3. (1) addi x0, x0, 0

(2) jr x1

(3) auipc ra, %pcrel_hi(offset)

jalr rd, %pcrel_lo(offset)(ra)

(4) add rd, rs, x0

(5) rdtime rd

(6) slli rd, rs, 16

srli rd, rd, 16

srai rd, rd, 16

7. (1) slti t3, t0, 0

slti t4, t1, 0

(2) add t0, t1, t2

sub t3, t0, t1

bne t3, t2, overflow

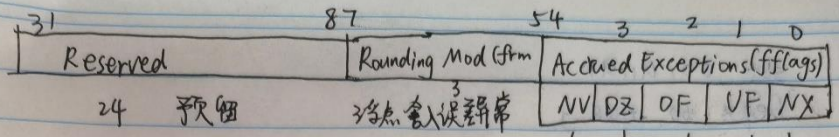
(3) x86&ARM: 加法指令的溢出标志位用于指示上一次加法操作是否发生溢出。标志位的值由处理器根据加法操作的结果计算得出，当结果超过一个有符号(ARM是无符号)整数的范围时，标志位被设置为1，否则为0，可以使用条件跳转指令根据标志位的值进行条件判断，从而实现检测加法溢出的操作

8. (1)

	DIVU	REMU	DIV	REM
rd	$2^{XLEN}-1$	x	-1	x

除数为0会引起RISCV打包出异常，程序在发出错误时可以进行处理

12)



24 预留　3浮点舍入误差异常

1 浮点无效异常　1 浮点上溢异常　1 浮点下溢异常　1 浮点除以0异常　1 浮点数操作异常

是。

13) X86：除数为0时，会异发"除0异常"，并将CPU状态的相应位设置为1，程序可以通过处理这个异常来避免程序崩溃。

ARM：除数为0时，结果将会是一个特殊值，取决于指令的类型和操作数的数据类型。在一些情况下，结果将会是无限大或无限小，而在其他情况下，结果将会是 NaN（不是数）。程序需要通过检查结果来判断是否发生了除以0的情况，并相应地处理。另外，ARM v8引入了一种"异常屏蔽"机制，可以用来防止除以0引起的异常

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

13.

```
add t0, zero, a0
add t1, zero, a1
add t2, zero, a2
addi t3, zero, 0
Loop: bge t3, 100, EndLoop
    slli t4, t3, 2  # 乘4, 1个int4字节
    add t5, t0, t4  # A[i]地址
    add t6, t1, t4  # B[i]地址
    lw a4, 0(t6)  # B[i]值
    lw a5, 0(t2)  # C值
    mul a5, a4, a5
    sw a5, 0(加t5)
    addi t3, t3, 1
    j Loop
EndLoop: lw a0, 0(t0)
    jr ra
```

14.
```
slt t0, a1, a0
beqz t0, else_j
add a2, a0, a1
else_j: sub a2, a0, a1
```

15.
```
add t2, zero, t0
sw t2, 0(t0)
addi t1, zero, 3
sw t1, 4(t0)
sll t3, t1, 2
add t3, t3, t0
sw t1, 0(t3)
```

16.
```
lw t2, 0(t0) # tomp = *a
lw t3, 0(t1) # t3 = *b
sw t3, 0(t0) # *a = t3
sw t2, 0(t1) * b = tmp
```

17. 计算 2^{30}