3. 1) nop = addi x_0 x_0 0

2) ret = jalr ra 0

3) auipc x_1 offset[31:12]

jalr x_1 offset[11:0](x_1)

4) add rd rs x_0

5) csrrs rd, cycle, x_0

6) addiw rd rs, x_0

7. 1) slti t_3 t_1 x_0

slt t_4 t_0 t_2

第一条指令比较 t_1 与 0，判断 t_1 是否为正，若 t_1 为正则 $t_3 = 0$，反之为 1

第二条指令比较 t_0 与 t_2，若 t_0 未溢出，则在 t_1 为正时 $t_0 > t_2$，t_4=

反之 $t_0 < t_2$，即 $t_4 = 1$，若 t_3 与 t_4 不同，说明 t_0 溢出.

2) sub t_3 t_0 t_1

bne t_3 t_2 overflow

3) x86 和 Intel 采用 OF 标志位判断，若溢出将 OF 位置 1，否则置 0，根据 OF 位判断异常

2.8 1).

指令	rs1	rs2	Op=DIVU 时 rd 值	Op=REMU 时 rd 值	Op=DIV 时 rd 值	Op=REM 时 rd 值
Op rd,rs1,rs2	x	0	2^L-1	x	-1	x

L为指令位数.

不会引起异常，因为除法器设置时可以采用在除法语句中添加分支实现。造成的开销很小，但如果单独为这异常设置异常处理，其不但会增加硬件开销且利用率不高(这将是唯一可能产生异常的情况)

2) NV：无效操作. DZ：除数为0. OF：溢出(上溢)

UF：溢出(下溢) NX：不精确

fflags被置位不会使处理器处于系统调用

3. 对于x86处理器，在除法操作时若除数0.会触发除法异常.处理器停止并执行异常处理程序

12. 1) 机器模式

2) 机器模式

3) 机器模式

4) 管理员模式

5) 用户模式

2.13

```
add   t3  x0  x0   # i=0
addi  t4  x0  400  # t4=4*100
lw    t5  0(t2)    # t5=C.

loop: bge   t3  t4 , END  # if(*i >=400 , 结束
```

```
add   t6  t3  t0    # & of (A+i)
add   t7  t3  t1    # & of (B+i)
lw    t8  0(t7)     # B[i]
mul   t8  t8  t5    # A[i]=B[i]*C
sw    t8  0(t6)     # 写入 A[i]
addi  t3  t3  4     # i=i+4
j     Loop
END:  lw  ra  0(t0)  # return A[0]
```

2.14.

```
bge   a1  a0, else  # if b>=a, 跳转
add   a2  a0  a1    # c=a+b
j     END
else: sub a2  a0  a1    # c=a-b
END:
```

2.15

```
sw    t0,  0(t0)    # p[0]=p
addi  t1   x0  3    # a=3
sw    t1,  4(t0)    # p[1]=a
slli  t2,  t1,  2   # a*4
sw    t1,  0(t2)    # p[a]=a
```

2.16

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

sw t2 , 0(t1) # *b = *a

ret

2.17

```
        addi a0,x0,0        #  a0 = 0
        addi a1,x0,1        #  a1 = 1
        addi a2,x0,30       #  a2 = 30
loop:   beq a0,a2,done      #  while( a0 != a2)
        slli a1,a1,1        #      a1 = 2a1
        addi a0,a0,1        #      a0 = a0 + 1.
        j loop
done:   # exit code          return
```

令 a1 左移30位. 得到 $a_1 = 2^{30}$.