计算机系统结构实验报告 实验 3

姓名：卞思沅

学号：518021910656

日期：2020/05/27

目录：

1. 实验概述

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

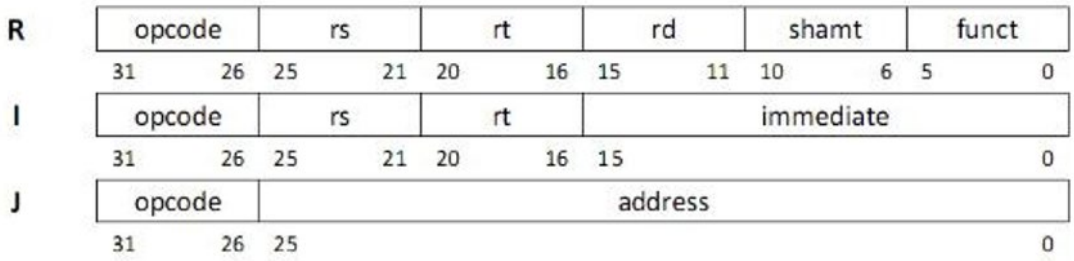
5．使用功能仿真

2. 实验描述

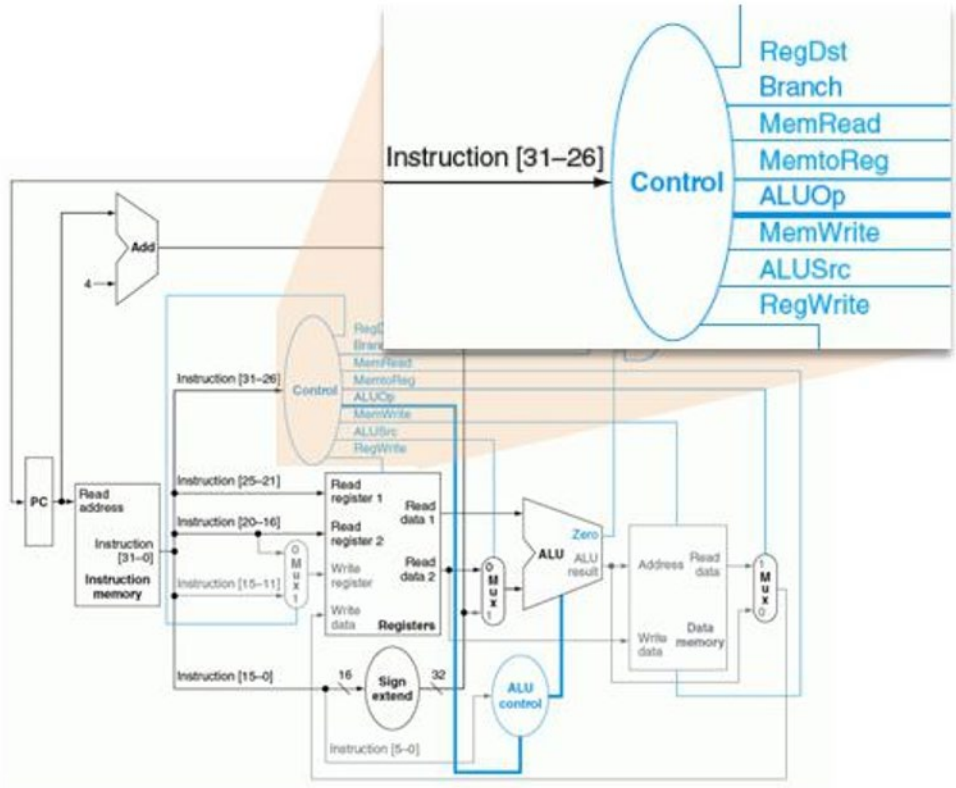
2.1 主控制器模块

2.1.1 主控制器模块简述

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。MIPS 基本指令格式与主控制模块 I/O 定义如下：



Mips 基本指令格式



主控制模块的 IO 定义

实验步骤：

1. 新建文件

2. 编写译码功能。示例代码如下图

```
14      reg RegDst;
15      reg ALUSrc;
16      reg  MemToReg;
17      reg RegWrite;
18      reg MemRead;
19      reg MemWrite;
20      reg Branch;
21      reg [1:0] ALUOp;
22      reg Jump;
23
24      always @(OpCode)
25      begin
26          case(OpCode)
27          6'b000000: //R type
28          begin
29              RegDst = 1;
30              ALUSrc = 0;
31              MemToReg = 0;
32              RegWrite = 1;
33              MemRead = 0;
34              MemWrite = 0;
35              Branch = 0;
36              ALUOp = 2'b10;
37              Jump = 0;
38          end
```

```
91
92          //add beq
93          //6'bxxxxxx;
94            begin
95            ...
96            end
97
98          //add lw
99          //add sw
100         //add Jump
101
102         default:
103         begin
104             RegDst = 0;
105             ALUSrc = 0;
106             MemToReg = 0;
107             RegWrite = 0;
108             MemRead = 0;
109             MemWrite = 0;
110             Branch = 0;
111             ALUOp = 2'b00;
112             Jump = 0;
113         end
114     endcase
```

