

# 汇 编 语 言

与

# 接 口 技 术

## 第4章 80X86汇编语言程序设计

**主编：王让定 朱莹**

宁波大学信息学院





# 本章主要内容



中断系统的功能

-  汇编语言分支程序设计
-  汇编语言循环程序设计
-  串处理程序设计
-  子程序设计
-  高级汇编语言程序设计
-  汇编语言与C语言混合编程



# MASM宏汇编语句结构以及开发过程

PART 01





# 汇编语言程序的特点与应用场景

## 1. 汇编语言的特点

- 1) 汇编语言与机器关系密切；
- 2) 汇编语言程序效率高（时间：运行速度快，空间：目标程序短）；
- 3) 编写汇编语言源程序繁琐；
- 4) 汇编语言程序调试困难，维护、交流和移植程序更困难。

## 2. 汇编语言的应用场景

- 1) 对软件的执行时间或存储容量有较高要求的场合。例如：系统程序的关键核心，智能化仪器仪表的控制系统，实时控制系统等。
- 2) 需要提高大型软件性能的场合。通常把大型软件中执行频率高的子程序(过程)用汇编语言编写，然后把它们与其他程序一起连接。
- 3) 软件与硬件关系密切，软件要有直接和有效控制硬件的场合。如设备驱动程序等。
- 4) 没有合适的高级语言的场合。



## 汇编语言的应用案例

- HQ-9主控计算机研制

- HQ-9是第一种国产第三代中高空中远程防空导弹，也是我国第一种采用垂直发射技术的地空导弹；
- HQ-9弹载计算机在研制初期，受国际环境制约，并没有适用的高级语言编译开发环境，核心编码使用汇编语言实现。科研人员克服困难，编制调试了超过5万行控制程序代码；
- 同期对比：Protel for windows 进行电路EDA设计；



# 汇编语言程序的语句类型和格式

## 1. 语句的类型

- 硬指令语句 (指令性语句)

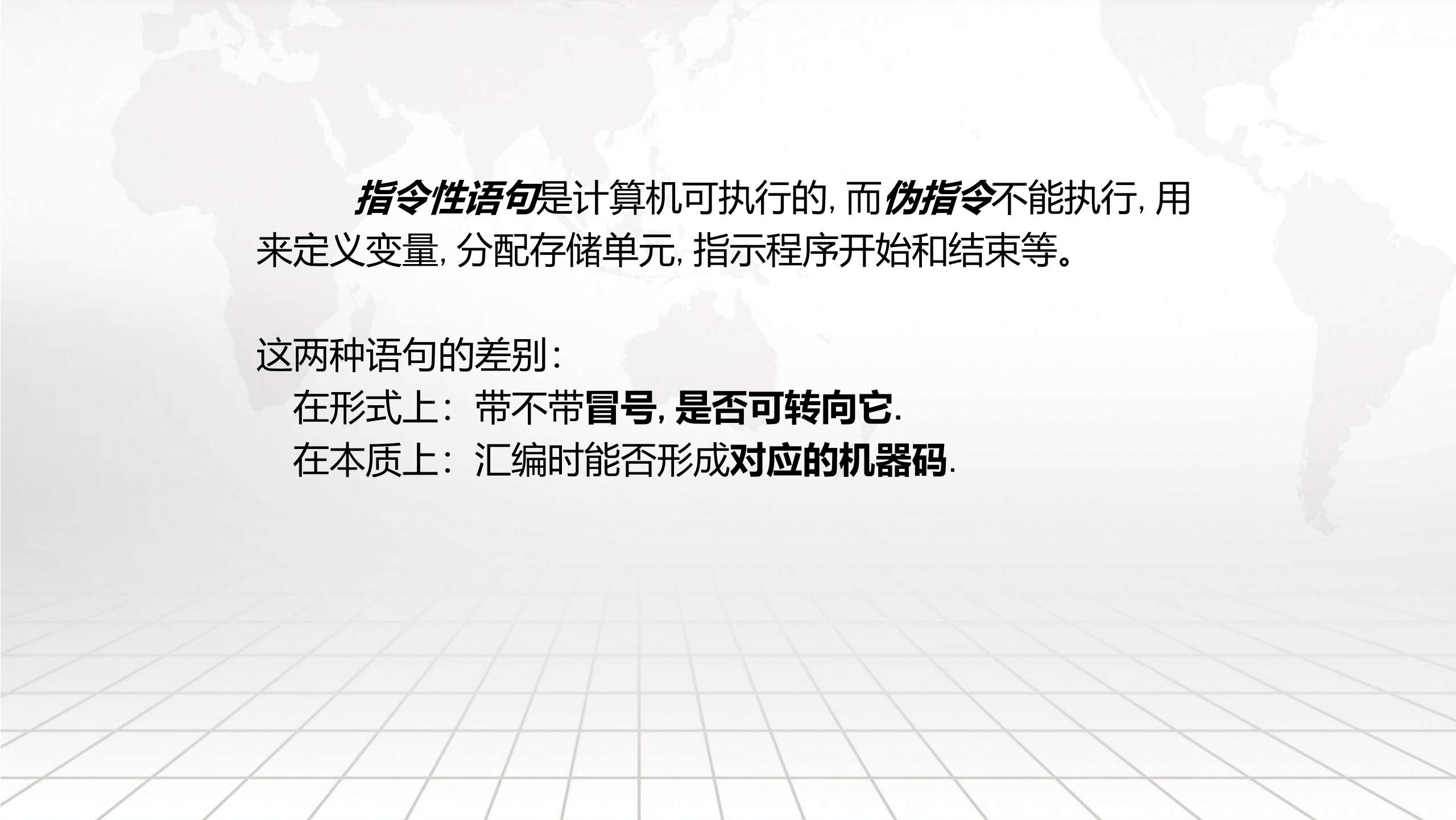
是指能产生目标代码，CPU可以执行的，能完成特定功能的语句，它主要由CPU指令组成。

- 伪指令语句 (指示性语句)

是一种不产生目标代码的语句，它仅仅在汇编过程中告诉汇编程序应如何汇编。

- 定义变量，定义过程，给变量分配存储单元，给数字或表达式命名等。所以伪指令语句是汇编程序在汇编时用的，不产生机器码。

- 宏指令语句 它是一个指令序列，汇编时凡有宏指令语句的地方都将用相应的指令序列的目标代码插入。



**指令性语句**是计算机可执行的, 而**伪指令**不能执行, 用来定义变量, 分配存储单元, 指示程序开始和结束等。

这两种语句的差别:

在形式上: 带不带**冒号**, 是否可转向它.

在本质上: 汇编时能否形成**对应的机器码**.



## 2. 语句的格式

- 汇编语言源程序由语句序列构成。
- 汇编语言源程序中的每条语句一般占一行，每行不超过132个字符（MASM 6.0开始可以是512个字符），
  - **执行性语句**
    - 标号：硬指令助记符 操作数，操作数；注释
  - **说明性语句**
    - 名字 伪指令助记符 参数，参数…；注释



## 汇编语言的程序格式

- 在实地址模式和虚拟8086模式下，按照逻辑段组织程序，具有代码段、数据段、附加段和堆栈段。
- 一个汇编语言源程序可以包含若干个代码段、数据段、堆栈段或附加段，段与段之间的顺序可随意排列。
- 需独立运行的程序必须包含一个代码段，并指示程序执行的起始位置，一个程序只有一个起始位置。
- 所有的可执行性语句必须位于某一个代码段内，说明性语句可根据需要位于任一段内

- 两种格式书写
- 第一种格式是从MASM 5.0开始支持的简化段定义格式（但其中的两个指令 `.STARTUP`和 `.EXIT`是6.0版本才引入的）。
- 第二种格式是MASM 5.0以前版本就具有的完整段定义格式。

