3. 伪指令 → 基本指令

(1) nop ⟶ addi x0, x0, 0 ，空指令

(2) ret ⟶ jalr x0, x1, 0 ，返回指令

(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, x0 读当前数值寄存器指令

(6) sext.w rd, rs ⟶ addiw rd, rs, 0 符号位扩展指令

7. (1) 有符号数加法溢出检测：

```
add  t0, t1, t2
slti t3, t2, 0
slt  t4, t0, t1
bne  t3, t4, overflow
```

(2) 无符号数：

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

(3) ARM汇编中可以通过检查程序状态寄存器 PSR 中的标志位来检测加法溢出.

例如，
```
MOV  R0, R2
ADDS R1, R0, #constant
BVS  overflow
```
如果发生溢出，则跳转到 overflow 标签处.

X86汇编可以通过检查程序状态寄存器 PSR 中的标志位来检测加法溢出.

如果发生了溢出，则程序状态寄存器 "OF" 标志位将被设为1，否则将被清零.

8. (1) Op = DIVU 时 rd = 0xffffffffffffffff

Op = REMU 时 rd = x

Op = DIV 时 rd = 0x7fffffffffffffff

Op = REM 时 rd = x

'div'和'divu'的结果是最大有符号整数和最大无符号整数，此时表示整数溢出的标准行为。在'rem'和'remu'中，除数为0时得被除数本身，这是为了为使余数仍有意义，便特法与操作对不同的被除数都有确定的行为，这可以确保不会出现未定义的行为，使代码更可靠。

(2) NX-非精确异常
NX=0时 没有产生非精确异常
NX=1时 产生非精确异常

UF - 下溢异常
UF=0时 没有下溢异常.
UF=1时 有下溢异常.

OF - 上溢异常
OF=0时 没有产生上溢异常
OF=1时 有上溢异常

DZ - 除0异常
DZ=0时 没有除0异常
DZ=1时 有除0异常

NV - 无效操作数异常
NV=0时 没有产生无效操作数异常
NV=1时 产生无效操作数异常
fflags被置位不会使得处理器陷入系统调用。

(3) x86架构中，除以0会产生一个异常。x86处理器使用一个称为"除法错误"的异常来处理这种情况。
处理器程序可以检查错误码，并根据需要采取适当措施

12. (1) Linux kernel 应当处于 S-mode
(2) Boot ROM 应当处于 M-mode
(3) Boot Loader 应当处于 M-mode
(4) USB Driver 应当处于 S-mode
(5) Vim 应当处于 U-mode

13.
```
start:  addi  sp, sp, -32
        sd    ra, 24(sp)
        sd    s0, 16(sp)
        addi  s0, sp, 32
        mv    a3, zero      # t3=i
        li    a4, 100       # t4=100
        lw
loop:   sll   a5, a3, 2     # a5=i*4
        add   a6, a5, t0
        lw    a6, 0(a6)     # a6=A[i]
        add   a7, a5, t1
        lw    a7, 0(a7)     # a7=B[i]
        mul   a6, a7, t2    # A[i]=B[i]·C
        addi  a3, a3, 1     # i++
        add   a7, a5, t0
        sw    a6, 0(a7)
        bge   a3, a4, end   # i<100
end:    ld    t0, 0(t0)
        ld    ra, 24(sp)
        ld    s0, 16(sp)
        addi  sp, sp, 32
        ret
```

14：假设 a、b 和 c 分别对应 a_0、a_1、a_2

则

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

→

```
    bgt   a1, a0, l
    add   a2, a1, a0
l:
    sub   a2, a0, a1
```

15.
```
sw   t0, 0(p)
addi t1, x0, 3
addi t2, t0, 4
sw   t1, 0(t2)
mulj t2, t1, 4
add  t2, t2, t0
sw   t1, 0(t2)
```

16:
```
swap:
    mv   t2, t0
    mv   t0, t1
    mv   t1, t2
    ret
```

17. 原RISC-V汇编代码转化C代码:

```
int  i , a0=0, a1=1;
for (i=0; a0<30; i++){      ,即将a1左移30位,即乘以2^30.
    a1 = a1<<2;
    i++;
    }
```