PS:代码中要把真值表里的所有情况按样例都要覆盖补齐

3.功能仿真，新建文件并添加如下逻辑，并进行仿真：

```
51
52  initial begin
53      // Initialize Inputs
54      OpCode = 0;
55
56      // Wait 100 ns for global reset to finish
57      #100;
58
59      #100 OpCode = 6'b000000;//R-type
60      //Add orther stimuluses here
61
```

2.1.2 主控制器模块实验代码及结果

Ctr.v 代码如下：

```
23  module Ctr(
24      input [5:0] OpCode,
25      output regDst,
26      output aluSrc,
27      output memToReg,
28      output regWrite,
29      output memRead,
30      output memWrite,
31      output branch,
32      output [1:0] aluOp,
33      output jump
34      );
35
36      reg RegDst;
37      reg ALUSrc;
38      reg MemToReg;
39      reg RegWrite;
40      reg MemRead;
41      reg MemWrite;
42      reg Branch;
43      reg [1:0] ALUOp;
44      reg Jump;
45
46      always @ (OpCode)
47      begin
48          case(OpCode)
49          6'b000000:  // R type
50          begin
51              RegDst = 1;
52              ALUSrc = 0;
```

```
46      always @ (OpCode)
47      begin
48          case(OpCode)
49          6'b000000:  // R type
50          begin
51              RegDst = 1;
52              ALUSrc = 0;
53              MemToReg = 0;
54              RegWrite = 1;
55              MemRead = 0;
56              MemWrite = 0;
57              Branch = 0;
58              ALUOp = 2'b10;
59              Jump = 0;
60          end
61
62          6'b100011:  // lw
63          begin
64              RegDst = 0;
65              ALUSrc = 1;
66              MemToReg = 1;
67              RegWrite = 1;
68              MemRead = 1;
69              MemWrite = 0;
70              Branch = 0;
71              ALUOp = 2'b00;
72              Jump = 0;
73          end
74
75          6'b101011:  // sw
76          begin
77              //RegDst = x;
78              ALUSrc = 1;
79              //MemToReg = x;
80              RegWrite = 0;
81              MemRead = 0;
82              MemWrite = 1;
83              Branch = 0;
84              ALUOp = 2'b00;
85              Jump = 0;
86          end
87
88          6'b000100:  // beq
89          begin
90              //RegDst = x;
91              ALUSrc = 0;
92              //MemToReg = x;
93              RegWrite = 0;
94              MemRead = 0;
95              MemWrite = 0;
96              Branch = 1;
97              ALUOp = 2'b01;
98              Jump = 0;
99          end
100
101         6'b000010:  // jump
102         begin
103             RegDst = 0;
```

