计算机系统结构实验报告　实验 3

简单的类 MIPS 单周期处理器部件实现：控制器与 ALU

方泓杰 518030910150

2020 年 6 月 5 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

本次实验有如下五个实验目的：

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. 算术逻辑运算单元 (ALU) 的实现；
5. 使用功能仿真验证实验的正确性。

2 原理分析

2.1 主控制器 (Ctr) 原理分析

主控制器需要对指令的最高 6 位的 OpCode 域进行解析，初步判断指令的类型并产生对应的处理器控制信号。我们在主控制器中将指令的类型做如下区分：R 型指令（具体的指令在这里不做区分，留给运算单元控制器 (ALUCtr) 区分）；I 型指令中的 load 指令 (lw)，store 指令 (sw) 与 branch 指令 (beq)；J 型指令中的 jump 指令 (j)。本次实验中用到的控制信号如表 1 所示。

信号	具体说明
ALUSrc	算术逻辑运算单元 (ALU) 的第二个操作数的来源 (0: 使用 rt；1: 使用立即数)
ALUOp (*)	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	条件跳转信号，高电平说明当前指令是条件跳转指令 (branch)
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 (jump)
memRead	内存读使能信号，高电平说明当前指令需要进行内存读取 (load)
memToReg	写寄存器的数据来源 (0: 使用 ALU 运算结果；1: 使用内存读取数据)
memWrite	内存写使能信号，高电平说明当前指令需要进行内存写入 (store)
regDst	目标寄存器的选择信号 (0: 写入 rt 代表的寄存器；1: 写入 rd 代表的寄存器)
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

表 1: 主控制器产生的控制信号

上表中标 (*) 的 ALUOp 信号需要进行特殊说明，其包含两个二进制位，所代表的具体含义以及解析方式如表 2 所示。

ALUOp 的信号内容	指令	具体说明
00	lw, sw, j	ALU 执行加法运算
01	beq	ALU 执行减法运算
1x	R 型指令	ALU 具体执行内容需要根据指令最后 6 位的 Funct 域决定

表 2: ALUOp 信号的具体含义以及解析方式

从表 2 中我们可以发现，ALUOp 实际上相当于在主控制器 (Ctr) 中预先得出的部分非 R 型指令（如 beq 指令、j 指令、lw 指令以及 sw 指令等）的 ALU 控制信号；例如当前指令为 beq 指令，那么

ALU 实际需要执行的运算为减法，于是 ALUOp 信号为 01，表示 ALU 应该执行减法操作。当然该控制信号并不能直接送入 ALU，还需要经过运算单元控制器 (ALUCtr) 进行进一步的处理，得到最终的运算单元控制信号 ALUCtrOut，之后才能送入 ALU 进行对其的控制。

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

OpCode 指令	000000 R 型指令	000010 j	000100 beq	100011 lw	101011 sw
ALUSrc	0	0	0	1	1
ALUOp	1x	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: 主控制器 (Ctr) 产生的各种控制信号与指令的对应方式

当出现其他暂不支持的指令时，我们将所有的控制信号均置为 0，将这条指令看作一条空指令 (nop)，使得该指令对数据没有影响，保证数据的正确性。

2.2 运算单元控制器 (ALUCtr) 原理分析

运算单元控制器 (ALUCtr) 对指令的最后 6 位的 Funct 域进行解析，结合主控制器 (Ctr) 产生的 ALUOp 信号，给出最终的运算单元控制信号 ALUCtrOut。该信号作用于 ALU，实现对于 ALU 不同功能的控制。运算单元控制器 (ALUCtr) 的解析方式如表 4 所示。

指令	ALUOp	Funct	ALUCtrOut	具体说明
add	1x	100000	0010	ALU 执行加法运算
sub	1x	100010	0110	ALU 执行减法运算
and	1x	100100	0000	ALU 执行逻辑与运算
or	1x	100101	0001	ALU 执行逻辑或运算
slt	1x	101010	0111	ALU 执行小于时置位运算
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
j	00	xxxxxx	0010	ALU 执行加法运算

