Lab3 控制器与 ALU

周鼎

April 4, 2016

1 mainCtr

1.1 译码逻辑

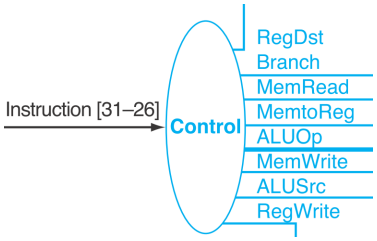

Figure 1: Ctr

指令类型	opCode
R-type	000000
lw	100011
sw	101011
beq	000100
j	000010

Figure 2: 指令操作码

opCode	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead/Write	Branch	ALUOp
000000	1	0	0	1	0 0	0	10
100011	0	1	1	1	1 0	0	00
101011	x	1	x	0	0 1	0	00
000100	x	0	x	0	0 0	1	01

Table 1: 主控模块真值表

注：jump 指令编码 000010，jump 信号输出 1，其余为 0

1.2 程序实现

模块以六位 opCode 为输入，利用 case 语句产生输出，实现主控制功能，

```
1  module Ctr(
2      input [5:0] opCode,
3      output reg regDst,     output reg aluSrc,
4      output reg memToReg,   output reg regWrite,
5      output reg memRead,    output reg memWrite,
6      output reg branch,     output reg jump
7      output reg [1:0] aluOp,
8      );
9       always @(opCode)
10      begin
```

```
11      case (opCode)
12         6'b000010:    begin                                          //jump
13              regDst = 0;       aluSrc = 0;
14               memToReg = 0;    regWrite = 0;
15              memRead = 0;      memWrite = 0;
16              branch =0;        aluOp = 2'b00;   jump = 1;
17         end
18
19         6'b000000:    begin                                          //R-format
20              regDst = 1;       aluSrc = 0;
21              memToReg = 0;     regWrite = 1;
22              memRead = 0;      memWrite = 0;
23              branch = 0;       aluOp = 2'b10;   jump = 0;
24         end
25
26         6'b100011:    begin                                          //lw
27              regDst = 0;       aluSrc = 1;
28              memToReg = 1;     regWrite = 1;
29              memRead = 1;      memWrite = 0;
30              branch = 0;       aluOp =  2'b00; jump = 0;
31         end
32
33         6'b101011:    begin                                          //sw
34              regDst = 1;       aluSrc = 1;
35              memToReg = 1;     regWrite = 0;
36              memRead = 0;      memWrite = 1;
37              branch = 0;       aluOp =  2'b00; jump = 0;
38         end
39
40         6'b000100:    begin                                         //beq
41              regDst = 0;       aluSrc = 0;
42              memToReg = 0;     regWrite = 0;
43              memRead = 0;      memWrite = 0;
44              branch = 1;       aluOp =  2'b01;
45              jmp = 0;
46         end
47
48         default: begin
49              regDst = 0;       aluSrc = 0;
50              memToReg = 0;     regWrite = 0;
51              memRead = 0;      memWrite = 0;
52              branch = 0;       aluOp =  2'b00; jump = 0;
53         end
54
55      endcase
56      end
57  endmodule
```

Ctr.v

1.3 实验中的问题和思考

该部分比较简单，在实现过程中注意设置了 default 情况下产生的输出与其他请款相区别，便于检查输入的错误（实际上输入几乎不会产生错误）。

1.4 仿真

使用 ModelSim 进行行为级仿真，仿真过程和仿真结果如下。

#	opCode
1	'b000010
2	'b000000
3	'b100011
4	'b101011
5	'b000100

Figure 3: 仿真指令

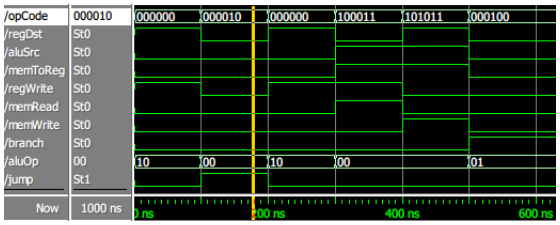

Figure 4: ModelSim Ctr 仿真波形

2 ALUCtr

2.1 译码逻辑

ALU 的控制信号采用 local decoding, 即通过 mainCtr 产生的 ALUop 和指令中的 funct field 产生 ALUctr 信号。

ALUOp	funct	ALUCtr
00	xxxxxx	0010
x1	xxxxxx	0110
1x	xx0000	0010
1x	xx0010	0110
1x	xx0100	0000
1x	xx0101	0001
1x	xx1010	0111