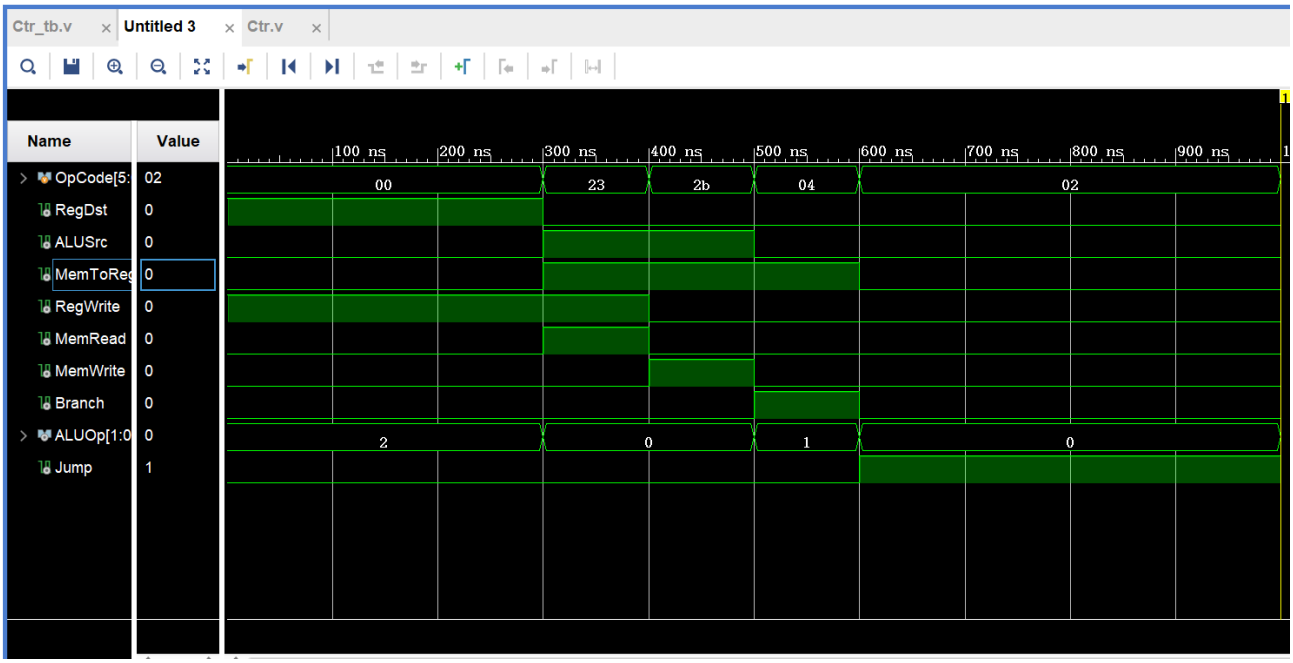
```
100
101         6'b000010:  // jump
102         begin
103             RegDst = 0;
104             ALUSrc = 0;
105             MemToReg = 0;
106             RegWrite = 0;
107             MemRead = 0;
108             MemWrite = 0;
109             Branch = 0;
110             ALUOp = 2'b00;
111             Jump = 1;
112         end
113
114         default:
115         begin
116             RegDst = 0;
117             ALUSrc = 0;
118             MemToReg = 0;
119             RegWrite = 0;
120             MemRead = 0;
121             MemWrite = 0;
122             Branch = 0;
123             ALUOp = 2'b00;
124             Jump = 0;
125         end
126         endcase
127     end
128
129     assign regDst = RegDst;
130     assign aluSrc = ALUSrc;
131     assign memToReg = MemToReg;
132     assign regWrite = RegWrite;
133     assign memRead = MemRead;
134     assign memWrite = MemWrite;
135     assign branch = Branch;
136     assign aluOp = ALUOp;
137     assign jump = Jump;
138 endmodule
139
```

Ctr_tb.v:

```
22  module Ctr_tb( );
23
24      reg [5:0] OpCode;
25      wire RegDst;
26      wire ALUSrc;
27      wire MemToReg;
28      wire RegWrite;
29      wire MemRead;
30      wire MemWrite;
31      wire Branch;
32      wire [1:0] ALUOp;
33      wire Jump;
34
35      Ctr u0(
36          .OpCode(OpCode),
37          .regDst(RegDst),
38          .aluSrc(ALUSrc),
39          .memToReg(MemToReg),
40          .regWrite(RegWrite),
41          .memRead(MemRead),
42          .memWrite(MemWrite),
43          .branch(Branch),
44          .aluOp(ALUOp),
45          .jump(Jump)
46      );
47
48      initial begin
49      // Initialize inputs
50      OpCode = 0;
```

```
48      initial begin
49      // Initialize inputs
50      OpCode = 0;
51
52      //wait 100ns
53      #100;
54
55      #100 OpCode = 6'b000000; // R type
56
57      #100 OpCode = 6'b100011; // lw
58
59      #100 OpCode = 6'b101011; // sw
60
61      #100 OpCode = 6'b000100; // beq
62
63      #100 OpCode = 6'b000010; // jump
64
65      end
66  endmodule
```

