

7. (1) add t0, t1, t2

~~xor t3, t0, t1
xor t4, t1, t2
add t0, t4, x0, 0~~

bne t3, t4, overflow

add t0, t1, t2

~~xor t3, t1, t2
xor t4, t0, t1
and t3, t3, t4
mv t4, x0~~

bne t3, t4, overflow

(2) add t0, t1, t2

sltu t3, t0, t1 #如果 t0 > t1, 则 t3 为 0 否则为 1.

bne t3, 2³², overflow

~~#加法溢出，t0 的值应该等于 t1 - t3 且 t3 为 1~~

(3) 在 X86 架构中，加法指令的溢出可以通过检查进位标志 (carry Flag) 和溢出标志 (overflow Flag) 实现。进位标志位用于检测无符号数加法的进位情况，溢出标志位用于检测有符号加法的溢出情况。

在 ARM 架构中，加法指令的溢出类似 X86 架构，此外，ARM 架构还提供了一个专门用于检测加法溢出的指令：sadds (signed Add with Saturation)，它将饱和加法和溢出检测结合在一起，可以更方便地实现加法溢出的检测。

UE 下溢						
18. (1) 指令	rs1	rs2	OP = DIVU 时 rd 为 0 $2^{N-LEN} - 1$	OP = REMU 时 rd 为 0 2^{N-LEN}	OP = DIVS 时 rd 为 0 -1	OP = REMS 时 rd 为 0 x
Op rd, rs1, rs2	x	0				

整型除法中除数为 0 会引起 RISC-V 抛出异常。这是因为在处理器硬件中，除法的实现需要进行除法的检查和调整，以保证结果的正确性。

这种设计的主要原因是保证处理器运算的正确性和可靠性，通过在硬件层面引入异常处理机制，可以及时捕获和处理异常情况。此外，还可以提高程序安全性和可靠性。所以，可能被恶意攻击者利用进行攻击和入侵，引入抛出异常可以有效防止这种攻击。

(2) NV (Invalid Operation)：表示发生了无效操作，如 $0/0$ 或 $\sqrt{-1}$ 等。

DZ (Divide by zero)：表示发生了除以 0 的操作。

OF (Overflow)：表示发生了数值溢出，结果超出了可表示的最大值。

UF (Underflow)：表示发生了数值下溢，结果超过了可表示的最小值。

NX (Inexact)：表示发生了不精确的结果，如舍入误差等。

当浮点异常发生时，对应的 flags 位会被置位，以便程序检测到异常并采取相应措施，如返回错误码或进行异常处理等。

Flags 被置位不会导致处理器陷入系统调用，但会触发浮点异常处理机制，根据设置的处理模式进行相应的处理。处理模式可以是默认的中断模式，也可是灵活的非中断模式。在非中断模式下，处理器会根据设置的控制位和状态位进行相应操作，如清除、封锁或进行异常处理而立即停止程序并进入系统调用。

(3) X86 架构：在 X86 架构中，当除数为 0 时，触发“除法错误 (division error)”。异常此时，处理器会将异常码写入 EFLAGS 寄存器，并将控制转移到相应的异常处理程序。异常处理程序通常会终止当前的进程或线程。

ARM 架构：触发“除以 0 (division by zero)”异常。处理器会控制转移到异常向量表中对应的异常处理程序。异常处理程序会终止当前进程。

12. (1) Linux kernel：处于机器模式 (Machine Mode) 允许直接访问硬件资源和执行敏感指令。

2) BootROM：机器模式，它是系统启动的第一个程序，需要访问硬件资源来初始化系统。

3) BootLoader：管理员模式 (Supervisor Mode)，它需要执行特权指令来访问一些硬件资源和内存地址空间，但在某些情况下可能需要进入机器模式。

4) USB Driver：管理员模式

5) vim：用户模式，它是用户空间的应用程序，只能访问受限的资源和内存地址空间。

13. VecMul :

```
addi    sp, sp, -16    # 预配栈帧  
sd      ra, 4(sp)    # 保存返回地址  
sd      s0, 2(sp)    # 保存旧 s0  
addi    s0, sp, 16  
li      s4, 0          # 计数器置零  
li      s5, 100         # t2, t0 为 A[0] 地址  
mv      t2, t0  
loop:  beq    s4, s5, end # i 达到 100 跳转到 end.  
lw      s2, 0(t1)    # 加载 B[i]  
lw      s3, 0(t2)    # 加载 C  
addi    s4, s4, 1  
addi    s5, s5, 4  
addi    s1, t1, 2  
j       loop.  
end:  lw      a0, 0(t0)    # 加载 A[0]  
ld      ra, 4(sp)    # 恢复 RA 值  
ld      s0, 2(sp)    # 恢复 S0 值  
addi    sp, sp, 16  
jr      ra            # 返回
```

15.

```
mul    s3, s2, s3    # 乘法计算  
sw      s3, 0(t0)    # 存储到 A[i]  
addi   s4, s4, 1  
addi   s5, s5, 4  
addi   s1, t1, 2  
j       loop.  
end:  lw      a0, 0(t0)  
ld      ra, 4(sp)  
ld      s0, 2(sp)  
addi    sp, sp, 16  
jr      ra
```

16.

```
14. bge  a0, a1, if    # 比较 b 大于等于 a  
sub    a2, a0, a1    # a < b 进行 a - b  
l:  beq  a0, a0, a1    # 判断 a = b  
add    a2, a0, a1    # a > b 进行 c = a + b  
j     end  
2:  addsub a2, a0, a1    # a = b 时 c = a - b  
end: jr    ra
```

15.
 sw t0, 0(t0) # $p[0] = p$
 li t1, 3 # int a = 3
 sw t1, 4(t0) # $p[1] = a$
 sw t2, t1, 2 # 将 t1 左移 2 位得到偏移量.
 add t2, t0, t2 # 得到 $p[c]$ 地址.
 sw t1, 0(t2) # $p[a] = a$.

16.
 lw t2, 0(t0) # $tmp = *a$
 lw t3, 0(t1) # $t3 = *b$
 sw t3, 0(t0) # $*a = *b$
 sw t2, 0(t1) # $*b = tmp$
 jr ra # return.

17.
 loop: addi a0, x0, 0 # $a0 = 0$
 addi a1, x0, 1 # $a1 = 1$
 addi a2, x0, 30 # $a2 = 30$.
 loop: beq a0, a2, done # $a0 = a2$ 跳出循环
 slli a1, a1, 1 # $a1 \leftarrow a2$
 addi a0, a0, 1 # $a0 = a0 + 1$
 j loop. # loop.

done: # exit code

计算 $a++$: 代码将 $a1$ 乘以 10 直到 $a0$ 达到 30 退出