3. 解: (1) addi x0, x0, 0

(2) jalr x0, x1, 0

(3) auipc x1, offset[31:12]
jalr x1, x1, offset[11:0]

(4) addi rd, rs, 0

(5) csrrs rd, cycle, x0

(6) addiw rd, rs, 0

7. 解: (1) SLti t_3, t_2, 0
SLt t_4, t_0, t_1

(2) add t_0, t_1, t_2
bltu t_0, t_1, overflow

(3) x86架构中加法溢出标志位为OF。两个有符号数相加发生溢出时，OF置为1，反之则为0；

ARM架构中加法溢出标志位为V。两个有符号数相加发生溢出时，V置为1，反之则为0；

指令集架构需要设置专门的标志位，才能检测加法溢出

8. 解: (1)

指令	rs1	rs2	Op=DLVU时 rd值	Op=REMU时 rd值	Op=DLV时 rd值	Op=REM时 rd值
Op rd, rs1, rs2	x	0	$2^{32}-1$	x	发生异常 无法确定	发生异常 无法确定

采取这样设计的原因：实现灵活的计算功能，发生错误时抛出异常

DLVU和REMU处理无符号数，当被除数不为0而除数为0时，rd的值为 $2^{32}-1$ 或余数为被除数，此时不会抛出异常信息。

DLV和REM处理有符号数，当被除数不为0而除数为0时，rd的值无法确定，出现"division by zero"的异常

(2) 第0位 IW 表示无效操作；第一位 UF 表示下溢；第二位 OF 表示上溢；第三位 DZ 表示被零除；第四位 PX 表示失真

fflags 被置位不会导致处理器陷入系统调用，会有相关程序来处理这些异常

(3) x86、ARM 等指令集架构遇到除数为0的情况时，CPU 将停止运行，跳转至相应的异常处理程序进行处理。

12. 解：(1) 机器模式

(2) 机器模式

(3) 机器模式

(4) 管理员模式

(5) 用户模式

13. 解：VecMul：

```
        addi  sp, sp, -16
        sw    ra, 0(sp)
        sw    s0, 4(sp)
        li    t5, 100
loop:
        lw    s0, (t1)
        mul   t4, t0, t2
        sw    t4, (t0)
        addi  t1, t1, 4
        addi  t0, t0, 4
```

```
addi    x5, x5, -1
bnez    x5 .loop
lw      a0 , (a)
lw      ra , 8(sp)
lw      s0 , 12(sp)
addi    sp , sp, 16
ret
```

4. 解：

```
    ble  a1, a0, L1
    sub  a2, a0, a1
    j    L2
L1:
    add  a2  a0  a1
    j    L2
```

15. 解：

```
sw   t0   0(t0)          # p[0] = p
li   t1   3              # int a = 3
sw   t0 , -20(s0)
lw   a5 , -20(s0)
addi a5 , a5 , 4
sw   t0 , 0(a5)          # p[1] = a
addi a5 , a5 , 8
sw   t0 , 0(a5)          # p[2] = a
```

16. 解：

```
swap :
    lw   t2 , 0(t0)
    lw   t3 , 0(t1)
    sw   t3 , 0(t0)
    sw   t2 , 0(t1)
    jr   ra
```

17. 解：寄存器 a_0 的值初始化为 0

寄存器 a_1 的值初始化为 1

寄存器 a_2 的值初始化为 30

如果 a_0 的值等于 a_2 的值，则退出

否则 a_1 的值乘 2，a_0 的值加 1，进入循环，直到 a_0 与 a_2 的值相等才退出