3. (1) nop addi x0, x0, 0 空指令 (2) ret jalr x0, x1, 0 子程序返回指令

(3) call offset auipc x6, offset[31:12] 跳转4KB-4GB空间函数

(4) mv rd rs addi rd, rs, 0 数据传送指令 (5) rdcycle csrrs rd, cycle x0 周期数读取指令

(6) sext.w rd rs addiw rd, rs, 0 符号位扩展指令

7. 1) add t0 t1 t2

slti t3 t2 0

slt t4 t0 t1

bne t3 t4 overflow

(2) add t2 t0 t1

bltu t2 t0 overflow

(3). X86: X86的EFLAGS寄存器有状态标志

ARM 通过CPSR状态寄存器反映溢出状态

MIPS: 指令触发中断产生溢出信号

8. (1)

指令	rs1	rs2	DIVU	REMU	DIV	REM
op rd, rs1, rs2	x	0	0xffff	x	0xffff...	x

RISC-V 当除数为0时 ~~[illegible]~~ 特殊情况的特殊处理有利于保证程序顺利进行, 同时处理会产生执行异常 bug

(2) NV Invalid operation

DZ Divide by ZERO

OF Overflow

UF Underflow

NX Inexact

fflags 设置位允许系统程序自检从而对符号等异常进行检查

(3) X86 使用CPU跳转指令 发现除0就跳过DIV

ARM 处理异常情况, 进入异常模式

12. 1)

LiNUX Kernel	Supervisor
BootROM	Machine
BootLoader	Machine
USB Driver	Supervisor
Vim	User

13.

```
vecMul:
        add   x10  x0   x0  # i=0
        addi  x11  x0   20  # x11=20
Loop:   bge   x10  x11  exit  # i>=20, exit
        sll   x12  x10  2   # i*4
        add   x13  x12  t0  # &(A[i])
        lw    x15  0(x13)   # x15 = A[i]
        add   x14  x12  t1  # &B[i]
        lw    x16  0(x14)   # x16 = B[i]
        lw    x17  0(t2)    # x17 = C
        mul   x15  x16  x17 # A[i]=B[i]*C
        sw    x15  0(x13)
        sw    x16  0(x14)
exit:   lw    a0   0(t0)
        ret
```

14. blez a0 a1 lf # a0≤a1 ⇒ a≤b, lf.

add a2 a0 a1 # c=a+b

lf: sub a2 a0 a1 # c=a-b

15. ~~lw x10 0(t0) # x10~~ = p

~~add x10 x10 t0~~

~~sw x10 0(t0)~~

sw t0 0(t0). # p[0]=p

addi t1 t1 3 # a=3

addi x10 t0 4 #&p[1]

lw x11 0(x10) # x11=p[1]

add x11 x0 t1 # p[1]=a

sw x11 0(x10)

sll x10 t1 4 # a*4

add x10 x10 t0 # &p[a]

lw x11 0(x10) # x11=p[a]

add x11 x0 t1 # p[a]=a

sw x11 0(x10)

16. lw x10 0(t0)

lw x11 0(t1)

sw x11 0(t0)

sw x10 0(t1)

nop.

17.

```
int a0 = 0
int a1 = 1
int a2 = 3.0

while ( a0 != a2)
{
    a1 = a1 * 2
    a0 = a0 + 1
```

计算 2^{a_2}