



计算机组成原理与汇编语言程序设计

(第 2 版)

徐
洁
编

洁

俸远桢

主

第 5 章

第 5 章 汇编语言层

本章主要内容：

80x86 宏汇编语言的语句格式

80x86 宏汇编语言的语法规则

基本程序结构及程序设计方法

汇编语言程序的开发方法

第1节 概述



1. 汇编语言

一种**面向机器**的**低级**程序设计语言
符号化的机器语言，汇编指令与机器指令一一对应。

2. 汇编语言源程序

用汇编语言编制的程序；不能由计算机直接执行。

3. 汇编程序

应用**汇编程序**将**汇编语言源程序**翻译成**目标代码**（即**机器语言程序**）

4. 汇编语言的特点

高性能

对计算机的完全控制

有助于更好地使用高级语言编程

有助于了解计算机的结构

第2节 汇编语言语句格式

汇编语言源程序的基本组成单位是语句。

1. 语句的种类

(1) 指令语句（可执行语句）

表示计算机的某种具体操作，汇编时产生指令代码（即目标代码），在程序运行时实现。

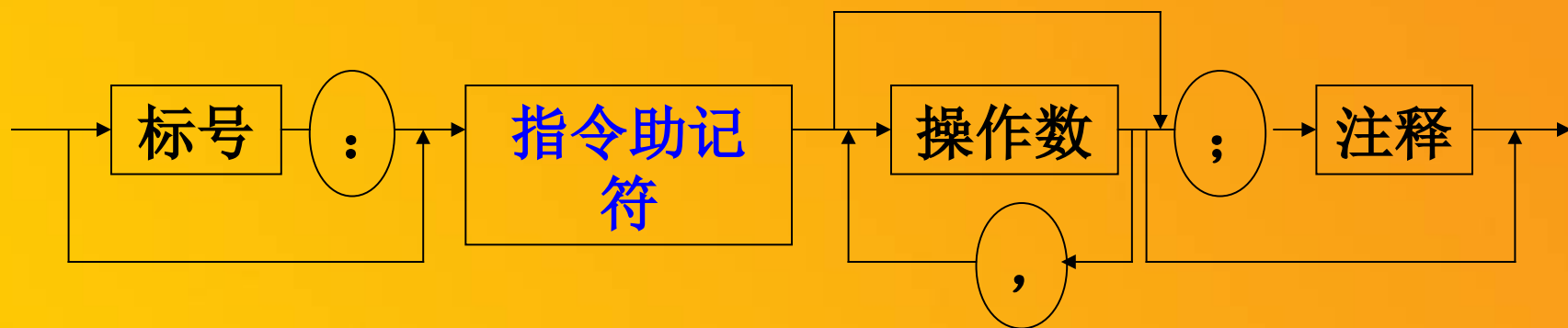
(2) 伪指令语句

指示汇编程序如何对源程序进行汇编，其功能在汇编时完成。除了所定义的数据项之外，其它项不产生目标代码。在第4节中介绍

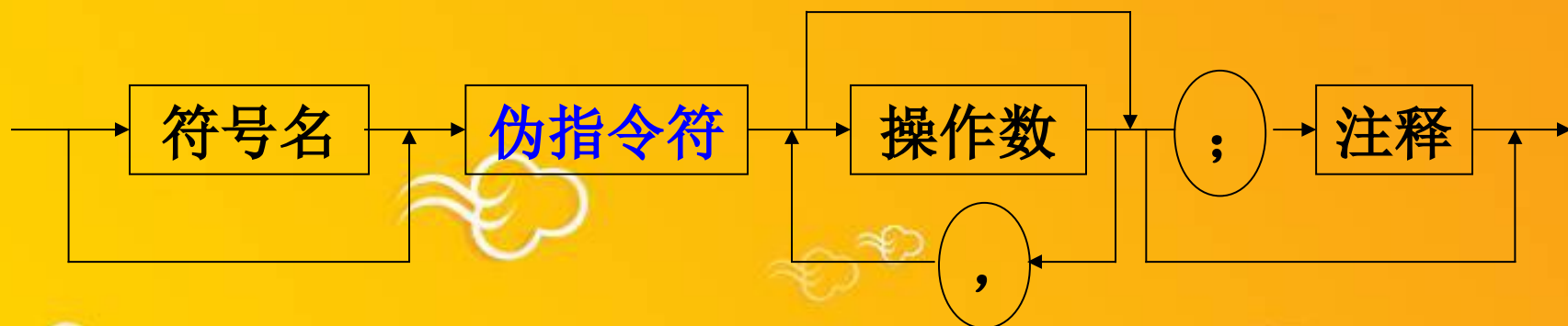
(3) 宏指令语句

2. 语句格式

(1) 指令语句格式



(2) 伪指令语句格式



3. 标识符

标号和符号名统称为标识符，由若干字符构成，规则如下：

- 字符个数 1 ~ 31 ；
- 第一个字符必须是字母或 5 个特殊字符之一
(? @ 下划线 _ 点号 · \$) ；
- 从第二个字符开始，可以是字母 、 数字和特殊字符；
- 不能与系统专用保留字相同。



第3节 80x86 宏汇编语言数据、表达式和运算符

5.3.1 常数

纯数值数据、无属性、值不能改变

1. 数值常数

可用二进制、八进制、十进制、十六进制数表示
如 11001010B、73Q、345D、4aEH、0AH

2. 字符常数

单引号或双引号扩起来的一个或多个字符，以ASCII码存储。如 'A' 的ASCII码为41H

5.3.2 变量

变量应先**定义**并**预置**初值，才能被引用

1、变量定义

数据定义伪指令实现变量的定义，格式如下

变量名 数据定义伪指令 <表达式 1>, ...

...
↓
可选

{ DB 定义字节
DW 定义字
DD 定义双字
DQ 定义 8 字节
DT 定义 10 字节

例:

DATA1 DB 10H



变量的 3 个属性

(1) 段属性 (SE

C) 表示变量存放在哪个逻辑段中，用变量所在段的段基值表示。

(2) 偏移地址属性 (OFFSET)

表示变量在逻辑段中离段起始单元的距离，用字节数表示。

上述两个属性构成了变量的**逻辑地址**。

(3) 类型属性 (TYPE)

表示单个变量占存储单元的字节数。

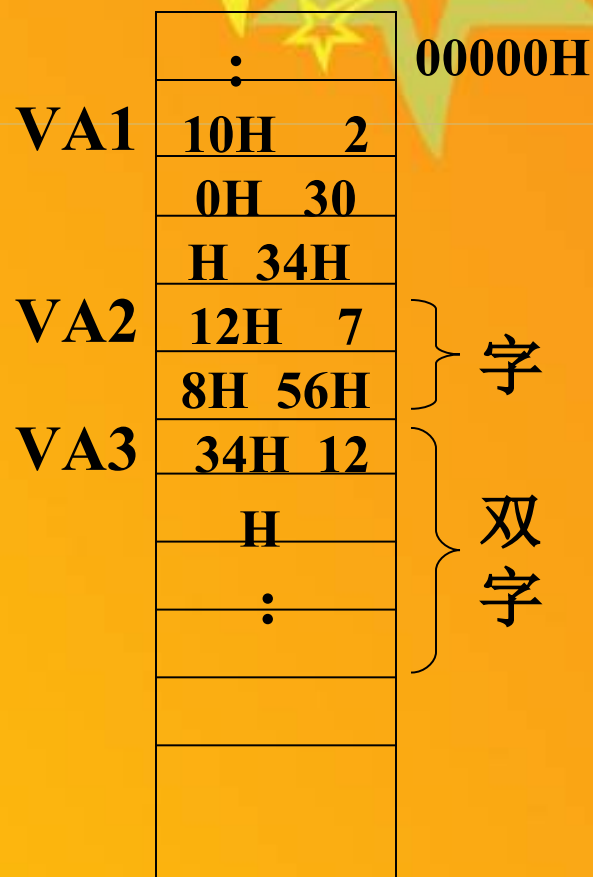
变量的初值

- 数值表达式

例:

```
DATA SEGMENT
VA1 DB 10H
      DB 20H, 30H
VA2 DW 1234H
VA3 DD 12345678H

DATA ENDS
```

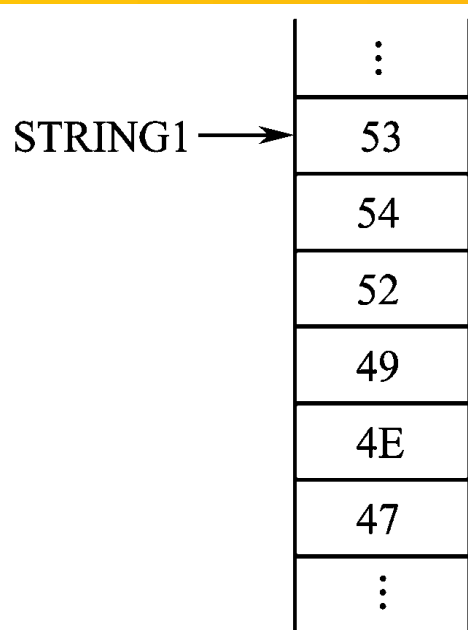


FFFFFH

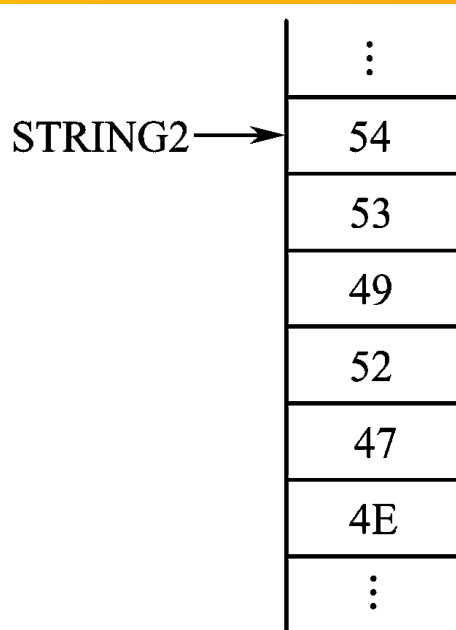
存储器分配图

● 字符串表达式 存放字符的 ASCII 码

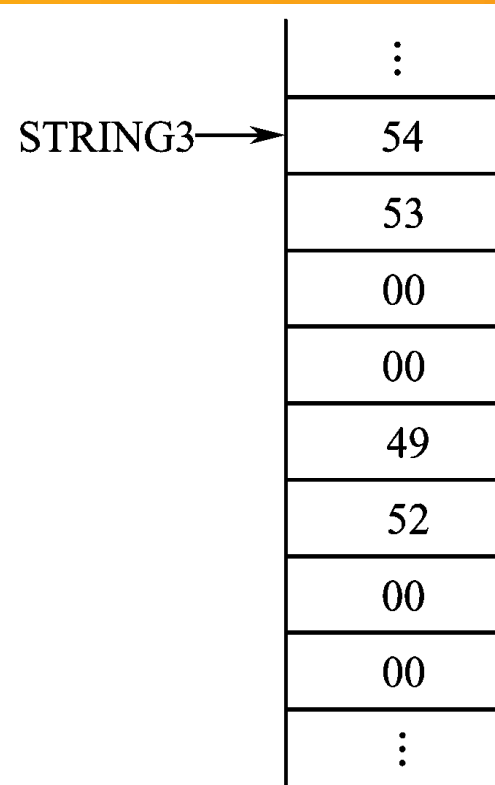
例 : STRING1 DB 'STRING'
 STRING2 DW 'ST', 'RI ', 'NG '
 STRING3 DD 'ST', 'RI ', 'NG '



