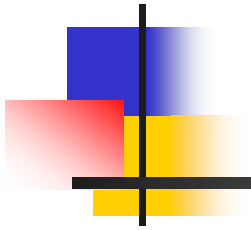
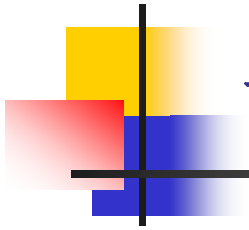


汇编语言程序格式



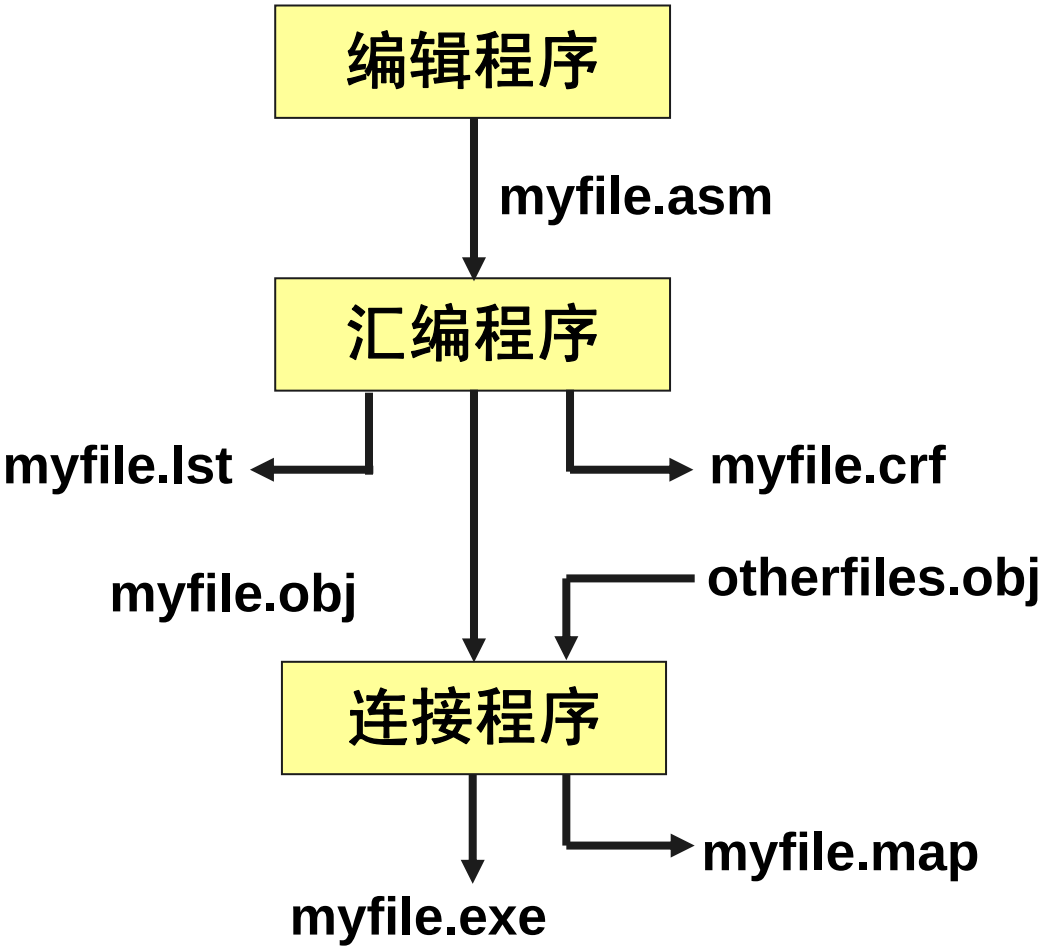
一个源程序从写出到执行的过程

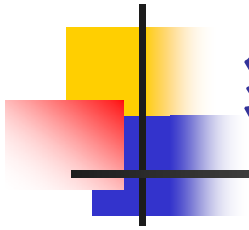
- 一个汇编语言程序从写出到最终执行的简要过程：

  编写--〉编译--〉连接--〉执行

- 演示

程序运行步骤及生成的文件



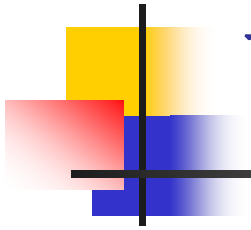


编写汇编源程序

- 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。

对源程序进行编译连接

- 使用汇编语言编译程序对源程序文件中的源程序进行编译，产生目标文件；再用连接程序对目标文件进行连接，生成可在操作系统中直接运行的可执行文件。



可执行文件

- 可执行文件中包含两部分内容：
  - 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）
  - 描述信息（比如：程序有多大、要占多少内存空间等）

