# 数字逻辑与处理器基础大作业

## 单周期处理器

无 42 陈誉博 2014011058

一、处理器结构

1. 试回答以下问题：

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？

答：31 代表$ra 寄存器；在执行 jal 指令的时候需要 RegDst 信号为 2，因为 jal 指令的作用包括将 PC+4 存入$ra 寄存器，需要对 31 号寄存器进行写入。

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？

答：输入 1 对应着 R 型指令中的 shamt 部分，代表位移量。在执行 sll，srl,sra 指令的时候需要 ALUSrc1 信号为 1，因为这些指令的功能是将某些值进行移位操作，需要读取指令中的 shamt 部分。

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？

答：输入 2 对应的是 PC+4；在执行 jal 和 jalr 指令的时候需要 MemtoReg 信号，因为这两条指令需要将 PC+4 写入寄存器。

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？

答：输入 2 对应的是 rs 寄存器中的数据；在执行 jr 和 jalr 指令的时候需要将$rs 中的数据存到 PC 中。

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？

答：因为在指令集当中立即数的扩展有符号扩展和无符号扩展两种形式；在符号扩展的情况下 ExtOp 为 1，无符号扩展的情况下 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？

答：在指令存储器后面加一个多路选择器，一端接存储器输出另一端接全 0，再在控制单元里面添加 Nop 控制信号控制多路选择器输出，如果 Nop==1，选择全 0，否则悬着指令寄存器的输出。

2. 根据对各控制信号功能的理解，填写如下真值表（填 0,1,2,x 等）。

| | PCSrc[1:0] | Branch | RegWrite | RegDst[1:0] | MemRead | MemWwrite | 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 | 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 | 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 |

二、完成控制器

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. Control.v 是控制器模块的代码。完成 Control.v。
3. 阅读 InstructionMemory.v，根据注释理解指令存储器中的程序。

```
MIPS Assembly

 0        addi $a0, $zero, 12345
 1        addiu $a1, $zero, -11215
 2        sll $a2, $a1, 16
 3        sra $a3, $a2, 16
 4        beq $a3, $a1, L1
 5        lui $a0, -11111
    L1:
 6        add $t0, $a2, $a0
 7        sra $t1, $t0, 8
 8        addi $t2, $zero, -12345
 9        slt $v0, $a0, $t2
10        sltu $v1, $a0, $t2
    Loop:
11        j Loop
```

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：执行足够长时间之后程序会停在 j Loop 这一句一直循环

0: $a0=$0+SignExt(0x3039)

$a0=0000_0000_0000_0000_0011_0000_0011_1001(0x00003039);

1: $a1=$0+SignExt(0xd431)(11215 二进制为 10101111001111，取二补码为 1111_1111_1111_1111_1101_0100_0011_0001，十六进制为 0xffffd431);

$a1=1111_1111_1111_1111_1101_0100_0011_0001(0xffffd431);

2: $a2=0xffffd431<<16;

$a2=1101_0100_0011_0001_0000_0000_0000_0000(0xd4310000);

3: $a3=0xffffd431>>16(高位补 1);

$a3=1111_1111_1111_1111_1101_0100_0011_0001(0xffffd431);

4: $a3=0xffffd431, $a1=0xffffd431, go to L1;

6: $t0=0xd4310000+0x00003039;

$t0=1101_0100_0011_0001_0011_0000_0011_1001(0xd4313039);

7: $t1=0xd4313039>>8(高位补 1)；

$t1=1111_1111_1101_0010_0011_0001_0011_0000(0xffd43130);

8: $t2=$0+SignExt(cfc7);

$t2=1111_1111_1111_1111_1100_1111_1100_0111(0xffffcfc7);

9: $a0=0x00003039, $t2=0xffffcfc7, $v0=0;

$v0=0000_0000_0000_0000_0000_0000_0000_0000(0x00000000);

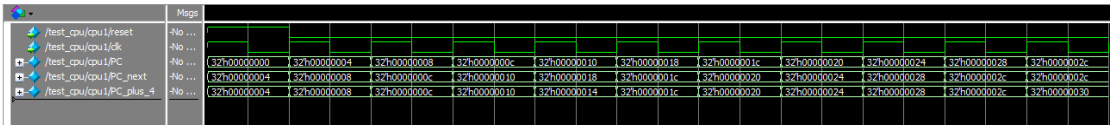
10: $a0=0x00003039, $t2=0xffffcfc7, 无符号条件下两者均视为正数，此时$v0=1;

$v0=0000_0000_0000_0000_0000_0000_0000_0001(0x00000001);

