3.21.

2-3. nop : addi $x_0, x_0, 0$

ret : jalr $x_0, x_1, 0$

call offset : auipc x_6 offset[31:12]

jalr x_1, x_6, offset[11:0]

mv rd, rs : addi rd, rs, 0.

rdcycle rd : csrrs rd, cycle, x_0

sext.w rd, rs : addiw rd, rs, 0.

2-7. (1) slti $t_3, t_2, 0$.

Slt t_4, t_0, t_1

(2) add t_0, t_1, t_2.

bltu t_0, t_1, overflow

(3). ~~x86: 通过设置溢出标志位 OF (Overflow Flag) 来检测溢出.~~

x86: 通设置进位符号位 CF (Carry Flag) 和溢出标志位 (overflow Flag) 来检测溢出。其中，前者检测无符号数的溢出. 即检测最高位是否会产生进位；后者检测无符号数的溢出，主要针对 正+正=负 和 负+负=正的情况。(正+负不会溢出).

当产生进位，CF=1，否则=0；当溢出，OF=1，否则=0

ARM: 设置 C, V 标志位.

C: 加法运算产生进位时 (无符号溢出). C=1，否则 C=0.

V: 当符号位溢出时. V=1，否则 V=0 (有符号溢出)

习题8

(1)

指令	rs1	rs2	rd:op=DIVU	rd:op=REMU	rd:op=DIV	rd:op=REM.
op rd,rs1,rs2	x	0	$2^{XLEN}-1$	x	-1	x.

不会产生异常。

设计原因：这样避免在处理器中引发异常，同时简化硬件设计。因为在产生异常时，会增加处理器用于处理异常的开销，降低效率。

(2). NV: Invalid operation : 无效操作

DZ: Divide by Zero. : 除以0.

OF: overflow : 运算结果溢出

UF: underflow : 运算结果下溢

NX: Inexact : 不精确

fflags置位不会使处理器陷入系统调用，只会产生浮点异常，从而引发异常处理例程。处理器会根据异常类型进行相应处理，如向用户提供异常信息、恢复寄存器状态、跳转到异常处理程序等。

(3) x86架构中，如果除数为0，会触发异常"divide error"，此时处理器产生一个异常处理程序。处理器会中止当前程序运行，保存断点信息，之后执行异常处理程序，执行完毕后恢复现场，继续执行接下来的指令。

ARM架构中，除以0产生"divide by zero"异常，处理器产生异常处理程序，该程序地址由异常向量表捕获中对应条目指令

2 2-12

Linux kernel : 机器模式

BootRom : 机器模式

BootLoader : 管理员模式

USB Driver : 用户模式

vim : 用户模式

: 2-13.

```
.text
.globl vecMul
.type vecMul, @function

vecMul:
    addi  sp, sp, -32
    sw    ra, 28(sp)
    sw    s0, 24(sp)
    addi  s0, sp, 32.        # 开栈
    sw    t0, 20(sp)
    sw    t1, 16(sp)
    sw    t2, 12(sp)         # 存参.

    lw    a0, 20(sp)         # 取参.
    lw    a1, 16(sp)         # a0~a2寄存器对应
    lw    a2, 12(sp)         int*A, int*B, int C.
```

```
# 设置计数器：t0是计数器，t1是控制循环结束标志。
addi    t0, zero, 0
addi    t1, zero, 396.
# 循环：
Loop:
    lw      t2, 0(a1)           # 取出 B[i]
    lw      t3, 12(sp)          # 取出 C.
    mul     t2, t2, t3          # 计算 C * B[i]
    sw      t2, 0(a0)           # A[i] = C * B[i]
    addi    a0, a0, 4           }
    addi    a1, a1, 4           } # i = i+1
    addi    t0, t0, 4.          # 计数器自增
    bne     t0, t1, loop.       # 循环结束.
# 准备返回值
    lw      a0, 20(sp)          # 栈里. 20(sp)是A[0]地址.
    lw      a0, 0(a0)           # A[0]的值存入a0待返回
# 关栈返回
    lw      s0, 24(sp)
    lw      ra, 28(sp)
    addi    sp, sp, 32.
    ret.
```

2 2.14.

```
bge   a1, a0, else.
add   a2, a0, a1
j     forward
else: sub   a2, a0, a1
forward: ...
```

2.15.

```
lw    t0, 0(t0)      # p[0] = p.
addi  t1, zero, 3.   # int a = 3.
addi  t0, t0, 4.
sw    t1, 0(t0).     # p[1] = a.
muli  t2, t1, 4.
addi  t0, t0, -4
addi  t0, t0, t2.
sw    t1, 0(t0)      # p[a] = a.
```

2.16.

```
.text
.globl swap
.type  swap, @function.

# 开栈:
swap: addi  sp, sp, -32.
      sw    ra, 28(sp)
      sw    s0, 24(sp)
      addi  s0, sp, 32.
```

操作：

```
lw   t2, 0(t0)      # 将*a存入寄存器t2，t2也可用作temp的寄存器.
lw   t3, 0(t1)      # *b存入t3
sw   t3, 0(t0)      # a* = b*
sw   t2, 0(t3)      # b* = temp.
  # 关栈返回
lw   s0, 24(sp)
lw   ra, 28(sp)
addi sp, sp, 32
  ret
```

17. 功能：求2^{30}次方的值（计算结果存入a1）.