《汇编语言》课件

第4章 第1个程序

- 4.1 一个源程序从写出到执行的过程
- 4.2 源程序
- 4.3 编辑源程序
- 4.4 编译
- 4.5 连接
- 4.6 以简化的方式进行编译和连接
- 4.7 1.exe的执行
- 4.8 可执行文件中的程序装入内存并运行的原理
- 4.9 程序执行过程的跟踪

4.1 一个源程序从写出到执行的过程

- 一个汇编语言程序从写出到最终执行的简要过程

编写 → 编译 → 连接 → 执行

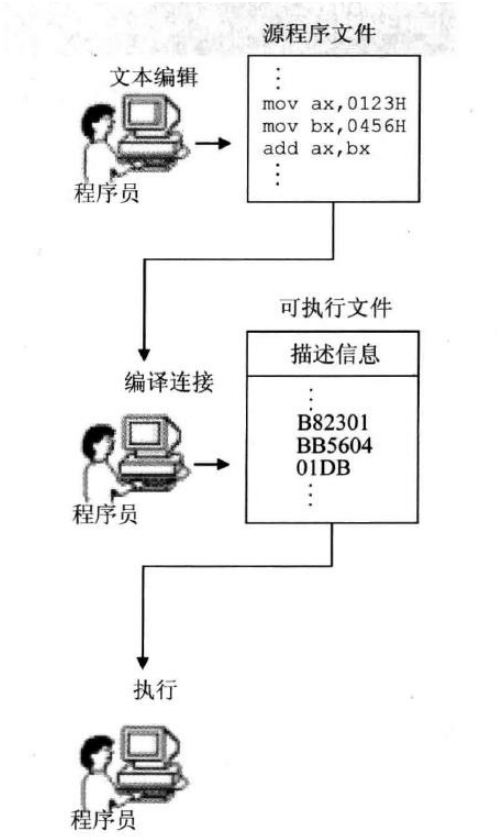


图 4.1　一个汇编语言程序从写出到执行的过程

4.1 一个源程序从写出到执行的过程

- 一个汇编语言程序从写出到最终执行的简要过程：

编写 → 编译 → 连接 → 执行

使用文本编辑器（如Edit、记事本等），用汇编语言编写**汇编源程序**。

4.1 一个源程序从写出到执行的过程

- 一个汇编语言程序从写出到最终执行的简要过程：

编写 → 编译 → 连接 → 执行

使用汇编语言**编译程序**对源程序进行编译，产生**目标文件**；

再用**连接程序**将目标文件连接成**可执行文件**。

包含两部分内容：

（1）程序（从原程序中的汇编指令翻译过来的机器码）和数据（源程序中定义的数据）

（2）相关的**描述信息**（比如：程序有多大、要占多少内存空间等）

4.1 一个源程序从写出到执行的过程

- 一个汇编语言程序从写出到最终执行的简要过程：

编写 → 编译 → 连接 → 执行

在操作系统中，执行可执行文件中的程序。

操作系统依照可执行文件中的描述信息，将可执行文件中的机器码和数据加载入内存，并进行相关的初始化（比如：设置CS:IP指向第一条要执行的指令），然后由CPU执行程序。

4.2 源程序

```
assume cs:codesg

codesg segment

start:  mov ax,0123H
        mov bx,0456H
        add ax,bx
        add ax,ax

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

- 汇编指令
- 伪指令

XXX segment

XXX ends

end

assume

4.2 源程序

- 汇编指令——有对应的机器码的指令，可以被编译为机器指令，最终为CPU所执行。

```
assume cs:codesg

codesg segment

start:  mov ax,0123H
        mov bx,0456H
        add ax,bx
        add ax,ax

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

4.2 源程序

- 伪指令——没有对应机器码的指令，最终不被CPU所执行。
  - 谁来执行伪指令呢？伪指令是由编译器来执行的指令，编译器根据伪指令来进行相关的编译工作。

定义一个段

- segment和ends是一对成对使用的伪指令
  - 功能：定义一个段，

```
assume cs:codesg

codesg segment

start:  mov ax,0123H
        mov bx,0456H
        add ax,bx
        add ax,ax

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

segment 说明一个段开始，
ends 说明一个段结束。

一个段必须有一个**段名称**来标号，使用格式为：

段名 segment
…
段名 ends

定义一个段

- 一个汇编程序是由多个段组成的，这些段被用来存放代码、数据或当作栈空间来使用。
  - 一个的汇编程序中至少要有一个段存放代码。

程序结束标记

- end 是一个汇编程序的结束标记
  - 功能：编译器在编译汇编程序的过程中，如果碰到了伪指令 end，就结束对源程序的编译。

