

SHANGHAI JIAOTONG UNIVERSITY



计算机系统结构实验报告 - Lab3

姓名：刘一凡

学号：518021910609

完成时间：2020/4/18

目录

[1. 概述 3](#_Toc40977518)

[1.1 实验名称 3](#_Toc40977519)

[1.2 实验目的 3](#_Toc40977520)

[2. 实验描述 3](#_Toc40977521)

[2.1 主控制器单元模块 3](#_Toc40977522)

[2.1.1 模块描述 3](#_Toc40977523)

[2.1.2 主控制器模块代码 5](#_Toc40977524)

[2.1.3 主控制器模块仿真测试代码 7](#_Toc40977525)

[2.1.4 仿真波形展示 8](#_Toc40977526)

[2.1.5 实验结论 9](#_Toc40977527)

[2.2 算术逻辑单元（ALU）控制器模块 9](#_Toc40977528)

[2.2.1 模块描述 9](#_Toc40977529)

[2.2.2 算术逻辑单元控制器模块代码 11](#_Toc40977530)

[2.2.3 算术逻辑单元控制器模块仿真测试代码 11](#_Toc40977531)

[2.2.4 仿真波形展示 13](#_Toc40977532)

[2.2.5 实验结论 13](#_Toc40977533)

[2.3 ALU模块 13](#_Toc40977534)

[2.3.1 模块描述 13](#_Toc40977535)

[2.3.2 ALU模块代码 14](#_Toc40977536)

[2.3.3 ALU模块仿真测试代码 15](#_Toc40977537)

[2.3.4 仿真波形展示 17](#_Toc40977538)

[2.3.5 实验结论 17](#_Toc40977539)

[3 实验心得 18](#_Toc40977540)

# 1. 概述

## 1.1 实验名称

简单的类MIPS 单周期处理器部件实现——控制器，ALU

## 1.2 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. 实验描述

## 2.1 主控制器单元模块

### 2.1.1 模块描述

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

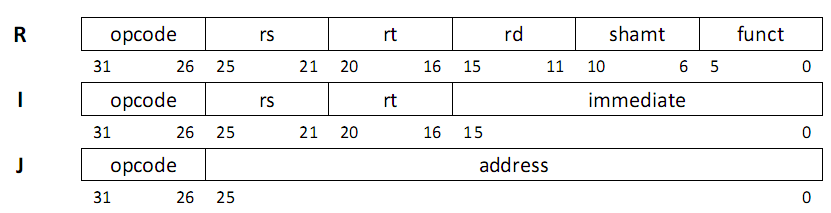


图1：MIPS基本指令格式

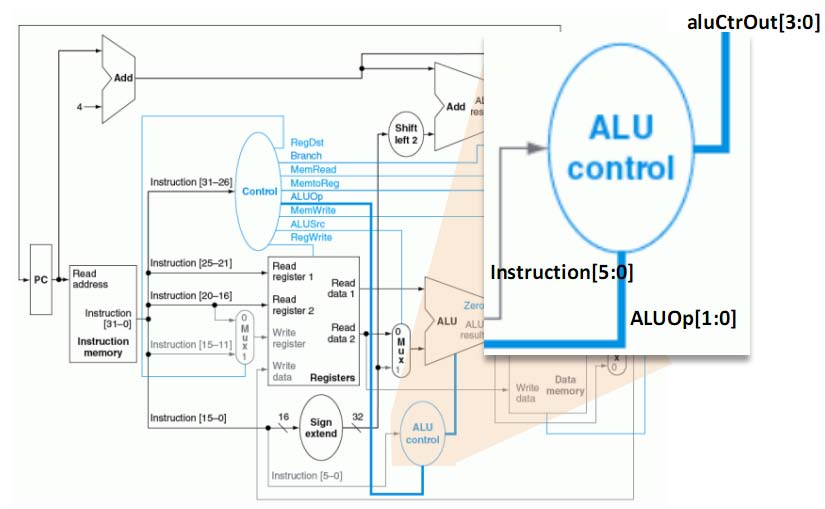


图2：主控制器模块的IO定义

我们可以写出主控制器模块真值表和指令操作码，如下图和表所示。

表1：主控制器模块真值表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input or output** | **Signal name** | **R-format** | **lw** | **sw** | **beq** | **jump** |
| Inputs | Op5 | 0 | 1 | 1 | 0 | 0 |
| Op4 | 0 | 0 | 0 | 0 | 0 |
| Op3 | 0 | 0 | 1 | 0 | 0 |
| Op2 | 0 | 0 | 0 | 1 | 0 |
| Op1 | 0 | 1 | 1 | 0 | 1 |
| Op0 | 0 | 1 | 1 | 0 | 0 |
| Outputs | RegDst | 1 | 0 | X | X | X |
| ALUSrc | 0 | 1 | 1 | 0 | X |
| MemtoReg | 0 | 1 | X | X | X |
| RegWrite | 1 | 1 | 0 | 0 | 0 |
| MemRead | 0 | 1 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 1 | 0 | 0 |
| Branch | 0 | 0 | 0 | 1 | 0 |
| ALUOp | 10 | 00 | 00 | 01 | XX |
| Jump | 0 | 0 | 0 | 0 | 1 |