· **例** 简化段定义的源程序格式

```
MODEL    SMALL    ; 定义程序的存储模式, 小型程序一般采用小模式SMALL
. STACK                                ; 定义堆栈段
. DATA                                ; 定义数据段
STRING    DB ' Hello, Everybody !' , 0DH, 0AH, ' $'
        ; 在数据段定义要显示的字符串
. CODE                                ; 定义代码段
. STARTUP ; 说明程序起始位置, 并建立DS、SS内容 (注1)
        MOV    DX, OFFSET STRING    ; 指定字符串在数据段的偏移地址
        MOV    AH, 9
        INT     21H    ; 利用DOS功能调用显示信息
. EXIT 0                                ; 程序结束点, 返回DOS (注2)
        END                                ; 汇编结束 (注3)
```

## 例 完整段定义的源程序格式

```
STACK    SEGMENT STACK           ; 定义堆栈段STACK
        DW          512 DUP (?)   ; 堆栈段的大小是1024
        字节 (512字) 空间
STACK    ENDS                     ; 堆栈段结束
DATA     SEGMENT                  ; 定义数据段DATA
STRING   DB  'Hello, Everybody ! ' , 0DH, 0AH, ' $'
; 在数据段定义要显示的字符串
DATA     ENDS                     ; 数据段结束
CODE     SEGMENT  'CODE'          ; 定义代码段CODE
        ASSUME  CS: CODE, DS: DATA, SS: STACK; 确定CS、DS、
        SS指向的逻辑段
        START: MOV AX, DATA      ; 设置数据段的段地址DS
                MOV DS, AX
```



