

# 汇编语言程序设计

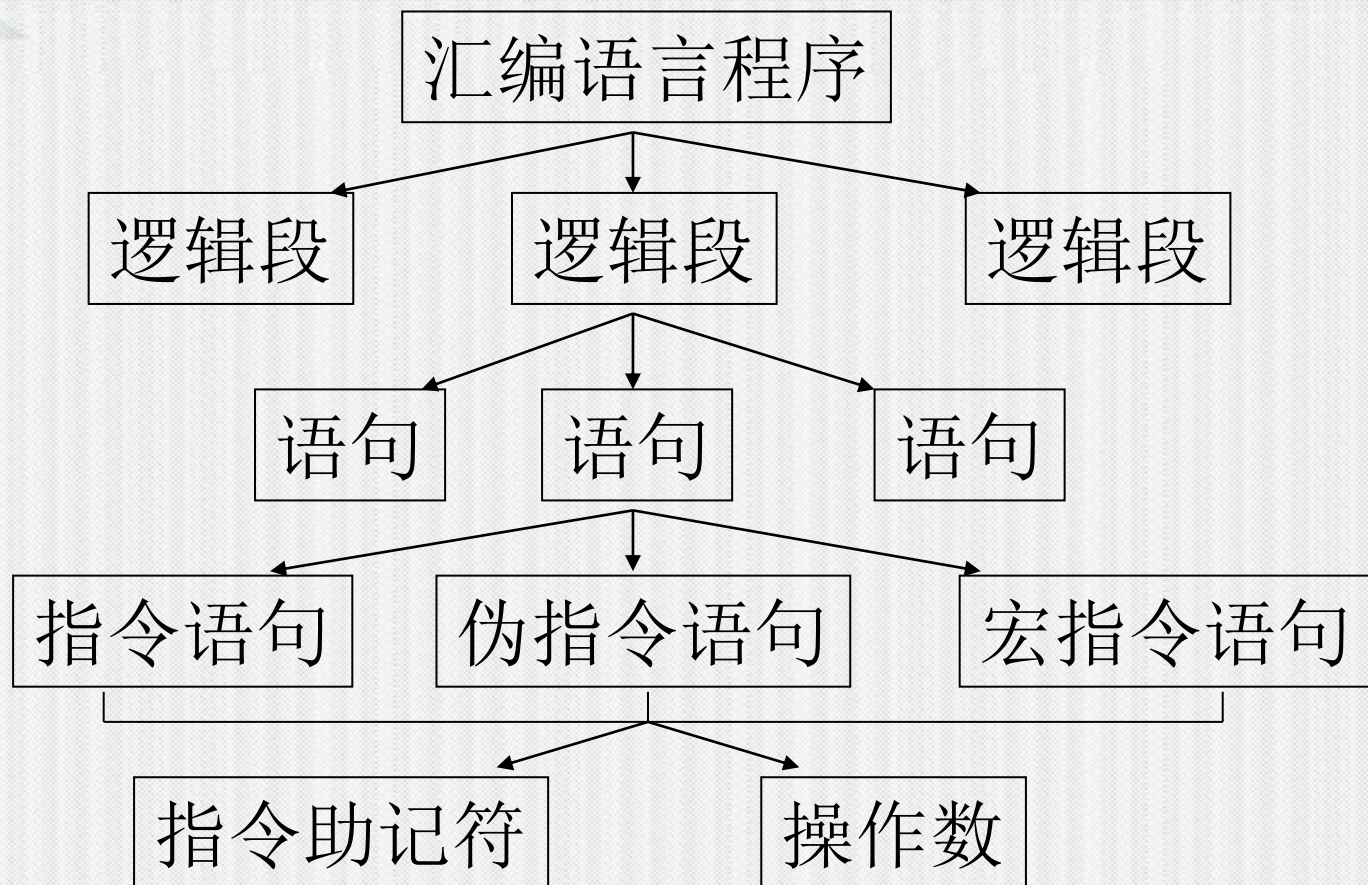
## *Assembly Language Programming*

### 第三章

## 汇编语言程序格式



# 3.1 汇编语言程序格式





# 1 逻辑段

♥ 汇编语言源程序由一个或多个逻辑段组成。

- ❖ 一个程序中可以有几个同一类型的逻辑段。
- ❖ 必须至少有一个代码段。

[注] 源程序分段的目的在于程序结构清晰、便于内存分配，寻址方便，一个源程序需要设置几个段应根据具体问题来定。



# 1 逻辑段

- ♥ 程序段（代码段）——主要由指令语句组成，完成源程序的功能。
