

# 第六讲 程序设计的基本方法

## 一、基本要求

评估一个程序优劣的标准：

1. 程序结构清晰、层次分明；
2. 解题方法简单、算法描述简洁明了，程序易读、易调试；
3. 执行速度快；
4. 占用空间省。

汇编语言程序设计的一般步骤：

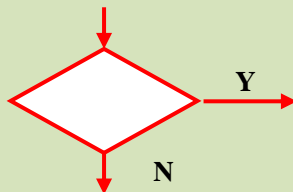
1. 分析问题，选择合适的解题方法；
2. 根据具体问题，确定输入/输出数据的格式；
3. 分配存贮区和寄存器，并给变量命名（注意变量类型字节、字或双字等）；
4. 绘制程序的流程图，即将解题方法和步骤用程序流程图表示；
5. 根据流程图编写程序；
6. 静态检查与动态调试相结合，达到所需要的结果。

流程图符号说明：

1. 起始、终止框





2. 判断框

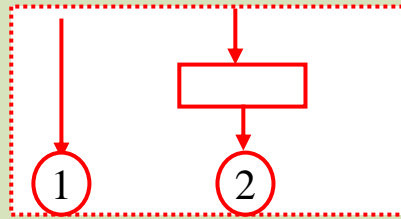


3. 处理说明框

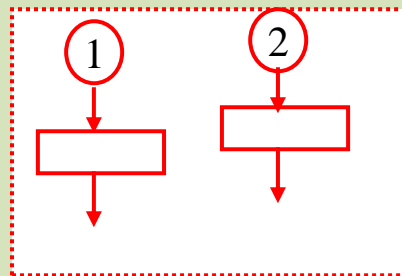


4. 子程序或过程调用框 

5. 流向框 



6. 连接框 



本讲主要讲解：

1、顺序程序

2、分支程序设计；

3、循环程序设计；

4、子程序设计。

} 各种转移指令格式、功能及使用条件

堆栈

## 二、转移指令

转移指令（修改 CS: IP/EIP 的值，改变程序执行次序）

{	有条件转移指令，又分为：	简单条件转移指令	单个标志位的值
		无符号数条件转移指令	无符号数有关标志位
		有符号数条件转移指令	有符号数有关标志位
		循环转移指令	计数 ECX, ZF
	无条件转移指令		

SF、OF（JS、JNS、JO 和 JNO）与**有符号数**密切相关，其他几个标志位主要与**无符号数**相关，**可以看出**：利用不同位的组合进行条件判断时，可以代表所关心的数是有符号数还是无符号数。

## 1、无符号数条件转移指令

当把操作数看作无符号数时，应根据**与无符号数特征有关标志位的组合值**确定转移方向。

① 高于转（即不低于且不等于转）**JA/JNBE**

**CMP OPD, OPS** ; (OPD) > (OPS) 转  
**JA P**

当 CF=0, 且 ZF=0 时转移。

② 高于或等于转（即不低于转）**JAE/JNB**

③ 低于转（即不高于等于转）**JB/JNAE**

④ 低于或等于转（即不高于转）**JBE/JNA**

分辨助记符：ABSolute

在汇编语言中，无符号数一般为：

- 1) 地址；
- 2) 字符 ASCII 码；
- 3) 循环计数器。

## 2、有符号数转移指令

由于有些指令是将操作数看作有符号数（补码表示），因此有符号的条件转移指

令将根据与符号数特征有关标志位的组合值 (SF,OF) 确定转移方向。

① 小于转 (或不大于等于转) **JL/JNGE**

例如: 设 (CX) = -1 = 0FFFFH; (B) = 7FFFH

**CMP CX, B**

**JL L1**

因为 (CX) < (B) 转 L1 执行

(与 JS 的异同)

② 小于等于转 (或不大于转) **JLE/JNG**

③ 大于转 (或不小于等于转) **JG/JNLE**

④ 大于等于转 (或不小于转) **JGE/JNL**

例: 清除数据段中 EA 为 8002H~7000H 号字中的内容。

(字节数为 8004H-7000H)

**MOV BX, 8002H**

**L: MOV WORD PTR [BX], 0**

**SUB BX, 2**

**CMP BX, 7000H**

**JAE L (或 JNB)** ; 此处用 **JGE** 不能循环, 因为清第一个字

后, EA 修改为 8000H, 为负, 与 7000H 相比不可能大于等于, 于是跳出循环。

### 3. 无条件转移指令 **JMP**

格式: 1. **JMP 标号** (直接跳转)

