# 数字逻辑与处理器基础大作业

单周期处理器

王晗
(2013011076)

2015 年 6 月 18 日

## 1 处理器结构

### 1.1 回答问题

**1. 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？**

31 代表 31 号寄存器 ($ra)。

在执行 jal 指令时需要 RegDst 信号为 2，将当前指令的下一句的指令存储器地址写入 $ra 寄存器，供程序跳转返回时使用。

（教材附录中说，jalr 中的 rd 默认值为 31，当未指定 rd 时，由汇编器将 rd 置为 31，或者由硬件将 RegDst 设为 2。）

**2. 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？**

指令 [10-6] 是移位偏移量 shamt[4:0]。

在执行 sll、srl、sra 指令时需要将 ALUSrc1 信号置为 1，因为这些指令涉及移位操作，需要通过 shamt[4:0] 传入移位量。

**3. 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？**

MemtoReg 为 2 时，表示将当前指令的下一句的指令存储器地址送入寄存器对写入数据总线。

在执行 jal、jalr 指令时需要 MemtoReg 信号为 2，将当前指令的下一句的指令存储器地址写入 $ra 寄存器，供程序跳转返回时使用。

**4. 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？**

PCSrc 信号为 2 时，表示将寄存器堆输出数据总线 1 上的数据作为下一条指令的指令存储器地址。
在执行 jr、jalr 指令时，需要 PCSrc 信号为 2，根据寄存器存储的地址进行跳转。

**5. 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？**

将 16 位数扩展为 32 位时，存在无符号扩展和有符号扩展两种扩展方式。
用最高位 (符号位) 填充高位时，ExtOp 为 1；否则 ExtOp 为 0。

**6. 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？**

空指令可以用 sll $0,$0,0 实现。不必改动现有的数据通路，只需将 nop 解释为从 0 号寄存器取值，左移 0 位，再存入 0 号寄存器即可。

## 1.2 填写真值表

| Instruction | 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 | x | 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 | 0 | 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 |

转下页...

接上页

| *Instruction* | *PCSrc[1:0]* | *Branch* | *RegWrite* | *RegDst[1:0]* | *MemRead* | *MemWrite* | *MemtoReg[1:0]* | *ALUSrc1* | *ALUSrc2* | *ExtOp* | *LuOp* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 | 0 | 0 |
| beq | 0 | 1 | 0 | x | 0 | 0 | x | 0 | 0 | 1 | 0 |
| j | 1 | x | 0 | x | 0 | 0 | x | x | x | x | x |
| jal | 1 | x | 1 | 2 | 0 | 0 | 2 | x | x | x | x |
| jr | 2 | x | 0 | x | 0 | 0 | x | x | x | x | x |
| jalr | 2 | x | 1 | 1 | 0 | 0 | 2 | x | x | x | x |

# 2 完成控制器

## 2.1 阅读 CPU.v

## 2.2 完成 Control.v

见文件 *Control.v*。

## 2.3 阅读 InstructionMemory.v

**这段程序执行足够长时间后会发生什么？**

程序停在 Loop: j Loop 语句。

**此时寄存器 $a0~$a3,$t0~$t2,$v0~$v1 中的值应是多少？写出计算过程。**

```
        addi $a0, $zero, 12345    $a0=0x00003039
        addiu $a1, $zero, -11215  $a1=0x0000d431
        sll $a2, $a1, 16          $a2=0xd4310000
        sra $a3, $a2, 16          $a3=0xffffd431
        beq $a3, $a1, L1          $a3≠$a1
        lui $a0, -11111           $a0=0xd4990000
  L1:   add $t0, $a2, $a0         $t0=0xa8ca0000
        sra $t1, $t0, 8           $t1=0xffa8ca00
        addi $t2, $zero, -12345   $t2=0xffffcfc7
        slt $v0, $a0, $t2         $a0<$t2<0, 故 $v0=0x00000001
        sltu $v1, $a0, $t2        0<$a0<$t2, 故 $v1=0x00000001
Loop:   j Loop
```

**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？**

不能。0xffffcfc7 可能表示无符号数 4294954951，也有可能表示有符号数-12345。无符号数和有符号数在寄存器中不可区分，最高位是否当作符号位由指令决定。