MOV DX, OFFSET STRING  
; 利用功能调用显示信息

MOV AH, 9  
INT 21H

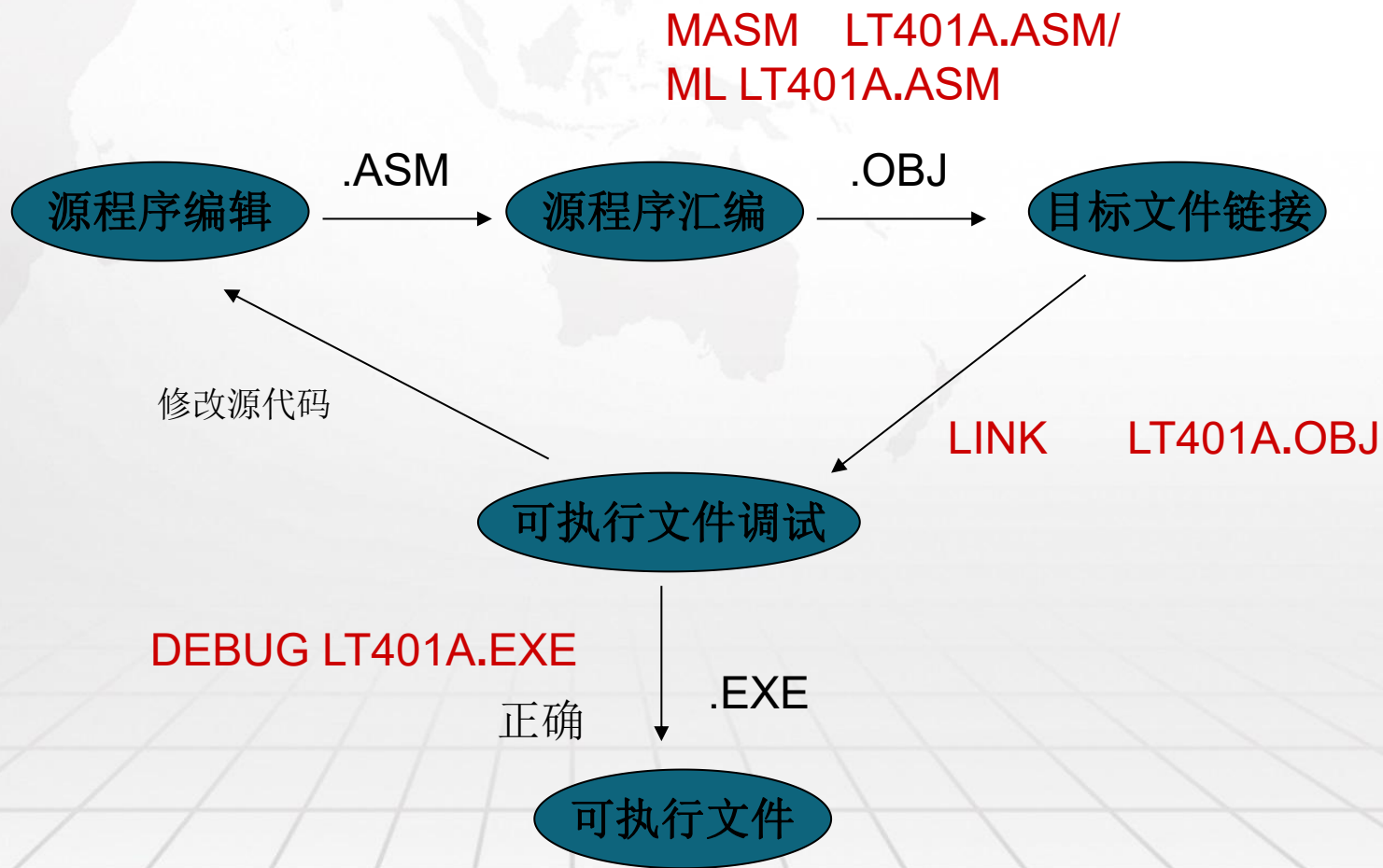
MOV AX, 4C00H ; 利用系统功能调用返回DOS  
INT 21H

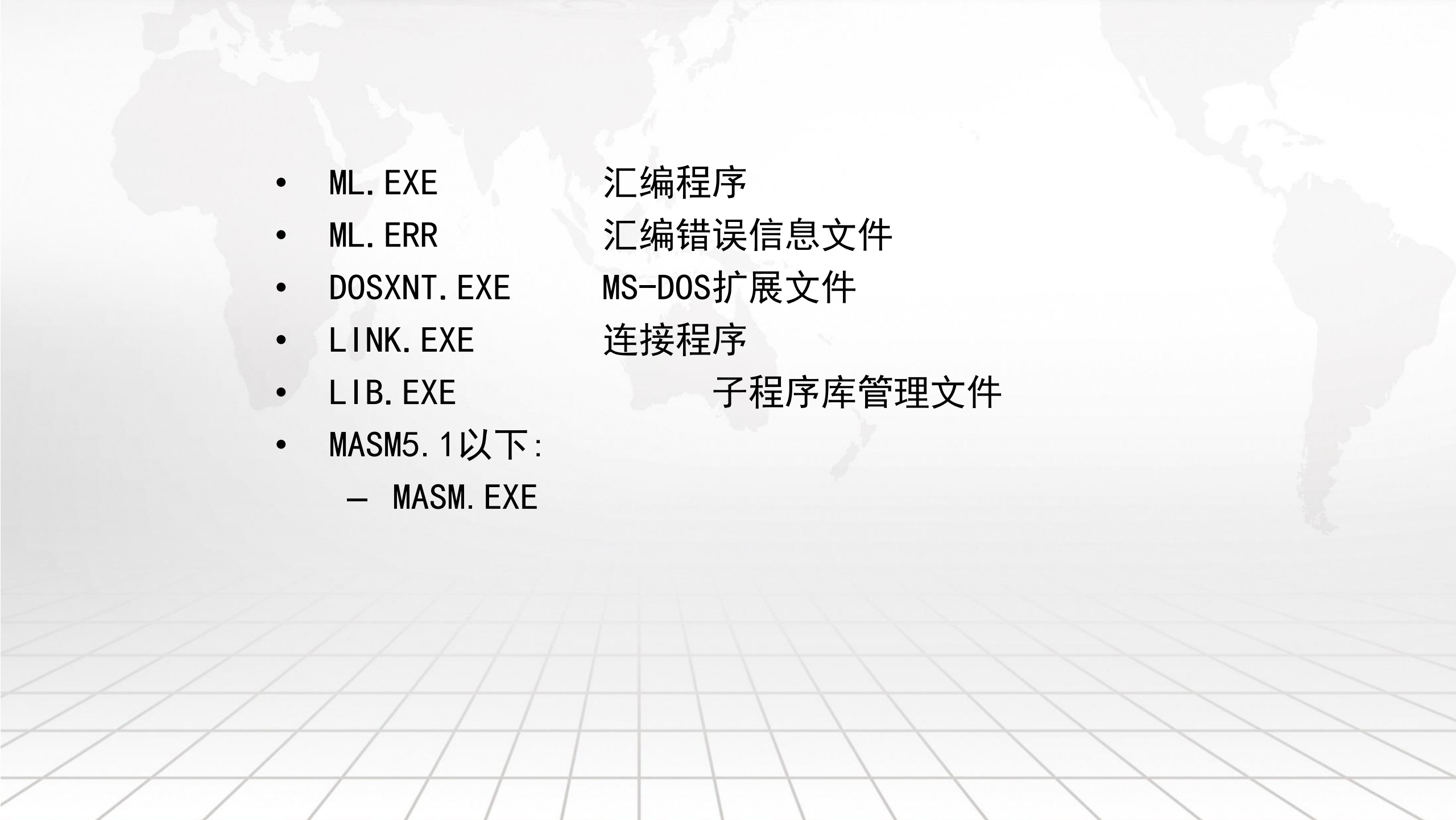
CODE ENDS ; 代码段结束

END START; 汇编结束, 同时表明程序起始位置为标号  
START处



# 汇编语言程序的开发过程



- 
- ML. EXE 汇编程序
  - ML. ERR 汇编错误信息文件
  - DOSXNT. EXE MS-DOS扩展文件
  - LINK. EXE 连接程序
  - LIB. EXE 子程序库管理文件
  - MASM5.1以下:
    - MASM. EXE



# MASM汇编语言表达式、运算符

PART 02



## 常量、运算符及表达式

### 1. 常量

#### (1) 数字常量

二进制常量, 以B结尾

十进制常量, 以D结尾或省略(汇编语言中默认无标记数为十进制数)

十六进制常量, 以H结尾, 如0A8C6H.



MASM提供基数控制，.RADIX伪指令可以改变默认进制。

伪指令格式：.RADIX n

要求：n为2~16范围内任何数值<sup>注1</sup>。

功能：把n表示的数值作为默认基数。

例如指令“.RADIX 16”，将缺省基数改为16

\*注1：emu8086实测，有效值域为{2,8,10,16}

## (2) 字符串常量

用**单引号**引起来的字符或字符串也代表常数。

例：' A' , ' BCDE' , 汇编时被翻译成对应的ASCII码  
41H和42H, 43H, 44H, 45H。

**字符串最大长度为255个字符**

### (3) 符号常量

利用一个标识符表达的一个数值。常数若使用有意义的符号名来表示，可以提高程序的可读性，同时更具有通用性。

MASM提供等价机制，用来为常量定义符号名，符号定义伪指令有“EQU”和“=”

## 2. 运算符

MASM 6.x支持多种运算符，如表4.1所示。

### (1) 算术运算符

实现加、减、乘、除、取余的算术运算。其中MOD也称为取模，它产生除法之后的余数，如 $19 \text{ MOD } 7 = 5$ 。

MOV      AX, 3 \* 4+5 ; 等价于MOV      AX, 17

加“+”和减“-”运算符还可以用于地址表达式。除加、减外，其它运算符的参数必须是整数。

### (2) 逻辑运算符

实现按位相与、相或、异或、求反的逻辑运算。例如：

OR     AL, 03H AND 45H                   ; 等价于OR AL, 01H

### (3) 移位运算符

实现对数值的左移、右移的逻辑操作，移入低位或高位的是0。

其格式为：SHL/SHR移位次数。例如：

```
MOV      AL, 0101B  SHL (2*2)
```

；等价于MOV AL, 01010000B

逻辑和移位运算符与指令助记符相同，并有类似的运算功能。汇编程序能够根据上下文判断它们是指令还是运算符，前者进行代码翻译，后者汇编时计算其数值。



#### (4) 关系运算符

用于比较和测试符号数值，MASM用FFFFH（补码 -1）表示条件为真，用0000H表示条件为假。例如：

```
MOV     BX, ( (PORT LT 5) AND 20) OR ( (PORT GE 5)
AND 30)
```

； 当PORT<5时. 汇编结果为MOV BX, 20

； 否则，汇编结果为MOV BX, 30

#### (5) 高低分离符

取数值的高半部分或低半部分。

HIGH、LOW从一个字数值或符号常量中得到高、低字节，例如：

```
MOVAH, HIGH 8765H ; 等价于MOV AH, 87H
```

MASM 6.0引入的HIGHWORD, LOWWORD取一个符号常量（不是一般的常量）的高字或低字部分，例如：

```
DD_VALUE EQU 0FFFF1234H; 定义一个符号常量
```

```
MOV AX, LOWWORD DD_VALUE ; 等价于MOV AX, 1234H
```

## 运算符

运算符类型	运算符号及说明
算术运算符	+（加） -（减） *（乘） /（除） MOD（取余）
逻辑运算符	AND（与） OR（或） XOR（异或） NOT（非）
移位运算符	SHL（逻辑左移） SHR（逻辑右移）
关系运算符	EQ（相等） NE（不相等） GT（大于） LT（小于） GE（大于等于） LE（小于等于）
高低运算符	HIGH（高字节） LOW（低字节） HIGHWORD（高字） LOWWORD（低字）

### 3、运算符的优先级

优先级	运算符
1	( ) < > [ ] • LENGTH SIZE WIDTH MASK
2	PTR OFFSET SEG TYPE THIS :
3	HIGH LOW
4	* / MOD SHL SHR
5	+ -
6	EQ NE GT LT GE LE
7	NOT
8	AND
9	OR XOR
10	SHORT

TYPE运算符用来取存储器的单元类型, 各单元类型对应值如下:

存储器单元类型	对应值
DB (字节)	1
DW (字)	2
DD (双字)	4
NEAR	-1
FAR	-2

若BUFFER1存储区是用如下伪指令定义的:

```
BUFFER1      DB    100 DUP (?)
```

则 TYPE BUFFER1等于1

LENGTH运算符用来计算一个存储区的单元(单元可以是字节,字或双字)的数目。

注意:LENGTH返回的存储区必须用复制操作符DUP( )来定义,否则返回值为1。

例:

若FEES被定义为: FEES DW 4, 5, 6

则 MOV CX, LENGTH FEES

汇编后, MOV CX, 1



SIZE运算符用来计算一个存储区的字节总数。

例：

若BUFFER2存储区是用如下伪指令定义的：

```
BUFFER2  DW      200 DUP (0)
```

则：

TYPE	BUFFER2	等于2
LENGTH	BUFFER2	等于200
SIZE	BUFFER2	等于400

## (6) 合成运算符

用来建立或临时改变变量或标号的类型或存储器操作的存储单元类型。包括：PTR、THIS、SHORT。

### PTR运算符

用来指定或修改存储器操作数的类型，通常和伪指令BYTE、WORD等连起来使用。

例：MOV        BYTE PTR[DI], 0

      MOV        WORD PTR[DI], 0

而 MOV        [DI], 0    ; 类型不定

例：STR11 DW ? ; 定义为字类型

可用如下语句：

PP EQU BYTE PTR STR11

QQ EQU BYTE PTR STR11+1

通过PP, QQ把STR11和STR11+1又规定为字节类型。

MOV STR11, AX 是合法的

MOV AL, STR11 是非法的

只能用：MOV AL, PP

或 MOV AL, BYTE PTR STR11

## THIS运算符

可以指定存储器操作数的类型。使用THIS运算符可以使标号或变量具有灵活性。

例：

```
AREAW EQU THIS WORD
```

```
AREAB DB 100 DUP (?)
```

## SHORT运算符

可以指定一个标号的类型为SHORT (短标号)，即标号到引用该标号之间的距离在 - 127— + 127个字节范围内。短标号可以用于转移指令中，使用短标号的指令比使用近程标号的指令少一个字节。

