**程序a1.c**

char ch; //定义char 型变量

char \*p; //定义char型指针变量，指针变量是专门用来存放地址的。

char \*\*pa;

char far \*pf;

int n; //定义int 型变量

main()

{

p=(unsigned char \*)0x1000;

/\* mov word ptr [01af],1000 在变量p的地址01af 处存放1000

且变量p是指针变量，存放的就是地址，即[01af]处存放偏移地址为1000的地址。\*/

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

/\*

1.(unsigned char \*)0x1000 将1000定义为指针地址，且这个地址放的是无符号char型数据，前面再加个\* 就是指向1000这个地址内的数据即：mov al,[1000]。

2. p是指针变量，专放地址的变量，通过上条指令可得知p变量放的地址为1000，我们的汇编代码先把1000给bx即：mov bx,[01af] \*p就相当于[bx],因为p内存放的地址也是放的是无符号char型数据，所以\*(unsigned char \*)0x1000+\*p对应为add al,[bx]

3. (unsigned char far \*)0x200 far表示这个0x200是由段地址和偏移地址组成即地址为:0x0000 0200 此地址是存放无符号char型数据的内存空间地址,前面加个\*，即指向这个地址的内存空间的数据。xor bx,bx mov es,bx 这两条指令是对es赋值为0，mov bx,0200 偏移地址赋值为0200 三个数据相加即:add al,es:[bx]。 对ch变量赋值 mov [01a8],al 变量ch地址为01a8。

\*/

p=&ch;

// 在变量p的地址01af 处存放ch变量地址01a8即：mov word ptr [01af],01a8

pa=&p;

/\*

在变量pa的地址01a6处存放p变量地址01af即:mov word ptr [01a6],01af 变量pa的偏移地址为01a6。

\*/

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

/\*

pa变量存放01af地址，\*pa 为指向01af的内存空间即:mov bx,[01a6] 对应bx=pa mov bx,[bx] 对应bx=\*pa,01af内存空间存放地址01a8，即\*\*pa为指向01a8的内存空间。mov al,[bx]对应为 al= \*\*pa inc al 对应为 al=al+1

\*\*pa对应汇编代码为: mov bx,[01a6] 、mov bx,[bx] 、[bx]

\*\*pa=al 即:mov [bx],al。

\*/

pf=(char far \*)&ch;

/\*

far表示取ch变量的段地址和偏移地址，将ch变量的地址强制转换为far型。此地址处存放char型数据即:mov [01ad],ds mov word ptr [01ab],01a8。即\*pf指向这个地址的内存空间，且这空间存放char型数据

\*/

\*pf=\*pf+1;

/\*

pf里放的是ch变量段地址和偏移地址，\*pf指向ch段地址：偏移地址处的内存空间。即指向ch 变量内存空间。les bx,[01ab],取偏移地址给bx,段地址给es。 mov al,es:[bx]将内存空间的数据放到al中， inc al 对应\*pf+1 。\*pf对应汇编代码les bx,[01ab]、es:[bx]

\*pf=al 即:mov es:[bx],al

\*/

n=(int)&ch;

/\*

(int)&ch 前面的(int)是指将ch变量的地址强制转换为int型 mov ax,01a8 mov [01a9],ax 这就是为什么不直接用mov word ptr[01a9],01a8 的原因。n变量地址为01a9。

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

/\* n变量放的是ch变量的地址，\*(char \*)n指向ch变量空间的char型数据然后将数据+1，最后赋值给指向的内存空间对应汇编代码为mov bx,[01a9] mov al,[bx] inc al mov bx,[01a9] mov [bx],al。

\*/

}

优先级顺序:

赋值运算符优先级最低 = &= ~=

逻辑运算符次低 与或非

比较运算符

加减乘除

！++ -- & \* ~

（） -> 最高。

char \*常量 即定义这个常量为地址 且这个地址放char 型数据 \*常量

char \*变量 即定义这个变量为指针变量，这个变量专放地址 \*变量

结论：指针基本上通过bx来进行寄存器间接寻址。

中间数据基本上存放在al,或ax寄存器中。

最前面的\*表示要访问的是一个内存空间

(\*)，括号里的\*是指明后面跟的是内存地址。

定义时，类型后面的\*表示定义指针变量。

**探究：**

对于\*操作，上面可知\*\*这种双重操作可以间接操作变量，是否可以定义多层次的指针？

我写了这样的代码；

char \*a;

char \*\*b;

char \*\*\*c;

char d;

char \*\*\*\*e;

char \*\*\*\*\*f;

main()

{

d=1;

a=&d;

b=&a;

c=&b;

e=&c;

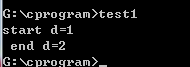
f=&e;

\*\*\*\*\*f=\*\*\*\*\*f+1;

printf("start d=1\n end d=");

printf("%d",d);

}

结果如下

操作成功，说明\*操作很可能没有层数限制，但是为了便于理解，我不建议这样使用。

而对于取地址符不能进行嵌套使用。

**程序b1.c**

typedef struct

{

int number;

char c;

char name[8];

