

实验1-第一个汇编程序

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 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

4.1 汇编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

- 源程序中的“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）

- 程序：源程序中最终由计算机执行、处理的指令或数据。

4.1 汇编源程序从写出到执行的过程

```
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

4.1 汇编源程序从写出到执行的过程

```
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**和**ends**是一对成对使用的伪指令，这是在写可被编译器编译的汇编程序时，必须要用到的一对伪指令。
- **segment**和**ends**的功能是定义一个段，**segment**说明一个段开始，**ends**说明一个段结束。
- 一个段必须有一个名称来标识，使用格式为：
段名 **segment**
段名 **ends**

4.1 汇编源程序从写出到执行的过程

```
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。
- 一个有意义的汇编程序中至少要有有一个段，这个段用来存放代码。

4.1 汇编源程序从写出到执行的过程

```
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
```

- End 是一个汇编程序的结束标记，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。
- 如果程序写完了，要在结尾处加上伪指令 **end**。否则，编译器在编译程序时，无法知道程序在何处结束。
- 注意：不要混淆了 **end** 和 **ends**。

4.1 汇编源程序从写出到执行的过程

```
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
```

- **assume**: 含义为“假设”。
- 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。
- 通过**assume**说明这种关联，在需要的情况下，编译程序可以将段寄存器和某一个具体的段相联系。

4.1 汇编源程序从写出到执行的过程

```
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
```

■ 标号

- 一个标号指代了一个地址。
- codesg: 放在 segment 的前面，作为一个段的名称，这个段的名称最终将被编译、连接程序处理为一个段的段地址。

4.1 汇编源程序从写出到执行的过程

```
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
```

■ 程序返回

□ 一个程序结束后，将CPU的控制权交还给使它得以运行的程序，我们称这个过程为：程序返回。

□ 应该在程序的末尾添加返回的程序段。

```
mov ax,4c00H
```

```
int 21H
```

■ 这两条指令所实现的功能就是程序返回。

4.1 汇编源程序从写出到执行的过程

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

4.1 汇编源程序从写出到执行的过程

■ 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
```

