计算机系统结构实验报告
实验 3

2021 年 6 月 23 日

摘要

本实验实现了简单的类 MIPS 处理器的几个重要部件：主控制器 (Ctr)、运算单元控制器 (ALUCtr) 以及算术逻辑运算单元 (ALU)，其作用分别是产生处理器所需要的各种控制信号、产生算术逻辑运算单元 (ALU) 所需要的控制信号以及根据运算单元控制器 (ALUCtr) 发出的信号执行相应的算术逻辑运算输出结果。本实验通过软件仿真的形式进行实验结果的验证。

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. 功能仿真

2 原理分析

2.1 主控制器模块

主控制器 (Ctr) 的输入为指令的操作码 (opCode) 字段，主控制器模块对操作码进行译码，向 ALU 控制器、寄存器、数据选择器等部件输出正确的控制信号。

本实验中，主控制器模块可以识别 R 型指令、lw、sw、beq、jump 指令并输出对应的控制信号。控制器产生的控制信号如表1所示。

信号	内部寄存器	具体说明
regDst	RegDst	目标寄存器的选择信号；低电平：rt 寄存器；高电平：rd 寄存器
aluSrc	ALUSrc	ALU 第二个操作数来源选择信号；低电平：rt 寄存器值，高电平：立即数拓展结果
memToReg	MemToReg	写寄存器的数据来源选择信号；低电平：ALU 运算结果，高电平：内存读取结果
regWrite	RegWrite	寄存器写使能信号，高电平说明当前指令需要进行寄存器写入
memRead	MemRead	内存读使能信号，高电平有效
memWrite	MemWrite	内存写使能信号，高电平有效
aluOp	ALUOp	2 位信号，发送给运算单元控制器 (ALUCtr) 用来进一步解析运算类型的控制信号
branch	Branch	条件跳转信号，高电平说明当前指令是条件跳转指令
jump	Jump	无条件跳转信号，高电平说明当前指令是无条件跳转指令

表 1: 主控制器产生的控制信号

其中 aluOp 信号代表的含义如表2所示。

aluOp 的信号内容	指令	具体说明
00	lw, sw	ALU 执行加法运算
01	beq	ALU 执行减法运算
1x	R 型指令	ALUCtr 结合指令 Funct 段决定最终操作

表 2: aluOp 信号的具体含义以及解析方式

主控制器 (Ctr) 产生的各种控制信号与指令 OpCode 段的对应方式如表 3 所示。

2.2 算术逻辑单元控制器模块

ALU 控制器模块接受指令中 Funct 段以及来自 Ctr 模块的 aluOp 信号，输出 aluCtr 信号，aluCtr 信号直接决定 ALU 模块进行的计算操作。

OpCode 指令	000000 R 型指令	000010 j	000100 beq	100011 lw	101011 sw
aluSrc	0	0	0	1	1
aluOp	10	00	01	00	00
branch	0	0	1	0	0
jump	0	1	0	0	0
memRead	0	0	0	1	0
memToReg	0	0	0	1	0
memWrite	0	0	0	0	1
regDst	1	0	0	0	0
regWrite	1	0	0	1	0

表 3: 各指令对应的主控制器 (Ctr) 控制信号

ALUCtr 信号输出与 ALUOp 及 Funct 的对应关系如表4所示。

指令	ALUOp	Funct	ALUCtr	说明
lw	00	xxxxxx	0010	ALU 执行加法运算
sw	00	xxxxxx	0010	ALU 执行加法运算
beq	01	xxxxxx	0110	ALU 执行减法运算
add	10	100000	0010	ALU 执行加法运算
sub	10	100010	0110	ALU 执行减法运算
and	10	100100	0000	ALU 执行逻辑与运算
or	10	100101	0001	ALU 执行逻辑或运算
slt	10	101010	0111	ALU 执行 slt 操作

表 4: 运算单元控制器 (ALUCtr) 的输入与输出关系

2.3 ALU 模块

ALU 模块接受 aluCtr 信号，并根据此信号选择执行对应的 ALU 计算功能。ALU 功能与 ALUCtr 信号的对应关系如表5所示。

ALUCtr	ALU 功能
0000	AND
0001	OR
0010	add
0110	sub
0111	set on less than (slt)
1100	nor

