第5章 [BX]和loop指令

第5章 [bx]和loop指令

- 5.1 [bx]
- 5.2 Loop指令
- 5.3 在Debug中跟踪用loop指令实现的循环程序
- 5.4 Debug和汇编编译器Masm对指令的不同处理
- 5.5 loop和[bx]的联合应用
- 5.6 段前缀
- 5.7 一段安全的空间
- 5.8 段前缀的使用

[bx]和内存单元的描述

- [bx]是什么呢？

由以前学习过的[address]内存格式可以猜到吗？

[bx]和内存单元的描述

- 完整描述一个内存单元，需要：
  - 地址（段地址，偏移地）
  - 类型（长度）

例：[0]表示一个内存单元时，

- 0 表示单元的偏移地址，段地址默认在ds中
- 单元的长度（类型）由具体指令中的其他操作对象（比如说寄存器）指出。

[bx]和内存单元的描述

- [bx] 表示一个内存单元，
  - **偏移地址**在bx中，
  - **段地址**在ds中
  - **单位**（长度）由另外一个操作数指出

  - mov ax,[bx]

  - mov al,[bx]

loop

- loop ——循环

描述性符号“()”

- 为了描述方便，符号“()”来表示一个寄存器或一个内存单元中的内容。

描述性符号“()”

- (X)的应用，如：

	描述的含义
(ax)=0010H	ax中的内容为0010H
(21000H)=0010H	(21000H)=0010H
(ax)=((ds)*16+2)	mov ax,[2]
((ds)*16+2)=(ax)	mov [2],ax
(ax)=(ax)+2	add ax,2
(ax)=(ax)+(bx)	add ax,bx

描述性符号“()”

- (X)的应用，如：

	描述的含义
(sp) = (sp)-2 ((ss)*16＋(sp))=(ax)	push ax
(ax)=((ss)*16+(sp)) (sp)=(sp)+2	pop ax

约定符号idata表示常量

- 指令

mov ax,[0]

表示将 ds:0 处的数据送入ax中。其中，“[...]”内常量0表示内存单元的偏移地址。

以后，我们用idata表示常量。

约定符号idata表示常量

- 比如：
  - mov ax, [idata]
    - mov ax,[1]、mov ax,[2]、mov ax,[3]等。
  - mov bx, idata
    - mov bx,l、mov bx,2、mov bx,3等。
  - mov ds, idata
    - mov ds, 1、mov ds, 2等，均为非法指令
    - 不能向ds写常量

5.1 [bx]

- 分析下面指令的功能：
  - mov ax,[bx]

    功能：bx 中存放的数据作为偏移地址EA，段地址SA 默认在ds 中，将SA:EA处的数据送入ax中。

    即： (ax)=((ds) *16 +(bx));

5.1 [bx]

- mov [bx],ax

  功能：bx中存放的数据作为偏移地址EA，段地址SA默认在ds中，将ax中的数据送入内存SA:EA处。

  即：((ds) *16 +(bx)) = (ax)。

5.1 [bx]

问题5.1

程序和内存中的情况如下图所示，写出程序执行后，21000H~21007H 单元中的内容。

```
mov ax,2000H
mov ds,ax
mov bx,1000H
mov ax,[bx]
inc bx
inc bx
mov [bx],ax
inc bx
inc bx
mov [bx],ax
inc bx
mov [bx],al
inc bx
mov [bx],al
```

内存中的情况

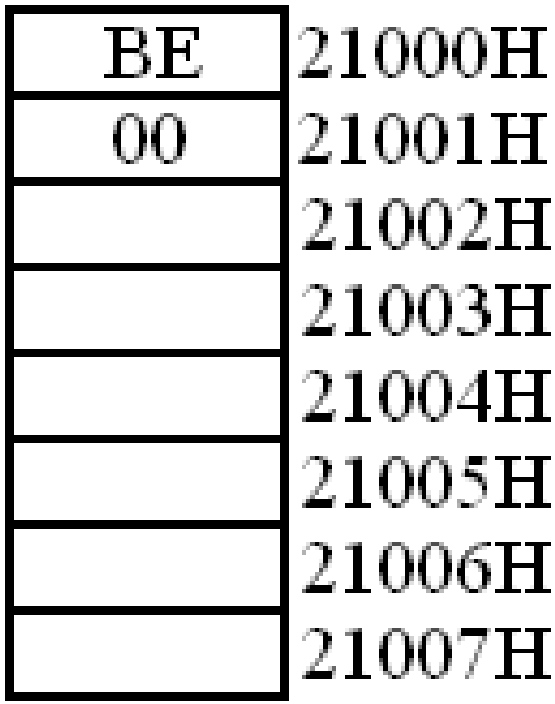
BE	21000H
00	21001H
	21002H
	21003H
	21004H
	21005H
	21006H
	21007H

图 5.1　问题 5.1 程序和内存情况

5.1 [bx]

- 问题5.1分析
  - （1）先看一下程序的前三条指令：

```
mov ax,2000H
mov ds,ax
mov bx,1000H
```



这三条指令执行后

ds = 2000H

bx = 1000H

5.1 [bx]

BE	21000H
00	21001H
	21002H
	21003H
	21004H
	21005H
	21006H
	21007H

- 问题5.1分析（续）
  - （2）再看第4条指令：

    mov ax,[bx]

指令执行前：

ds =2000H，bx=1000H，指令将把内存2000:1000处的字型数据送入ax中。

该指令执行后： ax=00beH

5.1 [bx]

