**c语言研究报告五**

1. **研究过程展示**

**1>.程序1部分：**

05demo1.c整体程序：

整体c程序：

char ch;

char \*p;

char \*\*pa;

char far \*pf;

int n;

main()

{

p = (unsigned char \*)0x1000;

ch = \*(unsigned char \*)0x1000 + \*p + \*(unsigned char far \*)0x200;

p = &ch;

\*p = \*p + 1;

pa = &p;

\*\*pa = \*\*pa + 1;

pf = (char far \*)&ch;

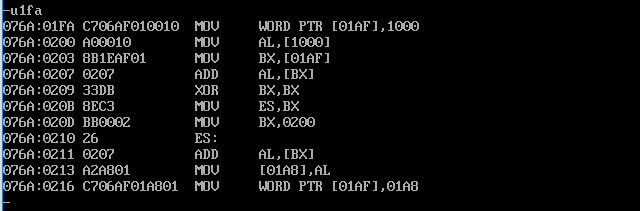
\*pf = \*pf + 1;

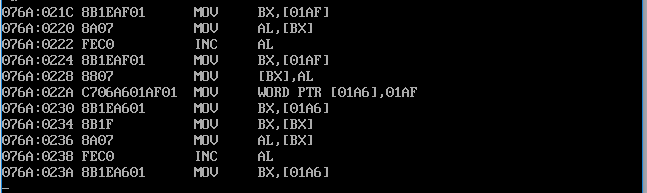
n = (int)&ch;

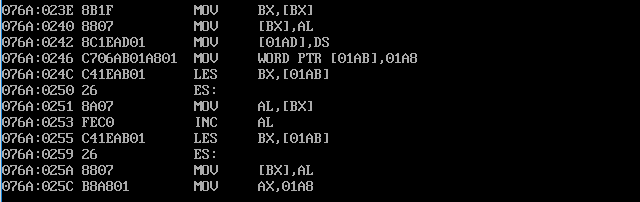
\*(char \*)n = \*(char \*)n + 1;

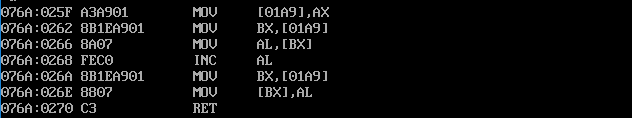
}

整体汇编程序：







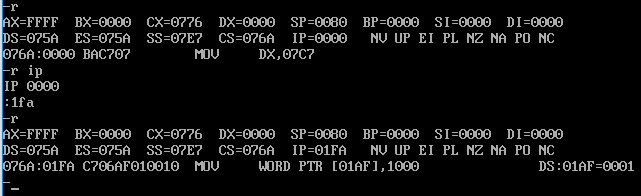


顺序执行并观察：

执行第一条C语句：

当前位置的信息，可以知道char \*变量p保存在起始地址为DS:01AFH、大小为两字节处，在这里是把1000H赋给一个偏移地址为01afH、大小为两字节的内存空间，p的当前数值是0001H。

