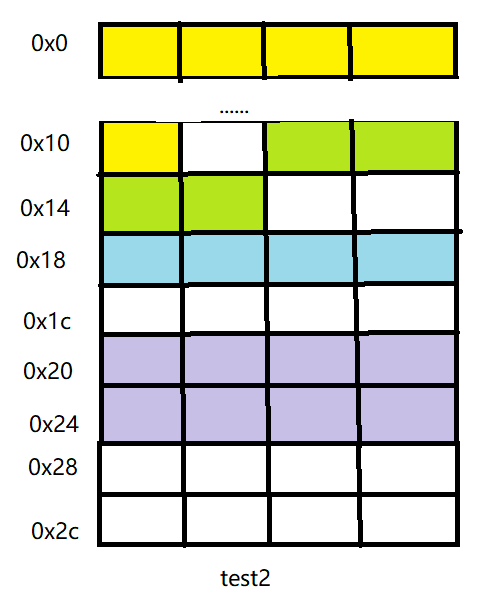
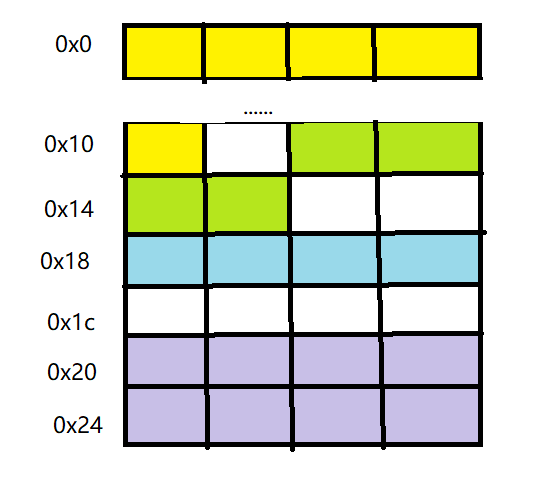
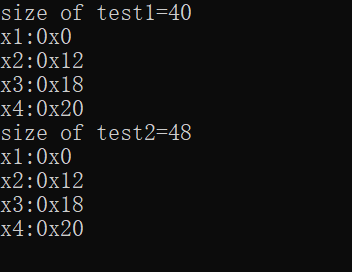
实验作业1

1.



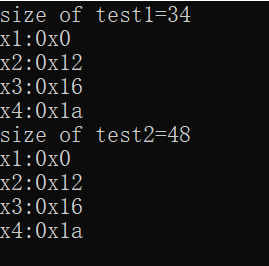
(自身对齐值是多少就必须从这个值的倍数的地方开始存放)

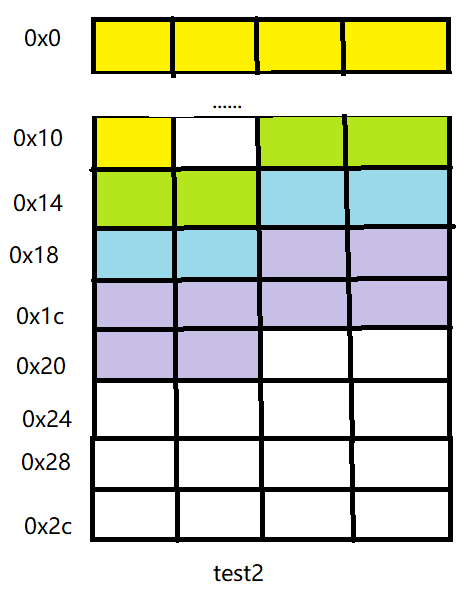
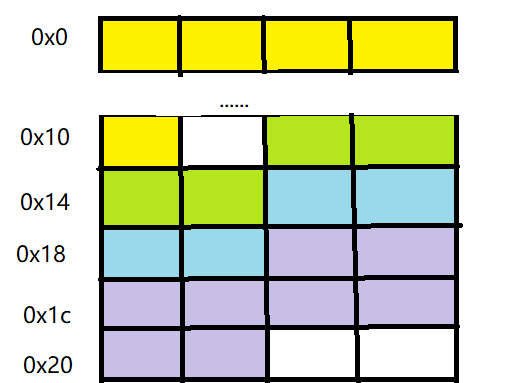


