b0010: result = input1 + input2;  // add
    4'b0110: result = input1 - input2;  // sub
    4'b0111: result = (input1 < input2) ? 32'b1 : 32'b0;  // slt
    4'b1100: result = ~(input1 | input2);  // nor
    default: result = 32'b0;  // Default to 0
  endcase
  zero = (result == 32'b0) ? 1'b1 : 1'b0;
end
```

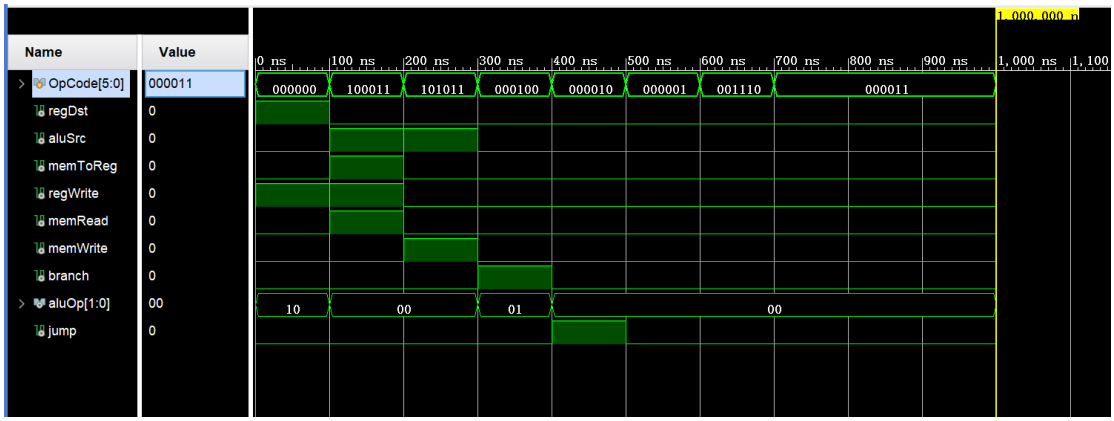
4. 结果验证

4.1 Crt 模块的测试

编写激励文件设置各输入初值，代码如下：

```
initial begin
  // legal inputs
  OpCode = 6'b000000;  // R-type: add, sub, and, or, slt
  #100;
  OpCode = 6'b100011;  // lw
  #100;
  OpCode = 6'b101011;  // sw
  #100;
  OpCode = 6'b000100;  // beq
  #100;
  OpCode = 6'b000010;  // jump
  #100;
  // illegal inputs
  OpCode = 6'b000001;  // illegal
  #100;
  OpCode = 6'b001110;  // illegal
  #100;
  OpCode = 6'b000011;  // illegal
  #100;
end
```

测试结果如图所示：



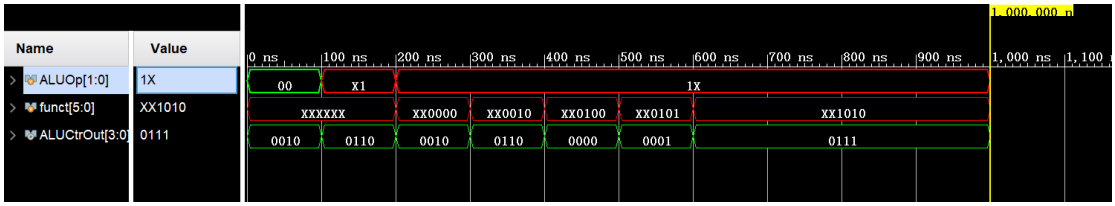
经比较，Crt 模块正确地执行了译码功能，在非法输入时输出默认值全 0。

4.2 ALUCrt 模块的测试

编写激励文件设置各输入初值，代码如下：

```
initial begin
  ALUOp = 2'b00;
  funct = 6'bxxxxxx;
  #100;  // add
  ALUOp = 2'bx1;
  funct = 6'bxxxxxx;
  #100;  // sub
  ALUOp = 2'b1x;
  funct = 6'bxx0000;
  #100;  // add
  ALUOp = 2'b1x;
  funct = 6'bxx0010;
  #100;  // sub
  ALUOp = 2'b1x;
  funct = 6'bxx0100;
  #100;  // and
  ALUOp = 2'b1x;
  funct = 6'bxx0101;
  #100;  // or
  ALUOp = 2'b1x;
  funct = 6'bxx1010;
  #100;  // slt
end
```

测试结果如图所示：



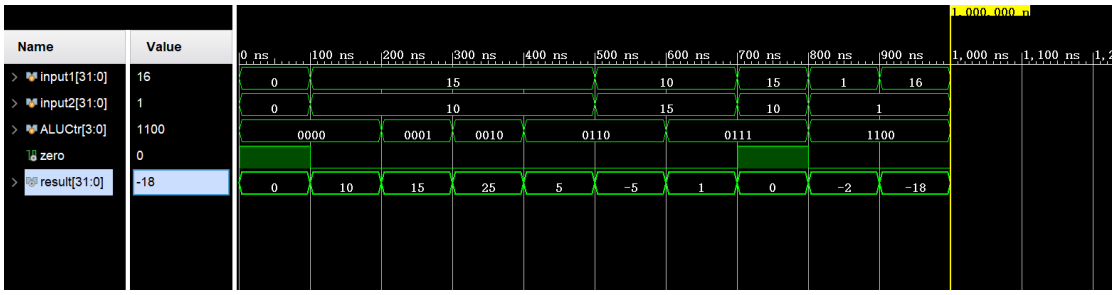
可见输出符合预期。

4.3 ALU 模块的测试

编写激励文件设置各输入初值，代码如下：

```
initial begin
    input1 = 32'd0;
    input2 = 32'd0;
    ALUCtr = 4'b0000;
    #100;
    input1 = 32'd15;
    input2 = 32'd10;
    ALUCtr = 4'b0000;
    #100;
    ALUCtr = 4'b0001;
    #100;
    ALUCtr = 4'b0010;
    #100;
    ALUCtr = 4'b0110;
    #100;
    input1 = 32'd10;
    input2 = 32'd15;
    ALUCtr = 4'b0110;
    #100;
    ALUCtr = 4'b0111;
    #100;
    input1 = 32'd15;
    input2 = 32'd10;
    ALUCtr = 4'b0111;
    #100;
    input1 = 32'd1;
    input2 = 32'd1;
    ALUCtr = 4'b1100;
    #100;
    input1 = 32'd16;
    ALUCtr = 4'b1100;
    #100;
end
```

测试结果如图所示：



输出与正确计算结果一致。

5. 总结与反思

在 Lab04 中，我通过使用 Vivado 开发环境，进一步熟悉了 Verilog HDL 的基本语法和编程技巧。通过这次实验，我掌握了使用 case 块模块化编写分支逻辑的方法，并学会了使用 initial begin-end 块编写时序激励文件。

我要感谢课程组为我们提供的详细指导书，它为我提供了清晰的实验步骤和示例代码，使我能够更好地理解和实践所学的知识。通过这次实验，我不仅巩固了 Verilog 的基础知识，还为的学习和设计打下了坚实的基础。