## 2.4 ModelSim 仿真

**PC 如何变化？**

每个时钟周期 PC 的值增加 4，当 PC 达到 0x2c(仿真时间 1100ns) 后保持不变。

**Branch 信号在何时为 1？它引起了 PC 怎样的变化？**

PC=0x10(仿真时间 400ns) 时 Branch 信号为 1。但由于 Zero 信号仍为 0，PC 移向下一条指令 (PC<=PC+4)。

**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？**

100~200ns 期间 PC=0x04，对应指令 addiu $a1, $zero, -11215，此时 $a1=0x00000000，200~300ns 期间 $a1=0x0000d431。

寄存器写入只发生在时钟沿上。100ns 上升沿时，写入信号还未产生，因此需要等到下一个上升沿才能写入寄存器 $a1。

下一条指令立即使用 $a1 的值并不会出现错误，$a1 将在时钟上升沿到来时立刻写入，寄存器写入的延时远小于下一条指令从指令存储器读出、译码产生控制信号、选择数据通路等消耗的时间。

**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？**

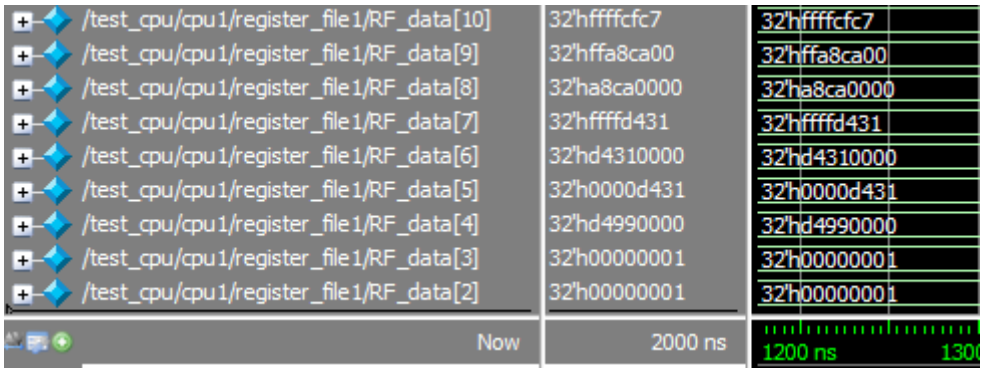


图 1: ASM1 运行足够长时间后寄存器中的值

从上图可以看出，ModelSim 仿真结果与计算结果一致。

# 3 执行汇编程序

## 3.1 阅读并理解汇编程序

**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？**

求 $\sum_{i=1}^{n} i$.

**Loop、sum、L1 各有什么作用？**

Loop、sum、L1 是 Label，用来标记指令地址，供汇编器在翻译跳转、分支指令机器码时使用。

本例中，Loop 是程序运行结束后停止的位置；sum 是递归深度增加时压栈操作开始的位置；L1 是递归深度控制变量 $a0 改变的位置，其后的语句完成了继续递归、回溯弹栈、计算返回值等功能。

为每一句代码添加注释。

```
        addi $a0, $zero, 3      $a0=3
        jal sum                 跳转至 sum，并修改 $ra
Loop:   beq $zero, $zero, Loop  停止在 Loop 处
 sum:   addi $sp, $sp, -8       栈指针下移 2 个字
        sw $ra, 4($sp)          $ra 入栈
        sw $a0, 0($sp)          $a0 入栈
        slti $t0, $a0, 1        $t0 = $a0 < 1
        beq $t0, $zero, L1      ~$t0 则跳转至 L1 继续递归
        xor $v0, $zero, $zero   $v0 清零
        addi $sp, $sp, 8        栈指针上移 2 个字
        jr $ra                  根据 $ra 存储的地址返回
  L1:   addi $a0, $a0, -1       $a0--
        jal sum                 跳转至 sum，并修改 $ra
        lw $a0, 0($sp)          $a0 出栈
        lw $ra, 4($sp)          $ra 出栈
        addi $sp, $sp, 8        栈指针上移 2 个字
        add $v0, $a0, $v0       $v0 = $a0 + $v0
        jr $ra                  根据 $ra 存储的地址返回
```