```
assume cs:codesg

codesg segment

start:  mov ax,0123H
        mov bx,0456H
        add ax,bx
        add ax,ax

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

一定要在程序结尾处加上伪指令end，否则，编译器无法知道程序在何处结束。

注意：不要搞混了end和ends

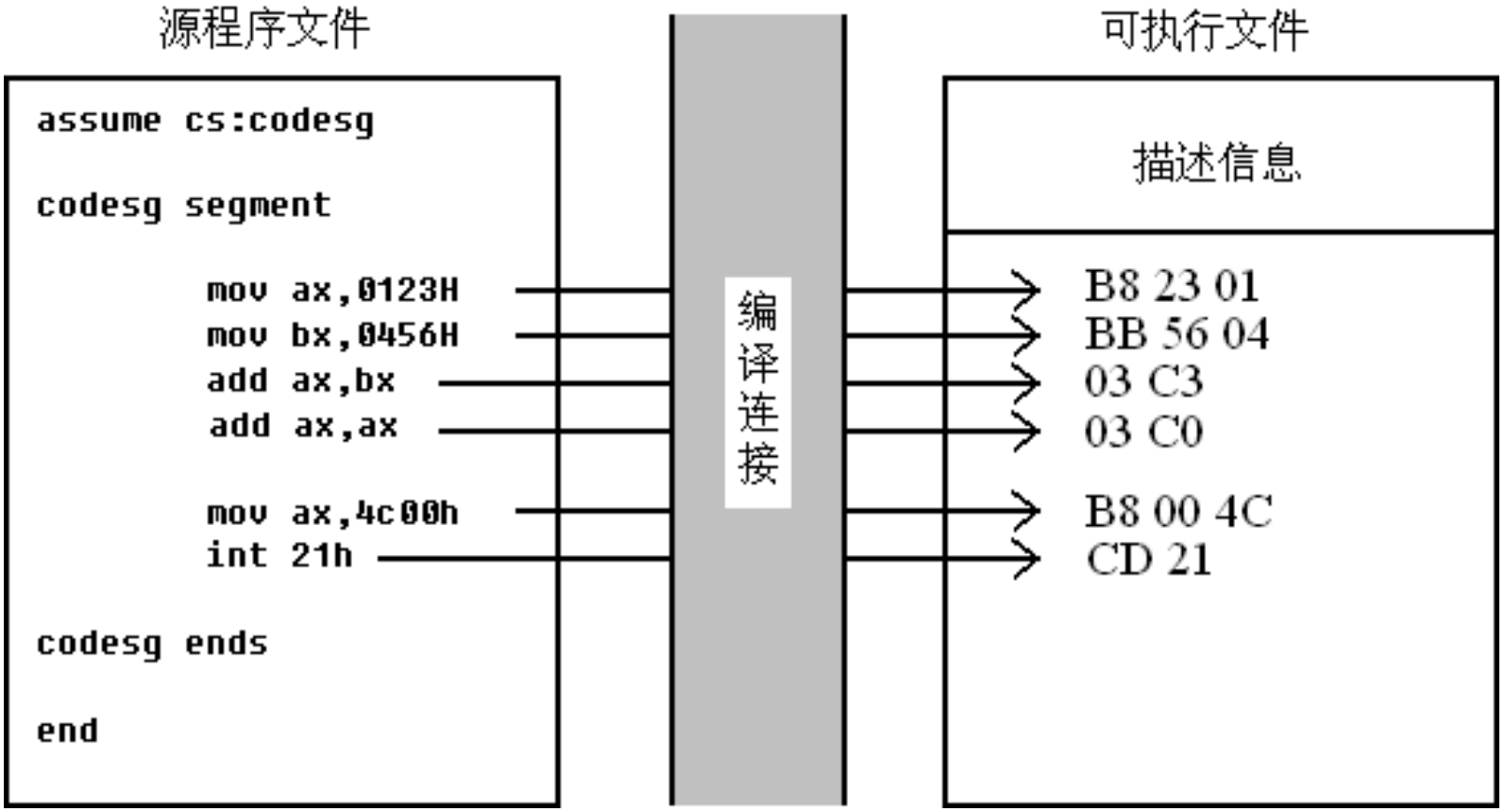
寄存器与段的关联假设

- assume：含义为“假设”。
  - 功能：它假设某一段寄存器和程序中的某一个用segment … ends 定义的段相关联。

```
assume cs:codesg

codesg segment

start:  mov ax,0123H
        mov bx,0456H
        add ax,bx
        add ax,ax

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

通过assume说明这种关联，编译程序将段寄存器和某一个具体的段相联系。

4.2 源程序



汇编源程序

伪指令 （编译器处理）

汇编指令（编译为机器码）

编译链接

程 序

由计算机执行的指令或数据。

4.2 源程序

- 标号
  - 功能：一个标号指代了一个地址。

```
assume cs:codesg

codesg segment

start:  mov ax,0123H
        mov bx,0456H
        add ax,bx
        add ax,ax

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

定义一个段，段名为codesg
段名最终将被编译、连接程序处理为一个段的段地址

4.2 源程序

- 程序的结构
  - 任务：编程运算 2^3。

```
assume cs:abc

abc segment
        mov ax,2
        add ax,ax
        add ax,ax
abc ends

end
```

段与段寄存器关联

定义一个段

实现处理任务

程序结束

4.2 源程序

- 程序返回
  - 一个程序结束后，要将CPU的控制权交还给操作系统——程序返回。
  - 返回方法：在程序段的末尾添加

```
mov ax,4c00H
int 21H
```

段结束、程序结束、程序返回

- 几个和结束相关的内容

目 的	相关指令	指令性质	指令执行者
通知编译器一个段结束	段名 ends	伪指令	编译时，由编译器执行
通知编译器程序结束	end	伪指令	编译时，由编译器执行
程序返回	mov ax,4c00H int 21H	汇编指令	编译时，由CPU执行

语法错误和逻辑错误

- 语法错误
  - 程序在编译时被编译器发现的错误；
  - 容易发现。

```
aume cs:abc
abc segment
    mov ax,2
    add ax,ax
    add ax,ax
end
```