执行可执行文件中的程序

- 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。

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
```

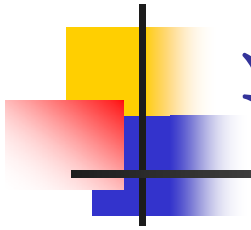
- 汇编指令
- 伪指令

  XXX segment

  XXX ends

  end

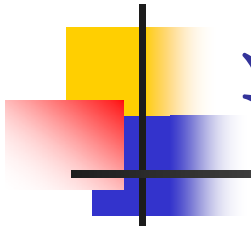
  assume



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。



源程序

- 伪指令

  没有对应的机器码的指令，最终不被**CPU**所执行。

- 谁来执行伪指令呢？

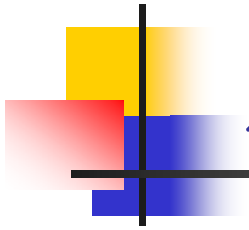
  由编译器来执行，编译器根据伪指令来进行相关的编译工作。

伪操作（伪指令）

伪操作是汇编程序对源程序进行汇编时处理的操作，完成处理器选择、存储模式定义、数据定义、存储器分配、指示程序开始结束等功能。

- 处理器选择伪操作
- 段定义伪操作
- 程序开始和结束伪操作
- 数据定义及存储器分配伪操作
- 表达式赋值伪操作
- 地址计数器与对准伪操作
- 基数控制伪操作

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

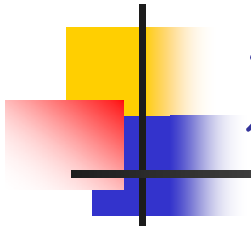


定义一个段

- **segment**和**ends**是成对使用的伪指令，写汇编程序时，必须要用到的一对伪指令。
- **segment**和**ends**
  - 功能：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，格式：