## (7) 其它运算符

### 1) 方括号[ ]

指令中用方括号表示存储器操作数，方括号里的内容表示操作数的偏移地址。例如：

MOV AX, [BX] ; 将 (BX) 和 (BX+1) 指向的存储器两个单元的内容送AX

### 2) 段超越运算符 “:”

运算符 “:” (冒号) 跟在某个段寄存器名 (DS、ES、SS或CS) 之后表示段超越，用来指定一个存储器操作数的段属性，而不管其原来隐含的段是什么。例如：MOV AX, ES:[DI] ;

把ES段中由DI指向的字操作数送 (AX)



## 4. 表达式

**表达式**是常量、寄存器、标号、变量与一些运算符组合的序列, 分数字表达式和地址表达式两种。汇编时按一定的优先规则对表达式进行计算后可得到一个数值或一个地址。

## 1. 数字表达式 (number expression)

数值表达式一般是指由运算符 (MASM统称为操作符 Operator) 连接的各种常量所构成的表达式。汇编程序在汇编过程中计算表达式, 最终得到一个数值。由于在程序运行之前, 就已经计算出了表达式, 所以, 程序运行速度没有变慢, 然而程序的可读性却增强了。

例如: `MOV DX, (6*A-B)/2`

指令的源操作数 $(6*A-B)/2$ 是一个表达式。若设变量A的值为1, 变量B的值为2, 则此表达式的值为 $(6*1-2)/2 = 2$ , 是一个数字结果, 此表达式是数字表达式。

## 2. 地址表达式 (address expressions)

地址表达式的结果是一个存储单元的地址。当这个地址中存放的是数据时，称为变量；当这个地址中存放的是指令时，则称为标号。

当在指令的操作数部分用到地址表达式时，应注意其物理意义。例如，两个地址相乘或相除是无意义的，两个不同段的地址相加减也是无意义的。经常使用的是地址加减数字量。

例如， $SUM+1$ 是指向 $SUM$ 字节单元的下一个单元的地址。

又如：`MOV AX, ES: [BX+SI+1000H]`

$BX+SI+1000H$ 为地址表达式，结果是一个存储单元的地址。

## 变量及其属性

### 1. 变量定义伪指令

变量是存储器中某个数据区的名字，在指令中可以作为存储器操作数。

变量定义（Define）伪指令可为变量申请固定长度的存储空间，并可以同时将相应的存储单元初始化。该类伪指令是最经常使用的伪指令。

**伪指令格式：[变量名] 伪指令 初值表**

说明：

- 变量名是用户自定义标识符，表示初值表首元素的逻辑地址，即用这个符号表示地址，常称为符号地址。变量名可以没有，这种情况，汇编程序将直接为初值表分配空间，无符号地址。设置变量名是为了方便存取它指示的存储单元。
- 初值表是用逗号分隔的参数，主要由数值常量，表达式或？、DUP组成。其中？表示初值不确定，即未赋初值；重复初值可以用DUP进行定义。DUP的格式为：

重复次数 DUP（重复参数）

- 变量定义伪指令有DB/DW/DD/DF/DQ/DT，它们根据申请的主存空间单位分类，



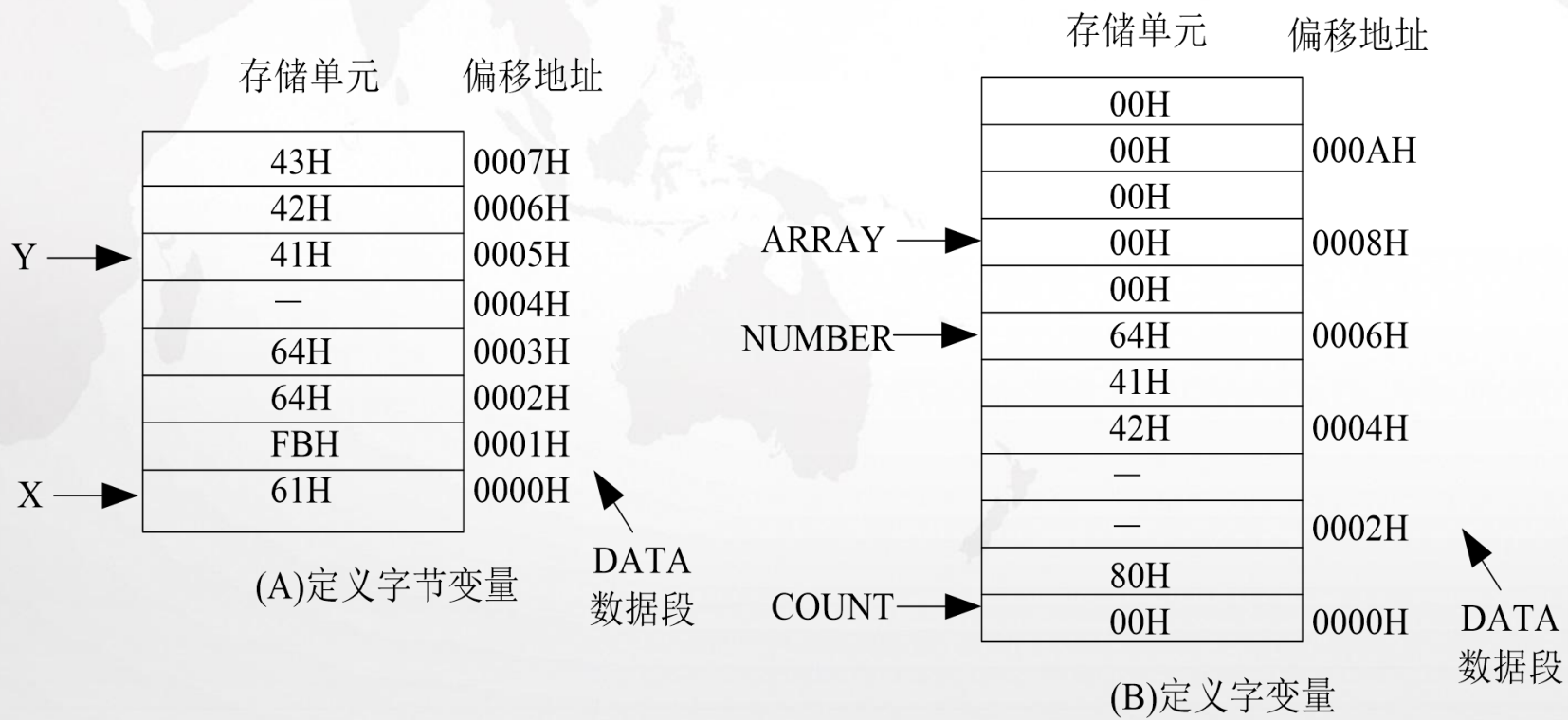
## (1) 定义字节单元伪指令DB

功能：定义变量的类型为BYTE，给变量分配字节或字节串。

要求：初值表中每个数据一定是字节量（Byte），可以是0~255的无符号数或是-128~+127带符号数，也可以是字符串常量。例如：

```
DATA    SEGMENT          ; 数据段
        X    DB    'a', -5
        DB    2    DUP (100), ?
        Y    DB    'ABC'
DATA    ENDS
```

- 存储器中的分配情况如图



## 数据定义的存储形式

针对上述的DATA定义，在执行下述汇编指令后，可以看到所定义的DATA中的数据发生了变化。

MOV       AL, X

      ; 此处X表示它的第1个数据，故 $AL \leftarrow 'a'$

DEC       X+1

      ; 对X为起始的第2个数据减1，故成为 -6

MOV Y, AL       ; 现在Y这个字符串成为'aBC'

## (2) 定义字单元伪指令DW

功能：定义变量的类型为WORD，给变量分配一个或多个字单元，并可以将它们初始化为指定值。

要求：初值表中每个数据一定是字（Word），一个字单元可用于存放任何16位数据，如一个段地址、一个偏移地址、两个字符、0~65535之间的无符号数或者是

-32768~+32767之间的带符号数。例如：

```
DATA          SEGMENT; 数据段
    COUNT      DW      8000H , ?, 'AB'
    MAXINT      EQU    64H
    NUMBER      DW      MAXINT
    ARRAY       DW      MAXINT DUP (0)
DATA          ENDS
```