表 4: 运算单元控制器 (ALUCtr) 的解析方式

从表 4 中可以看出，运算单元控制信号 ALUCtrOut 与 ALU 执行的操作类型有一一对应的关系，我们将在第 2.3 节给出具体的对应方式。

2.3 算术逻辑运算单元 (ALU) 原理分析

ALU 主要根据由运算单元控制器 (ALUCtr) 产生的运算单元控制信号 ALUCtrOut，对输入的两个数执行对应的算术逻辑运算，输出运算的结果以及部分控制信号。其输出的一个重要信号为 zero 信号，当运算结果为 0 时该信号为高电平，否则为低电平；该信号用于与 branch 指令结合判断是否满足转移条件。ALU 执行的算术逻辑运算类型与运算单元控制信号 ALUCtrOut 的对应方式如表 5 所示。

ALUCtrOut	ALU 执行算术逻辑运算类型
0000	逻辑与 (and)
0001	逻辑或 (or)
0010	加法 (add)
0110	减法 (sub)
0111	小于时置位 (slt)
1100 (*)	逻辑或非 (nor)

表 5: ALU 执行的算术逻辑运算类型与 ALUCtrOut 的对应方式

其中，用 (*) 标注的逻辑或非并未在第 2.2 节的表 4 中出现，但是由于 ALU 模块部分的实验有该方面的要求，因此在这里我们将其特殊标注。

3 功能实现

3.1 主控制器 (Ctr) 功能实现

在第 2.1 节中我们详细介绍了主控制器 (Ctr) 的设计，我们只需要按照第 2.1 节中的表 3 进行相应信号的实现即可。在这里我们使用了 Verilog 中的 `case` 语句进行实现，下面**节选部分代码进行展示**，完整代码详见附录 A.1。注意，这里的 `default` 选项表示出现不支持指令时，我们将所有信号置零，当作一条空指令 (nop) 进行处理，对数据不产生影响。

```
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
        // here we ignore some codes, you can refer to appendices for details.
        default:      // default
        begin
            RegDst = 0;
            ALUSrc = 0;
            MemToReg = 0;
            RegWrite = 0;
            MemRead = 0;
            MemWrite = 0;
            Branch = 0;
            ALUOp = 2'b00;
            Jump = 0;
        end
end
```

3.2 运算单元控制器 (ALUCtr) 功能实现

在第 2.2 节中我们详细介绍了运算单元控制器 (ALUCtr) 的设计，我们只需要按照第 2.2 节中的表 4 进行相应信号的实现即可。在这里我们使用了 Verilog 中的 `casex` 语句（即带通配符 `x` 的 `case` 语句）进行实现，下面展示代码中的重要部分，完整代码详见附录 A.2。

```
always @ (aluOp or funct)
begin
    casex ({aluOp, funct})
        8'b00xxxxx:
            ALUCtrOut = 4'b0010;
        8'b01xxxxx:
            ALUCtrOut = 4'b0110;
        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;
    endcase
end
```

3.3 算术逻辑运算单元 (ALU) 功能实现

在第 2.3 节中我们详细介绍了运算单元控制器 (ALUCtr) 的设计，我们只需要按照第 2.3 节中的表 5 进行相应信号的实现即可。在这里我们使用了 Verilog 中的 `case` 语句进行实现，同时在求出运算结果后，我们使用 `if` 语句进行 `zero` 信号的设置。下面展示代码中的重要部分，完整代码详见附录 A.3。

```
always @ (inputA or inputB or aluCtrOut)
begin
    case (aluCtrOut)
        4'b0000:  // and
            ALURes = inputA & inputB;
        4'b0001:  // or
            ALURes = inputA | inputB;
        4'b0010:  // add
            ALURes = inputA + inputB;
        4'b0110:  // sub
            ALURes = inputA - inputB;
        4'b0111:  // set on less than
            ALURes = ($signed(inputA) < $signed(inputB));
        4'b1100:  // nor
            ALURes = ~(inputA | inputB);
        default:
            ALURes = 0;
    endcase
    if (ALURes == 0)
        Zero = 1;
    else
        Zero = 0;
end
```