BE	21000H
00	21001H
	21002H
	21003H
	21004H
	21005H
	21006H
	21007H

- 问题5.1分析（续）
  - （3）再看第5、6条指令：

    inc bx

    inc bx

指令执行前：

bx=1000H。

执行后：

bx=0002H。

5.1 [bx]

BE	21000H
00	21001H
BE	21002H
00	21003H
	21004H
	21005H
	21006H
	21007H

- 问题5.1分析（续）
  - （4）再看第7条指令：

mov [bx],ax

指令执行前：

ds=2000H，bx=1002H，此指令将ax中的数据 送入内存2000:1002处。

指令执行后：

2000:1002单元的内容为BE，2000:1003单元的内容为00。

5.1 [bx]

BE	21000H
00	21001H
BE	21002H
00	21003H
	21004H
	21005H
	21006H
	21007H

- 问题5.1分析（续）
  - （5）再看第8、9条指令：

    inc bx

    inc bx

指令执行前：

bx = 1002H

执行后：

bx = 0004H

5.1 [bx]

BE	21000H
00	21001H
BE	21002H
00	21003H
BE	21004H
00	21005H
	21006H
	21007H

- 问题5.1分析（续）
  - （6）再看第10条指令：

    mov [bx], ax

指令执行前：

ds=2000H，bx=1004H，此指令将ax中的数据 送入内存2000:1004处。

指令执行后：

2000:1004单元的内容为BE，2000:1005单元的内容为00。

5.1 [bx]

BE	21000H
00	21001H
BE	21002H
00	21003H
BE	21004H
00	21005H
	21006H
	21007H

- 问题5.1分析（续）
  - （7）再看第11条指令：

    inc bx

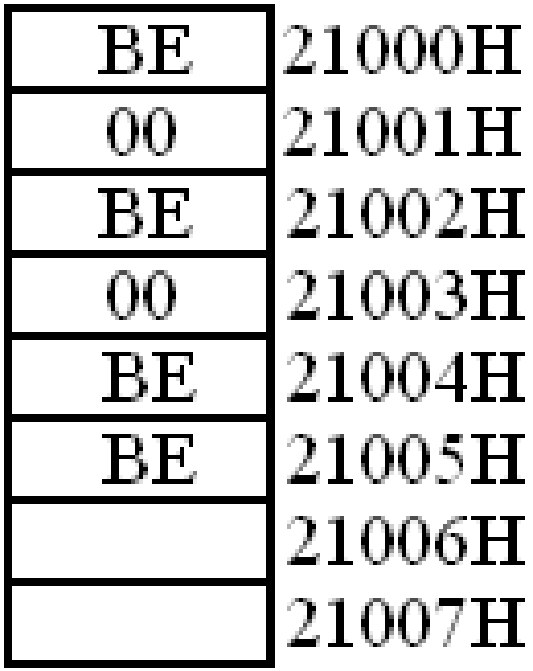
    指令执行前：

    bx=1004H。

    执行后：

    bx=1005H。

5.1 [bx]



- 问题5.1分析（续）
  - （8）再看第12条指令：

    mov [bx],al

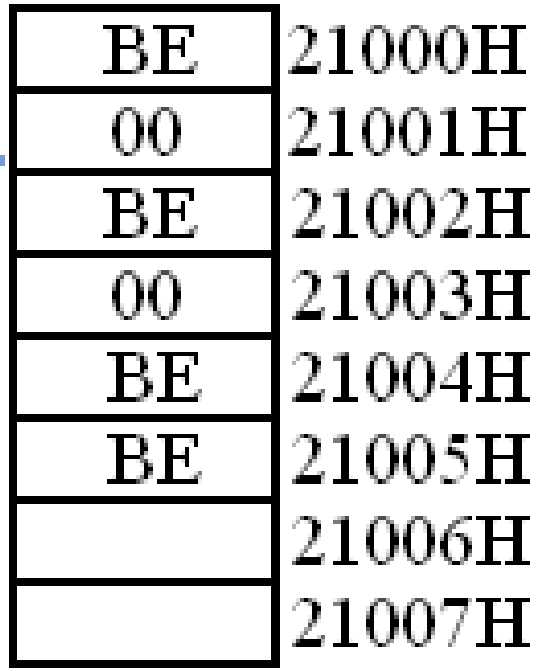
指令执行前：

ds=2000H，bx=1005H，此指令将al中的数据送入内存2000:1005处。

指令执行后：

2000:1005单元的内容为BE。

5.1 [bx]



- 问题5.1分析（续）
  - （9）接下来是第13条指令：

inc bx

指令执行前：

bx=1005H，

指令执行后：

bx=0006H。

5.1 [bx]

BE	21000H
00	21001H
BE	21002H
00	21003H
BE	21004H
BE	21005H
BE	21006H
	21007H

- 问题5.1分析（续）
  - （10）再看第12条指令：

    mov [bx],al

指令执行前：

ds=2000H，bx=1006H，此指令将al中的数据送入内存2000:1006处。

指令执行后：

2000:1006单元的内容为BE。

5.2 Loop指令

- 指令的格式：

  loop 标号

- CPU 执行loop指令的时候，要进行两步操作：

  1. (cx)=(cx)-1；
  2. 判断cx中的值，不为零则转至标号处执行程序，如果为零则向下执行。

- 通常（注意，我们说的是通常）用loop指令来实现循环功能，cx 中存放循环次数。

5.2 Loop指令

- loop指令应用举例：
  - 任务1：编程计算2^2，结果存放在ax中。
  - 任务2：编程计算2^3。
  - 任务3：编程计算2^12。

5.2 Loop指令

- 任务1：编程计算2^2，结果存放在ax中。

  分析：

  设(ax)=2，可计算：(ax)= (ax)*2，最后(ax)中为2^2的值。N*2可用N+N 实现。

5.2 Loop指令

- 任务1：编程计算2^2，结果存放在ax中。

程序代码：

```
assume cs:code

code segment
    mov ax,2
    add ax,ax

    mov ax,4c00h
    int 21h
code ends

end
```