## 3.2 将汇编程序翻译为机器码

**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）？**

beq 语句中的地址字段是目标位置相对当前指令的下一条指令位置的偏移量 (字数)，因此 beq $zero, $zero, Loop 中的 Loop 应当翻译为-1，beq $t0, $zero, L1 中的 L1 应当翻译为 3。

jal 语句中的地址字段是目标地址的 [27:2] 位，因此 jal sum 中的 sum 应当翻译为 0x0000003。

立即数-1 翻译为 0xffff，-8 翻译为 0xfff8。

翻译后的机器码对应如下：

```
        addi $a0, $zero, 3      {6'h08, 5'd00, 5'd04, 16'h0003};
        jal sum                 {6'h03, 26'h0000003};
Loop:   beq $zero, $zero, Loop  {6'h04, 5'd00, 5'd00, 16'hffff};
 sum:   addi $sp, $sp, -8       {6'h08, 5'd29, 5'd29, 16'hfff8};
        sw $ra, 4($sp)          {6'h2b, 5'd29, 5'd31, 16'h0004};
        sw $a0, 0($sp)          {6'h2b, 5'd29, 5'd04, 16'h0000};
        slti $t0, $a0, 1        {6'h0a, 5'd04, 5'd08, 16'h0001};
        beq $t0, $zero, L1      {6'h04, 5'd00, 5'd08, 16'h0003};
        xor $v0, $zero, $zero   {6'h00, 5'd00, 5'd00, 5'd02, 5'd00, 6'h26};
        addi $sp, $sp, 8        {6'h08, 5'd29, 5'd29, 16'h0008};
        jr $ra                  {6'h00, 5'd31, 15'h0000, 6'h08};
  L1:   addi $a0, $a0, -1       {6'h08, 5'd04, 5'd04, 16'hffff};
        jal sum                 {6'h03, 26'h0000003};
        lw $a0, 0($sp)          {6'h23, 5'd29, 5'd04, 16'h0000};
        lw $ra, 4($sp)          {6'h23, 5'd29, 5'd31, 16'h0004};
        addi $sp, $sp, 8        {6'h08, 5'd29, 5'd29, 16'h0008};
        add $v0, $a0, $v0       {6'h00, 5'd04, 5'd02, 5'd02, 5'd00, 6'h20};
        jr $ra                  {6'h00, 5'd31, 15'h0000, 6'h08};
```

### 3.3 ModelSim 仿真

**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？**

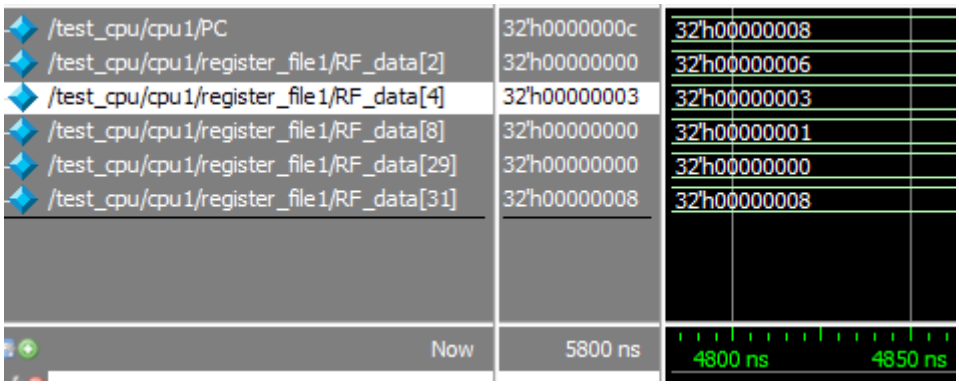


图 2: ASM2 运行足够长时间后寄存器中的值

从图中可以看出，运行足够长时间后，$a0=3, $v0=6, 和预期功能一致。

**观察、描述并解释 PC,$a0,$v0,$sp,$ra 如何变化。**

PC 的变化：除跳转外，每个时钟周期 PC<=PC+4。

0x00 → 0x04 $\xrightarrow{\text{跳转至 sum}}$ 0x0c → 0x10 → 0x14 → 0x18 → 0x1c $\xrightarrow{\text{跳转至 L1}}$ 0x2c → 0x30 $\xrightarrow{\text{跳转至 sum}}$