表 5: ALU 执行功能与 ALUCtr 信号的对应方式

ALU 模块产生的输出包括 32 位的运算结果以及 1 位 zero 信号；当运算结果为 0 时，zero 处于高

电平，其余时候 zero 处于低电平。

3 功能实现

3.1 主控制器模块的实现

本实验中，主控制器模块 (Ctr) 的输出结果由指令中 opCode 段决定。通过 case 语句，我们可以将指令与 opCode 对应，并根据每个指令的操作需求输出控制信号。

主控制器模块的完整实现见 Ctr.v, 部分核心代码如下：

```
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
    //后续代码类似，此处省略
    endcase
end
```

受限于篇幅，此处只展示 R 型指令与 lw 指令对应的实现。

代码中 RegDst、ALUSrc、MemToReg 等为控制信号寄存器，与对应输出信号线相连。

3.2 算术逻辑单元控制器模块的实现

算术逻辑单元控制器模块 (ALUCtr) 的输出由 aluOp 与 funct 共同决定，其行为与主控制器模块 (Ctr) 相似。不同的是，aluOp 与 funct 有部分位在部分指令中属于无关的位，因此我们使用 casex 替代 case。casex 中，可以用 x 表示我们不关心的位。

算术逻辑单元控制器模块的完整实现见 ALUCtr.v, 核心代码如下：

```
always @ (aluOp or funct)
begin
    casex({aluOp,funct})
        8'b00xxxxxx:    //add
            ALUCtrOut = 4'b0010;
        8'b01xxxxxx:    //sub
            ALUCtrOut = 4'b0110;
        8'b10xx0000:    //add
            ALUCtrOut = 4'b0010;
        8'b1xxx0010:    //sub
            ALUCtrOut = 4'b0110;
        8'b1xxx0100:    //and
            ALUCtrOut = 4'b0000;
        8'b1xxx0101:    //or
            ALUCtrOut = 4'b0001;
        8'b1xxx1010:    //set on less than
            ALUCtrOut = 4'b0111;
    endcase
end
```

3.3 ALU 模块的实现

ALU 模块根据 aluCtr 信号完成指定的功能，可以使用 case 语句选择操作，利用 verilog 自带的运算符完成运算。

ALU 模块的完整实现见 ALU.v, 核心代码如下：

```
always @ (input1 or input2 or aluCtr)
begin
    case(aluCtr)
    4'b0000:    //AND
        ALURes = input1 & input2;
    4'b0001:    //OR
        ALURes = input1 | input2;
```

```
    4'b0010:    //ADD
        ALURes = input1 + input2;
    4'b0110:    //SUB
        ALURes = input1 - input2;
    4'b0111:    //SLT
        ALURes = ($signed(input1) < $signed(input2));
    4'b1100:    //nor
        ALURes = ~(input1 | input2);
    endcase
    if(ALURes==0)
        Zero = 1;
    else
        Zero = 0;
end
```

SLT 运算功能的实现中，由于 Verilog 会默认以无符号数解释 wire 类型，需要通过 $signed 关键词将输入解释为带符号数后再进行比较。

在 ALU 模块的结尾，我们判断运算结果是否为 0，并以此设置 Zero 寄存器，最终作为 zero 信号输出。

4 结果验证

4.1 主控制器模块的仿真

先后向 Ctr 模块输入 R 型、lw、sw、beq、jump 和未定义指令的 OpCode，具体测试内容如表6所示。

OpCode	指令类型
000000	R 型
100011	lw
101011	sw
000100	beq
000010	jump
010101	(无定义)