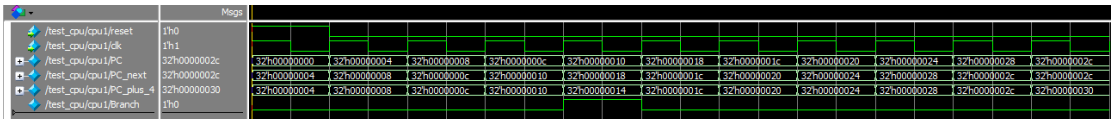
寄存器中的数值为 0xffffcfc7,则不能判断是有符号数还是无符号数，因为它有可能是一个负数，也有可能是扩充之后的无符号数。

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：

1) PC 如何变化？

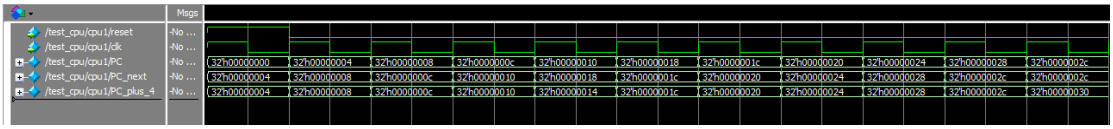


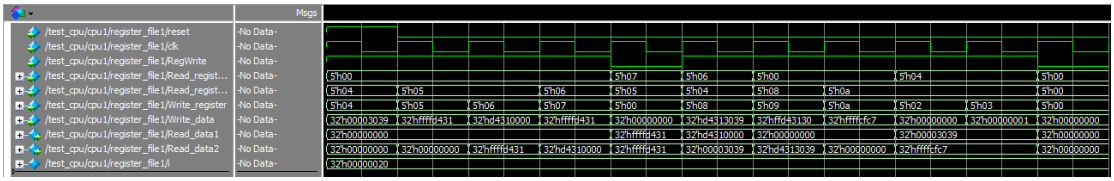

2) Branch 信号在何时为 1？它引起了 PC 怎样的变化？

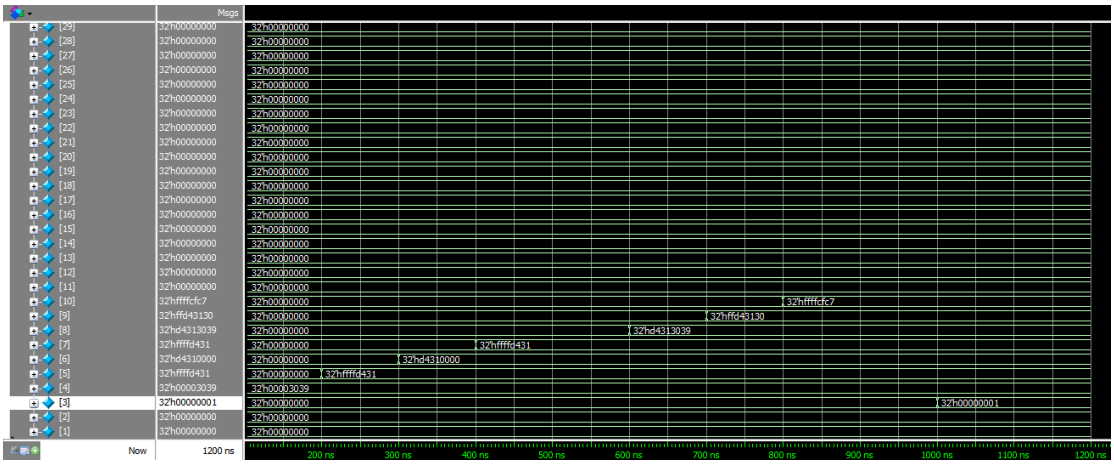



Branch 信号使得原本应该变为 32'h00000014 的 PC 信号变为了 32'h00000018.