仿真结果如下：

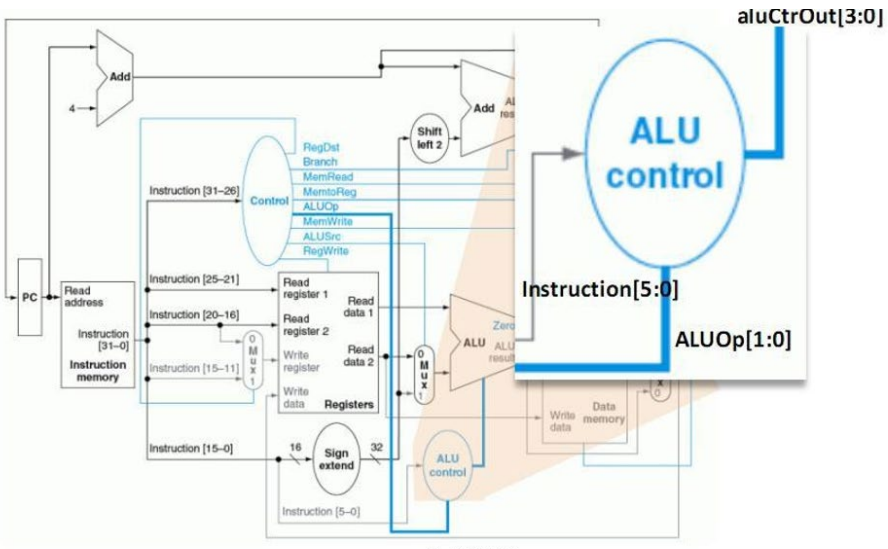


2.2 算术逻辑单元（ALU）控制器

2.2.1 算术逻辑单元（ALU）控制器模块简述

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。



ALU 控制器模块

实验步骤:

1. 新建文件，实现如下真值表

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

aluCtrOut 和 alu 操作的对应关系

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

2. 编写仿真测试

1. 新建 ALUCtr_tb 2. 在测试文件中设定不同的输入，需覆盖全部情形 3. 将 ALUCtr_tb 设为顶层

2.2.2 算术逻辑单元（ALU）控制器实验代码及结果

ALUCtr.v:

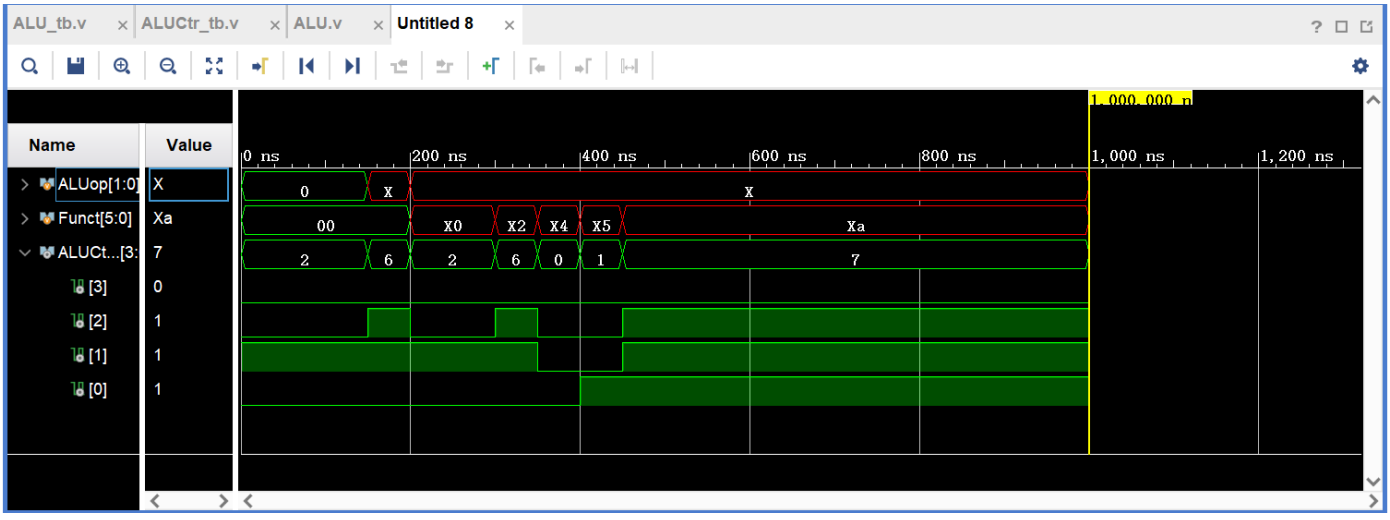
```
22
23  module ALUCtr(
24      input [1:0] ALUop,
25      input [5:0] Funct,
26      output [3:0] ALUCtrOut
27      );
28
29      reg[3:0] aLUCtrOut;
30
31      always @ (ALUop or Funct)
32      begin
33          casex ({ALUop, Funct})
34              8'b00xxxxxx : aLUCtrOut = 4'b0010;
35              8'b01xxxxxx : aLUCtrOut = 4'b0110;
36              8'b1xxx0000 : aLUCtrOut = 4'b0010;
37              8'b1xxx0010 : aLUCtrOut = 4'b0110;
38              8'b1xxx0100 : aLUCtrOut = 4'b0000;
39              8'b1xxx0101 : aLUCtrOut = 4'b0001;
40              default: aLUCtrOut = 4'b0111;
41          endcase
42      end
43      assign ALUCtrOut = aLUCtrOut;
44  endmodule
45
```

