

# 在代码段中使用数据

贺利坚 主讲



汇编语言程序设计  
Assembly Language

# 问题：这样做是危险的！

🖥️例：将内存ffff:0~ffff:b中的数据拷贝到 0:200~0:20b单元中。

## 🖥️问题

- 📁程序中直接写地址，危险！
- 📁“安全”位置存放数据，存哪里？

## 🖥️对策

- 📁在程序的段中存放数据，运行时由操作系统分配空间。
- 📁段的类别：数据段、代码段、栈段
- 📁各种段中均可以有数据
- 📁可以在单个的段中安置，也可以将数据、代码、栈放入不同的段中。

```
1 ;使用附加段寄存器
2 assume cs:code
3 日code segment
4     mov ax,0ffffh
5     mov ds,ax
6     mov ax,0020h
7     mov es,ax
8
9     mov bx,0
10    mov cx,12
11
12 日    s: mov dl,[bx]
13        mov es:[bx],dl
14        inc bx
15        loop s
16
17        mov ax,4c00h
18        int 21h
19 code ends
20 end
```



	ffff:0	0:200
	ffff:1	0:201
	ffff:2	0:202
	ffff:3	0:203
	ffff:4	0:204
	ffff:5	0:205
	ffff:6	0:206
	ffff:7	0:207
	ffff:8	0:208
	ffff:9	0:209
	ffff:a	0:20a
	ffff:b	0:20b

# 应用案例

问题：编程计算以下8个数据的和，结果存在ax 寄存器中

0123H , 0456H , 0789H , 0abcH , 0defH , 0fedH , 0cbaH , 0987H

解决方案1

只要求数据本身，并未指定在哪些内存单元中！

在代码段中定义数据

dw: define word ,  
定义字型数据

dw 定义一个字  
db 定义一个字节  
dd 定义一个双字

```
1  assume cs:code
2  code segment
3  dw 0123H,0456H,0789H,0abcH,0defH,0fedH,0cbaH,0987H
4
5      mov bx,0
6      mov ax,0
7      mov cx,8
8
9  s:  add ax,cs:[bx]
10     add bx,2
11     loop s
12
13     mov ax,4c00h
14     int 21h
15 code ends
16 end
```

23	CS:0	数据
01	CS:1	
56	CS:2	
04	CS:3	
	...	数据
87	CS:e	代码
09	CS:f	
	CS:10	
	CS:11	
	...	

# 这个程序有问题！

```
1  assume cs:code
2  code segment
3      dw 0123H,0456H,0789H,0abcH,0defH,0fedH,0cbaH,0987H
4
5      mov bx,0
6      mov ax,0
7      mov cx,8
8
9  s:  add ax,cs:[bx]
10     add bx,2
11     loop s
12
13     mov ax,4c00h
14     int 21h
15 code ends
16 end
```

```
C:\>debug p6-1.exe
-r
AX=FFFF BX=0000 CX=0026 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0000  NU UP EI PL NZ NA PO NC
076A:0000 2301      AND     AX,[BX+DI]      DS:0000=20CD
-u
076A:0000 2301      AND     AX,[BX+DI]
076A:0002 56          PUSH    SI
076A:0003 0489      ADD     AL,89
076A:0005 07          POP     ES
076A:0006 BC0AEF    MOV     SP,EFOA
076A:0009 0DED0F    OR      AX,0FED
076A:000C BA0C87    MOV     DX,870C
076A:000F 09BB0000    OR      [BP+DI+0000],DI
076A:0013 B80000    MOV     AX,0000
076A:0016 B90800    MOV     CX,0008
076A:0019 07          JNB     [BX]
076A:001A 07          JNB     [X,+02]
076A:001B 019        JNB     019
```