指针的话，大小是固定的。有的指针包含显式偏移地址，有的指针包含显式段地址与偏移地址。



可以看到当前DS:01AFH内存处存放的数据是0001H，也就是程序运行开始的时候，p的数值是0001H，但是这条语句执行之后，p的数值应该是1000H了。

看一下效果：



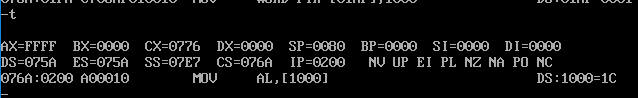
执行第二条C语句：

可以看到当前的DS:1000H内存处一个字节的内容是1CH。

比较一下前两条c程序中的语句：

第一句中：(unsigned char \*)0x1000

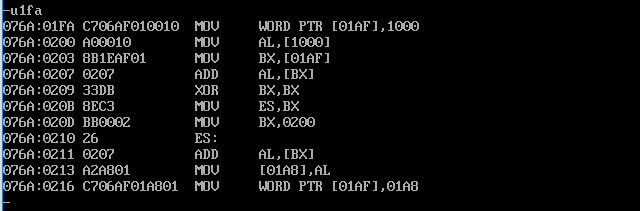
第二句中：\*(unsigned char \*)0x1000



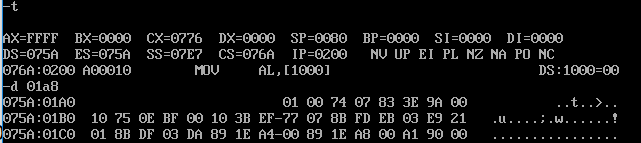
再对比一下汇编语句，第一句是针对1000H进行操作，而第二句则是针对[1000H]中的内容进行操作。也进一步从汇编角度证明此处的\*的作用。

ch = \*(unsigned char \*)0x1000 + \*p + \*(unsigned char far \*)0x200;

对应的汇编语句：



第二句刚开始的时候，DS:01A8H地址的第一个字节处存放数据为01H。



再分别观看下列内存中数据:

DS:1000H第一个字节处存放的数据是00H



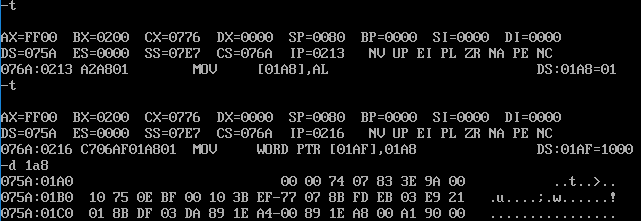
同样的，\*p应该也是00H

0：200H第一个字节处存放的数据是00H



所以，推测的ch的值是00H，即DS:01A8H地址的第一个字节处的数据是00H。

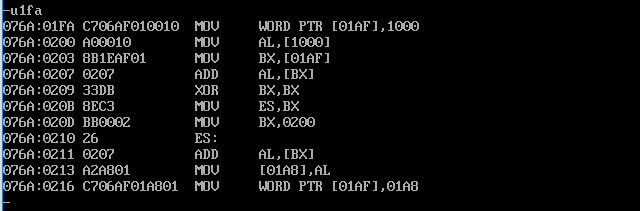
看一下：



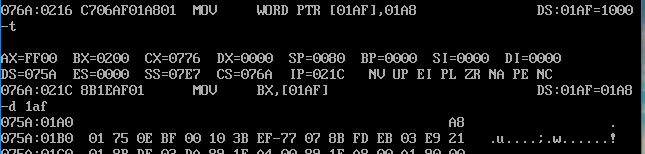
执行第三条C语句：

p = &ch;

对应的汇编语句：



查看DS:01AFH地址开始的两个字节内容：



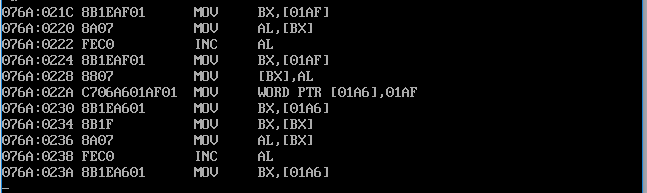
可见 ，p的值由1000H变为01A8H。

也就是说，指针p从原来指向DS:1000H改变为指向了DS:01A8H。

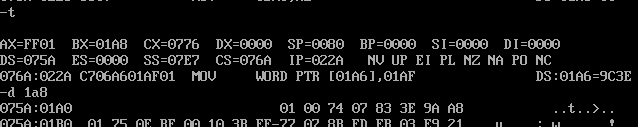
执行第四条C语句：

\*p = \*p + 1;