(a) DB伪指令



(b) DW伪指令



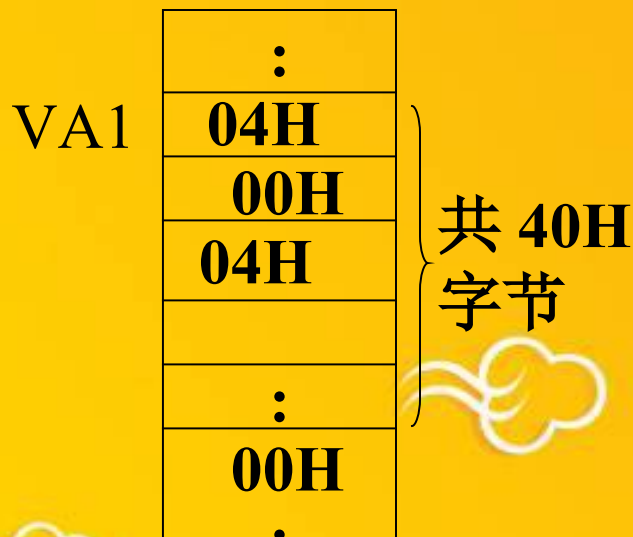
(c) DD伪指令

- “ ? ” 表达式
只分配存储单元，不指定初值

例： DB ? , ? , ?

- 带 DUP 表达式
为连续存储单元重复预置一组数据，格式如下：

变量名 < 数据定义伪指令 > < 表达式 1 > DUP (< 表达式 2 >)



↓
重复次数

↓
重复数据的内容

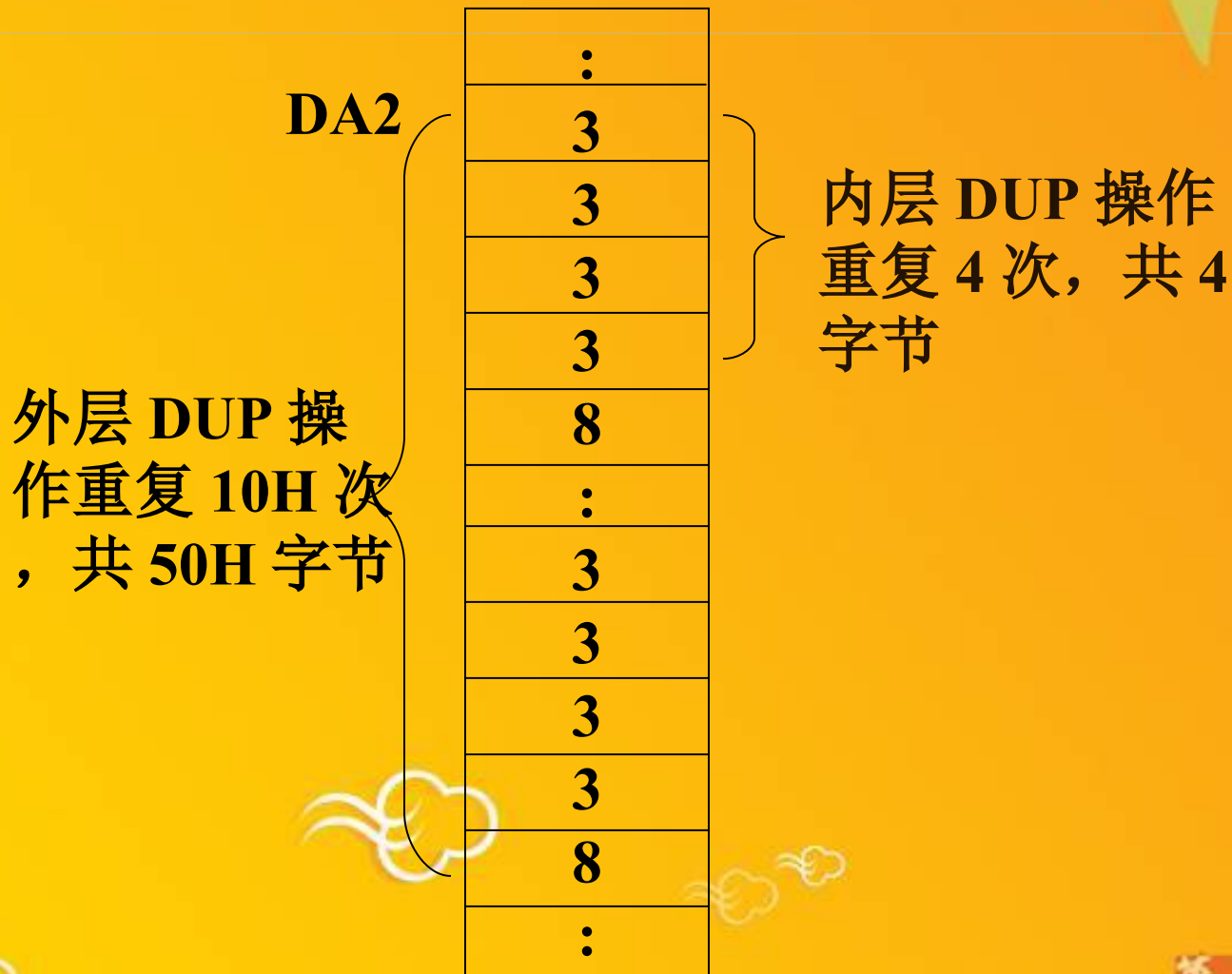
例：

VA1 DW 20H DUP(4)

存储器分配图

DUP 操作符的嵌套使用

DA2 DB 10H DUP (4 DUP (3), 8)



存储器分配图

2、变量的使用 引用变量名

(1) 在指令语句中引用变量名
变量名作为地址表达式的组成部分之一

例如，设在某数据段中有如下的变量定义：

VAR DB 40H DUP (?)

那么几种含有变量名的地址表达式如下：

直接寻址： VAR、 VAR+08H

变址寻址： VAR [SI]、 VAR+5 [DI]

基址寻址： VAR [BX]、 VAR+10H [BP]

基址变址寻址： VAR [BX][DI]、 VAR+06H [BP][SI]

(2) 在数据定义语句中引用变量名

在 DW 或 DD 数据定义语句的操作数字段上引用了变量名，那么在为 DW 或 DD 伪指令分配的存储单元中，将预置被引用变量名的地址部分（段基值和偏移地址）。

{ DW : 引用变量名的偏移地址
DD : 引用变量的段基值和偏移地址

例如: NUM1 DB 10H DUP (?)
NUM2 DW 10H DUP (?)
ARRAY DB 10H DUP ('ABCD')
ADR1 DW ARRAY
ADR2 DD ARRAY

5.3.3 标号

指令的**符号地址**，可作为转移类指令的**目标地址**。

例如，有程序段如下：

LOP : INC SI

:

JNZ **LOP**

1. 标号的属性

(1) 段属性 (SE 表示指令在哪个逻辑段中

（2）偏移地址属性（OFFSET）

表示这条指令目标代码的首字节离段起始单元之间的字节数。

上述两个属性构成了指令的逻辑地址。

（3）类型属性（TYPE）

表示指令的转移特性

{ NEAR（近）段内转移
FAR（远）段间转移

2. 标号类型的设置

（1）隐含方式 直接指定指令的标号

例如：NEXT: MOV AX, 3000H

(2) 用 LABEL 伪指令设置标号类型
格式如下:

名字 LABEL 类型

{ 标号
NEAR/FAR
变量名 }

BYTE/WORD/
DWORD

LABEL 语句与指令语句配合使用:

例: SUB1_FAR LABEL FAR

SUB1 : MOV AX , 1234H

MOV 语句有两个具有相同段和偏移地址属性的标号:
SUB1_FAR 和 SUB1 , 但类型属性不同。

LABEL 语句与数据定义语句配合使用

例： DATA_BYTE LABEL BYTE

DATA_WORD DW 20H DUP(567H)

DATA_WORD 和 DATA_BYTE 具有相同的段和偏移地址属性，但类型属性不同。

有语句如下：

MOV AX, DATA_WORD+4 传送第 3 个字
 (5 、 6 字节)

