3.21 第五周

3. 伪指令 | 基本指令或指令组合

(1) nop | addi x0, x0, 0

(2) ret | jalr x0, x1, 0

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

(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) srai ~~t1~~ t3, t0, 32
srai t4, t1, 32

(2) add t0, t1, t2
blt t0, t1, overflow
~~blt t0, t2, overflow~~

(3) ~~无符号~~: 在x86和ARM等指令集架构中,通常用标志位(flag)来检测加法溢出。

x86架构: 使用~~over~~ Overflow Flag (OF)标志位来表示加法操作是否发生溢出。当有符号数相加时,若结果的符号位改变,则表示发生溢出,此时OF标志位被置1;否则OF被清零。

ARM架构: 使用Carry Flag (CF)标志位来表示加法操作发生是否溢出。当无符号整数相加时,若最高位产生了进位,表示发生溢出,CF标被置1;否则CF被清零。

8. ~~DIVU时：抛出硬件异常~~ REMU时：抛出硬件异常

~~DIV时：产生未定义行为~~ REM时：产生未定义行为

8.4) DIVU和REMU：不会抛出异常，把寄存器设为全1

DIV和REM：抛出异常

原因：DIVU和REMU通常用于对无符号数进行除法和取模。设计为除数为0时输出最大无符号整数可以确保结果在合法的无符号整数范围内，且更容易地检测和处理这些情况，能提高运算效率（避免切换到异常处理程序）。

而DIV和REM用于对有符号数进行除法和取模，~~若设计除数为~~范围为2^{N-1}~2^{N-1}。若不设置抛出异常会导致后续计算过程的变号过程出现范围异出的错误而难以找到错误原因。

Q) NV：当执行无效的浮点操作时，该标志位将被置位。

DZ：当执行一个浮点除法时，除数为0时，该标志位将被置位。

OF：当执行浮点运算时结果超出了浮点数能够表示的范围时，该标志位被置位。

UF：当执行浮点运算时结果的精度超出了浮点数能够表示的范围时，该标志位将被置位。

NX：当浮点运算的结果无法完全精确地表示为浮点数时，将被置位。

fflags被置位不会使处理器陷入系统调用，但程序可以使用浮点异常处理机制来处理这些异常。

(3) x86架构：触发一个#DE异常，除法操作的结果无法被保存，并将控制权转移到异常处理程序。

ARM架构：触发一个UDIV或SDIV指令的异常，处理器将控制权传递给异常处理程序，程序可以捕获该异常并进行适当的处理。

12. (1) Linux Kernel 机器模式

(2) BootROM 机器模式

(3) Boot Loader 管理员模式

(4) USB Driver 用户模式

(5) vim 用户模式

13. vecMul:

```
    mv   t3, zero
    lw   t5, 0(t2)
loop:
    bge  t3, t00, exit
    lw   t6, 0(t1)
    mul  t6, t6, t5
    sw   t6, 0(t0)
    addi t3, t3, 1
    addi t0, t0, 4
    addi t1, t1, 4
    j    loop
exit:
    lw   a0, 0(t0)
    ret
```

14.

```
    blt  a1, a0, part1
    j    part2
part1:
    add  a2, a1, a0
part2:
    sub  a2, a0, a1
```

15.
```
sw    t0, 0(t0)
addi  t1, zero, 3
addi  t3, t0, 4
sw    t1, 0(t3)
addi  t3, t0,
mul   t4, t1, 4
sw    t1, 0(t4)
```

16.
```
swap:
addi  sp, sp, -32
sd    ra, 24(sp)
sd    s0, 16(sp)
addi  s0, sp, 32
mv    t2, t0
mv    t0, t1
mv    t1, t2
ld    ra, 24(sp)
ld    s0, 16(sp)
addi  sp, sp, 32
ret
```

17. 计算 2^{30}。