对应的汇编语句：



查看DS:01A8H执行之后是否由0变1：

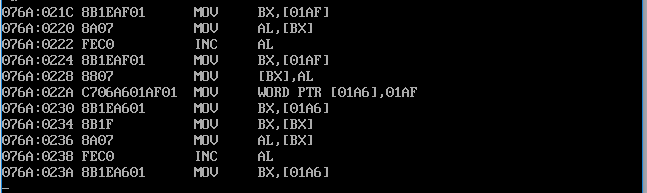


结论：是的。

执行第五条C语句：

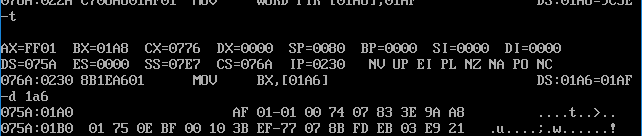
pa = &p;

对应的汇编语句：



也就是pa的地址是DS:01A6H，pa也是指针类型同样占据两个字节大小。pa的值是01AFH，即pa指向p。

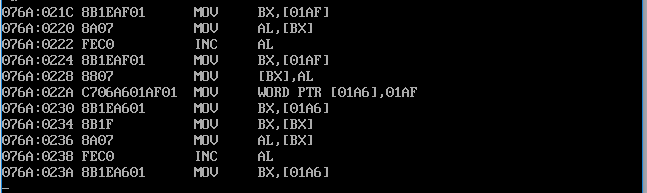
查看一下：

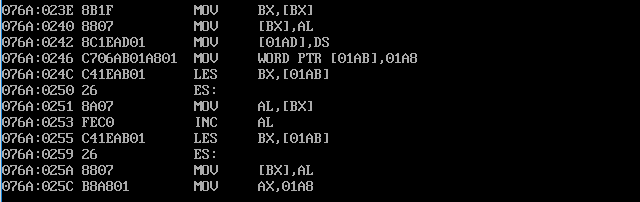


执行第六条C语句：

\*\*pa = \*\*pa + 1;

对应的汇编语句：



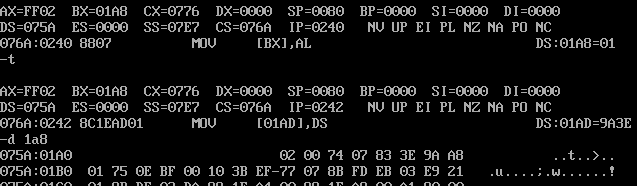


查看DS:01A8H执行之后是否由1变2：

前：



后：

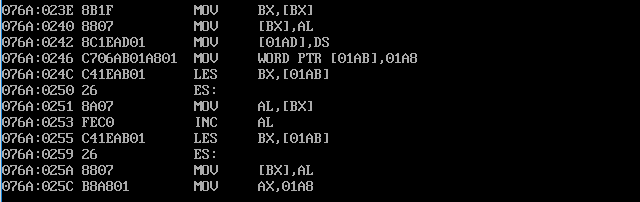


结论：成立。

执行第七条C语句：

pf = (char far \*)&ch;

