3. (1) nop: addi x0, x0, 0

(2) ret: jalr x0, x1, 0

(3) call offset: auipc x6, offset[31:12]

jalr x1, x6, offset[11:0]

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

(5) rdcycle rd: csrrs rd, cycle[h], x0

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

7. (1) sub t3, t0, t1

mv t4, t2

(2) add t0, t1, t2

~~bne~~ bge t1, t0, overflow

bge t2, t0, overflow

(3) x86架构：使用 JO 指令，检测标志寄存器中的 OF 标志位是否为1，1则跳转到指定目标地址

ARM架构：使用 VADD 和 VADDW 来对进行带进位加法运算；或使用 VC 和 VCC 来检测加法溢出（检查溢出标志 V）

8. (1) Op = DIVU: rd = 0xfff.....f ($2^{XLEN}-1$)

Op = REMU: rd值为 rs1的值 x

Op = DIV: rd = 0xff.....f ($2^{XLEN}-1$)

Op = REM: rd值为 rs1的值 x

不会抛出异常，若引发异常，这些异常会在大多数

执行环境中造成陷阱，需要语言实现者与执行环境和陷阱处理程序进行交互。

13.
```
add   a4, zero, zero   # i=0
addi  a5, zero, 100    # a5=100
loop:
bge   a4, a5, exit
slli  a6, a4, 2        # i*4
add   t3, t0, a6       # & of A+i
add   t4, t1, a6       # & of B+i
lw    t4, 0(t4)        # *(B+i)
mul   t4, t4, t2       # B[i]*C
sw    t4, 0(t3)        # *(A+i)=B[i]*C
addi  a4, a4, 1        # i++
j     loop
exit:
lw    t0, 0(t0)        # *A
```

15.
```
sw    t0, 0(t0)        # p[0]=p
addi  t1, zero, 3      # a=3
sw    t1, 4(t0)
slli  t3, t1, 2        # a*4
add   t3, t0, t3       # & of (p+a)
sw    t1, 0(t3)        # p[a]=a
```

16.
```
lw   t3, 0(t0)    # *a
lw   t4, 0(t1)    # *b
sw   t4, 0(t0)
sw   t3, 0(t1)
```

17. 将数值"1"的补二进制码左移30次，在32位mips上的寄存器条件下，得到 2^{30}，存在a1中。否则得0。