No.
Date

2-5: 编写以下伪指令等价的基本指令或指令组合

(1) nop : addi x0, x0, 0 (空操作)

(2) ret : jump ra.

(3) call offset : addi sp, sp, -32
addi ra, cp, 12
sd ra, 24(sp)
jump offset

(4) mv rd, rs : addi rd, rs, 0

(5) rdcycle rd: 不知道

(6) sext.w rd, rs : slli rd, rs, 16
srai rd, rd, 16

2-7: (1) add t0, t1, t2 _____ _____ bne t3, t4, overflow

填入正确指令使 t1、t2 两个加法溢出时，控制流可以跳转到 overflow.(t1、t2都是有符号数)

解: slti t3, t2, 0
slt t4, t0, t1

(2) 若 t1, t2 都是无符号数，给出尽量简单的检测加法溢出的指令序列

解: add t0, t1, t2
blt t0, t1, overflow

(3) 其他指令集:

ARM: 通过 CPSR 寄存器中的溢出标志位反映当前指令溢出情况

x86: 专门提供了一系列指令来检测是否发生溢出

No.
Date

1-8 (1) RISC-V 中，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	Op=REMU	Op=DIV	Op=REM
Op	rd, rs1, rs2	x	0	$2^{XLEN}-1$	x	-1	x

答：不会引起抛出异常，而是在每条除法指令后添加条件分支跳转指令。这样做的原因是可以减少开销，降低硬件设计复杂度.

(2) NV：无效操作　DZ：除数为0
OF：上溢　UF：下溢
NX：不精确

(3) ARM、X86中，遇到除数为0时会引起 CPU 停止执行，跳转至异常处理.

Date

2-12 写出以下程序在RISC-V中应当处于的特权等级.

(1) Linux kernel: 管理员模式

(2) BootROM: 机器模式

(3) Bootloader: 机器模式

(4) USB Driver: 管理员模式

(5) vim: 用户模式.

写出以下程序的32位RISC-V汇编代码.

2-13: 设A、B元素地址存在t0、t1, C的地址存于t2.

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

解:

```
vecMul: addi sp, sp, -32
        sd   ra, 24(sp)
        sd   s0, 16(sp)
        addi s0, sp, 32
        mv   t3, t0
        mv   t4, t1
        li   t5, 0
        li   t6, 100
        lw   t7, 0(t2)
loop:   lw   t8, 0(t4)
        mul  t8, t7, t8
        sw   t8, 0(t3)
        addi t3, t3, 4
        addi t4, t4, 4
        addi t5, t5, 1
        blt  t5, t6, loop
end:    lw   t0, 0(t0)
        ld   ra, 24(sp)
        ld   s0, 16(sp)
        addi sp, sp, 32
        ret
```

No.
Date . .

2-14 a,b,c分别对应寄存器 a0、a1、a2

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

解：
```
blt  a1 a0 8
add  a2 a0 a1
add  a2 a0 a1
```

2-15 p指针已通过 int *p=(int*) malloc(4*sizeof(int)) 得到. 且 p存放于t0, a存放于t1

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

解：
```
sw  t0 0(t0)
li  t1 3
sw  t1 1(t0)
add t2 t0 t1
sw  t1 0(t2)
```

2-16 指针 a、b 分别存放于 t0、t1.

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

解：
```
swap: addi sp, sp, -32
      sd   ra, 24(sp)
      sd   s0, 16(sp)
      addi s0, sp, 32
      lw   t2, 0(t0)
      lw   t3, 0(t1)
      sw   t3, 0(t0)
      sw   t2, 0(t1)
      ld   ra, 24(sp)
      ld   s0, 16(sp)
      addi sp, sp, 32
      ret
```

Date . .

2-17 解释以下代码的功能

```
    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
```

解：该代码完成了将 a1（初值为1）左移a2位（30位）的操作，最终得到 $a1 = 2^{30}$