- ♥ 数据段——定义数据及符号的伪指令组成。
- ♥ 附加段——定义数据及符号的伪指令组成。
- ♥ 堆栈段——定义堆栈伪指令组成。



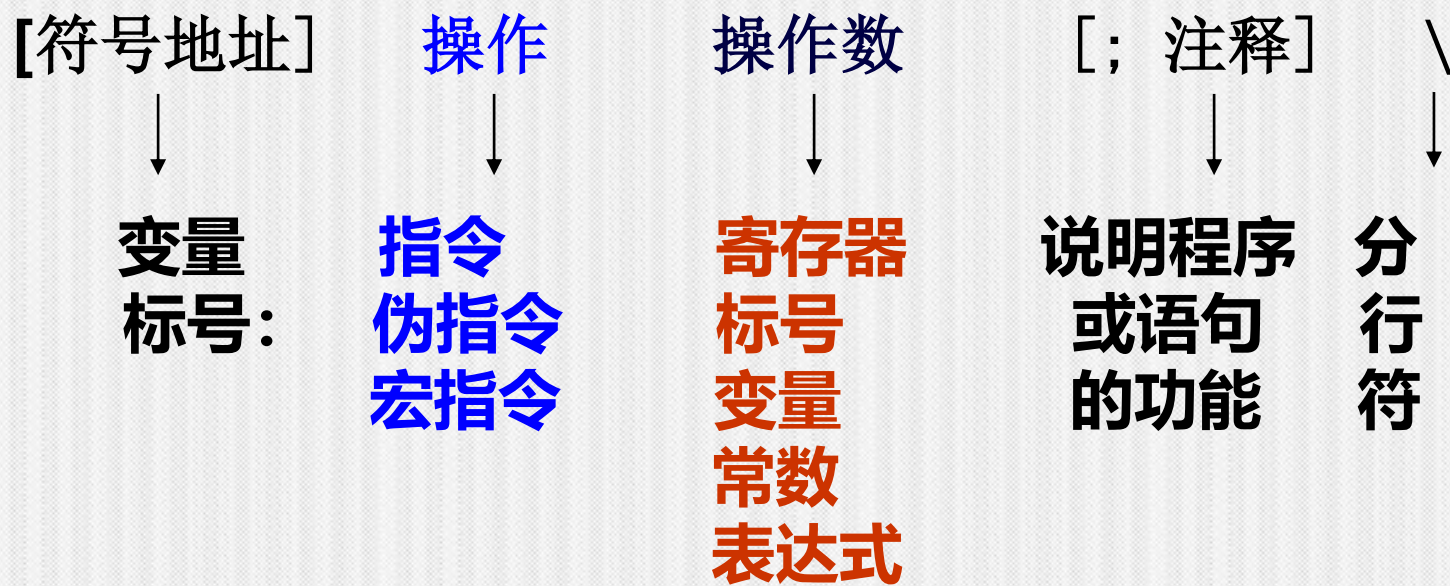
# 段定义伪操作

段名    segment    定位 组合 段字 '类别'  
         ...        ;语句序列  
段名    ends

- ♥ 完整段定义由SEGMENT和ENDS这一对伪指令实现，SEGMENT伪指令定义一个逻辑段的开始，ENDS伪指令表示一个段的结束
- ♥ 如果不指定，则采用默认参数；但如果指定，注意要按照上列次序
- ♥ 段名对外表现为立即数，为该段的段地址