对应的汇编语句：

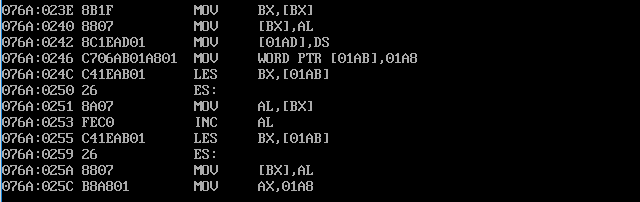


此处是将变量ch的地址的偏移地址、段地址依次存放入DS：01ABH开始处的四个字节内。

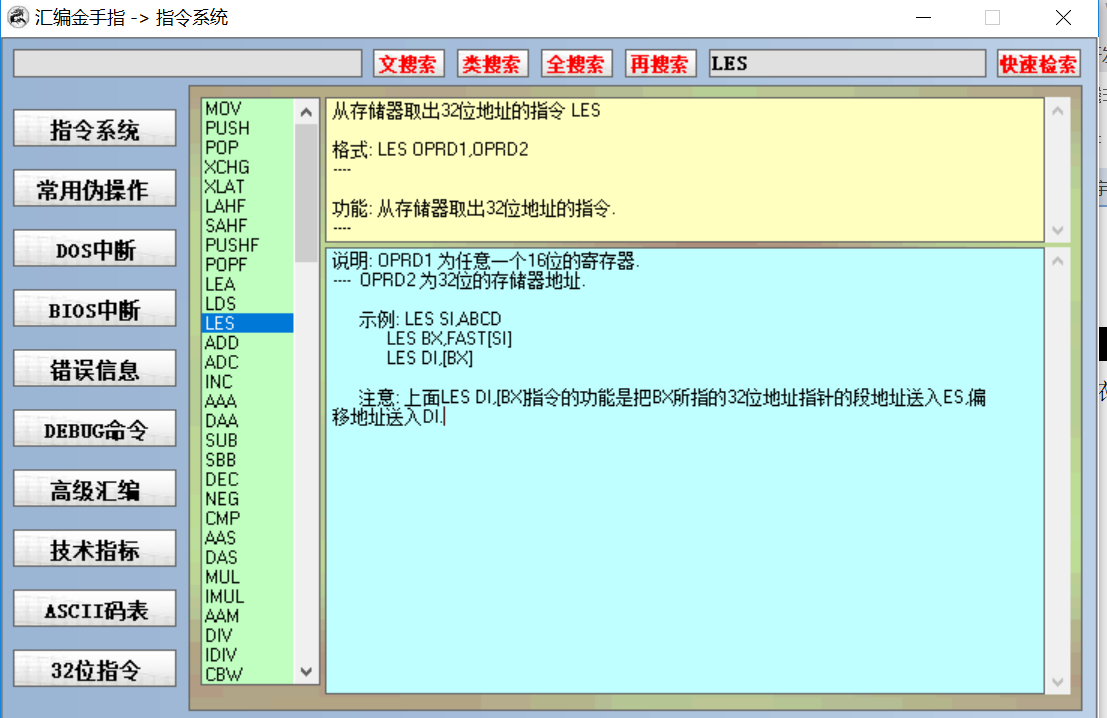
执行第八条C语句：

\*pf = \*pf + 1;

对应的汇编语句：



查询les作用：



查看DS:01A8H执行之后是否由2变3：

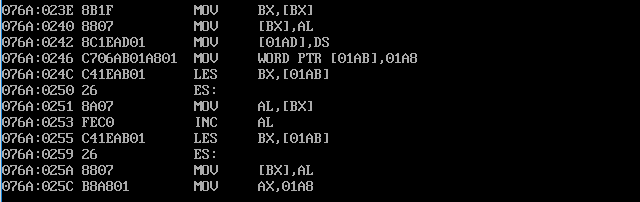


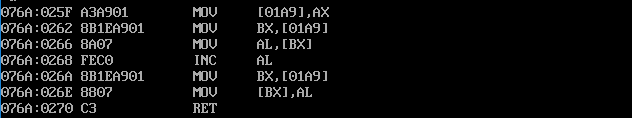
结论，成立。

执行第九条C语句：

n = (int)&ch;

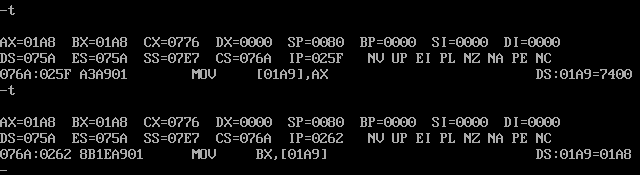
对应的汇编语句：





这两条语句是将变量ch的地址，所以此处是只是取得ch的偏移地址呢？还是取得ch的段地址和偏移地址呢？因为“mov ax,01a8H”只传递了两个字节，所以推测此处是只传递了偏移地址，加了（int）强制类型转换 ，目的是为了类型匹配，因为变量n的类型是int类型。