MOV AL, DATA_BYTE+4 传送第 5 个字节

5.3.4 表达式与运算符

表达式

常用作指令语句或伪指令语句的**操作数**；
由常数、变量、标号通过运算符连接而成；
有**数值表达式**和**地址表达式**；

汇编时，经计算得到一个**数值或地址**。

运算符

算术运算符

逻辑运算符

关系运算符

数值返回运算符

属性运算符

1、算术运算符

包括： +（加）、 -（减）、 *（乘）、 /（除）、 MOD（模除）、 SHL（左移）、 SHR（右移）

+、**-**、*****、**/** 运算的操作数和运算结果都是整数；

除法运算取商的整数， **MOD** 运算取除法的余数；

减法运算可用于同一段内的两个变量；

例： NUM=15*5

NUM=NUM / 8

NUM=NUM **MOD** 5

NUM=NUM + 4

NUM=NUM **SHR** 2

2、逻辑运算符

AND、OR、XOR、NOT

只用于数值表达式；按位进行逻辑操作；
在汇编过程中完成运算；通常出现在源操作数中。

例：
MOV AL, NOT 0F0H
MOV BL, 55H OR 0F0H
AND BH, 55H AND
0F0H

3、关系运算符， 55H XOR 50H

EQ（相等）、NE（不等）、LT（小于）、LE（小于等于）、GT（大于）、GE（大于等于）

格式： <表达式 1> <关系运算符> <表达式 2>

比较两表达式的值，两表达式的性质相同；

数值按无符号数比较，地址表达式比较偏移量；

关系成立，结果为全 1；关系不成立，结果为 0

例： DA1 DB 3 LT 8

DA2 DB 10 NE 0AH

MOV AL, 10 EQ 0AH

MOV BX, DA2 GE DA1


4、数值返回运算

SEG、OFFSET、TYPE、SIZE、LENGTH

格式: < 数值返回运算符 > < 地址表达式 >

{

 变量名
标号

存储器操作数
 

- (1) SEG 运算 **返回段基值**
- 符 2) OFFSET 运算 **返回偏移地址**
- 符 3) TYPE 运算符 **返回类型属性对应的数值**

	类型属性	运算结果
变量	BYTE	1
	WORD	2
	DWORD	4
标号	NEAR	-1
	FAR	-2

(4) LENGTH 运算符 **只用于变量**

若变量用 DUP 定义，返回 **外层 DUP 的重复次数**；
若变量没用 DUP 定义，则 **返回结果总是 1**。

(5) SIZE 运算 **只用于变量**
符是 **TYPE** 和 **LENGTH** 两个运算结果的 **乘积**。

例：

```
ORG 20H
VAR1 DB 10, 15, 20
VAR2 DW 0FFFFH, 100H
VAR3 DW 10H DUP (1, 2, DUP
(4) )
MOV AX, SEG VAR1
MOV SI, OFFSET VAR2
MOV BL, TYPE VAR2
MOV CL, LENGTH
MOV CH, SIZE VAR3
```

5、属性修改运算符 PTR

为已分配的存储单元临时设定类型属性

格式： 类型 PTR <地址表达式>

例：

DA_BYTE DB 20H DUP (0)

DA_WORD DW 30H DUP (0)

⋮

MOV WORD PTR DA_BYTE [10] , A

MOV BYTE PTR DA_WORD [DI] , B

INC BYTE PTR [SI]

SUB WORD PTR [BX] , 30H


X

L

6、运算符的优先级

规则：先高优先级，后低优先级；

若有多个同优先级的运算符，则从左到右；
圆括号可改变运算顺序。

- 
- 最高 (1) LENGTH, SIZE
(2) PTR, OFFSET, SEG, TYPE
(3) *, /, MOD, SHR, SHL
(4) +, -
(5) EQ, NE, LT, LE, GT, GE
(6) NOT
(7) AND
最低 (8) OR, XOR

第4节 80x86 宏汇编语言伪指令

5.4.1 符号定义语句

1、等值语句 EQU

常数 / 数值表达式

地址表达式

变量 / 标号 / 指令助记符

格式: 符号 EQU <表达式>

功能: 将表达式的值赋给符号

例: COUNT EQU 5
 ADR1 EQU DS:[BP+14]
 L1 EQU SUBSTART

EQU 伪指令不分配存储单元;

在同一源程序中, 同一符号不能用 EQU 伪指令重新定义;



2、等号语句 =

格式: 符号 = 表达式

功能与 EQU 语句相同

区别在于等号语句可重新定义符号。

5.4.2 处理器选择伪指令

通常放在源程序的开头位置。

用于确定选择使用哪种指令系统，缺省时为 8086/8088 指令系统和 8087 协处理器指令集。

.8086

.286

.286P

.

386

386P

.486

.486P

.586

.586P

其中，“P”表示保护模式



5.4.3 段结构伪指令

1、段定义伪指令 **SEGMENT/ENDS**

格式:

段名 **SEGMENT** [定位类型] [组合类型] [使用类型] [‘ 类
别名’]
⋮

段名 **ENDS**
功能: 指定逻辑段的名称和范围、段的起始边界、
段与段之间的连接关系等。

- (1) 段 **必选**, 开始与结尾的段名一致
各 (2) 定位类型 **可选**, 指定装入时的起始边界要求

4 种: **PAGE** (页)、**PARA** (节)、**WORD** (字)、**BYTE** (字节) **默认**

(3) 组合类型**可选**，指定段与段之间的连接方式
6种：NONE（隐含）、PUBLIC、COMMON、

STACK、MEMORY、AT
(4) 使用类型**可选**，指定386以上CPU的段模式
2种：USE16 段基值和偏移地址都是16位

USE32 段基值16位，偏移地址32位
(5) 类别名**可选**，**单引号扩起来**

2、段寻址伪指令 **ASSUME**

格式：ASSUME sr1 : seg1 , sr2 : seg2,
段名
段寄存器名

功能：建立段名与段寄存器之间的联系
17 / 9 / 4

例:

DS_DATA SEGMENT

VAR1 DB 12H

DS_DATA ENDS

ES_DATA SEGMENT

VAR2 DB 34H

ES_DATA ENDS

CODE SEGMENT

VAR3 DB 56H

ASSUME CS:CODE, DS:DS_DATA, ES:ES_DATA

START :

⋮

INC VAR1

INC VAR2

INC VAR3

⋮

CODE ENDS

END START

3、段寄存器的装载

(1) DS 和 ES 的 **用数据传送指令**

例: MOV AX, DATA_DS ; 设置 DS
MOV DS, AX
MOV AX, DATA_ES ; 设置 ES
MOV ES, AX

(2) SS 的装

载 **自动装载:** 定义堆栈段时, **组合类型置为 STACK**

```
STACK1 SEGMENT PARA STACK  
        DW 40H DUP ( 0 )    STACK1  
ENDS
```

用执行指令的方法装载: 与 DS/ES 的装载方法类似

(3) CS 的装
载 使用结束伪指令 END ， 格式如下：

END < 地址表达式 >

执行转移类指令时， CPU 自动修改 CS 和 IP 。

5.4.4 段组伪指令 GROUP

把程序中不同段名的段组成一个段组， 格式如下：

< 段组名 > GROUP < 段名 1 ， 段名 2 ， …… >

段组名由程序设计人员设定；

可直接引用段名， 也可用 SEG < 变量名 > / < 标号 > ；

段组内各段间的程序转移可按段内转移处理；

段组内各段的数据存取操作可用同一个段寄存器。

5.4.5 内存模式和简化段定义伪指令

1、内存模式伪指令

确定用户程序中代码和数据在内存中的存放方式。

格式: .MODEL < 内存模式

6 种模式: Tiny、Small、Medium

、

Compact、Large、Huge

2、简化段定义伪指令

.CODE [段名] ; 代码段

.DATA ; 数据段, 已初始化数据

.DATA ? ; 数据段, 未初始化数据

.CONST ; 常数段

.FARDATA [段名] ; 远数据段, 已初始化数据
!.FARDATA? [段名] ; 远数据段, 未初始化数据
:STACK[长度] ; 堆栈段

例:

```
.MODEL SMALL  
.STACK 20H; 定义堆栈段  
.DATA  
:  
:  
.CODE ; 定义代码段  
BEGIN : .....  
:  
MOV AH , 4CH  
INT 21H  
END BEGIN
```

3、预定义符号

类似于 EQU 伪指令定义的等价符号，例如

@Model、@Code、@Data、@Far
data、@Stack、@Codesize、@Datasi

5.4.6 定位和对准伪指令

1、位置计数器

(~~\$~~)记录正在汇编的数据或指令的目标代码在当前段内的偏移地址。

~~\$~~: 表示位置计数器的当前值。

2、定位伪指令 (ORG) 设置位置计数器的值

格式: ORG <表达式>

3、对准伪指令 (EVEN)

功能：将位置计数器的值调整为偶数。

格式： EVEN

5.4.7 过程定义伪指令 PROC/END

P

格式： 过程名 PROC NEAR/FAR
 ⋮
 RET } 指令序列
 ⋮
 过程名 ENDP

过程定义在逻辑段内； 过程名是必须的；

3 个属性：段、偏移地址、类型属性；

至少有一条返回指令 RET。



5.4.8 包含伪指令 INCLUDE

功能： 将指定文件插入到正在汇编的源程序中。

格式： INCLUDE < 文件名

>

5.4.9 标题伪指令 TITLE

功能： 为程序指定标题。

格式： TITLE < 文本 >

在源程序开始处使用；

不超过 80 个字符；

指定的标题在列表文件中每一页的第一行显示。

3 步：宏定义、宏调用和宏展开

```
宏名    MACRO [形参 1 , 形参 2 , .....  
          } 宏体  
ENDM
```

XCHAGE MACRO MEM1 , MEM2 , RE

XCHG REG , MEM2

MOV MEM1 , REG

ENDM

(2) 宏调用

宏名 [实参1, 实参2, ……]

例, 可对前面定义的宏调用如下

XCHAGE [SI], [DI], AX

(3) 宏展开

宏汇编程序扫描宏指令语句 (宏调用) 时, 用宏体的目标代码插入宏调用处; 对带参数的宏, 用实参代替形参, 并对宏体中出现参数的地方作适当修改。

2、宏操作符

(1) 连接操作符 &

功能: 在宏定义的宏体内连接形参。

例： SHIFT_VAR MACRO R_M, DRECT, COUNT
MOV CL, COUNT
S&DRECT R_M, CL
ENDM

宏调用： SHIFT_VAR AX, HL, 2

(2) 表达式操作符 %

格式： % 表达式

功能： 告诉宏汇编程序获取表达式的值，
而不是表达式本身。

(3) 文本操作符 <>

功能： 将包含分隔符的实参扩起来，
作为一个单一的实参。

例： XCHAGE <BYTE PTR [SI]>, [DI], AL

(4) 字符操作符!