- 数据在存储器的分配情况如图所示

### (3) 定义双字单元伪指令DD

功能：定义变量的类型为DWORD，用于分配一个或多个双字单元，并将它们初始化为指定值。

要求：初值表中每个数据是一个32位的双字（Double Word），可以是有符号或无符号的32位整数，也可以用来表达16位段地址（高位字）和16位的偏移地址（低位字）的远指针。

例如：

```
VARDD          DD    0, ?, 12345678H
```

```
FARPOINT DD    00400078H
```

注意：在内存中存放时，低位字在前，高位字在后。

#### (4) 其它数据定义伪指令

- 定义3字伪指令DF

功能：用于为一个或多个6字节变量分配空间及初始化。

用途：6字节常用在 32位CPU中表示一个48位远指针（16位段选择器：32位偏移地址）。

- 定义4字伪指令DQ

功能：用于为一个或多个8字节变量分配空间及初始化。

用途：8字节变量可以表达一个64位整数。

- 定义10字节伪指令DT

功能：用于为一个或多个10字节变量分配空间及初始化。



## (5) 定位伪指令

用数据定义伪指令分配的数据是按顺序一个接着一个存放在数据段中的。但有时，我们希望能够控制数据的偏移地址。例如使数据对齐可以加快数据的存取速度。MASM提供了这样的伪指令，这些伪指令我们称为定位伪指令。

## ORG伪指令

伪指令格式：ORG参数

功能：ORG伪指令是将当前偏移地址指针指向参数表达的偏移地址。例如：

ORG 100h ; 从100h处安排数据或程序

ORG \$+10 ; 偏移地址加10，即跳过10个字节空间

汇编语言程序中，符号“\$”表示当前偏移地址值。例如，在偏移地址100H单元开始定义“DW 1, 2, \$+4, \$+4”，那么在104H单元的值为108H，106H单元的值为10AH。又如：

```
ARRAY          DB          12, 34, 56
```

```
LEN            EQU    $ - ARRAY
```

那么，LEN的值就是ARRAY变量所占的字节数。

## EVEN伪指令

- 伪指令格式：EVEN
- 功能：EVEN伪指令使当前偏移地址指针指向偶数地址，即若原地址指针已指向偶地址，则不作调整；否则将地址指针加1，使地址指针偶数化。
- 用途：EVEN可以对齐字数据。

## ALIGN伪指令

- 伪指令格式：ALIGN      n
- 功能：ALIGN伪指令是将当前偏移地址指针指向n（n是2的乘方）的整数倍的地址，即若原地址指针已指向n的整数倍地址，则不作调整；否则将指针加以1 ~ n-1中的一个数，使地址指针指向下一个n的整数倍地址。

例如：

```
DATA          SEGMENT      ; 完整段定义
    DATA01      DB 1, 2, 3  ; DATA01的偏移地址为
                        0000H
    EVEN                          ; 等价于ALIGN 2
    DATA02      DW 5        ; DATA02的偏移地址为
                        0004H
    ALIGN 4
    DATA03      DD 6        ; DATA03的偏移地址为
                        0008H
    ORG          $+10H
    DATA04      DB 'ABC'    ; DATA04的偏移地址为
                        001CH
DATA          ENDS
```

## 2变量和标号的属性

变量、标号、段名及过程名都表示的是地址，那么，这些标号和名字一经定义便具有以下三种属性：

- 段值：标号和名字对应存储单元所在段的段地址。
- 偏移值：标号和名字对应存储单元所在段的段内偏移地址。
- 类型：标号、子程序名的类型可以是NEAR（近）和FAR（远），分别表示段内或段间；变量名的类型可以是BYTE（字节），WORD（字）和DWORD（双字）等。

## (1) 地址操作符

地址操作符可获得名字或标号的段地址和偏移地址两个属性值。

- 中括号 [ ] 表示将括起的表达式作为存储器地址指针；
- 符号 \$ 表示当前偏移地址；
- 段前缀的冒号：也是一种地址操作符，它表示采用指定的段地址寄存器。
- 另外，还有两个经常应用的地址操作符 OFFSET 和 SEG。



## OFFSET操作符

- 使用格式：OFFSET名字/标号
- 功能：返回名字或标号的偏移地址

## SEG操作符

- 使用格式：SEG名字/标号
- 功能：返回名字或标号的段地址

例如把字节变量ARRAY的段地址和偏移地址送入DS和BX就可用下列指令序列实现。

```
MOVAX, SEG ARRAY
```

```
MOVDS, AX
```

```
MOVBX, OFFSET ARRAY
```

；等价于LEA BX, ARRAY

- 加、减运算符同样可以用于地址表达式，例如：

```
MOVCL, ARRAY+4
```

；等效于MOV CL, ARRAY[4]，这里的“4”表示4个字节单元

## (2) 类型操作符

- 类型操作符对名字或标号的类型属性进行有关设置。该类操作符有PTR、THIS、SHORT和TYPE。

### ①PTR操作符

- 使用格式：类型名 PTR 名字/标号
- 要求：PTR操作符中的“类型名”可以是  
BYTE/WORD/DWORD/FWORD/QWORD  
/TBYTE，或者是NEAR/FAR，还可以是由STRUCT，RECORD，  
UNION以及TYPEDEF定义的类型。
- 功能：使名字或标号具有指定的类型

例：

```
MOV  AL, BYTE PTR W_VAR
```

； W\_VAR是一个字变量， BYTE PTR使其作为一个字节变量

```
JMP  FAR PTR N_LABEL
```

； N\_LABEL是一个标号， FAR PTR使其作

； 为段间转移使用PTR操作符，可以临时改变名字或标号的类型。

## ②THIS操作符

- 使用格式：THIS类型名
- 功能：创建当前地址，具有指定的类型。类型名同PTR操作符中的类型。例如：

B\_VAR EQU THIS BYTE；按字节访问变量B\_VAR，但与变量  
； W\_VAR的地址相同

W\_VAR DW 10 DUP (0) ；按字访问变量W\_VAR

F\_JUMP EQU THIS FAR

；用F\_JUMP为段间转移 (F\_JUMP LABEL FAR)

N\_JUMP: MOV AX, W\_VSR

；用N\_JUMP为段内近转移，但两者指向；同一条指令

MASM中还有一个LABEL伪指令，它的功能等同于“EQU THIS”。

### ③SHORT操作符

使用格式：SHORT标号

要求：转移范围为  $-128 \sim +127$  字节内。

功能：设定标号为短转移标号。SHORT指定标号作为  $-128 \sim +127$  字节范围内的短转移。例如：

JMP            SHORT        N\_JUMP

### ④TYPE操作符

使用格式：TYPE    名字/标号

功能：返回一个属性为字数值，表明名字或标号的类型。

- 类型的返回数值

类型	返回数值
变量	该变量类型的每个数据占用的字节数
结构	每个结构元素占用的字节数
常量	0
标号	距离属性值
寄存器	该寄存器具有的字节数



- 操作符SIZEOF和LENGTHOF具有类似TYPE的功能，分别返回整个变量占用的字节数和整个变量的数据项数（即元素数）。实际上：
- $\text{SIZEOF返回值} = \text{LENGTHOF返回值} \times \text{TYPE返回值}$
- 请注意，在MASM 5.x仅支持SIZE和LENGTH操作符。LENGTH对于变量定义使用 DUP的情况，返回分配给该变量的单元数，其它情况为1。SIZE返回LENGTH与TYPE的乘积。

**例**下面程序的功能为：在屏幕上显示字符串“1357?????????”

```
.MODEL      SMALL
.STACK
.DATA
    V_BYTE EQU      THIS BYTE; V_BYTE是字节类型的变量，但与变
        ; 量V_WORD地址相同
    V_WORD  DW      3332H, 3735H ; V_WORD是字类型的变量
TARGET  DW  5 DUP (20H) ; 分配数据空间2 × 5 = 10个字节
CRLF    DB          0DH, 0AH, '$'
FLAG    DB          0
N_POINT DW OFFSET S_LABEL ; 取得标号S_LABEL的偏移地址
.CODE
.STARTUP
MOV     AL, BYTE PTR V_WORD ; 用PTR改变V_WORD的
        ; 类型，否则与AL寄存器类型不匹配，指令行后(AL)=32H
```

