
上海交通大学
网络学院

微机原理与应用

The Principle & Application of Microcomputer

王春香　副教授

wangcx@sjtu.edu.cn

第四章 汇编语言程序设计

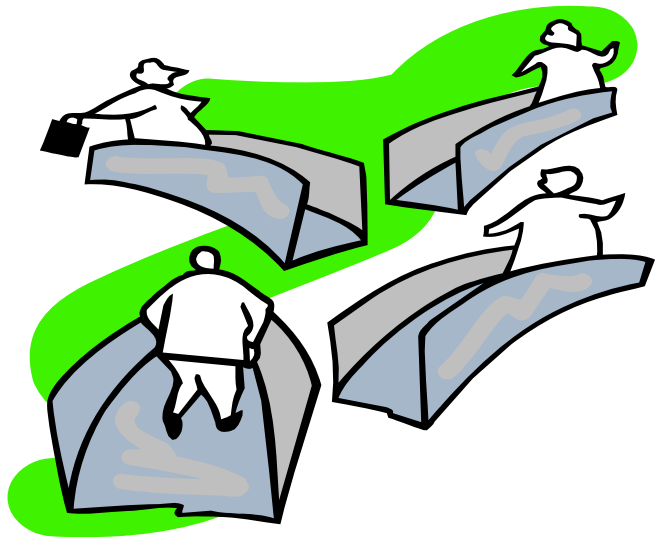
主要内容

4.1 程序设计语言概述

4.2 8086/8088汇编语言的基本语法

4.3 8086/8088汇编语言程序设计基本方法

4.4 软件调试技术



第四章 汇编语言程序设计

学习要求

- 通过学习8086/8088汇编程序实例，理解源程序结构：分段、行语句和字段。
- 学习汇编语言语句的类型及格式，掌握指令语句与伪指令语句的异同点。
- 学习8086/8088汇编语言的数据项时，着重分清变量与标号的区别。
- 学习表达式和运算符时，着重掌握地址表达式3个属性。
- 要熟练掌握和灵活运用汇编语言程序的3种基本结构：顺序结构、分支结构、循环结构。
- 掌握基本的DOS和BIOS中断调用功能。

第四章 汇编语言程序设计

汇编语言具有执行速度快和易于实现对硬件的控制等独特优点，至今仍是用户使用得较多的程序设计语言。特别是在对程序的空间和时间要求很高的场合，以及需要直接控制设备的应用场合，汇编语言更是必不可少了。

由于汇编语言本身的特点，本章将选择目前国内广泛使用的IBM PC机作为基础机型，着重讨论8086/8088汇编语言的基本语法规则和程序设计的基本方法，以掌握一般汇编语言程序设计的初步技术。

4.1 程序设计语言概述

程序设计语言是专门为计算机编程所配置的语言。
它们按照形式与功能的不同可分为3种，即：

◆ **机器语言**

机器码表示，例如 B8H、C3H （天书）

◆ **汇编语言**

用指令助记符表示机器码 （难学）

例： 机器码 B8H、C3H的助记符为

MOV AX，BX

注：CPU不同，机器码不同，助记符不同

◆ **高级语言**

语言规范，可用于不同的 CPU （通用）

4.1 程序设计语言概述

■ 机器语言 (Machine Language)

机器语言是由0、1二进制代码书写和存储的指令与数据。

特点：

能为机器直接识别与执行；

程序所占内存 空间较少。

缺点：

难认、难记、难编写、易出错。

4.1 程序设计语言概述

■ 汇编语言 (Assembly Language)

汇编语言是用指令的助记符、符号地址、标号等书写程序的语言，简称符号语言。

特点：

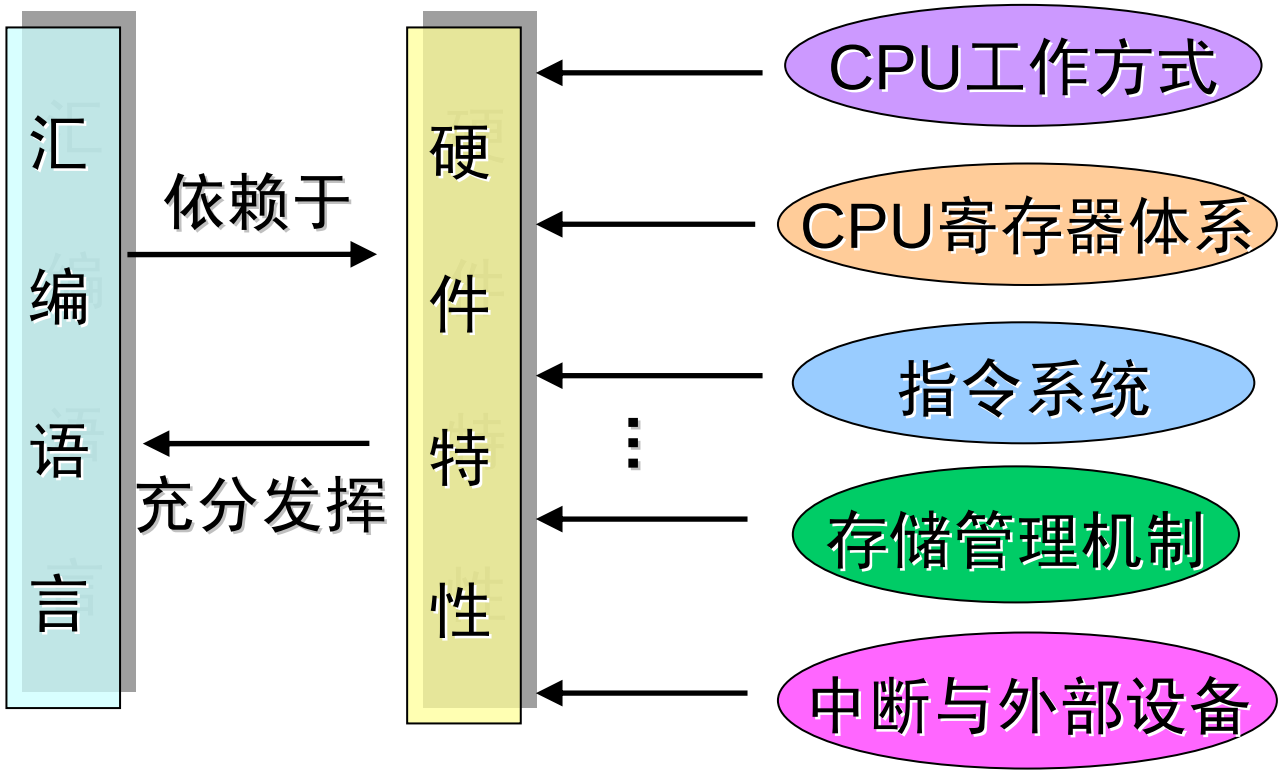
易读、易写、易记。

缺点：

不能像机器语言那样为计算机所直接识别，也不如高级语言那样具有很好的通用性和可移植性。

4.1 程序设计语言概述

■ 汇编语言 (Assembly Language)



4.1 程序设计语言概述

■ 高级语言 (High Level Language)

高级语言是脱离具体机器(即独立于机器)，面向用户的通用语言，不依赖于特定计算机的结构与指令系统。

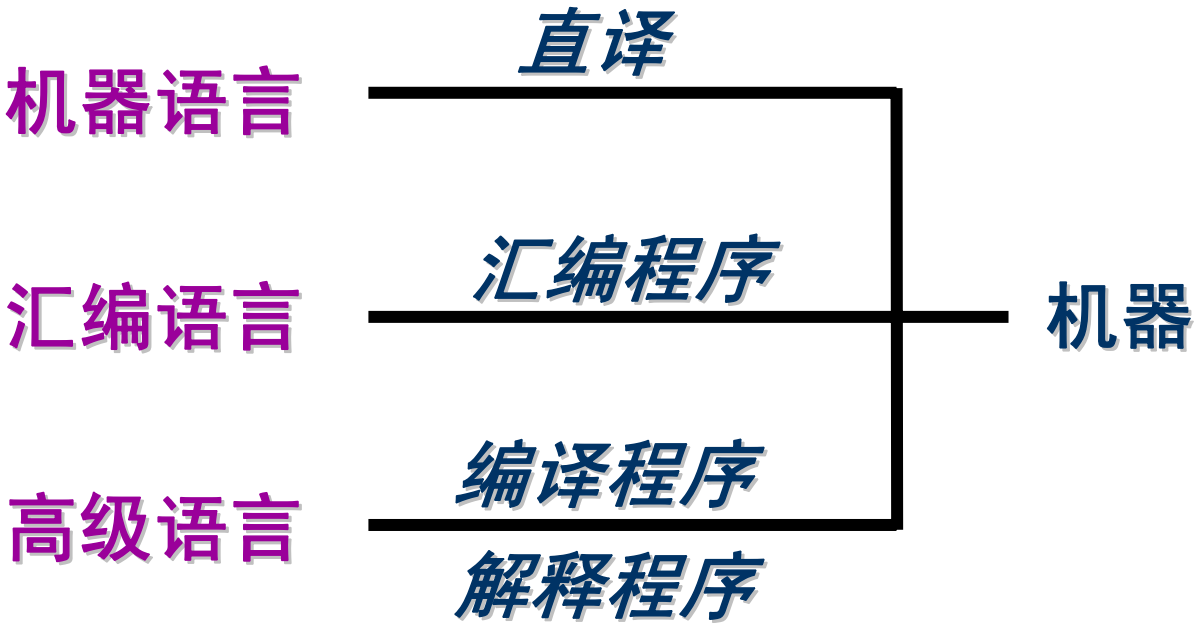
用同一种高级语言编写的源程序，一般可在不同计算机上运行而获得同一结果。

由于高级语言的通用性特点，对于高级语言程序员来说，不必熟悉计算机内部具体结构和机器指令，而只需要把主要精力放在程序结构和算法描述上面。

所以，高级语言具有更广泛的领域。

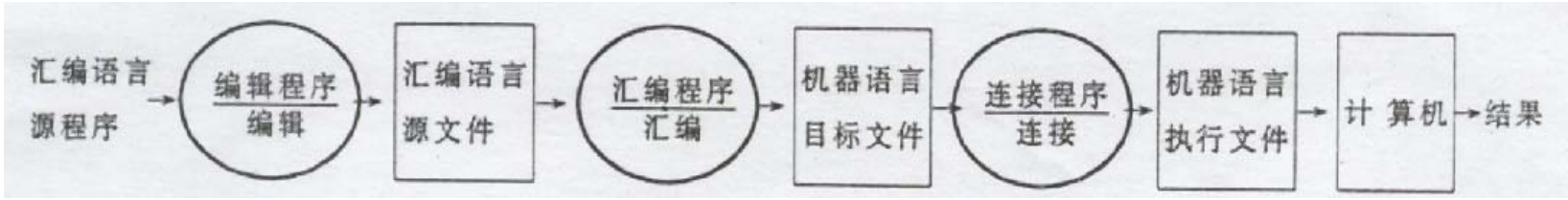
4.1 程序设计语言概述

- 从计算机语言到计算机机器码



4.1 程序设计语言概述

■ 汇编语言程序的上机与处理过程



椭圆表示系统软件及其操作，横线上部是系统软件的名称，横线下部是软件所作的操作。方框表示磁盘文件。

首先，用户编写汇编语言源文件，经过汇编程序进行汇编，产生属性为•OBJ的以二进制代码表示的目标程序并存盘。

然后通过连接程序(LINK)把目标文件与库文件以及其他目标文件连接在一起，形成属性为•EXE的可执行文件，才能在DOS环境下在机器上执行之。

4.1 程序设计语言概述

■ 汇编语言程序使用的系统软件

汇编源程序： 按严格的语法规则用汇编语言编写的程序。

编辑程序： 建立和修改汇编语言源程序，形成属性为 •ASM 的源文件。

常用软件： 全屏幕文本编辑EDIT，记事本NOTEPAD

汇编程序： 对源文件进行汇编，产生属性为•OBJ的以二进制代码表示的目标程序并存盘。

常用软件： 小汇编ASM和宏汇编MASM。

连接程序： OBJ文件经过连接程序(LINK)，把目标文件与库文件以及其它目标文件连接在一起，形成属性为 •EXE 的可执行文件。

4.1 程序设计语言概述

■ 汇编语言常用术语

汇编(过程)： 将汇编源程序翻译成机器码目标程序的过程。分为手工汇编和机器汇编。

驻留汇编： 在小型机上配置汇编程序，并在译出目标程序后在本机上执行，又称为本机自我汇编。

交叉汇编： 多用户终端利用某一大型机的汇编程序进行它机汇编，然后在各终端上执行，以共享大型机的软件资源。

4.1 程序设计语言概述

■ 汇编语言(汇编源程序)与汇编程序

- 汇编语言
  - ✓ 用指令助记符表示指令机器码
  - ✓ 用符号地址表示存储器地址
  - ✓ 用伪指令管理源程序
- 汇编程序

用汇编程序 MASM.EXE 对 *.ASM 源程序进行汇编，将指令助记符翻译为指令机器码

4.1 程序设计语言概述

■ 宏汇编程序MASM

20世纪80年代Microsoft公司推出了MASM 1.0，随着微处理器的升级，MASM也相应改版。

MASM 4.0：支持80286/80287

MASM 5.0：支持80386/80387，增加了简化段定义伪指令和存储模式伪指令，使汇编和连接速度更快。

MASM 6.0：支持80486，提供了许多类似高级语言的新特点。1991年推出的。

MASM 6.11，MASM 6.14：支持Pentium以上高档微处理器，引入了流程控制伪指令。

4.1 程序设计语言概述

■ 汇编语言应用程序开发过程

- **源程序的编辑**
  - ✓ 使用源程序编辑软件 **EDIT.COM**或记事本NOTEPAD
  - ✓ 产生源程序文件　　如：*. ASM
- **目标程序的汇编**
  - ✓ 使用目标程序汇编软件 **MASM.EXE**
  - ✓ 产生目标程序文件　　如：*. OBJ
- **执行程序的连接**
  - ✓ 使用执行程序连接软件 **LINK.EXE**
  - ✓ 产生执行程序文件　　如：*. EXE

