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　　x0，0(x1)

3） calloffset:

　　auipc　　x1，offset[31:12]+offset[11]

　　jalr　　x1，offset[11:0](x1)

4） mvrd, rs:

　　addi　　rd，rs，0

5） rdcyclerd:

　　csrrs　　rd，cycle[0],x0

6） sext.wrd, 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：

addt0, t1, t2

_____

_____

bnet3, 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）

slti　　t3,t2,0

slt　　t4,t0,t1

2） 当 t0 和 t1 都是无符号数时，请给出尽量简单的检测 addt0, t1, t2 指令加法是否溢出的指令序列。

add　　t0,t1,t2

slt　　t3,t0,t1

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：

overflowflag，溢出标志位，它记录的不是二进制下的进位，而是十进制下的不合理结果。OF 标志位根据操作数的符号及其变化情况来设置：若两个操作数的符号相同，而结果的符号与之相反时，OF=1，否则 OF=0。溢出位既然是根据数的符号及其变化来设置的，当然他使用来表示带符号数的溢出的。

ARM：

加法指令中(包括比较指令 CMN，根据两数相加结果进行比较)，当结果产生了进位，则 C=1，其他情况 C=0；(进位标志，发生进位时 C=1)

CPSR 的 V 位表示溢出标志（oVerflowflag)：

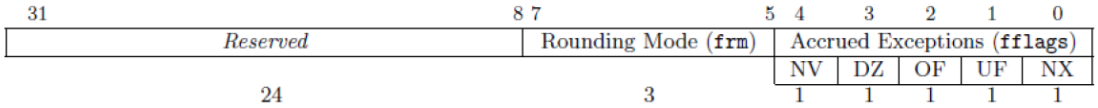
在加/减法运算指令中，当操作数与运算结果为二进制的补码表示的带符号数时，V=1 表示符号位溢出。如：两个正数执行 ADDS 指令后，和运算结果的 MSB 为 1（视为负数），则发

生溢出。
注意：C 和 V 标志位的利用由程序员来决定，即若认为操作数为无符号数，则关心 C 标志；若认为操作数为有符号数，则关心 V 标志。

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^L-1
x
-1
x
不会抛出异常。因为极少数程序需要这种行为，而且在那些软件中可以很容易地检查是否除零。
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？



NV：无效操作（包括 0/0、负数开平方根等）
DZ：除数为 0
OF：溢出（计算过程中数字过大，超出浮点数表示范围）
UF：下溢（输入过程中位数过长溢出到下位，结果过小无法正常显示）
NX：不精确（Inexact）
当浮点数运算产生异常时，相关的 fflags 位将被设置为 1，否则为 0。如果 fflags 被置位，处理器不会陷入系统调用，但程序可以通过检查这些位来判断浮点运算的结果是否正常。如果需要处理异常，可以通过相应的异常处理程序来处理。
4） 调研其他指令集架构（如 x86、ARM 等）是如何处理除数为 0 的。
x86 将除数为 0 的情况认为是溢出。
ARM 中除以 0 事件由内核的配置控制寄存器 CCR 的 DIV_0_TRP 控制。该位清 0 时，系统不对除以 0 事件触发异常，硬性返回 0 值作为结果；该位为 1 时，触发除 0 异常。

12. 写出以下程序在 RISC-V 中应当处于的特权等级。
1） LinuxKernel
管理员模式
2） BootROM
机器模式
3） BootLoader

机器模式

4） USBDriver

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

```
.globl vecMul
.type vecMul, @function
vecMul:
    addi    sp, sp, -16         # 分配栈空间
    sd      ra, 0(sp)           # 保存返回地址
    sd      s0, 8(sp)           # 保存 s0

    addi    s0, zero, 0         # i = 0
    addi    s1, zero, 100       # s1 = 100
loop:
    beq     s0, s1, exit        # i == 100，跳出循环

    slli    t0, s0, 2           # i * 4
    add     t1, a1, t0          # &B[i]
    lw      t2, 0(t1)           # B[i]
    mul     t3, t2, a2          # B[i] * C
    add     t4, a0, t0          # &A[i]
    sw      t3, 0(t4)           # A[i] = B[i] * C

    addi    s0, s0, 1           # i = i + 1
    j       loop

exit:
    lw      a0, 0(a0)           # 返回 A[0]
    ld      ra, 0(sp)           # 恢复返回地址
    ld      s0, 8(sp)           # 恢复 s0
    addi    sp, sp, 16          # 释放栈空间
    ret                         # 返回
```

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, plus
sub     a2, a0, a1
j       exit

plus:
add     a2, a0, a1

exit:
```

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;
```

```
addi    t2, t2, t0
sw      t2, 0(t2)

addi    t1, zero, 3
addi    t2, t0, 4
sw      t1, 0(t2)

slli    t3, t1, 2
add     t2, t0, t3
sw      t1, 0(t2)
```

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;
```

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

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
```

用循环来计算 2^{30} 并存入 a1