3/21

3. 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]

~~auipc x6, offset[31:12]~~

4). mv rd, rs addi rd, rs, 0

5). rdcycle rd csrrs rd, cycle, x0

6). sext.w rd, rs addiw rd, rs, 0

7. 1). sub t3, t0, t1 / slti t3, t2, 0

mv t4, t2 / slt t4, t0, t1

2). bltu t0, t1, overflow

3). 在X86中，EFLAGS寄存器存储了一些标志位，其中CF用于指明对无符号整数算术操作的溢出条件；OF指明了对有符号整数算术操作的溢出条件；0代表不溢出，1代表溢出

在使用时，可以用 pushf 指令将标志压入栈中，在堆栈上进行读取，CF 位于第 0 位，OF 位于第 11 位。

在 ARM 架构中，APSR / CPSR 寄存器的前四位是 NZCV 标志位，其中 C 进位标志用于判断无符号整数操作是否溢出；V 溢出条件标志用于判断有符号整数算术操作是否溢出。

8 1)、

OP = DIVU 时 rd 值	OP = REMU 时 rd 值	OP = DIV 时 rd 值	OP = REM 时 rd 值
$2^{XLEN}-1$	x	-1	x

会引起异常，返回值被设置为全 1，因此对无符号数而言是 $2^{XLEN}-1$，对有符号数而言是 -1，这有助于简化硬件。

2)、 NV (Invalid Operation) 无效操作

DZ (Divide by zero) 除 0 异常

OF (Overflow) 结果溢出

UF (Underflow) 结果下溢

NX (Inexact) 结果不精确

fflags 被置位不会使得处理器陷入系统调用，在 RISC-V 中，为了保持 ISA 精简，并没有由浮点异常标志和内容控制的分支。

3)、 X86 架构中除 0 会引发中断，具体而言属于内部中断中的异常，处理过程为 1. 保存断点，将 FLAGS 寄存器，CS 和 IP 压栈。2. 执行中断服务程序 3. 返回断点继续执行。

ARM 架构中，可以通过 CCR 寄存器的 DIV_0_TRP 标志位控制是否触发异常，在不触发异常的配置下，除 0 操作会返回 0。

12. 1). S 2). M 3). M 4). S 5). U

13.
```
    add   a1, x0, x0        # a1 = 0
    addi  a2, x0, 100       # a2 = 100
LOOP:
    beq   a1, a2, end
    slli  a5, a1, 2         # i x 4
    add   a3, a5, t0        # & of A + 4i
    add   a4, a5, t1        # & of B + 4i
    lw    a4, 0(a4)         # *(B + 4i)
    mul   a6, a4, t2        # B[i] x C
    sw    a6, 0(a3)         # A[i] = B[i] x C
    addi  a1, a1, 1         # i++
    j     LOOP
end:
    mv    a0, t0
```

14.
```
    bltu  a1, a0, 1f
    j     2f
1:  add   a2, a0, a1
    j     end
2:  sub   a2, a0, a1
    j     end
```

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

16.
```
lw   a1, 0(t0)
lw   a2, 0(t1)
sw   a2, 0(t0)
sw   a1; 0(t1)
```

17.
```
int i = 0;
int a1 = 1;
for (i=0; i<30; i++)
    a1 = a1 × 2;
```
功能：计算 2^{30}