4 结果验证

4.1 主控制器 (Ctr) 结果验证

我们使用 Verilog 编写激励文件，采用软件仿真的形式对主控制器 (Ctr) 模块进行测试（代码实现参见附录 B.1）。我们在激励文件中对于 R 型指令、load 指令 (lw)、store 指令 (sw)、branch 指令 (beq)、jump 指令 (j) 以及其他不支持的指令都进行了测试，测试结果如图 1 所示。

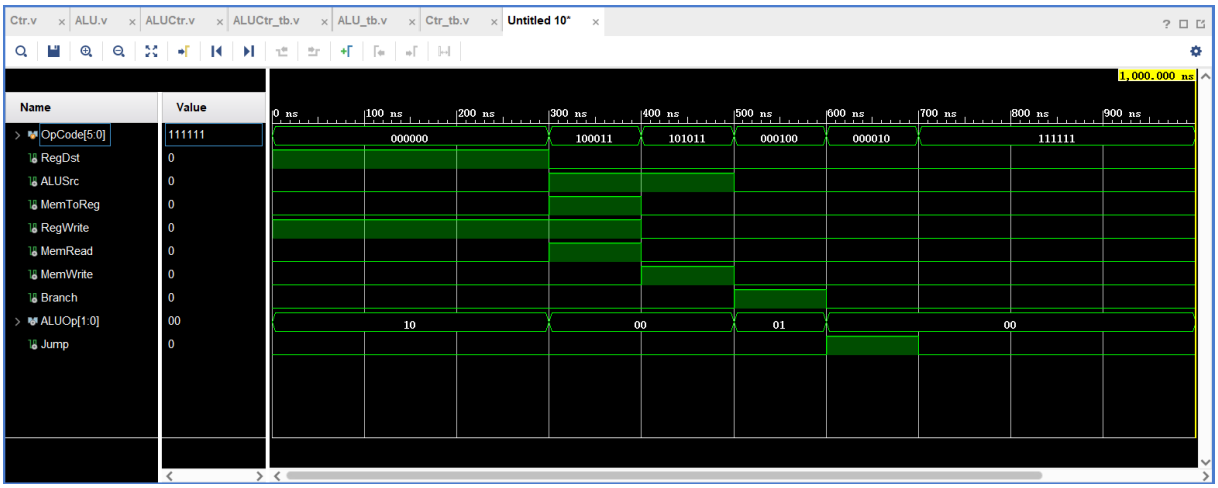

图 1: 对于主控制器 (Ctr) 的测试结果

从图 1 中可以看出，主控制器正确产生了第 2.1 节中表 1 中的所有控制信号；同时对于暂时不支持的其他无效指令，进行了正确的处理（所有控制信号置零，当作空指令 (nop)）。

4.2 运算单元控制器 (ALUCtr) 结果验证

我们使用 Verilog 编写激励文件，采用软件仿真的形式对运算单元控制器 (ALUCtr) 模块进行测试（代码实现参见附录 B.2）。我们在激励文件中对于加法指令 (add)、减法指令 (sub)、逻辑与指令 (and)、逻辑或指令 (or)、小于时置位指令 (slt) 以及 ALUOp 为 00 或 01 的指令（如 load 指令 (lw)、branch 指令 (beq) 等）都进行了测试，测试结果如图 2 所示。