查看DS:01A9H地址处的两个字节内容：

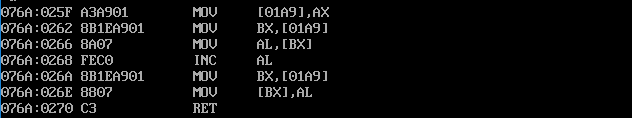


简单的赋值操作。

执行第十条C语句：

\*(char \*)n = \*(char \*)n + 1;

对应的汇编语句：



查看DS:01A8H执行之后是否由3变4：



结论：成立。

**2>.程序2部分：**

05demo2.c整体程序：

整体c程序：

typedef struct

{

int number;

char c;

char name[8];

}stu;

stu a;

char \*pchar;

int \*pint;

stu \*pstu;

main()

{

pstu = &a;

pstu->number = 1;

(\*pstu).c = 80;

pstu->name[0] = 'T';

pstu->name[1] = 'O';

(\*pstu).name[2] = 'm';

(\*pstu).name[3] = 0;

pchar = 0;

pint = 0;

pstu = 0;

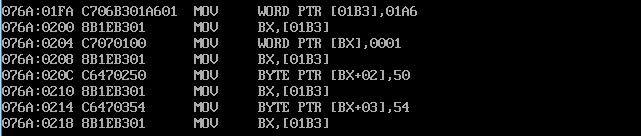
pchar = pchar + 1;

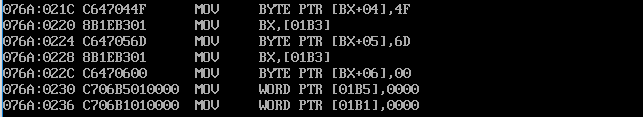
pint = pint + 1;

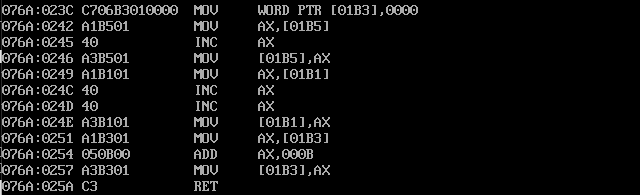
pstu = pstu + 1;

}

整体汇编程序：







在原程序中对结构体指针的使用有两种方法：pstu->number=1和(\*pstu).c=80,观察汇编程序可以发现这两种方式的结果是一样的。而bx的值始终为\*pstu的值，在调用结构体数据项的时候会以bx+数值的方式访问。

之后对char型、int型和stu型的指针+1进行比较。对char型指针，pchar+1执行了一次inc ax语句；对int型指针，pint+1执行了两次inc ax语句；而对stu型指针，pstu+1是执行了add ax，000b语句，也就是将ax的值加了11，而stu所占的内存空间正好是11个字节。所以这里指针+1不是单纯地将指针的值加一，而是将指针的值加一个指针指向的数据类型的大小，也就是指向的地址向后移动一个存储的数据大小。

**3>.程序3部分：**

05demo3.c整体程序：

整体c程序：

char \*p;

char far \*pf;

char str[20] = "hello world!";

int a[8] = {11,22,33,44,55,66,77,88};

int n;

main()

{

pf = (char far \*)0x200;

for(n = 0; str[n]; n++)

{

\*(pf + n) = str[n];

}

p = str;

pf = (char far \*)0x210;

for(n = 0;p[n];n++)

{

pf[n] = \*(str + n);

}

for(n = 0;n < 8;n++)

{

((int far\*)0x220)[n] = \*(a + n);

}

for(n = 0;n < 8;n++)