0x0c → 0x10 → 0x14 → 0x18 → 0x1c $\xrightarrow{\text{跳转至 L1}}$ 0x2c → 0x30 $\xrightarrow{\text{跳转至 sum}}$ 0x0c → 0x10 → 0x14

$\rightarrow$ 0x18 $\rightarrow$ 0x1c $\xrightarrow{\text{跳转至 L1}}$ 0x2c $\rightarrow$ 0x30 $\xrightarrow{\text{跳转至 sum}}$ 0x0c $\rightarrow$ 0x10 $\rightarrow$ 0x14 $\rightarrow$ 0x18 $\rightarrow$ 0x1c
$\rightarrow$ 0x20 $\rightarrow$ 0x24 $\rightarrow$ 0x28 $\xrightarrow{\text{跳转至 \$ra}}$ 0x34 $\rightarrow$ 0x38 $\rightarrow$ 0x3c $\rightarrow$ 0x40 $\rightarrow$ 0x44 $\xrightarrow{\text{跳转至 \$ra}}$ 0x34
$\rightarrow$ 0x38 $\rightarrow$ 0x3c $\rightarrow$ 0x40 $\rightarrow$ 0x44 $\xrightarrow{\text{跳转至 \$ra}}$ 0x34 $\rightarrow$ 0x38 $\rightarrow$ 0x3c $\rightarrow$ 0x40 $\rightarrow$ 0x44 $\xrightarrow{\text{跳转至 \$ra}}$ 0x08

\$a0 的变化：

0x00 $\xrightarrow{+3}$ 0x03 $\xrightarrow{\text{压栈，-1}}$ 0x02 $\xrightarrow{\text{压栈，-1}}$ 0x01 $\xrightarrow{\text{压栈，-1}}$ 0x00 $\xrightarrow{\text{弹栈}}$ 0x01 $\xrightarrow{\text{弹栈}}$ 0x02 $\xrightarrow{\text{弹栈}}$ 0x03

\$v0 的变化：

0x00 $\xrightarrow{\text{清零}}$ 0x00 $\xrightarrow[\text{第 3 层递归，回溯}]{+\$a0}$ 0x01 $\xrightarrow[\text{第 2 层递归，回溯}]{+\$a0}$ 0x03 $\xrightarrow[\text{第 1 层递归，回溯}]{+\$a0}$ 0x06

\$sp 的变化：

0x00 $\xrightarrow[\text{第 1 层递归}]{-8}$ 0xf8 $\xrightarrow[\text{第 2 层递归}]{-8}$ 0xf0 $\xrightarrow[\text{第 3 层递归}]{-8}$ 0xe8 $\xrightarrow[\text{第 4 层递归}]{-8}$ 0xe0
$\xrightarrow[\text{第 4 层递归，回溯}]{+8}$ 0xe8 $\xrightarrow[\text{第 3 层递归，回溯}]{+8}$ 0xf0 $\xrightarrow[\text{第 2 层递归，回溯}]{+8}$ 0xf8 $\xrightarrow[\text{第 1 层递归，回溯}]{+8}$ 0x00

\$ra 的变化：

0x00 $\xrightarrow[\text{\$ra<=Line2Addr}]{\text{Line1: jal sum}}$ 0x08 $\xrightarrow[\text{第 1 层递归}]{\text{压栈}}$ $\xrightarrow[\text{\$ra<=Line13Addr}]{\text{Line12: jal sum}}$ 0x34 $\xrightarrow[\text{第 2 层递归}]{\text{压栈}}$ $\xrightarrow[\text{\$ra<=Line13Addr}]{\text{Line12: jal sum}}$ 0x34 $\xrightarrow[\text{第 3 层递归}]{\text{压栈}}$
$\xrightarrow[\text{\$ra<=Line13Addr}]{\text{Line12: jal sum}}$ 0x34 $\xrightarrow[\text{第 4 层递归}]{\text{压栈}}$ $\xrightarrow{\text{丢弃栈顶}}$ $\xrightarrow{\text{弹栈}}$ 0x34 $\xrightarrow{\text{弹栈}}$ 0x34 $\xrightarrow{\text{弹栈}}$ 0x08