4.2 8086/8088汇编语言基本语法

各种机器的汇编语言其语法规则不尽相同，但基本语法结构形式类似。

现以8086/8088汇编语言为例加以具体讨论。

4.2.1 8086/8088汇编源程序实例

在具体讨论8086/8088汇编语言的繁琐语法规则之前，先举一个具有完整段定义格式的汇编源程序(即MASM程序)实例，以便对汇编语言的有关规定和格式有个初步了解。

例：求从1开始连续50个奇数之和，并将结果存放在名字为SUM的字存储单元中。

4.2.1 8086/8088汇编源程序实例

```
DATA    SEGMENT                  ;定义数据段,DATA为段名
SUM     DW      0                ;由符号(叫变量名)SUM指定的内存单元类型定义为一个字,初值为0
DATA    ENDS                     ;定义数据段结束
STACK   SEGMENT    STACK         ;定义堆栈段,这是组合类型伪指令,它规 定在伪指令后须跟STACK类型名
        DB  200  DUP(0)          ;定义堆栈段为200个字节的连续存储区, 且每个字节的值为0
STACK   ENDS                     ;定义堆栈段结束
CODE    SEGMENT                  ;定义代码段
ASSUME  DS:DATA,SS:STACK,CS      ;由ASSUM伪指令定义各段寄存器的内容
        :CODE
```

4.2.1 8086/8088汇编源程序实例

```
START:  MOV     AX,DATA     ;将DS初始化为数据段首址
        MOV     DS,AX       的段值DATA
        MOV     CX,50       ;CX置入循环计数值
        MOV     AX,0        ;清AX累加器
        MOV     BX,1        ;BX置常量1
NEXT:   ADD     AX,BX       ;累加奇数和,计50次
        INC     BX          ;求下一个奇数
        INC     BX
        DEC     CX          ;循环计数器作减1计数
        JNE     NEXT        ;未计完50次时,转至NEXT循环
        MOV     SUM,AX      ;累加和送存SUM单元
        MOV     AH,4CH      ;DOS功能调用语句,机器将结束本
        INT     21H         程序的运行,并返回DOS状态
CODE    ENDS                ;代码段结束
        END     START       ;整个程序汇编结束
```

4.2.1 8086/8088汇编源程序实例

段

汇编源程序一般由若干段组成，每个段都有一个名字(叫段名)。

以SEGMENT作为段的开始，以ENDS作为段的结束，这两者(伪指令)前面都要冠以相同的名字。

段可以从性质上分为代码段、堆栈段、数据段和附加段4种，但代码段与堆栈段不可缺少，数据段与附加段可根据需要设置。

在上面这个例子中，一共定义了3个段:

1个数据段

1个堆栈段

1个代码段

4.2.1 8086/8088汇编源程序实例

提示

每一行只有一条语句，且不能超过128个字符（从MASM 6.0开始可以是512个字符），但一条语句允许有后续行，最后均以回车作结束。

整个源程序必须以END语句来结束，它通知汇编程序停止汇编。END后面的标号START表示该程序执行时的起始地址。

每一条汇编语句最多由4个字段组成，它们均按照一定的规则分别写在一个语句的4个区域内，各区域之间用空格或制表符(TAB键)隔开。

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

1. 语句的种类

在8086/8088汇编语言中，有3种基本语句：

指令语句
伪指令语句
宏指令语句

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

- **指令语句：**是一种执行性语句，它在汇编时，汇编程序将为之产生一一对应的机器目标代码。

汇编指令	机器码
MOV DS, AX	8E D8
ADD AX, BX	03 C3

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

- **伪指令语句：**是一种说明性语句，它在汇编时只为汇编程序提供进行汇编所需要的有关信息。

  如定义符号，分配存储单元，初始化存储器等，而本身并不生成目标代码。

```
DATA SEGMENT
      AA   DW 20H,-30H
DATA ENDS
```

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

- **宏指令：**是以某个宏名字定义的一段指令序列，在汇编时，凡有宏指令出现的地方都将用相应的指令序列的目标代码插入。

  宏指令语句是一般性指令语句的扩展。

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

2. 语句格式

8086/8088的汇编语句一般由4个字段组成，根据其不同种类的语句格式来描述。

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

- 指令语句的格式

[标号:] [前缀] 指令助记符 [操作数表] [;注释]

[ ]表示可以任选的部分;

操作数表是由逗号分隔开的多个操作数。

1) 标号

标号代表“:”后面指令所在的存储地址(逻辑地址)，供JMP、CALL和LOOP等指令作操作数使用，以寻找转移目标地址。

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

2) 前缀

8086/8088中有些特殊指令，它们常作为前缀同其他指令配合使用。

例如，和“串操作指令”(MOVS、CMPS、SCAS、LODS 与STOS)连用的5条“重复指令”(REP、REPE/REPZ、REPNE/REPNZ)

总线封锁指令LOOK等

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

3) 指令助记符

包括8086/8088的指令助记符，以及用宏定义语句定义过的宏指令名。

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

4) 操作数表

对于8086/8088的一般性执行指令，操作数表可以是一个或两个操作数。

若是两个操作数，则称左边操作数为目标操作数，右边操作数为源操作数。

对宏指令来说，可能有多个操作数。操作数之间用逗号分隔。

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

5) 注释

以“；”开始，用来简要说明该指令在程序中的作用(不是重复解释指令本身的功能)，以提高程序的可读性。

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

- 伪指令语句的格式

[名字] 伪指令 [数表] [；注释]

1）名字

可以是标识符定义的常量名、变量名、过程名、段名以及宏名等。

所谓标识符是由字母开头,由字母、数字、特殊字符(如?、下划线、@等)组成的字符串。

默认情况下，汇编程序是不区分大、小写字母的。

注意：名字的后面没有冒号，这是伪指令语句同指令语句在格式上的主要区别。

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

2) MASM中的常用伪指令助记符

分　　类	助　　记　　符
符号定义伪指令	EQU，=，LABEL
数据定义伪指令	DB，DW，DD，DQ，DT，RECORD*，STRUC*
段定义伪指令	SEGMENT，ENDS，GROUP，ASSUME，ORG
模块定义与通信伪指令	EXTRN，PUBLIC，NAME，END
过程定义伪指令	PROC，ENDP
宏处理伪指令	MACRO*，ENDM*，LOCAL*，REPT*，IRPC*，IRP*，PURGE*
条件汇编伪指令	IF，ENDIF，IF1，IF2，IFB，IFNB，IFE，IFDIF，IFDEF，IFNDEF，IFIDN，ELSE
列表伪指令	PAGE，TITLE，SUBTTL，LIST，XLIST，%OUT
其他伪指令	COMMENT，RADIX，INCLUDE，EVEN

表中有* 者为 MASM 所仅有的，其余指令同 ASM。

4.2.2 8086/8088汇编语言语句

4.2.2.1 汇编语言语句的种类和格式

3) 参数表

在伪指令语句的参数表中，包含有用逗号分隔的多个参数，它们可以是常数、变量名、表达式等。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

一条指令必须包括一个指令助记符，以及充分的寻址信息，以使汇编程序能将其转换成一条机器指令的操作码字段及由操作数寻址方式指定的操作数字段。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

1. 标号 (Label)

(1) 标号及其属性

标号是为一组机器指令所起的名字，用来作为汇编语言源程序中转移、调用以及循环等指令的操作数—程序转移的转向地址(目标地址)。

标号表示指令地址, 是指令符号地址, 具有3种属性：

段地址、段内偏移量(或相对地址)以及类型。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

a) 段地址(Segment Base)

标号所在段的段地址(16位数), 是标号所在段的20位起始地址的前16位。

b) 段内偏移量(Offset)

标号与段起始地址之间相距字节数，16位无符号数。

c) 类型(Type)

标号所代表的指令的转移范围，分NEAR(近)与FAR(远)两种。

4.2.2 8086/8088汇编语言语句

- **NEAR类型**的标号仅在同一段内使用，用2字节指针给出转移的偏移量属性(即只改变IP值，不改变CS值)；
- **FAR类型**的标号无此限制，必须用 4 字节指针指出转移的段地址与段内偏移量。
- 当标号用作**JMP或CALL**等指令的目标操作数时，若为段内转移或调用则采用NEAR类型；若为段间转移或调用则应当采用FAR类型。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

1. 标号 (Label)

(2) 标号的定义

a) 标号的组成

标号用一标识符定义，即以字母开头，由字母、数字、特殊字符（如？、下划线、@等）组成的字符串表示。

标号最大长度一般不超过31个字符，除宏指令名外。

标号不能与保留字相同。保留字看上去类似标识符，但它们在语言中有被机器赋于的特殊意义。

标号最好用具有一定含义的英文单词或单词缩写表示，便于阅读。

4.2.2 8086/8088汇编语言语句

8086/8088保留字

① 8086/8088CPU **寄存器**名；

② 8086/8088CPU 指令系统的全部**指令助记符**；

③ 汇编语言的**伪指令**；

④ **其他名字**

ABS	AT	BYTE	COMMENT	CON
DUP	EQ	FAR	GE	HIGH
LE	LENGTH	LINE	LOW	LT
MASK	MEMORY	MOD	NE	NEAR
NOTHING	OFFSET	PAGE	PARA	PREFIX
PROCLEM	PTR	SEG	SHORT	SIZE
STACK	THIS	TYPE	WIDTH	WORD

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

b) 在指令助记符之前，使用标号并紧跟一个冒号“：”，表示该标号被定义为一个类型为NEAR的标号。

当然，标号也可单列一行。

例如：SUBROUT:

MOV AX, 3000H

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

c) 使用过程定义伪指令PROC定义一个“过程”时，为该过程起的名字也是一个标号，该标号作为CALL指令的操作数使用。

PROC 定义格式为：

过程名 PROC NEAR；NEAR可省略

或　过程名 PROC FAR

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

1. 标号 (Label)

(3) 标号的使用

通常，“标号”只在循环、转移和调用指令中使用。

a) 在循环或条件转移指令中：

标号类型必须是NEAR，使用该标号的指令与定义该标号指令的距离必须在 -128～+127字节之间。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

b) 在无条件转移或调用指令中：

段间使用时： 采用FAR类型

段内使用时： 采用NEAR类型较好，也可采用FAR类型。

段内短转移： 采用NEAR类型时，若定义标号与引用标号的两个指令距离在-128～+127字节之间，标号前加运算符SHORT。

段内长转移： 距离在-32768～+32767字节之间。

4.2.2 8086/8088汇编语言语句

范围	寻址方式	操作数类型	操作数使用方式	示例
段内转移	直接	1字节立即数 2字节立即数	加入IP 加入IP	JMP SHORT SUBOUT JMP SUBOUT
	间接	寄存器操作数 存储器操作数(2字节)	送入IP 送入IP	JMP BX JMP WORD PTR [BP+JTABLE]
段间转移	直接	4字节立即数	送入IP及CS	JMP NEXTROUTING
	间接	存储器操作数(4字节)	送入IP及CS	JMP DWORD PTR ADDRESS[BX]

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

2. 指令助记符（Instruction Mnemonics）

执行性指令中的指令助记符主要为8086/8088CPU指令系统中指令助记符。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

3. 操作数（Operand）

操作数的汇编语言表示法及规则比较复杂：

既要能充分体现出汇编语言中使用符号操作数和指令助记符的优越性，使程序员能尽可能地减少在存储分配和地址计算方面的工作；

又要能被汇编程序有效地翻译成对应的特定处理器所具有的各种寻址方式。

立即数操作数
寄存器操作数
存储器操作数

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

(1) 立即操作数

指令中直接给出，不需要使用寄存器，也不涉及访问数据区的操作，只能作为源操作数。

立即操作数是整数，可以是1字节或2字节。

在汇编语言中，立即操作数用常量（包括数值常量和符号常量）以及由常量与有关运算符组成的数值表达式表示。

如：MOV BX，1000+5*3

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

(2) 寄存器操作数

通用寄存器 AX、BX、CX、DX、BP、SP、DI、SI
以及段寄存器 CS、SS、DS、ES 都可以作为操作数。

如：MOV BX，AX

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

(3) 存储器操作数

以指定的存储单元中的内容作为指令的处理对象，汇编指令中的存储器操作数实际上是存储单元的逻辑地址。

4.2.2 8086/8088汇编语言语句

例如：

a) MOV WORD PTR[0A00H], 0000H

b) MOV AX, [BX]

c) DEC BYTE PTR[BP+12H]

d) DEC WORD PTR[SI+33H]

e) DEC WORD PTR[BP+DI+20H]

f) DEC WORD PTR DS: [BP+12H]
ADD AL, SS:[DI+3]

