No. Date

3. (1) addi zero, zero, 0

(2) jalr x_0, x_1, 0

(3) auipc x_1, offset[31:12]

jalr x_1, rs, 0

(4) addi rd, rs, 0

(5) csrrs rd, cycle, x0

(6) addiw rd, rs, 0

7. (1) 填入 slti t_3, t_2, 0

slt t_4, t_0, t_1

首先比较 t_2 与 0 的大小

① 若 t_2 为负，则向 t_3 写入 1，此时加和结果应当~~不不~~不大于另一个加数 t_1，比较 t_0, t_1 大小

若 $t_0 < t_1$，未溢出 向 t_4 写入 1 $t_3 = t_4$ 不会跳转

否则结果溢出，向 t_4 写入 0 $t_3 \neq t_4$ 跳转到处理分支

② 若 t_2 大于等于 0，此时加和结果应当不小于另一个加数 t_1，向 t_3 写入 0

比较 t_0, t_1 大小

若 $t_0 < t_1$，溢出，向 t_4 写入 1 $t_3 \neq t_4$ 跳转到处理分支

否则未溢出，向 t_4 写入 0 $t_3 = t_4$ 不跳转

(2) ~~只需填入 bltu~~ 序列如下：

add t0, t1, t2

bltu t0, t1, overflow

两无符号数相加之和应当大于其中一个乘数，否则即溢出，需跳转。

(3) x86 用两个标志位 OF, CF 来指示是否发生溢出（OF为1表有符号溢出，CF为1表无符号溢出），以这两个标志位为依据决定是否跳转（如 jo overflow）。ARM 指令集也是通过设置标置位来指示的，具体如下

No. Date

ARM有几类加法指令如ADD/ADDS，会根据计算结果设置条件码和溢出标置，带有S后缀的将更新状态寄存器标置位；ADDS{cond}只更新条件码和溢出标置，不修改目的寄存器值，可根据标志位是否为1设置分支跳转来检测加法是否溢出

8. (1) DIVU REMU DIV REM

$2^{XLEN}-1$ x -1 x XLEN为寄存器位数。

除以0，结果的商的所有位被置为1（即无符号除法商为$2^{XLEN}-1$，有符号除法商为-1）；结果的余数被置为被除数

不会引起抛出异常，除0操作均被置有确定结果。当语言标准要求一个除零异常必须导致一个立即的控制流改变时只需在每个除法操作时增加一条分支指令置于其后。

若设计为抛出异常，会导致标准ISA中唯一的算术自陷，需要语言实现者另外设计与执行环境的自陷函数交互的部分，繁琐不便，如此设计提高了效率，保证ISA的简单化

(2) 各位： NV：非法操作 DZ：除以0 OF 上溢出

UF：下溢出 NX不精确。

不会，需要软件自身明确地对这些标置位进行检查。

(3) x86架构会触发异常，中断当前的程序执行，跳转到预定义的异常处理程序来处理当前程序。

~~ARM~~ 引发0号中断 #DE

在ARMv7-A中处理方法与x86类似，引发#DE异常，中断该程序并跳转异常处理

在ARMv8-A中会~~将异常~~产生未定义行为，即结果不确定，除零异常自己处理方式由未定义行为确定。

No. Date

12. (1) 管理员模式 (2) 机器模式 (3) 机器模式
(4) 管理员模式 (5) 用户模式

⑬ vecMul:

```
add  t3, zero, zero  #i
addi t4, zero, 100   #100次
loop:
bge  t3, t4, exit    #是否循环完
sll  t5, t3, 2       #偏移量.
add  a4, t0, t5      #&a[i]
add  a5, t1, t5      #&b[i]
lw   a6, 0(a5)
lw   a7, 0(t2)
mul  a6, a6, a7      #b[i]*c
sw   a6, 0(a4)
addi t3, t3, 1       #i++
j    loop
exit:
lw   a0, 0(t0)
ret.
```

⑭ begining:

```
blt  a1, a0, part
sub  a2, a0, a1
j    end
part:
add  a2, a0, a1
end:
add  a2, a2, zero
```

⑮

```
add  t2, t0, zero
sw   t2, 0(t0)    #p[0]=p.
addi t1, zero, 3  #a=3.
addi t3, t0, 4
sw   t1, 0(t3)    #p[1]=a.
sll  t4, t1, 2
sw   t1, 0(t4)    #p[0]=a.
```

⑯ swap:

```
lw  t2, 0(t0)
lw  t3, 0(t1)
sw  t3, 0(t0)
sw  t2, 0(t1)
jr  ra
```

17.

```
        addi  a0, x0, 0        # 将a0置为0
        addi  a1, x0, 1        # 将a1置为1
        addi  a2, x0, 30       # 将a2置为30（循环终止条件）.
loop:   beq   a0, a2, done     # 若a0=a2=30，跳转到done(退出)
        slli  a1, a1, 1        # 否则将a1的值左移1位，即乘2
        addi  a0, a0, 1        # a0值自增1
        j     loop             # 跳转回loop，执行循环
done:   # exit code            # 程序结束
```

T功能：将1循环左移30位，即计算2^{30}的值.