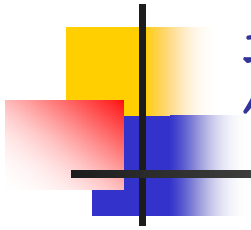
  段名 **segment**

  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。
- 一个有意义的汇编程序中至少要有一个段，这个段用来存放代码。

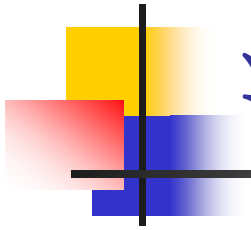


程序结束标记

- **end** 是一个汇编程序的结束标记，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：不要搞混了**end**和**ends**。

寄存器与段的关联假设

- **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。



源程序

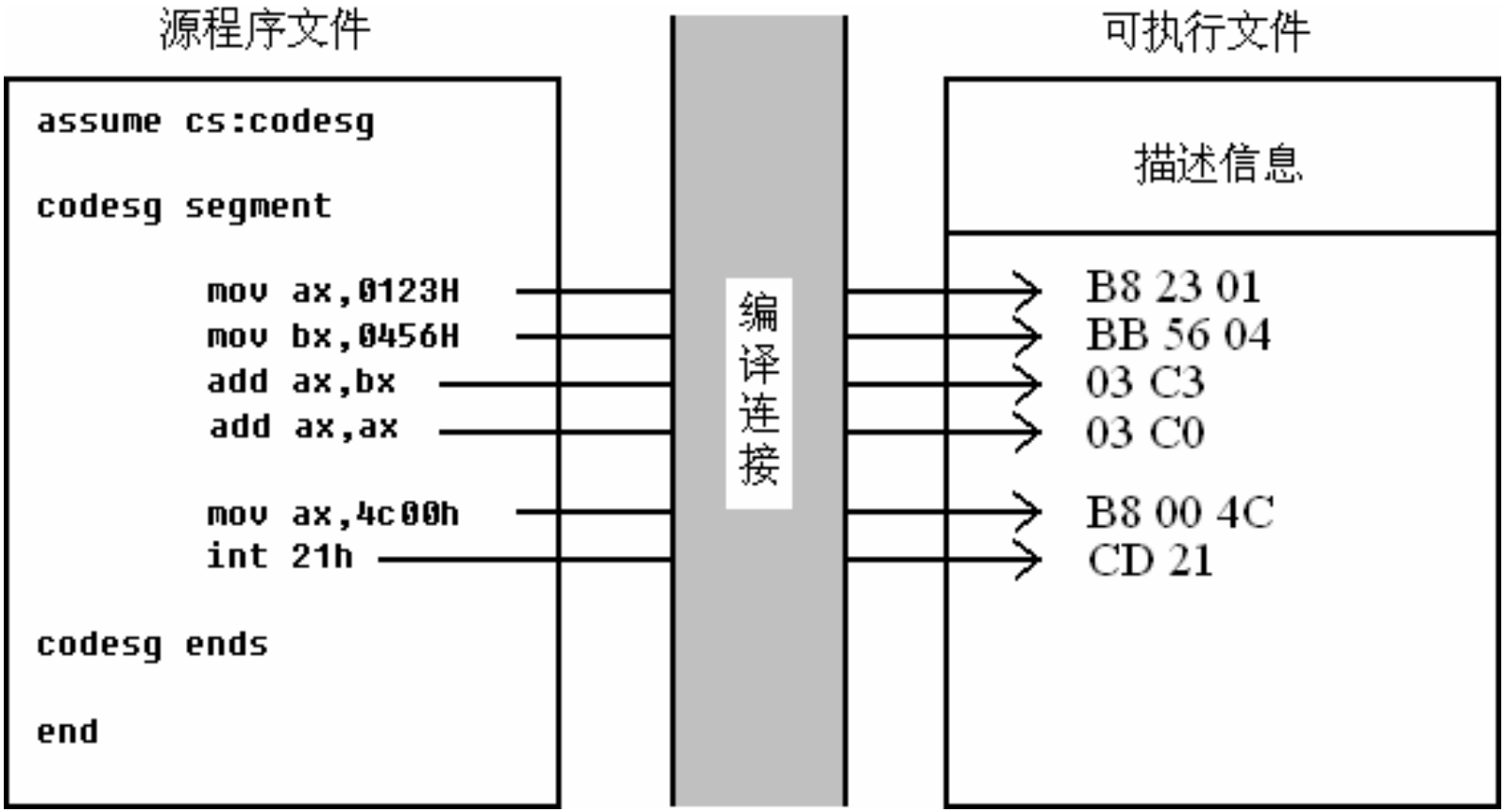
- 源程序中的“程序”
  - 汇编源程序：

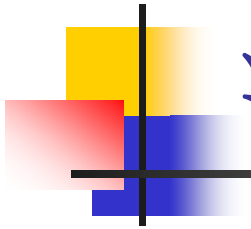
    伪指令　（编译器处理）

    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
- 图示

程序经编译连接后变为机器码





源程序

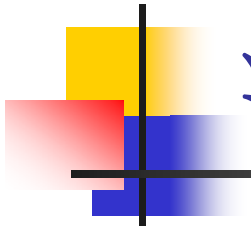
- 标号
  - 一个标号指代了一个地址。
  - **codesg**：放在**segment**的前面，作为一个段的名称，这个段的名称最终将被编译、连接程序处理为一个段的段地址。

源程序

- 程序的结构
  - 任务：编程运算2^3。
    - 定义一个段
    - 实现处理任务
    - 程序结束
    - 段与段寄存器关联

汇编程序

