数字逻辑与处理器基础大作业

聂浩　无 31　2013011280

2015 年 6 月 18 日

1　处理器结构

1.1　试回答以下问题：

a) 由 RegDst 信号控制的多路选择器，输入 2 对应常数 31。这里的 31 代表什么？在执行哪些指令时需要 RegDst 信号为 2？为什么？

答:31 代表 $ra 寄存器，表示在 $ra 里写下返回地址。jal 指令、和 jalr 指令，因为在执行这两条命令时需要把返回地址写入 $ra。

b) 由 ALUSrc1 信号控制的多路选择器，输入 1 对应的指令 [10-6] 是什么？在执行哪些指令时需要 ALUSrc1 信号为 1？为什么？

答: 偏移量。对应的指令是 sll，srl，sra，因为它们需要确定移的位数。

c) 由 MemtoReg 信号控制的多路选择器，输入 2 对应的是什么？在执行哪些指令时需要 MemtoReg 信号为 2？为什么？

答: 将下一条程序计数器 +4 之后的值写入内存。jal 和 jalr 需要用到，因为它们需要把返回地址写入寄存器。

d) 图中的处理器结构并没有 Jump 控制信号，取而代之的是 PCSrc 信号。PCSrc 信号控制的多路选择器，输入 2 对应的是什么？在执行哪些指令时需要 PCSrc 信号为 2？为什么？

答: 将寄存器读出的值写入程序计数器。jr 和 halr 需要，因为它们需要从寄存器中读出跳转的地址。

e) 为什么需要 ExtOp 控制信号？什么情况下 ExtOp 信号为 1？什么情况下 ExtOp 信号为 0？

	PCSrc[1:0]	Branch	RegWrite	RegDst[1:0]	MemRead	MemWrite	MemtoReg[1:0]	ALUSrc1	ALUSrc2	ExtOp	LuOp
lw	0	0	1	0	1	0	1	0	1	1	0
sw	0	0	0	x	0	1	x	0	1	1	0
lui	0	0	1	0	0	0	0	0	1	1	1
add	0	0	1	1	0	0	0	0	0	x	x
addu	0	0	1	1	0	0	0	0	0	x	x
sub	0	0	1	1	0	0	0	0	0	x	x
subu	0	0	1	1	0	0	0	0	0	x	x
addi	0	0	1	0	0	0	0	0	1	1	0
addiu	0	0	1	0	0	0	0	0	1	1	0
and	0	0	1	1	0	0	0	0	0	x	x
or	0	0	1	1	0	0	0	0	0	x	x
xor	0	0	1	1	0	0	0	0	0	x	x
nor	0	0	1	1	0	0	0	0	0	x	x
andi	0	0	1	0	0	0	0	0	1	0	0
sll	0	0	1	1	0	0	0	1	0	x	x
srl	0	0	1	1	0	0	0	1	0	x	x
sra	0	0	1	1	0	0	0	1	0	x	x
slt	0	0	1	1	0	0	0	0	0	x	x
sltu	0	0	1	1	0	0	0	0	0	x	x
slti	0	0	1	0	0	0	0	0	1	1	0
sltiu	0	0	1	0	0	0	0	0	1	1	0
beq	0	1	0	x	0	0	x	0	0	1	0
j	1	0	0	x	0	0	x	x	x	x	x
jal	1	0	1	2	0	0	2	x	x	x	x
jr	2	0	0	x	0	0	x	x	x	x	x
halr	2	0	1	1	0	0	2	x	x	x	x

