5. (1) nop → addi x0,x0,0

(2). ret → jalr x0,x1,0

(3). call offset → auipc x6, offset[32:12]

jalr x1, x6, offset[11:0]

(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) slti t3, t2, 0

slt t4, t0, t1

(2). bltu t0, t1, overflow

(3). X86架构：X86指令集中，加法指令有"ADD.ADC.ADDX等"，其中ADC指令表示"带进位加"，可用于处理有符号数与无符号数的加法，并检测加法是否溢出，如果进位标志CF被设置，则表示溢出

ARM架构：ARM指令集中，加法指令包括ADD与ADDS两种，其中ADDS表示"带标志位的加法"，它将同时执行加法和更新状态标志。如果结果溢出，那么状态标志中的V位将被设置为1

MIPS架构：指令集中包括ADD和ADDU两种，ADD用于有符号加法，ADDU用于无符号加法，如果溢出，那么对应标志将被设置

8、(1)

指令	rs1	rs2	op=DIVU rd	op=REMU rd	op=DIV rd	op=REM rd
op rd,rs1,rs2	x	0	抛出除零异常			

整型除法中除数为0会引起RISC-V抛出异常

采取抛出除零异常的设计是因为除以0是一个未定义的操作，没有意义，计算机中除0会导致程序崩溃或产出意外的结果，所以需要抛出除零异常，也可以使程序员更容易发现程序中的错误，提高代码的可靠性和调试效率，保证了计算结果的正确性

(2). NV：发生无效操作异常

DZ：浮点除法异常

OF：发生溢出异常

UF：发生下溢异常

NX：发生不精确异常

浮点运算发生异常，相应的flags位将被置位，但不会使处理器陷入系统调用。

(3). X86架构中，浮点除法除数为0时会引发除零异常，浮点控制寄存器中C0位设置为1，运算终止，程序转入异常处理例程。

ARM架构中，浮点除法除数为0时，引发浮点除零异常，FPSCR寄存器中的DN位与IOC位被设置为1，DN表示非数系异常，IOC表示操作异常，运算终止，程序转入异常处理例程。

MIPS架构中，浮点除法除数为0时，引发浮点除零异常，FPU控制寄存器FCSR中Cause字段的V位设置为1，运算中止，程序转入异常处理例程。

12. (1). Linux Kernel，第3级，机器模式

(2). Boot Rom，第3级，机器模式

(3). BootLoader，第1级，管理员模式

(4). USB Driver 第0级 用户模式

(5). vim 第0级 用户模式

13.

```
    .text
    .align 2
    .global vecMul
vecMul:
    li t3,0
vecMul_loop:
    li t8,100
    blt t3,t8, vecMul_loop_body
    lw t6,0(t0)
    jr ra
vecMul_loop_body:
    lw t4,0(t1)
    lw t8,0(t2)
    mul t7,t4,t8
    sw t7,0(t0)
    addi t0,t0,4
    addi t1,t1,4
    addi t3,t3,1
    j vecMul_loop
```

C 语言代码

```
int vecMul(int *A, int *B, int C){
    for(int i=0; i<100; ++i){
    A[i]=B[i]*C;
    }
    return A[0];
}
```

注：t0,t1分别为A,B的起始地址，t2为C的地址，t3用于计数

t4，t5分别存储B[i]，A[i]，t6：A[0] t7：B[i]*C

14. C程序代码：　　a、b、C对应寄存器a0, a1, a2

```
int a,b,c;
if(a>b){
  c= a+b;
}
else{
  c=a-b;
}
```

```
lw   a0, 0(sp)
lw   a1, 4(sp)
lw   a2, 8(sp)
bgt  a0, a1, then
sub  a2, a0, a1
  j  end
then :
  add a2, a0, a1
end:
```

15. 指针P已通过 int *P=(int*) malloc(4*sizeof(int))得到

P存于t0. a存于t1

P[0]=P ; int a =3;

P[1]=a ; P[a]=a

```
la   t0,P          # P→t0
li   t1,3          # a→t1
sw   t0,0(t0)      # P[0]=P
addi t2,t0,4
sw   t1,0(t2)      # P[1]=a
add  t3,t0,t1
sw   t1,0(t3)      # P[a]=a
```

1b. a→t0. b→t1

C:
```
void swap(int *a, int *b){
    int tmp = *a;
    a* = *b;
    *b = tmp;
    return;}
```

汇编:
```
swap:
    lw   t2,0(t0)      # t2=*a
    lw   t3,0(t1)      # t3=*b
    sw   t3,0(t0)      # *a=t3
    sw   t2,0(t1)      # *b=t2
    jr   ra
```

17：将 a0设置为0，a1设置为1，a2设置为30

代码中的循环体将a1不断左移位，a0不断加1直到 a0=30，循环30次，a1中最后存储的结果为 2^{30}，程序执行完后，跳转到done处结束程序