test1按照自然边界对齐，即char按1个字节，short按2个字节，int按4个字节，long long按8个字节。所以char x1[17]按1个字节对齐，占17个字节，从0x0到0x10,。short x2[2]按2个字节对齐，所以从0x12开始存，占4个字节。int x3按4个字节对齐，从0x18开始存。long long x4按8个字节对齐，从0x20开始存。一共占40个字节。

test2按照16字节对齐， 且占用空间也是16的整数倍，因此在存完long long 以后还要空出8个字节，保证size是16的倍数。

**#pragma pack(2)**



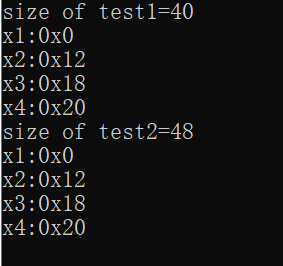


#pragma pack(2)表示当自然边界>2时，按2个字节对齐。

test1：char x1[17]按1个字节对齐，占17个字节，从0x0到0x10。short x2[2]按2个字节对齐，所以从0x12开始存，占4个字节。int x3按2个字节对齐，从0x16开始存。long long x4按2个字节对齐，从0x1a开始存。一共占34个字节。

test2与test1类似，但因为占用空间必须是16的整数倍，因此在存完long long 以后还要空出14个字节，保证size是16的倍数。

**#pragma pack(32)**



（存法如第一部分的图）

#pragma pack(32)表示当自然边界>32时，按32个字节对齐。

test1：char x1[17]按1个字节对齐，占17个字节，从0x0到0x10,。short x2[2]按2个字节对齐，所以从0x12开始存，占4个字节。int x3按4个字节对齐，从0x18开始存。long long x4按8个字节对齐，从0x20开始存。一共占40个字节。

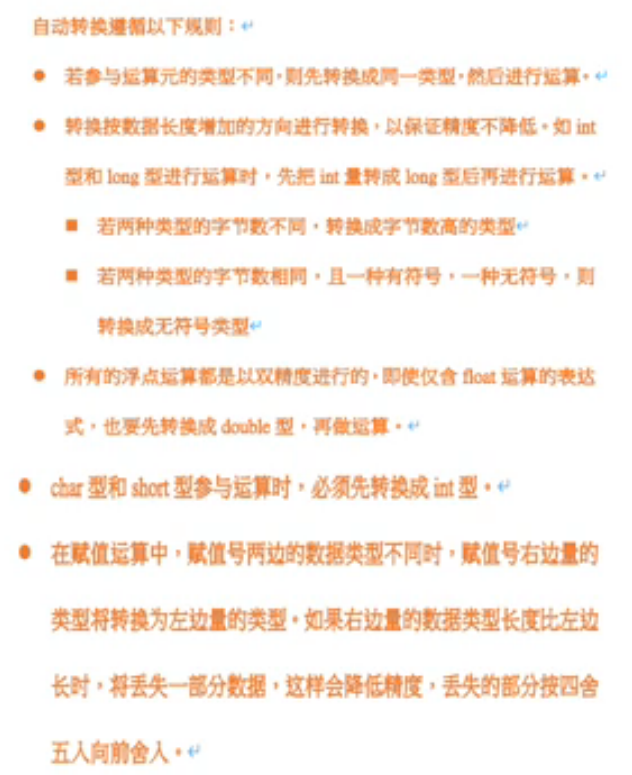
test2按照16字节对齐， 且占用空间也是16的整数倍，因此在存完long long 以后还要空出8个字节，保证size是16的倍数。

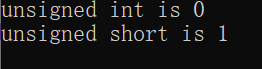
2.