格式: ! 字符

“!”后的字符不作操作符使用,而是字符本身。

3、LOCAL 为指令

格式: LOCAL <符号表

只能用于宏定义中,宏体第一条语句;

汇编时,符号展开为 ??XXXX 的形式。

4、宏库

多个宏定义以文件形式组织成宏库。

使用时用 INCLUDE 伪指令。



第6节 汇编语言程序设计基本技

5.6.1 程序设计步骤

分析问题，建立数学模型；

确定算法；

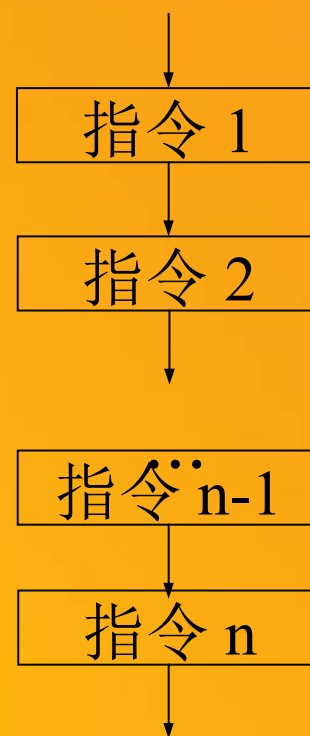
编制程序流程图；

编制程序；

调试程序。

5.6.2 顺序程序设计

按编写指令的顺序执行，且每条指令只执行一



顺序程序结构图

图

例：试编程序，计算下列公式的值，并将结果存放在 FUN 存储单元中。

$$F = \frac{10 \times (X + Y) - 3 \times (Z - 1)}{2}$$

其中 X ， Y ， Z 的值分别存放在 **VARX**、**VARY**、**VARZ** 三个字存储单元中，且计算过程的中间值和最后结果仍在 16 位二进制数的范围内。编制源程序如下：

```
TITLE  EXAMPLE PROGRAM
DATA  SEGMENT  ; 设置数据段
    VARX DW 123H      ; 变量 X
    VARY DW 456H      ; 变量 Y
    VARZ DW 789H      ; 变量 Z
    FUN  DW ? ; 结果单元
```

```
DATA  ENDS
```

```
STACK1 SEGMENT PARA STACK ; 设置堆栈段
    DW 20H DUP ( 0 )
```

```
STACK1 ENDS
```



例：用查表方法将一位十六进制数（0 ~ 9，A ~ F）转换成它对应的 ASCII 码。

首先在数据段建立一个表 TABLE，按照十六进制数从小到大（即从 0 ~ 9 到 A ~ F）的顺序，在表中存入它们对应的 ASCII 码值（十六进制数用大写英文字母 A ~ F）。为查出某个数的 ASCII 码，需计算它在内存中的地址。

用简化段定义伪指令，编制源序如下：

```
.MODEL SMALL    ;设置内存模式
```

```
.DATA           ;设置数据段
```

```
TABLE DB 30H,31H,32H,33H,34H,35H,36H,37H
```

```
        DB 38H,39H,41H,42H,43H,44H,45H,46H
```

```
HEX     DB 4
```

```
ASCII   DB ?
```

. STACK 100H ; 设置堆栈段
. CODE ; 设置代码段

START: MOV AX, @DATA
MOV DS, AX
LEA BX, TABLE ; 取表首址
XOR AH, AH ; AH 清零
MOV AL, HEX ; 取一位十六进制

数

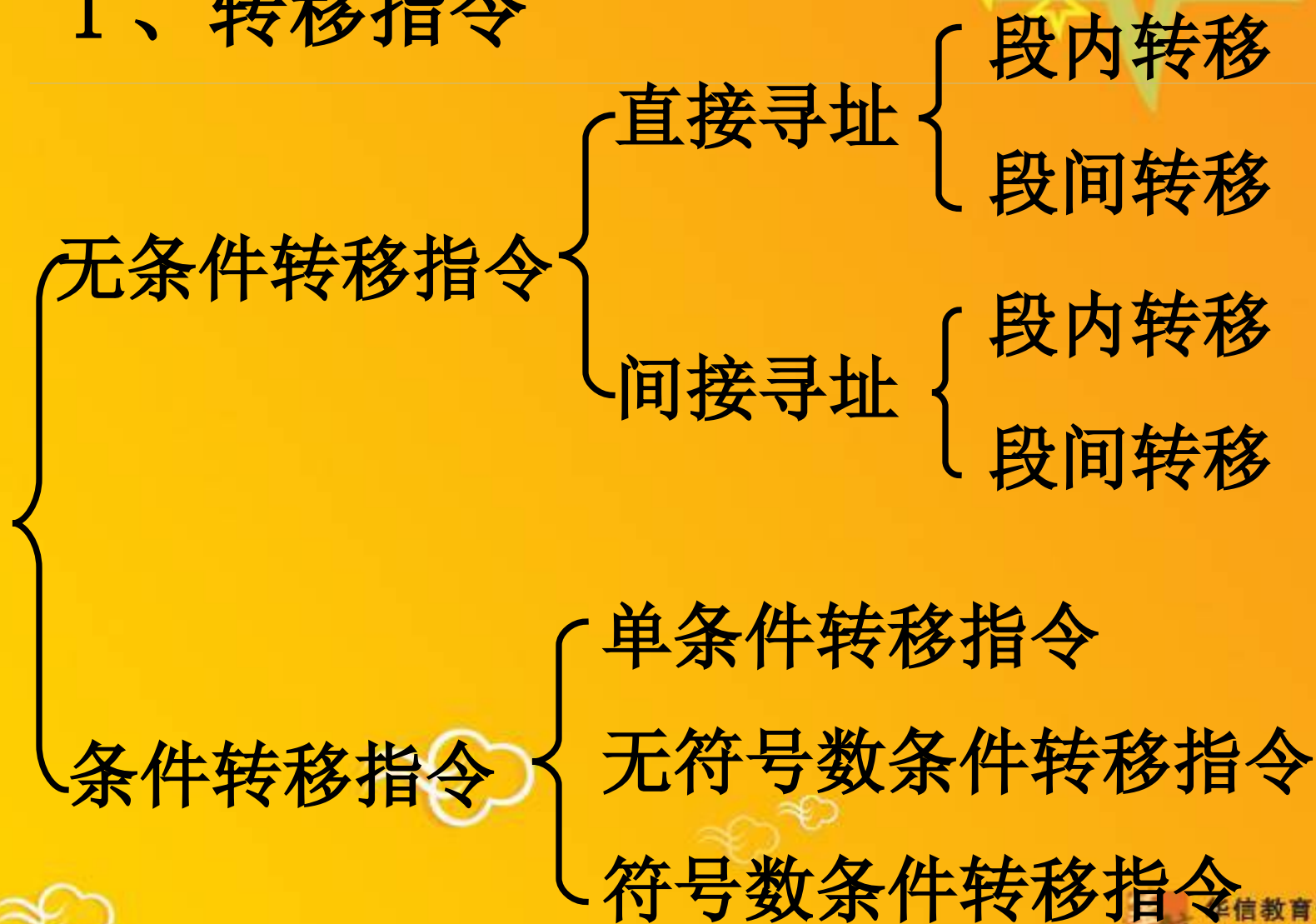
ADD BX, AX ; 确定查表位置
MOV AL, [BX] ; 查表
MOV ASCI, AL ; 存结果
MOV AH, 4CH ; 终止程序, 返回 D

OS

INT 21H
END START

5.6.3 分支程序设计

1、转移指令



(1) 无条件转移指令

格式: **JMP** **目标地址**

JMP 指令的下一指令与目标地址所指的指令之间的字节距离

目标地址有两种表达方式

a. 直接寻址: **目标地址通常是标号**

例如: **JMP** **NEXT** ;**NEXT 为标号**

段内转移 相对转移, 指令给出位移量 DISP ;

执行操作: $IP \leftarrow (IP) + DISP$

段间转移 指令中给出目标地址的段基值和偏移地址;

执行操作: $IP \leftarrow$ 偏移地址

$CS \leftarrow$ 段基值



b. 间接寻址:

目标地址通常由寄存器或存储单元提供

段内转移: 寄存器或存储单元提供偏移地址

执行操作: $IP \leftarrow \text{偏移地址}$

段间转移: 由一个双字单元提供目标地址的段基值和偏移地址;

执行操作: $IP \leftarrow \text{偏移地址}$
 $CS \leftarrow \text{段基值}$

(2) 条件转移指令 \rightarrow **xx 为转移条件**

格式: $J_{xx} \quad \text{目标地址}$

只能在段内转移, 而且是相对转移。

a. 单条件转移指令

指令	转移条件	
JC	CF=1	有进位 / 借位转移
JNC	CF=0	无进位 / 借位转移
JE/JZ	ZF=1	相等 / 等于 0 转移
JNE/JNZ	ZF=0	不相等 / 不等于 0 转移
JS	SF=1	是负数转移
JNS	SF=0	是正数转移
JO	OF=1	有溢出转移
JNO	OF=0	无溢出转移
JP/JPE	PF=1	有偶数个“1”转移
JNP/JPO	PF=0	有奇数个“1”转移

b. 无符号数条件转移指令

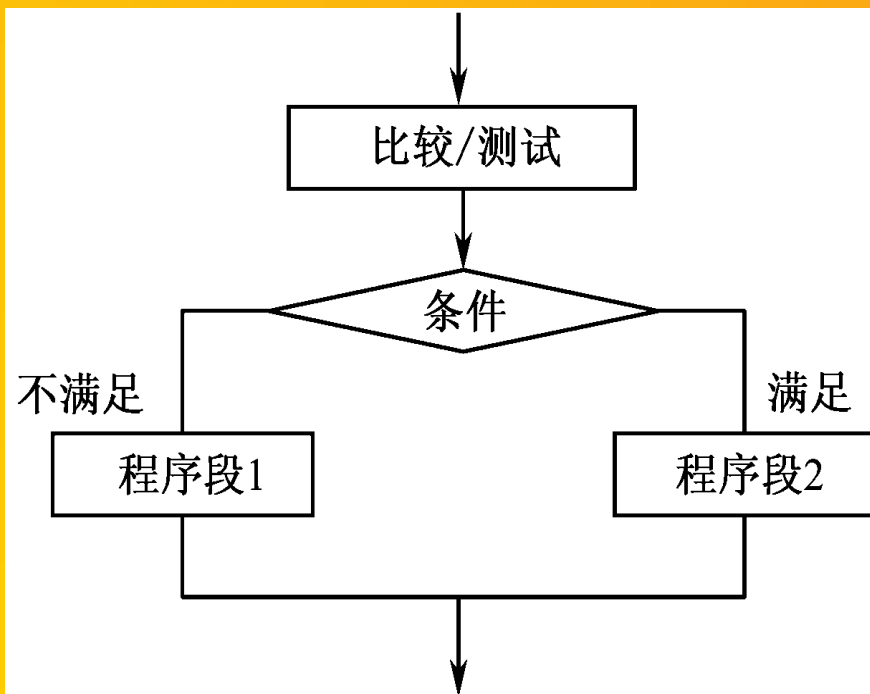
指 令	转 移 条 件
JA/JNBE	CF=0 AND ZF=0 A>B 转移
JAE/JNB	CF=0 OR ZF=1 A≥B 转移
JB/JNAE	CF=1 AND ZF=0 A < B 转移
JBE/JNA	CF=1 OR ZF=1 A≤B 转移