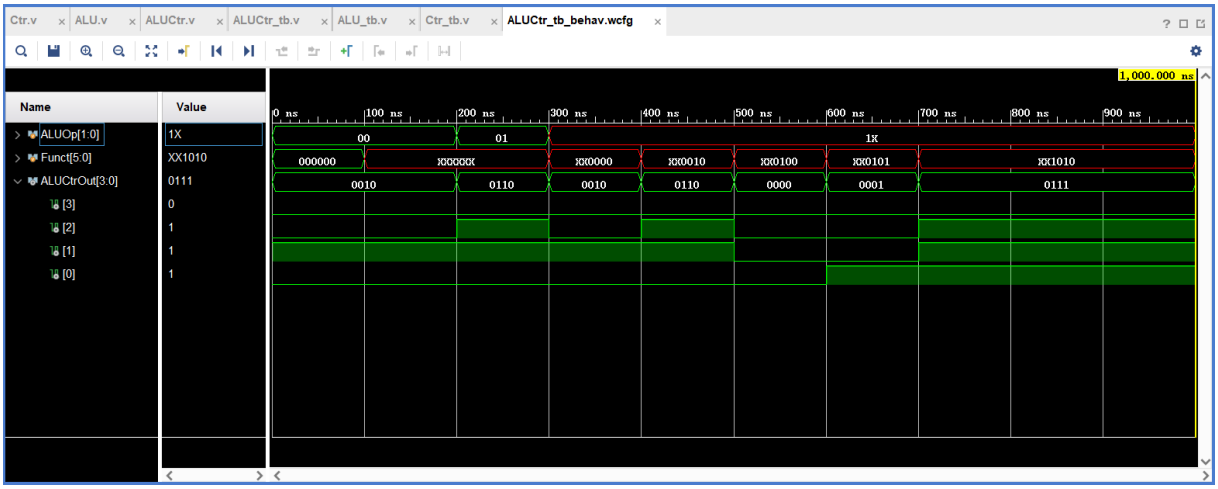

图 2: 对于运算单元控制器 (ALUCtr) 的测试结果

从图 2 中可以看出，运算单元控制器依据第 2.2 节中表 4 中内容正确产生了控制信号 ALUCtrOut。

4.3 算术逻辑运算单元 (ALU) 结果验证

我们使用 Verilog 编写激励文件，采用软件仿真的形式对算术逻辑运算单元 (ALU) 模块进行测试 (代码实现参见附录 B.3)。我们在激励文件中对于加法、减法、逻辑与、逻辑或、小于时置位、逻辑或非等算数逻辑运算都进行了测试，测试结果如图 3 所示。

