3. (1) nop　　addi x0, x0, 0

(2) ret　　jalr x0, 0(x1)

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

(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)　slti t3, t2, 0

slt t4, t0, t1

(2) add t3, t1, t2

slt t4, t3, t1　　# 如果t3<t1，说明发生溢出，t4设置为1

bne t4, x0, overflow

(3) x86 引入CF标志(对应ARM的C标志)来指示是否有为无符号整数的溢出

引入OF标志(对应ARM的V标志)来指示是否为有符号整数的溢出，

判定溢出后，相应标志被置为1

8. (1)

指令	rs1	rs2	op=divu	op=remu	op=div	op=rem
op rd,rs1,rs2	x	0	0xffffffffffffffff	x	-1	x

不会出现异常，所有位的值都设置为1，表示最大的无符号数，以简化除法电路

商的

(2) NV：不支持的操作　DZ：除数为0　OF：溢出　UF：下溢(负溢出)　NX：inexact：结果不精确

不一定会使处理器陷入系统调用.

(3) x86和ARM架构中，除数为0会产生"除以零"(#DE)异常.

12. (1) Linux kernel 特权等级 3 (M)

(2) BootROM 特权等级 3 (M)

(3) BootLoader 特权等级 1 (S)

(4) USB Driver 特权等级 1 (S)

(5) vim 特权等级 0 (U)

13.

```
    add  t3, x0, x0    # i=0
    addi t4, x0, 100   # t4=100
Loop: bge  t3, t4, exit
    sll  t5, t3, 2     # i*4
    addi t6, t5, 0
    add  t6, t6, t0    # &of A+i
    add  t5, t5, t0    # &of B+i
    ~~lw  t6, 0(t6)    # *(A+i)~~
    lw   t5, 0(t5)     # *(B+i)
    lw   t7, 0(t2)     # *C
    mul  t8, t5, t7    # A[i]= B[i] *C
    sw   t8, 0(t6)     # &of A+i
    addi t3, t3, 1     # i++
    j    Loop
exit: lw   t9, 0(t0)   # return A[0]
    jr ra
```

14. bgt a0, a1, greater_than # if a0 > a1, then branch.

    sub a2, a0, a1 # c = a - b

    j end

greater_than:

    add a2, a0, a1 # c = a + b

end: ...

15. sw t0, 0(t0) # P[0] = P

    addi t1, x0, 3 # int a = 3

    sw t1, 4(t0) # P[1] = a

    sw t1, 12(t0) # P[3] = a.

16. lw t2, 0(t0) # *a

    lw t3, 0(t1) # *b

    ~~mv t4, t2 # tmp = *a~~

    ~~mv~~

    sw t3, 0(t0) # *a = t3

    sw t2, 0(t1) # *b = t2

17. 功能：将 a0 累加到 30，同时 a1 左移 30 位，目标可能是计算 2^{30} 的值，可以用于计算指数函数等指数部分.