结果不同。-5<5按照int型比较因此结果为1。-5<5u按照unsigned型比较，因此0xfffffffb>0x00000005,结果为0

3.

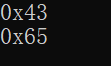


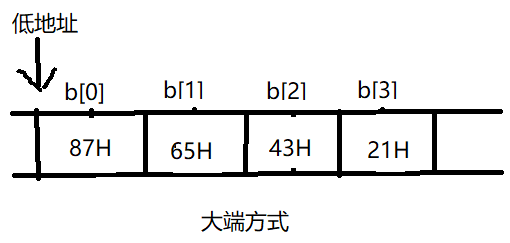
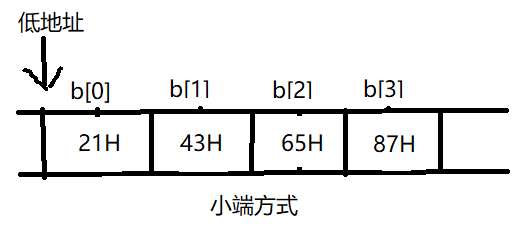


比较unsigned int和char时，按unsigned int比较，因此0x00000003<0xfffffffd,结果为0.

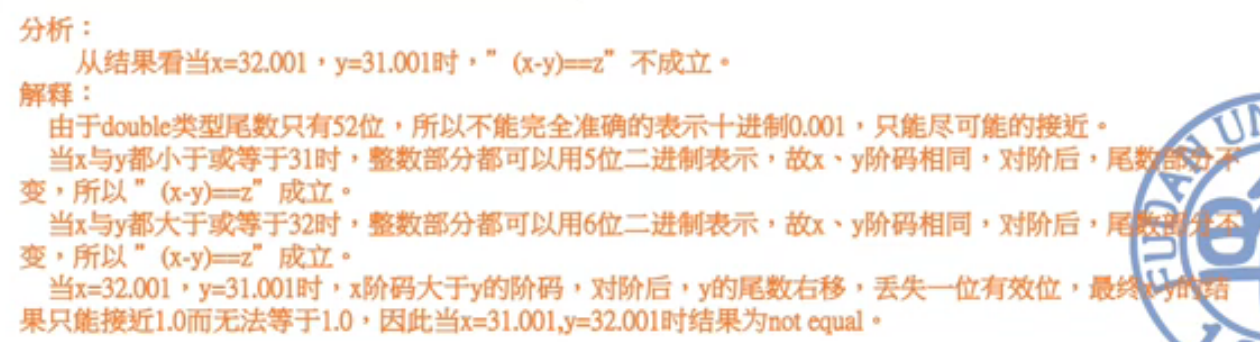
比较unsigned short和char时，按int比较，因此0x00000003>0xfffffffd,结果为0.

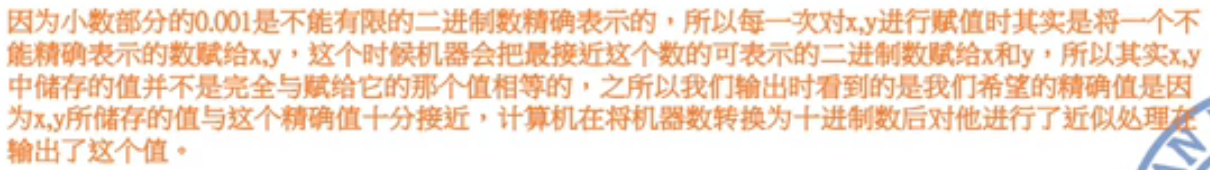
4.