```
assume cs:abc
abc segment
        mov ax,2
        add ax,ax
        add ax,ax
abc ends
end
```



源程序

- 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。

- 几个和结束相关的内容

段结束、程序结束、程序返回

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

不能识别aume

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
```

无法知道abc段何处结束

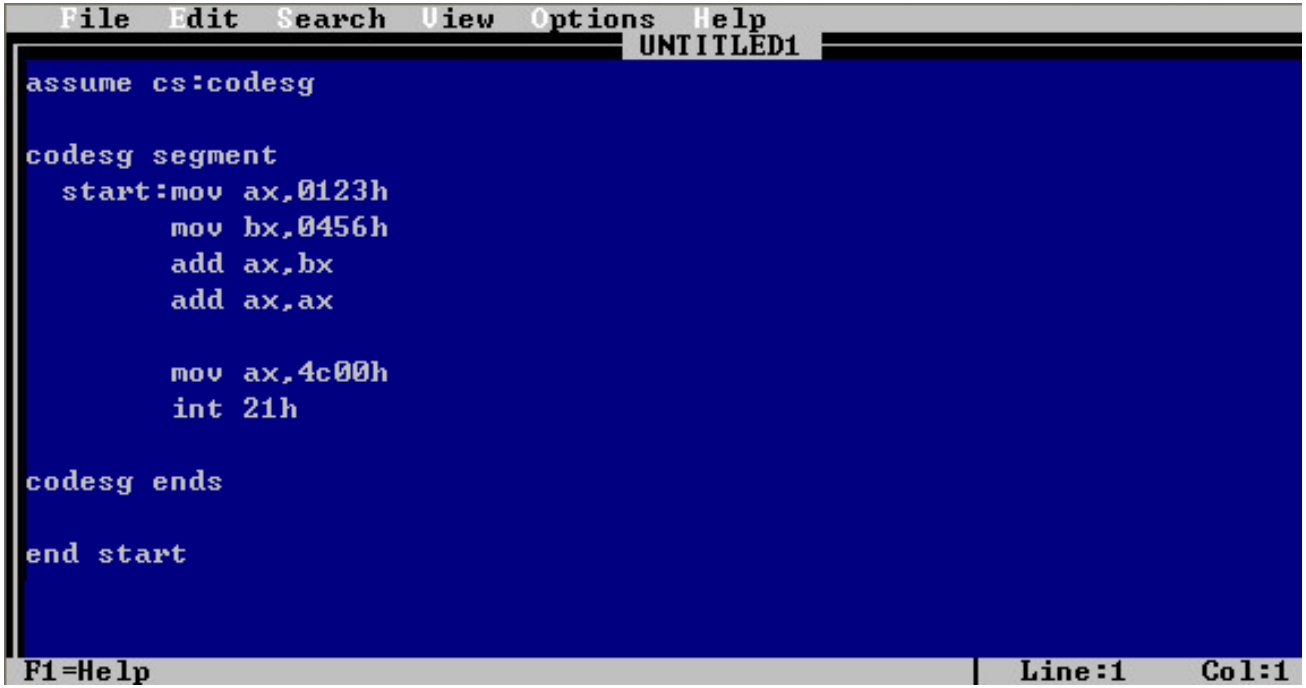
语法错误和逻辑错误

- 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
```

编辑源程序

- 进入**DOS**方式，运行**Edit**，在其中编辑程序（可避免全角字符问题），如下图所示：



编译

- 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.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

- 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。

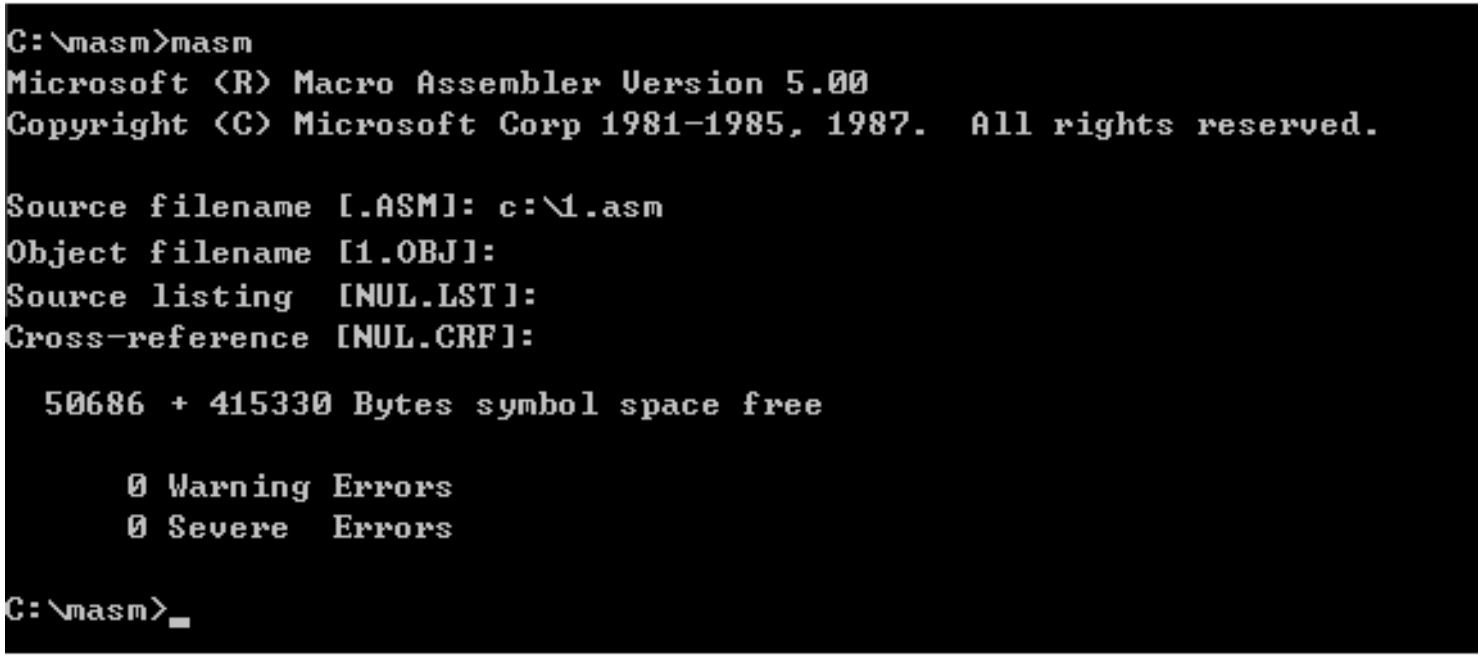
编译

- 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。

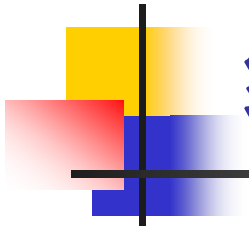
编译

- 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

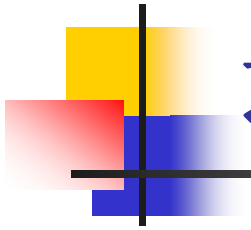


- 对源程序的编译结束，编译器输出的最后两行告诉我们这个源程序没有警告错误和必须要改正的错误。



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**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”**。

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]:
```