4.1 汇编源程序从写出到执行的过程

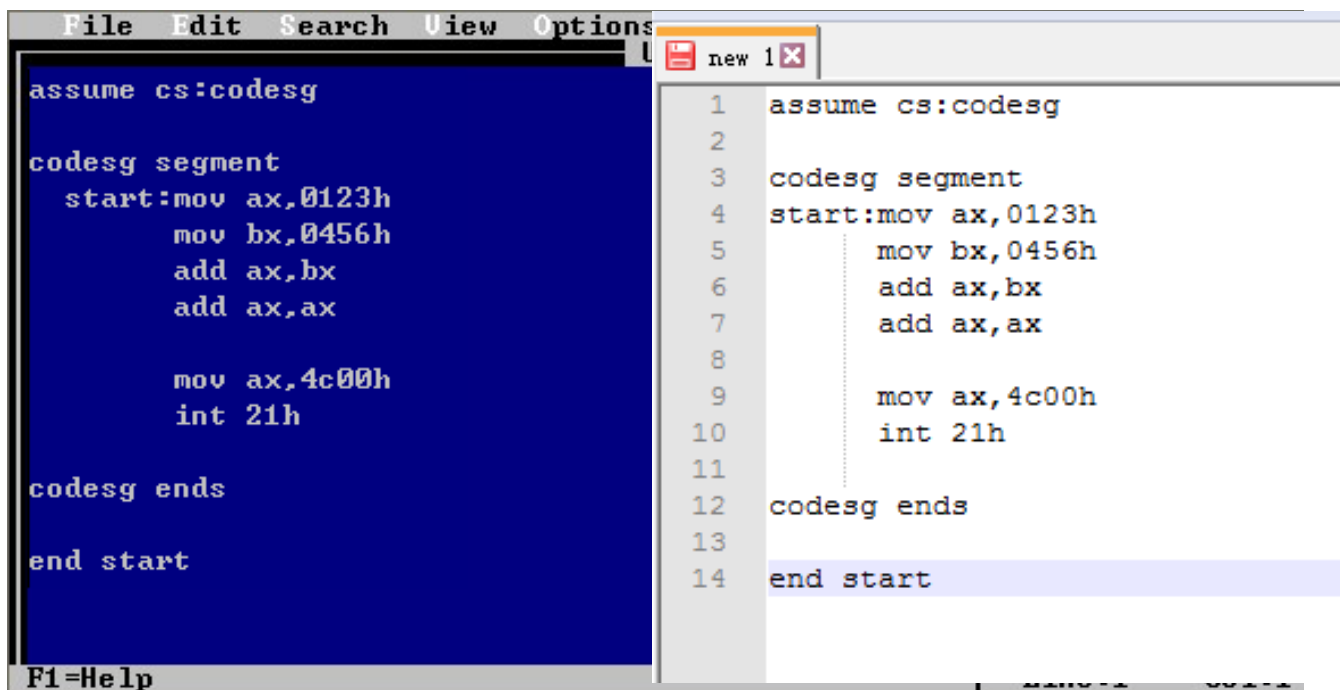
■ 逻辑错误

- ❑ 程序在编译时不能表现出来的、在运行时发生的错误；
- ❑ 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
```

编写源程序

- 在编辑器中编辑程序（dos中的edit，如记事本、notepad++、ultraedit等等），在其中编辑程序，如下图所示：



```
File Edit Search View Options
new 1 x

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 start

F1=Help
```

```
1  assume cs:codesg
2
3  codesg segment
4  start:mov ax,0123h
5           mov bx,0456h
6           add ax,bx
7           add ax,ax
8
9           mov ax,4c00h
10          int 21h
11
12 codesg ends
13
14 end start
```

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]:
```

- 通过cmd.exe，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.txt。
- 在输入源程序文件名的时候一定要指明它所在的路径。如果文件就在当前路径下，只输入文件名就可以。

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。

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]:
```

- 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。

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。

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。

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。

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`。

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”。

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。

连接

```
C:\masm>link

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

Object Modules [L.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。

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。

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行告诉我们，这个程序有一个警告错误：“没有栈段”，这里我们不理会这个错误。

连接

- 连接的作用有以下几个：
 - 当源程序很大时，可以将它分为多个源程序文件来编译，每个源程序编译成为目标文件后，再用连接程序将它们连接到一起，生成一个可执行文件；
 - 程序中调用了某个库文件中的子程序，需要将这个库文件和该程序生成的目标文件连接到一起，生成一个可执行文件；

连接

- 连接的作用有以下几个（续）：
 - 一个源程序编译后，得到了存有机机器码的目标文件，目标文件中的有些内容还不能直接用来生成可执行文件，连接程序将这此内容处理为最终的可执行信息。

所以，在只有一个源程序文件，而又不需要调用某个库中的子程序的情况下，也必须用连接程序对目标文件进行处理，生成可执行文件。
- 注意，对于连接的过程，可执行文件是我们要得到的最终结果。

4.2 以简化的方式进行编译和连接

- 我们编译、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。我们可以用一种较为简捷的方式进行编译、连接。

4.2 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
```

4.2 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>_
```


4.3 1.exe的执行

- 现在，终于将我们的第一个汇编程序加工成了一个可在操作系统下执行的程序文件。**1.exe**的执行情况：

```
C:\masm>1
```

```
C:\masm>
```

- 程序没有向显示器输出任何信息，只是做了一些将数据送入寄存器和加法的操作，因此屏幕上没有显示。

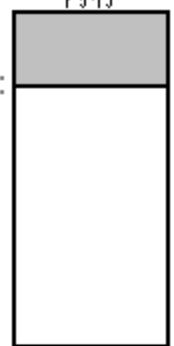



4.4 程序执行过程的跟踪

- 为了观察程序的运行过程，我们可以使用Debug。

```
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  NU UP EI PL NZ NA PO NC
12AE:0000 0000 B82301          MOV     AX,0123
-
```

- 可以看到，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。**1.exe** 中程序的机器码共有**15**个字节。

EXE文件中的程序的加载过程

<p>1</p>  <p>内存</p> <p>SA :</p>	<p>2</p>  <p>内存</p> <p>SA :</p> <p>PSP</p>	<p>3</p>  <p>内存</p> <p>SA :</p> <p>SA+10H:0</p> <p>PSP</p> <p>程序</p>	<p>4</p>  <p>内存</p> <p>SA :</p> <p>SA+10H:0</p> <p>PSP</p> <p>CS:IP</p> <p>程序</p>
<p>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；</p>	<p>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；</p> <p>（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。）</p>	<p>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；</p> <p>（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：</p> <p>空闲内存区：SA:0 PSP区：SA:0 程序区：SA+10H:0</p> <p>注意：PSP区和程序区虽然物理地址连续，却有不同段地址。）</p>	<p>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。</p>

4.4 程序执行过程的跟踪

- 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  NU 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
```

4.4 程序执行过程的跟踪

- 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，我们要用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  NU 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  NU UP EI PL NZ AC PO NC
1402:000D CD21          INT     21
-p

Program terminated normally
_
```

4.4 程序执行过程的跟踪

- int 21 执行后，显示 “Program terminated normally”，返回到Debug中。
- 表示程序正常结束。
- 注意，要使用P命令执行int 21。

■ 实验一