

3. 7. 8. 12. 13. 14. 15. 16. 17

3. (1) ~~nop~~ addi $x_0, x_0, 0$

(2) ret jalr $x_0, x_1, 0 \Leftrightarrow x_1, ra$ 返回地址
 $pc \leftarrow pc + Regs[x_1]$

(3) call offset 远程调用了程序

anipo $x_1, offset[31:12]$

jalr $x_1, x_1, offset[11:0]$

(4) mv rd, rs addi rd, rs, 0

(5) rdcycle rd crrs rd, cycle, x_0

(6) sext.w rd, rs addiw rd, rs, 0

7 (1) add to, t1, t2

sub t3, to, t1

addi t4, t2, 0

bne t3, t4, overflow

(2) ~~addi t3, zero, 4294967295~~

bgt zero, to, overflow

(3) X86 通过 ~~叶架构~~ 将标志位来测：由符号数相加时

超过了有符号数所表示的范围，叶标志位置 1

ARM 架构中的 ALU (算术逻辑单元) 也有一个溢出标志位，用于检测加减法溢出。有符号数相加或相减时， $-V \oplus 1$

$$8. \quad OP = DIVU \quad OP = REMU \quad OP = DIV \quad OP = REM$$

X X

$0x1111111111111111$ $0x1111111111111111$

会引发异常，CPU会抛出异常，中断当前正在执行的程序

意义 1. 避免错误的计算结果

2. 增强代码的可靠性，将此视为异常，可以帮助程序员在开发和调试

过程中更容易地发现与排除错误

3. 确保程序的可移植性，可确保代码在不同的处理器和操作系统上的

② 无效操作。发生 ^{无效操作}，非法操作，此值被置 1

DZ 除以 0：当除数为 0，此值被设置 1

溢出：结果的绝对值大，无法被表示，置 1

下溢，太小，置 1

不精确 结果不能准确地用浮点数表示，置 1

不会直接陷入，但一些情况下需要由操作系统的异常处理程序来处理

(3) x86：除数为 0 会引发 “divide error” 异常，即 CPU 异常。这是一种硬件异常，当发生这种异常时，CPU 会将异常号压入堆栈中，并跳转到异常处理程序中。在异常处理程序中，可以通过读取状态寄存器中的标志位来确定发生异常的原因，并采取相应的处理措施。

ARM：除数 0 会引发 “division by zero” 异常，即未定义指令异常。当发生这种异常时，CPU 会将异常号压入堆栈中，并跳转到异常处理程序中。在异常处理程序中，可以通过读取协处理器中的状态寄存器来确定发生异常的原因。

- 12.
- (1) Linux Kernel S
 - (2) BootROM M
 - (3) BootLoader M
 - (4) USB Driver S
 - (5) vim U

vecMul.

13

add	X_{11}, X_0, X_0	# $i=0$	addi	$sp, sp - 32$
addi	$X_{12}, X_0, 1W$	# $X_{12} = 100$	sd	$ra - 24(sp)$
			sd	$so, 16(sp)$
			addi	$so, sp, 32$

Loop:

bge $X_{11}, X_{12}, \text{exit}$
 sll $X_{13}, X_{11}, 2$ # $i \times 4$
 add X_{15}, X_{13}, t_0 # &ofA + i
 add X_{14}, X_{13}, t_1 # &ofB + i
 lw $X_{15}, 0(X_{15})$ # CA + i
 lw $X_{14}, 0(X_{14})$

mul X_{15}, X_{14}, t_2

addi $X_{11}, X_{11}, 1$

j Loop

sw $X_{15}, 0(X_{13})$

exit: ld $ra, 24(sp)$

ld $so, 16(sp)$

addi $sp, sp, 32$

ret.

Part:1

14.

bgt a₀, a₁, Part 2

sub a₂, a₀, a₁

Part:2 j exit

add a₂, a₀, a₁

j exit

exit:

15. addi t₂, t₀, 0 # $t_2 = t_0 = p$
 lw t₂, 0(t₂) # $t_2^* = p^*$
 addi t₂, zero, t₀ # $p[0] = p$
 sw t₂, 0(t₀) # 存储进 t₀
 addi t₃, t₂, t₀, 4 # t₃ 指向 PLI
 lw t₃, 0(t₃) # *t₃
 addi t₃, zero, t₁ # *t₃ 是 p[4] = a
 addi t₄, t₃, sw t₃, 0(t₃)
 sll t₄, t₁, 2 # a * 4
 addl t₅, t₀, t₄ # t₅ 指向 PLA
 lw t₅, 0(t₅) # *t₅
 addi t₅, zero, t₁ # p[4] = a
 sw t₅, 0(t₅)

swap

16. lw t₂, 0(t₀) # $t_2 = *a$
 lw t₃, 0(t₁) # $t_3 = *b$
 addi t₄, t₂, 0 # $t_4 = \text{tmp} = *a$
 addi t₂, t₃, 0 # *a = *b
 addi t₃, t₄, 0 # *b = tmp
 sw t₂, 0(t₀)
 sw t₃, 0(t₁)

ret