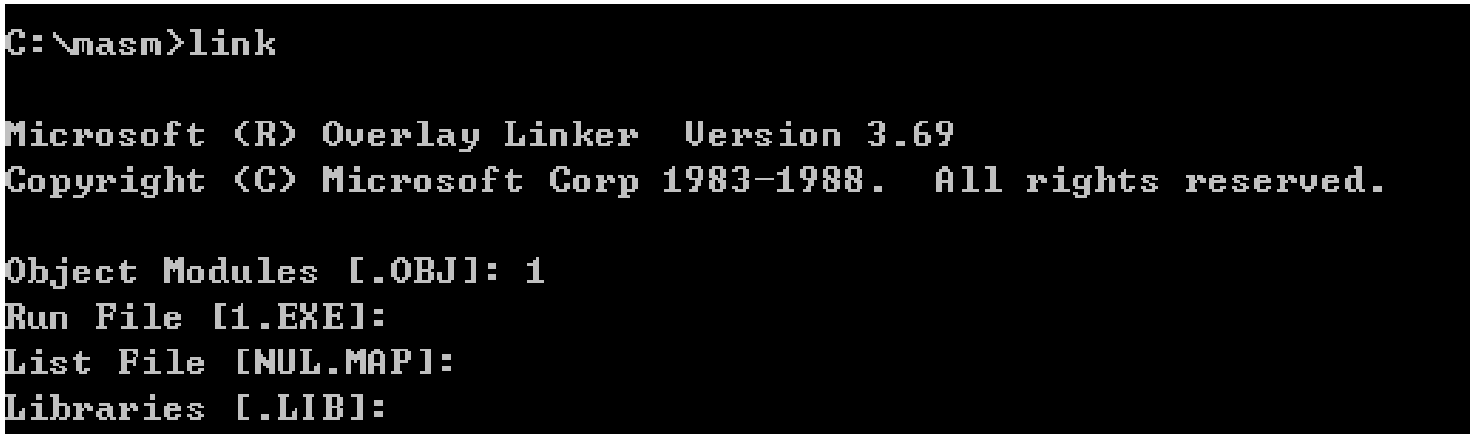
- 输入要连接的目标文件名后，按**Enter**键。
- 可执行文件是我们对一个程序进行连接要得到的最终结果。
- 连接程序默认要输出的可执行文件名为 **1.EXE**，所以可以不必再另行指定文件名。
- 我们直接按 **Enter** 键，使用连接程序设定的可执行文件名。

连接

```
C:\masm>link

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

Object Modules [.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

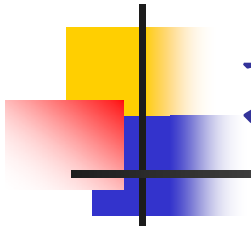


- 连接程序提示输入库文件的名称。
- 库文件里包含了一些可以调用的子程序，如果我们的程序中调用了某一个库文件中的子程序，就需要在连接的时候，将这个库文件和我们的目标文件连接到一起，生成可执行文件。
- 如果没有调用任何子程序，直接按**Enter**键即可。

连接

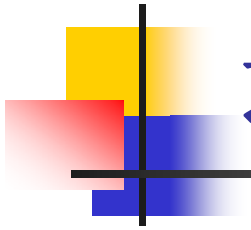
```
C:\masm>link

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。



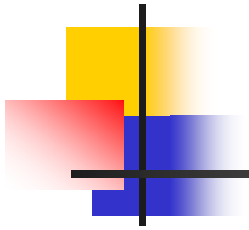
连接

- 我们用汇编语言编程，就要用到：编辑器（**Edit**）、编译器（**masm**）、连接器（**link**）、调试工具（**debug**）等所有工具，而这些工具都是在操作系统之上运行的程序，所以我们的学习过程必须在操作系统的环境中进行。



连接

- 连接的作用有以下几个：
  - 源程序很大，分为多个源程序文件，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；



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。

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
```

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>_
```

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：

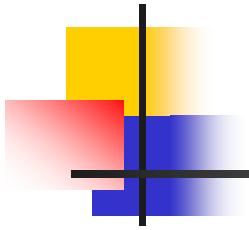
```
C:\masm>1

C:\masm>
```

- 程序到底运行没有？

  程序当然运行了，只是从屏幕上不可能看到任何运行结果。

定义数据指令

数据定义及存储器分配的伪操作



变量定义语句（简单格式）

格式：

变量名　**DB/DW/DD**　初始值 **[，.....]**

格式:

[变量] 助记符 操作数 [, 操作数 , …] [; 注释]

– 助记符(Mnemonic)说明伪操作的助记符,定义数据类型:

DB 定义字节,其后的每一个操作数占一个字节（8位）

DW 定义字（16位）

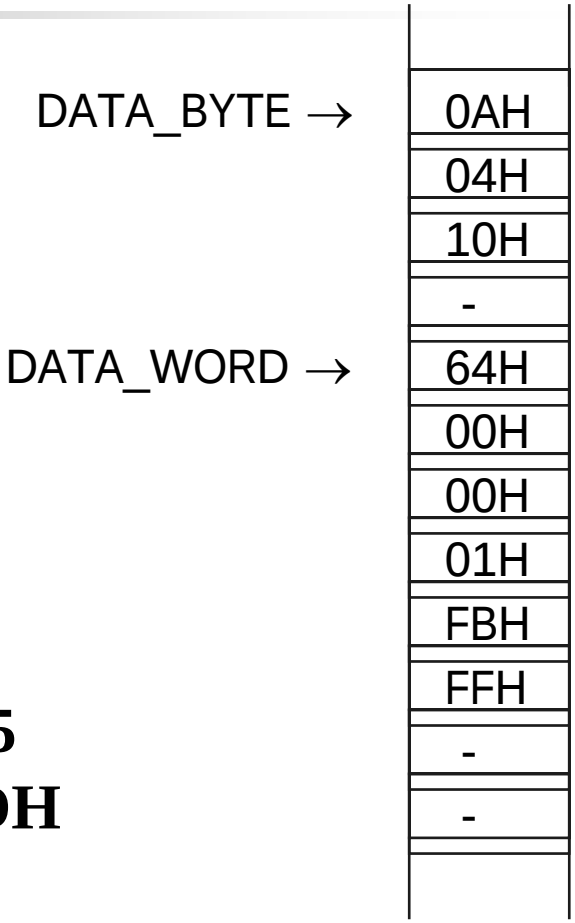
DD 定义双字,其后的每一个操作数占两个字（32位）

DQ 定义四个字,其后的每一个操作数占有四个字(64位)

DT 定义十个字节，形成压缩的BCD码

[变量] 助记符 操作数[,操作数,...][;注释]

助记符：**DB DW DD DF DQ DT**



–操作数可以是常数、或者表达式，

例如:

DATA-BYTE DB 10, 4, 10H

DATA-WORD DW 100, 100H, -5

DATA-DW DD 3*20, 0FFFDH

- **1、定义一组数据**

  例1： BUFF DW 1234H, 0ABCDH, 8EH

  DW –79DH, 7B6AH

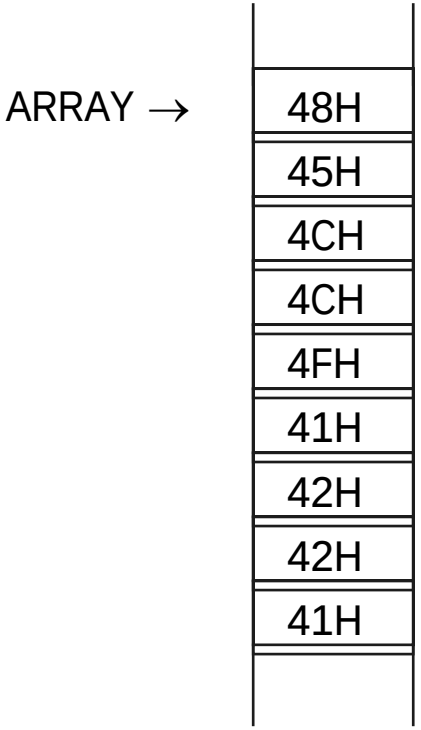
- **2、定义一串字符**

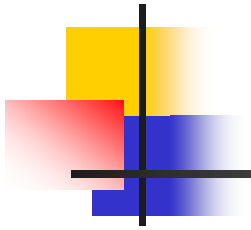
  例2： STR DB ‘ Welcome !’

- **3、定义保留存储单元，但不存入数据**

  例3： SUM DW ?, ?

```
ARRAY  DB  'HELLO'
       DB  'AB'
       DW  'AB'

PAR1  DW  100,200
PAR2  DW  300,400
ADDR_TABLE  DW  PAR1,PAR2
```





