

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) jalr     x0, x1, 0
3) auipc    x1, offset [31:12]
   jalr     x1, x1, offset [11:0]
4) addi     rd, rs, 0
5) csrwr    rd, cycle[h], 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
```

```
slti t3,t2,0
```

```
slti t4,t0,t1
```

```
bne t3,t4,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。

```
2) add     t0, t1, t2
   bltu     t0, t1, overflow
```

3) x86: 31x overflow flag (OF) /
carry flag (CF)

ARM: 31x overflow flag (V)

当正数加负数/负数加正数时，overflow flag = 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	$2^{xLEN} - 1$	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				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。

2) NV = Invalid Operation 无效操作.
DZ: Divide by Zero 除数为 0
OF: Overflow 溢出
UF: Underflow 下溢.
NX: Inexact 不精确

会.

3) x86: 除数为 0, 除溢出引起 0 号中断

ARM: 没有特定的除法溢出中断, 但会导致其他中断

12. 写出以下程序在 RISC-V 中应当处于的特权等级。

1) Linux Kernel Supervisor
2) BootROM Machine
3) BootLoader Machine
4) USB Driver Supervisor
5) vim User

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];  
}
```

Assume x8 holds pointer to A
x9 holds pointer to B
Assign x10 = i

```
add x10, x0, x0 # i = 0  
addi x11, x0, 100 # x11 = 100  
loop:  
bge x10, x11, exit
```

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];
}
```

```
# Assume x8 holds pointer to A
#          x9 holds pointer to B
#          x10 C
# Assign x11 = i
```

```
add x11, x0, x0      # i = 0
addi x12, x0, 100     # x12 = 100
loop:
    bge x11, x12, exit
    sll x13, x11, 2
    add x13, x13, x8
    add x14, x14, x8
    lw x13, 0(x13)
    lw x14, 0(x14)
    mul x13, x14, x10
    sw x13, 0(x13)
    addi x11, x11, 1
} loop
exit
```

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。

```
x8, x9, x10
int a, b, c;
if(a > b){
    c = a + b;
}
else{
    c = a - b;
}
```

```
bge x8, x9, 2f
```

```
1: sub x10, x8, x9
   j exit
```

```
2: add x10, x8, x9
   j exit
```

```
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;
```

```
lw x3 0(t0)
addi t1 x0 3
sb t0 4(t1)
li x4 t1 2
sb t0 x4(t1)
```

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;
}
```

```
lw t0 0(x9)
add x8 x0 x9
sw t1 0(x8)
```

17. 解释以下 RISC-V 汇编代码实现的功能。

<code>addi a0,x0,0</code>	<code>a0 = 0</code>
<code>addi a1,x0,1</code>	<code>a1 = 1</code>
<code>addi a2,x0,30</code>	<code>a2 = 30</code>
<code>loop: beq a0,a2,done</code>	<code>if a0 = a2 jump to done.</code>
<code>slli a1,a1,1</code>	<code>a1 = 2 a1</code>
<code>addi a0,a0,1</code>	<code>a0++</code>
<code>j loop</code>	<code>loop</code>
<code>done: # exit code</code>	

```
int a0 = 0, a1 = 1, a2 = 30;
for ( ; a0 != a2; a0++) {
    a1 = 2 a1;
}
```