3. 解：伪指令　　等价指令

(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[31:0], x0

(6) sext.w rd rs　　addiw rd, rs, 0

7. 解：~~对RV64I指令集~~

(1)
```
add  t0, t1, t2
slti t3, t2, 0
slt  t4, t0, t1
bne  t3, t4, overflow
```

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

(3) X86：有符号数，OF=1时检测到数据溢出；对加减法都同进行判断
　　无符号数，CF=1时检测到数据溢出

ARM：有符号数：由CPSR中的V位表示溢出标志，V=1表示符号位溢出
　　无符号数：由CPSR中的C位表示进位或借位标志，对加法，C=1时产生进位

8. 解：(1) 除数0不会引起异常

OP=DIVU时，rd值为 $\sum^{XLEN-1}$

OP=REMU时，rd值为 x

OP=DIV时，rd值为 -1

OP=REM时，rd值为 x

原因：若触发异常，这个异常在绝大多数执行环境里，会导致一个自陷。但这将成为标准ISA中唯一的算术自陷，为了改进这种情况，在除法操作后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

flags被置位不会使处理器陷入系统调用，而是将设置禁止，需要软件明确地对标志进行检查

(3) X86：当除数为0时，则认为是溢出，引起0号中断（int 0）
ARM：会明确发异常，跳转到异常向量执行中断操作

12. (1) M级，可选S级（在M级下运行）
(2) M级
(3) M级或S级
(4) M级，需要直接访问硬件资源
(5) U级，不需要访问特权资源，就在用户模式

13.
```
add  t3, zero, zero     # i=0
lw   a3, 0(t2)          # a3=C
addi t4, zero, 100      # t4=100
Loop:
bge  t3, t4, exit
sll  t5, t3, 2          # i*4
add  t6, t5, t1         # &Ari
lw   a4, 0(t6)          # *(Ari)
add  t6, t5, t0         # &Ari
mul  a5, a4, a3
```

```
    sw   a5, 0(t6)      # *(A+i)=*(B+i)*C
    addi t3, t3, 1      # i++
    j    Loop
exit:
    lw   a0, 0(t0)      # *A
    ret
```

14.解:

```
part1:  bge  a1, a0, part2
        add  a2, a0, a1
        j    end
part2:  sub  a2, a0, a1
        j    end
end:
```

15.解:

```
    sw   t0, 0(t0)
    addi t1, zero, 3
    sw   t1, 4(t0)
    sll  t3, t1, 2      # a*4
    add  t3, t0, t3     # p+a*4
    sw   t1, 0(t3)
```

16.解:

```
lw   t2, 0(t0)    # temp=*a
lw   t3, 0(t1)
sw   t3, 0(t0)    # *a=*b
sw   t2, 0(t1)    # *b=tmp
ret
```

17.解: 设 a0, a1, a2 中分别存的是 i, a, 30

那么大致的C语言为

```
{
    int a=1;
    int i;
    for(i=0; i<30; i++)
        a*=2;
    return
}
```

∴ 它想实现的功能是通过循环累乘得到 $2^{30}\cdot a$, 即 2^{30}