4、复制操作 DUP

- 复制操作符DUP（Duplication）可预置重复的数值

例4： ALL_ZERO DB 0，0，0，0，0

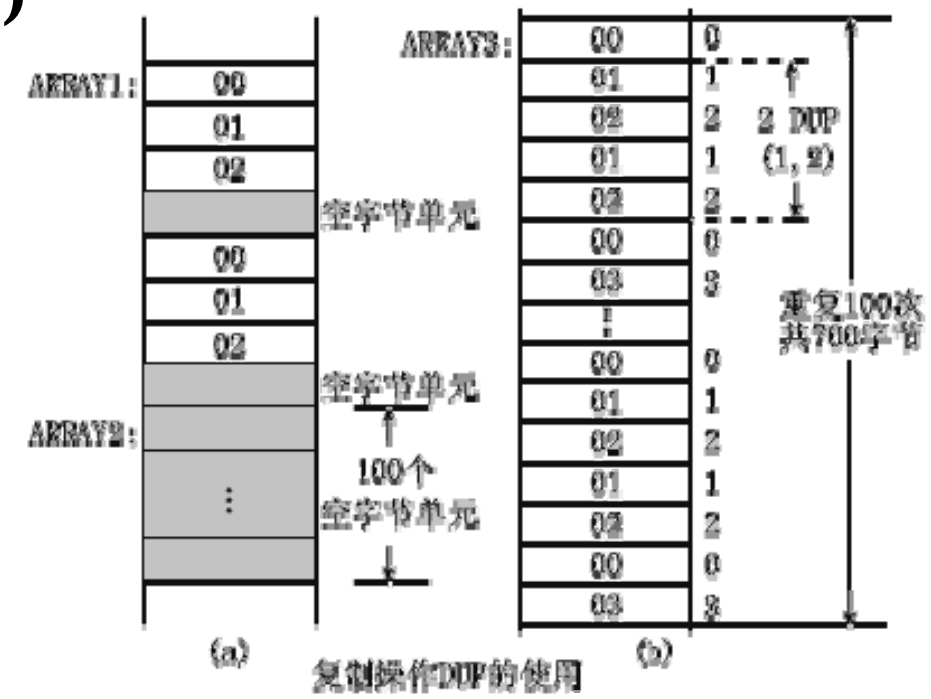
用复制操作可改为：

ALL_ZERO DB 5 DUP（0）

DUP 使用？来预留空单元

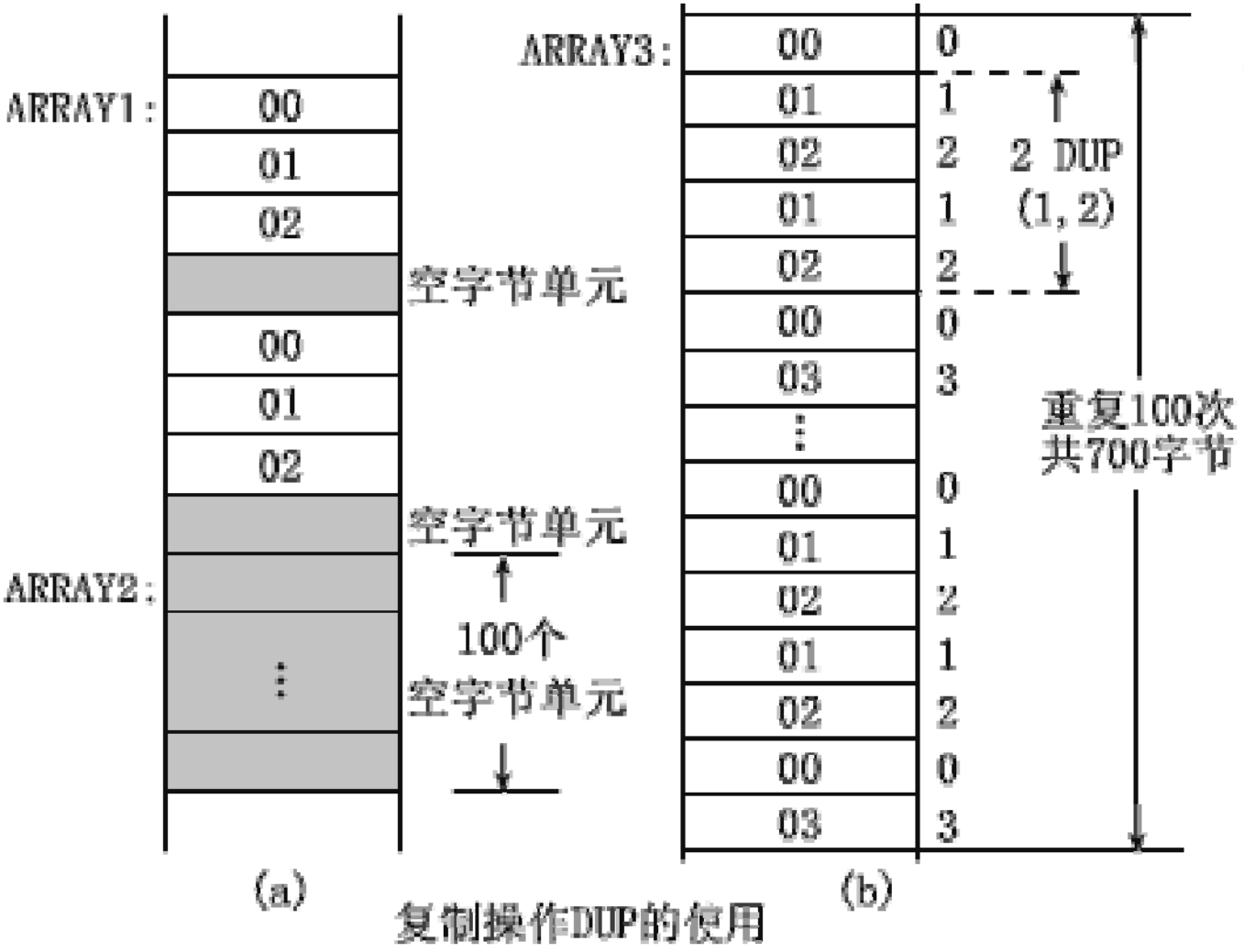
ARRAY1 DB 2 DUP(0,1,2,?)
ARRAY2 DB 100 DUP(?)