c. 符号数条件转移指令

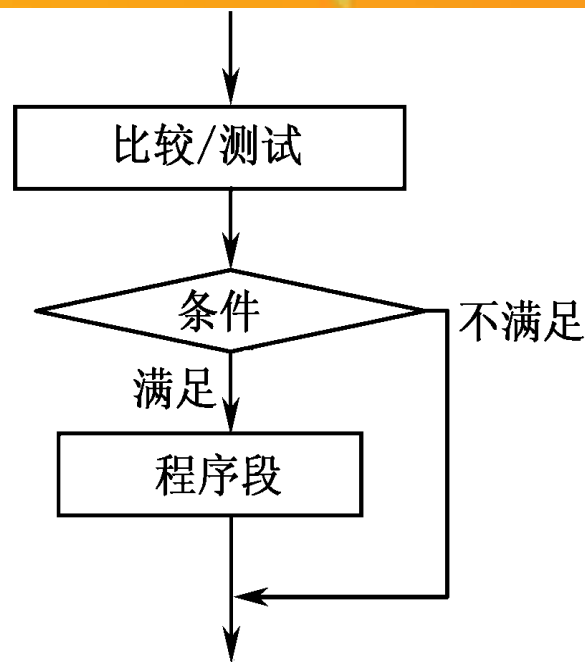
指 令	转 移 条 件
JG/JNLE	SF=OF AND ZF=0 A>B 转移
JGE/JNL	SF=OF OR ZF=1 A≥B 转移
JL/JNGE	SF≠OF AND ZF=0 A < B 转移
JLE/JNG	SF≠OF OR ZF=1 A≤B 转移

2、分支程序设计

(1) 比较 / 测试——分支结构



(a) IF-THEN-ELSE结构

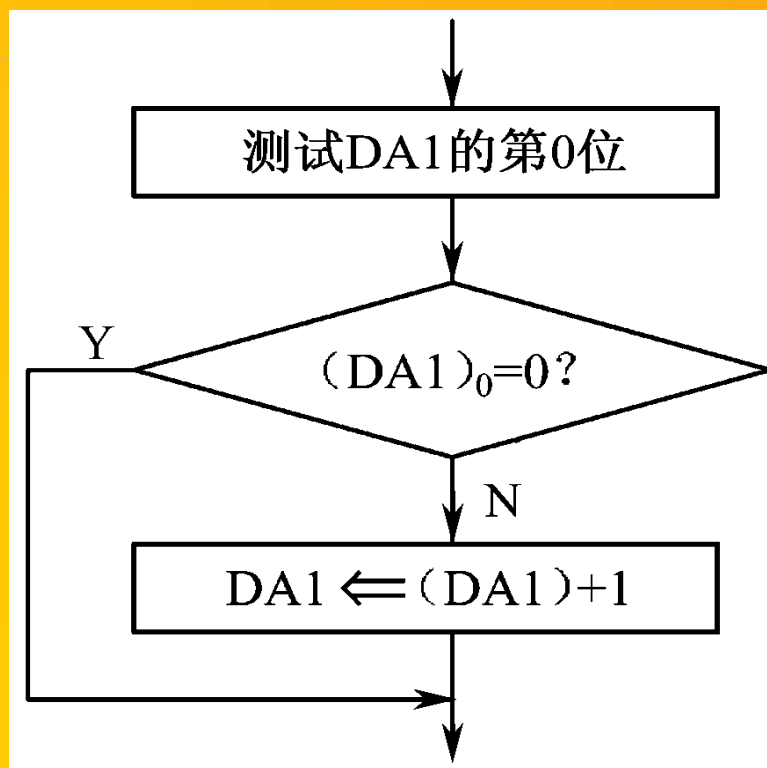


(b) IF-THEN结构

比较 / 测试——分支结构程序流程

例：编程序段，把 DA1 字节数据变为偶数。

分析：若二进制数最低位为 0，则为偶数。



程序段流程

程序段如下：

TEST DA1, 01H

JE NEXT

INC DA1

NEXT:

例：设数据段中 NUM1, NUM2 两字节单元中有无符号整数，编程完成下面的操作：

- 如两个数均是偶数，两个数加 1 后分别送入
- 如两个数均是奇数，两个数分别直接送入 DA1、DA2 字节单元中；
- 如一个是奇数，一个是偶数，则奇数直接送 DA1 字节单元，偶数直接送 DA2 字节单元

分析：依次测试 NUM1 和 NUM2 的奇偶性，有 4 种情况

NUM1 NUM2

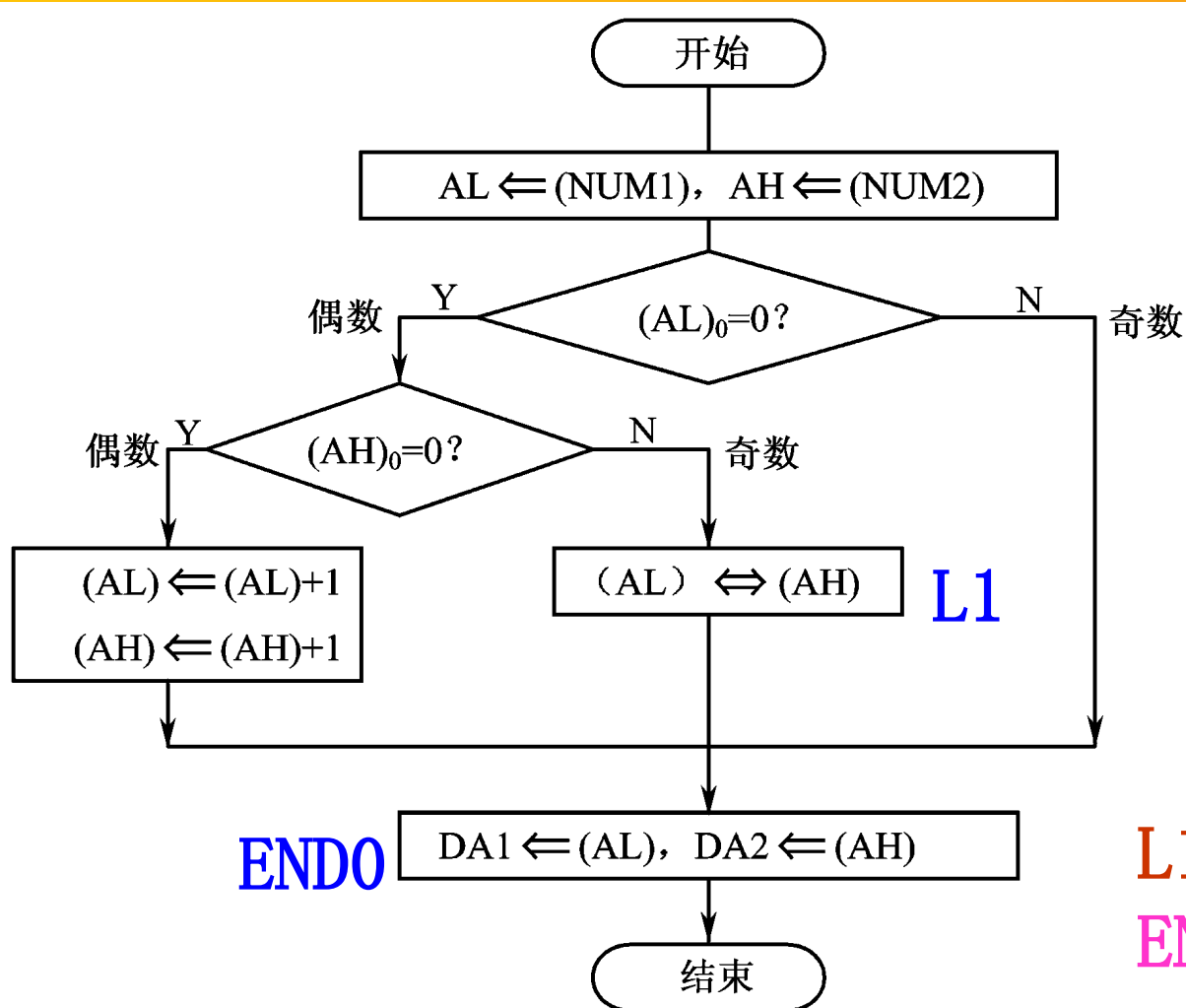
奇 奇 $DA1 \leftarrow NUM1, DA2 \leftarrow NUM2$

偶 偶 $DA1 \leftarrow NUM1, DA2 \leftarrow NUM2$

偶 奇 $DA1 \leftarrow NUM2, DA2 \leftarrow NUM1$

偶 偶 $DA1 \leftarrow NUM1+1, DA2 \leftarrow NUM2+1$

根据分析，画出流程图如下：



程序段如下：

```
MOV    AL, NUM1
MOV    AH, NUM2
TEST   AL, 01H
JNE    END0
TEST   AH, 01H
JNE    L1
INC    AL
INC    AH
JMP    END0
L1:    XCHG AL, AH
END0:  MOV DA1, AL
        MOV DA2, AH
```

