

实验2-包含多个段的程序

段的综述

- 我们可以将一段内存定义为一个段，用一个段地址指示段，用偏移地址访问段内的单元。这完全是我们自己的安排。
- 我们可以用一个段存放数据，将它定义为“数据段”；
- 我们可以用一个段存放代码，将它定义为“代码段”；
- 我们可以用一个段当作栈，将它定义为“栈段”；

段的综述（续）

- 我们可以这样安排，但若要让**CPU**按照我们的安排来访问这些段，就要：
 - 对于数据段，将它的段地址放在 **DS**中，用**mov**、**add**、**sub**等访问内存单元的指令时，**CPU**就将我们定义的数据段中的内容当作数据段来访问；

段的综述（续）

- 对于代码段，将它的段地址放在 **CS** 中，将段中第一条指令的偏移地址放在 **IP** 中，这样 **CPU** 就将执行我们定义的代码段中的指令；

段的综述（续）

- 对于栈段，将它的段地址放在**SS**中，将栈顶单元的偏移地址放在 **SP** 中，这样 **CPU**在需要进行栈操作的时候，比如执行 **push**、**pop** 指令等，就将我们定义的栈段当作栈空间来用。

第6章 包含多个段的程序

- 6.1 在代码段中使用数据
- 6.2 在代码段中使用栈
- 6.3 将数据、代码、栈放入不同的段

引言

- 我们将学习两种层次的使用方式：
 - （1）在一个段中存放数据、代码、栈，我们先来体会一下不使用多个段时的情况；
 - （2）将数据、代码、栈放入不同的段中。

6.1 在代码段中使用数据

- 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。
- 在前面的课程中，我们都是累加某些内存单元中的数据，并不关心数据本身。
- 可现在我们要累加的就是已经给定了数值的数据。

6.1 在代码段中使用数据

■ 程序6.1

```
assume cs:codesg
codesg segment
    dw 0123h,0456h,0789h,0abch,0defh,0fedh,0cbah,0987h
    mov bx,0
    mov ax,0
    mov cx,8
s: add ax,cs:[bx]
    add bx,2
    loop s
    mov ax,4c00h
    int 21h
codesg ends
end
```

程序第一行中的“dw”的含义是定义字型数据 dw即define word，我们使用dw定义了8个字型数据（数据之间以逗号分隔），它们所占的内存空间的大小为16个字节

6.1 在代码段中使用数据

- 这8个数据的偏移地址是多少呢？

因为用dw定义的数据处于代码段的最开始，所以偏移地址为0，这8个数据就在代码段的偏移0、2、4、6、8、A、C、E处。

程序运行时，它们的地址就是CS:0、CS:2、CS:4、CS:6、CS:8、CS:A、CS:C、CS:E。

6.1 在代码段中使用数据

- 我们将前面的**程序p61**编译、连接为可执行文件**p61.exe**，并用**debug**加载调试一下。
- 我们可以在源程序中指明程序的入口所在，具体做法见下面的程序**6.2**。

6.1 在代码段中使用数据

■ 程序6.2

```
assume cs:codesg
```

```
codesg segment
```

```
    dw 0123h,0456h,0789h,0abch,0defh,0fedh,0cbah,0987h
```

```
start: mov bx,0
```

```
        mov ax,0
```

```
        mov cx,8
```

```
    s: add ax,cs:[bx]
```

```
        add bx,2
```

```
        loop s
```

```
        mov ax,4c00h
```

```
        int 21h
```

```
codesg ends
```

```
end start
```

再次调试！

6.1 在代码段中使用数据

- 有了这种方法，我们就可以这样来安排程序的框架：

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

- 探讨**end**的作用：
end 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。

6.2 在代码段中使用栈

- 完成下面的程序，利用栈，将程序中定义的数据逆序存放。

```
assume cs:codesg
```

```
codesg segment
```

```
    dw 0123h,0456h,0789h,0abch,0defh,0fedh,0cbah,0987h
```

```
    ?
```

```
code ends
```

```
end
```

- 程序思路

6.2 在代码段中使用栈

- 程序的思路大致如下：
 - 程序运行时，定义的数据存放在`cs:0~cs:15`单元中，共8个字单元。依次将这8个字单元中的数据入栈，然后再依次出栈到这8个字单元中，从而实现数据的逆序存放。
 - 我们可以在程序中通过定义数据来取得一段空间，然后将这段空间当作栈空间来用。