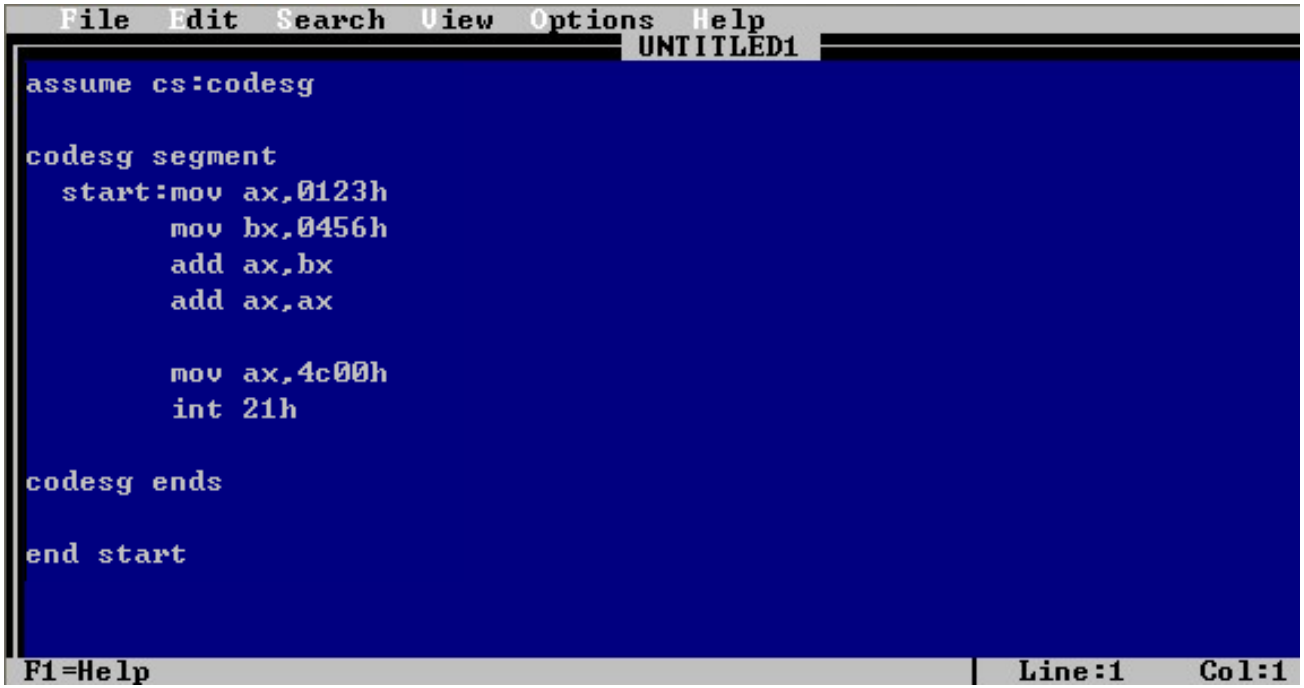
语法错误和逻辑错误

- 逻辑错误
  - 程序在编译正常但运行时发生的错误；
  - 不容易发现。

```
assume cs:abc

abc segment
        mov ax,2
        add ax,ax
        add ax,ax

        mov ax,4c00H
        int 21H
abc ends

end
```

4.4 编辑源程序

- 用任何文本编辑器编辑代码：



4.4 编译

```
C:\masm>masm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Source filename [.ASM]: c:\1.asm
Object filename [1.OBJ]:
```

- 进入DOS方式，进入 C:\masm 目录，运行 masm.exe。
- 如果源程序文件不是以 asm 为扩展名的话，就要输入它的全名。比如p1.asm。
- 在输入源程序文件名的时一定要指明所路径。若文件在当前路径下，可只输入文件名。

4.4 编译

```
C:\masm>masm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Source filename [.ASM]: c:\1.asm
Object filename [1.OBJ]:
```

- 输入要编译的源文件文件名后，按 Enter键。
- 目标文件（*.obj）是我们对一个源程序进行编译要得到的最终结果。
- 编译程序默认要输出的目标文件名为1.obj，所以可以不必再另行指定文件名。

4.4 编译

```
C:\masm>masm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Source filename [.ASM]: c:\1.asm
Object filename [1.OBJ]:
Source listing  [NUL.LST]:
```

- 列表文件是编译器将源程序编译为目标文件的过程中产生的中间结果。

- 可以不生成这个文件，直接按 Enter键即可。

4.4 编译

```
C:\masm>masm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Source filename [.ASM]: c:\1.asm
Object filename [1.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:
```

- 编译程序提示输入交叉引用文件的名称。
- 这个文件同列表文件一样，是编译器将源程序编译为目标文件过程中产生的中间结果。
- 可以不生成这个文件，直接按 Enter 键即可。

4.4 编译

```
C:\masm>masm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Source filename [.ASM]: c:\1.asm
Object filename [1.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  50686 + 415330 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

C:\masm>_
```

- 对源程序的编译结束，编译器输出的最后两行告诉我们这个源程序没有警告错误和必须要改正的错误。

4.4 编译

- 一般来说，有两类错误使我们得不到所期望的目标文件：

  1. 我们程序中有“Severe Errors”；
  2. 找不到所给出的源程序文件。

4.5 连接

- 在对源程序进行编译得到目标文件后，我们需要对目标文件进行连接，从而得到可执行文件。
- 继续上一节的过程，我们再将C:\masm\1.obj连接为C:\masm\1.exe。

4.5 连接

```
C:\masm>link

Microsoft (R) Overlay Linker  Version 3.69
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Object Modules [.OBJ]: 1
```

- 进入DOS方式，进入C:\masm目录，运行link.exe。
- 如果目标文件不是以obj为扩展名的话，就要输入它的全名。比如：p1.bin。
- 在输入目标文件名的时候，要注意指明它所在的路径。这里，我们要连接的文件是当前路径下1.obj，所以此处输入“1”。

4.5 连接

```
C:\masm>link

Microsoft (R) Overlay Linker  Version 3.69
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Object Modules [.OBJ]: 1
Run File [1.EXE]:
```

- 输入要连接的目标文件名后，按Enter键。
- 可执行文件是我们对一个程序进行连接要得到的最终结果。
- 连接程序默认要输出的可执行文件名为 1.EXE，所以可以不必再另行指定文件名。
- 我们直接按 Enter 键，使用连接程序设定的可执行文件名。

4.5 连接

```
C:\masm>link

Microsoft (R) Overlay Linker  Version 3.69
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Object Modules [.OBJ]: 1
Run File [1.EXE]:
List File [NUL.MAP]:
```