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

4. 各种寻址方式下操作数的表达式

(1) 常量与数值表达式

a) 常量是指在汇编过程中已经有确定数值的量。

主要用作指令语句中的立即操作数、变址寻址和基址加变址寻址中的位移量DISP，或在伪指令语句中用于给变量赋初值。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

b) 常量分“数值常量”与“符号常量”两种。

数值常量：以各种进位制数值形式表示，以后缀字符区分各种进位制。

符号常量：预先给常量定义一个“名字”，然后在汇编语句中用该“名字”表示该常量。

4.2.2 8086/8088汇编语言语句

符号常量：预先给常量定义一个“名字”，然后在汇编语句中用该“名字”表示该常量。

优点：改善程序可读性；如将符号常量作为程序的参数，可方便地实现参数的修改，增强程序的通用性。

其定义需用伪指令：“EQU”或“=”。

例：
```
ONE  EQU 1
DATA1=2*12H
MOV AX，DATA1+ONE
```

即把25H送AX。

常量是没有属性的纯数据，其值是在汇编时确定的。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

c) 各种形式的常量格式

数据形式	格式	取值范围	示　例	注　　解
二进制	XXXXB	0～1	101011B	
八进制	XXXO XXXQ	0～7	765O 765Q	O 是英文字母
十六进制	XXXXH	0～F	7A65H 0FA9H	首位必须是数字
十 进制	XXXXD XXXX	0～9	965D 965	十进制数后缀可省
ASCII 码	'XXX' “XXXX”	ASCII 字 符编码值	'IBM PC' " ok"	由 DB 伪指令定义

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

d) 数值表达

一个能被计算并产生数值的表达式称为数值表达式（Constant Expression）。

一个数值表达式可自由常量、字符常量以及代表常量或串常量的名字等以算术、逻辑和关系运算符（Operator）连接而成。

4.2.2 8086/8088汇编语言语句

① 算术运算符

（只能用于数值表达式）

运算符	运算	举例
+	加	MOV AL，5+6 ；AL ←11
-	减	MOV AL，5-6 ；AL ←0FFH
*	乘	MOV AL，5*6 ；AL ←30
MOD	取余	MOV AL，32 MOD 6 ；AL ←2
SHL	左移	MOV AL，32 SHL 1 ；AL ←64
SHR	右移	MOV AL，32 SHR 1 ；AL ←16

4.2.2 8086/8088汇编语言语句

算术运算符的使用特点：

- 数值表达式中可使用所有算术运算符
- 地址表达式仅使用 +、- 算术运算符
- 算术运算符不影响标志位

例1：完成 80H + 90H

```
解：MOV AL，80H + 90H  ；使用数值表达式
                       ；(AL)= 10H、CF = 不变

    MOV AL，80H
    ADD AL，90H        ；使用加法指令
                       ；(AL)= 10H、CF = 1
```

4.2.2 8086/8088汇编语言语句

例2：求17除7的余数。

```
解：MOV AL，17 MOD 7  ；使用数值表达
                      ；(AL)= 3
```

例3：NUMB=01010101 B, 求左移1位后结果。

```
解：MOV AL，NUMB SHL 1  ；使用数值表达式
                        ；(AL)= 10101010 B
```

4.2.2 8086/8088汇编语言语句

② 逻辑运算符

（只能用于数值表达式）

运算符	运算	举例
AND	与	MOV AL, 5H AND 8FH ; AL ←05H
OR	或	MOV AL, 5H OR 80H ; AL ←85H
XOR	异或	MOV AL,10 XOR 10 ; AL ←00H
NOT	取反	MOV AL, NOT 24H ; AL ←0DBH

4.2.2 8086/8088汇编语言语句

逻辑运算符的使用特点：

- 逻辑指令助记符与逻辑运算符形同意不同

- 逻辑指令助记符出现在指令语句的开始

  AND CX，00FFH AND

- 逻辑运算符出现在指令语句的操作数段

  AND CX，00FFH AND

  注：等价指令 AND CX，00AEH

4.2.2 8086/8088汇编语言语句

例：

MOV AX, 756AH AND 2465H

; (AX)= 2460 H

MOV AL, NOT 25H

; (AL)= DAH

4.2.2 8086/8088汇编语言语句

③ 关系运算符

关系不成立时，结果为全0；关系成立时，结果为全1。（关系式的两个操作数必须是数字或是同一段内的两个地址单元）。设（AI）=6。

运算符	运算	举例
GT	大于	MOV AL，5 GT AI ；AL ←0H
LT	小于	MOV AL，5 LT AI ；AL ←0FFH
EQ	等于	MOV AL，5 EQ AI ；AL ←0H
GE	大等于	MOV AL，5 GE AI ；AL ← 0H
LE	小等于	MOV AL，5 LE AI ；AL ←0FFH
NE	不等于	MOV AL，5 NE AI ；AL ←0H

4.2.2 8086/8088汇编语言语句

关系运算符的使用特点：

- 两性质相同的操作数间的比较

  若关系正确为真（T）， 有全 1 操作数

  若关系错误为假（F）， 有全 0 操作数

- MOV BL，(PORT LT 5) AND 20H
  - ✓ 若 PORT = 2，(PORT LT 5)= FFH，为真
    原式为 MOV BL，20H
  - ✓ 若 PORT = 7，(PORT LT 5)= 00H，为假
    原式为 MOV BL，00H

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

4. 各种寻址方式下操作数的表达式

(2) 变量与地址表达式

a) 变量及其属

“变量”(Aariable)是内存中一个**数据区名字**，即数据所存放地址的符号地址，它可作为指令中的存储器操作数来引用。

存储器分段使用，对源程序中所定义的变量有多种属性。

段属性：与该变量相对应的数据区所在段的段地址。
偏移量属性：该变量与段起始地址相距的字节数。
数据类型属性：数据区中数据项存取单位，字节、字等。

4.2.2 8086/8088汇编语言语句

变量与标号区别：

- 变量指数据区的名字；
  标号是某条执行指令起始地址的符号表示。

- 变量的类型是指数据项存取单位的字节数大小；
  标号的类型指使用该标号的两条指令之间的距离远近，
  即NEAR或FAR。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

b) 变量的定义

在数据段或附加段中使用伪指令DB、DW、DD、DQ和DT来进行定义变量的，这些伪指令称为数据区定义伪指令。

其格式为：

［变量名］ 数据区定义伪指令 表达式

4.2.2 8086/8088汇编语言语句

数据区定义伪指令：

伪指令	数据项类型	数据存取单位
DB	BYTE	1 个字节
DW	WORD	2 个字节
DD	DWORD	4 个字节
DQ	QBYTE	8 个字节
DT	TBYTE	10 个字节

4.2.2 8086/8088汇编语言语句

表达式确定数据区的大小及其初值：

- **数值表达式；**
- **地址表达式：**只适用于DW和DD两条伪指令；
- **？：**表示所定义的数据项无确定初值；
- **n DUP(?)：** DUP为重复因子，定义n个数据项，它们都是未确定的实值。
- **n DUP(表达式)：**定义n个数据项，其初值由表达式确定。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

c) 变量的使用

变量是存储器数据区的符号表示，因此指令中的存储器操作数可以用变量形式给出。

4.2.2 8086/8088汇编语言语句

使用变量作为存储器操作数时要注意几个问题：

① 必须在程序中**明确**一条指令是完成8位数据操作还是16位数据操作。

② 变量作为指令中的存储器操作数使用时，其段属性与该指令使用的缺省段寄存器内容**必须相符**，若不相符则必须使用“跨段前缀”，否则指令无法从存储器中取得正确的操作数进行操作。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

d) 地址表达

① 汇编语言中表达式有两类

- **数值表达式：**在汇编时产生一数值，**仅具有大小而无其他属性**，可作为执行性指令中的立即操作数和数据区中的初值使用。

- **地址表达式：**它表示存储器地址，其值一般都是段内的偏移地址，因此它**具有段属性、偏移值属性、类型属性**。地址表达式主要用来表示执行性指令中的多种形式的操作数。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

d) 地址表达式

② 地址表达式由变量、标号、常量、寄存器BX，BP，SI，DI的内容 (用寄存器名以及方括号表示)以及一些运算符组成。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

d) 地址表达

③ 地址表达式中可使用的运算符及使用规则：

a 加法和减法运算符（＋、－）

- 变量或标号可加上或减去某个结果为整数的数值表达式，其结果仍为变量或标号，类型及段地址属性不变，仅修改偏移值属性。
- 一切数值表达式的运算符都可在地址表达式中出现。同一段内的两个变量或标号可以相减，但结果不是地址，而是一个数值，表示两者间相距的字节数。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

③ 地址表达式中可使用的运算符及使用规则：

b 方括号及寄存器BX,BP,SI,DI

如这几个寄存器不用方括号括起来，表示寄存器本身或操作数。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

③ 地址表达式中可使用的**运算符及使用规则**：

c PTR运算符号

- 说明某个变量、标号或地址表达式的类型属性，或者临时兼有与原定义所不同的类型属性，但保持原来的段属性和偏移地址属性不变。

  格式为：数据类型 PTR地址表达式

- 根据地址表达式的不同值，数据类型可以是BYTE、WORD、DWORD、NEAR、FAR等。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

③ 地址表达式中可使用的运算符及使用规则：

d 段超越运算符

- 临时给变量、标号或地址表达式指定一个段属性。
- 段超越运算符格式：

  段名：地址表达式

  或

  段寄存器名：地址表达式

4.2.2 8086/8088汇编语言语句

例：INC BYTE PTR ES:［BP+3］

ES：为跨段前缀，冒号“:”前的ES段寄存器指明了操作数当前所在的段为附加数据段。

操作数的物理地址将由ES中的内容左移4位与偏移地址[BP+3]相加而求得。

表示将附加数据段中偏移地址为[BP+3]的内存单元中的数据加1后仍保留在该单元中。

跨段时，物理地址的计算是由系统自动完成的。

如果没有跨段前缀“ES:”，那么由[BP+3]地址表达式所表示的偏移地址将被系统默认为是在堆栈段中。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

4. 各种寻址方式下操作数的表达式

(3) 运算符综述

- IBM宏汇编中有**5种**运算符，即：
  - 算术运算符 (Arithmetic operators)
  - 逻辑运算符 (Logical operators)
  - 关系运算符 (Relational operators)
  - 分析运算符 (Analytic operators)
  - 合成运算符 (Synthetic operators)

- 前3种运算符已介绍过，下面介绍后2种运算符。

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

a) 分析运算符

用来把存储器操作数(变量或标号)分解为它的组成部分(**段地址、偏移值、类型 、数据字节总数、数据项总数**等)，并以**数值形式**回送给存储器操作数(变量或标号)。

4.2.2 8086/8088汇编语言语句

SEG	汇编结果将回送变量或标号的**段地**址
OFFSET	汇编结果将回送变量或标号的**偏移值**
TYPE	汇编结果将回送反映变量或标号**类型**的1个数值。 **变量**：数值为字节数，DB为1，DW为2，…… **标号**：数值代表标号类型，NEAR为-1，FAR为-2
SIZE	汇编结果将回送变量数据区的**数据字节**总数
LENGTH	汇编结果将回送变量数据区的**数据项**总数
HIGH	汇编结果取地址表达式或16位绝对值的**高8位**
LOW	汇编结果取地址表达式或16位绝对值的**低8位**

SIZE返回值=LENGTH返回值*TYPE 返回

4.2.2 8086/8088汇编语言语句

例：

```
DATA1  DW 100 DUP(?)
```

则

```
LENGTH DATA1  ; 值为100
SIZE DATA1    ; 值为200
TYPE DATA1    ; 值为2
```

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

b) 合成运算符

用来把存储器操作数的属性部分建立一个新的存储器地址操作数。

只在本语句中有效，并不永久改变变量的属性。

格式为： 类型 PTR 表达式

例：TWO_BYTE DW ?

MOV AL, BYTE PTR TWO_BYTE

4.2.2 8086/8088汇编语言语句

4.2.2.2 指令语句

c) 运算符的优先

优先级	运　算　符
高 ↑	1. LENGTH，SIZE，WIDTH，MASK，（ ），[ ]，<>中的项目，结构变量
	2. 段超越运算符
	3. PTR，OFFSET，SEG，TYPE，THIS
	4. HIGH，LOW
	5. * ，/，MOD，SHL，SHR
	6. +，−
	7. EQ，NE，LT，LE，GT，GE
	8. NOT
低	9. AND
	10. OR，XOR
	11. SHORT

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

伪指令语句又称为说明性指令或指示语句。

- 高级语言程序中的可执行语句被翻译成机器语言时，必须有非执行语句用于实现赋值、保留存储器，给常数分配符号名字、形成数据结构和终止编译等。
- 汇编语言被翻译成机器语言时，也必须包括有执行类似任务的伪指令。
- 8086/8088依靠段寄存器工作，须包括一些在汇编过程中告诉汇编程序把某个段分配给哪一个段寄存器的伪指令。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

伪指令语句格式为：

［名字］ 伪指令助记符 ［参数表］ ［;注释］

名字是一标识符，一般**不能有“：”**结尾，名字可以是**符号常量名、段名、变量名**等，由不同的伪指令决定。

参数表是用“，”分隔开的一系列参数（包括操作数）。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

1. 数据定义伪指令

(1) 5种数据定义命令

- DB —— 变量为字节数据类型（8位）， BYTE
- DW—— 变量为字数据类型（16位）， WORD
- DD —— 变量为双字数据类型（32位）， DWORD
- DQ —— 变量为 4 字数据类型（64位）， QBYTE
- DT —— 变量为 10 字节数据类型（80位），TBYTE