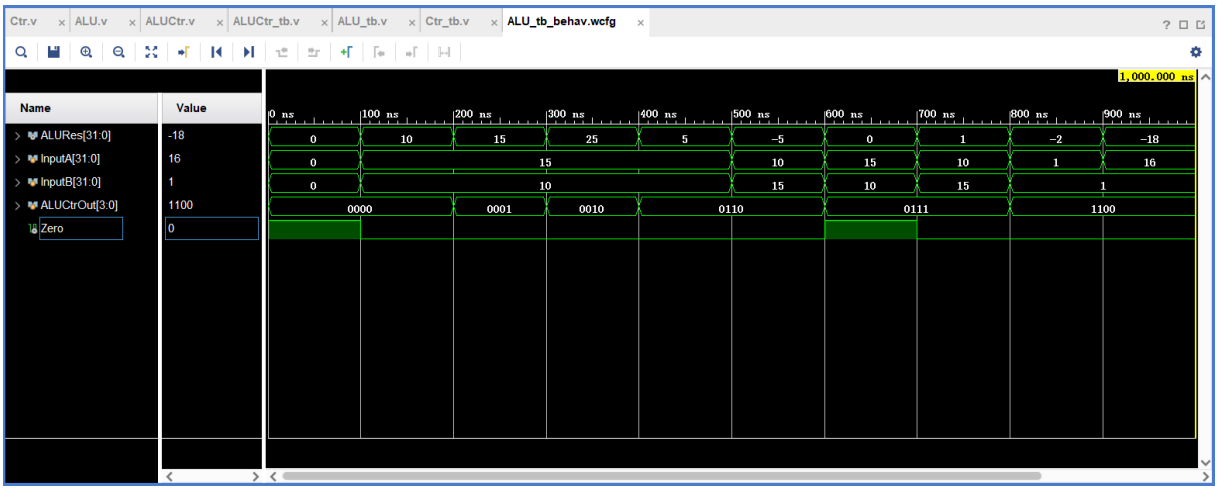

图 3: 对于算术逻辑运算单元 (ALU) 的测试结果

我们在这里特别截取或非运算的测试部分作为例子详细展示，如图 4 所示。

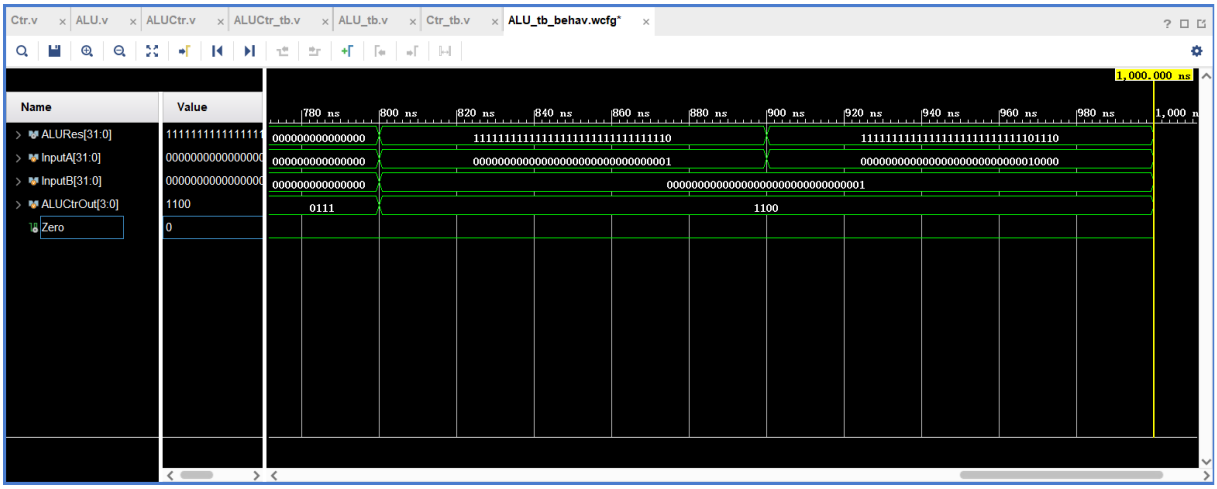

图 4: 对于算术逻辑运算单元 (ALU) 中或非运算 (nor) 的测试结果

从图 3 和图 4 中可以看出，算术逻辑运算单元根据第 2.3 节中表 5 中的内容执行了正确的运算并且得到了正确的结果以及控制信号 zero。

5 总结与反思

本实验设计并实现了类 MIPS 处理器的三个重要组成部件：主控制器 (Ctr)、运算单元控制器 (ALUCtr) 以及算术逻辑运算单元 (ALU)，并且通过软件仿真模拟的方法验证了它们的正确性，为后面的单周期类 MIPS 处理器以及流水线处理器的实现奠定基础。同时，这也运用了我在第二次实验的报告中所提到的“子元件”的思想，这种思想可以将一个较为复杂的电路（如整个处理器）拆分成许多功能简单、易于实现的元件进行实现（如本次实验中实现的三个小元件）。

同时，本次实验实现的内容都较为基础，并没有涉及到更多类型的指令，如果需要支持更多类型的指令可能需要对某些控制信号（如 ALUOp 和 ALUCtrOut）做进一步的设计（这个在接下来的实验中会有所体现）。另外，本次实验中为了方便，我们运用 Verilog 语言的提供的运算符号进行 ALU 的设计，并没有运用其提供的运算单元（如全加器等等）进行实现；事实上，在实际的设计中是使用下一级的“子元件”（如全加器等）进行实现的，这也是一个需要注意的地方。