- 映像文件是连接程序将目标文件连接为可执行文件过程中产生的中间结果。
- 可以不生成这个文件，直接按 Enter 键即可。

4.5 连接

```
C:\masm>link

Microsoft (R) Overlay Linker  Version 3.69
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Object Modules [.OBJ]: 1
Run File [1.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
```

- 连接程序提示输入库文件的名称。
- 库文件里包含了一些可以调用的子程序，如果我们的程序中调用了某一个库文件中的子程序，就需要在连接的时候，将这个库文件和我们的目标文件连接到一起，生成可执行文件。
- 如果没有调用任何子程序，直接按Enter键即可。

4.5 连接

```
C:\masm>link

Microsoft (R) Overlay Linker  Version 3.69
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Object Modules [.OBJ]: 1
Run File [1.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment

C:\masm>
```

- 对目标文件的连接结束，连接程序输出的最后一个警告错误：“没有栈段”，这里我们不理会这个错误。

4.5 连接

- 强调：
  - 学习汇编的目的是通过用汇编语言进行编程而深入地理解计算机底层的基本工作机理，达到可以随心所欲地控制计算机的目的。
  - 编程希望直接对硬件编程，却不同用机器码编程。
  - 操作系统和工具软件——编辑器（Edit）、编译器（masm）、连接器（link）、调试工具（debug）——会用这即可。

4.5 连接

- 连接的作用是什么呢？
  - 连接多个文件，便于大程序编写
    - 大源程序可分为多个小源程序，每个源程序可独立编译成一个目标文件，将它们连接生成一个可执行文件；
  - 连接库文件，调用库的子程序
    - 程序中调用某个库文件中的子程序时，将这个库文件和该程序的目标文件连接到一起，生成一个可执行文件；
  - 连接后将目标文件转化为可执行文件
    - 源程序编译生成目标文件，目标文件中有些内容不能直接执行，连接可将这些内容处理为最终的可执行信息。所以，只有一个文件的源程序，即使不调用库文件，也必须用连接程序对目标文件进行处理，生成可执行文件。

4.6 以简化的方式进行编译和连接

- 编译、连接的最终目的是用源程序文件生成可执行文件。在这个过程中所产生的中间文件都可以忽略。

- 可用一种较为简捷的方式进行编译、连接。

4.6 以简化的方式进行编译和连接

- 编译：

```
C:\masm>masm c:\1;
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.


  50686 + 415330 Bytes symbol space free

       0 Warning Errors
       0 Severe  Errors
```

4.6 以简化的方式进行编译和连接

- 连接：

```
C:\masm>link 1;

Microsoft (R) Overlay Linker  Version 3.69
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

LINK : warning L4021: no stack segment

C:\masm>_
```

4.7 1.exe的执行

- 在dos操作系统下输入文件名1.exe，回车，即可启动该程序运行。
- 程序执行完成后，返回，屏幕上再次出现操作系统的提示符。

```
C:\masm>1

C:\masm>
```

  - 屏幕上怎么看不到运行过程和结果？
    - 该程序只是做了一些数据操作，程序中没有向显示器输出任何信息。

4.8 可执行文件中的程序装入内存并运行的原理

- 汇编程序从写出到执行的过程：

编程 → 1.asm → 编译 → 1.obj → 连接 → 1.exe → 加载 → 内存中的程序 → 运行

(edit)　　(masm)　　(link)　　(操作系统)　　(CPU)

4.9 程序执行过程的跟踪

- 怎么观察程序的运行过程？——使用Debug。
  - Debug 可以将程序加载入内存，
  - 设置CS:IP指向程序的入口，
  - 但Debug并不放弃对CPU 的控制，这样，我们就可以使用Debug 的相关命令来单步执行程序，查看每条指令指令的执行结果。

4.9 程序执行过程的跟踪

- 接下来可以用R命令看一下各个寄存器的设置情况：

```
C:\masm>debug 1.exe
-r
AX=0000  BX=0000  CX=000F  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=129E  ES=129E  SS=12AE  CS=12AE  IP=0000   NV UP EI PL NZ NA PO NC
12AE:0000 0000 B82301        MOV     AX,0123
-
```