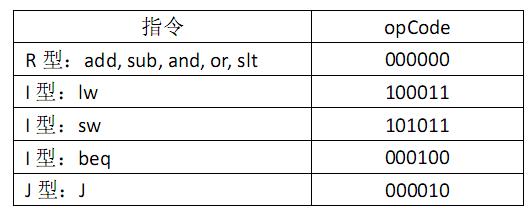


图3：指令操作码

由此，我们可以通过verilog 硬件描述语言写出上述真值表并模拟主控制器模块的功能。

### 2.1.2 主控制器模块代码

1. `timescale 1ns / 1ps

4. module Ctr(
5. input [5:0] opcode,
6. output RegDst,
7. output ALUSrc,
8. output MemToReg,
9. output RegWrite,
10. output MemRead,
11. output MemWrite,
12. output Branch,
13. output [1:0] ALUOp,
14. output Jump
15. );
17. reg RegDst;
18. reg ALUSrc;
19. reg MemToReg;
20. reg RegWrite;
21. reg MemRead;
22. reg MemWrite;
23. reg Branch;
24. reg [1:0] ALUOp;
25. reg Jump;
27. always @ (opcode)
28. begin
29. **case**(opcode)
30. 6'b000000://alu
31. begin
32. RegDst=1;
33. ALUSrc=0;
34. MemToReg=0;
35. RegWrite=1;
36. MemRead=0;
37. MemWrite=0;
38. Branch=0;
39. ALUOp=2'b10;
40. Jump=0;
41. end
43. 6'b000100://beq
44. begin
45. RegDst=0;
46. ALUSrc=0;
47. MemToReg=0;
48. RegWrite=0;
49. MemRead=0;
50. MemWrite=0;
51. Branch=1;
52. ALUOp=2'b01;
53. Jump=0;
54. end
56. 6'b100011://lw
57. begin
58. RegDst=0;
59. ALUSrc=1;
60. MemToReg=1;
61. RegWrite=1;
62. MemRead=1;
63. MemWrite=0;
64. Branch=0;
65. ALUOp=2'b00;
66. Jump=0;
67. end
69. 6'b101011://sw
70. begin
71. RegDst=0;
72. ALUSrc=1;
73. MemToReg=0;
74. RegWrite=0;
75. MemRead=0;
76. MemWrite=1;
77. Branch=0;
78. ALUOp=2'b00;
79. Jump=0;
80. end
82. 6'b000010://jump
83. begin
84. RegDst=0;
85. ALUSrc=0;
86. MemToReg=0;
87. RegWrite=0;
88. MemRead=0;
89. MemWrite=0;
90. Branch=0;
91. ALUOp=2'b00;
92. Jump=1;
93. end
95. **default**:
96. begin
97. RegDst=0;
98. ALUSrc=0;
99. MemToReg=0;
100. RegWrite=0;
101. MemRead=0;
102. MemWrite=0;
103. Branch=0;
104. ALUOp=2'b00;
105. Jump=0;
106. end
107. endcase
108. end
109. endmodule

### 2.1.3 主控制器模块仿真测试代码

1. `timescale 1ns / 1ps
3. module Ctr\_tb();
4. // Inputs
5. reg [5:0] opcode;
7. // Outputs
8. wire RegDst;
9. wire ALUSrc;
10. wire MemToReg;
11. wire RegWrite;
12. wire MemRead;
13. wire MemWrite;
14. wire Branch;
15. wire [1:0] ALUOp;
16. wire Jump;
18. Ctr uut(
19. .opcode(opcode),
20. .RegDst(RegDst),
21. .ALUSrc(ALUSrc),
22. .MemToReg(MemToReg),
23. .RegWrite(RegWrite),
24. .MemRead(MemRead),
25. .MemWrite(MemWrite),
26. .Branch(Branch),
27. .ALUOp(ALUOp),
28. .Jump(Jump)
29. );
30. initial begin
31. // Initialize Inputs
32. opcode = 0;
34. // Wait 100 ns for global reset to finish
35. #100;
37. #100 opcode = 6'b000000;//alu
38. #100 opcode = 6'b100011;//lw
39. #100 opcode = 6'b101011;//sw
40. #100 opcode = 6'b000100;//beq
41. #100 opcode = 6'b000010;//jump
42. #100 opcode=6'b010101;//exception
43. end
45. endmodule

