3.1）nop：等效于 addi x0,x0,0（将寄存器 x0 和 0 相加，结果仍存储在 x0 中）
2）ret：等效于 jalr x0,x1,0（跳到 x1 寄存器指向的地址）
3）call offset：等效于 auipc x6,offset
jalr x1,x6,offset
4）mv rd,rs：等效于 addi rd,rs,0 （将 rs 寄存器的值加上 0，结果存储在 rd 寄存器中，实现了将 rs 寄存器的值复制到 rd 寄存器的目的）
5）rdcycle rd：等效于 csrrs rd, cycle, x0 （周期数读取指令）
6）sext.w rd,rs：等效于 addiw rd,rs,0 （将 rs 寄存器的低 16 位符号扩展为 32 位，并将结果存储在 rd 寄存器中）

7.1）sub t3,t0,t1
addi t4,t2,0
2） add t0, t1, t2
bltu t0, t1, overflow
bltu t0, t2, overflow
无符号数的加法溢出表现为结果小于其中一个加数，如果结果小于其中一个加数，则说明加法发生了溢出，跳转到 overflow 处理代码处。
3）x86：当进行一次加法操作时，CPU 会根据加数和被加数的符号及结果的符号位（即最高位）是否相同，以及进位是否发生，来判定是否发生了溢出。如果发生了溢出，OF 标志位就会被设置为 1，否则为 0。程序可以通过检查 OF 标志位来判断加法是否发生了溢出，并做出对应的处理。
ARM：ARM 使用状态寄存器（CPSR）中的标志位来检测加法溢出。在执行加法指令之后，ARM 会将结果和溢出标志存储在 CPSR 中。如果结果无法用 32 位的有符号整数表示，则 CPSR 的溢出标志将被设置为 1，否则为 0。如果发生了加法溢出，那么可以使用 CPSR 来查看此状态并相应地调整程序行为。

8.1)

指令	rs1	rs2	Op=DIVU 时 rd 值	Op=REMU 时 rd 值	Op=DIV 时 rd 值	Op=REM 时 rd 值
Op rd,rsl,rs2	x	0	0xffffffffffffffff	求余结果为被除数	0xffffffffffffffff	求余结果为被除数

2)
NV：无效操作数异常。
DZ：除以零异常。
OF：上溢出异常。
UF：下溢出异常。
NX：不精确结果异常。
当 FFLAGS 被置位后，处理器不会陷入系统调用。相反，处理器将在下一条指令完成之后检查 FFLAGS，如果 FFLAGS 中任何标志位已被设置，则将抛出浮点异常，并将 PC 寄存器设置为相应的异常处理程序，以便程序员可以处理异常并采取适当的措施。
3）x86 架构：在 x86 架构中，除法指令 DIV 和 IDIV 用于执行带符号和无符号整数除法。如果除数为 0，将导致一个异常，称为“被零除异常”。在发生异常时，执行期间的异常处理程序可以捕获该异常并采取适当的措施。
ARM 架构：在 ARM 架构中，如果发生除以零操作，则处理器将抛出一个“系统级故障”（system-level fault），并在异常向量表中查找相应的异常处理程序。处理器会将控制权

转移到异常处理程序，该程序可以选择终止当前进程或线程，也可以采取其他适当的行动。

12. Linux Kernel: Machine Mode
BootROM: Machine Mode
BootLoader: Supervisor Mode 和 User Mode
USB Driver: Supervisor Mode。
vim: User Mode

13.

```
.text
.align 2
.global vecMul

vecMul:
    addi sp, sp, -32
    sd   ra, 24(sp)
    sd   s0, 16(sp)
    addi s0, sp, 32

    mv t5, a0
    mv t6, a1

    # 循环 i = 0, i < 100
    li t1, 0                # t1 循环计数器 I,将其赋值 0
    li t2, 100              # t2 循环终止条件

    Loop:
    beq t1, t2, end_loop            #当 i >= 100，跳转到 end_loop

    # 加载 B[i] 到寄存器 t3
    lw t3, 0(t6)

    # 将 B[i] 与 C 相乘，结果存放到寄存器 t4
    mul t4, t3, a2

    # 将结果存放到 A[i]
    sw t4, 0(t5)

    # 下一个元素
    addi t5, t5, 4
    addi t6, t6, 4
    addi t1, t1, 1
    j Loop
```

```
end_loop:
# 加载第一个元素到返回寄存器 a0
lw a0, 0(a0)
ld    ra,24(sp)
ld    s0,16(sp)
addi sp,sp,32
ret
```

14.
```
bge a1, a0, 1f        #有符号大于等于分支指令
add a2, a0, a1        # c = a + b
j end
1f:
sub a2, a0, a1        #c = a - b
j end
end:
// Dumping point is here.
```

15.
```
li t1, 3              #a = 3
sw t1, 4(t0)          #p[1] = a
#计算 p[a]的地址
addi a3, x0, 0        #设置循环变量 i = 0
add a4, x0, t1        #设置循环终止条件
addi a5, t0, 0        #保存 p[0]地址在 a5 中
Loop:
beq a3, a4, end       #判断循环是否终止跳转
addi a5, a5, 4        #地址后移 4 位
addi a3, a3, 1        #i + =1
j Loop

end:
sw t1, 0(a5)          #p[a] = a
```

16.
```
.text
.align 2
.global vecMul

swap:
addi sp, sp, -32
sd    ra, 24(sp)
sd    s0, 16(sp)
addi s0, sp, 32
```

```
mv t0, a0
mv t1, a1
lw t2, 0(t0)          #int tmp = *a
lw t3, 0(t1)          #*a = *b
sw t3, 0(t0)
sw t2, 0(t1)          #*b = tmp

ld    ra, 24(sp)
ld    s0, 16(sp)
addi sp ,sp, 32
ret
```

17. 解释以下 RISC-V 汇编代码实现的功能。

```
addi a0,x0,0          #设置循环变量 i = 0
addi a1,x0,1          #a1 = 1
addi a2,x0,30         #设置循环终止条件
loop: beq a0,a2,done  #相等分支指令
slli a1,a1,1          #左移指令，相当于乘 2
addi a0,a0,1          #i + =1
j loop                #循环
done: # exit code
```

该代码实现的功能是实现乘方运算，计算结果为 a1 寄存器中的值等于 2 的 30 次方。