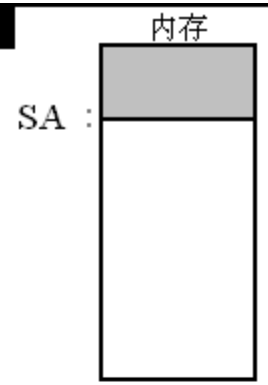
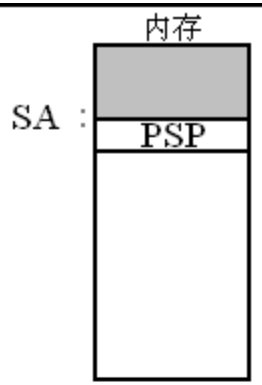
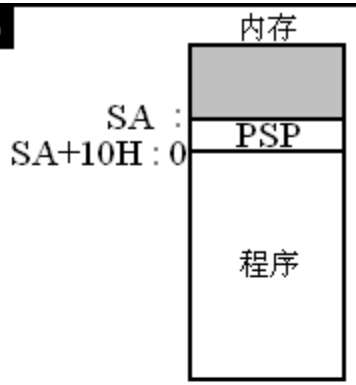
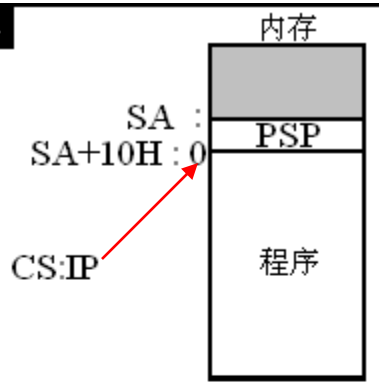
Debug将程序从可执行文件加载入内存后，CX中存放的是程序的长度(15字节)。