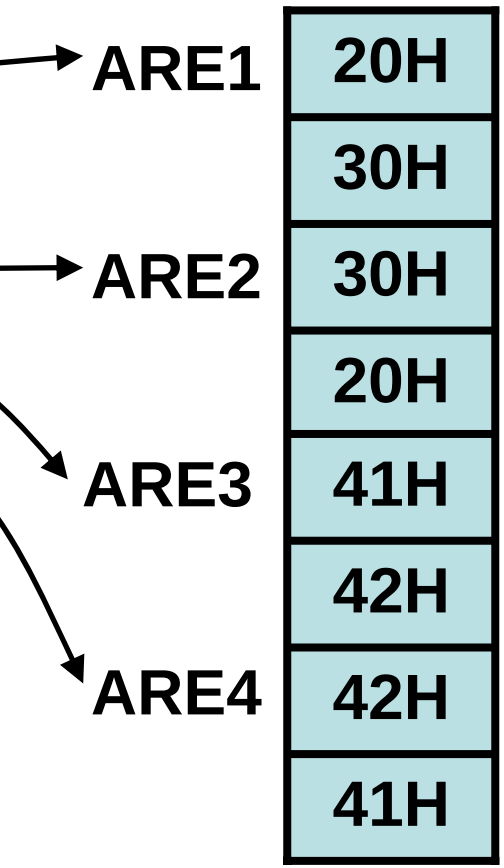
4.2.2 8086/8088汇编语言语句

- DW定义字符串只允许包含两个字符，否则必须用DB指令
- 字符串的个数不超过255个
- 字符串必须用单引号引起来

4.2.2 8086/8088汇编语言语句

例1：DB、DW 的应用特点。

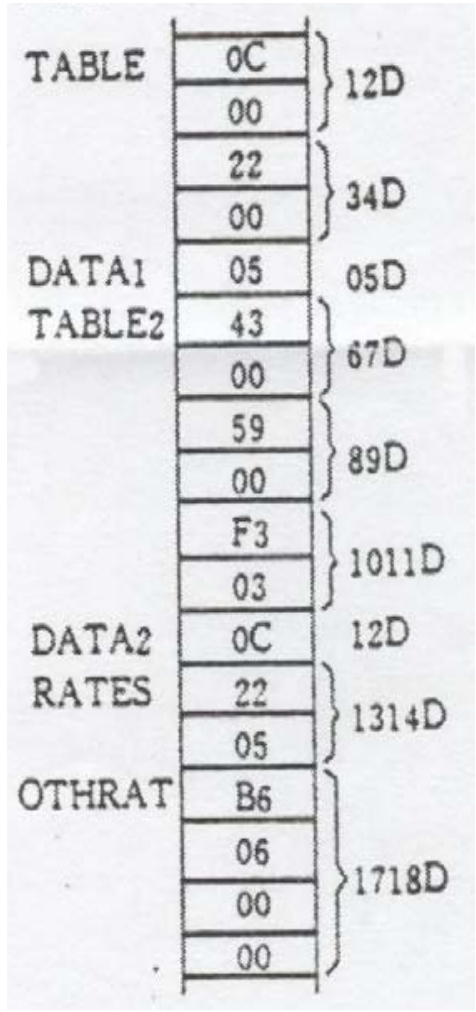
```
DATA  SEGMENT
    ARE1  DB  20H，30H
    ARE2  DW  2030H
    ARE3  DB  ‘AB’
    ARE4  DW  ‘AB’
DATA  ENDS
```



4.2.2 8086/8088汇编语言语句

例2：

```
DSEG  SEGMENT
   TABLE   DW  12      ;OFFSET=0
           DW  34
   DATA1   DB  5       ;OFFSET=4
   TABLE2  DW  67      ;OFFSET=5
           DW  89
           DW  1011
   DATA2   DB  12      ;OFFSET=11
   RATES   DW  1314    ;OFFSET=12
   OTHRAT  DD  1718    ;OFFSET=14
DSEG  ENDS
```



4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

1. 数据定义伪指令

(2) 存储器初始化

- DB、DW、DD可用于**存储器初始化**。一个存储单位可以是字节、字、双字。
- 表达式有**数值表达式与地址表达式**之分：
  使用地址表达式时，只可在DW或DD伪操作命令中出现，绝不允许出现在DB中。

4.2.2 8086/8088汇编语言语句

例3：

```
FOO  SEGMENT AT 55H
    ZERO  DB   0
    ONE   DW   ONE          ；内容为 0001H
    TWO   DD   TWO          ；内容为00550003H
                            ；即高位字为55H，低位字为3
    FOUR  DW   FOUR+5       ；内容为7+5=12
    SIX   DW   ZERO_TWO     ；内容为0-3=-3
    ATE   DW   5*6          ；内容为30
FOO   ENDS
```

4.2.2 8086/8088汇编语言语句

① 从0003H单元开始分配4个存储单元。

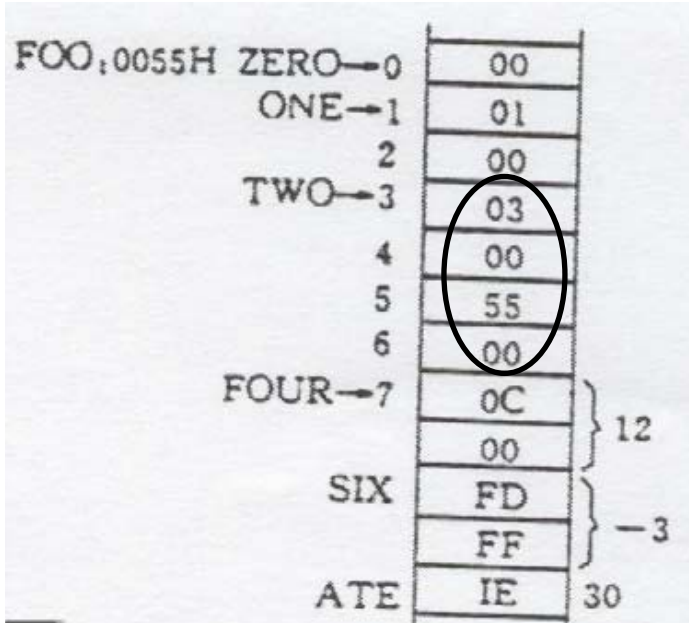
② 为0003H~0006H 4个字节存储单元设置初值。

汇编后将变量TWO的偏移量0003H存入其前2个字节内存单元；

而将段FOO的段地址0055H存入其后2个字节内存单元中。

DD伪指令中的2个字节即表示变量TWO的偏移地址及段地址。

一个字节的操作数也可以是某个字符的ASCII代码，注意只允许在DB伪操作命令中用字符串来初始化存储器。



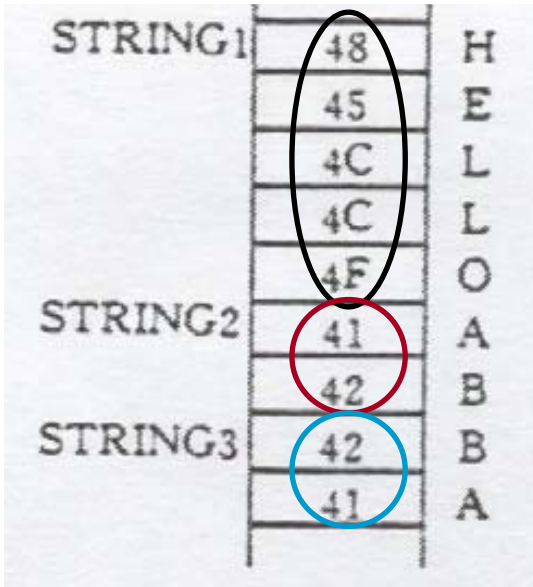
4.2.2 8086/8088汇编语言语句

例4：

STRING1 DB ‘HELLO’

STRING2 DB ‘AB’

STRING3 DW ‘AB’



4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

2. 符号定义伪指令

汇编语言中所有的**变量名、标号名、过程名、指令助记符、寄存器名**等统称“符号”。

这些符号可以通过伪指令重新命名，也可以通过伪指令为其定义其他名字及新的类型属性。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

2. 符号定义伪指令

(1) EQU (赋值伪指令)

a) 为常量定义一个符号，以便在程序中使用符号来表示常量。

格式：
符号常量名 EQU 数值表达式

例：ONE EQU 1
TWO EQU 2
SUM EQU ONE+TWO

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

b) 给变量或标号定义新的类型属性并起一个新的名字。

格式：
变量名或标号名 EQU [类型 PTR] 变量或标号

例：
```
BYTES     DB 4 DUP(?)
FIRSTW    EQU WORD PTR BYTES
FIRSTDW EQU DWORD PTR BYTES
          .......
INCHS：  MOV BYTES, AL
          ........
MILES EQU FAR PTR INCHS
          ........
```

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

c) 给由地址表达式指出的任意存储单元定义一个名字。

格式：
符号名 EQU 地址表达式

符号名可以是“变量”或“标号”，取决于地址表达式的类型。

例：
```
XYZ  EQU [BP+3]
A    EQU ARRAY[BX][SI]
P    EQU ES: ALPHA
```

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

d) 用来为汇编语言中的任何符号定义一个新的名字。

格式：

新的名字 EQU 原符号名

例： COUNT EQU CX

LD EQU MOV

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

e) 使用EQU伪操作命令时，EQU左端的符号名不能是程序已定义过的符号名。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

2. 符号定义伪指令

(2) ＝（等号伪指令）

a) 使用“＝”定义的符号名可以被重新定义，使符号名具有新值。

例：

```
X=12        ；先将12赋于符号名X
X=X+1       ；将符号名X重新定义使其具有新值
```

则在第2个语句经过汇编后，最终X=13。

b) 习惯上“＝”主要用来定义符号常量。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

2. 符号定义伪指令

(3) LABEL（类型定义伪指令）

为当前存储单元定义一个指定类型的变量或标号。

格式为：

变量名或标号名 LABEL 类型

- 对于数据项，类型可以是BYTE、WORD、DWORD；对于可执行的指令代码，类型为NEAR和FAR。
- LABEL伪指令不仅给名字（标号或变量）定义一个类型属性，而且隐含有给名字定义段属性和段内偏移量属性。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

3. 段定义伪指令

段定义伪指令指示汇编程序应如何按段来组织程序和使用存储器。

命令主要有**SEGMENT，ENDS，ASSUME，ORG**等。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

(1) SEGMENT和ENDS伪指令

用来把程序模块中的指令或语句分成若干逻辑段。

格式如下：

段名 SEGMENT [定位类型] [组合类型] [‘类别’]
… ；一系列汇编指令

段名 ENDS

必须成对出现, SEGMENT与ENDS之间为段体，给其赋于一个名字，名字由用户指定，是不可省略的，而定位类型、组合类型和类别是可选的。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

a) 定位类型

类型	段物理地址20位	意义
BYTE(字节型)	XXXX XXXX XXXX XXXX XXXX	段可起始于任意地址
WORD(字型)	XXXX XXXX XXXX XXXX XXX0	段可起始于偶地址
PARA(节型)	XXXX XXXX XXXX XXXX 0000	段起始地址是16整数倍
PAGE(页型)	XXXX XXXX XXXX 0000 0000	段起始地址是256整数倍

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

b) 组合类型

组合类型又称“联合方式”或“连接类型”。

指示连接程序，如何将某段与其他段组合起来的关系。

连接程序不但可将不同模块的同名段进行组合，并根据组合类型，可将各段顺序地连接在一起或重叠在一起。

共有6种组合类型。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

c) 类别

类别是用单引号括起来的字符串，表示该段的类别，连接程序只使同类别的段发生关联，连接时用于组成段组的名字。

典型的类别如‘STACK’、’CODE’ 、’DATA’等，也允许用户在类别中用其他的表示。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

(2) ASSUME伪指令

段定义伪指令，一般出现在代码段中，告诉汇编程序哪一个段寄存器是其对应段的段地址寄存器。

它也可用来取消某段寄存器与其原来设定段之间的对应关系(使用NOTHING即可)。

引用该伪指令后，汇编程序才能对使用变量或标号的指令汇编出正确的目标代码。

格式为：ASSUME　段寄存器：段名, [段寄存器名：段名]

例：　　ASSUME CS: SEGA, DS: SEGB, SS: NOTHING

4.2.2 8086/8088汇编语言语句

在汇编时代码段寄存器的赋值是在程序初始化时自动完成的，而其它段寄存器的赋值须在代码段的编写中通过MOV指令来赋值。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

(3) ORG 伪指令

用来指出其后的程序段或数据块存放的起始地址偏移量。

格式为：

ORG 表达式

汇编程序把语句中表达式之值作为起始地址，连续存放程序和数据，直到出现一个新的ORG指令。

若省略ORG，则从本段起始地址开始连续存放。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

4. 过程定义伪指令

(1) 过程（分2类）

外部过程： 当调用该过程的主程序与该过程不在一个源程序文件中时，该过程应定义成外部过程。这时在主程序文件中应说明该过程（设过程名为PROCD）为外部过程：

```
EXTRN   PROCD：FAR
```