ALUCtr_tb.v:

```
23  module ALUCtr_tb( );
24
25      reg[1:0] ALUop;
26      reg[5:0] Funct;
27      wire[3:0] ALUCtrOut;
28
29      ALUCtr u0(
30         .ALUop(ALUop),
31         .Funct(Funct),
32         .ALUCtrOut(ALUCtrOut)
33      );
34
35      initial begin
36      // Initialize inputs
37      ALUop = 0;
38      Funct = 0;
39
40      //wait 100ns
41      #50
42
43      #50
44      ALUop = 2'b00;
45
46      #50
47      ALUop = 2'bx1;
48
```

```
48
49      #50
50      ALUop = 2'b1x;
51      Funct = 6'bxx0000;
52
53      #50
54      ALUop = 2'b1x;
55      Funct = 6'bxx0000;
56
57      #50
58      ALUop = 2'b1x;
59      Funct = 6'bxx0010;
60
61      #50
62      ALUop = 2'b1x;
63      Funct = 6'bxx0100;
64
65      #50
66      ALUop = 2'b1x;
67      Funct = 6'bxx0101;
68
69      #50
70      ALUop = 2'b1x;
71      Funct = 6'bxx1010;
72
73      end
74  endmodule
75
```

仿真结果：



2.3 ALU 模块

2.3.1 ALU 模块简述

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。

实验步骤：

1. 创建文件，完成以下对应：

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

2. 行为仿真，新建测试文件 ALU_tb，并运行仿真

2.3.2 ALU 模块实验代码及结果

ALU.v:

```
23  module ALU(
24      input [31:0] input1,
25      input [31:0] input2,
26      input [3:0] aluCtr,
27      output zero,
28      output [31:0] aluRes
29      );
30
31      reg Zero;
32      reg [31:0] AluRes;
33
34      always @(input1 or input2 or aluCtr)
35      begin
36          Zero = 0;
37          if(aluCtr == 4'b0000) // and
38              AluRes = input1 & input2;
39          else if(aluCtr == 4'b0001) // or
40              AluRes = input1|input2;
41          else if(aluCtr == 4'b0010) //add
42              AluRes = input1 + input2;
43          else if(aluCtr == 4'b0110) // sub
44          begin
45              AluRes = input1 - input2;
46              if(AluRes == 0)
47                  Zero = 1;
48              else
49                  Zero = 0;
50          end
```

```
51          else if(aluCtr == 4'b0111) // set on less than
52          begin
53              AluRes = input1 - input2;
54              if(AluRes == 0)
55                  Zero = 1;
56              else
57                  Zero = 0;
58          end
59          else if(aluCtr == 4'b1100)
60          begin
61              AluRes = ~(input1 | input2);
62              if(AluRes == 0)
63                  Zero = 1;
64              else
65                  Zero = 0;
66          end
67      end
68
69      assign aluRes = AluRes;
70      assign zero = Zero;
71
72  endmodule
73
```

ALU_tb.v:

```
23  module ALU_tb( );
24      reg [31:0] input1;
25      reg [31:0] input2;
26      reg [3:0] aluCtr;
27      wire zero;
28      wire [31:0] aluRes;
29
30      ALU u0(
31          .input1(input1),
32          .input2(input2),
33          .aluCtr(aluCtr),
34          .zero(zero),
35          .aluRes(aluRes)
36      );
37
38      initial begin
39      input1 = 32'hff00;
40      input2 = 32'hf0f0;
41
42      #100
43      aluCtr = 4'b0000;
44
45      #100
46      aluCtr = 4'b0001;
47
48      #100
49      aluCtr = 4'b0010;
```

```
50
51      #100
52      aluCtr = 4'b0110;
53
54      #100
55      aluCtr = 4'b0111;
56
57      #100
58      aluCtr = 4'b1100;
59
60      #100
61      input1 = 32'hffff;
62      input2 = 32'hffff;
63      aluCtr = 4'b0110;
64
65      end
66
67  endmodule
68
```

仿真结果:

3. 实验心得与总结

本次实验需要自己编写的部分多了很多。这对我对代码的理解提出了更高的要求。

编写初期，我对 reg 和 wire 的区别不够了解，在编写时由于声明类型不对，造成了许多编译错误。在上网参考其它资料以及自己的摸索下逐渐解决了问题。然而，我认为我对 reg 和 wire 区别的理解依旧浮于表面，仅仅是能够编译成功并仿真，却不明白两者在硬件层面真正的区别，以后需要继续深入理解。

4、参考资料

2020 计算机系统结构实验指导书-LAB03_M