4.9 程序执行过程的跟踪

- 程序已从 1.exe 中装入内存后，若要查看它的内容，应该查看那里的内容呢？
- 程序被装入内存的什么地方？
- 我们如何得知？

EXE文件中的程序的加载过程

- 在DOS系统中.EXE文件中的程序的加载过程：

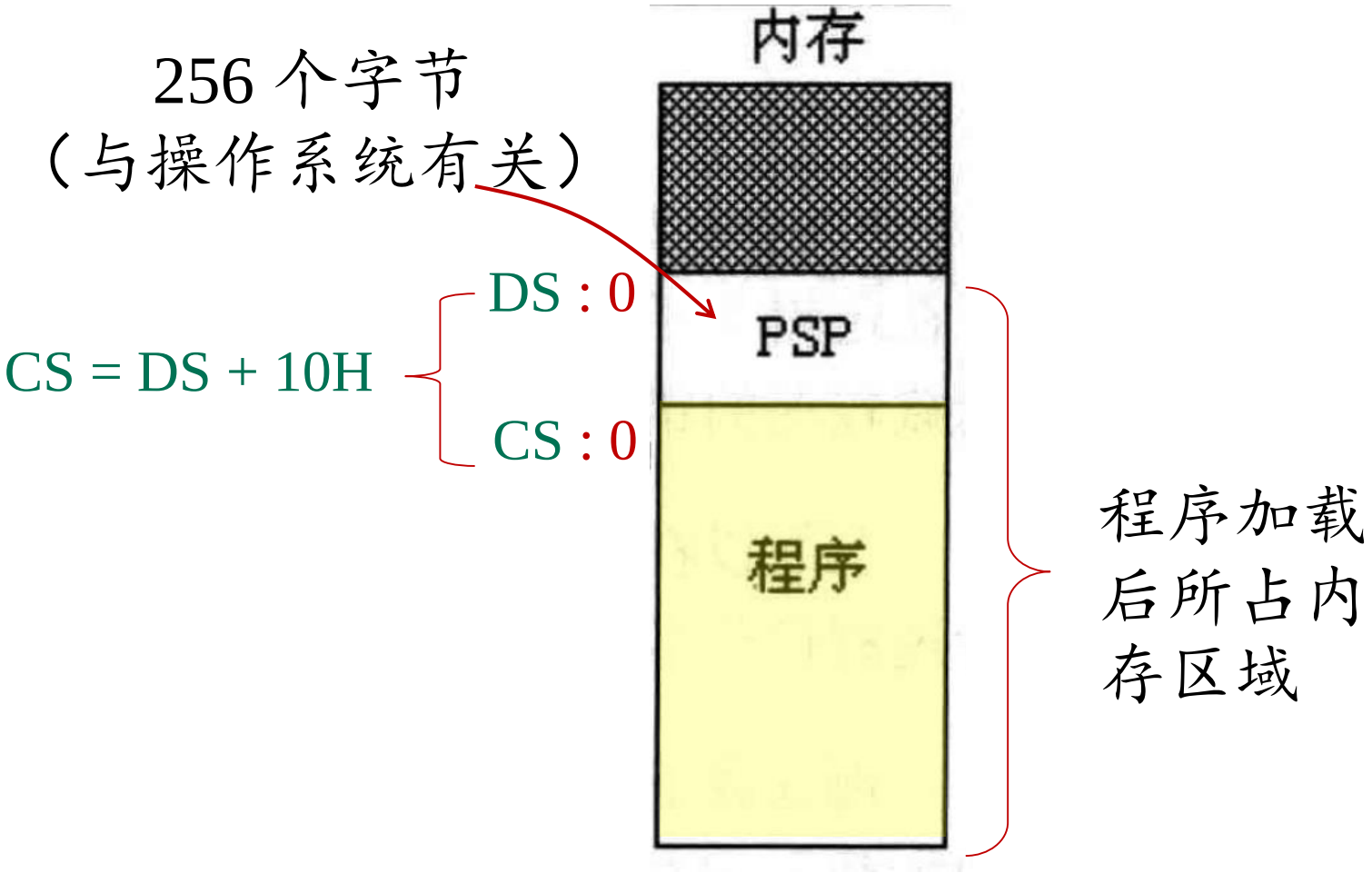
1	2	3	4
			
找到一段起始地址为SA:0000（即起始地址的偏移地址为0）的容量足够的空闲内存区；	在这段内存区的前256个字节中，创建一个称为程序的前缀（PSP）的数据区，DOS要利用PSP来和被加载程序进行通信； （读者可能不理解PSP的作用，不过没有关系。我们并不研究DOS的原理，只要知道有这个东西就可以了。）	从这段内存区的256字节处开始（在PSP的后面），将程序装入，程序的地址被设为SA+10H:0； （空闲的内存区从SA:0开始，0~255字节为PSP，从256字节处开始存放程序，为更好地区分PSP和程序，DOS一般将它们划分到不同的段中，所以，有了这样的地址安排： 空闲内存区：SA:0 PSP区：SA:0 程序区：SA+10H:0 注意：PSP区和程序区虽然物理地址连续，却有不同的段地址。）	将该内存区的段地址存入DS中，初始化其他相关寄存器后，设置CS:IP指向程序的入口。 DS=SA CS=SA+10H

EXE文件中的程序的加载过程

- 注意：有一步称为重定位的工作我们在上面没有讲解，因为这个问题和操作系统的关系较大，我们不作讨论。

EXE文件中的程序的加载过程

- EXE文件加载后的内存情况：



4.9 程序执行过程的跟踪

- Debug中用U命令查看一下其他指令：

```
C:\masm>debug 1.exe
-r
AX=0000  BX=0000  CX=000F  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=129E  ES=129E  SS=12AE  CS=12AE  IP=0000   NV UP EI PL NZ NA PO NC
12AE:0000 0000 B82301        MOV     AX,0123
-u