## 2 语句格式





## 3.2 常数和表达式

♥ 常数（常量）表示一个固定的数值

- (1) **十进制常数**：以字母D或d结尾或缺省
- (2) **十六进制常数**：以字母H或h结尾，以字母A~F开头的十六进制数，前面要用0表达。
- (3) **二进制常数**：由0或1两个数字组成，以字母B或b结尾
- (4) **八进制常数**：以字母Q或q结尾
- (5) **字符串常数**：字符的ASCII码，如'abc'



## (6) 符号常数 (常量)

### ❖ 等价EQU伪指令

符号名 EQU 数值表达式

符号名 EQU <字符串>

### ❖ 等号=伪指令

符号名 = 数值表达式

### ❖ =和EQU区别:

前者可多次使用, 后者只能使用一次



# 数值表达式

♥ 数值表达式——用常数、符号常数和算术、逻辑、关系运算符组成的表达式。  
如：  $(75 * 2 + X) / Y$



# 表达式

- ♥ 汇编程序在汇编过程中计算表达式，最终得到一个数值（常数或地址）
- ♥ 程序运行之前，就已经计算出了表达式；所以，程序运行速度没有变慢，但增强程序的可读性



## 3.3 变量和标号

### ♥ 变量

- ❖ 变量——是指数据单元的符号地址。
- ❖ 变量的书写格式：不能使用系统保留字、不能以数字开头
- ❖ 变量的定义：用数据定义伪指令来定义

### ♥ 标号

- ❖ 标号——一个指令单元的符号地址。
- ❖ 标号的书写格式：不能使用系统保留字、不能以数字开头
- ❖ 标号的定义：在一条指令语句前输入标号且用 “:” 隔开，即定义了该标号；或用Label来定义



# 1 变量定义伪指令

♥ 格式:

❖ [变量] DB / DW / DD / DF / DQ / DT 操作数1, ..., 操作数n

♥ 功能:

- ❖ 定义变量;
- ❖ 在内存中分配一组存储单元;
- ❖ 并对单元进行初始化。

♥ 分类

- ❖ DB: 用来定义字节, 其后每个操作数占用一个字节。
- ❖ DW: 用来定义字, 其后每个操作数占用一个字。
- ❖ DD: 用来定义双字, 其后每个操作数占用两个字。
- ❖ DQ: 用来定义四个字, 其后每个操作数占用四个字。
- ❖ DT: 用来定义十个字节, 其后每个操作数占用十个字节。



# 操作数

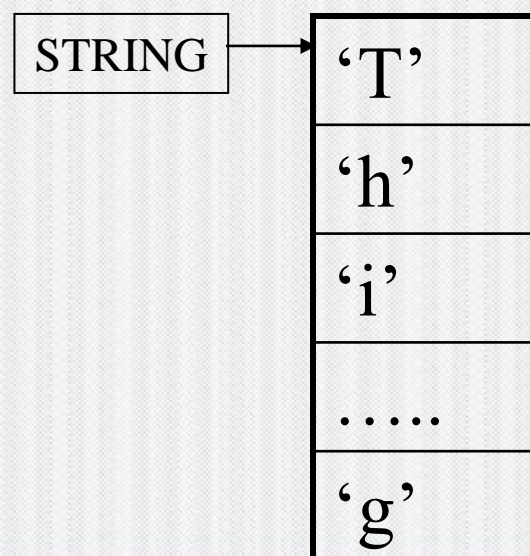
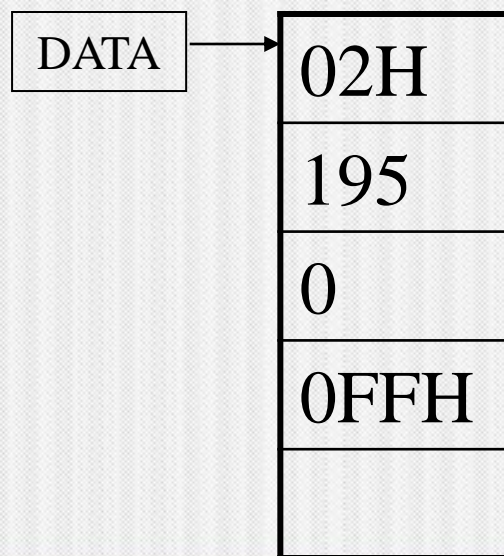
♥ 常数或一组常数或数值表达式;