表 6: 仿真测试使用的 OpCode 及解释

仿真结果如图1所示，我们实现的主控制器模块运行符合预期设计。

4.2 算术逻辑单元控制器模块的仿真

按照表7对算术逻辑单元控制器模块进行测试，仿真结果如图2所示。可见 ALU 控制器模块的输出结果符合预期。

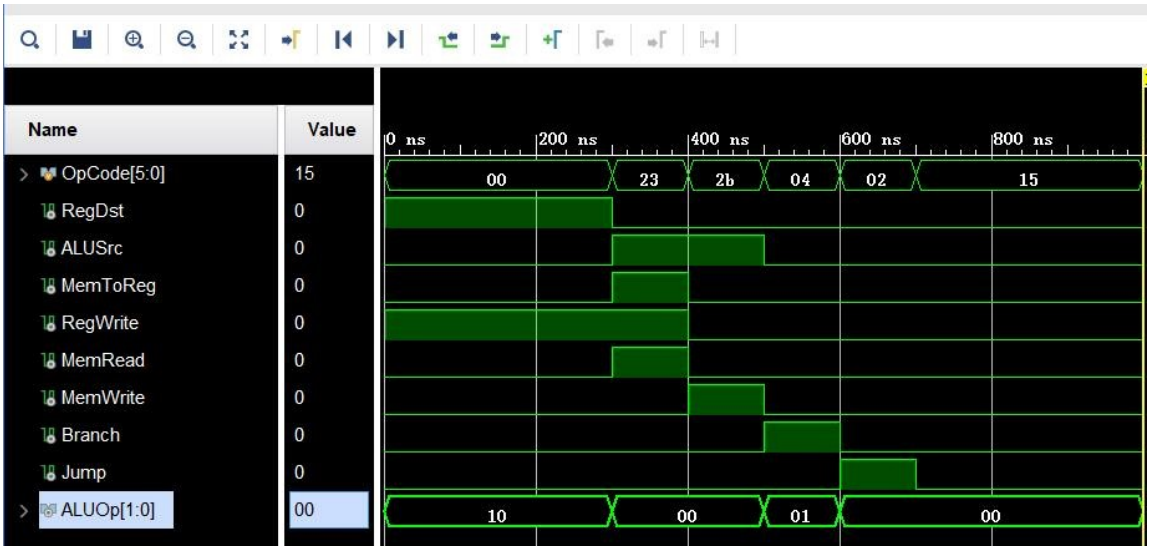

图 1: 主控制器模块的仿真结果

ALUOp	Funct	运算类型
00	xxxxxx	add
01	xxxxxx	sub
10	100000	add
10	100010	sub
10	100100	and
10	100101	or
10	101010	SLT

表 7: 仿真测试使用的 ALUOp、Funct 及解释

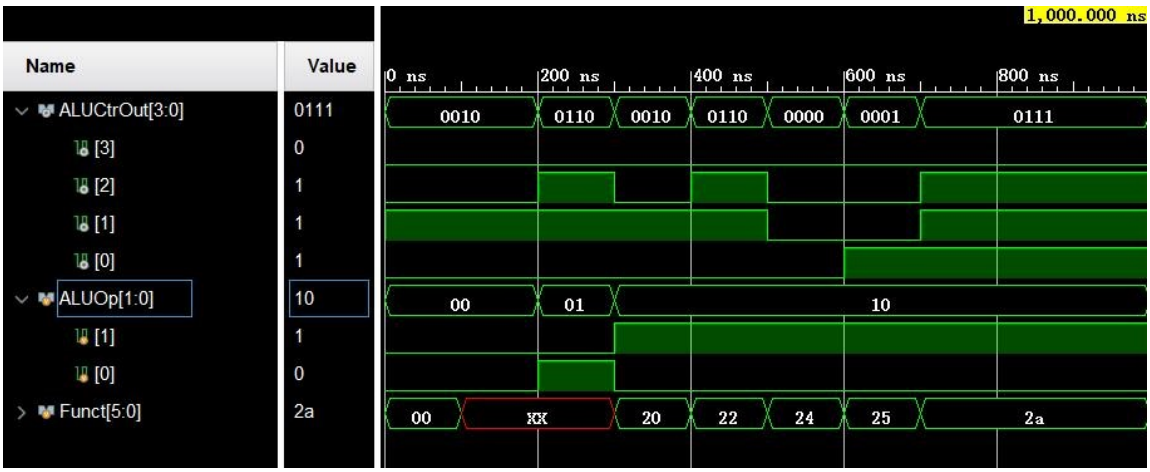

图 2: 算术逻辑单元控制器模块的仿真结果

4.3 ALU 模块的仿真

按照表8向 ALU 模块输入 ALUCtr 以及两个操作数，仿真结果如图3。ALU 模块正确地完成了所有运算测试。

ALUCtr	操作数 1	操作数 2	解释
0000	15	10	15 AND 10
0001	15	10	15 OR 10
0010	15	10	15 + 10
0110	15	10	15 - 10
0110	10	15	10 - 15
0111	15	10	15 < 10
0111	10	15	10 < 15
1100	1	1	1 NOR 1
1100	16	1	16 NOR 1