在定义该过程的程序文件中应说明该过程可被其他程序文件调用：

```
PUBLIC   PROCD
```

内部过程： 当调用该过程的主程序与该过程在同一个源程序文件中时，该过程叫做内部过程。

4.2.2 8086/8088汇编语言语句

4.2.2.3 伪指令语句

(2) 过程定义伪指令格式

过程名 PROC 〔类型〕
…　；指令序列
过程名 ENDP

类型可选作NEAR或FAR。如果类型省略，则系统取NEAR类型。

选NEAR时，该过程一定要与主程序在一个段；

选FAR时，该子程序可以与主程序在同一个段，也可与主程序不在同一个段。

4.2.2.3 伪指令语句

(3) 调用过程

用“CALL 过程名”来实现。

其中过程名是个标识符，可作为调用此过程的指令中的操作数。

过程可“嵌套”使用，即过程中又可调用别的过程；

过程还可“递归”使用，即过程中又可调用过程本身。

调用指令使用要和过程属性一致，一个过程已定义为FAR的话，即使当前调用指令和此过程有同一代码段，也必须使用段间调用指令，否则将会出错。

4.2.2.3 伪指令语句

(4) 过程返回

通常子程序中包括一至多条返回指令，即当过程运行至某种条件满足时返回至主程序中调用指令的下一条指令继续执行。

4.2.2 8086/8088汇编语言语句

* 过程名为过程的入口地址（首地址）
* PROC和ENDP指令须成对使用
* 过程中至少要设置一条RET指令
* 过程与代码段可嵌套，但不可交叉
* 远过程可由其它代码段调用，近过程只能由本代码段调用

4.3 8086/8088汇编语言程序设计基本方法

- 与高级语言编程相似，编写汇编语言源程序也应首先理解和分析题意与要求，选择适当的数据结构和算法，然后，再着手用汇编语言来实现。

- 8086/8088汇编源程序的结构及语法规则，前面已有详述。可知，DOS环境下的8086/8088汇编语言程序结束时，通常用DOS的4CH号中断调用，以便使程序控制返回DOS。

```
MOV AH, 4CH
INT 21H
```

4.3 8086/8088汇编语言程序设计基本方法

在计算机上建立和运行汇编语言的过程为：

1. 用编辑程序建立 .ASM源程序文件；
2. 用汇编程序（MASM）把.ASM文件汇编成 .OBJ文件；
3. 用连接程序（LINK）把OBJ文件转换成 .EXE文件；
4. 在DOS命令状态下直接键入文件名就可执行该文件。

   （若在程序中没有用到DOS输出功能调用指令，则执行后在屏幕上看不到执行的结果）

一般初次编写的源程序需要在执行.EXE文件之前，先在DEBUG环境下进行调试，调试完后，再执行。

4.3 8086/8088汇编语言程序设计基本方法

程序设计应力求：

1. 程序结构化；
2. 简明、易读、易调试；
3. 执行速度快，占用存储空间少（充分利用CPU中的存器）。

4.3 8086/8088汇编语言程序设计基本方法

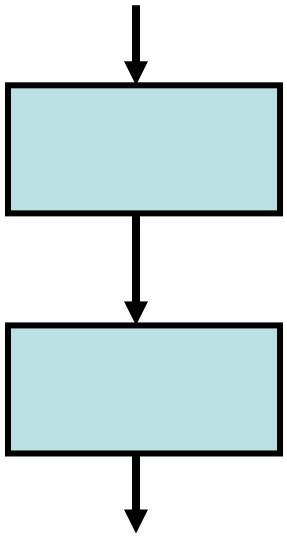
程序设计步骤：

1. 分析问题，抽象出数学模型；
2. 确定实现数学模型的算式；
3. 绘制描述程序的流程图（包括确定内存单元和分配寄存器）；
4. 编写源程序；
5. 上机运行调试。

4.3 8086/8088汇编语言程序设计基本方法

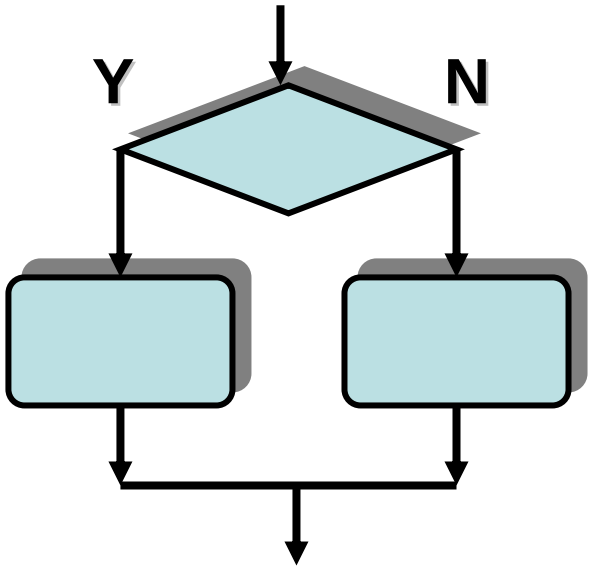
■ 程序的基本结构

顺序程序结构



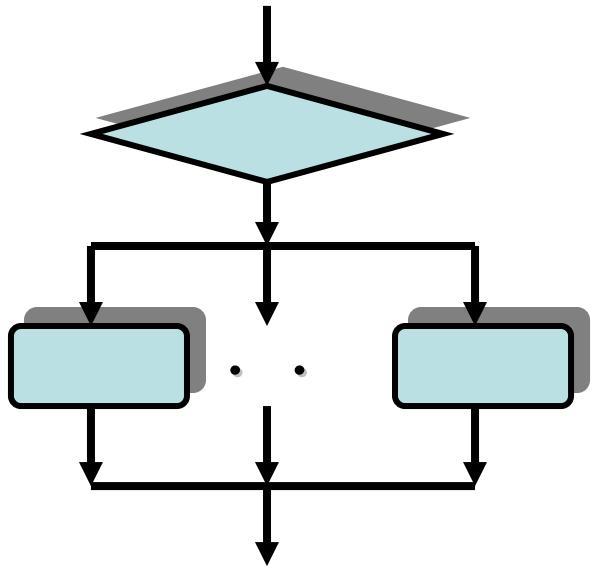
一般语句

条件程序结构



If – else 语句

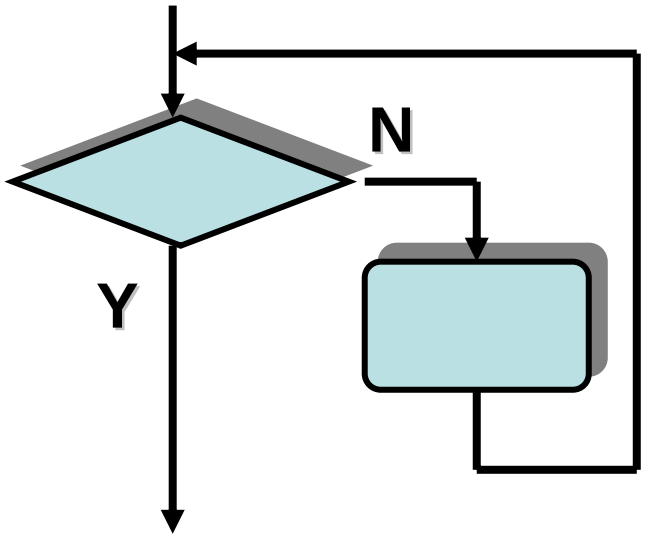
分支程序结构



Switch 语句

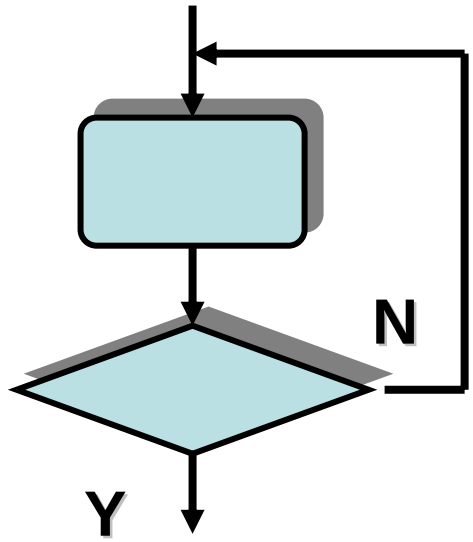
■ 程序的基本结构

循环控制结构



while 语句

循环控制结构



do --while 语句

■ 基本程序设计

- 顺序程序设计

  指令指针 IP 值线性增加，IP = IP + 1

- 条件程序设计

  IP 值受标志位的影响而跳变
  影响标志的指令 CMP、TEST、JXX

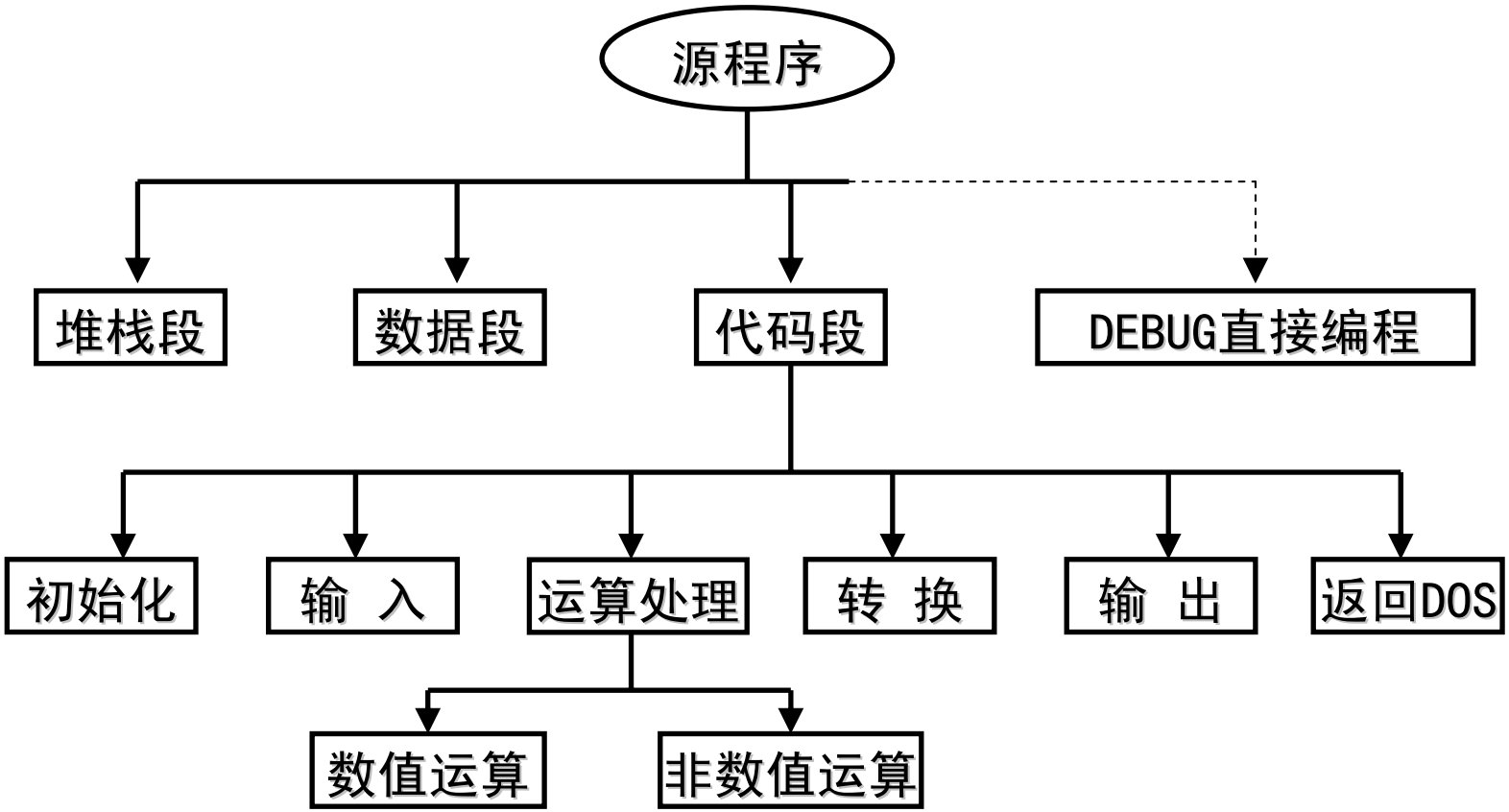
- 循环程序设计

  IP 值受计数器 CX 中的值不为零而循环

  影响标志的指令 DEC

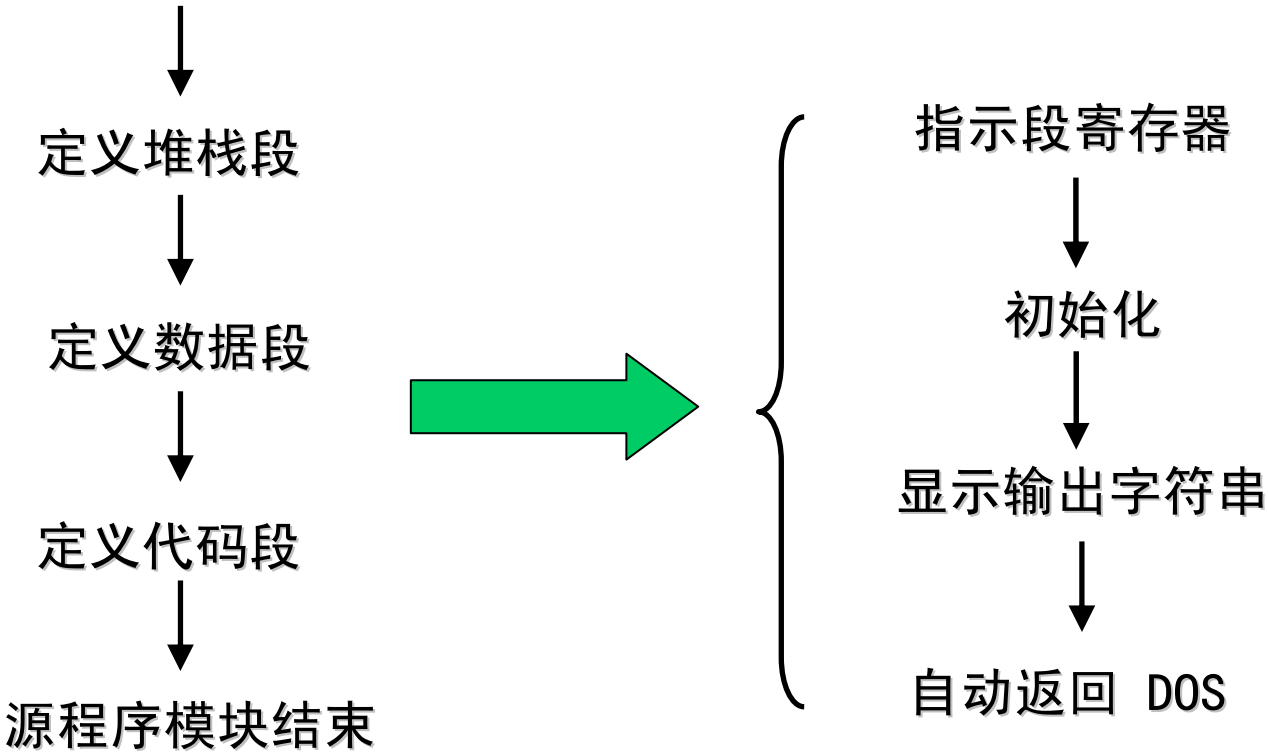
4.3 8086/8088汇编语言程序设计基本方法

■ 程序的层次模块结构



4.3 8086/8088汇编语言程序设计基本方法

例：程序输出显示：“ Welcome !”，流程描述如下。



4.3 8086/8088汇编语言程序设计基本方法

```
STACKS   SEGMENT   STACK
     DW   128   DUP(?)
STACKS   ENDS

DATAS    SEGMENT
     STRING   DB   'Welcome!',13h,10h,'$'
DATAS    ENDS

CODES    SEGMENT
     ASSUME CS:CODE,DS:DATAS
START: MOV   AX,  DATAS
       MOV   DS,  AX
       LEA   DX,  STRING
       MOV   AH,  09H
       INT   21H
       MOV   AX,  4C00H
       INT   21H
CODES    ENDS
       END   START
```