3) 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？

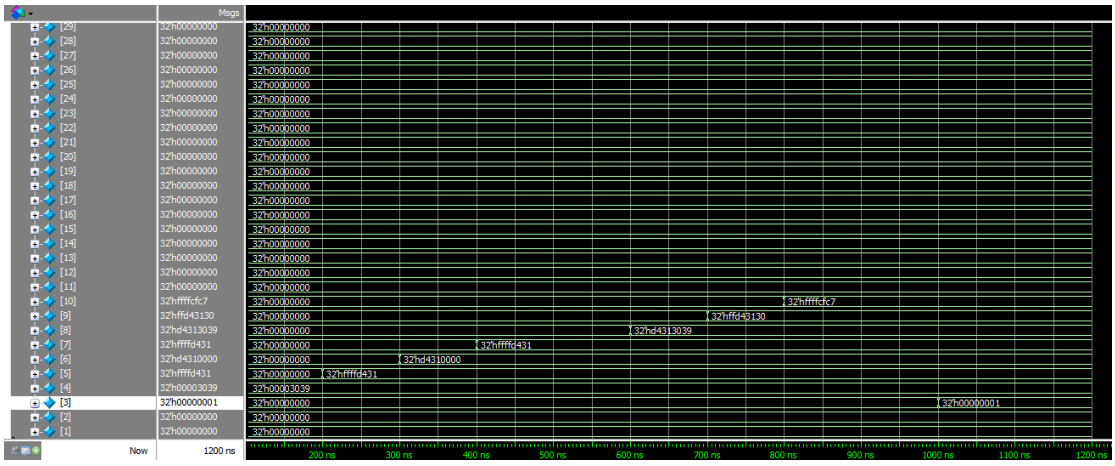



100ns-200ns 期间，PC 为 32'h00000004；对应的指令为:"`addiu $a1, $zero, -11215`"；此时$a1 的值为 32'h0,因为该条指令还没有执行完毕，$a1 的值还没有被写入；200ns-300ns 期间$a1=32'hfffffd431，因为此时第二条指令已经执行完毕，$a1 的值已经被写入为 32'hfffffd431.如果下一条指令用到了$a1,不会出现错误，因为每一条指令都是在上一条指令执行完毕之后才开始执行，不会出现数据冒险的问题。

4) 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？




如图，可以看出运行足够长时间后，各寄存器的值如下：

$a0(4)=32’h00003039

$a1(5)=32’hfffffd431

$a2(6)=32’hd4310000

$a3(7)=32’hfffffd431

$t0(8)=32’hd4313039

$t1(9)=32’hffd43130

$t2(10)=32’hfffffcfc7

$v0(2)=32’h00000000

$v1(3)=32’h00000001

和预期完全一致

三、执行汇编程序

阅读并理解下面这段汇编程序。

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。
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）？
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。

1) 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？

2) 观察、描述并解释 PC,$a0,$v0,$sp,$ra 如何变化。

MIPS Assembly

```
 0        addi $a0, $zero, 3
 1        jal sum
    Loop:
 2        beq $zero, $zero, Loop
    sum:
 3        addi $sp, $sp, -8
 4        sw $ra, 4($sp)
 5        sw $a0, 0($sp)
 6        slti $t0, $a0, 1
 7        beq $t0, $zero, L1
 8        xor $v0, $zero, $zero
 9        addi $sp, $sp, 8
10        jr $ra
    L1:
11        addi $a0, $a0, -1
12        jal sum
13        lw $a0, 0($sp)
14        lw $ra, 4($sp)
15        addi $sp, $sp, 8
16        add $v0, $a0, $v0
17        jr $ra
```

答：

1. 注释：

```
    addi $a0, $zero, 3          %% $a0=3
    jal sum                     %% jump to sum, $ra=PC+4
Loop:
    beq $zero, $zero, Loop      %% Loop forever
sum:
    addi $sp, $sp, -8           %% push stack
    sw $ra, 4($sp)
    sw $a0, 0($sp)
    slti $t0, $a0, 1            %% if $a0<1 $t0=1
    beq $t0, $zero, L1          %% if $t0=0 jump to L1
    xor $v0, $zero, $zero       %% $v0=0
    addi $sp, $sp, 8
    jr $ra                      %% jump back to main
L1:
    addi $a0, $a0, -1           %% $a0=$a0-1
    jal sum                     %% jump to sum, $ra=PC+4
    lw $a0, 0($sp)              %% pop stack
    lw $ra, 4($sp)
    addi $sp, $sp, 8
    add $v0, $a0, $v0           %% $v0=$a0+$v0
    jr $ra                      %% jump back to the last stack
```

功能：计算 1+2+3+……n

Loop:死循环

sum:存储中间变量、压栈、设定结束条件

L1:弹栈，循环计算求和的值

2. 机器码:

0:00100000000001000000000000000011(6’h08 5’d0 5’d4 16’h0003)

1:00001100000000000000000000000011(6’h03 26’d3)

2:00010000000000001111111111111111(6’h04 5’d0 5’d0 16’hffff)

3:00100011101111011111111111111000(6’h08 5’d29 5’d29 16’hfff8)

4:10101111101111110000000000000100(6’h2b 5’d29 5’d31 16’h0004)

5:10101111101001000000000000000000(6’h2b 5’d29 5’d4 16’h0000)

