

### 3. 伪指令 基础指令

nop addi x0, x0, 0

ret jalr x0, 0(ra)

call offset { auipc t0, offset[31:12]  
jalr t1, t0, offset[11:0]

mv rd, rs addi rd, rs, 0

recycle rd { rdtime rd  
csrr rd, cycle

sext.w rd, rs addiw rd, rs, 0

7. (1) add t0, t1, t2

stl srl t3, t0, 31

stl srl t4, t1, 31

bne t3, t4, overflow

(2) add t0, t1, t2 add t0, t1, t2

stl

bgt

add t0, t1, t2

stl t0, t0, t1

bnez t0, overflow

(3) x86 架构中设置 EFLAGS 寄存器的 OF(进位) 和 OF(溢出) 位来检测

当有符号产生进位或相加时，对应的标志位会设为 1

ARM 架构中设置 APSR 寄存器的 C(进位) 和 V(溢出) 位来检测加法溢出

8.  $op = \text{divu}$   $op = \text{remu}$   $op = \text{div}$   $op = \text{rem}$   
(1)  $op$   $rd, rs1, rs2$   $0xffffffffffffffffffff$   $x$   $0xffffffffffffffffffff$   $x$

会引起 RISC-V 抛出异常。

这样设计便易于检测异常，且不用区别有符号整型与无符号整型。

(2) ~~0~~: 除零标志位，当一个浮点数被零除时，该标志位被置位。

1: 上溢标志位 当浮点运算结果太大，不能用浮点数表示时，该标志位被置位。

2: 下溢标志位，当浮点运算结果太小，不能用浮点数表示时，该标志位被置位。

3, 4, 5: 保留位，未使用。

如果 fflags 被置位，不会导致处理陷入系统调用，它们可以通知软件发生了浮点数运算异常。

(3) 在 x86 中，当执行整数除法指令时，如果除数为 0，则会引发“除零异常”，该异常可由异常处理程序捕获和处理。

在 ARM 中，如果除数为 0，则会引发一个称为“被 0 除错误”的异常，该异常可由异常处理程序捕获和处理。ARM 并不提供一个单独的异常处理状态寄存器，而是将异常状态信息保存在相关程序状态寄存器中。

3. ~~result~~:

start: ~~t1 t3, 0~~

3.13. vecMul:

start: li t3, 0  
li t4, 100  
loop: beg t3, t4, end  
mv ~~t3~~ t5, ~~t4~~ t1  
mv ~~t4~~ t6, ~~t3~~ ~~t2~~  
mul t5, t5, t6  
sw t5, 0(t0)  
addi t0, t0, 4  
addi t1, t1, 4  
addi t3, t3, 1  
j loop  
end: lw 0(t0)  
ret

4. part1: bge a0, a1, ~~else~~ part2

negw a1, a1  
add a2, a0, a1  
j end

part2: add a2, a0, a1

end: ret

15. mv t2, t0  
sw t2, 0(t0) # p[0]=p  
li t3, 3 # a=3  
sw t3, 4(t0) # p[1]=a  
sw t3, 12(t0) # p[a]=a

16. swap:

mv t2, t1  
mv t1, t0  
mv t0, t2

17. 该代码实现的功能是计算  $2^{30}$  的值。

首先令  $a_0=0, a_1=1, a_2=31$

再使用循环。当  $a_0 \neq a_2$  时， $a_1$  左移一位（相当于乘2）， $a_0$  自增1

这样当  $a=31$  时，跳出循环，此时  $a_1$  左移30位，相当于  $1 \times 2^{30}$

最后，循环结束，程序返回