表 1: 控制器真值表

答: 判断扩展时是否做补码处理。当为 andi 等与立即数位操作时为 0，在其他有 offset 和 imm 的情况下为 1。

f) 若想再多实现一条指令 nop（空指令），指令格式为全 0，需要如何修改处理器结构？

答: 不用修改，会直接执行 sll 指令。

1.2 根据对各控制信号功能的理解，填写真值表 1

2 完成控制器

2.1 CPU.v 实现了处理器的整体结构。阅读 CPU.v，理解其实现方式。

2.2 Control.v 是控制器模块的代码。完成 Control.v。

补充的代码为

```
// Your code below
assign  PCSrc[1:0]=
    (OpCode == 6'h02 || OpCode==6'h03)?2'b01:
    (OpCode==0&&(Funct==6'h08 || Funct==6'h09))
        ?2'b10:0;

assign Branch=
    (OpCode== 6'h04)?1:0;

assign RegWrite=
    (OpCode== 6'h2b || OpCode== 6'h04 || OpCode
        ==6'h02)?0:
    (Funct == 6'h08 && OpCode==0)?0:1;

assign RegDst=
    (OpCode== 6'h03)?2'b10:
    (OpCode== 6'h00)?2'b01:
```

```
16      2'b00;
17
18  assign MemRead=
19      (OpCode== 6'h23)?1:0;
20
21  assign MemWrite=
22      (OpCode== 6'h2b)?1:0;
23
24  assign MemtoReg=
25      (OpCode== 6'h03 || Funct==6'h09)?2'b10:
26      (OpCode== 6'h23)?1:0;
27
28  assign ALUSrc1=
29      (OpCode==0 && (Funct==6'h02 || Funct==6'h03
            || Funct==0))?1:0;
30  assign ALUSrc2=
31      (OpCode==0 || OpCode==6'h04)?0:1;
32
33  assign ExtOp=
34      (OpCode==6'h0c)?0:1;
35
36  assign LuOp=
37      (OpCode==6'h0f)?1:0;
38
39  // Your code above
```

2.3 阅读 InstructionMemory. v，根据注释理解指令存储器中的程序。这段程序执行足够长时间后会发生什么？此时寄存器 $a0 $a3, $t0 $t2,$v0 $v1 中的值应是多少？写出计算过程。注意理解有符号数、无符号数以及各种进制表示的数之间的关系。如果已知某一时刻在某寄存器中存放着数 0xffffcfc7，能否判断出它是有符号数还是无符号数？为什么？

答: 足够长的时间后会进入 Loop: j Loop 死循环。

$a0=0x00003039, $a1=0xffffd431,

$a2=0xD4310000, $a3=0xffffd431,

$t0=0xD4313039, $t1=0xfD431303,

$t2=0xffffcfc7, $v0=1,$v1=1.

无法判断，因为无符号数和有符号数都可以存储为这样。

2.4 使用 ModelSim 等仿真软件进行仿真。仿真顶层模块为 test_cpu，这是一个 testbench，用于向 CPU 提供复位和时钟信号。观察仿真结果中各寄存器和控制信号的变化。回答以下问题：

a) PC 如何变化？

答: 不断加 4，到 10000 时跳变到 11000，之后继续加 4 直到 101100 不再改变。

b) Branch 信号在何时为 1？它引起了 PC 怎样的变化？

答:400~500ns。PC 从 10000 跳变为 11000 而非 10100.

c) 100~200ns 期间，PC 是多少？对应的指令是哪条？此时 $a1 的值是多少？200~300ns 期间 $a1 的值是多少？为什么会这样？下一条指令立即使用到了 $a1 的值，会出现错误吗？为什么？

答:0x00000004。对应的是 addiu $a1, $zero, -11215。$a1 是 0。200~300ns $a1=0000_0000_0000_0000_1101_0100_0011_0001。因为右十六位是 -11215 的补码。虽然仿真正确，但是可能会出现错误，因为 regwrite 是

上升沿触发 $a1 的值是在下一个时钟上升沿才改变的，而读取也是以下一个时钟上升沿读取的，立即使用可能会有冲突。

d) 运行时间足够长之后（如 1100ns 时）寄存器 $a0 $a3,$t0 $t2,$v0 $v1 中的值是多少？与你的预期是否一致？

答: 和预计值一致。

3 执行汇编程序

3.1 如果第一行的 3 是任意正整数 n，这段程序能实现什么功能？Loop，sum，L1 各有什么作用？为每一句代码添加注释。

答: 计算 $\sum_{i=1}^{n} i$。Loop 使得程序最终死循环，不退出，sum 是求和函数的循环部分（类似 for 语句），L1 是求和中的计算部分和函数的返回部分。

```
addi $a0, $zero, 3          \\a0初始值为3
jal sum                     \\跳转到sum
    部分并保存当前PC到ra
Loop:
    beq $zero, $zero, Loop  \\一直循环
sum:
    addi $sp, $sp, -8       \\内存中设置栈空间
    sw $ra, 4($sp)          \\将ra入栈
    sw $a0, 0($sp)          \\将a0入栈
    slti $t0, $a0, 1        \\判断是否有a<0
    beq $t0, $zero, L1      \\a>=0则跳转至L1
    xor $v0, $zero, $zero   \\将v0初始化为0
    addi $sp, $sp, 8        \\栈指针加8
    jr $ra                  \\函数返回
L1:
    addi $a0, $a0, -1       \\a0=a0-1
    jal sum                 \\跳到sum并保存当前
        PC到ra
```

```
17      lw $a0, 0($sp)              \\重新读入保存的a0
18      lw $ra, 4($sp)              \\重新读入保存的ra
19      addi $sp, $sp, 8            \\栈指针加8
20      add $v0, $a0, $v0           \\$v0=$v0+$a0
21      jr $ra                      \\函数返回
```