在本次实验中，我还学习了 Verilog 语言的一些小技巧，如使用 `x` 通配符以及 `casex` 进行带通配符的匹配等等，这些都让我的编程能力得到了提高。

6 致谢

感谢本次实验中指导老师在课程微信群里为同学们答疑解惑；

感谢上海交通大学网络信息中心提供的远程桌面资源；

感谢计算机科学与工程系相关老师对于课程指导书的编写以及对于课程的设计，让我们可以更快更好地学习相关知识，掌握相关技能；

感谢电子信息与电气工程学院提供的优秀的课程资源。

附录 A　设计文件完整代码实现

A.1　主控制器 (Ctr) 的代码实现

```
1  `timescale 1ns / 1ps
2
3  module Ctr(
4      input [5 : 0] opCode,
5      output regDst,
6      output aluSrc,
7      output memToReg,
8      output regWrite,
9      output memRead,
10     output memWrite,
11     output branch,
12     output [1 : 0] aluOp,
13     output jump
14     );
15
16     reg RegDst;
17     reg ALUSrc;
18     reg MemToReg;
19     reg RegWrite;
20     reg MemRead;
21     reg MemWrite;
22     reg Branch;
23     reg [1 : 0] ALUOp;
24     reg Jump;
25
26     always @(opCode)
27     begin
28         case(opCode)
29             6'b000000:    // R Type
30             begin
31                 RegDst = 1;
32                 ALUSrc = 0;
33                 MemToReg = 0;
34                 RegWrite = 1;
35                 MemRead = 0;
36                 MemWrite = 0;
37                 Branch = 0;
38                 ALUOp = 2'b10;
```

```
39              Jump = 0;
40          end
41          6'b100011:    // lw
42          begin
43              RegDst = 0;
44              ALUSrc = 1;
45              MemToReg = 1;
46              RegWrite = 1;
47              MemRead = 1;
48              MemWrite = 0;
49              Branch = 0;
50              ALUOp = 2'b00;
51              Jump = 0;
52          end
53          6'b101011:    // sw
54          begin
55              RegDst = 0;
56              ALUSrc = 1;
57              MemToReg = 0;
58              RegWrite = 0;
59              MemRead = 0;
60              MemWrite = 1;
61              Branch = 0;
62              ALUOp = 2'b00;
63              Jump = 0;
64          end
65          6'b000100:    // beq
66          begin
67              RegDst = 0;
68              ALUSrc = 0;
69              MemToReg = 0;
70              RegWrite = 0;
71              MemRead = 0;
72              MemWrite = 0;
73              Branch = 1;
74              ALUOp = 2'b01;
75              Jump = 0;
76          end
77          6'b000010:    // jump
78          begin
79              RegDst = 0;
```

```
80                ALUSrc = 0;
81                MemToReg = 0;
82                RegWrite = 0;
83                MemRead = 0;
84                MemWrite = 0;
85                Branch = 0;
86                ALUOp = 2'b00;
87                Jump = 1;
88            end
89            default:      // default
90            begin
91                RegDst = 0;
92                ALUSrc = 0;
93                MemToReg = 0;
94                RegWrite = 0;
95                MemRead = 0;
96                MemWrite = 0;
97                Branch = 0;
98                ALUOp = 2'b00;
99                Jump = 0;
100           end
101       endcase
102   end
103
104   assign regDst = RegDst;
105   assign aluSrc = ALUSrc;
106   assign memToReg = MemToReg;
107   assign regWrite = RegWrite;
108   assign memRead = MemRead;
109   assign memWrite = MemWrite;
110   assign branch = Branch;
111   assign aluOp = ALUOp;
112   assign jump = Jump;
113 endmodule
```

A.2 运算单元控制器 (ALUCtr) 的代码实现

```
`timescale 1ns / 1ps

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

    reg [3 : 0] ALUCtrOut;

    always @ (aluOp or funct)
    begin
        casex ({aluOp, funct})
            8'b00xxxxxx:
                ALUCtrOut = 4'b0010;
            8'b01xxxxxx:
                ALUCtrOut = 4'b0110;
            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;
        endcase
    end

    assign aluCtrOut = ALUCtrOut;
endmodule
```

A.3 算术逻辑运算单元 (ALU) 的代码实现

```
1  `timescale 1ns / 1ps
2
3  module ALU(
4      input [31 : 0] inputA,
5      input [31 : 0] inputB,
6      input [3 : 0] aluCtrOut,
7      output zero,
8      output [31 : 0] aluRes
9      );
10
11     reg Zero;
12     reg [31 : 0] ALURes;
13
14     always @ (inputA or inputB or aluCtrOut)
15     begin
16         case (aluCtrOut)
17             4'b0000:  // and
18                 ALURes = inputA & inputB;
19             4'b0001:  // or
20                 ALURes = inputA | inputB;
21             4'b0010:  // add
22                 ALURes = inputA + inputB;
23             4'b0110:  // sub
24                 ALURes = inputA - inputB;
25             4'b0111:  // set on less than
26                 ALURes = ($signed(inputA) < $signed(inputB));
27             4'b1100:  // nor
28                 ALURes = ~(inputA | inputB);
29             default:
30                 ALURes = 0;
31         endcase
32         if (ALURes == 0)
33             Zero = 1;
34         else
35             Zero = 0;
36     end
37
38     assign zero = Zero;
39     assign aluRes = ALURes;
40 endmodule
```