5.2 Loop指令

- 任务2：编程计算2^3。

  分析：

  2^3=2*2*2，若设(ax)=2，可计算：(ax)=(ax)*2*2，最后(ax)中为2^3的值。N*2可用N+N 实现。

5.2 Loop指令

- 任务2：编程计算2^3。

  程序代码：

```
assume cs:code

code segment
    mov ax,2
    add ax,ax
    add ax,ax

    mov ax,4c00h
    int 21h

code ends

end
```

5.2 Loop指令

- 任务3：编程计算2^12。

  分析：

  2^12=2*2*2*2*2*2*2*2*2*2*2*2，若设(ax)=2，可计算：

  (ax)= (ax)*2*2*2*2*2*2*2*2*2*2*2，最后(ax)中为2^12的值。N*2可用N+N 实现。

5.2 Loop指令

- 任务3：编程计算2^12。

程序代码：

```
assume cs:code
code segment
    mov ax,2
    ;做11次add ax,ax
    mov ax,4c00h
    int 21h
code ends
end
```

这需要11条重复的指令add ax, ax

——可用loop来简化程序

5.2 Loop指令

- 任务3：编程计算2^12。

  程序代码：

```
assume cs:code

code segment
      mov ax,2
      mov cx,11
 s:  add ax,ax
      loop s

      mov ax,4c00h
      int 21h
code ends

end
```

5.2 Loop指令

- 任务3：编程计算2^12。

程序代码：

```
assume cs:code
code segment
    mov ax,2
    mov cx,11
 s: add ax,ax
    loop s

    mov ax,4c00h
    int 21h
code ends
end
```

程序分析：

（1）标号s标号代表一个地址

5.2 Loop指令

- 任务3：编程计算2^12。

程序代码：

```
assume cs:code
code segment
    mov ax,2
    mov cx,11
 s: add ax,ax
    loop s

    mov ax,4c00h
    int 21h
code ends
end
```

程序分析：

（1）标号s标号代表一个地址

（2）CPU 执行loop s的时候，要进行两步操作：

① (cx)=(cx)-1；

② 判断cx 中的值，不为0 则转至标号s 处执行，否则执行下一条指令。

5.2 Loop指令

- 任务3：编程计算2^12。

程序代码：

```
assume cs:code
code segment
    mov ax,2
    mov cx,11
s:  add ax,ax
    loop s

    mov ax,4c00h
    int 21h
code ends
end
```

