3. nop → addi x0, x0, 0

ret → jalr x0, x1, 0

call offset → auipc x1, offset[31:12]

mv rd, rs → addi rd, rs, 0

rdcycle rd → csrrs rd, cycle, x0

sext.w rd, rs → addiw rd, rs, 0.

7. (1) slt t3, t1, t0

slt t4, t0, t2

(2) sub t3, t0, t2

bne t1, t3, overflow.

(3) x86架构使用方法如下：

1. 使用嵌入汇编代码检查处理器的 Overflow 标记

2. 将32位加法转换成64位加法，然后与32位加法值比较

3. 对运算结果所在区间进行判断和验证：由于补码的定义导致一旦发生溢出结果的区间是可以预先判断的。eg. 两正数相加，溢出结果一定小于其中任一操作数；两负数相加，溢出结果一定大于其任一操作数.

ARM架构通过使用标志位来检测：

eg：C标志：无符号加法溢出设为1，否则为0

V标志位：有符号数加法溢出设为1，不则为0

8：

	OP=DIVU	OP=REMU	OP=DIV	OP=REM
1.	2^{XLEN} 两所有位置1	X	-1	X.

2) IV 非法操作

DZ 除0

OF 上溢

UF 下溢

IX 不精确

不会：一旦相应置位，将会通过以下方式处理

1.触发浮点数异常处理程序

2.生成浮点异常

3.忽略浮点异常

其会根据设置的异常处理方式和对应程序中的代码逻辑来确定下一步操作

3) ARM架构中，如果除数为0将会产生异常，具体来说有以下两种异常

1."UDIV BY ZERO"异常：无符号除数为0时，会产生此异常

2."SDIV BY ZERO"异常，当有符号除法指令（SDIV）的除数为最小负数（-2^(n-1)）且除数为-1时会产生此异常

发生异常时，处理器会中止当前指令的执行，并引导到异常处理例程中执行相应操作，例如向异常处理程序发送中断信号，以在特定的异常处理模式下执行

x86：如果除数为0，将会引发一个异常，称为"除以0异常"，这个异常将会导致CPU引导到一个预设的异常处理例程，其中包含一段处理除以0的代码，这个例程会发送一个中断，让操作系统处理这个异常

12：M级别 机器模式

M级别

M级别

S级别

U级别

13

```
VecMul:
start: addi sp, sp, -32
       sd   ra, 24(sp)
       sd   s0, 16(sp)
       addi s0, sp, 32
part1: add  x1, x0, x0
       addi a2, x0, 100

       j part2
part2: bge  x1, a2, exit
       sll  a3, x1, 2
       add  a3, a3, t0
       sll  a4, x1, 2
       add  a4, a4, t1

       lw   x2, 0(a4)
       lw   x3, 0(t2)
       mul  a5, x2, x3
       sw   a5, 0(a3)
       addi x1, x1, 1
       j    part2
exit:  lw   a5, 0(t0)
       mv   a0, a5
       ld   ra, 24(sp)
       ld   s0, 16(sp)
       addi sp, sp, 32
       ret
```

14:

```
start: addi sp, sp, -32
       sd   ra, 24(sp)
       sd   s0, 16(sp)
       addi s0, sp, 32
part1 :
       bge  a1, a0, part2
       sub  a3, a0, a1
       j end
part 2 :
       add  a3, a0, a1
       j end
end :
       ld   ra, 24(sp)
       ld   s0, 16(sp)
       addi sp, sp, 32
       ret
```

15:

```
P[0]=P:   li a1, P
          sw a1, 0(t0)
int a=3   li a2, 3
          sw a2, 0(t1)
P[1]=a ;  addi a3, t0, 4
          lw a2, 0(t1)
          sw a2, 0(a3)
```

~~Part1:~~

~~Part1: add a3, x0, x0~~

P[a]=a : add a4, x0, x0

Part1: lw a5, 0(t1)

bltu a4, a5, Part2

sw a5, 0(a3)

...

Part2: slli a3, a4, 2

addi a4, a4, 1

add a3, a3, t0

j part1

16: swap: lw a1, 0(t0)

lw a2, 0(t1)

sw a2, 0(t0)

sw a1, 0(t1)

~~ret~~

17: 算出2^{30}并将其存入a_1