### 2.1.4 仿真波形展示

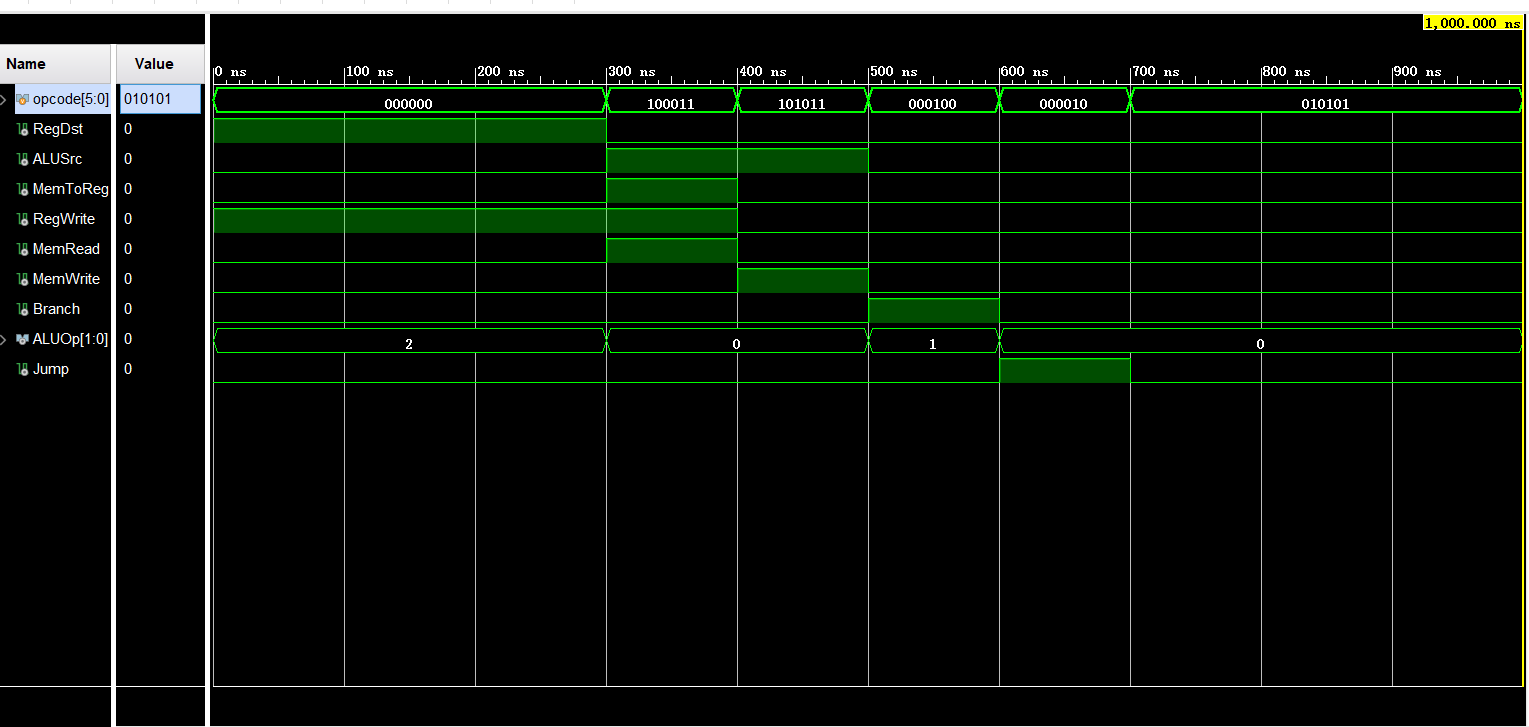


图4：主控制器单元仿真波形

### 2.1.5 实验结论

输出的仿真波形与Ctr模块真值表完全对应，可以根据不同的opcode输出不同的控制信号，以对应不同的输入的指令，Ctr模块实验成功。

## 2.2 算术逻辑单元（ALU）控制器模块

### 2.2.1 模块描述

ALU 的控制器模块（ALUCtr）是根据主控制器的ALUOp 来判断指令类型。根据指令的后6 位区分R 型指令。综合这两种输入，控制ALU 做正确的操作。

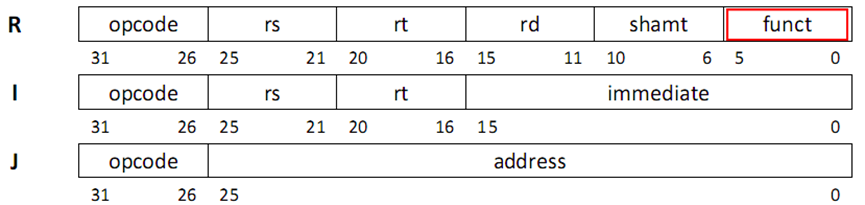


图5：MIPS基本指令格式

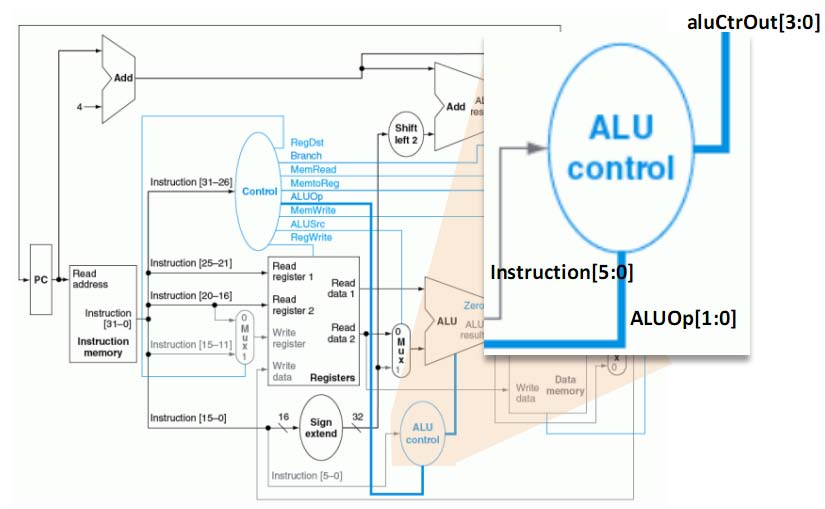


图6：ALU控制器模块

ALU Control （即ALU 的输出aluCtrOut[3:0]）的值与ALU 操作的对应关系如图7，同时我们也可以写出Funct，ALUOp 与ALU Control 编码关系和输入输出真值表，如图8，图9。

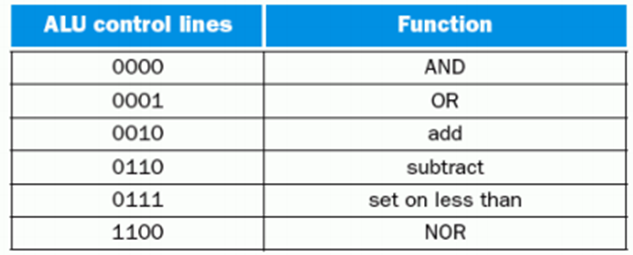


图7：aluCtrOut 和alu 操作的对应关系

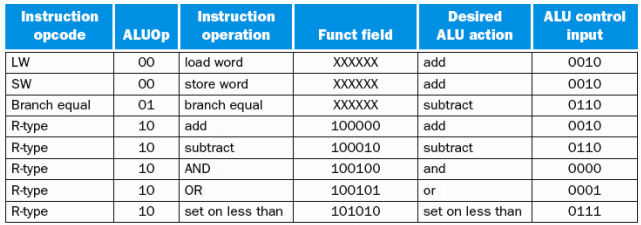


图8：Funct，ALUOp 与ALU Control 编码关系

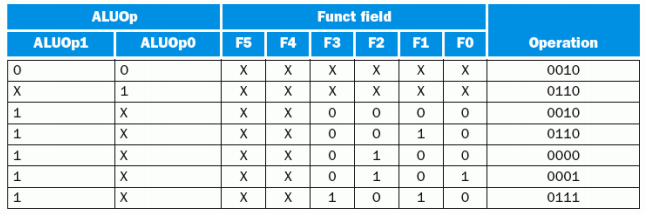


图9：输入输出真值表

由此，我们可以用Verilog语言写出上述真值表并执行相关的操作。

### 2.2.2 算术逻辑单元控制器模块代码

1. `timescale 1ns / 1ps
3. module AluCtr(aluOp, funct, aluCtr);
4. input [1:0] aluOp;
5. input [5:0] funct;
6. output [3:0] aluCtr;
8. reg [3:0] aluCtr;
10. always @ (aluOp or funct)
11. casex ({aluOp, funct})
12. 8'b00xxxxxx: aluCtr = 4'b0010;
13. 8'bx1xxxxxx: aluCtr = 4'b0110;
14. 8'b1xxx0000: aluCtr = 4'b0010;
15. 8'b1xxx0010: aluCtr = 4'b0110;
16. 8'b1xxx0100: aluCtr = 4'b0000;
17. 8'b1xxx0101: aluCtr = 4'b0001;
18. 8'b1xxx1010: aluCtr = 4'b0111;
19. 8'b10100111: aluCtr = 4'b1100;
20. endcase
21. endmodule

### 2.2.3 算术逻辑单元控制器模块仿真测试代码

1. `timescale 1ns / 1ps
3. module AluCtr\_tb(
4. );
5. // Inputs
6. reg [5:0] funct;
7. reg [1:0] aluOp;
9. // Outputs
10. wire [3:0] aluCtr;