程序分析：

（1）标号s标号代表一个地址

（2）CPU 执行loop s的时候，要进行两步操作：

① (cx)=(cx)-1；

② 判断cx 中的值，不为0 则转至标号s 处执行，否则执行下一条指令。

（3）根据以上，可利用cx来控制loop的循环次数。

5.2 Loop指令

- 下面详细分析程序的执行过程，体会如何用cx和loops 相配合实现循环功能。

5.2 Loop指令

- 程序的执行过程：

```
    mov cx,11
s:  add ax,ax
    loop s
```

循环执行了多少遍？

5.2 Loop指令

- 程序的执行过程：

```
    mov cx,11
s:  add ax,ax
    loop s
```

（1）执行mov cx,11，设置(cx)=11；

（2）执行add ax,ax（第1次）；

（3）执行loop s 将(cx)减1，(cx)=10，(cx)不为0，所以转至s处；

（4）执行add ax,ax（第2次）；

（5）执行loop s 将(cx)减1，(cx)=9，(cx)不为0，所以转至s处；

5.2 Loop指令

- 程序的执行过程：

```
    mov cx,11
s:  add ax,ax
    loop s
```

（6）执行add ax,ax（第3次）；

（7）执行loop s 将(cx)减1，(cx)=8，(cx)不为0，所以转至s处；

（8）执行add ax,ax（第4次）；

（9）执行loop s 将(cx)减1，(cx)=7，(cx)不为0，所以转至s处；

5.2 Loop指令

- 程序的执行过程：

```
    mov cx,11
s:  add ax,ax
    loop s
```

（10）执行add ax,ax（第5次）；

（11）执行loop s 将(cx)减1，(cx)=6，(cx)不为0，所以转至s处；

（12）执行add ax,ax（第6次）；

（13）执行loop s 将(cx)减1，(cx)=5，(cx)不为0，所以转至s处；

5.2 Loop指令

- 程序的执行过程：

```
    mov cx,11
s:  add ax,ax
    loop s
```

（14）执行add ax,ax（第7次）；

（15）执行loop s 将(cx)减1，(cx)=4，(cx)不为0，所以转至s处；

（16）执行add ax,ax（第8次）；

（17）执行loop s 将(cx)减1，(cx)=3，(cx)不为0，所以转至s处；

5.2 Loop指令

- 程序的执行过程：

```
    mov cx,11
s:  add ax,ax
    loop s
```

（18）执行add ax,ax（第9次）；

（19）执行loop s 将(cx)减1，(cx)=2，(cx)不为0，所以转至s处；

（20）执行add ax,ax（第10次）；

（21）执行loop s 将(cx)减1，(cx)=1，(cx)不为0，所以转至s处；

5.2 Loop指令

- 程序的执行过程：

```
    mov cx,11
s: add ax,ax
    loop s
```

（22）执行add ax,ax（第11次）；

（23）执行loop s 将(cx)减1，(cx)=0，(cx)为0，所以向下执行。

（结束循环）

5.2 Loop指令

- cx和loop 指令相配合实现循环功能的三个要点：

  1. 在cx中存放循环次数；

  2. loop 指令中的标号所标识地址要在前面；

  3. 要循环执行的程序段，要写在标号和loop指令的中间。

```
      mov cx,循环次数
s:    循环执行的程序段
      …
      loop s
```

5.2 Loop指令

- 问题5.2
  - 用加法计算123 × 236，结果存在ax 中。

- 思考后看分析。
  - 分析:
    可用循环完成，将123加236次。可先设(ax)=0，然后循环做236次(ax)=(ax)+123。

  - 程序代码

5.2 Loop指令

- 问题5.2程序代码

```
assume cs:code
code segment
    mov ax,0
    mov cx,236
s:add ax,123
    loop s

    mov ax,4c00h
    int 21h
code ends
end
```

5.2 Loop指令

问题5.3

改进问题5.2程序，提高123×236 的计算速度。

- 思考后看分析。
  - 分析：
    可将236 加123次。可先设(ax)=0，然后循环做123次(ax)=(ax)+236，这样只需用123 次加法，少于问题5.2程序的236 次加法。
  - 程序代码请自行实现。

5.3 在Debug中跟踪loop循环

- 问题：计算ffff:0006单元中的数乘以3，结果存储在dx中。

- 分析：
  - （1）运算后的结果是否会超出dx的存储范围？

    ffff:0006 单元中的数是一个字节型的数据，范围在0~255之间，则用它和3相乘结果不会大于65535，可以在dx 中存放下。

5.3 在Debug中跟踪loop循环

- 分析：
  - （2）我们用循环累加来实现乘法，用哪个寄存器进行累加？
    我们将ffff:0006单元中的数赋值给ax，用dx进行累加。先设(dx)=0，然后做3次(dx)=(dx)+(ax)。

5.3 在Debug中跟踪loop循环

- 分析：
  - （3） ffff:0006单元是字节单元，ax是一个16 位寄存器，数据长度不一样，如何赋值？

要实现ffff:0006单元向ax 赋值，课令(ah)=0，(al)=(ffff6H)。

- 设ffff:0006单元中的数据是XXH，若要ax中的值和ffff:0006单元中的相等，ax中的数据应为00XXH。
- 例如：8 位数据01H和16位数据0001H的数据长度不一样，但它们的值是相等的。

5.3 在Debug中跟踪loop循环

- 实现计算ffff:0006单元中的数乘以3，结果存储在dx中的程序代码。

```
assume cs:code
code segment
        mov ax,0ffffh
        mov ds,ax
        mov bx,6
        mov al,[bx]
        mov ah,0
        mov dx,0
        mov cx,3
s:      add dx,ax
        loop s

        mov ax,4c00h
        int 21h
code ends
end
```

汇编源程序中，数据不能以字母开头，要在前面加0。

5.3 在Debug中跟踪loop循环

- 可对程序的执行过程进行跟踪，进一步了解loop实现的原理。
  - 先编辑为源程序文件并保存为 p3.asm ；
  - 对其进行编译连接后生成p3.exe;
  - 然后再用Debug对p3.exe中的程序进行跟踪。

- 跟踪详细过程参阅课本。

5.3 在Debug中跟踪loop循环

- 更改程序，实现：计算 ffff:0006单元中的数乘以 123，结果存储在dx中。

- 更改程序：只要将寻循环次数改为123。
- 更改后程序的跟踪过程如下：

5.3 在Debug中跟踪loop循环

- Debug加载 跟踪

```
C:\masm>debug p4.exe
-r
AX=0000  BX=0000  CX=001B  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0B2D  ES=0B2D  SS=0B3D  CS=0B3D  IP=0000   NV UP EI PL NZ NA PO NC
0B3D:0000 B8FFFF        MOV     AX,FFFF
-u 0b3d:0
0B3D:0000 B8FFFF        MOV     AX,FFFF
0B3D:0003 8ED8          MOV     DS,AX
0B3D:0005 BB0600        MOV     BX,0006
0B3D:0008 8A07          MOV     AL,[BX]
0B3D:000A B400          MOV     AH,00
0B3D:000C BA0000        MOV     DX,0000
0B3D:000F B97B00        MOV     CX,007B
0B3D:0012 03D0          ADD     DX,AX
0B3D:0014 E2FC          LOOP    0012
0B3D:0016 B8004C        MOV     AX,4C00
0B3D:0019 CD21          INT     21
0B3D:001B E83E0D        CALL    0D5C
0B3D:001E 83C404        ADD     SP,+04
-
```

