

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: `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: `csrr t0, cycle`; // 读取 cycle 计数器的值并将其存储在寄存器 t0 中

`add rd, x0, t0`; // 将寄存器 t0 中的值添加到寄存器 x0 (0) 中，并将结果存储在目标寄存器 rd 中

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

7. ISC-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`

`slt t3,t0,t1`

`slti t4,t2,0`

`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。

`blt t0,t1,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

在大部分指令集架构中，检测加法溢出通常使用条件码寄存器（Condition Code Register）中的溢出标志位（Overflow Flag）来实现。

在执行加法指令时，如果结果溢出，溢出标志位会被设置为1，否则为0。程序可以通过读取条件码寄存器来检测加法指令是否发生了溢出。

以x86指令集为例，执行加法指令 `add` 会将结果存储在目标操作数中，同时更新条件码寄存器的各个标志位。其中，溢出标志位OF（Overflow Flag）用于检测加法是否发生了溢出。

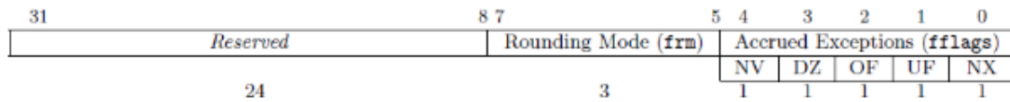
例如，执行指令 `addl %eax, %ebx` 将%eax寄存器的值加到%ebx寄存器的值上，并将结果存储到%ebx寄存器中。如果加法操作导致溢出，OF标志位会被设置为1，否则为0。程序可以通过检查OF标志位来判断加法是否发生了溢出。其他指令集架构也有类似的实现方式。

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) 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)

- op = DIVU时,  $rd = 2^L - 1$
- op = REMU时,  $rd = x$
- op = DIV时,  $rd = -1$
- op = REM时,  $rd = x$

原因：当语言标准要求除以零异常必须导致立即控制流改变时，每个除法操作只需要添加一个分支指令，该分支指令在除法之后插入，通常不被预测为执行，这会增加运行时的开销。

2)

Flag Mnemonic	Flag Meaning
NV	Invalid Operation
DZ	Divide by Zero
OF	Overflow
UF	Underflow
NX	Inexact

不会处理器陷入系统调用

3) 在x86架构中，当执行DIV指令时，如果除数为0，就会触发一个#DE (Divide Error) 异常，程序会跳转到相应的异常处理程序。在ARM架构中，当执行SDIV或UDIV指令时，如果除数为0，就会触发一个#DE (Division by zero) 异常。一些指令集架构，如MIPS和RISC-V，不会触发异常，而是返回一个特殊的结果值来表示除数为0的情况。

12. 写出以下程序在RISC-V中应当处于的特权等级。

- 1) Linux Kernel: 特权等级3或更高的S模式
- 2) BootROM: 特权等级M模式
- 3) BootLoader: 特权等级M模式或S模式
- 4) USB Driver: 特权等级S模式或更高的特权等级
- 5) vim: 用户态，无需特权等级

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

```

```

.text
.globl vecMul
.type vecMul,@function
vecMul:
    addi    sp,sp,-64
    sd      ra,56(sp)
    sd      s0,48(sp)
    addi    s0,sp,64

    #传参
    #64位系统中，一个指针占8个字节
    sd      t0,-56(s0) # int *A
    sd      t1,-48(s0) # int *B
    sw      t2,-40(s0) # int C

.for1:
    sw      zero,-36(s0)
    addi    a3,x0,100 #100
.for2:
    lw      a7,-36(s0)
    bgt     a7,a3,.end
.for:
    #取i
    lw      a7,-36(s0)
    slli    a7,a7,2
    #取A
    ld      a4,-56(s0)
    #A[i]的地址
    add     a4,a4,a7
    #取B
    ld      a5,-48(s0)
    #B[i]的地址
    add     a5,a5,a7

    #取B[i]
    ld      a5,0(a5)
    #取C
    lw      a6,-40(s0)
    mul     a6,a6,a5
    sw      a6,0(a4)

.for3:
    #取i
    lw      a7,-36(s0)
    addi    a7,a7,1

```

```

sw      a7,-36(s0)
j       .for2

.end:
#取A
ld      a4,-56(s0)
lw      a4,0(a4)
mv      a0,a4

ld      ra,56(sp)
ld      s0,48(sp)
addi    sp,sp,64
ret

```

14. 32位riscv, a,b,c在a0,a1,a2

```

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

```

```

main:
    addi    sp,sp,-32
    sd      ra,24(sp)
    sd      s0,16(sp)
    addi    s0,sp,32

    sw      a0,-28(s0)
    sw      a1,-24(s0)
    sw      a2,-20(s0)

.if:
    lw      a3,-28(s0)
    lw      a4,-24(s0)
    ble     a3,a4,.else
    add     a5,a3,a4
    j       .outif
.else:
    sub     a5,a3,a4
.outif:
    sw      a5,-20(s0)

```

15. 32位, p在t0, a在t1

```

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

```

```

main:
    addi    sp,sp,-32
    sd      ra,24(sp)

```

```

sd      s0,16(sp)
addi    s0,sp,32

sw      t0,-28(s0)
sw      t1,-24(s0)

# p[0] = p
ld      a5,-28(s0)
sext.w  a4,a5
ld      a5,-28(s0)
sw      a4,0(a5)

#int a = 3
li      a5,3
sw      a5,-24(s0)

#p[1] = a
ld      a5,-28(s0)
addi    a5,a5,4
lw      a4,-24(s0)
sw      a4,0(a5)

#p[a] = a
ld      a5,-28(s0)
lw      a4,-24(s0)
slli    a3,a4,2
add     a5,a5,a3
sw      a4,0(a5)

```

16.32位, a在t0, b在t1

```

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

```

```

.text
.globl  swap
.type  swap,@function
swap:
    addi    sp,sp,-48
    sd      s0,40(sp)
    addi    s0,sp,48
    sd      t0,-40(s0)
    sd      t1,-48(s0)
# int tmp = *a
    ld      a5,-40(s0)
    lw      a5,0(a5)

```

```

        sw        a5,-20(s0)
# *a = *b
        ld        a5,-48(s0)
        lw        a4,0(a5)
        ld        a5,-40(s0)
        sw        a4,0(a5)
# *b = tmp
        ld        a5,-48(s0)
        lw        a4,-20(s0)
        sw        a4,0(a5)

.end:
        ld        s0,40(sp)
        addi      sp,sp,48
        ret

```

17.

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

```

计算2的30次方