6:00101000100010000000000000000001(6’h0a 5’d4 5’d8 16’h0001)

7:00010001000000000000000000000011(6’h04 5’d8 5’d0 16’h0003)

8:00000000000000000001000000100110(6’h00 5’d0 5’d0 5’d2 5’d0 6’h26)

9:00100011101111010000000000001000(6’h08 5’d29 5’d29 16’h0008)

10:00000011111000000000000000001000(6’h00 5’d31 5’d0 5’d0 5’d0 6’h08)

11:00100001000010011111111111111111(6’h08 5’d4 5’d4 16’hffff)

12:00001100000000000000000000000011(6’h03 26’d3)

13:10001111101001000000000000000000(6’h23 5’d29 5’d4 16’h0000)

14:10001111101111110000000000000100(6’h23 5’d29 5’d31 16’h0004)

15:00100011101111010000000000001000(6’h08 5’d29 5’d29 16’h0008)

16:00000000100000100001000000100000(6’h00 5’d2 5’d4 5’d2 5’d0 6’h20)

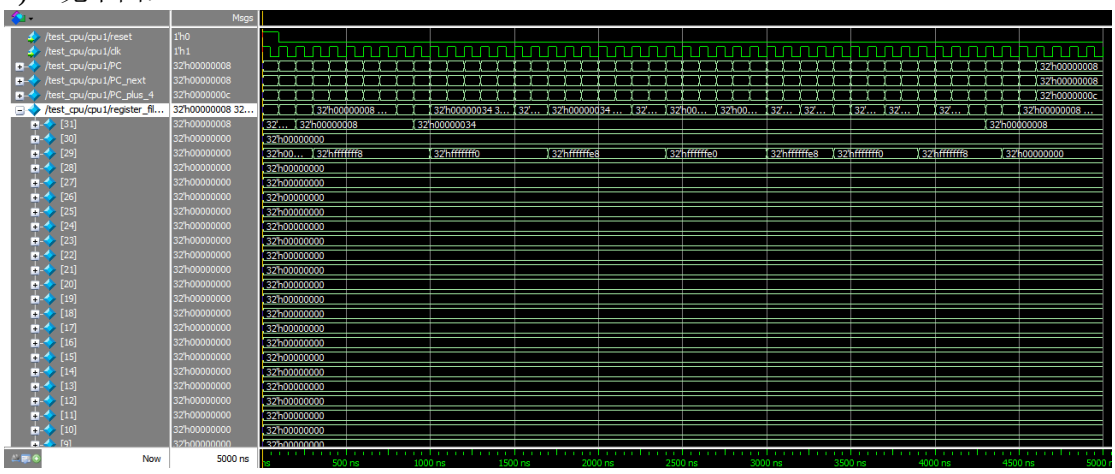
17:00000011111000000000000000001000(6’h00 5’d31 5’d0 5’d0 5’d0 6’h08)

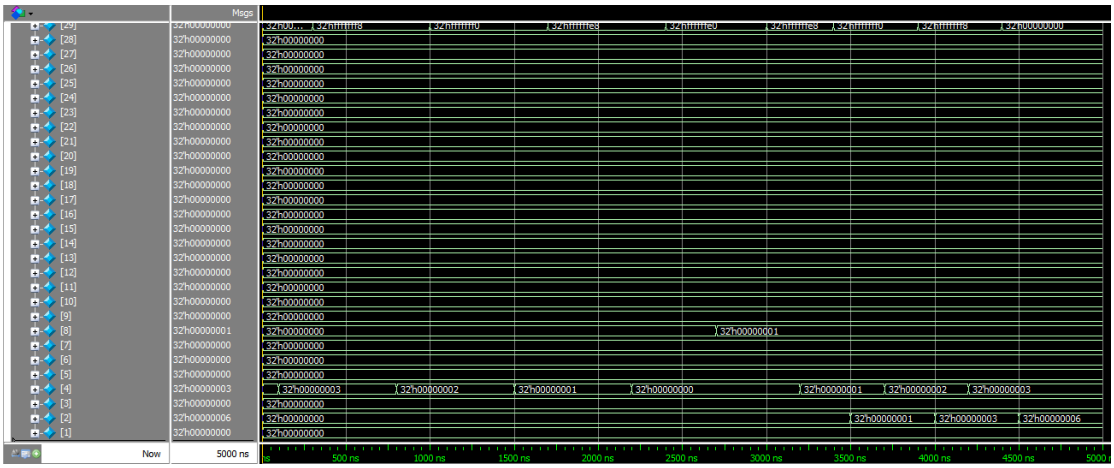
在指令里面的跳转写的是 Loop、sum、L1 这些标记，但实际上效果相当于跳转到标记之下的第一条指令继续执行。在这三种标记中，Loop 和 L1（beq 跳转）为相对寻址，偏移量为-1 和 3，所以翻译为 16’hffff 和 16’h0003;sum（jal 跳转）为直接寻址，直接跳转到第三条指令，所以翻译为 16’0003.