6.2 在代码段中使用栈

■ 程序6.3

```
1  assume cs:codesg
2
3  codesg segment
4
5      dw 0123h,0456h,0789h,0abch,0defh,0fedh,0cbah,0987h
6      dw 0,0,0,0,0,0,0,0
7
8  start: mov ax,cs
9         mov ss,ax
10        mov sp,32
11
12        mov bx,0
13        mov cx,8
14        s: push cs:[bx]
15            add bx,2
16            loop s
17
18        mov bx,0
19        mov cx,8
20        s0: pop cs:[bx]
21            add bx,2
22            loop s0
23
24        mov ax,4c00h
25        int 21h
26
27 codesg ends
28 end start
```

■ 注意程序6.3中的指令：

mov ax,cs

mov ss,ax

mov sp,32

我们将 cs:16 ~ cs:31 的内存空间当作栈来用，初始状态下栈为空，所以 ss:sp 要指向栈底，则设置 ss:sp 指向 cs:32。

6.3 将数据、代码、栈放入不同的段

- 在前面的内容中，我们将数据、栈和代码都放到了一个段里面。

这样做显然有两个问题：

- （1）把它们放到一个段中使程序显得混乱；
- （2）前面程序中处理的数据很少，用到的栈空间也小，加上没有多长的代码，放到一个段里面没有问题。但如果数据、栈和代码需要的空间超过**64KB**，就不能放在一个段中。

6.3 将数据、代码、栈放入不同的段

■ 程序6.4

```
1  assume cs:code,ds:data,ss:stack
2
3  data segment
4      dw 0123h,0456h,0789h,0abch,0defh,0fedh,0cbah,0987h
5  data ends
6
7  stack segment
8      dw 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9
10 stack ends
11
12 code segment
13
14 start:mov ax,stack
15       mov ss,ax
16       mov sp,30h
17
18       mov ax,data
19       mov ds,ax
20
21       mov bx,0
22
23       mov cx,8
24 s:    push [bx]
25       add bx,2
26       loop s
27
28       mov bx,0
29
30       mov cx,8
31 s0:   pop [bx]
32       add bx,2
33       loop s0
34
35       mov ax,4c00h
36       int 21h
37
38 code ends
39
40 end start
```

- 我们在源程序中用伪指令“assume cs:code,ds:data,ss:stack”将cs、ds和ss分别和code、data、stack段相连。

- ✓ 标号“start”在“code”段中，这样CPU就将code段中的内容当作指令来执行了。
- ✓ 设置ss指向stack，设置ss:sp指向stack:16，CPU执行这些指令后，将把stack段当做栈空间来用。
- ✓ CPU若要访问data段中的数据，则可用ds指向data段，用其他的寄存器（如：bx）来存放data段中数据的偏移地址。