3.2 将这段汇编程序翻译成机器码。对于 beq 和 jal 语句中的 Loop，sum，L1，你是怎么翻译的？立即数 -1、-8 被翻译成了什么

（用 16 进制或 2 进制表示）?

```
1  case (Address[9:2])
2      //addi $a0, $zero, 3
3          8'd0:    Instruction <= {6'h08, 5'd0 ,
               5'd4 , 16'h3};
4      //jal sum
5          8'd1:    Instruction <= {6'h03,26'd3};
6      //Loop:
7      //beq $zero, $zero, Loop
8          8'd2:    Instruction <= {6'h04,5'd0,5'
               d0,16'hffff};
9      //sum:
10     //addi $sp, $sp, -8
11         8'd3:    Instruction <= {6'h08,5'd29,5'
               d29,16'hfff8};
12     //sw $ra, 4($sp)
13         8'd4:    Instruction <= {6'h2b,5'd29,5'
               d31,16'h04};
14     //sw $a0, 0($sp)
15         8'd5:    Instruction <= {6'h2b,5'd29,5'
               d4,16'h00};
16     //slti $t0, $a0, 1
```

```
17          8'd6:     Instruction <= {6'h0a,5'd4,5'
                d8,16'h01};
18      //beq $t0, $zero, L1
19          8'd7:     Instruction <= {6'h04,5'd8,5'
                d0,16'h03};
20      //xor $v0, $zero, $zero
21          8'd8:     Instruction <= {6'h0,5'd0,5'd0
                ,5'd2,5'd0,6'h26};
22      //addi $sp, $sp, 8
23          8'd9:     Instruction <= {6'h08,5'd29,5'
                d29,16'h08};
24      //jr $ra
25          8'd10:    Instruction <= {6'h0,5'd31
                ,15'd0,6'h08};
26      //L1:
27      //addi $a0, $a0, -1
28          8'd11:    Instruction <= {6'h08,5'd4,5'
                d4,16'hffff};
29      //jal sum
30          8'd12:    Instruction <= {6'h03,26'd3};
31      //lw $a0, 0($sp)
32          8'd13:    Instruction <= {6'h23,5'd29
                ,5'd4,16'h0};
33      //lw $ra, 4($sp)
34          8'd14:    Instruction <= {6'h23,5'd29
                ,5'd31,16'h04};
35      //addi $sp, $sp, 8
36          8'd15:    Instruction <= {6'h08,5'd29
                ,5'd29,16'h08};
37      //add $v0, $a0, $v0
38          8'd16:    Instruction <= {6'h0,5'd4,5'
                d2,5'd2,5'd0,6'h20};
39      //jr $ra
```

```
40          8'd17:      Instruction <= {6'h0,5'd31
                ,15'd0,6'h08};
41      default: Instruction <= 32'h00000000;
42  endcase
```

答: 使 PC 的值变为 Loop，sum，L1 对应的指令地址即可跳转。-1 为 0xffff，-8 为 0xfff8（十六位）。

3.3 修改 InstructionMemory.v，使 CPU 运行上面这段程序。注意 case 语句的输入是地址的 [9- 2] 比特。仿真观察各控制信号和寄存器的变化。

a) 运行时间足够长之后（如 5000ns 时），寄存器 $a0,$v0 的值是多少？和你预期的程序功能是否一致？

答:$a0 为 0x00000003,$v0 为 0x00000006. 与预期一致。

b) 观察、描述并解释 PC,$a0,$v0,$sp,$ra 如何变化。

答: 按照周期记录值：PC: 一共 17 条指令，所以只记录仅记录 6 2 位，其余都是 0.

00000,00001, 初始化。

00011,00100,00101,00110,00111,01011,01100;

00011,00100,00101,00110,00111,01011,01100;

00011,00100,00101,00110,00111,01011,01100; 三次递归，直到 a0=0

00011,00100,00101,00110,00111,

01000,01001,01010; 初始化 v0，然后第一次返回。

01101,01110,01111,10000,10001;v0=a0+v0，然后返回，这样执行三次

01101,01110,01111,10000,10001;

01101,01110,01111,10000,10001,

00010; 进入 Loop 死循环

$a0 依次为（10 进制）3,2,1,0,1,2,3.a0 先每次递归减一，之后出栈则依次恢复上一层的值。

$v0 依次为（10 进制）0,1,3,6. 最后出栈时，每出一次栈，v0=v0+a0.

$sp 依次为 0x00000000,0xfffffff8,0xfffffff0,0xffffff78,0xffffff70, 之前都为入栈（四次）。

0xffffff78,0xfffffff0,0xfffffff8,0x00000000. 出栈。

$ra 同样只记录 6 2 位。

00000,00010,01101,00010. 第一次 um 时 ra 变为 00010，之后因为数次 jal sum 都是指令 12，所以此时 ra 一直为 01101，数次出栈 ra 也不会改变。最后一次出栈使得 ra 变为 0010，之后在死循环中 ra 不再改变。