-1 翻译为 16’hffff，-8 翻译为 16’hfff8.

3.

1) 见下图：





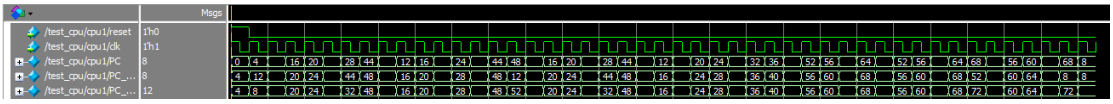
$v0(2)=32'h00000006

$a0(4)=32'h00000003

预计的程序功能为计算 1+2+……n 的值，$v0 的值为求和结果，$a0 的值为最后一个被加上去的数，所以输出结果和预计功能一样。

2)

**PC 的变化：**



（第三行为当前 PC，第四行为下一个 PC，第五行为当前 PC+4）

PC 在一开始从 0 变为 4，遇到了 jal 指令，此时 PC 跳到了 12，即第三条指令。然后开始执行 sum 下面的语句，PC 一直+4 直到第七条指令为 beq，PC 跳到了 44，即第 11 条指令。跳到 11 条指令之后 PC 又+4 跳到了 jal 指令，接下来进入循环压栈的过程。当执行到最后一次循环的 beq 指令的时候，不满足跳转条件，所以 PC 从 28 变为 32，而不是原来的 44。在执行到第 10 条指令的时候，此时 PC 为 40，下一步要跳转到$ra 寄存器存储的值，即为第 12 条 jal 指令的地址+4，所以下一步 PC 为 52.之后进入循环弹栈过程。弹栈结束之后，PC 跳到第一条 jal 指令对应的$ra 寄存器的值，为 8，程序进入死循环，计算过程结束。

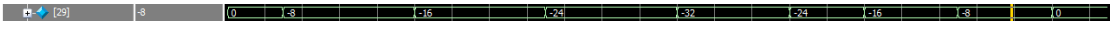
**$a0 的变化：**

第一条指令执行结束之后，$a0 的值等于 3。之后经过五个周期之后，$a0 的值-1，此时 PC 为 48，对应刚刚执行完的第 11 条指令。之后进入循环压栈的过程，每经过 7 个周期，$a0 的值都会减一，直至等于 0.从$a0 刚刚变为 0 经过 7 个周期之后，beq 指令的跳转条件不满足，继续执行了 3 条指令，直到第 13 条指令，将之前存在内存中的$a0 的值加载到$a0 寄存器中，$a0 变为 1.之后进入循环弹栈的过程，每过五个周期$a0 的值加 1。当弹栈过程结束之后，程序进入 Loop 死循环，$a0 的值保持不变，为 3.

**$v0 的变化：**

$v0 的第一次变化发生在第一次弹栈的过程中，在从内存加载$a0 的值之后又执行了三条指令，此时$v0=$v0+$a0,从 0 变成了 1；由于每个弹栈的过程执行五条指令，所以每过五个周期，$v0 的值都会增加$a0 当前的数值。程序执行结束之后，$v0 保持不变，为 1+2+3=6.

**$sp 的变化：**



当程序执行了三条指令之后$sp 发生了变化。由于每个压栈的过程执行 7 条指令，所以每过 7 个周期$sp 发生一次变化。当$sp 为-32 时，$t0 为 0，beq 语句不发生跳转，只需要再经过 6 个周期，$sp 就会发生变化。由于每个弹栈的过程执行 5 条指令，所以每过 5 个周期$sp 就会发生一次变化。（第一次弹栈没有经过第十七条指令的跳转，所以只需要四个周期）

**$ra 的变化：**


经过了前两个周期之后，jal 指令执行完毕，$ra=当前 PC+4，为 8.又经过七个周期，第 12 条 jal 指令执行完毕，$ra=当前 PC+4，为 52.在循环压栈和循环弹栈的过程中，由于每次存入和装载的$ra 的值始终对应第 13 条指令，所以一直保持$ra=52 不变。最后一次弹栈过程，被载入$ra 的值时在$ra 还没有变为 52 之前$ra 的值，所以$ra=8 直至程序运行结束。