DEC AL

MOV V\_BYTE, AL ; 对V\_WORD的第一个字节操作, 原  
; 来是32H, 现在是31H

N\_LABEL: CMP FLAG, 1

JZ S\_LABEL ; FLAG单元为1, 则转移

INC FLAG

JMP SHORT N\_LABEL ; 短转移

S\_LABEL: CMP FLAG, 2

JZ NEXT ; FLAG单元为2转移

INC FLAG

JMP N\_POI; 段内的存储器间接寻址, 转

; 移到标号S\_LABEL处

NEXT: MOV AX, TYPE V\_WORD ; 汇编结果为MOV AX, 2

MOV CX, LENGTH TARGET; 汇编结果为MOV CX, 5

MOV SI, OFFSET TARGET

```
W_AGAIN:      MOV     [SI], AX                ; 对字单元操作
INC     SI                ; SI指针加2
INC     SI
LOOP    W_AGAIN            ; 循环
MOV     CX, SIZE          TARGET ; 汇编结果为MOV CX, 0AH
MOV     AL, '?'
MOV     DI, OFFSET        TARGET
B_AGAIN:      MOV     [DI], AL                ; 对字节单元操作
INC     DI                ; DI指针加1
LOOP    B_AGAIN            ; 循环
MOV     DX, OFFSET V_WORD ; 显示结果: 1357??????????
                        MOV     AH, 9
                        INT     21H
                        .EXIT 0
END
```



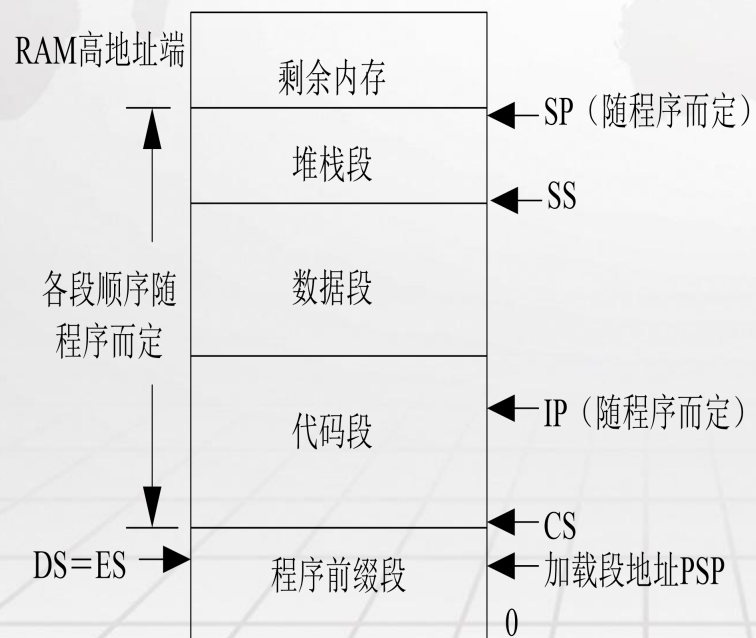
# 程序段的定义和属性

PART 03

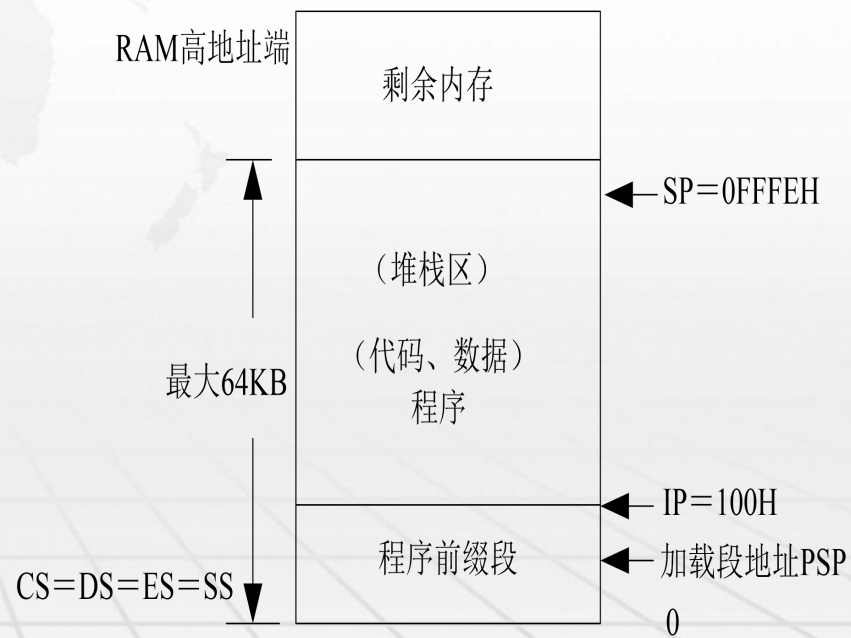


# DOS的程序结构

## 1. EXE程序



## 2. COM程序







## 简化段定义的格式

. MODEL	SMALL	； 定义程序的存储模式（一般采用
SMALL)		
. STACK		； 定义堆栈段
. DATA		； 定义数据段
. . .		； 数据定义
. CODE		； 定义代码段
. STARTUP		； 程序起始点，并建立DS，SS内容（注1）
. . .		； 程序代码
. EXIT 0		； 程序结束点，返回DOS（注2）
. .		； 子程序代码
END		； 汇编结束（注3）

## 1. 存储模式伪指令

伪指令格式：MODEL 存储模式[, 语言类型][, 操作系统类型][, 堆栈选项]

MASM有7种不同的存储模式

TINY（微型模式）

SMALL（小型模式）

COMPACT（紧凑模式）

MEDIUM（中型模式）

LARGE（大型模式）

HUGE（巨型模式）

FLAT（平展模式）

## 2. 简化段定义伪指令

- (1) .STACK堆栈段伪指令
- (2) .DATA数据段伪指令
- (3) .CODE 代码段伪指令

3. 程序开始伪指令

. STARTUP

4. 程序终止伪指令

. EXIT[返回数码]

. EXIT 0 等效于

MOV       AX, 4C00H

INT        21H

5. 汇编结束伪指令 END



## 完整段定义的格式

```
STACK SEGMENT                ; 定义堆栈段STACK
                                ; 分配堆栈段的大小
                                . . .
STACK ENDS                    ; 堆栈段结束
DATA SEGMENT                  ; 定义数据段DATA
                                ; 定义数据
                                . . .
DATA ENDS                     ; 数据段结束
CODE SEGMENT 'CODE'           ; 定义代码段CODE
ASSUME CS: CODE, DS: DATA, SS: STACK ; 确定CS/DS/SS指向
                                ; 的逻辑段

START:  MOV AX, DATA           ; 设置数据段的段地址DS
        MOV DS, AX
        . . .                  ; 程序代码
        MOV AX, 4C00H          ; 返回DOS
        INT 21H
        . . .                  ; 子程序代码
CODE ENDS                      ; 代码段结束
END START                      ; 汇编结束, 程序起始位置为START
```

**例** 设计一个COM程序，实现按任意键后响铃的功能。


```
.MODEL      TINY                ; 采用微型模式
.CODE                               ; 只有一个段，没有数据
    段和堆栈段
.STARTUP                               ; 等效于ORG 100H，汇编程序自动产生
    MOV     DX, OFFSET STRING        ; 显示信息
    MOV     AH, 9
    INT     21H
    MOV     AH, 01H                  ; 等待按键
    INT     21H
    MOV     AH, 02H                  ; 响铃
    MOV     DL, 07H
    INT     21H
.EXIT 0
STRING DB 'PRESS ANY KEY TO CONTINUE !$'
    ; 数据安排在不与代码冲突的地方
END
```