Table 2: ALUCtr 真值表

2.2 程序实现

```
1  module AluCtr(
2      input [1:0] aluOp,
3      input [5:0] funct,
4      output reg [3:0] aluCtr
5      );
6
7    always @ (aluOp or funct)
8      casex ({aluOp, funct})
9        8'b00xxxxxx: aluCtr = 'b0010;
10       8'bx1xxxxxx: aluCtr = 'b0110;
11       8'b1x100000: aluCtr = 'b0010;
12       8'b1x100010: aluCtr = 'b0110;
13       8'b1x100100: aluCtr = 'b0000;
14       8'b1x100101: aluCtr = 'b0001;
15       8'b1x101010: aluCtr = 'b0111;
16     endcase
17
18 endmodule
```

ALUCtr.v

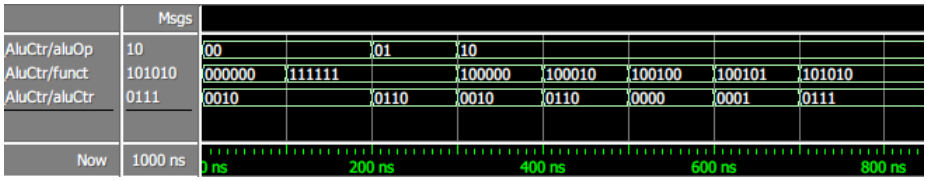

Figure 5: ModelSim ALTCtr 仿真波形

2.3 仿真

使用 ModelSim 进行行为级仿真，仿真过程和仿真结果见 Table 3, Figure 5。

#	ALUOp	funct	#	ALUOp	funct
1	00	000000	5	10	100010
2	00	111111	6	10	100100
3	01	111111	7	10	100101
4	10	100000	8	10	101010

Table 3: ALUCtr 仿真指令

2.4 思考

使用 casex 语句而不是 case 语句，利用了指令编码的特点，增加了设计的容错能力。

3 ALU

向 ALU 输入两个操作数和 ALUCtr 指令，ALU 输出计算结果，若结果为 0，还将 zero 置 1，

3.1 ALU 结构

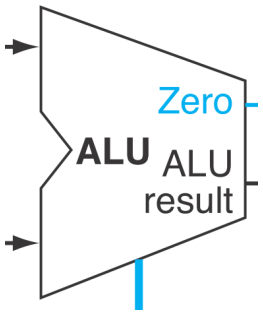

Figure 6: ALU

Control Lines	Function
0000	AND
0001	OR
0010	add
0110	substract
0111	set on less than
1100	NOR

Figure 7: ALU Fuction

3.2 程序实现

```
module Alu(
    input [31:0] input1,    input [3:0] aluCtr,
    input [31:0] input2,    output reg zero,
    output reg [31:0] aluRes
    );
   always @(input1 or input2 or aluCtr)
   case(aluCtr)
     'b0000:  begin                        //AND
    aluRes = input1 & input2;
    zero = 0;
    end
    'b0001:  begin                        //OR
    aluRes = input1 | input2;
    zero = 0;
    end
    'b0010:  begin                        //add
    aluRes = input1 + input2;
    zero = 0;
    end
    'b0110:  begin                        //sub
      aluRes = input1 - input2;
      if(aluRes == 0) zero = 1;
      else zero = 0;
    end
    'b0111:  begin    //set on less than
      zero = 0;
      if(input1<input2)  aluRes = 1;
      else aluRes = 0;
    end
    'b1100:  begin                       //NOR
      aluRes = ~(input1 | input2);
      zero = 0;
    end
  endcase
endmodule
```

ALU.v

3.3 仿真

使用 ModelSim 进行行为级仿真，仿真过程和仿真结果见 Table 4, Figure 8。

#	input1	input2	operation	#	input1	input2	operation
1	0	0	AND	4	1	1	SUB
2	255	170	AND	5	255	170	SLT
3	255	170	OR	6	0	1	NOR

Table 4: ALU 仿真指令

注：0–100ms 内为初始化后的值，程序还未开始执行，因此 zero 为 0，

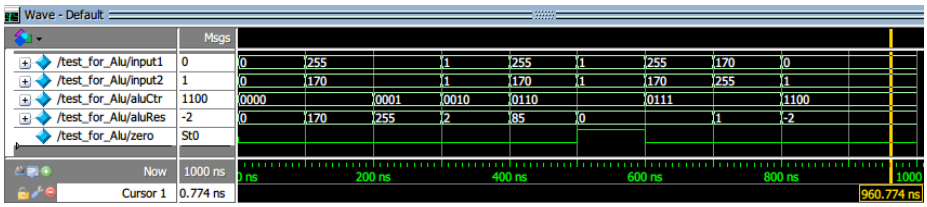

Figure 8: ModelSim ALU 仿真波形

3.4 思考

建议每一 case 之中都写明 zero，在没有指明时，综合过程会为 zero 生成一个无必要的锁存器。