堆栈

数据

代码

源程序模块结束

4.3.1 顺序结构程序

例1：编程计算W=X+Y+24-Z。其中变量X、Y、Z均为32位数。

```
DATA   SEGMENT
   X  DW 19, 86
   Y  DW 3, 25
   Z  DW 1987, 325
   W DW 2DUP(?)
DATA   ENDS

CODE   SEGMENT
       ASSUME CS:CODE,
                    DS:DATA
START:   MOV AX, DATA
         MOV DS, AX

         MOV AX, X
         MOV DX, X+2
         ADD AX, Y
         ADC DX, Y+2

         ADD AX, 24
         ADC DX, 0

         SUB AX, Z
         SBB DX, Z+2

         MOV W,  AX
         MOV W+2, DX

         MOV AH, 4CH
         INT 21H
CODE ENDS
         END START
```

4.3.1 顺序结构程序

例2： 用变量D1及D2表示的两个8字节无符号数相加。
两数之和按从高到低依次放在SI，BX，CX，DX中。

```
DATA   SEGMENT
   D1  DB 12H,34H,56H,78H,9AH,0ABH,0BCH,0CDH
   D2  DB 0CDH,0BCH,0ABH,9AH,78H,56H,34H,12H
DATA   ENDS

CODE   SEGMENT
       ASSUME CS:CODE, DS:DATA
BG:    MOV AX,DATA        ;给DS赋段值
       MOV DS,AX
```

4.3.1 顺序结构程序

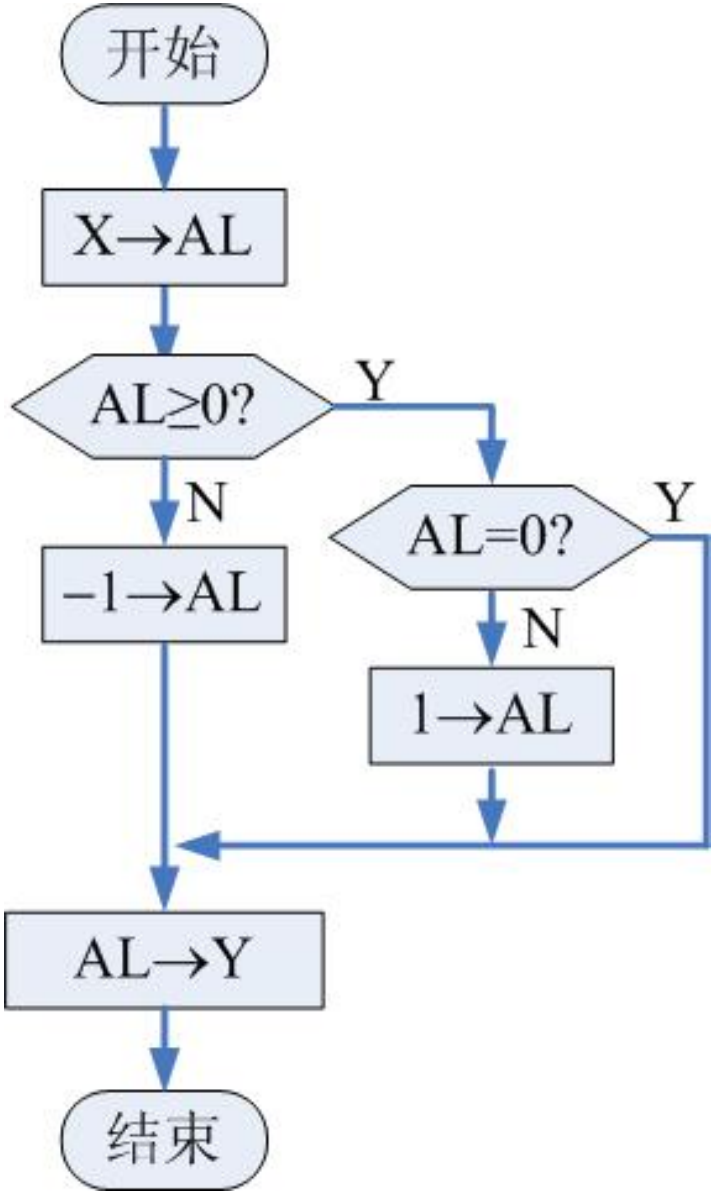
```
        LEA DI,D1                ;将D1表示的偏移地址送DI
        MOV DX,[DI]
        MOV CX, [ DI+2 ]         ;取操作数到寄存器中
        MOV BX, [ DI+4 ]
        MOV SI, [ DI+6 ]
        LEA DI,D2                ;将D2表示的偏移地址送DI
        ADD DX, [ DI ]
        ADC CX, [ DI+2 ]
        ADC BX, [ DI+4 ]
        ADC SI, [ DI+6 ]
        MOV AH,4CH
        INT 21H
CODE    ENDS
        END  BG
```

4.3.2 分支结构程序

例3：符号函数

$$Y=\begin{cases}1 & X>0\\ 0 & X=0\\ -1 & X<0\end{cases}$$

设X为8位有符号数。



4.3.2 分支结构程序

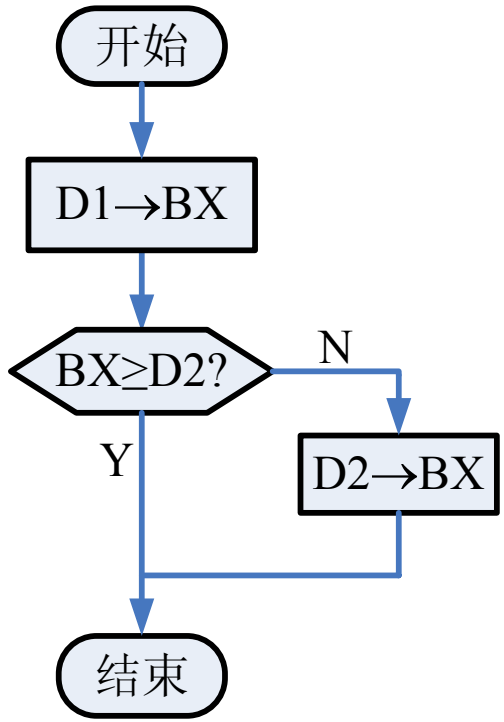
```
DATA   SEGMENT
   X  DB  -18
   Y  DB  ?
DATA   ENDS

CODE   SEGMENT
       ASSUME CS:CODE,
                    DS:DATA

START:   MOV AX, DATA
         MOV DS, AX
         MOV AL, X
         CMP AL, 0
         JGE BIGR
         MOV AL, -1
         JMP STOP
BIGR:    JE STOP
         MOV AL, 1
STOP:    MOV Y, AL
         MOV AH, 4CH
         INT 21H
CODE     ENDS
         END START
```

4.3.2 分支结构程序

例4： 比较变量D1和D2表示的两个有符号字数大小，将其中较大数据放在BX寄存器中。



4.3.2 分支结构程序

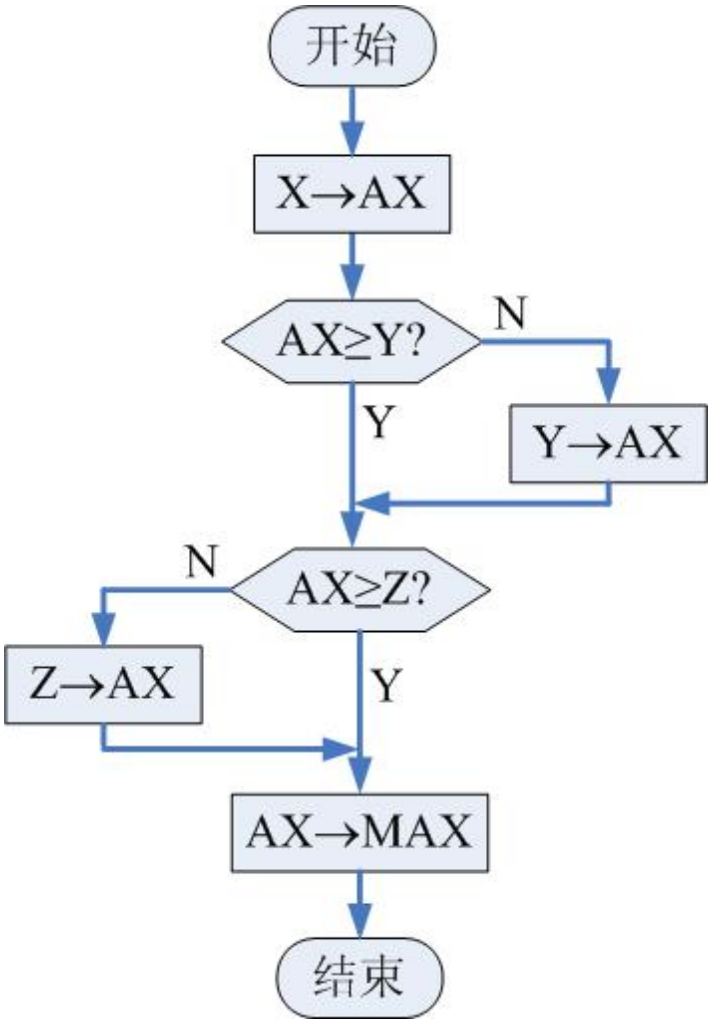
```
DATA    SEGMENT
    D1      DW -123H                ;补码为FF85H
    D2      DW -120H                ;补码为FF88H
DATA    ENDS

CODE    SEGMENT
            ASSUME CS:CODE,DS:DATA

BEGIN:      MOV AX, DATA
            MOV DS, AX              ;给DS赋段值
            MOV BX, D1
            CMP BX, D2
            JGE NEXT                ;若D1≥D2，则不交换，转NEXT
            MOV BX, D2              ;若D1<D2，则交换
NEXT:       MOV AH, 4CH
            INT 21H
CODE    ENDS
            END BEGIN
```