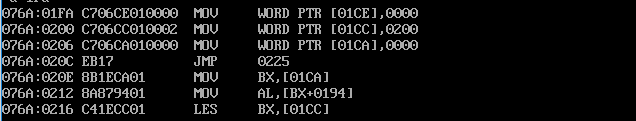
{

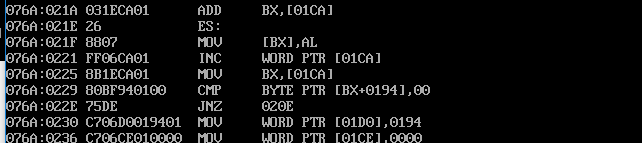
\*(int far\*)(0x230 + n\*2) = \*(&a[0] + n);

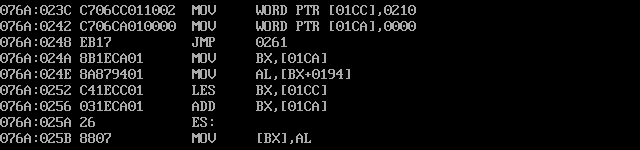
}

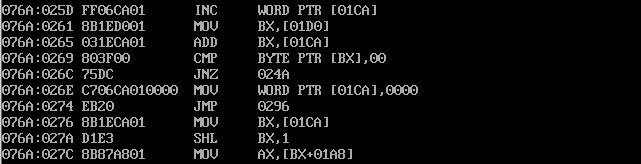
}

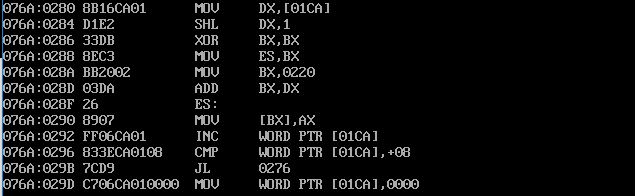
整体汇编程序：

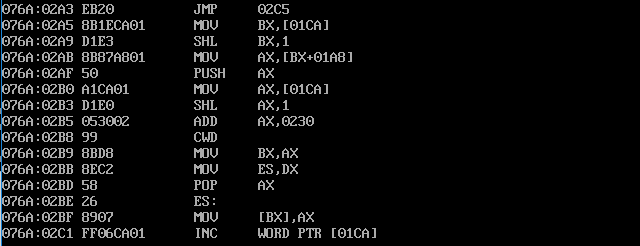














汇编程序是怎么实现for循环的：这里n的地址为01ca，开始现进行一次判断，cmp byte ptr [bx+194],0.194是str的偏移地址相当于str[0]，所以这里[bx+194]就相当于str[n]。如果str[n]不为0，则jnz会跳转到语句执行处，将str[n]赋给\*（pf+n），之后n加1再进行判断。这里pf的偏移地址为01cc。

之后看下一个for循环的语句pf[n]=\*（str+n）；发现它的汇编语句与\*（pf+n）=str[n]的汇编语句是一样的。也就是说pf[n]与\*（pf+n）是等价的。在这样是因为c语言里对指针和数组的操作就是对地址的操作，他们具有相同的性质。

那么（（int far \*）0x220）[n]就相当于（（int far \*）0x220+n\*2），这里n\*2因为（int far \*）0x220是一个地址而不是指针变量，而且这个地址的存储单位是int型即2个字节。

同样的观察汇编代码可以发现\*（a+n）等价于\*(&a[0]+n),即a与&a[0]是等价的。这是因为a[0]是数组的第一项，所以它的地址就是数组a的地址。

所以数组名就是一个指针，[]运算是在指针的值上加上一个偏移量，使其指向下一个存储单元的数据。

指针的话，我更想称其为地址类型变量。这样更符合我的理解。

1. **已思考研究并已解决问题汇总**
2. **已思考研究并未解决问题汇总**
3. **研究感想（心得体会）**

结合之前学习的汇编，计算机中最多的就是数字了。这些数字，在c语言中从不同的角度看，具有不同的使命。数据、指令、地址。但是三者之间又是相互联系的。啊呀，一拍脑门，这不又与计算机的组成中三大总线数据总线、控制总线、地址总线一一对应起来了吗？可以，这很指针。