L1: XCHG AL, AH

END0: MOV DA1, AL

MOV DA2, AH

(2) 用跳转表形成多路分支结构

假设某程序根据不同情况在 5 个计算公式中选择 1 个。可编制 5 个程序段，分别完成 1 个公式的运算。在程序中构造跳转表，有两种情况

例：由分支的入口地址构成跳转表。

跳转表在**数据段**，表中每一项都是一个**分支的偏移地址**

为转移到第 N 个公式，需先找到该分支的**入口地址**。
入口地址在跳转表中的偏移量为 $2*(N-1)$ 。

表首址→	
	SUB1—L
	SUB1—H
	SUB2—L
	SUB2—H
	SUB3—L
	SUB3—H
	SUB4—L
	SUB4—H
	SUB5—L
	SUB5—H

(a) 由入口地址组成

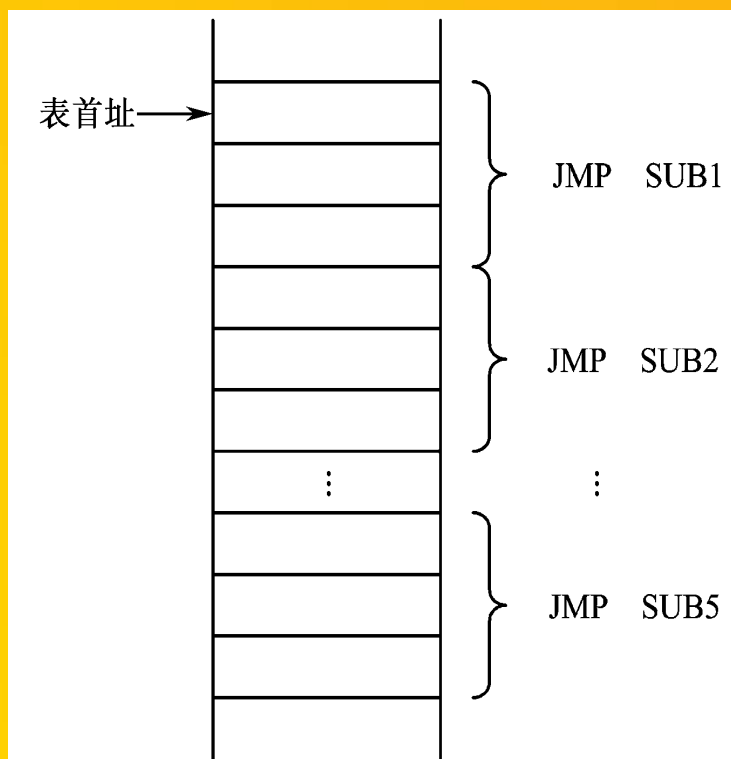
数据段如下：

```
DATA SEGMENT
JUMP_TABLE DW
SUB1 , SUB2 , SUB3 , SUB4 , SUB5
PARAM      DB 3
DATA ENDS
```

实现多路分支的程序段如下：

```
XOR    AX , AX
MOV     AL , PARAM    ; 取参数
DEC     AL             ; 计算 2*(PARAM-1)
SHL     AL , 1
MOV     BX , OFFSET JUMP_TABLE
ADD     BX , AX
MOV     AX , [BX]      ; 取转移的入口地址
JMP     AX             ; 间接转移到分支入口
```

例：由转移指令构成跳转表的多路分支程序设计。
 跳转表在代码段，表中每一项都是 JMP 指令代码
 为转移到第 N 个公式，需先转移到跳转表中对应的
 的 JMP 指令（在表中的偏移量为 $3*(N-1)$ ），再通
 过这里的 JMP 指令转移到对应的计算公式。



(b) 由无条件转移指令组成

实现多路分支的程序段如下：

```

XOR    BX, BX
MOV     BL, PARAM      ; 取参数
DEC     BL              ; 参数减 1
MOV     AL, BL          ; 再乘 3
SHL     BL, 1
ADD     BL, AL
ADD     BX, OFFSET JUMP_TABLE
JMP     BX              ; 转至跳转表
JUMP_TABLE: JMP SUB1    ; 转至分支
    
```

.....

5.6.4 循环程序设计

1、循环控制指令 属于程序转移类指令

(1) LOOP 指令

格式: LOOP 目标地址

功能: 循环计数 (CX 减 1) 后, 判断循环是否结束:
若 (CX)=0, 则继续循环, 否则顺序执行。

例: 编制程序, 产生 n 个数据的斐波纳契数列。

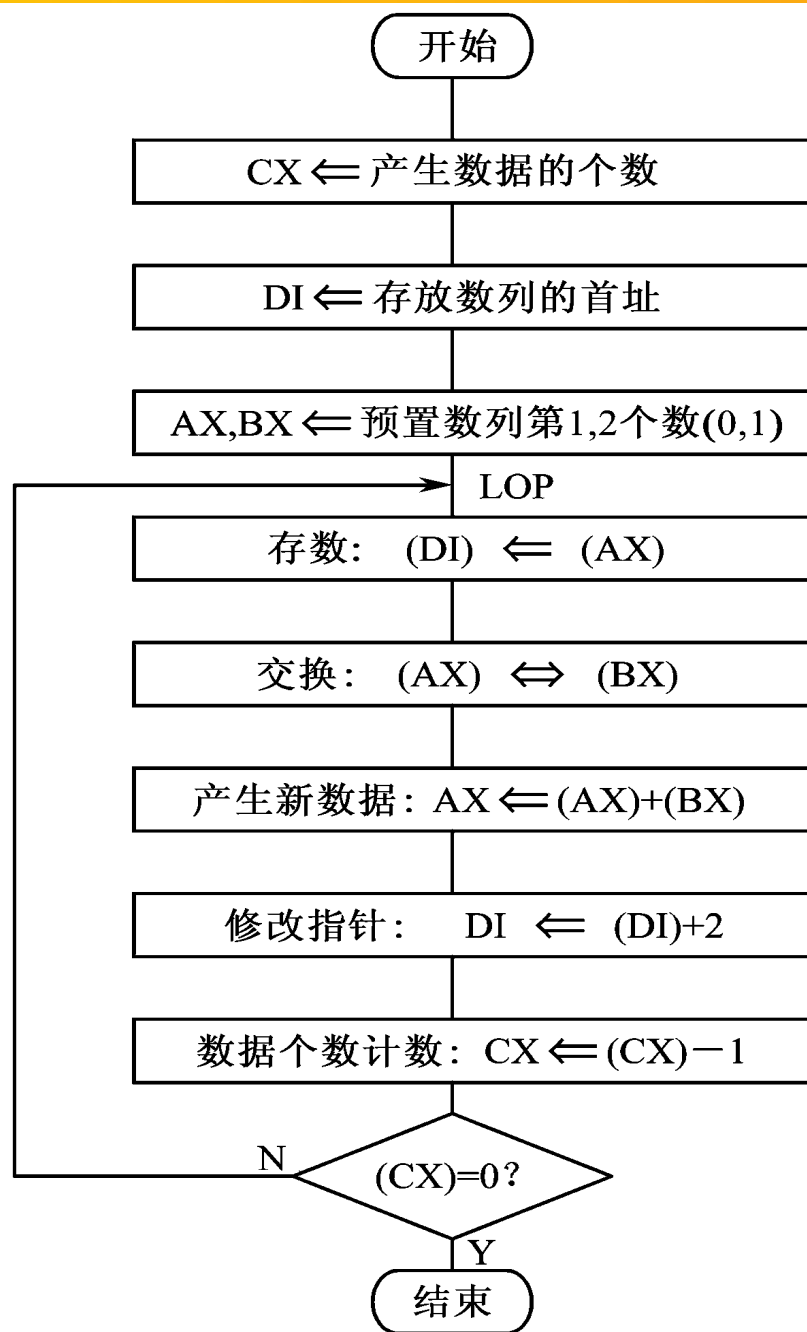
斐波纳契数列中, $a_1=0$, $a_2=1$,

从第 3 个数开始, $a_n = a_{n-1} + a_{n-2}$

数据段中数据定义如下:

FIBONA DW 100 DUP(0) ; 存放数列

NUM DB 20H ; 数据个数



产生数列的程序段如下:

⋮

```
XOR    CX,CX
MOV     CL , NUM
LEA     DI , FIBONA
MOV     AX , 0
MOV     BX , 1
LOP:    MOV     [DI] , AX
        XCHG    AX , BX
        ADD     AX , BX
        ADD     DI , TYPE FIBONA
        LOOP    LOP
```

⋮

(2) LOOPE/LOOPZ 指令

格式: L00PE 目标地址

LOOPZ 目标地址

功能: 循环计数 (CX 减 1) 后, 判断循环是否结束:

若 (CX)=0 且 ZF=1, 则继续循环, 否则
顺

例: 编程, 在字符串中查找第一个非空字符, 并将其在字符串中序号 (1~n) 送入 INDEX 单元。
程序执行。

若未找到非空字符, 则将全 1 送入 INDEX 单元。
分析: 逐个字符与空字符 (ASCII 码为 20H) 进行比较, 用 L00PE 循环指令。循环结束有两种情况: 计数为 0 或找到非空字符, 再进一步分析。

数据段中数据定义如下：

```
STRING DB      'CHECK STRING'  
COUNT EQU  $-STRING  
INDEX  DB      ?           ; 存结果
```

```
程序段：      MOV CX, COUNT  
              MOV BX, -1  
NEXT: INC  BX  
      CMP STRING[BX], 20H  
      LOOPE NEXT  
      JNE  OK  
      MOV BL, 0FEH      ; 未找到  
OK:   INC  BX  
      MOV INDEX, BL     ; 存结果
```

(3) LOOPNE/LOOPNZ 指令

格式: **LOOPNE** 目标地址

LOOPNZ 目标地址

功能: 循环计数 (CX 减 1) 后, 判断循环是否结束:

若 $(CX)=0$ 且 $ZF=0$, 则继续循环, 否则
顺

例: 设数据段中有一个以 ARRAY 为首地址的字节数组。
现要求编制一程序, 对数组中每一数据除以 0FH, 用它的
余数构造一个新数组 YUSHU。当 ARRAY 数组中数据
处理完毕, 或某次相除时余数为 0, 便停止构造新数组。
程序最后将新数组的数据个数存放在 LEN 单元中。

