3. 1) nop → addi x0, x0, 0

2) ret → jalr x0, 0(x1)

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

4) mv rd, rs → addi rd, rs, 0

5) rdcycle rd → csrrs rd, cycle, x0

6) sext.w rd, rs → addiw rd, rs, 0

7. 1) sub t_3, t_0, t_1
mv t_4, t_2

2) add t_0, t_1, t_2
sltu t_3, t_0, t_1
sltu t_4, t_0, t_2
若 t_3 和 t_4 均为1，则加法运算发生溢出

3) x86：通过CPU的状态寄存器，执行加法指令后，CPU会根据运算的结果来设置状态寄存器中的进位标志位和溢出标志位，若发生溢出，会将溢出标志位置1，条件分支指令会检测这些标志的值做出不同的处理。

ARM：对于有符号整数使用 V Flag（溢出标志位），无符号整数使用 C Flag（进位标志位），还可以使用ADC指令，SBC指令或异或运算来判断

8. 1) Op = DIVU 时：产生未定义结果
Op = REMU 时：结果未定义
Op = DIV时："Division-by-zero"异常
Op = REM 时：结果未定义，可能得到0或未定义余数值

2) NV：表示发生无效操作异常（如操作数不是数值）

采取这种的设计保证计算的正确性和稳定性，通过抛出异常指向异常处理程序。

DZ：发生
OF：发
UF：发
NX：发
#lags
常处理
3) x8
应用程
ARM
设置
12.

DZ：发生了非规格化操作数异常

OF：发生了溢出异常

UF：发生了下溢异常

NX：发生了精度损失异常

flags被置位不会导致处理器陷入系统调用，程序可以通过处理器提供的异常处理机制来捕获并处理这个异常。

3) x86：会在执行除以0的指令时抛出此异常，这个异常可以被操作系统或者应用程序捕获并处理。

ARM：会在除以0时返回一个固定的值，如执行SDIV或UDIV指令时，会将结果设置为0xFFFFFFFF

12. 1) Linux kernel 最高级别的机器模式

2) BootROM 最高级别的机器模式

3) BootLoader 特权模式：操作系统内核之前的第一层软件

4) LSB Driver 特权模式

5) vim 用户模式

13.
```
    mv   t3, t0
    lw   a3, 0(t2)
    addi a4, x0, 0
    addi a5, x0, 100
loop: beq  a4, a5, end
    lw   a1, 0(t0)
    lw   a2, 0(t1)
    mul  a1, a2, a3
    sw   a1, 0(t0)
    addi t0, t0, 4
    addi t1, t1, 4
    addi a4, a4, 1
    j    loop
end: lw  a0 t3
```

14.
```
    bge a1, a0, part1
    sub a2, a1, a0
    j end
part 1: add a2, a1, a0
end: # exit code
```

15.
```
    sw t0 0(t0)
    addi a3, a3, 3
    sw a3 4(t0)
    sw a3 12(t2)
```

16.
```
    lw a0, t0
    lw a1, t1
    sw a1, t0
    sw a0, t1
```

17. 计算 2×10^{30} 即 a1 寄存器中的值