5.3 在Debug中跟踪loop循环

- Debug执行“g 0012”后，CS:0012前的程序段被执行，从各个相关的寄存器中的值，我们可以看出执行的结果：

```
-g 0012

AX=0034  BX=0006  CX=007B  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=FFFF  ES=0B2D  SS=0B3D  CS=0B3D  IP=0012   NV UP EI PL NZ NA PO NC
0B3D:0012 03D0           ADD     DX,AX
-
```

5.3 在Debug中跟踪loop循环

- 跟踪了两次循环的过程，可以确定循环程序段在逻辑上是正确的。
- 还剩 121（cx=79h）次循环，若不再继续一步一步观察循环过程了，可使用P命令将循环一次执行完，Debug就会自动重复执行循环中的指令，直到(cx)=0为止。

5.4 Debug和Masm的不同处理

- Debug中写过类似的指令：

  mov ax,[0]

  表示将ds:0处的数据送入al中。

- **注意**：在汇编源程序中，指令“mov ax,[0]”被编译器当作指令“mov ax,0”处理。

5.4 Debug和Masm的不同处理

- 示例：将内存2000:0、2000:1 、2000:2、2000:3单元中的数据送入al，bl，cl，dl中。
- （1）在Debug中编程实现
- （2）汇编程序实现

- 两种实现的实际实施情况

5.4 Debug和汇编编译器Masm对指令的不同处理

- 在Debug中编程实现：

```
mov ax,2000h
 mov ds,ax
 mov al,[0]
 mov bl,[1]
 mov cl,[2]
 mov dl,[3]
```

```
C:\masm>debug
-a
0AE8:0100 mov ax,2000
0AE8:0103 mov ds,ax
0AE8:0105 mov al,[0]
0AE8:0108 mov bl,[1]
0AE8:010C mov cl,[2]
0AE8:0110 mov dl,[3]
0AE8:0114
-u
0AE8:0100 B80020        MOV     AX,2000
0AE8:0103 8ED8          MOV     DS,AX
0AE8:0105 A00000        MOV     AL,[0000]
0AE8:0108 8A1E0100      MOV     BL,[0001]
0AE8:010C 8A0E0200      MOV     CL,[0002]
0AE8:0110 8A160300      MOV     DL,[0003]
```

5.4 Debug和汇编编译器Masm对指令的不同处理

- 汇编程序实现：

```
assume cs:code
code segment
    mov ax,2000h
    mov ds,ax
    mov al,[0]
    mov bl,[1]
    mov cl,[2]
    mov dl,[3]

    mov ax,4c00h
    int 21h
code ends
end
```

将汇编源程序编译连接成.exe文件，用debug加载，如图：

```
C:\masm>debug compare.exe
-r
AX=0000  BX=0000  CX=0012  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0B2D  ES=0B2D  SS=0B3D  CS=0B3D  IP=0000   NV UP EI PL NZ NA PO NC
0B3D:0000 B80020        MOV     AX,2000
-u 0b3d:0000
0B3D:0000 B80020        MOV     AX,2000
0B3D:0003 8ED8          MOV     DS,AX
0B3D:0005 B000          MOV     AL,00
0B3D:0007 B301          MOV     BL,01
0B3D:0009 B102          MOV     CL,02
0B3D:000B B203          MOV     DL,03
0B3D:000D B8004C        MOV     AX,4C00
0B3D:0010 CD21          INT     21
```

5.4 Debug和汇编编译器Masm对指令的不同处理

- 在masm汇编中，访问内存单元时，需要注明使用的段寄存器。

如：若要访问2000:0单元：

```
mov ax,2000h
mov ds,ax
mov al,ds:[0]
```

5.5 loop和[bx]的联合应用

- 问题：计算ffff:0~ffff:b单元中的数据的和，结果存储在dx中。
- 分析：

（1）运算结果是否超出 dx 所能存储的范围？

ffff:0～ffff:b内存单元中的数据是字节型数据，范围在0～255之间，12个这样的数据相加，结果不会大于 65535，可以在dx中存放下。

5.5 loop和[bx]的联合应用

- 问题：计算ffff:0~ffff:b单元中的数据的和，结果存储在dx中。
- 分析：

（2）我们是否将 ffff:0～ffff:b中的数据直接累加到dx中？

当然不行，因为ffff:0～ffff:b中的数据是8位的，不能直接加到16位寄存器dx中。

数据类型不匹配问题

5.5 loop和[bx]的联合应用

- 问题：计算ffff:0~ffff:b单元中的数据的和，结果存储在dx中。
- 分析：

（3）我们能否将ffff:0～ffff:b中的数据累加到dl中，并设置(dh=0，从而实现累加到dx中的目标？

这也不行，因为dl是8位寄存器，能容纳的数据的范围在小 255 之间，ffff : 0～ffff:b中的数据也都是 8 位，如果仅向dl中累加12个 8 位数据，很有可能造成进位丢失。

