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) 空操作指令：add x0 , x0 , 0

2) 子程序返回：jalr x0 , x1 , 0

3) 调用远程子程序： auipc x1 offset[31:12]

   jalr x1 , rs , 0

4) 复制寄存器： addi rd rs 0

5) 读取 cycle CSR 的低 XLEN 位.(即硬件线程开始执行以来的时钟周期数)

   ⇒ csrrs rd , cycle , x0

6) 将一个32位有符号整数低16位进行符号扩展，扩展为一个32位有符号整数

   ⇒ auipc rd , rs

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 　　t1+t2=t0

____

____

bne t3,t4,overflow 　　t3≠t4 　跳 overflow

若 t1 和 t2 都是有符号数，请在横线处填入正确的指令，使得当 t1 和 t2 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 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。

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) slti t3 , t2 , 0 　　/ t2<0 时 t3 为1，反之 /

   slt t4 , t0 , t1 　　/ t0<t1 时 t4 为1，反之 /

2) addu t0 , t1 , t2 　　/ 不做溢出限制的加法 /

   bne. t0 , t1 , overflow

3) 对于 ARM 体系结构，通过 CRSR 的状态寄存器反映当前指令溢出状态。对于 MIPS，则通过指令触发中断的方式溢出信号，通知处理器

对于X86，利用CF、OF标志位分别识别无符号数运算与有符号数运算的溢出

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	$2^{XLEN}-1$	x	-1	x

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？

31–8	7–5	4	3	2	1	0
Reserved	Rounding Mode (frm)	Accrued Exceptions (fflags)				
		NV	DZ	OF	UF	NX
24	3	1	1	1	1	1

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) 会引起异常，这个异常在绝大多数执行环境中会导致自陷。

当语言标准要求一个除以0异常必须导致一个立即的控制流改变时，只需在每个除法操作时增加一条分支指令即可。

2) NV：非法操作

DZ：除以0

OF：上溢

UF：下溢

NX：不精确

会陷入系统调用

3) X86除法中，若除数为0，则运行时会触发0号中断。

ARM除法中，若除数为0，会触发 undefine instruction.

12. 写出以下程序在 RISC-V 中应当处于的特权等级。
1) Linux Kernel
2) BootROM
3) BootLoader
4) USB Driver
5) vim

1) 机器模式

2) 机器模式 （Boot ROM是嵌入处理器芯片内的一小块掩模ROM或写保护闪存）

3) 管理员模式 （Boot Loader负责查找和加载在该芯片上运行的最终操作系统或固件）

4) 管理员模式 （USB driver →设备驱动程序）

5) 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   x10 , x0, x0      # i=0
addi  x11 , x0, 100     # x11=100 .
Loop:
 bge   x10 , x11, exit    # i>100时.exit
 addi  x10 , x10, 1       # ++i
 sll   x12 , x11, 2       # i*4
 add   x1 , x12 , t1      # 地址变为 t1+x12.
 lw    x1 , 0(x1)         # *(B+i) =x1
 mul   x0 , x1 , t2.      # x0内值为B[i]*C 即x0内值为A[i]
add    x2 , x12. t0.      # A[i]地址.
 sw    x0 , 0(x2).        # 在A[i]中存x0值 .
 j     Loop
exit:
jalr   x0., t0., 0        # return  A[0]
```

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;
}
```

```
blt   a0 , a1 , part2.    # a<b 时跳转 part2
add   a2 , a0 , a1.       # c=a+b.
j     part3
```

```
part2:
 sub  a2, a0, a1          # c = a - b
part3:
  nop
```

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;
```

```
# Assume  x8  holds  pointer to  p
sw  t0 , 0(x8)
li  t1 , 3
addi x9 , x8, 4     # x9存p[1]地址.
sw  t1 , 0(x9)
sll  x10 , t1 , 2   # a*4.
add  x10 , x10 , x8   # x10存p[a]地址
sw  t1 , 0(x10)
```

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;
}
```

```
# Assign  x8 = tmp.
 lw  x8 , 0(t0).    # tmp = *a
 lw  x9 , 0(t1)     # x9 = *b.
 sw  x9 , 0(t0)     # *a = x9 = *b
 sw  x8 , 0(t1)     # *b = tmp
 nop
```

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
```

实现功能为进行2的累乘运算(重复30次) ⇒ 得到 2^{30}.