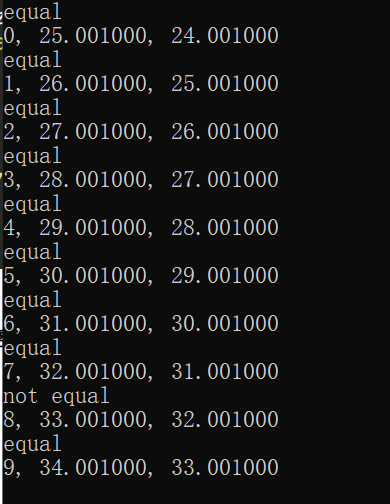


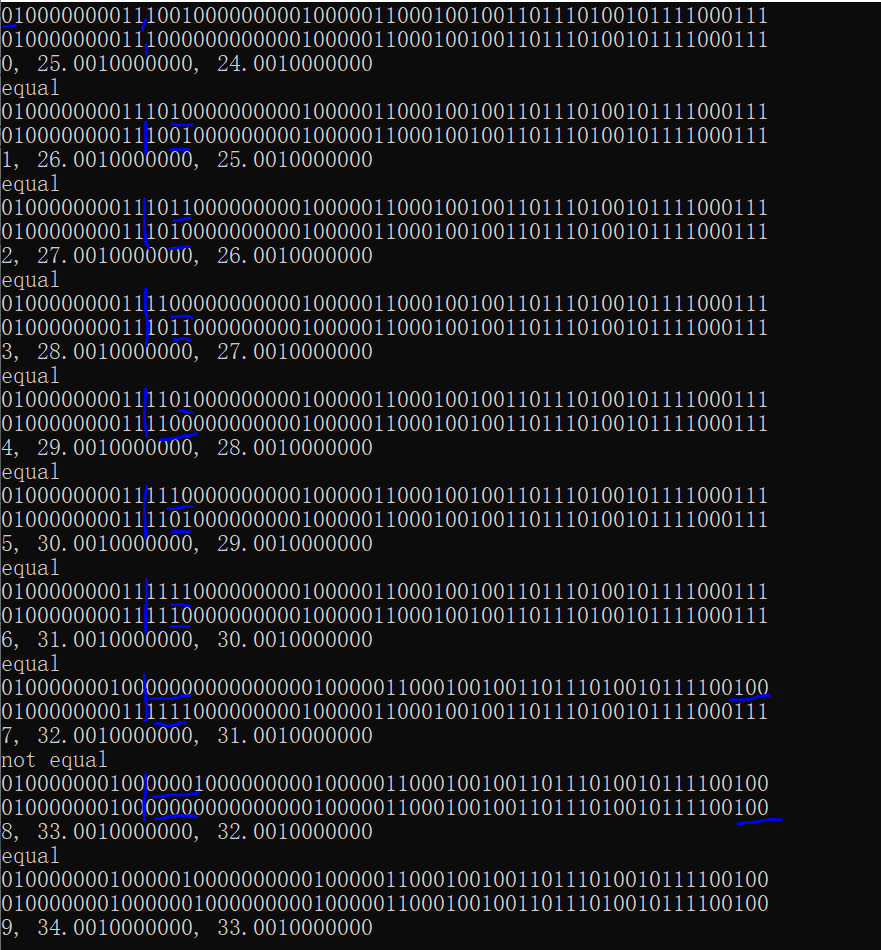


因为b[1]中存放0x43，因此是小端方式

5.