13. AluCtr uut (
14. .aluOp(aluOp),
15. .funct(funct),
16. .aluCtr(aluCtr)
17. );
18. //wave A
19. initial begin
20. aluOp = 0;
21. funct = 0;
22. #100;
23. #100 funct = 6'bxxxxxx;
24. #100 aluOp = 2'b01;
25. #100 funct = 6'bxx0000;
26. aluOp = 2'b10;
28. #100 funct = 6'bxx0010;
29. #100 funct = 6'bxx0100;
30. #100 funct = 6'bxx0101;
31. #100 funct = 6'bxx1010;
32. end
33. //Wave B
34. /\*initial begin
35. aluOp = 0;
36. funct = 0;
37. #100;
38. #100 aluOp = 2'b01;
39. #100 aluOp = 2'b10;
40. #100 funct = 6'b000010;
41. #100 funct = 6'b000100;
42. #100 funct = 6'b000101;
43. #100 funct = 6'b001010;
44. end \*/
46. endmodule

### 2.2.4 仿真波形展示

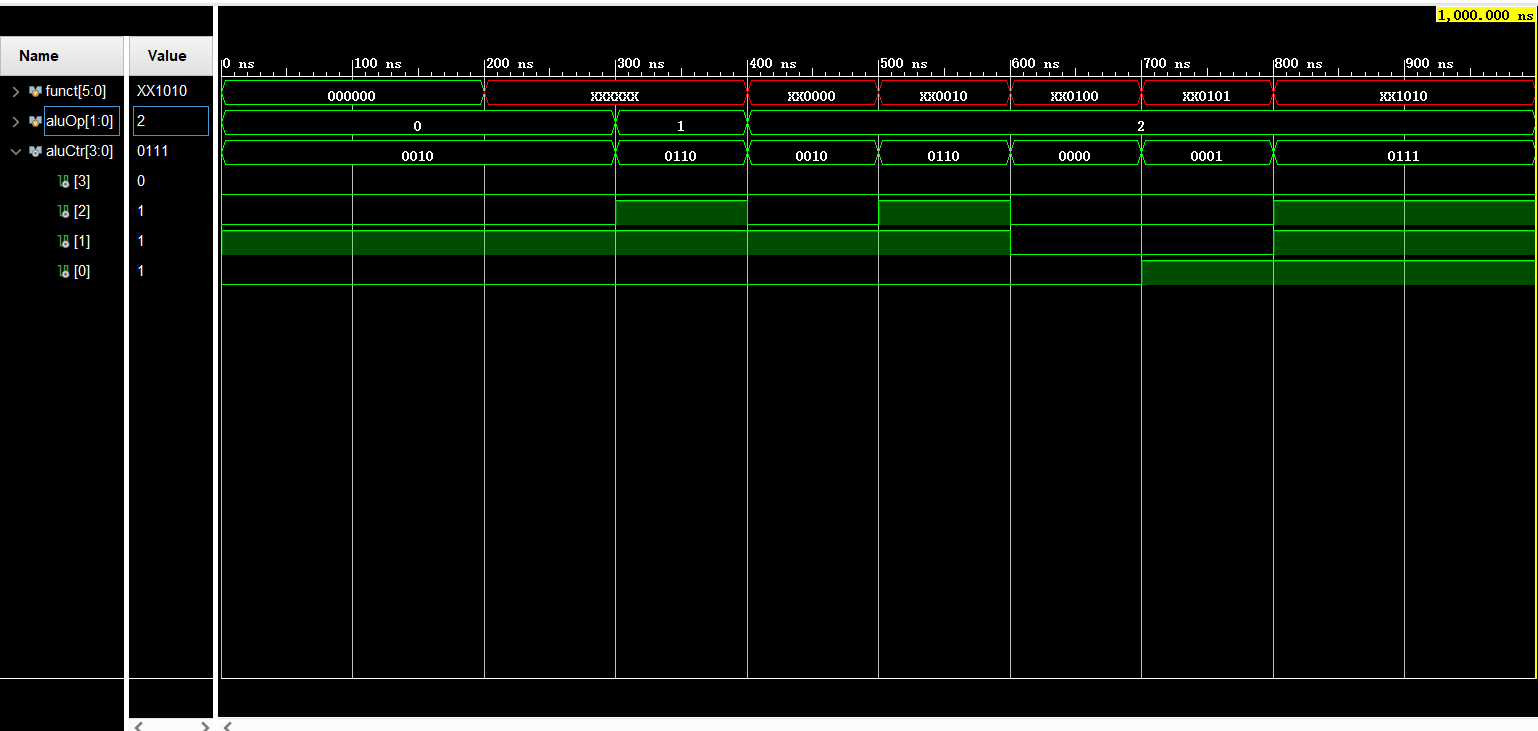
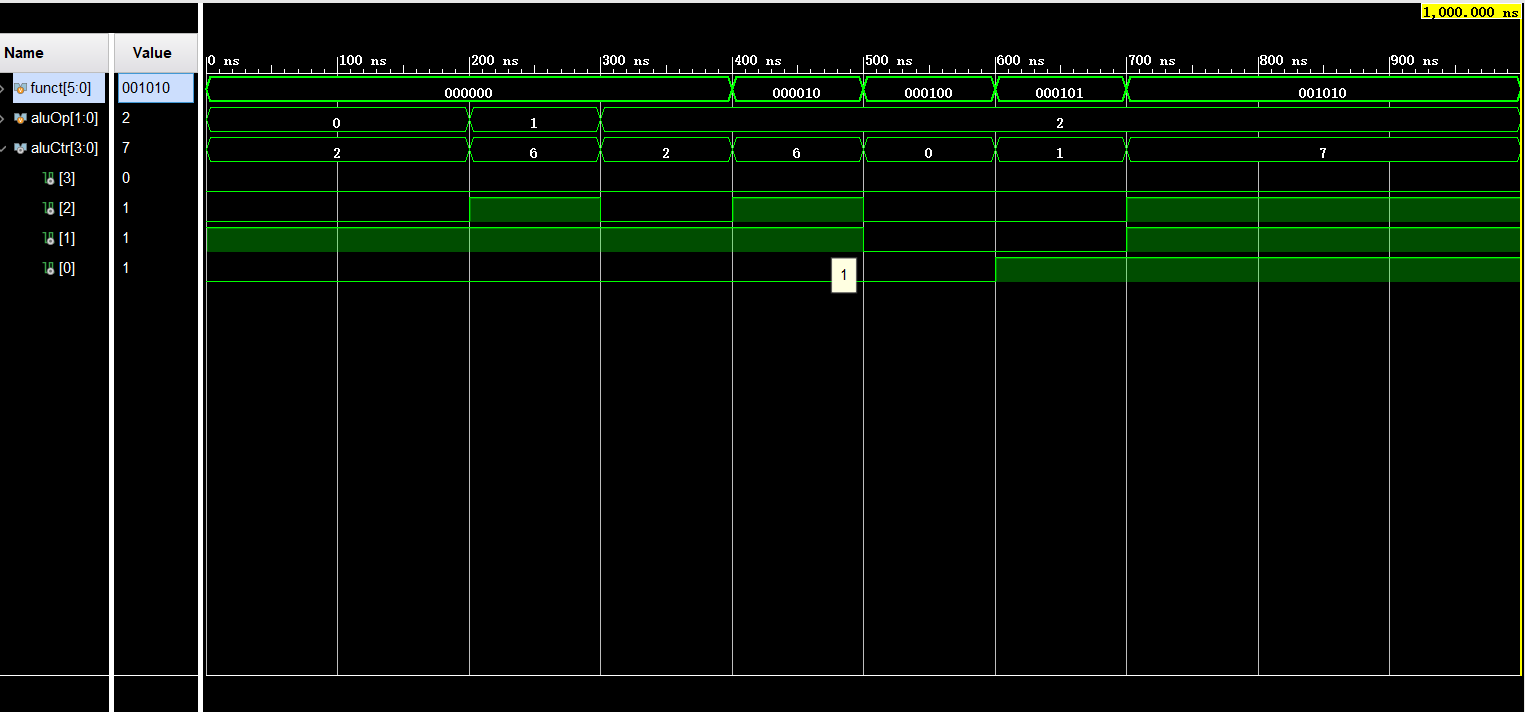