数据越界问题

5.5 loop和[bx]的联合应用

- 问题：计算ffff:0~ffff:b单元中的数据的和，结果存储在dx中。
- 分析：

（4）我们到底怎样将用ffff:0～ffff:b中的8位数据，累加到16位寄存器dx中？

5.5 loop和[bx]的联合应用

- 做加法的时候，我们有两种方法：
  - 第一种方法

    (dx)=(dx)+内存中的8位数据

    存在数据类型不匹配问题

  - 第二种方法

    (dl)=(dl)+内存中的8 位数据；

    可能存在数据超界问题。

5.5 loop和[bx]的联合应用

- 问题：计算ffff:0~ffff:b单元中的数据的和，结果存储在dx中。
- 分析：

  怎样解决这两个看似矛盾的问题？

  目前的方法（在后面的课程中我们还有别的方法）就是我们得用一个16位寄存器来做中介。

5.5 loop和[bx]的联合应用

- 我们将内存单元中的8位数据赋值到一个16位寄存器ax中，再将ax中的数据加到dx上，从而使两个运算对象的类型匹配并且结果不会超界。

5.5 loop和[bx]的联合应用

- 程序代码

```
assume cs:code

code segment

    mov ax,0ffffh
    mov ds,ax                   ;设置(ds)=ffffh

    mov dx,0                    ;初始化累加寄存器，(dx)=0

    mov al,ds:[0]
    mov ah,0                    ;(ax)=((ds)*16+0)=(ffff0h)
    add dx,ax                   ;向dx中加上ffff:0单元的数值

    mov al,ds:[1]
    mov ah,0                    ;(ax)=((ds)*16+1)=(ffff1h)
    add dx,ax                   ;向dx中加上ffff:1单元的数值

    mov al,ds:[2]
    mov ah,0                    ;(ax)=((ds)*16+2)=(ffff2h)
    add dx,ax                   ;向dx中加上ffff:2单元的数值

    mov al,ds:[3]
    mov ah,0                    ;(ax)=((ds)*16+3)=(ffff3h)
    add dx,ax                   ;向dx中加上ffff:3单元的数值

    mov al,ds:[4]
    mov ah,0                    ;(ax)=((ds)*16+4)=(ffff4h)
    add dx,ax                   ;向dx中加上ffff:4单元的数值

    mov al,ds:[5]
    mov ah,0                    ;(ax)=((ds)*16+5)=(ffff5h)
    add dx,ax                   ;向dx中加上ffff:5单元的数值

    mov al,ds:[6]
    mov ah,0                    ;(ax)=((ds)*16+6)=(ffff6h)
    add dx,ax                   ;向dx中加上ffff:6单元的数值

    mov al,ds:[7]
    mov ah,0                    ;(ax)=((ds)*16+7)=(ffff7h)
    add dx,ax                   ;向dx中加上ffff:7单元的数值

    mov al,ds:[8]
    mov ah,0                    ;(ax)=((ds)*16+8)=(ffff8h)
    add dx,ax                   ;向dx中加上ffff:8单元的数值

    mov al,ds:[9]
    mov ah,0                    ;(ax)=((ds)*16+9)=(ffff9h)
    add dx,ax                   ;向dx中加上ffff:9单元的数值

    mov al,ds:[0ah]
    mov ah,0                    ;(ax)=((ds)*16+0ah)=(ffffah)
    add dx,ax                   ;向dx中加上ffff:a单元的数值

    mov al,ds:[0bh]
    mov ah,0                    ;(ax)=((ds)*16+0bh)=(ffffbh)
    add dx,ax                   ;向dx中加上ffff:b单元的数值

    mov ax,4c00h                ;程序返回
    int 21h

code ends

end
```

程序5.5

5.5 loop和[bx]的联合应用

- 问题5.4

  应用loop 指令，改进程序5.5，使它的指令行数让人能够接受。

  思考后看分析。

5.5 loop和[bx]的联合应用

分析

这12段代码可一般化为：

```
mov al,ds:[x]
mov ah,0
add dx,ax
```

```
assume cs:code

code segment

    mov ax,0ffffh
    mov ds,ax                   ;设置(ds)=ffffh

    mov dx,0                    ;初始化累加寄存器，(dx)=0

    mov al,ds:[0]
    mov ah,0                    ;(ax)=((ds)*16+0)=(ffff0h)
    add dx,ax                   ;向dx中加上ffff:0单元的数值

    mov al,ds:[1]
    mov ah,0                    ;(ax)=((ds)*16+1)=(ffff1h)
    add dx,ax                   ;向dx中加上ffff:1单元的数值

    mov al,ds:[2]
    mov ah,0                    ;(ax)=((ds)*16+2)=(ffff2h)
    add dx,ax                   ;向dx中加上ffff:2单元的数值

    mov al,ds:[3]
    mov ah,0                    ;(ax)=((ds)*16+3)=(ffff3h)
    add dx,ax                   ;向dx中加上ffff:3单元的数值

    mov al,ds:[4]
    mov ah,0                    ;(ax)=((ds)*16+4)=(ffff4h)
    add dx,ax                   ;向dx中加上ffff:4单元的数值

    mov al,ds:[5]
    mov ah,0                    ;(ax)=((ds)*16+5)=(ffff5h)
    add dx,ax                   ;向dx中加上ffff:5单元的数值

    mov al,ds:[6]
    mov ah,0                    ;(ax)=((ds)*16+6)=(ffff6h)
    add dx,ax                   ;向dx中加上ffff:6单元的数值

    mov al,ds:[7]
    mov ah,0                    ;(ax)=((ds)*16+7)=(ffff7h)
    add dx,ax                   ;向dx中加上ffff:7单元的数值

    mov al,ds:[8]
    mov ah,0                    ;(ax)=((ds)*16+8)=(ffff8h)
    add dx,ax                   ;向dx中加上ffff:8单元的数值

    mov al,ds:[9]
    mov ah,0                    ;(ax)=((ds)*16+9)=(ffff9h)
    add dx,ax                   ;向dx中加上ffff:9单元的数值

    mov al,ds:[0ah]
    mov ah,0                    ;(ax)=((ds)*16+0ah)=(ffffah)
    add dx,ax                   ;向dx中加上ffff:a单元的数值

    mov al,ds:[0bh]
    mov ah,0                    ;(ax)=((ds)*16+0bh)=(ffffbh)
    add dx,ax                   ;向dx中加上ffff:b单元的数值

    mov ax,4c00h                ;程序返回
    int 21h

code ends

end
```