真正的代码并不应该从0000开始

完全乱套的代码！

其实这一段本来就是数据+代码！

真正的代码，从0010开始

```
-d
076A:0000 23 01 56 04 89 07 BC 0A-EF 0D ED 0F BA 0C 87 09  #.U.....
076A:0010 BB 00 00 B8 00 00 B9 08-00 2E 03 07 83 C3 02 E2  .....
076A:0020 FB B8 00 00 CD 21 E8 9F-0E 83 C4 04 3D FF FF 74  ...L.!.....=.t
076A:0030 03      8B 56 FE 05 0C      ..../.P.F..U...
076A:0040 00      7B 0E 83 C4 04      .RP..H...P.f....
076A:0050 3D      26 BA 47 0C 2A      =..t.....^.&.G.*
```

从0000开始的是数据！

```
-u 0010
076A:0010 B80000      MOV     BX,0000
076A:0013 B80000      MOV     AX,0000
076A:0016 B90800      MOV     CX,0008
076A:0019 2E          CS:
076A:001A 0307      ADD     AX,[BX]
076A:001C 83C302      ADD     BX,+02
076A:001F E2F8      LOOP    0019
076A:0021 B8004C      MOV     AX,4C00
076A:0024 CD21      INT     21
```

解决问题的关键：让数据从cs:0000开始，让代码从cs:0010开始！

# 这样改进

🖥️问题：编程计算以下8个数据的和，结果存在ax 寄存器中

0123H , 0456H , 0789H , 0abcH , 0defH , 0fedH , 0cbaH , 0987H

🖥️解决方案2

```
1  assume cs:code
2  code segment
3      dw 0123H,0456H,0789H,0abcH,0defH,0fedH,0cbaH,0987H
4
5  start: mov bx,0
6         mov ax,0
7         mov cx,8
8
9  s: add ax,cs:
10     add bx,2
11     loop s
12
13     mov ax,
14     int 3h
15 code ends
16 end start
```

定义一个标号，指示代码开始的位置。

end的作用：除了通知编译器程序结束外，还可以通知编译器程序的入口在什么地方。

```
assume cs:code
code segment
:
: 数据
:
begin:
:
: 代码
:
:
code ends
end begin
```

程序的一般框架

🖥️效果：程序加载后，CS:IP指向要执行的第一条指令在start处！

# 改过的程序木有问题鸟！

```
1  assume cs:code
2  code segment
3      dw 0123H,0456H,0789H,0abcH,0defH,0fedH,0cbaH,0987H
4
5  start: mov bx,0
6          mov ax,0
7          mov cx,8
8
9      s: add ax,cs:[bx]
10         add bx,2
11         loop s
12
13         mov ax,4c00h
14         int 21h
15 code ends
16 end start
```

```
C:\>debug p6-2.exe
-r
AX=FFFF BX=0000 CX=0026 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0010  NU UP EI PL NZ NA PO NC
076A:0010 BB0000          MOV     BX,0000
-u
076A:0010 BB0000          MOV     BX,0000
076A:0013 BB0000          MOV     AX,0000
076A:0016 B90800          MOV     CX,0008
076A:0019 2E              CS:     ADD     AX,[BX]
076A:001A 0307              ADD     BX,+02
076A:001C 83C302              LOOP    0019
076A:001F E2F8              MOV     AX,4C00
076A:0021 B8004C          INT     21
076A:0024 CD21              CALL    0ECB
076A:0026 E89F0E          ADD     SP,+04
076A:0029 83C404          CMP     AX,FFFF
076A:002C 3DFFFF          JZ      0034
076A:002F 7403
```

真正的代码  
从0010开始

是这些代码！

```
-d cs:0
076A:0000 23 01 56 04 89 07 BC 0A-EF 0D ED OF BA 0C 87 09  #.U.....
076A:0010 BB 00 00 BB 00 00 B9 08-00 2E 03 07 83 C3 02 E2  ...L.!.....=.t
076A:0020 F8 B8 1C CD 21 E8 9F-0E 83 C4 04 3D FF FF 74  ....P.F..U...
076A:0030 0C 8B 56 FE 05 0C 8B 7B 0E 83 C4 04 3D FF FF 74  ...H...P.F..U...
```

从cs:0000处开始的  
依然是数据！



在代码段中使用数据