图10：仿真测试波形图（A）(B)

### 2.2.5 实验结论

输出的仿真波形与ALUCTR模块真值表完全对应，能够根据不同的ALUOp及Funct的输入，输出正确的ALU控制信号，ALU控制器模块实验成功。

## 2.3 ALU模块

### 2.3.1 模块描述

算术逻辑单元ALU 根据ALUCtr 信号将两个输入执行对应的操作，ALURes 为输出结果。若做减法操作，当ALURes 结果为0 时，则Zero 输出置为1。

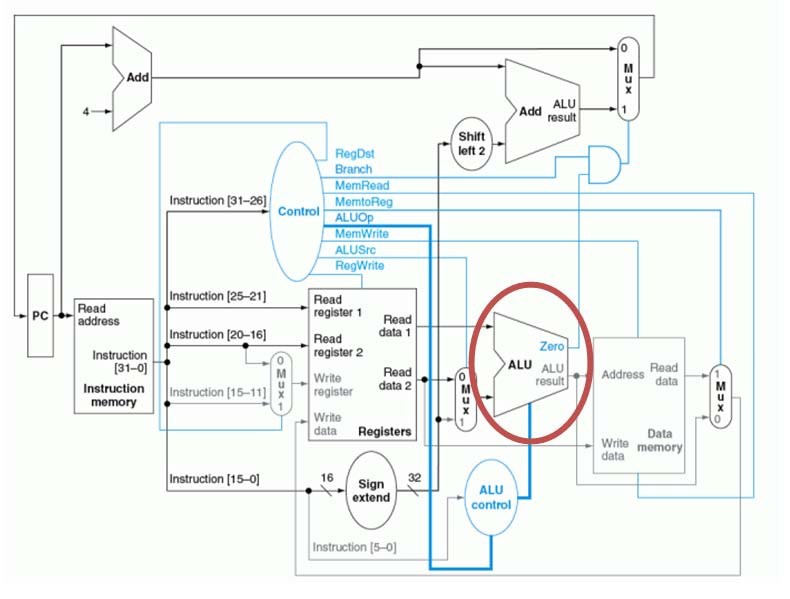


图11：ALU模块的IO定义

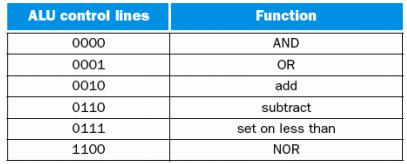


图12：aluCtrOut[3:0]的值与ALU 操作的对应关系

### 2.3.2 ALU模块代码

1. `timescale 1ns / 1ps
3. module Alu(input1, input2, aluCtr, zero, aluRes);
4. input [31:0] input1;
5. input [31:0] input2;
6. input [3:0] aluCtr;
7. output zero;
8. output [31:0] aluRes;
9. reg zero;
10. reg [31:0] aluRes;
12. always @ (input1 or input2 or aluCtr)
13. begin
14. **if** (aluCtr == 4'b0010) //add
15. aluRes = input1 + input2;
16. **else** **if** (aluCtr == 4'b0110) //sub
17. begin
18. aluRes=input1-input2;
19. **if**(aluRes==0)
20. zero=1;
21. **else** zero=0;
22. end
23. **else** **if** (aluCtr == 4'b0000) //and
24. aluRes = input1 & input2;
25. **else** **if** (aluCtr == 4'b0001) //or
26. aluRes = input1 | input2;
27. **else** **if** (aluCtr == 4'b0111) //slt
28. **if** (input1 < input2)
29. aluRes = 1;
30. **else**
31. aluRes = 0;
32. **else** **if** (aluCtr == 4'b1100) //NOR
33. begin
34. aluRes = ~(input1 | input2);
35. **if** (aluRes == 0)
36. zero = 1;
37. **else**
38. zero = 0;
39. end
40. end
41. endmodule

### 2.3.3 ALU模块仿真测试代码

1. `timescale 1ns / 1ps
3. module Alu\_tb(
5. );
6. //input
7. reg [31:0] input1;
8. reg [31:0] input2;
9. reg [3:0] aluCtr;
11. //output
12. wire zero;
13. wire [31:0] aluRes;
15. Alu u(
16. .input1(input1),
17. .input2(input2),
18. .aluCtr(aluCtr),
19. .zero(zero),
20. .aluRes(aluRes)
21. );
23. initial begin
24. // Initialize input,and
25. input1 = 0;
26. input2 = 0;
27. aluCtr = 0;
29. //or
30. #100;
31. input1 = 255;
32. input2 = 100;
33. aluCtr = 4'b0001;
35. //add
36. #100;
37. input1 = 1;
38. input2 = 1;
39. aluCtr = 4'b0010;
41. //sub
42. #100;
43. input1 = 1;
44. input2 = 1;
45. aluCtr = 4'b0110;
46. #100;
47. input1 = 255;
48. input2 = 100;
50. //slt
51. #100;
52. input1 = 100;
53. input2 = 255;
54. aluCtr = 4'b0111;
55. #100;
56. input1 = 255;
57. input2 = 100;
59. //nor
60. #100;
61. input1 = 32'b11111111111111111111111111101110;
62. input2 = 5'b10000;
63. aluCtr = 4'b1100;
64. #100
65. input1=32'b11111111111111111111111111111110;
66. input2=1;
68. end
69. endmodule

