1. Intel X86-64 位CPU中，有哪些通用的64位寄存器？有哪些通用的32位寄存器？有哪些通用的16位寄存器？有哪些通用的8位寄存器？

答：64位通用寄存器有16个：rax,rbx,rcx,rdx,rbp,rsi,rdi,rsp,r8,r9,r10,r11,r12,r13,r14,r15

32位通用寄存器有16个：累加器eax，基址寄存器ebx，计数寄存器ecx，数据寄存器edx，栈指针寄存器esp，基址指针寄存器ebp，源变址寄存器esi，目标变址寄存器edi，r8d-r15d;

16位通用寄存器有16个：ax,bx,cx,dx,bp,si,di,sp,r8w-r15w

8位通用寄存器有16个:al,bl,cl,dl,bpl,sil,dil,spl,r8b-r15b； AH、BH、CH、DH 在一定条件下仍可使用。

1. 写出求长度为n（二进制位数，如8、16、32）的整数（带正负号）的补码表示的计算方法。

答：先不看正负号，直接将整数（相当于原数的绝对值）转换为二进制。若位数小于 n位，则将高位补 0，补足 n 个二进制位；若位数大于 n 位，则将高位舍去，只保留低的n个二进制位。   
 之后，再看正负号。若无符号，等同于正号。前面所求的结果即为该数的补码表示。若有负号，则把前面得到的二进制数逐位求反，每一位 0 变 1，1 变 0；再把该结果 加1，得到的数即为该数的补码表示。

1. 设 n＝16 （即字数据，相当于short），将下列带符号数用补码表示。

-69DAH -3E2DH 1AB6H -7231H 9876H

答：

[-69DAH]补 = 1001 0110 0010 0110 B = 9626H

[-3E2DH]补 = 1100 0001 1101 0011 B = C1D3H

[ 1AB6H]补 = 0001 1010 1011 0110 B = 1AB6H

[-7231H]补 = 1000 1101 1100 1111 B = 8DCFH

[ 9876H]补 = 1001 1000 0111 0110 B = 9876H

1. 简述在进行加法运算时，标志位 ZF、SF、CF、OF的设置规则。

答：ZF:运算结果为0时设置为1，否则设置为0；

SF:运算结果最高位二进制位为1时设置为1，否则设置为0；

CF:运算时从最高位向前产生进位设置为1，否则设置为0；

OF:两个加数相加前最高二进制位相同，且相加结果最高位与原加数最高位相反设置为1，否则设置为0；

1. 已知8位二进制数x1和x2的值，计算[x1]补＋[x2]补（结果还是8位，即一个字节），并指出每次加法运算后标志位SF、ZF、CF、OF的值。【建议采用16进制计算，不容易出错。SF是看结果的最高二进制位】

(1) x1＝＋0110011B； x2＝＋1011010B （x1=0x33, x2=0x5A）

设结果为x，x= 0x8D = 1000 1101B

则SF=1,ZF=0,CF=0,OF=1

(2) x1＝－0101001B； x2＝－1011101B （x1=-0x29, x2=-0x5D）

x= 0xD7+0xA3=0x7A = 01111010B

SF=0, ZF=0, CF=1, OF=1

(3) x1＝＋1100101B； x2＝－1011101B （x1=0x65, x2=-0x5D）

x=0x65+0xA3 =0x08=00001000

SF=0,ZF=0,CF=1,OF=0

1. 对于两个无符号数相加，用什么标志位判断产生了溢出？对于两个有符号数相加，用什么标志位判断产生了溢出？【注意，add 指令是不区分有符号加、还是无符号加的；但程序员可以编写语句，用不同的标志位来判断】

答：

对于无符号数相加，使用 CF来判断是否发生了溢出。如果 CF=1，则表示产生了进位，即发生了溢出。如果 CF=0，则表示没有产生进位，即没有发生溢出。

对于有符号数相加，使用 OF来判断是否发生了溢出。如果 OF=1，则表示发生了溢出。如果 OF=0，则表示没有发生溢出。

7、设有如下 C 语句段，运行后显示的结果是什么？并解释为什么？

short x = 0xFFFF;

short y =-1;

unsigned short z =0xFFFF;

printf(“%d %d %d\n”, x, y, z）；

答：显示结果为 -1 -1 65535

原因：对y进行分析，可以发现short类型的-1的补码为0xFFFF，实际上和x相同，也就是说其与x的输出结果会相同。在printf函数里使用的是%d，打印的x,y都是有符号类型，并且都为负数，因此高位全部补1，最后输出的结果补码为0xFFFFFFFF，也就是-1，因此x,y的输出结果是-1。而z是无符号整型，补位时高位都补0，最后输出结果为0x0000FFFF,也就是65535。