❖ 例如: DATA DB 2, 100\*2-5, 0, -1

♥ 一组字符串

❖ 如: STRING DB 'This is a string', 其数值是每个字符对应的ASCII码的值

❖ 注意比较DB "ab"和 DW "ab"





# 操作数

## ♥ 变量名或标号名

- ❖ ADDR1 DW BLOCK ; BLOCK偏址放在ADDR1单元
- ❖ ADDR2 DD BLOCK ; BLOCK的偏址和段址依次存放在ADDR2四字节单元中。

## ♥ 一组 “?”

- ❖ 只分配空间，不进行初始化
- ❖ 例如：BLOCK DW ?,? ; 分配两个字，但为随机值



# 操作数

## ♥ 重复DUP语句

- ❖ 格式：重复数  $n$  DUP (重复内容)
- ❖ 功能：将DUP后的内容重复定义 $n$ 次。
- ❖ 例如：BLOCK DB 3 DUP (0,1,-1)
- ❖ 如同：BLOCK DB 0,1,-1, 0,1,-1, 0,1,-1
- ❖ [注] DUP语句中可以包含DUP语句。

BLOCK

00H

01H

0FFH

00H

01H

0FFH

00H

01H

0FFH



# 起始地址和对准语句

## ♥ ORG

- ❖ 格式：ORG 表达式
- ❖ 功能：指定随后指令或者定义数据的偏移地址
- ❖ 说明：
  - “ORG”伪指令可设置程序段、数据段任何位置。
  - 若程序中没有设置“ORG”语句，一般情况每个逻辑的起始地址为0000H。



# 起始地址和对准语句

## ♥ EVEN

- ❖ 格式: EVEN
- ❖ 功能: 偶地址对齐指令。若当前地址是奇数, 则加1;

## ♥ ALIGN

- ❖ 格式: ALIGN  $n$
- ❖ 功能: 使随后的数据或者指令起始于 $n(2,4,8\dots)$ 的倍数地址



# NOTICE

♥ 汇编语言强类型!

♥ 变量有类型!

`BUFFER DW 1234H`

❖ `MOV AL, BUFFER`

❖ `MOV AL, BYTE PTR BUFFER`

❖ `MOV AX, BUFFER`



# 地址表达式

- ♥ 地址表达式——由“变量、标号、“+”、“-”数值表达式组成。
- ❖ 如：DATA+5;
- ❖ [注] 含有变量的地址表达式其类型与该变量一致，如VARY + 4与VARY类型一样(VARY[BX]亦然);
- ❖ 地址表达式可以相减，不能相加。