程序5.5

5.5 loop和[bx]的联合应用

- 分析：（续）

  而这些不同的偏移地址是可在0≤X≤0bH的范围内递增变化的。

  我们可以用数学语言来描述这个累加的运算：

$$sum = \sum_{x=0}^{0bh} (0ffffh \times 10h + x)$$

5.5 loop和[bx]的联合应用

- 分析：（续）
  - 从程序实现上，我们用循环实现：

    (al)=((ds)*16+X)

    (ah)=0

    (dx)=(dx)+(ax)

    一共循环12次，
  - 循环开始前(dx)=0ffffh，X=0，ds:X指向第一个内存单元。
  - 每次循环后，X递增，ds:X指向下一个内存单元。

5.5 loop和[bx]的联合应用

- 分析：（续）完整的算法描述

初始化：

(ds)=0ffffh

X=0

(dx)=0

循环12次：

(al) =((ds)*16+X)

(ah) = 0

(dx)=(dx)+(ax)

X=X+1

5.5 loop和[bx]的联合应用

- 分析：（续）完整的算法描述

初始化：

(ds)=0ffffh

X=0

(dx)=0

循环12次：

(al) =((ds)*16+X)

(ah) = 0

(dx)=(dx)+(ax)

X=X+1

内存单元偏移地址X应该是一个变量，循环过程中X递增。

——将偏移地址放到 bx中，用[bx]的方式访问内存单元。

在循环开始前设(bx)=0，每次循环，将bx中的内容加1即可。

5.5 loop和[bx]的联合应用

- 分析：（续）

  如何实现循环12次？

  用cx控制loop循环次数

5.5 loop和[bx]的联合应用

- 分析：（续）更详细的算法描述初始化

```
(ds)=0ffffh
(bx)=0
(dx)=0
(cx)=12
循环12 次：
s:(al)=((ds)*16+(bx))
(ah)=0
(dx)=(dx)+(ax)
(bx)=(bx)+1
loop s
```

5.5 loop和[bx]的联合应用

- 汇编代码

程序 5.6

```
assume cs:code
code segment

    mov ax,0ffffh
    mov ds,ax
    mov bx,0                ;初始化 ds:bx 指向 ffff:0

    mov dx,0                ;初始化累加寄存器 dx，(dx)=0

    mov cx,12               ;初始化循环计数寄存器 cx，(cx)=12

s:  mov al,[bx]
    mov ah,0
    add dx,ax               ;间接向 dx 中加上((ds)*16+(bx))单元的数值
    inc bx                  ;ds:bx 指向下一个单元
    loop s

    mov ax,4c00h
    int 21h

code ends
end
```

5.5 loop和[bx]的联合应用

- 编程中常需要用同一种方法处理**地址连续的内存单元**中的数据的问题——可用循环来解决，每次循环中按照同一种方法来改变要访问的内存单元的地址的**变量**。

- 例如：“mov al,[bx]”中的 bx就可以看作一个代表内存单元地址的变量，通过改变bx中的数值，改变指令访问的内存单元。

5.6 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。
- 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，如“ds:”、“cs:”、“ss:”或“es:”——段前缀。

5.7 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。
- 比如下面的指令：

```
mov ax,1000h
mov ds,ax
mov al,0
mov ds:[0],al
```