}stu;

stu a;

char \*pchar; //指向char型的指针变量

int \*pint; //指向int型的指针变量

stu \*pstu; //指向结构体的指针变量

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; //将三个指针变量清零

/\*通过上面的定义可猜想 pcahr+1应该加一个字节大小

pint 应该加两个字节大小

pstu应该加12个字节的大小，本该11个，但是应该按2个字节对齐

\*/

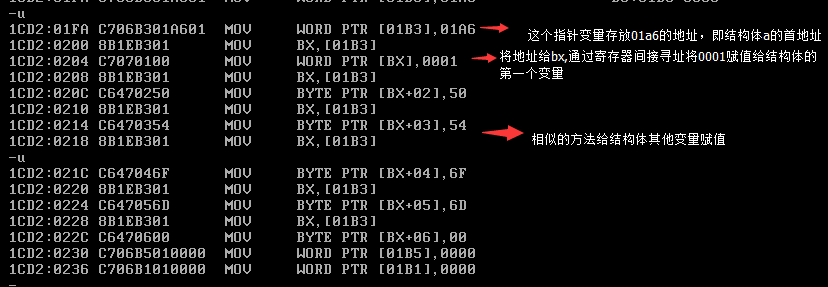
pchar=pchar+1;

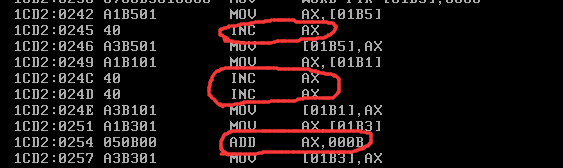
pint=pint+1;

pstu=pstu+1;

}

查看汇编代码





通过汇编代码发现pstu应该加12个字节的大小，但是事实上加了11个字节大小，其他都与猜想一致。难道全局结构体变量和局部结构体变量分配空间不同?

我加入了结构体变量，得到的新函数如下：

typedef struct

{

int number;

char c;

char name[8];

}stu;

stu a;

char \*pchar;

int \*pint;

long \*plong

stu \*pstu;

stu \*pstu1;

main()

{

stu b;

pstu1=&b;

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; //将三个指针变量清零

bplong=0;

pchar=pchar+1;

pint=pint+1;

plong=plong+1;

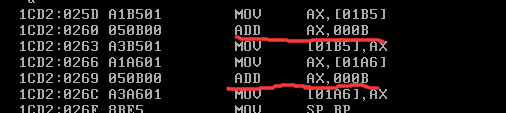
pstu=pstu+1;

pstu1=pstu1+1;

}

查看汇编代码





发现虽然分配了12空间，但是

pstu=pstu+1;

pstu1=pstu1+1;

都是 加了11个字节的大小。

所以 结构体指针变量+1 对应的是加结构体的空间大小。

对于long型指针变量



所以正确的结论为：

char型指针变量+1应该加一个字节大小

int型指针变量+1 应该加两个字节大小

long型指针变量+1应该加四个字节的大小

结构体指针变量+1应该加结构体实际大小

**探索**

对于指针变量可否不按类型匹配

做如下代码

char \*a;

long \*d;

int \*h;

int c;

char b;

long g;

main()

{

b=1;

c=1;

g=1;

a=&c;

d=&c;

h=&c;

printf("%c\t",\*a);

printf("%ld\t",\*d);

printf("%d\n",\*h);

a=&b;

d=&b;

h=&b;

printf("%c\t",\*a);

printf("%ld\t",\*d);

printf("%d\n",\*h);

a=&g;

d=&g;

h=&g;

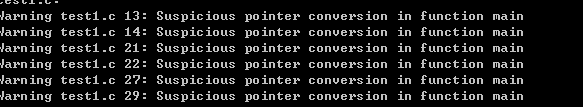
printf("%c\t",\*a);

printf("%ld\t",\*d);

printf("%d",\*h);

}

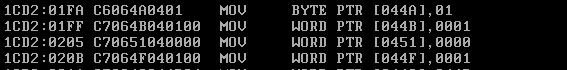
可以发现有一些警告，可以指针变换。但是可以编译链接。



执行后结果如下



72024065 -44b 0001 257-101 1241514241-4a00 0101



可得 b的首地址044a ,c的首地址044b d的首地址044f

查看内存很难得出 可能在调用printf中会改变044a附近的数据。

结论：对于匹配的指针，输出完全正确，对于大指针，也可以出现正确结果，对于小指针，如果值在小指针类型的范围内，可能正确显示（由占位符决定）。但是对于超过范围的值不能正确操作，

为了不出错，应该使用正确的指针匹配。

**程序c1.c**

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);

}

首先按自己的猜想理解这个程序

pf存放0：200地址，是char型的，pf+1等于0：201

\*pf 为指向0：200内存空间，则\*（pf+1）指向0：201内存空间

所以 for(n=0;str[n];n++) \*(pf+n)=str[n];