## 2 变量和标号属性

### ♥ 变量的属性:

- ❖ 段属性——变量的段地址
- ❖ 偏移属性——变量的偏移地址
- ❖ 类型属性——变量所指单元的类型，字节变量、字变量、双字变量等

### ♥ 标号的属性:

- ❖ 段属性——是指定义标号所在段的段地址。
- ❖ 偏移属性——是指定义标号处到段地址的距离。
- ❖ 类型属性——NEAR型和FAR型。



# 地址操作符

## ♥ SEG 运算符

- ❖ 格式: SEG 变量或标号
- ❖ 功能: 分离出其后变量或标号所在段的段首址。
- ❖ MOV AX, SEG ARR
- ❖ MOV DS, AX

## ♥ OFFSET运算符

- ❖ 格式: OFFSET 变量或标号
- ❖ 功能: 分离出其后变量或标号的偏移地址。
- ❖ MOV BX, OFFSET BUF



# 类型操作符

## ♥ 类型名 PTR

- ❖ 功能：指定类型 例：MOV WORD PTR [BX], 5

## ♥ THIS 类型

- ❖ 功能：指定随后地址，具有指定类型
- ❖ 例：TA EQU THIS BYTE  
TB DW 100 DUP (?)



# 类型操作符

## ♥ TYPE运算符

- ❖ 格式：TYPE 变量或标号
- ❖ 功能：分离出其后变量或标号的类型。

变量	类型	类型值
	BYTE	1
	WORD	2
	DWORD	4



# 类型操作符

## ♥ LENGTHOF运算符

- ❖ 格式: LENGTHOF 变量
- ❖ 功能: 取出变量所含的数据存储单元个数。

## ♥ SIZEOF运算符

- ❖ 格式: SIZEOF 变量
- ❖ 功能: 取出变量所含的数据存储区大小。
- ❖ 其返回值为: LENGTHOF 变量 \* TYPE 变量



## 3.4 汇编程序编写

- ♥ 源程序分别用两种格式书写
  - ❖ 第一种格式从MASM 5.0开始支持
    - ❖ 简化段定义格式
  - ❖ 第二种格式MASM 5.0以前就具有
    - ❖ 完整段定义格式



# 典型完整段定义格式

## MASM 5.x支持

;在数据段定义数据

stack

stack  
data

data

code

start:

code

...

ends

segment

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

mov ax, data

mov ds, ax

...

mov ah, 4ch

int 21h

...

ends

end start

;在代码段填入指令序列

;子程序代码



# (1) 段类型说明伪操作

- ♥ 在代码段开始必须用ASSUME伪操作声明段和寄存器之间的关系，格式为：

**ASSUME 段寄存器：段名 [,段寄存器名：段名, ...]**

- ♥ 通知MASM，建立段寄存器与段的缺省关系；在需要时自动插入段超越前缀。这是ASSUME伪指令的主要功能。
- ♥ 实际上，数据段之所以成为数据段，是由于DS指向它。由于程序运行时可以改变DS的值，使得任何段都可以成为数据段。



DATA1 SEGMENT

X DB 1

DATA1 ENDS

DATA2 SEGMENT

Y DB 2

DATA2 ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA1 ,ES:DATA2

START:

MOV AX,DATA1

MOV DS,AX

MOV AX,DATA2

MOV ES,AX

MOV AL,X

MOV AH,Y

MOV AH,4CH

INT 21H

CODE ENDS

END START

MOV AL,DS:[0000]

MOV AH,ES:[0000]



# NOTICE

- ♥ ASSUME语句位于在程序段开始位置
- ♥ 在ASSUME语句中并没有给段寄存器赋值。



# NOTICE

♥ DS、ES的初值必须在程序中设置：

- ❖ `MOV AX,<段名>`
- ❖ `MOV DS/ES/SS,AX`

♥ CS与IP的初值不能在程序中显式设置，由系统自动设置为END后指定的起始地址

♥ 为SS与SP的初值

- ❖ 可在程序中显式设置：SS同上，SP用 `MOV SP,St_TOP`
- ❖ 堆栈段定义时给出了属性STACK，则由系统自动设置。
- ❖ 其他，则由系统指定堆栈，编译时给出警告错误



## (2) 汇编结束伪指令

♥ 格式:

❖ END [标号]

♥ 功能:

- ❖ 指示源码到此结束;
- ❖ 指示程序开始执行点 (标号处)。



## 简化段定义格式 MASM 6.x支持

;example.asm

.model small

.stack

.data

...