分析: 对数组元素依次作除法, 判断余数是否为 0。用循
环指令 LOOPNZ 控制循环, 结束循环后, 再进一步分析。

数据定义如下:

```
ARRAY DB 12H, .....  
NUM EQU $-ARRAY  
YUSHU DB NUM DUP(0)  
LEN DB ?
```

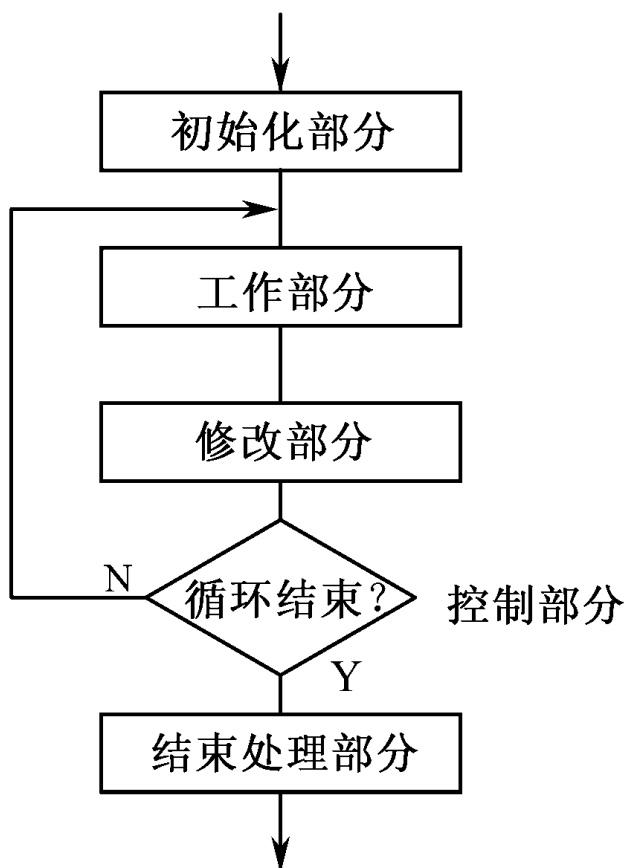
程序段:

```
MOV CX, NUM  
XOR BX, BX  
MOV DL, 0FH  
NO_ZERO: MOV AL, ARRAY[BX]  
XOR AH, AH  
DIV DL ; 除 0FH  
MOV YUSHU[BX], AH  
INC BX  
CMP AH, 0 ; 余数为 0 吗 ?  
LOOPNE NO_ZERO  
JNE END0 ; 有余数为 0 ?  
DEC BL ; 有余数为 0  
END0: MOV LEN, BL
```

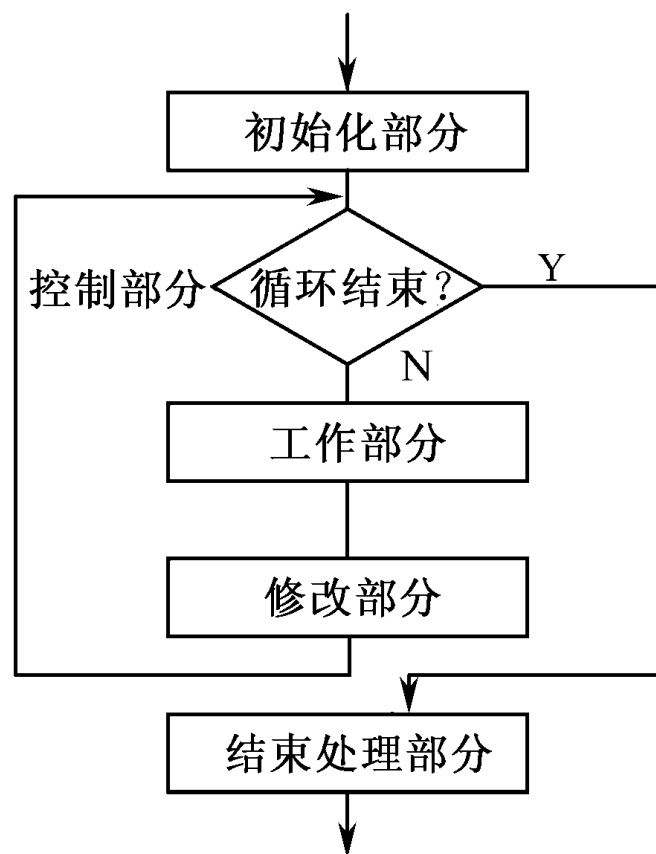
(4) JCXZ 指令 测试寄存器 CX 的内容

格式: JCXZ 目标地址

2、循环程序结构



(a) “先工作后判断” 结构形式



(b) “先判断后工作” 结构形式

3、循环控制方法

(1) 计数控制循环

例：编程，统计数组中相邻两数之间的符号变化的次数。

程序段：

计数器

MOV CX, COUNT-1

XOR BL, BL

EXCHANG: MOV AL, [SI]

XOR AL, [SI+1]

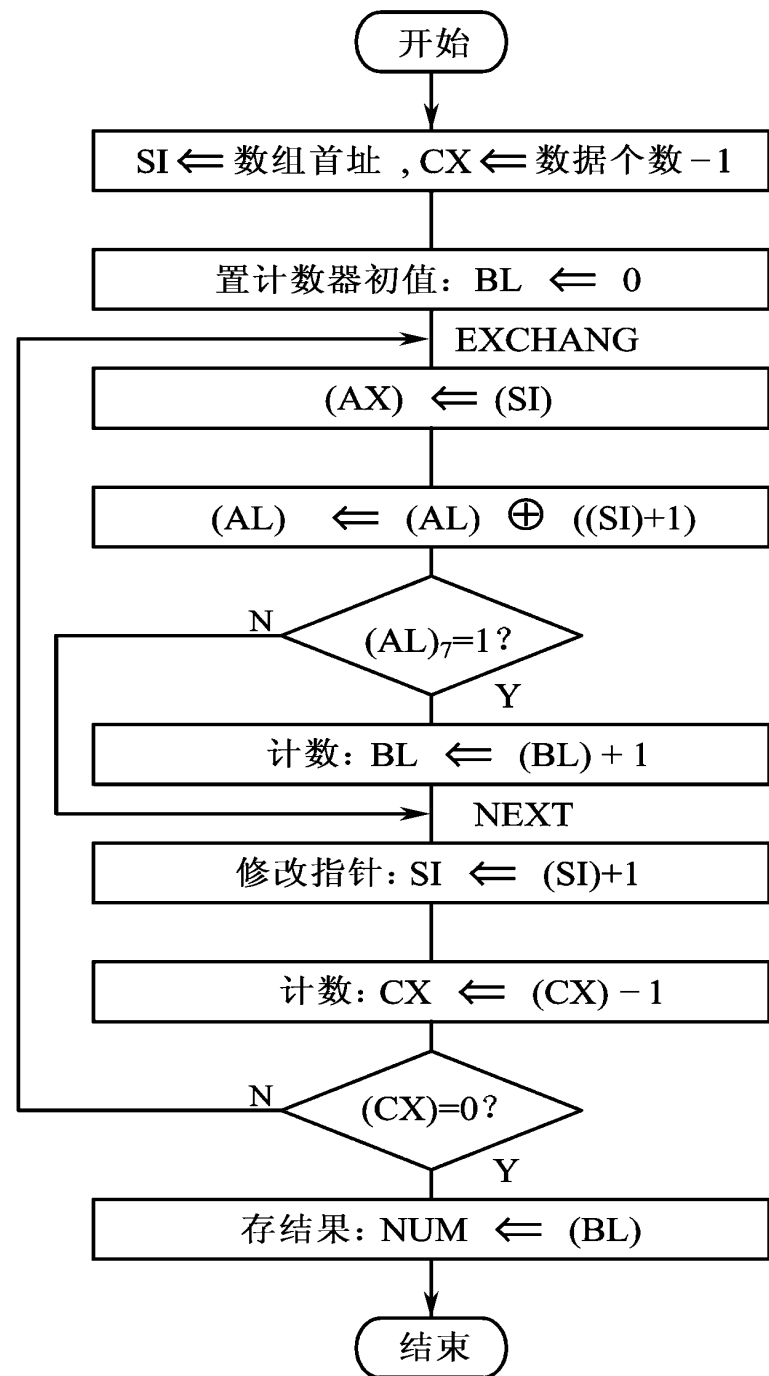
TEST AX, 80H

JE NEXT

INC BL

NEXT: INC SI

LOOP EXCHANG



(2) 条件控制循环

每循环一次，测试并判断循环终止条件是否成立。

例：编程，产生给定数以内的斐波纳契数列，并把数列的个数存入 LEN 单元中。

分析：循环次数事先未知，只能用条件控制循环。

循环终止条件：当新产生的数据大于给定数，则结束循环。

程序段：

```
LEA    DI, FIBOINA
XOR     CL, CL
MOV     AX, 0
MOV     BX, 1
LOP:    MOV     [DI], AX
        XCHG    AX, BX
        ADD     AX, BX
        ADD     DI, TYPE FIBONA
        INC     CL
        CMP     AX, NUM    ; 测试
        JA      END0      ; 大于，结束
        JMP     LOP        ; 小于，继续
END0:   MOV     LEN, CL
```

5.6.5 子程序设计

子程序用过程定义伪指令定义。

1、调用与返回

调用指令格式： CALL 子程序名 / 过程名

返回指令格式： RET

带参数的返回指令格式： RET n ;N 为偶数

(1) 段内调用与段内返回

段内直接调用：入口地址直接由子程序名提供

段内间接调用：入口地址由寄存器 / 存储单元提供

执行操作：保留段点的偏移地址（当前 IP 值），
 $IP \leftarrow$ 子程序入口地址的偏移地址。

段内返回： $IP \leftarrow ((SP)), SP \leftarrow (SP) + 2$ [, $SP \leftarrow (SP) + n$]

(2) 段间调用与段间返回

段间直接调用：入口地址直接由子程序名提供

段间间接调用：入口地址由双字单元提供

执行操作：保留段点的地址（当前 CS/IP 值），
CS \leftarrow 子程序入口地址段基值；
IP \leftarrow 子程序入口地址的偏移地址。

段间返回：IP \leftarrow ((SP)), SP \leftarrow (SP) + 2;

CS \leftarrow ((SP)), SP \leftarrow (SP) + 2;

[SP \leftarrow (SP) + n]

