3.7.8, 12, 13, 14, 15, 16, 17

3. 1) nop: 等价于 addi x0, x0, 0, 空操作

2) ret: 等价于 jr ra, 这条指令将控制传递回调用者

3) call offset: 等价于 auipc ra, %pcrel_hi(offset) // 加载地址高位到ra
jalr ra, %pcrel_lo(offset)(ra) // 加载地址低位, 跳转到子程序

4) mv rd, rs: 伪指令mv等价于基本指令addi rd, rs, 0, 这条指令将rs的值加上0, 然后存在rd中。

5) rdcycle rd: 等价于 rdtime rd // 读时钟中断计数器寄存器, 将结果存在rd中
csrrs rd, mcycle, rd // 读取机器周期计数器, 将结果加上rd然后存回rd

6) sext.w rd, rs: 等价于 slli rd, rs, 16 // 将rs左移16位
srai rd, rs, 16 // 将rd算术右移16位, 即进行符号扩展

7 1) add t0, t1, t2
xor t3, t0, t1
xor t4, t0, t2
bne t3, t4, overflow

2) add t0, t1, t2
bltu t0, t1, overflow // 如果 t0 < t1 则溢出

3) 在x86架构中, 加法指令的溢出通过状态寄存器(PSW)中的标志位来检测, 加法指令可以设置标志位中的进位标志(CF)和溢出标志(OF), 如果溢出, 则OF被置为1, 否则为0。类似地, 在ARM, 有C(进位标志)和V(溢出标志)

8. op = DIVU , rd = 0xFFFFFFFF

op = REMU , rd = x

op = DIV , rd = 0x FFFFFFF (如果x最高位为1) 或 0x 00000000 (如果x最高位

op = REM , rd = x

在 RISC-V 中，整型除法除数为 0 不会抛出异常，而是根据指令类型分别设置相应的商和余数。这是为了提高指令的执行效率。

12. 1) Linux Kernel 应运行在最高特权等级的机器模式（Machine Mode），以便访问所有的硬件资源和控制系统的所有权

2) BootROM 应运行在 Machine Mode，以便访问所有的硬件资源，初始化硬件和系统，以及在系统初始化后跳转到操作系统引导程序

3) BootLoader 应运行在特权等级较高的特权级别（Supervisor Mode），以便执行系统初始化和硬件设置，加载操作系统，并将控制权交给操作系统内核

4) USB Driver 应运行在较低特权等级的 User Mode，以便通过操作系统提供的接口访问资源，并且由于 USB Driver 只是一个设备驱动程序，它应由操作系统内核或其他具有足够特权等别的程序来加载和执行

5) vim 应运行在最低特权等级的特权级别（User Mode），因只是一个用户级程序

13
```
#函数开始
.globl vecMul
vecMul:
    addi  sp, sp, -16
    sw    ra, 0(sp)
```

```
    sw   s0, 4(sp)
    li   s0, 0        # i=0
Loop:
    bge  s0, 100, exit   #如果i>=100，则退出循环
    lw   a0, 4(t1)       # a0=B[i]
    lw   a1, 0(t2)       # a1=C
    mul  a0, a0, a1      # a0 = B[i]*C
    sw   a0, 4(t0)       # A[i] = a0

    addi s0, s0, 1       #i++
    addi t0, t0, 4       # A+1, 如 A[0]跳到A[1]
    addi t1, t1, 4       # B+1
    j    Loop
exit:
    lw   a0, 0(t0)   # a0=A[0]
    lw   s0, 4(sp)   #恢复s0
    lw   ra, 0(sp)
    addi sp, sp, 16
    jr   ra
```

14.
```
lw a0, 0(a)
lw a1, 0(b)
lw a2, 0(c)
# if(a>b)
slt t0, a1, a0   # t0 = (a<b)
beq t0, x0, else
add a2, a0, a1   # c = a+b
j end
else:
  sub a2, a0, a1
end:
  sw a2, 0(c)
```

15.
```
addi sp, sp, -4
sw ra, 0(sp)
# p[0] = p
mv t1, t0
sw t1, 0(t0)
#int a=3
li t1, 3
# p[1] = a
addi t0, t0, 4
sw t1, 0(t0)
```

```
# p[a]=a
slli t2,t1,2 # a*4
add t2,t2,t0 # p[a]
sw t1,0(t2)

lw ra,0(sp)
addi sp,sp,4
```

16.
```
swap:
    lw t2,0(t0)    # tmp=*a
    lw t3,0(t1)    # *b
    sw t3,0(t0)    # *a=*b
    sw t2,0(t1)    # *b=tmp
    jr ra
```

17.
```
    addi a0,x0,0    # a0=0
    addi a1,x0,1    # a1=1
    addi a2,x0,30   # a2=30
loop: beq a0,a2,done # if done a0=a2
    slli a1,a1,1    # a1=a1*2
    addi a0,a0,1    # a0+=1
    j loop
done:   # exit code
```

程序循环 loop 30次，期间一次进行 a1=a1*2

相当于 for(i=0;i<30;i++){
    a1*=2;
}