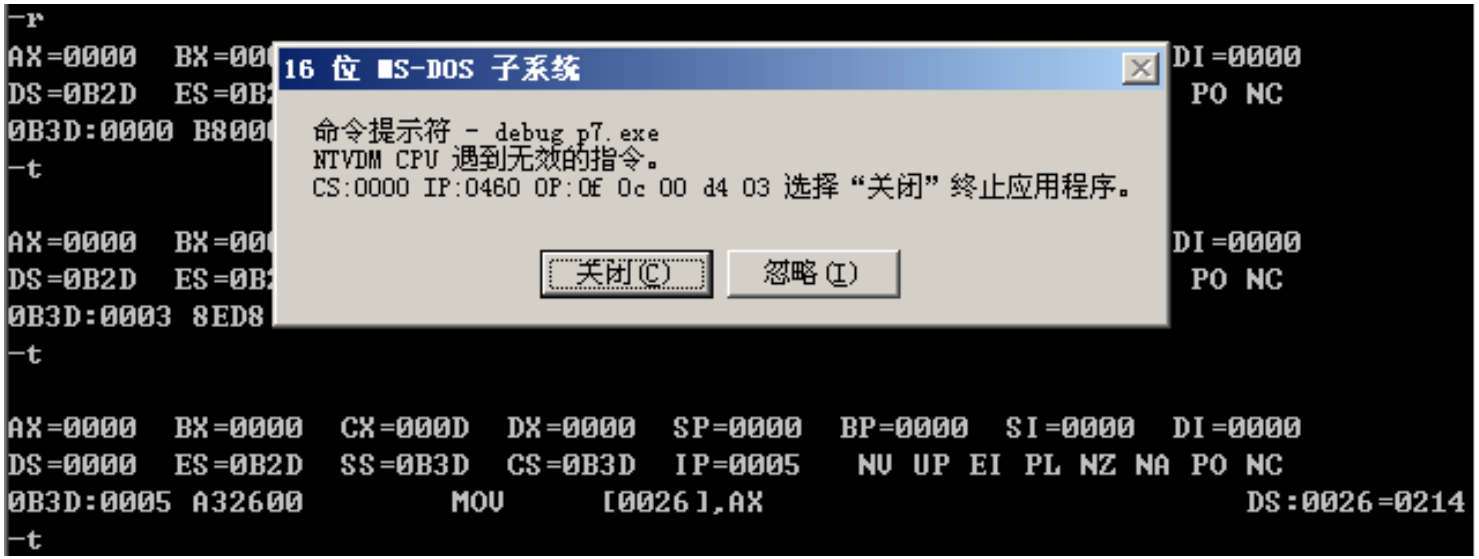
5.7 一段安全的空间

- 以前在Debug中，为了讲解上的方便，写过类似的指令。
- 但这种做法是不合理的，因为之前我们并没有论证过1000:0中是否存放着重要的系统数据或代码。
- 如果1000:0中存放着重要的系统数据或代码，“mov ds:[0],al”将其改写，将引发错误。
- 比如程序

5.7 一段安全的空间

- 在Windows2000的DOS方式中，在Debug里执行“mov [0026],ax”的结果。如果在实模式（即纯DOS方式）下执行程序p7.exe，将会引起死机。

  产生这种结果的原因是0:0026处存放着重要的系统数据，而“mov [0026],ax”将其改写。



5.7 一段安全的空间

- 结论：若不能确定一段内存空间中是否存放着重要的数据或代码的时候，不要修改它。

- 学习汇编语言是为理解计算机底层的基本工作机理，我们尽量直接对硬件编程，而不去理会操作系统。
  - 虽然我们是在操作系统的环境中工作，操作系统管理所有的资源，也包括内存。

5.7 一段安全的空间

- 注意：
  - 在纯DOS方式（实模式）下，可以不理会DOS，直接用汇编语言去操作真实的硬件。
  - 但在Windows 2000、UNIX这些运行于CPU保护模式下的操作系统中，不可能去操作真实的硬件。
  - 在一般的PC机中，DOS方式下，0:200h~0:2FFh的256个字节的空间可安全安全使用。

5.7 一段安全的空间

- 总结：
  - 我们需要直接向一段内存中写入内容；
  - 这段内存空间不应存放系统或其他程序的数据或代码，否则写入操作很可能引发错误。
  - DOS方式下，一般情况，0:200~0:2FF 空间中没有系统或其他程序的数据或代码；
  - 以后，我们需要直接向一段内存中写入内容时，就使用0:200~0:2FF这段空间。

5.8 段前缀的使用

- 问题：将内存ffff:**0**~ffff:**b**单元中的数据拷贝到0:200~0:20b单元中。
- 分析：
  - 目标内存空间等价于0020:**0**~0020:**b**。
    - 这样目标单元的偏移地址和源始单元的偏移地址相同，用变了X是表示。
    - 可循环得将源单元ffff:X数据复制到目标单元0020:X，我们用bx来存放变量X。

5.8 段前缀的使用

- 分析：（续）
  - 拷贝的过程应用循环实现，简要描述如下：

1. 初始化：X=0
2. 循环12次：
   2.1 将ffff:X单元中数据送入0020:X（需寄存器中）
   2.2. X=X+1

5.8 段前缀的使用

```
assume cs:code
code segment
        mov bx,0          ;(bx)=0,偏移地址从0开始
        mov cx,12         ;(cx)=12，循环12次
s:      mov ax,0ffffh
        mov ds,ax         ;(ds)=0ffffh
        mov dl,[bx]       ;(dl)=((ds)*16+(bx))，将ffff:bx中的数据送入dl
        mov ax,0020h
        mov ds,ax         ;(ds)=0020h
        mov [bx],dl       ;((ds)*16+(bx))=(dl)，将dl的数据送入0020:bx
        inc bx            ;(bx)=(bx)+1
        loop s

        mov ax,4c00h
        int 21h
code ends
end
```

因源单元ffff:X和目标单元0020:X 相距大于64KB，不在同一段内，程序每次循环中要设置两次ds——正确但是效率不高。

若使用两个段寄存器分别存放源单元ffff:X和目标单元0020:X的段地址，可省略循环中重复的ds设置。

5.8 段前缀的使用

- 改进后程序

用ds存放源空间ffff:0~ffff:b的段地址

用es存放目标空间0020:0~0020:b的段地址

```
assume cs:code
code segment
        mov ax,0ffffh
        mov ds,ax
        mov ax,0020h
        mov es,ax
        mov bx,0
        mov cx,12
s:      mov dl,[bx]
        mov es:[bx],dl
        inc bx
        loop s

        mov ax,4c00h
        int 21h
code ends
end
```

```
        ;(ds)=0ffffh

        ;(es)=0020h
        ;(bx)=0,此时ds:bx指向ffff:0, es:bx指向0020:0
        ;(cx)=12, 循环12次
        ;(dl)=((ds)*16+(bx)), 将ffff:bx中的数据送入dl
        ;((es)*16+(bx))=(dl), 将dl的数据送入0020:bx
        ;(bx)=(bx)+1
```

显式地用段前缀“es:”给出单元的段地址，这样就不必在循环中重复设置ds。