定除数与被除数均为有符号数，且符号相同？有符号取余指令则类似。

11、下列指令使用的寻址模式

(1) jal ra, 0x88　　PC相对地址寻址，0x88为偏移量。

(2) jalr x0, ra, 0　　偏移量寻址（基址ra + 偏移量0）。

(3) addi a0, a1, 4　　立即数寻址。

(4) mul a0, a1, a2　　寄存器寻址。

(5) lw a4, 16(sp)　　偏移量寻址（基址sp + 偏移量16）。

3.21 第5周 Chapter two

3、RISC-V 汇编中存在许多的伪指令，它们一般是具有特殊操作数的基本指令或指令组合，罗下列伪指令的等价指令（基本指令 or 指令组合）

(1) nop　addi x0, x0, 0

(2) ret　jalr x0, x1, 0

(3) call offset　auipc x6, offset[31:12]　jalr x1, x6, offset[11:0]

(4) mv rd, rs　addi rd, rs, 0

(5) rdcycle rd　csrrs rd, cycle[h], x0

(6) sext.w rd, rs　addiw rd, rs, 0

7、RISCV 标准指令集未对加法指令溢出引入专用的标志位，因此通常需要额外的指令加以检查

(1) 考虑如下指令序列

```
add  t0, t1, t2
slti t3, t2, 0
-------------------
slt  t4, t0, t1
-------------------
bne  t3, t4, overflow      t0, t1为有符号数
```

bltu <

slt 计< _←1

else _←0

(2) 当t0,t1为无符号数时，检测方法为：

add t0,t1,t2

bltu t0,t1,overflow

(3) x86中引入了 CF标志来指示前面无符号整数的溢出与否，用OF标志来指示有符号整数的溢出与否；RAM中则是引入C标志与V标志分别指示无符号与有符号整数的溢出情况.

8. (1) 整型除法中除数为0，RISC-V不会抛出异常.

指令	rs1	rs2	op=DIVU时	op=REMU时	op=DIV时
op rd,rs1,rs2 (f表示4个1)	X	0	rd值 0x…f 即 op=REM时 -1	rd值 X(被除数)	rd值 0x…f 即-1

(2) 浮点数除法除数为0引发 fcsr控制寄存器中 fflags部分被置位. fflags为浮点异常累积标志寄存器.

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

NV：非精确异常　UF：下溢异常　OF：上溢异常　NX：无效操作数异常

DZ：除0异常

一般情况下处理器会将控制权交给操作系统内核，由内核来处理异常，因此fflags被置位并不会直接使处理器陷入系统调用，但会触发异常处理机制。

(3) x86处理除0异常时会在编译器中加入(插入)一个 idivl命令，当此命令遇到除数为0的情况时 CPU会产生一个除以0的陷阱，类似于中断；而ARM则会用编译器检查数据否为0，如果为0则跳过除法指令操作并设置一个标志位.

12. 当以下程序在 RISC-V 中运行的特权等级

(1) Linux Kernel S-mode 管理员模式

(2) BootROM M-mode 机器模式

(3) BootLoader M-mode 机器模式

(4) USB Driver S-mode 管理员模式

(5) Vim U-mode 用户模式

13、(返回值存到 a0 寄存器中)

```
add  t3, x0, x0   # i=0
addi t4, x0, 100  # t4=100
Loop:
bge  t3, t4, end
slli t5, t3, 2
add  t5, t5, t1
lw   t5, 0(t5)
lw   t6, 0(t2)
mul  t7, t5, t6
slli t5, t3, 1
add  t5, t5, t0
sw   t7, 0(t5)
j    Loop
end:
lw   a0, 0(t0)
ret
```

14、

```
part1:
bge  a1, a0, part2
add  a3, a1, a0
j    exit
part2:
sub  a3, a0, a1
j    exit
exit:
ret
```

15、

```
part:
add  t3, t0, x0
addi t1, x0, a
sw   t0, 0(t3)
add  t3, t0, 4
sw   t1, 0(t3)
slli t4, t1, 2
add  t3, t0, t4
sw   t1, 0(t3)
```

16、

```
lw   t2, 0(t0)
lw   t3, 0(t1)
add  t4, t3, x0
add  t2, t3, x0
add  t3, t4, x0
sw   t2, 0(t0)
sw   t3, 0(t1)
ret
```

或者

```
lw   t2, 0(t0)
lw   t3, 0(t1)
sw   t3, 0(t0)
sw   t2, 0(t1)
ret
```

17.

```
addi a0, x0, 0    # a0=0
addi a1, x0, 1    # a1=1
addi a2, x0, 30   # a2=30
loop:
beq  a0, a2, done # a0≥a2, →done
slli a1, a1, 1    # a1=2a1
addi a0, a0, 1    # a0=a0+1
j    loop
done:  #exit code
```

```
int S=1
for (int i=0; i<30; i++)
    S=2*S
```

功能：计算 2^{30}.

结果保存于 a1 寄存器中.