### 2.3.4 仿真波形展示

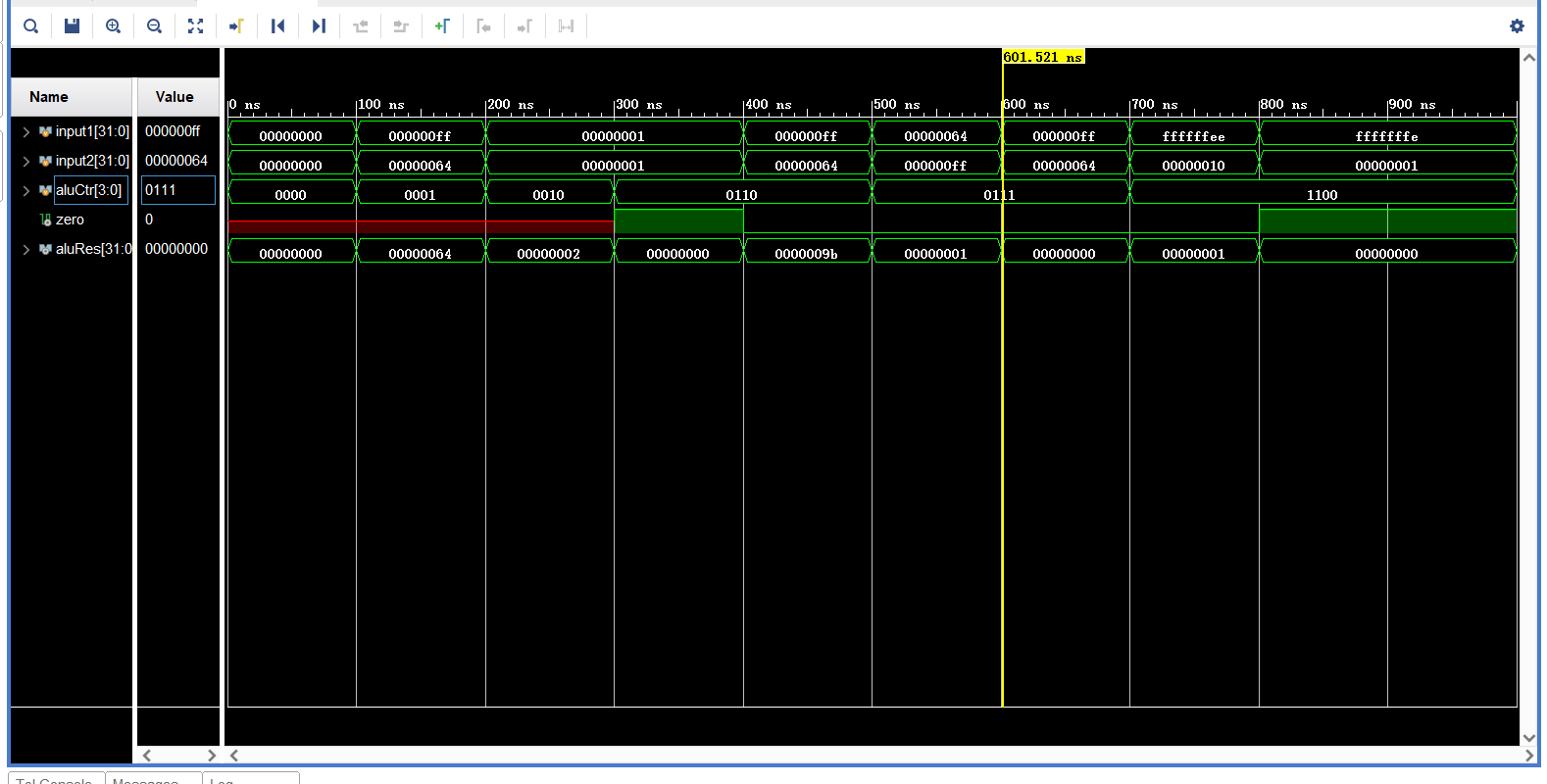


图13：ALU模块仿真波形图

### 2.3.5 实验结论

仿真波形与ALU模块真值表完全对应，能够根据不同的ALUCtr，对input1和input2做出正确的运算，输出到zero和ALURes。说明ALU模块代码书写正确。例如对于fffffffe和00000001进行按位的或非运算，可以正确得到结果00000000；对于两数进行比较运算，可以正确得到大小关系，其它的运算也都可以得到正确的结果。ALU模块实验成功。

# 3 实验心得

本次实验中所提供的代码与前两次实验相比较为简略，所以需要对前两次的实验代码较为熟悉并了解原理才能正确写出这次实验的代码。在这次实验中，我依照给出的代码实例和自己对之前写过的代码的理解，实现了三个模块的功能。在实现功能的过程中，我对于ALU模块，ALU输出的控制信号和控制器模块有了更深的认识，掌握了逻辑运算在计算机中的实际实现过程和处理过程，很好地将上课讲的东西逐步消化。

这次实验中，我也明白了wire语句的作用，掌握了将自己所写的模块实例化的方法，对于编程语言有了进一步的理解。

同时，这次实验我在ALU模块的仿真测试阶段花费的时间比较长，主要原因是无法将信号的持续时间调整到我希望持续的时间，输出和预想的也有差距。经过老师和同学的帮助，我明白了控制输入持续时间的方法，对于casex语句也有了一些理解，最终成功得到了预想的波形。

在这一次自己实现功能的过程中，我也逐步理解了Vivado的功能架构以及各个模块之间的关系，相信这些经验会对我接下来解决更复杂的实验有很大的帮助。