将”hello world!”拷贝到0：200起始地内存中

p=str;str是数组名本身就是数组的首地址 即指针变量p存放数组的首地址

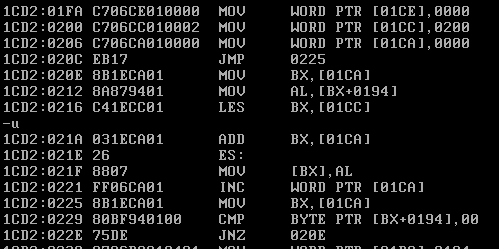
p[n]的意义等同于\*（p+n）

所以 for(n=0;p[n];n++) pf[n]=\*(str+n);

str本身就是地址，str+n就是地址+n 。 \*(str+n)就是指向地址str+n的内存空间

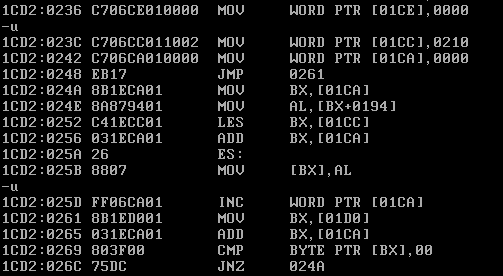
也可以翻译为 for(n=0;str[n];n++) \*(pf+n)=str[n];

查看汇编代码



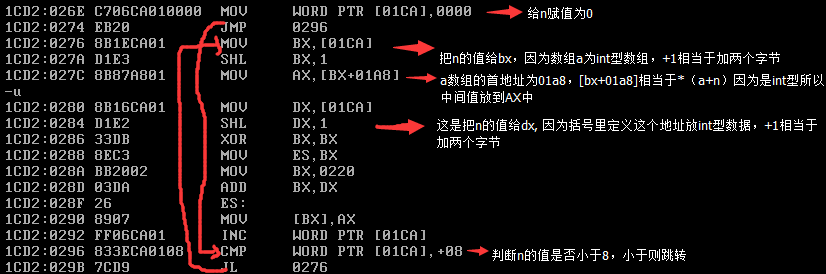
从图上可以看出 指针变量pf首地址01cc 低地址存放偏移地址，高地址存放段地址。 变量n的地址01ca 先给n赋值为0，然后跳转到0225处，判断[0194+n]内存空间是否为0,不是则跳转到020e处，str[n]对应的汇编代码为 先将n放到bx里 即mov bx,[01ca]然后加上数组的首地址得到中间值放在al中，str就像标号一样，就是代表地址且地址为0194。 mov al,[bx+0194],pf指针变量里存放的段地址和偏移地址先分别放到es和bx中，然后bx在加上n变量的值。对应的汇编代码：les bx,[01cc] add bx,[01ca]。最后将al所代表的中间值放到es:[bx]处，就是\*（pf+n）处即mov es:[bx],al。接下来将n的值加1，对应汇编代码为 inc word ptr [01ca]。01ca为变量n的地址。

所以 for(n=0;str[n];n++) \*(pf+n)=str[n];执行顺序为 1.先对n赋值，2.然后是判断str[n]是真还是假，是假则退出for循环，是真的话 3.执行后面这条指令，4.最后n=n+1。然后又到2 这样循环重复，直到2的条件为假退出。



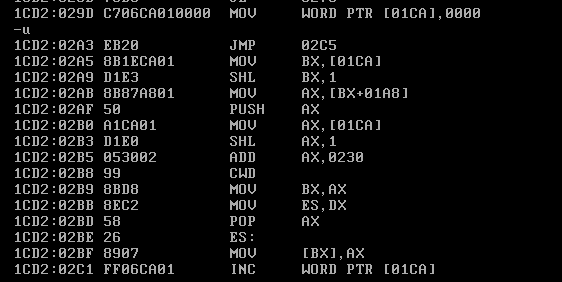
和上面图片的汇编代码基本一致。所以我的猜想是对的，是行得通的。

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);

查看上面c语言对应的汇编代码会发现和上面图片里基本上一样。





**计算机中内存都是有操作系统分配的，为什么非要提供far类型的指针？**

答：如果没有far类型指针，那么你的指针就不能灵活的执行任意段的某一偏移地址。

C字符串是当成字符串操作，而字符串是以\0结尾。但是如果定义这样的字符数组

Str[3]=”abc”；int a[8]={11,22,33,44,55,66,77,88}在内存中是这样



并没有\0这个空字符这是为什么？

因为Str[3]已经限制大小，且这个大小已经放满，系统没法自动补\0

对于str[]=’’abxasd”会为你自动分配大小

函数也是地址，可以定义指向函数的指针变量void (\*init)(u8); 函数指针、

void \*init(u8) 是函数指针。

指针和数组可以灵活使用u8 \*membase[SRAMBANK];

总结：指针可以像数组一样使用，数组也可以像指针一样使用。[]和你定义的类型一致，假如是 int型 [1]和[2]相差2个字节 long型 [1]和[2]相差4个字节。数组和指针唯一不同的是 数组的指向不可以改变，因为数组名就是常量地址，这个地址确定后就不可以改变。而指针可以随意改变首地址，你将数组a的首地址给指针变量p 则p就可以像数组a一样使用，当你你将数据b的首地址给指针变量p 则p就可以像数组b一样使用，但是数组a不能想指针一样得到数组b的首地址。所以指针更加灵活。