表 8: ALU 仿真使用的 ALUCtr 以及操作数

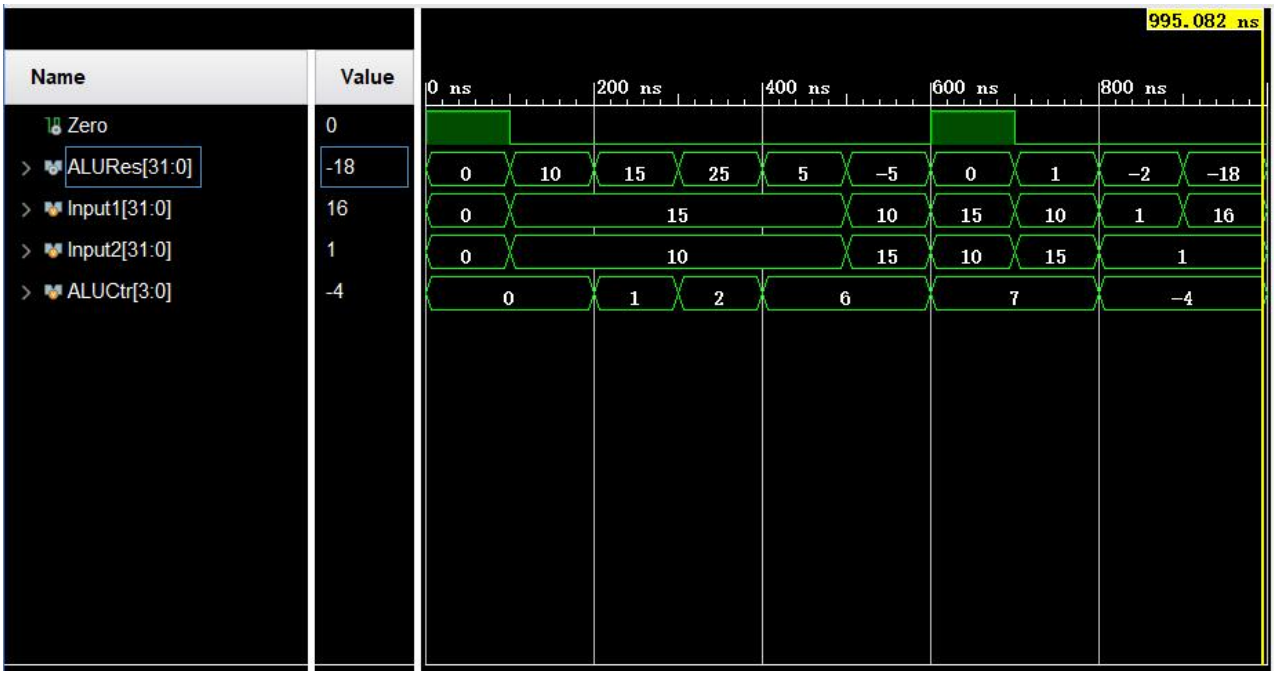

图 3: ALU 模块的仿真结果

5 总结与反思

在本实验中，我实现了主控制器模块、ALU 控制器模块、ALU 模块。在开始实验之前，我本以为需要通过复杂的逻辑电路完成指令到控制信号的转换；开始试验后，我才知道可以通过 Verilog 的 case 与 casex 语句快速完成控制信号输出。实验难度比我的预期简单，这增加了我的信心。

在实验过程中，我曾经混淆了 case 与 casex 语句，在同学的帮助下，我顺利发现了错误并完成了实验。通过本实验，我对 Verilog 的分支语句、逻辑操作符有了初步了解。

本实验中实现的模块功能较为简单，主控制器模块只能识别少量指令，ALU 控制器模块、ALU 模块只能完成基础的加减法与逻辑运算，无法满足后续实验的需求；但是本实验完成了这些模块框架的搭建，后续实验可以便捷地根据需求拓展功能。作为处理器实现系列实验的开始，本实验难度适中，是一个非常合适的引入。