

3. RISC-V 汇编中存在许多伪指令，它们一般是具有特殊操作数的基本指令或指令组合。
请写出与以下伪指令等价的基本指令或指令组合。

- 1) nop
- 2) ret
- 3) call offset
- 4) mv rd,rs
- 5) rdcycle rd
- 6) sext.w rd,rs

(1) `addi x0, x0, 0`

(2) `jalc x0, x1, 0`

(3) `auipc x1, offset[31:12]`
`jalc x1, x6, offset[11:0]`

(4) `addi rd, rs, 0`

(5) `csrwr rd, cycle, x0`

(6) `addiw rd, rs, 0`

7. RISC-V 标准指令集并未为加法指令的溢出引入专用的标志位，因此通常需要额外的指令以检查加法溢出。

- 1) 考虑如下的指令序列：

`add t0,t1,t2`

`bne t3,t4,overflow`

若 t0 和 t1 都是有符号数，请在横线处填入正确的指令，使得当 t0 和 t1 的加法发生溢出时，控制流可以正确跳转到 overflow 位置。（请勿使用除 t0~t4 以外的任何寄存器）

- 2) 当 t0 和 t1 都是无符号数时，请给出尽量简单的检测 add t0,t1,t2 指令加法是否溢出的指令序列。

- 3) 调研其他指令集架构（如 x86、ARM 等）是如何检测加法溢出的。

(1) `mv t3, t0`

`sext.w t4, t0`

(2) `srli t3, t1, 31`

`srli t4, t2, 31`

`srli t5, t0, 31`

`and t3, t3, t4`

`not t5, t5`

`and t3, t3, t5`

`bnez t3, overflow`

3) 调研其他指令集架构 (如 x86、ARM 等) 是如何检测加法溢出的。

x86 架构使用标志位来检测加法溢出。具体来说, 当两个有符号数相加时, 若结果超出了有符号整数的范围, 标志位 OF 会被设置为 1, 表示发生了溢出。

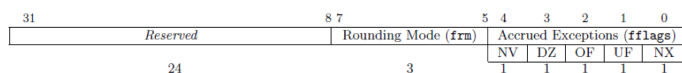
ARM 架构也使用标志位来检测加法溢出, 使用的是 CPSR 寄存器, 其包含了一些标志位, 包括 N (结果为负)、Z (结果为零)、C (进位标志位) 和 V (溢出标志位)。当进行有符号整数加法时, 若结果超出了有符号整数的范围, 标志位 V 会被设置为 1, 表示发生了溢出。同时, 若进位从最高位产生, 标志位 C 也会被设置为 1。

8. 阅读 RISC-V 规范以了解 RISC-V 对除数为 0 的除法指令的处理方法, 回答以下问题。

1) 对整型除法, 填写下表。整型除法中除数为 0 是否会引起 RISC-V 抛出异常? 试分析为什么采取这样的设计。

指令	rs1	rs2	Op=DIVU 时 rd 值	Op=REMU 时 rd 值	Op=DIV 时 rd 值	Op=REM 时 rd 值
Op rd,rs1,rs2	x	0	2^{LEN-1}	\propto	-1	\propto

2) 对浮点除法, 除数为 0 将会引起 fcsr 控制寄存器中的相关标志位被置位。下图给出了 fcsr 的构成, 请说明 fflags 的各位分别代表什么含义。fflags 被置位是否会使得处理器陷入系统调用?



3) 调研其他指令集架构 (如 x86、ARM 等) 是如何处理除数为 0 的。

(1) 不会触发异常。若触发异常, 则该异常在绝大多数执行环境里会导致一个自陷。然而, 这将成为目标 ISA 中唯一的算术自陷, 并在此情形下, 需要语言实现者与执行环境的自陷处理函数交互。

(2) NV: 非法操作 OF: 上溢 NX: 不精确

DZ: 除以 0 UF: 下溢

不会

(3) 当除数为 0 时, x86 和 ARM 会触发“除以零”异常, 且都支持处理浮点异常的机制。

12. 写出以下程序在 RISC-V 中应当处于的特权等级。

- 1) Linux Kernel 机器模式
- 2) BootROM 机器模式
- 3) BootLoader 管理员模式
- 4) USB Driver 管理员模式
- 5) vim 用户模式

13. 写出实现以下 C 程序的 32 位 RISC-V 汇编代码。假设 A 和 B 的起始地址存放于寄存器 t0 和 t1, C 的地址存放于寄存器 t2。

```
int vecMul(int *A, int *B, C){  
    for(int i = 0; i < 100; ++i){  
        A[i] = B[i] * C;  
    }  
    return A[0];  
}
```

add a0, x0, x0 # a0 = i

addi a1, x0, 100 # a1 = 100

loop:

bge a0, a1, exit

slli a2, a0, 2

add a3, t0, a2 # &A[i]

add a4, t1, a2 # &B[i]

lw a2, 0(a3) # *(B+i)

lw a4, 0(t2) # C

mul a2, a2, a4

sw a2, 0(a3)

addi a0, a0, 1

j loop

exit:

lw a0, 0(t0)

14. 写出实现以下 C 程序的 32 位 RISC-V 汇编代码。假设 a、b 和 c 分别对应寄存器 a0、a1 和 a2。

```
int a, b, c;  
if(a > b){  
    c = a + b;  
}  
else{  
    c = a - b;  
}
```

bge a0, a1, if

add a2, a0, a1

j 2f

l:

sub a₂, a₀, a₁

2:

Dumping point.

15. 写出实现以下 C 程序的 32 位 RISC-V 汇编代码。假设指针 p 已经通过程序 `int *p=(int *) malloc(4*sizeof(int))` 得到, 且 p 存放于 t0 中, a 存放于 t1 中。

```
p[0] = p;  
int a = 3;  
p[1] = a;  
p[a] = a;
```

sw t0, 0(t0)

li t1, 3

addi a0, x0, 1

slli a0, a0, 4

sw t0, 0(a0)

add a0, x0, t1

slli a0, a0, 4

sw t0, 0(a0)

16. 写出实现以下 C 程序的 32 位 RISC-V 汇编代码。假设指针 a 和 b 分别存放于 t0 和 t1 中。

```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
    return;  
}
```

swap:

addi a0, t0, 0

mv t0, t1

mv t1, a0

ret

17. 解释以下 RISC-V 汇编代码实现的功能。

```
addi a0,x0,0
addi a1,x0,1
addi a2,x0,30
loop: beq a0,a2,done
      slli a1,a1,1
      addi a0,a0,1
      j loop
done:  # exit code
```

$a_0 = 0$ i
 $a_1 = 1$
 $a_2 = 30$
for ($i=0; i<30; i++$)
 $a_1 = a_1 \cdot 2$

功能: 求 2 的 30 次方.