3、RISC-V汇编中有许多伪指令，它们一般定义具有特殊操作数的基本指令或指令组合。请写出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 ⇔ csrrs rd, cycle, x0

6) sext.w rd, rs ⇔ 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t1和t2都是有符号数，请在横线中填入正确的指令，使得当t1和t2加法溢出时，控制流可以正确跳转到overflow位置（勿使用除t0～t4外的任何寄存器）

解：如果有符号数发生溢出，符号位将会被改变，只有两个正数或两个负数相加才可能溢出。

对 $-2^{w-1} \leq x, y \leq 2^{w-1}-1$ 的有符号整数 x 和 y：

$$x+y=\begin{cases} x+y-2^w, & x+y \geq 2^{w-1}, \text{正溢出} \\ x+y, & -2^{w-1} \leq x+y < 2^{w-1}, \text{正常} \\ x+y+2^w, & x+y < -2^{w-1}, \text{负溢出} \end{cases}$$

因此可以考虑通过算术右移获取求和结果和一加数的符号位，判断溢出，方法如下：(假设为RV32I)

```
add t0,t1,t2
sra t3,t0,31  #或63，如果为64位，通用表示为XLEN-1
sra t4,t1,31  #同上 (XLEN-1)
bne t3,t4,overflow
```

2) 当 t1 和 t2 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：

无符号数加法：

对满足 $0 \leq x, y < 2^w$ 的 x 和 y 有

$$x+y=\begin{cases} x+y, & x+y < 2^w, \text{正常} \\ x+y-2^w, & 2^w \leq x+y < 2^{w+1}, \text{溢出} \end{cases}$$

因此，如果发生溢出，t0 中存放的值将比 t1 或 t2 小

可用如下方法判断溢出：

```
bltu t0,t1,overflow
bltu t0,t2,overflow
```

3) 调研其他指令集架构(如x86，ARM)是如何检测溢出的。

调研对象：x86

X86架构的CPU有一组单个位条件码(condition code, cc)寄存器，用来描述最近的算术或逻辑操作指令。可以检测这些寄存器来判判是否溢出，例如：

CF：进位标志，最近的操作使最高位产生了进位，可用来检测无符号操作的溢出。

OF：溢出标志，最近的操作导致了一个补码溢出。可用于检测有符号加法的正溢出或负溢出。

可以通过条件码的设置来检测 x86 的加法溢出。

3. 阅读 RISC-V 规范了解 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 是否为引起 RISC-V 抛出异常？试分析为什么采取这样的设计。

Op rd, rs1, rs2　rs1：x　rs2：0

Op = DIVU 时，rd 的值：$2^{XLEN}-1$，其中 XLEN 为处理器位宽，下同。

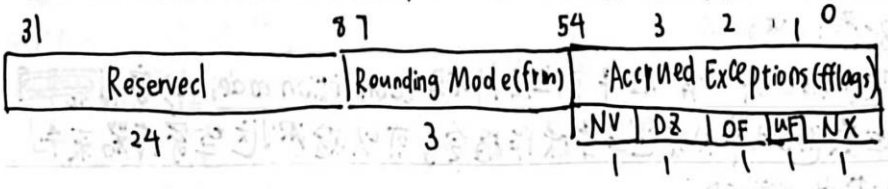
Op = REMU 时，rd：x

Op = DIV 时，rd：-1

Op = REM 时，rd：x

当除数为 0 时，RISC-V 不会对此情况引发异常。这是因为这种方法可以简化处理器设计的实现，对除数为 0 的情况，结果一定是所有位都被设置为 1（即 unsigned int 的 $2^{XLEN}-1$ 与 signed int 的 -1），指定相同的结果将简化电路设计，而设置异常则需要额外的设计和性能损失。此外，极少数软件需要在不触发异常时自己检查和控制除 0 的情况，而这样的设计也可以让开发者更容易控制这些情况。

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



fcsr 控制寄存器中的 fflags 是浮点异常位标志。

fflags 各位含义：

NV：Invalid Operation，第4位，表示无效操作异常。

DZ：Divide by Zero，除以零异常。

OF：Overflow，表示上溢异常，即浮点运算的结果太大，超出表示范围。

UF：Underflow，表示下溢出异常，当运算结果过于接近0。

NX：Inexact，最低位，表示不准确异常。

fflags 置位并不会直接导致浮点异常，而是要求在软件中针对标志位进行显示的检查。开发者或操作系统应检查 fflags，对异常情况进行处理。

3)，调研其他指令集架构（x86 ARM 等）如何处理除数为0。

① 整数除法：x86 架构中使用 #DE (divide error) 异常指示处理除数为0的异常。

② 浮点数除法：当被除数非0，结果为 +∞ 或 -∞，当被除数也为0则结果为 Not a Number (NaN)，并可以选择引发一个浮点异常。

2、写出以下程序在 RISC-V 中应当处于的特权等级。

1) Linux Kernel：操作系统内核，为管理员模式。

2) BootROM：机器模式

3) BootLoader：是运行操作系统的前提，为机器模式。

4) USB Driver：设备驱动，为管理员模式

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t0和t1，C的地址存放于t2.

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

分析：由函数声明，我认为C应该是值存放于t2，而非地址，否则将变为 int *C. 如C的地址存放于t2，需额外的操作将C值从栈中取出. 汇编代码如下：

```
start: addi  sp, sp, -8     #考虑sp而非s0寄存器
       sw    ra, 4(sp)      #保存返回地址
       li    t3, 0          # i=0
       li    t4, 100

loop:
       beq   t3, t4, end
       slli  t5, t3, 2      #4*i→t5
       add   a3, t1, t5
       lw    a4, 0(a3)
       mul   a4, a4, t2
       add   a6, t0, t5
       sw    a4, 0(a6)      A[i]=a4=B[i]*C
       addi  t3, t3, 1
       j     loop

end:   lw    a0, 0(t0)     #A[0]→a0, 返回值
       lw    ra, 4(sp)     # 从栈中恢复ra
       addi  sp, sp, 8
       ret
```

14. 写出实现以下C程序的32位RISC-V汇编代码，假设a、b、c分别存放在寄存器a0、a1和a2

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

汇编代码：

```
    bgt  a0,a1,case1   # if a>b
    neg  a1,a1
case1:
    add  a2,a0,a1
```

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;
```

汇编代码：

```
    sw   t0,0(t0)     # p[0]=p
    li   t1,3         # int a=3
    sw   t1,4(t0)     # p[1]=a
    slli t2,t1,2
    add  t3,t0,t2
    sw   t1,0(t3)     # p[a]=a
```

16. 写出实现以下C程序的32位RISC-V汇编. 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;
}
```

汇编代码:

```
swap:
  addi sp, sp, -16
  sw   ra, 12(sp)
  lw   t2, 0(t0)
  lw   t3, 0(t1)
  sw   t2, 0(t1)
  sw   t3, 0(t0)

  lw   ra, 12(sp)
  addi sp, sp, 16
  ret
```

11. 解释以下 RISC-V 汇编语代码实现的功能

```
    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
```

上述代码实现了将 a1 寄存器的左移 30 次的操作. 当 a0 中的值与 a2 不等时, 不断循环, 在每次循环时对 a1 寄存器的值左移 1 位 (相当于对无符号数进行 x2 操作), 并使 a0 中的值自增 1, 直到 a0 自增直至与 a2 寄所存储的值大小相同就跳出循环。由于一开始有 addi a0, x0, 0 和 addi a2, x0, 30, 因此一共左移 30 次.