12AE:0000 B82301        MOV     AX,0123
12AE:0003 BB5604        MOV     BX,0456
12AE:0006 03C3          ADD     AX,BX
12AE:0008 03C0          ADD     AX,AX
12AE:000A B8004C        MOV     AX,4C00
12AE:000D CD21          INT     21
12AE:000F 83E201        AND     DX,+01
12AE:0012 BA85E2        MOV     DX,E285
12AE:0015 2E            CS:
12AE:0016 A385E2        MOV     [E285],AX
12AE:0019 E9A5FC        JMP     FCC1
12AE:001C 803EE70400    CMP     BYTE PTR [04E7],00
-
```

4.9 程序执行过程的跟踪

- Debug中用T命令单步执行程序中的每一条指令，并观察每条指令的执行结果，

- 到了 int 21，要用P命令执行：

```
AX=0AF2  BX=0456  CX=000F  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1402  CS=1402  IP=000A   NV UP EI PL NZ AC PO NC
1402:000A B8004C        MOV     AX,4C00
-t

AX=4C00  BX=0456  CX=000F  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1402  CS=1402  IP=000D   NV UP EI PL NZ AC PO NC
1402:000D CD21          INT     21
-p

Program terminated normally
-
```

表示int 21 执行后程序正常结束，返回debug。