2、子程序设计方法

(1) 适当地划分并确定子程序功能

(2) 选择适当的参数传递方法

(3) 信息的保存

(4) 编写子程序的文字说明

3、子程序设计举例

(1) 使用寄存器传递参量

例：编程，对数据段中一组字数据用减奇数法求平方根，结果（平方根）依次存入 PFG 的字节数组中。

分析： 把求平方根的运算作为一个子程序；

入口参量 (AX)：被开方数；**出口参量 (CL)：**平方根。

减奇数法开平方的算法：被开方数 S 逐个减去 1 开始的连续自然数的奇数 1, 3, 5, ……，直到相减结果等于 0，或不够减下一个奇数为止。**够减的次数**就是 S 的近似平方根。

主程序段:

```
LOP:MOV  AX, [SI]      ; 寄存器传递入口参数
      PUSH CX          ; 保存信息
      CALL SQR_PROC
      MOV  [DI], CL     ; 存结果
      POP  CX          ; 恢复信息
      .....
```

子程序段: SQR_PROC PROC

```
      XOR CL, CL
      MOV DX, 1
SQR:  SUB  AX, DX      ; 减奇数
      JB  EXIT        ; 够减?
      INC CL          ; 够减, 计数
      ADD DX, 2       ; 形成下一奇数
      JMP SQR         ; 继续循环
EXIT: RET
SQR_PROC ENDP
```


(2) 使用存储单元传递参量

对上一例，入口参量（被开方数 KFS）和出口参量（平方根 SQRT）用存储单元传递，程序修改如下

主程序段：

```
LOP:MOV AX, [SI]
      MOV KFS, AX
      CALL SQR_PROC
      MOV AL, SQRT
      MOV [DI], AL
      .....
```

子程序段：

```
SQR_PROC PROC
.....
      MOV AX, KFS
      MOV SQRT, 0
      MOV DX, 1
SQR:   SUB AX, DX
      JB  EXIT
      INC SQRT
      .....
```

SQR_PROC ENDP

(3) 使用地址表传递参量

参量较多时，先把参量所在的地址组成一个地址表，将地址表的首地址传递给子程序。

例：编程，将两个 8 位和 16 位的二进制数分别转换

分析：主程序提供待转换数据、数据位数和转换后存放 ASCII 码的首址等 3 个参数的地址，并组成组成一个地址表，传递地址表首地址给子程序。

。 设数据定义如下：

```
BIN8          DB    35H
BIN16         DW    0AB48H
NUM           DB    8, 16
ASCBUF        DB    20H DUP(0)
ADR_TAB       DW    3    DUP(0)
```

主程序段:

.....

MOV ADR_TAB, OFFSET BIN8

MOV ADR_TAB+2, OFFSET NUM

MOV ADR_TAB+4, OFFSET ASCBUF

MOV BX, OFFSET ADR_TAB ; 通过 BX 传递地址表首址

CALL ; 子程序调用

.....

子程序段:

MOV DI, [BX] ; 取二进制数地址

MOV DH, [DI] ; 取二进制数

.....

MOV DI, [BX+2] ; 转换位数的地址

MOV CL, [DI] ; 转换位数

.....

MOV DI, [BX+4] ; ASCII 码首地址

.....

(4) 使用堆栈传递参量

将前一例改为用堆栈传递参量。

主程序段:

.....

MOV AH, BIN8

XOR AL, AL

PUSH AX

LEA AX, ASCBUF

PUSH AX

MOV AX, 8

PUSH AX

CALL BINASC

.....

子程序段:

.....

PUSH BP

MOV BP, SP

MOV DX, [BP+14]

MOV DI, [BP+12]

MOV CX, [BP+10]

.....

5.6.6 系统功能子程序调用

两组功能子程序 $\begin{cases} \text{DOS} \\ \text{BIOS} \end{cases}$

通过软中断指令 **INT** 实现功能调用。

软中断指令格式如下：

INT n **n** 为中断类型码，值为 00~0FFH

功能调用步骤如下：

—— 送入口参数给指定寄存器

—— **AH** ≤ 功能号

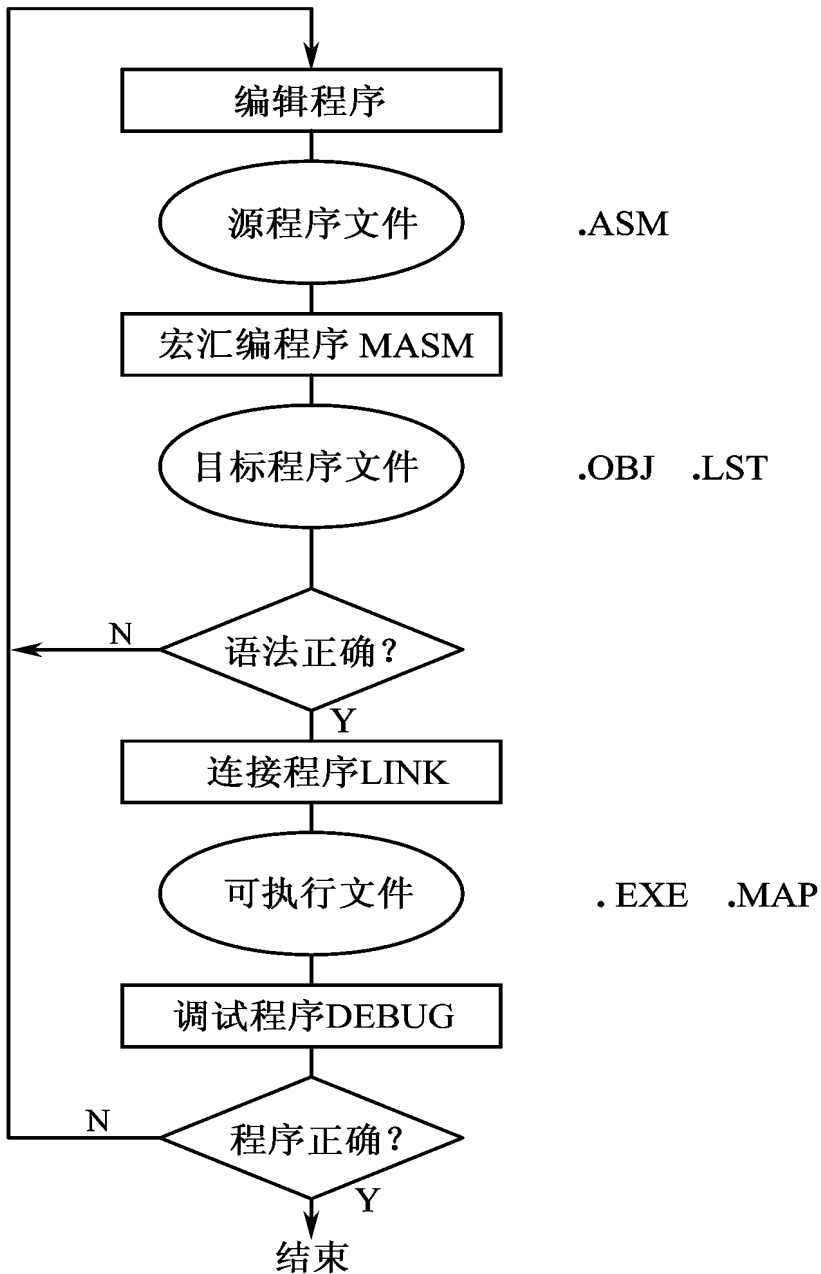
—— **INT n**

5.6.7 汇编语言程序的开发

1、编辑——建立源程序
编辑程序：**EDIT** 等；

按**逻辑段**来组织源程序

以 **END** 语句结束源程序
源文件扩展名为 **.asm**，
且不能省略。



2、汇编——生成目标程序

汇编程序: **MASM**

格式: **MASM *.AS**

M

主要功能

检查语法错误
实现宏替换
生成目标程序

生成文件

目标文件 (**.OBJ**)
列表文件 (**.LST**)
交叉引用文件 (**.CRF**)

3、连接——生成可执行程序

连接程序: **LINK**

格式: **LINK *.OBJ**

若需连接多个 OBJ 文件, 则用 “+”

连接。
生成的主要文件 { 可执行文件 (.EXE)
内存映像文件 (.MA

4、调试与运行^P 调试程序: **DEBUG**

(1) DEBUG 的进入与退出

D:> DEBUG ↵ 进入 DEBUG, 出现提示符 -

装载文件: - **N TEST.EXE ↵**

- **L ↵**

进入 DEBUG 时，同时装载文件：

D:> DEBUG TEST.EXE↵

退出 DEBUG 并返回操作系统：

格式： - Q↵

(2) 显示命令

D 命令——显示内存单元内容

格式： D [地址] 或 D

[范围]
例： - D DS:100↵

- D 100↵

- D DS:100 10F↵

R 命令——显示寄存器内容

格式: - R ↙

显示所有寄存器内容、标志位情况及下一条指令

```
AX=102A BX=0000 CX=0100 DX=0000 SP=0040 BP=0000 SI=0000 DI=0000
DS=1528 ES=1428 SS=1723 CS=1822 IP=0003  NV UP DI PL NZ NA PO
NC
1822:0003 8ED8    MOV DS,AX
```

U 命令——显示汇编源程序命令（反汇编）

格式: U [地址] 或 U

[范围]

例: - U CS:0 ↙

反汇编 32 个字节

- U CS:0 10 ↙

偏移量 0-10H 的单元

元

17 / 9 / 4 U CS:0 L 10 ↙

前 10H 个字节的代码

(3) 修改命令

E 命令——修改内存单元内容

格式: E [地址] [内容表]

例: - E DS:0 ✓

1200:100 20 31 ✓ ; 将 20H 修改为 31H

R 命令——修改寄存器内容

格式: R <寄存器名>

例: - R AX ✓

AX 0000 ; 显示 AX 原有内容
: 1111 ✓ ; 修改 AX 内容

A 命令——汇编命令

格式: A [地址]

例: - A CS:100✓

1723:0100 MOV AX , 29✓

1723:0103 MOV BX , 85✓

1723:0106 ADD AX , BX✓

1723:0108 ✓

(4) 程序运行命令

G 命令——连续运行方式

格式: - G [=地址][,地址][,地址].....✓

T 命令——跟踪运行方式

格式: - T [=地址][值]✓