;在数据段定义数据

.code

.startup

...

;在代码段填入指令序列

.exit 0

...

;子程序代码

end



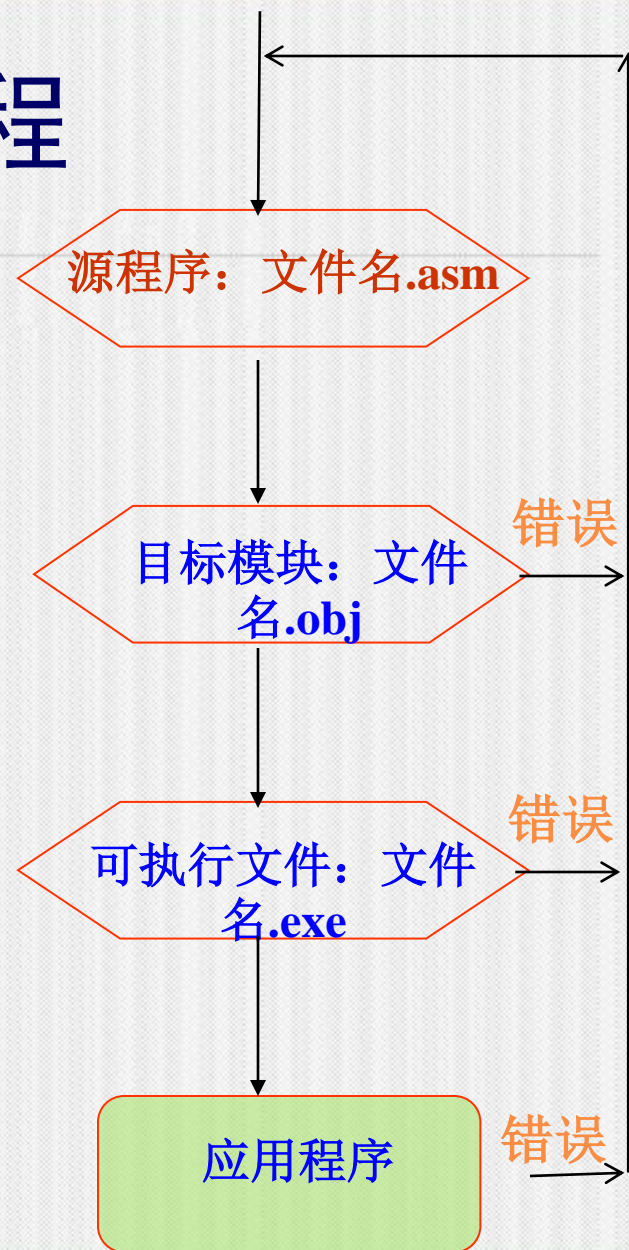
## 3.5 汇编语言的上机过程

汇编语言程序的上机过程

- 1、编写源程序
- 2、将源程序编译为目标程序
- 3、把目标程序连接为 DOS 可执行程序

汇编程序的主要功能是：

- 1、报告源程序中的语法错误
- 2、形成目标程序





# 建立汇编语言的工作环境- MASM 5.x

MASM.EXE

LINK.EXE

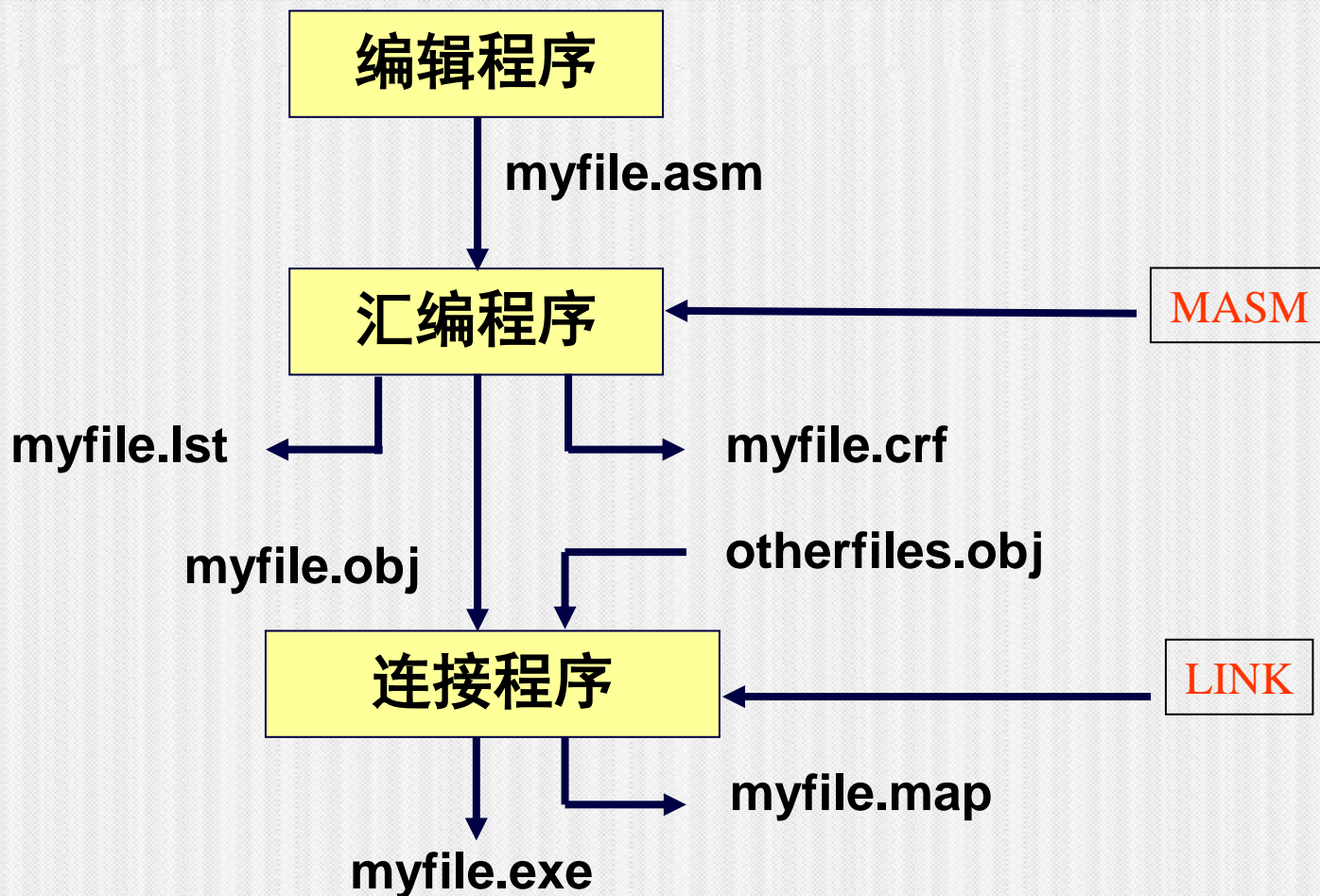
DEBUG.COM

使用： MASM myfile.asm ;编译

LINK myfile.obj ;连接



# 编译和连接





# 建立汇编语言的工作环境- MASM 6.1x

ML.EXE

LINK.EXE

CV.EXE

使用： ML myfile.asm ;编译连接

ML /c myfile.asm ;只编译

ML /Zi myfile.asm ;加入调试信息



# 调 试

## ♥ 程序错误

- ❖ 语法错误
- ❖ 逻辑错误

## ♥ 用debug/CV调试程序

- ❖ Debug \*.exe
- ❖ -U
  - MOV AX, \*\*\*\*H
- ❖ -G
- ❖ -D \*\*\*\*:0000