第1个语句表示把括号内的操作数0，1，2和？依次重复定义两次，而第2个语句表示要预留出100个空的字节单元。



ARRAY3 DB 100 DUP(0, 2 DUP(1,2), 0, 3)

存储情况如图



复制操作DUP的使用

DUP 嵌套使用

变量的类型属性:

例如：汇编可以用隐含的类型属性来确定某些指令是字指令还是字节指令。

```
OPER1   DB ？，？
OPER2   DW ？，？
MOV   OPER1，0
MOV   OPER2，0
```

例如：

```
OPER1  DB  1，2
OPER2  DW  1234H，5678H
MOV  AX，OPER1+1
MOV   AL，OPER2
```

例： **OPER1 DB ?, ?**
OPER2 DW ?, ?
…
MOV OPER1, 0 ; 字节指令
MOV OPER2, 0 ; 字指令

变量的类型属性：例如：汇编可以用隐含的类型属性来确定某些指令是字指令还是字节指令

例： **OPER1 DB 1, 2**
OPER2 DW 1234H, 5678H
…
MOV AX, OPER1+1 ×
MOV AL, OPER2 ×

类型属性不匹配

OPER1	01
	02
OPER2	34
	12
	78
	56

MOV AX, WORD PTR OPER1+1
MOV AL, BYTE PTR OPER2

(AX)=3402H (AL)=34H

不同属性量操作的解决方法：

类型　PTR　变量±常量表达式

上例中的：　**MOV AX，WORD PTR OPER1+1**

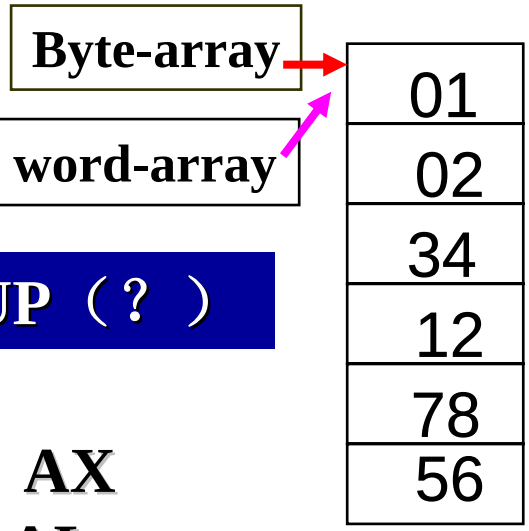
MOV AL，BYTE PTR OPER2

同一个空间（变量），可以具有不同的类型属性。除了用属性操作符给以定义之外，还可以用**LABLE**伪操作来定义

格式为：**name　lable　type**

例如：

```
Byte_Array  Lable  byte
WORD-ARRAY  DW   50 DUP（？）
```




```
MOV  WORD-ARRAY+2，AX
MOV  BYTE-ARRAY+2，AL
```

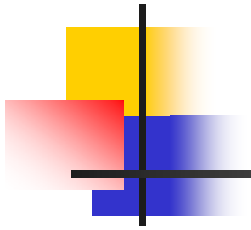
表达式赋值伪操作

1、等值语句

格式：符号名 EQU 表达式

功能：给表达式赋予一个名字

例：
(1) **PORT EQU 1234**
(2) **BUFF EQU PORT＋58**
(3) **MEM EQU DS:[BP＋20H]**
(4) **COUNT EQU CX**
(5) **ABC EQU AAA**



2、等号语句

格式：　　NUM＝34

……

NUM＝34＋1

表达式赋值伪操作EQU

格式：变量名 EQU 表达式

如：
```
CONSTANT  EQU  256
DATA      EQU  HEIGHT+12
ALPHA     EQU  7
BETA      EQU  ALPHA-2
ADDR      EQU  VAR+BETA
B         EQU  [BP+8]
```

注意：EQU伪操作不允许重复定义

而=伪操作可以

如：
```
AA=8
AA=AA+3
```

地址计数器$

地址计数器用来保存正在汇编的指令的地址。
地址计数器的值可用$来表示。
可以在程序中直接引用。常出现两种情况：

如：

在指令中出现时候：

JNE $+6 （JNE指令的首地址加上6）

指令中表示该指令的首字节地址

如：

数据定义时候：

ARRAY DW 1，2， $+4，3，4 ，$+4

数据定义时表示即将分配出去的字节的地址

假设**ARRAY** 分配的地址为**0074**，则存储图如右表示；

01	0074
00	0075
02	0076
00	0077
7C	0078
00	0079
03	007A
00	007B
04	007C
00	007D
82	007E
00	007F
…	…

对准伪操作

ORG 伪操作使下一个字节的地址成为常数表达式的值

如： **EE SEGMENT**

```
ORG 20
AA  DW 2，4
ORG  50
BB   DB ？
```

AA的偏移地址值为：14H

BB的偏移地址值为：32H

ORG 伪操作：

```
SEG1   SEGMENT
       ORG   10
       VAR1  DW  1234H
       ORG   20
       VAR2  DW  5678H
       ORG   $+8
       VAR3  DW  1357H
SEG1   ENDS
```

```
BUFFER  LABEL  BYTE
       ORG    $+8
```

```
BUFFER  DB  8 DUP (?)
```

请问：
VAR1、 VAR2、 VAR3
BUFFER的偏移地址值为?