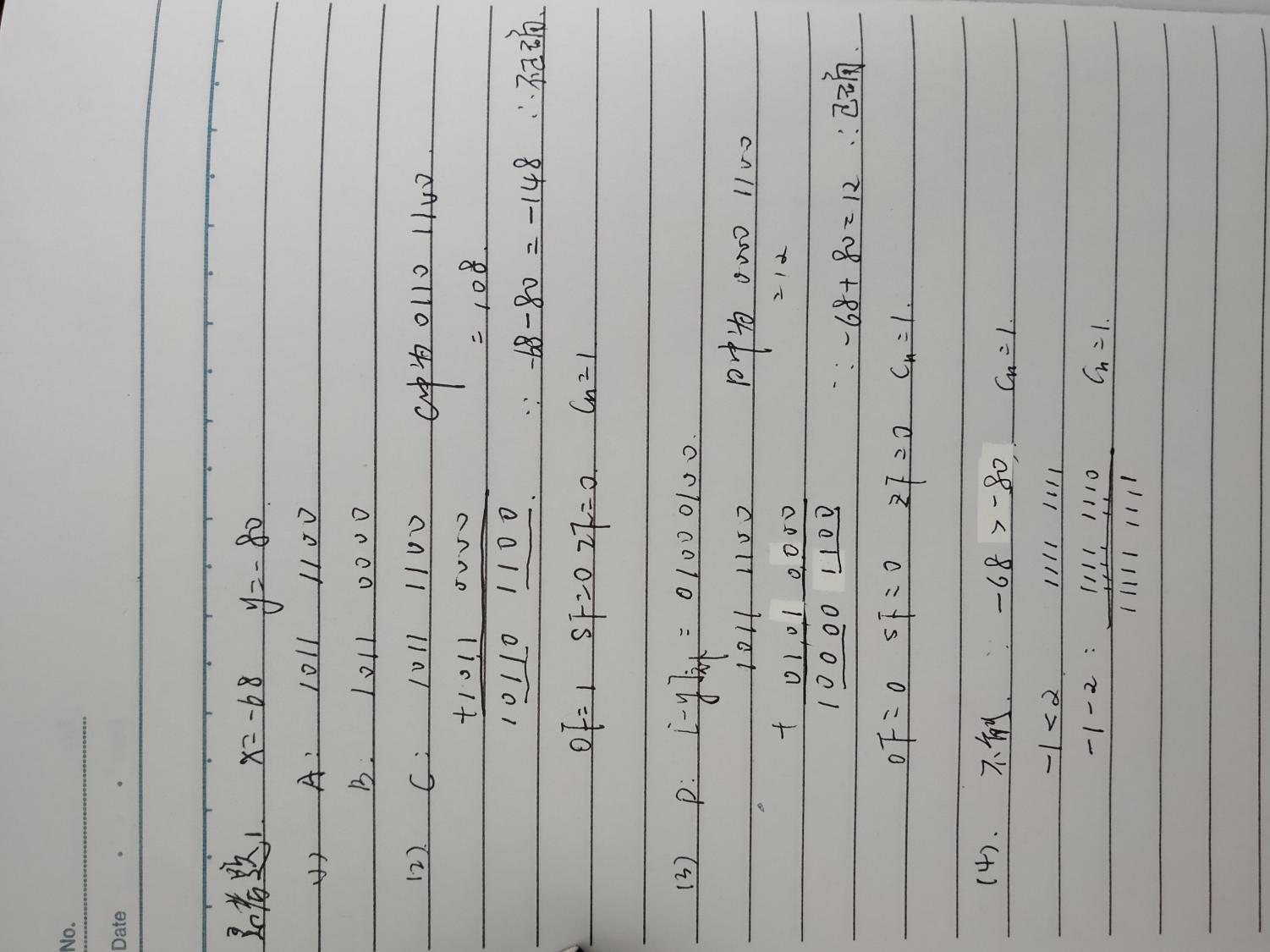
前几次都是equal，到第8次运算结果为not equal， 接下来又是equal

即当x=32.001,y=31.001时,(x-y)不等于1.0

根据输出的二进制结果可知x=32.001000是舍入后的结果，实际x的值为0x40400020c49ba5e4

由于浮点数相加时要对阶，1.0对阶后变成0.001\*2^3，31.001+1.0的时候发生右规，进行舍入的时候导致尾数发生变化，而y此时没有改变，因此得到not equal。到下一步时y也进行了同样的操作，导致x-y的结果又为1.

6.



7.