总结

■ 1、在代码段中使用数据

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

总结

■ 2、在代码段中使用栈

```
assume cs:codesg  
codesg segment
```

定义数据
定义一段空间为栈

start: 设置SS:SP的值指向栈

代码

```
mov ax,4c00h  
int 21h
```

```
codesg ends  
end start
```

总结

■ 3、将数据、代码、栈放入不同的段

assume cs:code,ds:data,ss:stack

data segment

数据

data ends

stack segment

栈

stack ends

code segment

start: 代码

codesg ends

end start

关于段前缀

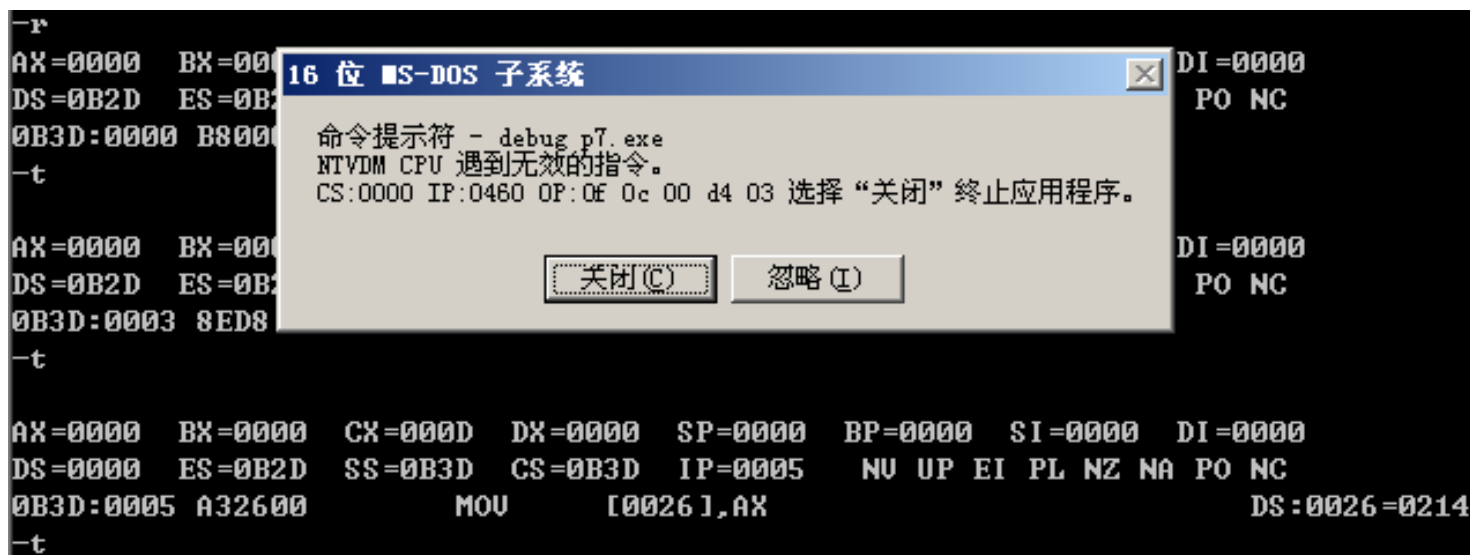
- 指令“`mov ax,[bx]`”中，内存单元的偏移地址由**bx**给出，而段地址默认在**ds**中。
- 我们可以在访问内存单元的指令中显式地给出内存单元的段地址所在的段寄存器。
- 例如：
 - `mov ax,ds:[bx]`
 - `mov ax,es:[bx]`

段前缀

- 这些出现在访问内存单元的指令中，用于显式地指明内存单元的段地址的“**ds:**”、“**cs:**”、“**ss:**”或“**es:**”，在汇编语言中称为段前缀。

一段安全的空间

- 在8086模式中，随意向一段内存空间写入内容是很危险的，因为这段空间中可能存放着重要的系统数据或代码。



The screenshot shows a DOS debug window with a memory dump and an error message. The memory dump displays registers (AX, BX, CX, DX, SP, BP, SI, DI) and memory locations (DS:0000, ES:0000, 00B3D:0000, 00B3D:0003, 00B3D:0005) with their respective values. An error message box titled "16 位 MS-DOS 子系统" (16-bit MS-DOS subsystem) is overlaid on the dump, stating: "命令提示符 - debug p7.exe", "NTVDM CPU 遇到无效的指令。" (NTVDM CPU encountered an invalid instruction), and "CS:0000 IP:0460 OP:0f 0c 00 d4 03 选择“关闭”终止应用程序。" (Choose "Close" to terminate the application). The error box has two buttons: "关闭(C)" (Close) and "忽略(I)" (Ignore).

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0B2D ES=0B2D SS=0B3D CS=0B3D IP=0005 NU UP EI PL NZ NA PO NC
00B3D:0000 B80000 MOV     [0026],AX
-t
AX=0000 BX=0000 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0B2D ES=0B2D SS=0B3D CS=0B3D IP=0005 NU UP EI PL NZ NA PO NC
00B3D:0003 8ED8 MOV     [0026],AX
-t
AX=0000 BX=0000 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0000 ES=0B2D SS=0B3D CS=0B3D IP=0005 NU UP EI PL NZ NA PO NC
00B3D:0005 A32600 MOV     [0026],AX
-t
```


一段安全的空间

- 我们在纯**DOS**方式（实模式）下，可以不理睬**DOS**，直接用汇编语言去操作真实的硬件，因为运行在**CPU**实模式下的**DOS**，没有能力对硬件系统进行全面、严格地管理。

一段安全的空间

- 但在Windows 2000、UNIX这些运行于CPU保护模式下的操作系统中，不理睬操作系统，用汇编语言去操作真实的硬件，是根本不可能的。硬件已被这些操作系统利用CPU保护模式所提供的功能全面而严格地管理了。

一段安全的空间

- 在一般的PC机中，DOS方式下，DOS和其他合法的程序一般都不会使用0:200~0:2FF（0:200h~0:2FFh）的256个字节的空間。所以，我們使用这段空間是安全的。

实验二