3. (1) nop: addi zero, zero, 0

(2) ret: ~~jalr zero, 0(ra)~~ jalr zero, ra, 0

(3) call offset: jal ra, offset

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

(5) rdcycle rd: rdinstret rd

~~(6) sext.w rd, rs: lw to, 0(rs) lw rd, 0(rs)~~

~~sext.w rd, t0 srai rd, rd, 32~~

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

7. (1) slti t_3, t_1, zero 因为~~数~~有符号数发生溢出时相当于会直接将数值增加或减

slt t_4, t_0, t_2 小一个周期，即比如范围为-8~7时，若add后结束为8，则值实际为-8，从而会使大小关系改变. 若$t_1<0$时，t_3结束果为1，此时若没有产溢出，在正常情况下 $t_0=t_1+t_2<t_2$，故t_4值为1，若溢出了会有$t_0>t_2$，t_4为0，所以溢出时t_3和t_4才会不等并发生跳转

7.(2) 对于无符号数，比如数的范围为0~15，那么16被视为0，即发生溢出时会减去一个最大所表示的数. 由于无符号数均为大于等于0的，所以出现结果比加数小即为溢出

~~add t_0, t_1, t_2~~ 设无符号数范围为0~M，则溢出时结果 $a+b-(M+1)$ $\begin{cases}<a\\<b\end{cases}$

~~bltu t_0, t_1~~

只要进行一次比较：add t_0, t_1, t_2

→ blt t_0, t_1, overflow 是充要条件

7.(3) x86、ARM 采用标志位(Flag)方式检测加法溢出.

x86中，有 carry flag 来检测无符号数相加时是否进位，有 overflow flag 检测有符号数相加时是否溢出；在ARM中，也有上面的两种flag，且作用相同不过多了一个 zero flag，用于检测无符号数的运算结果是否为0

8.(1)

指令	rs1	rs2	Op=DIVU时 rd	Op=REMU rd	Op=DIV rd	Op=REM rd
Op rd,rs1,rs2	x	0	$2^{XLEN}-1$	x	-1 会导致中断	x

整型除法中除数为0会抛出异常。~~这个是标准指令集中唯一一个会发生的算术计算异常~~

不过这种异常被处理起来也并不复杂，因为只需要在每个除操作中多加一条分支跳转指令即可，并且这个分支跳转可以在绝大多数情况下被预测不会发生，所以不怎么增加开销

之所以在无符号除法除数为0时返回 $2^{XLEN}-1$，因为它莫衷是一个全1的值，所以实现这个值可以简化电路，而在有符号数中全1代表的值为-1，所以 DIV 时除数为0结果默认为-1

(2) fcsr 是 RISC-V 中浮点计算的一个控制状态寄存器(CSR)

NV: Invalid Operation　DZ: Divide by Zero　OF: overflow

UF: underflow　NX: inexact

不会陷入系统调用，而是会在软件中显示这些标志，因为为了保证ISA的简洁性

(3) x86中会在除数为0时抛出异常"#DE Divide Error"

ARM会将商和余数寄存器设置为特殊值，并将标志寄存器中的Z标志和C标志设为1

12-17:

12. (1) Linux Kernel: 管理员模式

(2) BootRom: 是处理器的一部分 机器模式

(3) BootLoader: 管理员模式 (4) USB Driver: 管理员模式 (5) vim: 用户模式

17. 在开始时，初始化 $a_0=0$, $a_1=1$, $a_2=30$，然后开始循环，判断 a_0 若等于 a_2 则程序结束，否则 a_1 左移 1 (乘2)，a_0 自加1，最终可从 a_1 中求得 2 的幂次，返回 2^{a_2}，在本题中即计算出 2^{30} 的值

13.
```
addi a0, t0, 0
addi a1, t1, 0
lw   a2, 0(t2)
~~addi a4, zero, 100~~
addi a3, zero, 0
addi a4, zero, 100
loop: beq  a3, a4, end
addi a3, a3, 1
lw   a6, 0(a1)  # B[i]
mul  a5, a6, a2 # B[i]*C
sw   a5, 0(a0)
addi a0, a0, 4
addi a1, a1, 4
j    loop
end: lw  a0, 0(t0)
ret
```

14.
```
blt  a0, a1, else
beq  a0, a1, else
add  a2, a0, a1
j    end
else: sub a2, a0, a1
end: # exit code
```

15.
```
sw   t0, 0(t0)
addi t1, zero, 3
sw   t1, 4(t0)
slli t3, t1, 2
add  t3, t0, t3
sw   t1, 0(t3)
```

16.
```
lw  a0, 0(t0) //tmp
lw  a1, 0(t1) //b
sw  a1, 0(t0)
sw  a0, 0(t1)
ret
```