附录 B　激励文件完整代码实现

B.1　主控制器 (Ctr) 激励文件的代码实现

```
`timescale 1ns / 1ps

module Ctr_tb(
    );

    reg [5 : 0] OpCode;
    wire RegDst;
    wire ALUSrc;
    wire MemToReg;
    wire RegWrite;
    wire MemRead;
    wire MemWrite;
    wire Branch;
    wire [1 : 0] ALUOp;
    wire Jump;

    Ctr u0(.opCode(OpCode), .regDst(RegDst),
           .aluSrc(ALUSrc), .memToReg(MemToReg),
           .regWrite(RegWrite), .memRead(MemRead),
           .memWrite(MemWrite), .branch(Branch),
           .aluOp(ALUOp), .jump(Jump));

    initial begin
        // Initialize Inputs
        OpCode = 0;

        // Wait 100 ns 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
        #100 OpCode = 6'b111111; // default
    end
endmodule
```

B.2 运算单元控制器 (ALUCtr) 激励文件的代码实现

```
1  `timescale 1ns / 1ps
2
3  module ALUCtr_tb(
4      );
5
6      reg [1 : 0] ALUOp;
7      reg [5 : 0] Funct;
8      wire [3 : 0] ALUCtrOut;
9      ALUCtr u0(.aluOp(ALUOp), .funct(Funct), .aluCtrOut(ALUCtrOut));
10
11     initial begin
12         // Initialize Inputs
13         ALUOp = 0;
14         Funct = 0;
15         // Wait 100 ns for global reset to finish
16         #100;
17         // testing
18         ALUOp = 2'b00;
19         Funct = 6'bxxxxxx;
20         #100;
21         ALUOp = 2'b01;
22         Funct = 6'bxxxxxx;
23         #100;
24         ALUOp = 2'b1x;
25         Funct = 6'bxx0000;
26         #100;
27         ALUOp = 2'b1x;
28         Funct = 6'bxx0010;
29         #100;
30         ALUOp = 2'b1x;
31         Funct = 6'bxx0100;
32         #100;
33         ALUOp = 2'b1x;
34         Funct = 6'bxx0101;
35         #100;
36         ALUOp = 2'b1x;
37         Funct = 6'bxx1010;
38         #100;
39     end
40 endmodule
```

B.3 算术逻辑运算单元 (ALU) 激励文件的代码实现

```
`timescale 1ns / 1ps

module ALU_tb(
    );

    wire [31 : 0] ALURes;
    reg [31 : 0] InputA;
    reg [31 : 0] InputB;
    reg [3 : 0] ALUCtrOut;
    wire Zero;

    ALU u0(.inputA(InputA), .inputB(InputB),
           .aluCtrOut(ALUCtrOut), .zero(Zero),
           .aluRes(ALURes));

    initial begin
        // Initialize Inputs
        InputA = 0;
        InputB = 0;
        ALUCtrOut = 0;

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

        // testing and
        InputA = 15;
        InputB = 10;
        ALUCtrOut = 4'b0000;
        #100;

        // testing or
        InputA = 15;
        InputB = 10;
        ALUCtrOut = 4'b0001;
        #100;

        // testing add
        InputA = 15;
        InputB = 10;
        ALUCtrOut = 4'b0010;
```

```
41          #100;
42
43          // testing sub 1
44          InputA = 15;
45          InputB = 10;
46          ALUCtrOut = 4'b0110;
47          #100;
48
49          // testing sub 2
50          InputA = 10;
51          InputB = 15;
52          ALUCtrOut = 4'b0110;
53          #100;
54
55          // testing set on less than 1
56          InputA = 15;
57          InputB = 10;
58          ALUCtrOut = 4'b0111;
59          #100;
60
61          // testing set on less than 2
62          InputA = 10;
63          InputB = 15;
64          ALUCtrOut = 4'b0111;
65          #100;
66
67          // testing nor 1
68          InputA = 1;
69          InputB = 1;
70          ALUCtrOut = 4'b1100;
71          #100;
72
73          // testing nor 2
74          InputA = 16;
75          InputB = 1;
76          ALUCtrOut = 4'b1100;
77          #100;
78      end
79  endmodule
```