11. 1) jal ra, 0x88 立即数寻址.
2) jalr x0, ra, 0. ~~偏移量寻址~~. 寄存器间接寻址
3) addi a0, a1, 4. 立即数寻址.
4) mul a0, a1, a2. 寄存器寻址.
5) ld a4, 16(sp). 偏移量寻址.

3/21. Chapter2.
3. 写出伪指令背后的基本指令/指令组合.
1). nop. addi x0, x0, 0.
2). ret. jalr x0, 0(x1).
3). call offset auipc x1, offset[31:12]+offset[11].
jalr x1, offset[11:0](x1)
4). mv rd, rs addi rd, rs, 0.
5). rdcycle rd csrrs rd, cycle, x0. (read cycle counter)
6) sext.w rd, rs addiw, rd, rs, 0 (sign extension).

~~7. 1). xor t3, t1, xor t3, t0, t1~~
~~xor t4, t0, t0 xor t4, t0, t2~~.
~~2~~. set less than immediate (if $t_2<0$, $t_3=1$, else $t_3=0$).
7. 1). slti t3, t2, 0
slt t4, t0, t1
set less than (if $t_0<t_1$, $r_4=1$, else $r_4=0$).
2) bltu t0, t1, overflow
branch less than.

3) 在x86架构中，加法指令有多种不同形式，有些指令会设置标志位来检测加法溢出，有些则不会！

例如，add指令会将2个操作数相加，并将结果存储到第一个操作数中，同时根据结果设置标志位，其中CF（Carry Flag）标志位表示加法是否产生了进位（溢出）。OF（Overflow Flag）标志位表示加法是否溢出。

而addx指令不设置OF位，它将2个操作数相加后把结果存储到第一个操作数中，同时根据结果设置CF和ZF（Zero Flag）标志位。

在ARM架构中，不同的加法指令用不同的标志位来检测加法溢出。

例如，add指令将2个操作数相加，并将结果存储到第一个操作数中。同时根据结果设置标志位，C表示是否进位，V表示是否溢出。

adds：N(Negative)、Z(zero)、C和V.(overflow)

8. 1). 指令 | rs1 | rs2 | Op=DIVU时 rd值 | Op=REMU时 rd值

Op.rd,rs1,rs2 | x | 0 | 2^L-1 | x

Op=DIV时 rd值 | Op=REM时 rd值

-1 | x

L: the width of the operation in bits: XLEN for DIVU and REMU.
or 32 for DIVUW and REMUW.

整型除法中除数为0会引起RISC-V异常。

这些异常在大多数执行环境中会导致陷阱。然而，这是标准ISA中唯一的算术陷阱（浮点异常会设置标志并写入默认值，但不会导致陷阱），并且需要语言实现指与执行环境的陷阱处理程序交互处理此情况；此外，若语言标准规定除以0异常必须导致立即的控制流变化，则每个除法操作只需要添加一个分支指令，并且可以在除法之后插入此分支指令，而且通常被非常可预测地不会被执行，从而只增加很少的运行时的开销。

提醒程序员检查。

DIVU：指令访问异常. DIVUW：除零异常. ⇒ 异常处理程序被调用.

此处，RISC-V中采取将除零的结果设置为所有位均为1的整数，这样可简化除法器，是简单无符号分频路实现的自然结果.

2). fflags 位是一个5位的标志位，用于记录浮点运算中的异常情况：

(Flag Mnemonic) 标志位名称	(Flag Meaning) 含义
NV	Invalid Operation
DZ	Divide by Zero.
OF	Overflow
UF	Underflow.
NX	Inexact. result.

① NV：当某个操作数为NaN时，或者某个算术运算的结果不是合法的浮点数时，会触发该异常.

② DZ：除数为0.

③ OF：操作结果超过了浮点数所能表示的范围.

④ UF：当某个操作结果非常接近于0，但无法被表示为一个正常的浮点数时.

⑤ NX：当某个运算结果无法被准确地表示为一个浮点数时.

某个异常被触发时，其对应的Flag会被置位.

当fflags中任意一个标志位置被置位时，处理器不会陷入系统调用，但会产生异常，执行异常处理程序.

1). ① x86：触发"除以零异常"。⇒ 异常处理程序。
② ARM：触发"除以零异常" ⇒ 异常处理程序。
③ MIPS：触发"算术异常"。⇒ 异常处理程序。

12. 1) Linux Kernel
2) Boot ROM
3) ~~USB Driver~~ Boot Loader
4) USB Driver
5) vim.

2. (Machine) (Supervisor).
3. (Machine).
2. (Supervisor).
2. (Supervisor).
0. (User).

~~3. # Assume t0 holds pointer to A. t1 holds pointer to B.~~
~~# Assume t2 holds pointer to C.~~
~~vecMul: add t3, x0, x0 # i=0.~~
~~addi t4, x0, 100 # t4=100.~~

~~Loop: bge t3, t4, exit~~
~~slli t5, t3, 2 # i*4.~~
~~add t5, t5, t0. # & of A+i. mv t6, t5.~~
~~add t5~~ add t5, t5, t0. # & of Ati
~~slli t6, t3, 2. # i*4. add t6, t6, t1. # & of Bti~~
~~add t6, t6, t1. # & of Bti lw t5, 0(t5). # *(A~~
~~lw t6, 0(t6). # *B~~

13. # Assume t_0, t_1, t_2 hold the pointer to A, B, *C.

```
    add  t3, x0, x0   # i=0
    addi t4, x0, 100  # t4=100.

Loop: bge  t3, t4, exit
    sll  t5, t3, 2    # i*4.
    add  t5, t5, t1   # & of (B+i).
    lw   t5, 0(t5).
    mul  t5, t5, t2.
    sll  t6, t3, 2
    add  t6, t6, t0.  # & of (A+i).
    sw.  t5, 0(t6).
    addi t3, t3, 1
    j    Loop

exit: lw a0, 0(t0).
    jr ra.
```

14. # Assume a_0, a_1, a_2 hold the pointer to *a, *b, *c

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

15. ① sw t0, 0(t0). # p[0]=p.

② li t1, 3 (或 addi. t1, zero, 3). # int a=3.

③ addi t2, t0, 4. # p[1]=a.

sw t1, 0(t2)

④ sll $t_1, t_1, 2$. # a*4.
add t_0, t_0, t_1.
sw. t_1, 0(t_0) # p[a] = a.

0(t_0). 16. # Assume t_0, t_1 hold the pointer to a, b.

优化前: mv t_2, t_0.	lw t_2, 0(t_0).
mv t_0, t_1.	lw t_3, 0(t_1).
mv t_1, t_2.	sw t_2, 0(t_1).
ret.	sw t_3, 0(t_0).
	ret.

17. ④ 功能: 计算 z^{30}. 存在 a_1 内.