4.3.2 分支结构程序

例5： 已知X、Y、Z为三个无符号16位二进制数，求三者中最大值， 送MAX单元。



4.3.2 分支结构程序

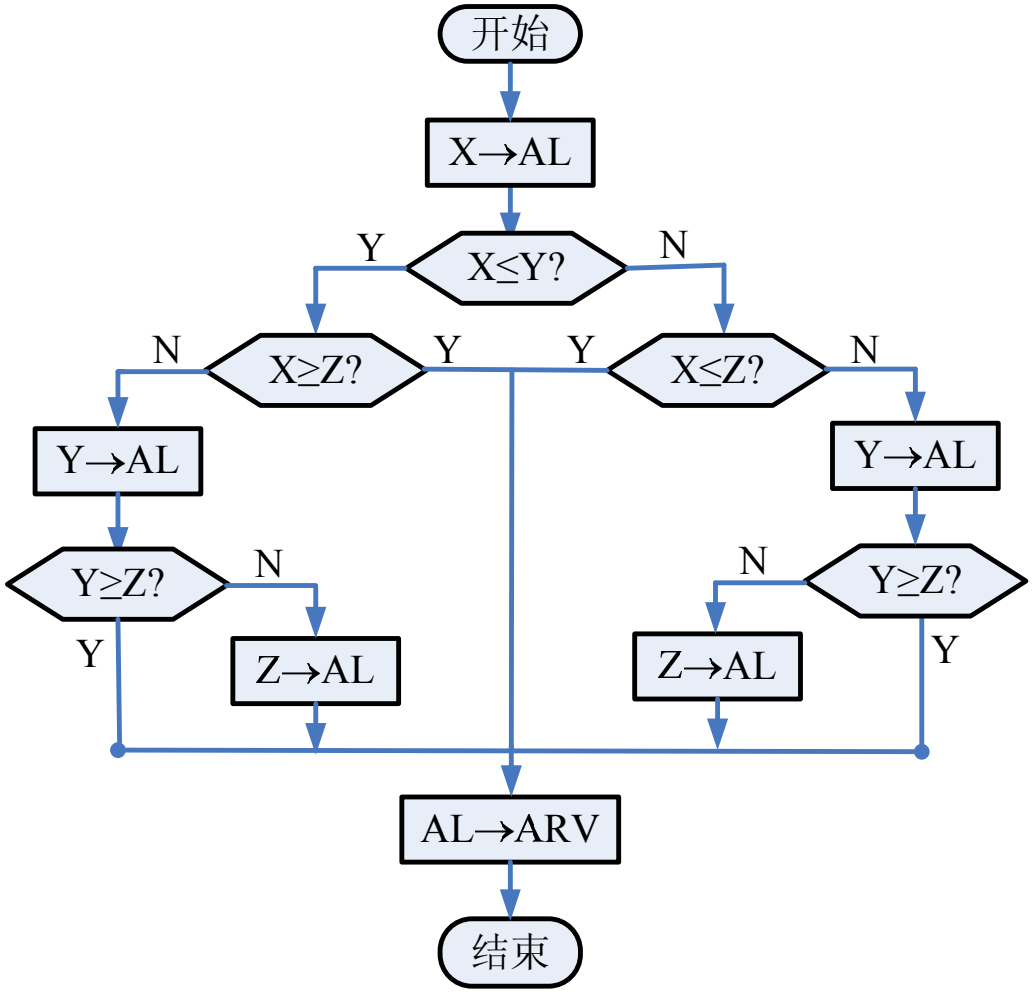
```
DATA    SEGMENT
    X     DW   180
    Y     DW   850
    Z     DW   350
    MAX DW   ?
DATA    ENDS

CODE   SEGMENT
        ASSUME CS:CODE,
                    DS:DATA
START:  MOV AX, DATA
        MOV DS, AX
        MOV AX, X
        CMP AX, Y
        JAE XGY
        MOV AX, Y
XGY:    CMP AX, Z
        JAE GMAX
        MOV AX, Z
GMAX:   MOV MAX, AX
        MOV AH, 4CH
        INT 21H
CODE   ENDS
        END START
```

4.3.2 分支结构程序

例6：已知X、Y、Z为三个带符号8位二进制数，

求三者中中间值，送ARV单元。



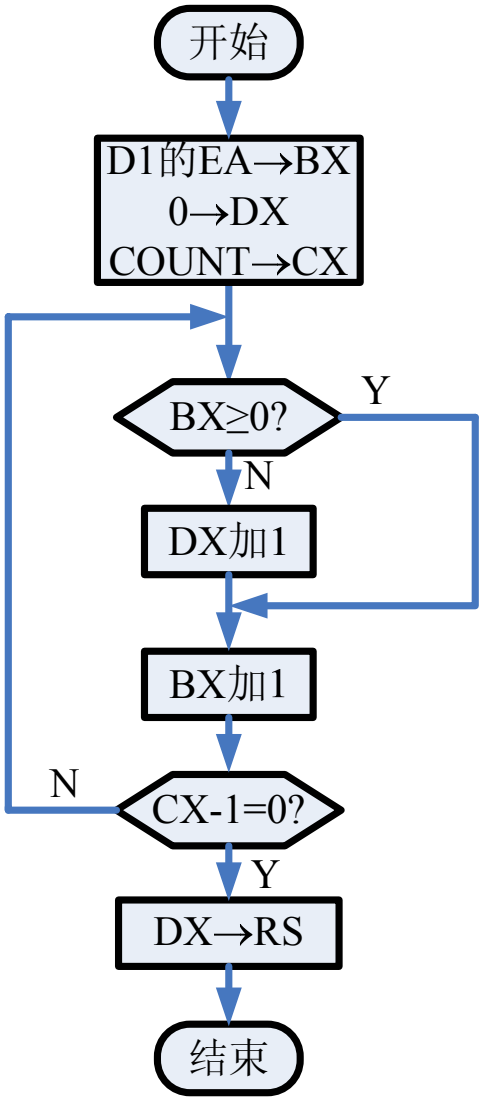
4.3.2 分支结构程序

```
DATA    SEGMENT
    X     DB   -5
    Y     DB   4
    Z     DB   6
    ARV DB   ?
DATA    ENDS

CODE   SEGMENT
        ASSUME CS:CODE,
                      DS:DATA
START:    MOV AX, DATA
          MOV DS, AX
          MOV AL, X
          CMP AL, Y
          JL XLY
          CMP AL, Z
          JLE MARV
          MOV AL, Y
          CMP AL, Z
          JGE MARV
          MOV AL, Z
          JMP MARV
XLY:      CMP AL, Z
          JGE MARV
          MOV AL, Y
          CMP AL, Z
          JLE MARV
          MOV AL, Z
MARV:     MOV ARV, AL
          MOV AH, 4CH
          INT 21H
CODE   ENDS
        END START
```

4.3.3 循环结构程序

例7：统计一个数据块中负数的个数，个数送 RS中。



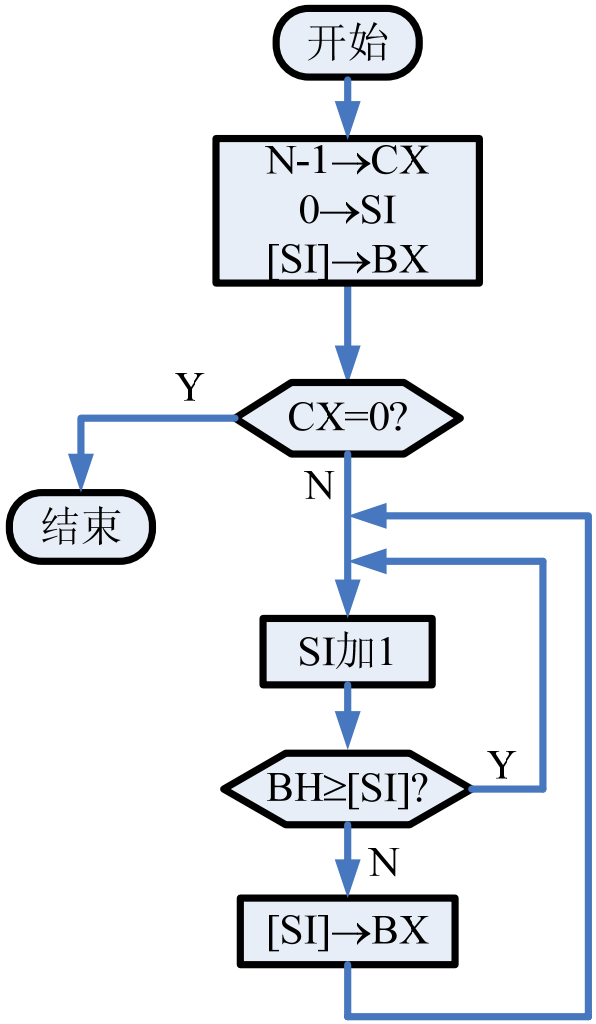
4.3.3 循环结构程序

```
DATA    SEGMENT
    D1    DB   -1, -3, 5, 6, 9,  -5
    COUNT EQU $-D1
    RS DW  ?
DATA    ENDS

CODE   SEGMENT
          ASSUME CS:CODE,
                       DS:DATA
START:     MOV AX, DATA
           MOV DS, AX
BEGIN:     MOV AX, DATA
           MOV DS, AX
           MOV BX, OFFSET D1
           MOV CX, COUNT
           MOV DX, 0
LOP1:      MOV AL, [BX]
           CMP AL, 0
           JGE JUS
           INC DX
JUS:       INC BX
           DEC CX
           JNZ LOP1
           MOV RS, DX
           MOV AH, 4CH
           INT 21H
CODE   ENDS
           END BEGIN
```

4.3.3 循环结构程序

例8： 找出从无符号字节数据存储变量VAR开始存放的N个数中的最大数放在BH中。



4.3.3 循环结构程序

```
DSEG    SEGMENT
  VAR   DB 5,7,19H,23H,0A0H
  N     EQU $-VAR
DSEG    ENDS
CSEG    SEGMENT
      ASSUME  CS:CSEG,
                DS:DSEG
```

```
BG:     MOV AX, DSEG
        MOV DS, AX
        MOV CX, N-1
        MOV SI, 0
        MOV BH, VAR［SI］
        JCXZ LAST
AGIN:   INC SI
        CMP BH, VAR［SI］
        JAE NEXT
        MOV BH, VAR［SI］
NEXT:   LOOP AGIN
LAST:   MOV AH, 4CH
        INT 21H
CSEG    ENDS
        END  BG
```

4.3.4 DOS及BIOS中断调用

为节省编程工作量与优化程序结构，在DOS及BIOS中预先设计好了一系列通用子程序，以便供DOS及BIOS调用。

由于这种调用采用的是以中断指令INT n的内部中断方式进行的，所以常称为DOS及BIOS中断调用。

在一个中断服务程序中往往包含有多个功能相对独立的子程序，所以也将中断调用称为系统功能调用或功能调用或中断功能调用。

4.3.4 DOS及BIOS中断调用

4.3.4.1 DOS模块和ROM BIOS的关系

IBM PC及兼容机的ROM中有一系列外部设备管理软件，组成了基本的输入输出系统(ROM BIOS)。

DOS在此基础上开发了输入输出设备处理程序IBMBIO.COM，这也是DOS与ROM BIOS的接口。

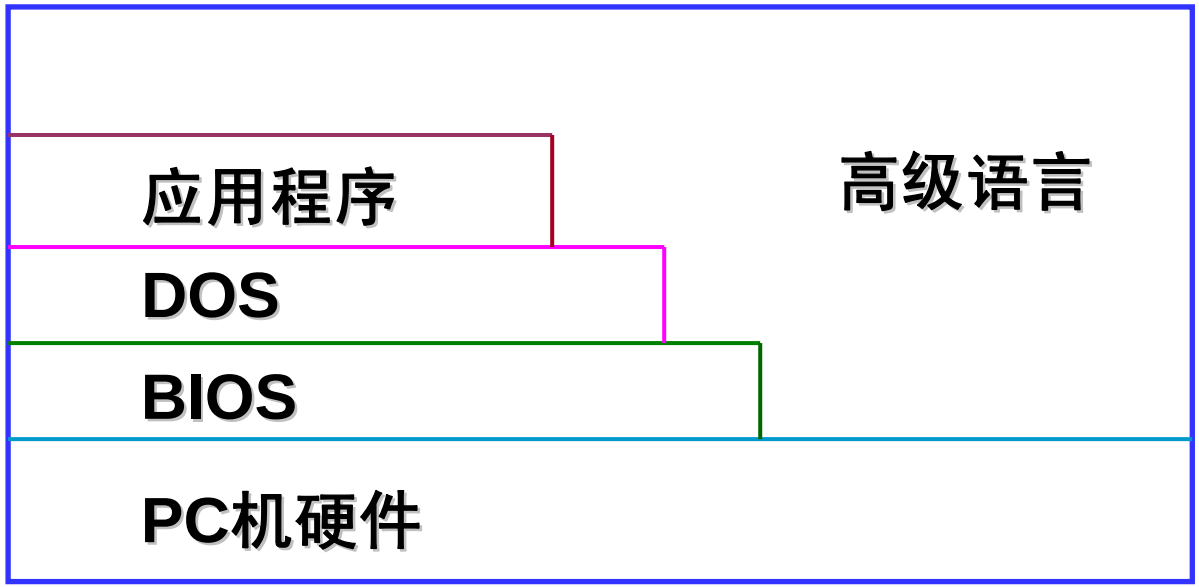
在IBMBIO.COM基础上，DOS还开发了文件管理和一系列处理程序IBMDOS.COM。

另外，DOS还有命令处理程序COMMAND.COM，它与前两种程序构成基本DOS系统。

4.3.4 DOS及BIOS中断调用

4.3.4.1 DOS模块和ROM BIOS的关系

通常用户程序通过IBMDOS.COM使用外部设备。使用汇编语言编程，可以直接使用ROM BIOS中的“中断程序”，甚至还可以直接用 IN和OUT 指令对设备端口编程。



4.3.4 DOS及BIOS中断调用

4.3.4.2 中断调用及中断服务子程序返回

中断调用是一种内部中断方式，通过执行INT n 指令来实现。

即执行INT n指令，使CPU根据中断类型号“n”，找中断向量表中第n项作为此服务程序的入口。

0段相对地址4*n+0处字为IP，0段相对地址4*n+2处字为CS。

INT n指令功能：

①当前标志寄存器的内容压栈，保存TF

② TF ← 0 , TF ← 0

③当前断点的CS值压栈，当前IP值压栈

④IP，CS←中断向量第 n 项的4字节内容

4.3.4 DOS及BIOS中断调用

4.3.4.2 中断调用及中断服务子程序返回

中断向量分配情况如下：

0~1FH，80H~F0H 是ROM BIOS的中断向量号。
20H~3FH 是DOS的中断向量号，
40H~7FH 用户备用。

一个中断服务程序有多种功能，每一种功能用一个相应的编号表示，称为**功能号**。

对应某一中断向量的某一功能，往往要指出其规定的输入参数，中断服务完毕后，服务程序会有相应的输出。

中断调用的步骤如下：

①准备入口参数
②功能号送AH
③INT n

4.3.4 DOS及BIOS中断调用

4.3.4.2 中断调用及中断服务子程序返回

当中断服务子程序返回时，要执行IRET指令。

其功能是:

①栈顶弹出一个字到 IP

②栈顶弹出一个字到 CS

③栈顶弹出一个字到标志寄存器

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

DOS功能调用：指DOS为系统程序员和用户提供的一组常用子程序。

用中断指令INT 21H进入各功能调用子程序的总入口，再为每个功能调用规定一个功能号以便进入相应各子程序的入口。

子程序的入口参数及出口参数在每个功能调用的说明中可以查到。

在DOS中断服务程序中，有近百个功能供用户选择使用，其中，功能最多的是向量号为21H的向量中断。

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

所有DOS提供给用户的功能调用格式都是一样的，一般分为4个步骤：

① 在AH寄存器中设置系统功能调用号；

② 在指定的寄存器中设置中设置入口参数；

③ 用INT 21H指令执行功能调用；

④ 根据出口参数分析功能调用的执行情况。

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

1. 返回 DOS

向量号 21H

功能号 4CH

功能： 使系统结束程序运行后返回DOS状态。

例：
```
MOV   AH，4CH
INT   21H
```

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

2．键盘输入并显示

向量号 21H

功能号 1

功能： 从键盘输入1个字符，将其ASCII码保存在AL中，输入字符回显在CRT上。

例：
```
MOV AH, 1
INT  21H
```

中断返回时，输入字符的ASCII码被存放在AL中，该字符并显示在屏幕上。

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

3. 键盘输入但不显示输入字符

向量号 21H

功能号 8

功能： 输入一字符，其 ASCII 码存放在AL中，但不显示。往往在设置口令时使用。

例：

```
MOV AH, 8
INT 21H
```

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

4. 显示一字符

向量号 21H

功能号 2

入口参数： DL= 待显示字符的 ASCII 码

功能： 显示DL中的字符。

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

5. 在打印机上打印一字符

向量号 21H

功能号 5

入口参数： DL= 待打印字符的 ASCII 码

例： 打印数字 9

```
MOV   AH，  5
MOV   DL，’9’
INT   21H
```

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

6. 显示以“$”结尾的字符串

向量号 21H

功能号 9

入口参数： DS：DX 指向字符串的首地址

例： 在显示器上显示 “How are you？”。
然后读一个字符，但不显示此字符。
若读入字符是“Y”，则显示“OK”。

4.3.4 DOS及BIOS中断调用

```
D       SEGMENT
D1      DB 'HOW ARE YOU?', 0DH, 0AH, '$'     ; 0D回车， 0A换行
D2      DB 'OK', 0DH, 0AH, '$'
D       ENDS
C       SEGMENT
        ASSUME CS: C, DS: D                  ; 说明代码段、数据段
BG:     MOV AX, D
        MOV DS, AX                           ; 给DS赋段值
        MOV DX, OFFSET D1
        MOV AH, 9
        INT 21H                              ; 显示"HOW ARE YOU ?"
        MOV AH, 8
        INT 21H                              ; 不显示方式读一字符到AL
        CMP AL, 'Y'
        JNE NEXT                             ; 不等则转
        LEA DX, D2
        MOV AH, 9
        INT 21H
NEXT:   MOV AH, 4CH
        INT 21H
C       ENDS
        END  BG
```

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

7. 字符串输入

向量号 21H

功能号 0AH

入口参数： DS：DX 指向输入缓冲区

输入缓冲区格式如下：

第1字节： 预定的最大输入字符数。

第2字节： 空出，待中断服务程序填入键盘连续输入到回车前实际输入字符数。

第3字节及以后字节： 待中断服务程序填入输入字符串的ASCII码。

4.3.4 DOS及BIOS中断调用

例： 屏幕显示 “PASSWORD？”。

随后从键盘读入字符串，并比较这个字符串与程序内部设定的字符串。

若二者相同则显示“OK”，否则不作任何显示 (0DH 是回车的 ASCII 码，0AH 是换行的 ASCII码)。

4.3.4 DOS及BIOS中断调用

```
D       SEGMENT
PASS1   DB   '12AB'
N       EQU  $-PASS1
D1      DB   'PASSWORD ?', 0DH, 0AH, '$'
PASS2   DB   20
        DB   ?
        DB   20 DUP(?)
D2      DB   0DH, 0AH, 'OK$'
D       ENDS
C       SEGMENT
        ASSUME CS:C, DS:D, ES:D      ; 说明代码段、数据段、附加段
BG:     MOV  AX, D
        MOV  DS, AX                  ; 给DS赋段值
        MOV  ES, AX                  ; 给ES赋段值
        LEA  DX, D1                  ; 将D1表示的相对地址送DX
        MOV  AH, 9
        INT  21H                     ;显示"PASSWORD ?"并回车换行
        LEA  DX, PASS2
```

4.3.4 DOS及BIOS中断调用

```
        MOV AH,0AH
        INT 21H                     ;输入字符串
        LEA SI,PASS1
        LEA DI,PASS2
        CMP BYTE PTR [DI+1] ,N
        JNE  LAST                   ;或者JNZ LAST也可以
        MOV CX,N
        LEA DI,PASS2+2
        CLD
        REPZ CMPSB                  ;  重复比较
        JZ DISOK
LAST:   MOV AH,4CH
        INT 21H
DISOK:  LEA DX,D2
        MOV AH,9
        INT 21H                     ;显示“OK”
        JMP LAST
C       ENDS
        END  BG
```

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

8. 异步通信口输入

向量号 21H

功能号 3

功能：从标准异步通信口等待输入一字符，然后送AL中。

启动DOS时，异步通信口波特率为2400，设有偶校验位，数据长度为8个二进制位。

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

9. 异步通信口输出

向量号 21H

功能号 4

入口参数： DL= 待输出的数据

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

10. 设置日期

向量号 21H

功能号 2BH

入口参数： CX= 年号（范围：1980～2099）

DX=月份（1～12）

DL=日

4.3.4 DOS及BIOS中断调用

例： 将微机日期设置成1994年1月2日，可用下列程序段实现：

```
MOV  CX，1994
MOV  DH，1
MOV  DL，2
MOV  AH，2BH
INT   21H
```

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

11. 取日期

向量号 21H

功能号 2AH

与功能 2BH 操作相反

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

12. 设置时间

向量号 21H

功能号 2DH

入口参数： CH= 小时（0～23）

CL= 分（0～59）

DH= 秒（0～59）

DL= 百分之一秒（0～99）

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

13. 取时间

向量号 21H

功能号 2CH

与功能 2DH 操作相反

4.3.4 DOS及BIOS中断调用

4.3.4.3 DOS常用功能调用举例

DOS的功能调用远不只这些，通过这一部分功能调用介绍，可以掌握如何调用DOS中断功能子程序的方法，实现通用外部设备的输入输出和利用系统提供的基本功能。

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

前面介绍的DOS中断功能，只是简单概括了BIOS中的某些功能。进一步了解BIOS中断功能有如下几个理由：

①调用BIOS中断程序虽然比调用DOS中断程序要复杂一些，但运行速度快，功能更强；

②DOS的中断功能只是在DOS的环境下适用，而BIOS功能调用则不受任何操作系统的约束；

③某些功能只有BIOS具有。 BIOS中颇具特色的有显示中断子程序，其向量号是10H。此外，还有读键盘中断子程序（其向量号为16H）与通信口中断子程序（其向量号为14H）。

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

1. 设置显示方式

向量号 10 H

功能号 0

入口参数： AL=显示方式号（0～7）

4.3.4 DOS及BIOS中断调用

方式含意如下：

显示方式号	显示方式
0	40列×25行 黑白文本方式
1	40列×25行 彩色文本方式
2	80列×25行 黑白文本方式
3	80列×25行 彩色文本方式
4	320列×200行 黑白图形方式
5	320列×200行 彩色图形方式
6	640列×200行 黑白图形方式
7	单显80列×25行 黑白文本方式

4.3.4 DOS及BIOS中断调用

例： 屏幕设置成 80×25 彩色文本方式。

```
MOV    AH，0    ；设功能号
MOV    AL，3    ；设显示方式号
INT    10H
```

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

2. 设置光标大小

向量号 10 H

功能号 1

入口参数： CH =光标顶值（范围：0～11）

CL =光标底值（范围：1～12）

例： 将光标置成一个闪烁方块。

```
MOV AH, 1
MOV CH,  0
MOV CL, 12
INT    10H
```

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

3. 设置光标位置

向量号 10 H

功能号 2

入口参数：BH =页号，通常取 0页；
DH =行号，取值 0～24；
D L =列号，对于40列文本，取值 0～39
对于80列文本，取值 0～79

例：将光标置在第10行30列。

```
MOV  BH,  0
MOV  DH, 10
MOV  DL,  30
MOV  AH,  2
INT   10H
```

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

4. 屏幕上滚

向量号 10 H

功能号 6

入口参数： AL=上滚行数，AL=0时，清除屏幕矩形方框；

CH、CL= 矩形方框左上角行、列号；

DH、DL= 矩形方框右下角行、列号；

BH= 增加空行的属性

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

5. **屏幕下滚**

向量号 10 H

功能号 7

与功能6滚动方向相反，其它相同。

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

6. 在当前光标处写字符和属性

向量号 10 H

功能号 9

入口参数：BH=页号；

AL= 显示字符的ASCII码

BL= 属性；

CX= 重复显示次数

4.3.4 DOS及BIOS中断调用

例：用兰色清屏，然后在第10行30列显示20个红底白字“A”。

```
MOV AL, 0
MOV BH, 10H
MOV AH, 6
MOV CX, 0
MOV DH, 24
MOV DL, 79
INT 10H                ; 清屏幕
MOV AH, 2
MOV BH,0
MOV DH, 10
MOV DL, 30
INT 10H                ; 光标设置在第10行30列
MOV AL, 'A'
MOV CX, 20
MOV BH, 0
MOV BL, 47H            ; 显示属性是：红底白色
MOV AH, 9
INT 10H                ; 显示20个字母‘A’
```

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

7. 在光标位置写字符（不改属性）

向量号 10 H

功能号 0AH

入口参数： BH=页号；

AL= 显示字符的ASCII码

CX= 重复显示次数

其功能与功能号 9 的功能类似。

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

8. 设置图形方式显示的背景和彩色组

向量号 10 H

功能号 0BH

入口参数：BH=0时，BL=背景颜色，范围0～15；

BH=1时，BL=颜色组，范围0～1；

0 表示 绿/红/黄，

1 表示 青/品红/白

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

9. 写光点

向量号 10 H

功能号 0CH

入口参数：DX= 行号

CX= 列号

AL= 彩色值（当AL第7位为1时，原显示彩色与当前彩色作按位相加运算）

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

10. 读点

向量号 10 H

功能号 0DH

该功能与功能 0 C H操作相反

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

11. 读当前显示状态

向量号 10 H

功能号 0FH

返回参数：AL=当前显示方式（见0号功能）

BH=当前页号

AH=屏幕字符列数

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

以下为ROM BIOS 键盘管理中断:

12. 读键盘

① 向量号 16 H
功能号 0

返回参数：AL=输入字符的ASCII码

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

以下为ROM BIOS 键盘管理中断:

12. 读键盘

② 向量号 16 H

功能号 1

返回参数:

若按过键(键盘缓冲区不空)，则ZF标志置0，AL=输入的ASCII码;

若没有按键，则ZF标志置1。

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

以下为ROM BIOS 键盘管理中断:

12. 读键盘

③ 向量号 16 H

功能号 2

返回参数： AL=特殊功能键的状态

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

以下为ROM BIOS的异步通信功能:

13. 通讯口初始化

向量号 14 H

功能号 0

返回参数：AL=初始化参数

DX=0 表示对COM1初始化；

DX=1 表示对COM2初始化。

初始化参数格式：

7 6 5 4 3 2 1 0

4.3.4 DOS及BIOS中断调用

位 7 、 6 、 5 表示波特率：

0	0	0	110 波特
0	0	1	150 波特
0	1	0	300 波特
0	1	1	600 波特
1	0	0	1200 波特
1	0	1	2400 波特
1	1	0	4800 波特
1	1	1	9600 波特

位 4 、 3 表示奇偶校验设定：

0	0	无奇偶校验
0	1	奇校验
1	1	偶校验

位 2 表示终止位数设定：

0	1 位终止位
1	2 位终止位

位 1 、 0 表示通信数据位数设定：

1	0	7 位通信数据
1	1	8 位通信数据

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

以下为ROM BIOS的异步通信功能:

14. 通讯口输出

向量号 14 H

功能号 1

返回参数：AL=待输出的数据

DX=0 表示对COM1输出；

DX=1 表示对COM2输出。

4.3.4 DOS及BIOS中断调用

4.3.4.4 ROM DOS常用中断调用举例

以下为ROM BIOS的异步通信功能:

15. 通讯口输入

向量号 14 H

功能号 2

入口参数: DX=0 表示对COM1输入;
DX=1 表示对COM2输入。

返回参数:

输入成功时，AH 第7位=0，AL=输入数据;
输入失败时，AH 第7位=1，第0到6位=通信口状态。

4.3.4 DOS及BIOS中断调用

通信口状态格式：

7	6	5	4	3	2	1	0

位 0 为 1 时表示接收数据准备好；
位 1 为 1 时表示超越错；
位 2 为 1 时表示奇偶错；
位 3 为 1 时表示帧格式错；
位 4 为 1 时表示间断；
位 5 为 1 时表示发送保持器空；
位 6 为 1 时表示发送移位寄存器空；
位 7 为 0 时表示正确；为 1 时表示出错。


Question？
第四章部分1结束！