对于一个典型的MASM程序，完整段可用下述格式定义。

```
STACK          SEGMENT STACK      ; 定义堆栈段STACK
               . . .             ; 分配堆栈段的大小
STACK          ENDS               ; 堆栈段结束
DATA           SEGMENT           ; 定义数据段DATA
               . . .             ; 定义数据
DATA           ENDS              ; 数据段结束
CODE           SEGMENT 'CODE'     ; 定义代码段
CODE
```

```
ASSUME CS: CODE, DS: DATA, SS: STACK
      ; 确定CS/DS/SS指向的逻辑段
```



```
START:  MOV  AX, DATA    ; 设置数据段的段地址DS
        MOV  DS, AX
        . . .           ; 程序代码
        MOV  AX, 4C00H   ; 返回DOS
        INT  21H
        . . .           ; 子程序代码
CODE    ENDS             ; 代码段结束
END START                ; 汇编结束, 程序起始位置为START
```

## 1. 完整段定义伪指令

- 完整段定义由SEGMENT和ENDS这一对伪指令实现。
- 伪指令格式：
- 段名      SEGMENT [定位] [组合] [段字] [‘类别’]  
                 . . .            ; 语句序列
- 段名      ENDS
- 功能：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。

### (1). 段定位 (Align) 属性

指定逻辑段在主存储器中的边界，该关键字可表示为：

- BYTE: 段开始地址为字节地址 (xxxx xxxxB)，属性值为1；
- WORD: 段开始地址为字地址 (xxxx xxx0B)，属性值为2；
- DWORD: 段开始地址为4的倍数 (xxxx xx00B)，属性值为4
- PARA: 段开始地址为16的倍数 (xxxx 0000B)，即节地址，属性值为16
- PAGE: 表示段开始地址256的倍数 (0000 0000B)，即页地址，属性值为256

## (2) 段组合 (Combine) 属性

- 指定多个逻辑段之间的关系。组合的关键字有：
  - PRIVATE:
  - PUBLIC:
  - STACK:
  - COMMON:

### (3) 段字 (Use) 属性

- 这是为支持32位段而设置的属性。

### (4) 段类别 (Class) 属性

- 当连接程序组织段时，将所有的同类别段相邻分配。
- 段类别可以是任意名称，但必须位于单引号中；
- 大多数MASM程序使用 'CODE'，'DATA' 和 'STACK' 来分别指名代码段、数据段和堆栈段，以保持所有代码和数据的连续。



## 2. 指定段寄存器伪指令

- 伪指令格式：ASSUME 段寄存器：段名[, 段寄存器名：段名, . . . ]
- 功能：ASSUME伪指令通知MASM用指定的段寄存器来寻址对应的逻辑段，即建立段寄存器与段的缺省关系。

```
CODE  SEGMENT
```

```
ASSUME CS: CODE, DS: DATA, SS: STACK, ES: DATA
```

```
MOV     AX, DATA
```

```
MOV     DS, AX
```

```
MOV     ES, AX
```

```
MOV     AX, STACK
```

```
MOV     SS, AX
```

```
⋮
```

```
CODE  ENDS
```

### 3. GROUP段组伪指令

- 伪指令格式： 组名 GROUP 段名[, 段名, . . . ]
- 功能：伪指令GROUP把多个同类段合并为一个64KB物理段，并用一个组名统一存取它。
- 例如：

```
DATA1 SEGMENT      WORD PUBLIC 'CONST' ; 常量数据  
段
```

```
        CONST1          DW          100
```

```
DATA1          ENDS
```

```
DATA2 SEGMENT WORD PUBLIC ' VARS' ; 变量数据段
```

```
        VAR1          DW  ?
```

```
DATA2          ENDS
```

```
DATAGROUP GROUP DATA1, DATA2; 进行组合
```

## 4. 段顺序伪指令

段在主存中的实际顺序是可以设置的.

### (1) SEG段顺序伪指令

- 伪指令格式: . SEG
- 功能: 按照源程序的各段顺序排序。

### (2) DOSSEG段顺序伪指令

- 伪指令格式: . DOSSEG
- 功能: 其它微软的程序设计语言按照标准DOS规定进行排序, 即代码段、数据段、堆栈段。

### (3) ALPHA段顺序伪指令

- 伪指令格式: . ALPHA
- 功能: 按照段名的字母顺序排序。

## 5. 定位伪指令

- (1) **ORG伪指令**

ORG 100h ; 从100h处安排数据或程序

ORG \$+10 ; 使偏移地址加10, 即跳过10个字节空间

- (2) **EVEN伪指令**

- EVEN伪指令使当前偏移地址指针指向偶数地址, 即若原地址指针已指向偶地址, 则不作调整; 否则将地址指针加1, 使地址指针偶数化。

- 用途: EVEN可以对齐字数据。

- (3) **ALIGN伪指令**

- 将当前偏移地址指针指向n (n是2的乘方) 的整数倍的地址, 即若原地址指针已指向n的整数倍地址, 则不作调整;



# 复杂数据结构

PART 04





## 2. 结构变量的定义

- 变量名 结构名 <字段初值表>

STU1          STUDENT      <1, ' ZHANG' , 85, 90>

STU2          STUDENT      <2, ' WANG' , , >

STUDENT      100 DUP (< >)

; 预留100个结构变量空间

### 3. 结构变量及其字段的引用

- 结构变量名. 结构字段名
- `MOV       STU1.MATH, 95`  
      ; 执行指令后, 将对MATH域  
      ; 的值更新为95。



# 记录

## 1. 记录类型的说明

记录名RECORD位段[, 位段. . . ]

记录中位段的格式定义如下:

位段名: 位数[=表达式]

例如:

PERSON RECORD YEAR: 4, SEX: 1=0, MARRIAGE: 1=1

## 2. 记录变量的定义

记录变量定义的格式为：

记录变量名 记录名 <段初值表>

例如：

ZHANG PERSON <1000B, 1, 0> ; 该字节值为：00100010B=22H

WANG PERSON <1001B, 0, 0> ; 该字节值为：  
00100100B= 24H

### 3. 记录变量的引用和记录操作符

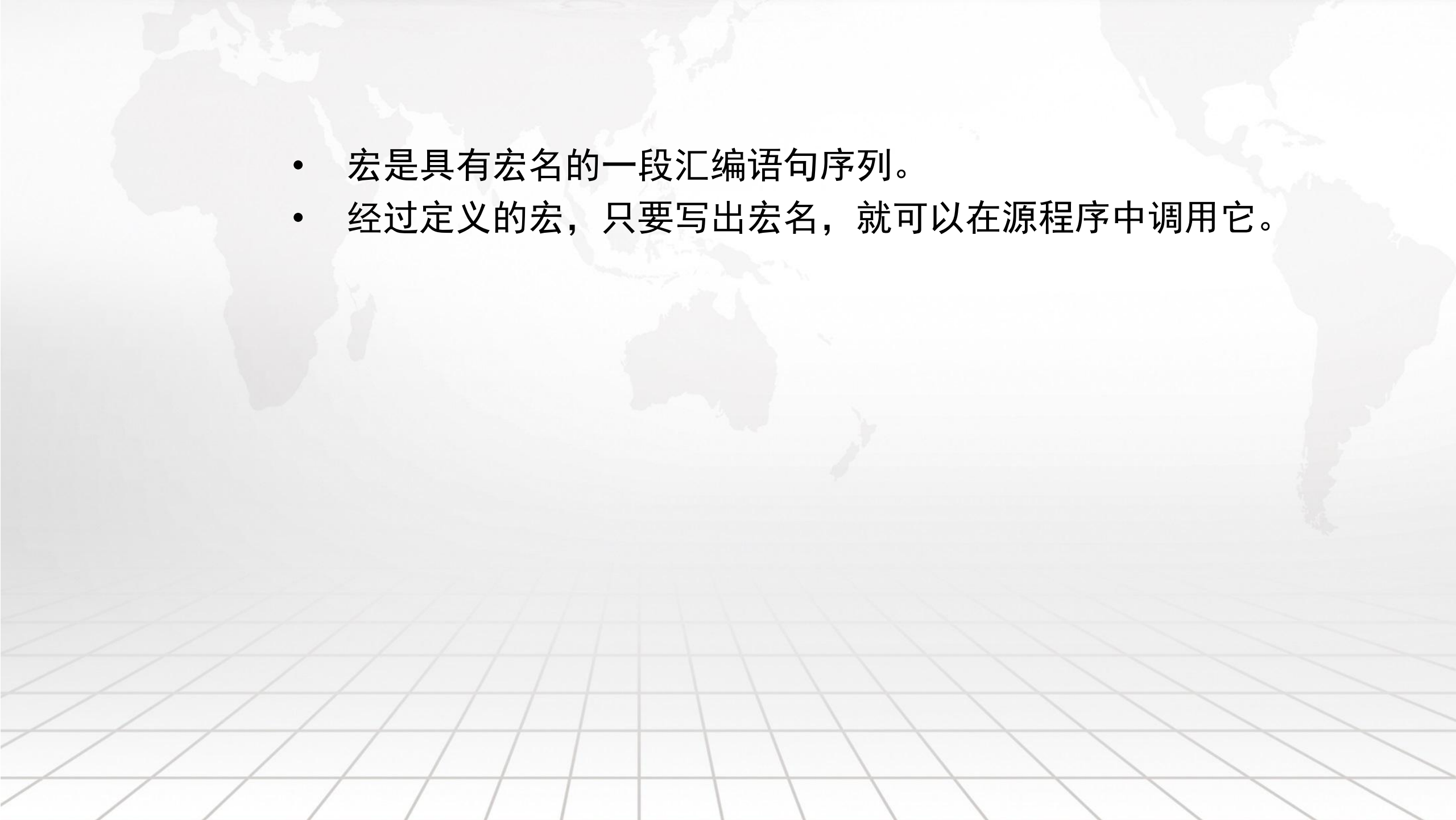
- 记录变量通过它的变量名直接引用，表示它的字节或字值



# 宏汇编

PART 05



- 
- 宏是具有宏名的一段汇编语句序列。
  - 经过定义的宏，只要写出宏名，就可以在源程序中调用它。



# 宏的定义和调用

## 1. 宏定义

宏定义由一对宏汇编伪指令MACRO/ENDM来完成。格式：

宏名   MACRO[形参表]

宏定义体

ENDM

例如：

MAINBEGIN    MACRO   ； 定义名为MAINBEGIN的宏，无参数

    MOVAX, @DATA    ； 宏定义体

    MOV DS, AX

ENDM

； 宏定义结束

## 2. 宏调用

宏定义之后就可以使用它，即宏调用。宏调用遵循先定义后调用的原则。宏调用格式为：

伪指令格式：宏名[实参表]

(1) 宏定义中可以有宏调用，只要遵循先定义后调用的原则。

例如：

```
DOSINT21      MACRO      FUNCTION          ; 宏定义
```

```
                MOV      AH, FUNCTION
```

```
                INT      21H
```

```
                ENDM
```

```
DISPMSG MACRO MESSAGE          ; 含有宏调用的宏定义
```

```
                MOV      DX, OFFSET MESSAGE
```

```
                DOSINT21 9          ;
```

宏调用

```
                ENDM
```

上述宏定义汇编后的列表文件如下。

```
DISPMSG MSG                ; 宏调用
MOV      DX, OFFSET MSG    ; 宏展开（第一层）
DOSINT21      9
MOV      AH, 9              ; 宏展开（第二层）
INT      21H
```

## (2) 宏定义允许嵌套

即宏定义体内可以有宏定义，对这样的宏进行调用时需要多次分层展开。宏定义内也允许递归调用，这种情况需要用到后面将介绍的条件汇编指令给出递归出口条件。



## 宏的参数

例 具有多个参数的宏定义。

使用8086的移位指令有时感到不便，因为当移位次数大于1时，必须利用CL寄存器。现在用宏指令SHLEXT扩展逻辑左移SHL的功能。例如：

```
SHLEXT    MACRO    SHLOPRAND, SHLNUM
    PUSH    CX
    MOV CL, SHLNUM    ; SHLNUM表示移位次数
    SHL SHLOPRAND, CL ; SHLOPRAND表示被
                        ; 移位的操作数
    POP CX
ENDM
```

当我们要将AX左移6位时，可以采用如下宏指令：

```
SHLEXT      AX, 6
```

汇编后，宏展开为：

```
PUSH      CX
```

```
MOV CL, 06
```

```
SHL AX, CX
```

```
POP CX
```



## 宏操作符：

- &** 替换操作符，用于将参数与其它字符分开。
- < >** 字符串传递操作符。用于括起字符串。
- !** 转义操作符。用于指示其后的一个字符作为一般字符，而不含特殊意义。
- %** 表达式操作符。用在宏调用中，表示将随后的一个表达式的值作为实参，而不是将表达式本身作为参数。例如，已经有宏定义DSTRING（见下一页），那么：  
DSTRING % (1024 - 1) ; 宏调用  
DB '1023', 0DH, 0AH, '\$' ; 宏展开
- ;** 宏注释符。用于表示在宏定义中的注释。采用这个符号的注释，在宏展开时不出现。

另外，宏定义中还可以用“: REQ”说明设定不可缺参数，用“: =默认值”设定参数默认值。

例 用于字符串的宏定义参数

```
DSTRING      MACRO  STRING  
    DB          '& STRING&', 0DH, 0AH, '$'  
    ENDM
```

例如，要定义字符串 'THIS IS A EXAMPLE.'，可以采用如下宏调用。

```
DSTRING      <THIS IS A EXAMPLE. >
```

它产生的宏展开为：

```
    DB  'THIS IS A EXAMPLE.' , 0DH,  
    0AH, '$'
```

## 与宏有关的伪指令

### 1. 局部标号伪指令LOCAL

问题提出： 宏定义可被多次调用，当宏定义中使用了标号，同一源程序对它的多次调用就会造成标号的重复定义，汇编将出现语法错误。

解决：如果宏定义体采用了标号，可以使用局部标号伪指令LOCAL加以说明。

伪指令格式为：LOCAL 标号列表

每次宏展开时汇编程序将对其中的标号自动产生一个唯一的标识符（其形式为“??0000”到“??FFFF”），避免宏展开后的标号重复

注意：LOCAL伪指令只能在宏定义体内使用，而且是宏定义MACRO语句之后的第一条语句，两者间也不允许有注释和分号。

ABSOL MACRO OPRD

LOCAL NEXT

CMP OPRD, 0

JGE NEXT

NEG OPRD

NEXT:

ENDM ; 这个伪指

; 令要独占一行

采用例4.15宏定义的宏调用形式为:

ABSOL WORD PTR [BX]

ABSOL BX

上述宏调用下的宏展开为:

CMP WORD PTR [BX], 0

JGE ??0000

NEG WORD PTR [BX]

?? 0000:

CMP BX, 0

JGE ?? 0001

NEG BX

?? 0001:

## 2. 宏定义删除伪指令PURGE

当我们不需要某个宏定义时，可以把它删除，删除宏定义伪指令的格式为：

PURGE    宏名表

允许宏名与其它指令包括伪指令同名，此时宏名优先级最高。当不再使用这个宏定义时，及时用PURGE删除即恢复原指令功能。

### 3. 宏定义退出伪指令EXITM

- EXITM
- 它可用于宏定义体、重复汇编的重复块以及条件汇编的分支代码序列中，汇编程序执行EXITM指令后立即停止它后面部分的宏展开。





## 宏与子程序

- 共同点
  - 都可以把一段程序用一个名字定义，简化源程序的结构和设计。
- 区别：
  - 宏调用在汇编时进行程序语句的展开，不需要返回；子程序调用在执行时由CALL指令转向子程序体，需要执行RET指令返回；
  - 宏调用仅是源程序级的简化，并不减小目标程序，子程序还是目标程序级的简化，形成的目标代码较短

- 宏调用简单、直接，不需要返回；子程序需要利用堆栈保存和恢复转移地址、寄存器等，要占用一定的时空开销；
- 宏调用的参数通过形参、实参结合实现传递，子程序需要利用寄存器、存储单元或堆栈等传递参数。
- 对宏调用来说，参数传递错误通常是语法错误，会由汇编程序发现；而对子程序来说，参数传递错误通常反映为逻辑或运行错误，不易排除。
- 当程序段较短或要求较快执行时，应选用宏；当程序段较长或为减小目标代码时，要选用子程序。