2. **JMP OPD** (间接跳转)

功能: a. 段内直接跳转: **JMP 标号**

标号的 EA  $\rightarrow$  IP/EIP

b. 段间直接跳转: **JMP** 标号 或: **JMP FAR PTR** 标号

标号所在段首址 $\rightarrow$ CS, 标号的 EA $\rightarrow$ IP/EIP

以上寻址方式均为立即寻址方式。

c. 段内间接跳转 **JMP OPD**

(OPD)  $\rightarrow$  IP/EIP 例如: **JMP AX** 表示 (AX)  $\rightarrow$  IP

d. 段间间接转移: **JMP OPD**

OPD 只能为存储器寻址方式。

如在 16 位段中: 双字类型变量/存储区的内容 (OPD)  $\rightarrow$  IP,

(OPD+2)  $\rightarrow$  CS

**JMP** 的作用: 1、完成不需要条件判断的转移;

2、转移的范围大, 灵活性大。

### 三、分支程序举例

例: (P110) 编制计算下面函数值的程序 (X, Y 的值域在 -128 ~ +127 之间)

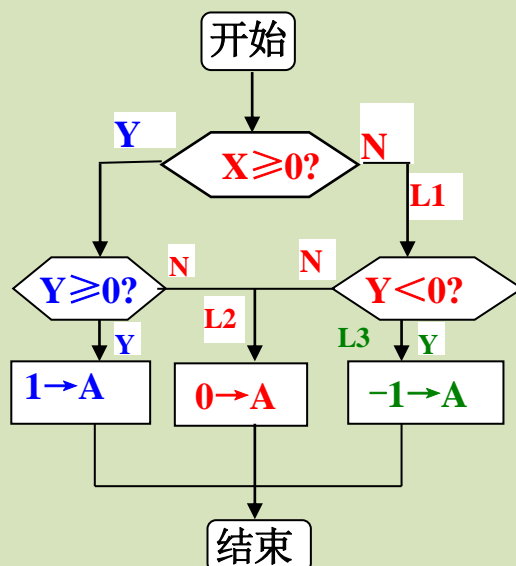
$$a = \begin{cases} 1 & x \geq 0, y \geq 0 \\ 0 & x, y \text{ 异号} \\ -1 & x < 0, y < 0 \end{cases}$$

DATA SEGMENT

**X** DB 34H

**Y** DB -7AH

**A** DB 0



DATA ENDS

⋮

CMP X, 0

JS L1 ; x < 0 转 L1

CMP Y, 0 ; x ≥ 0

JS L2 ; x ≥ 0、y < 0 转 L2

MOV A, 1 ; x ≥ 0、y ≥ 0 则 1 → A

JMP EXIT

L1: CMP Y, 0 ; x < 0

JL L3 ; x < 0、y < 0 转 L3

L2: MOV A, 0 ; x ≥ 0、y < 0 或 x < 0、y ≥ 0, 0 → A

JMP EXIT

L3: MOV A, -1 ; x < 0、y < 0 则 -1 → A

EXIT: MOV AH, 4CH

INT 21H

⋮

(程序编写的次序与流程图关系：从上到下，从左到右)。

在分支程序设计中应注意的地方：

1. 选择合适的转移指令
2. 要为每个分支准备好出口
3. 应将各分支的公共部分尽量提到分支前或分支后的公共程序段去执行，使程序简短、清晰。
4. 分支比较多时，**框图**对每个分支的判断先后次序应尽量与**问题提出的先后**次序一致。而**程序**对各分支的安排也需要与**框图**的安排次序一致，这样在编写程序时就不会漏掉某一支，而且写出的程序清晰，容易阅读和检查。

5. 在调试分支程序时，要**假定**各种可能的输入数据，**沿着每一支路**逐一检查，测试程序是否正确。只有所有分支都满足设计要求时，才能保证整个程序满足设计要求。

## 四、循环程序设计


循环程序的结构：

置循环初值 -》工作部分-》修改部分-》判断控制 -----》结束



置循环初值 -》判断控制 -》工作部分-》修改部分

结束



在汇编语言中，一般控制循环的方法有两种：

1. 计数控制：一般采取倒计数，正计数用得少。

```
        |  
        MOV    CX, 循环次数  
  
NEXT:   |  
        DEC    CX  
        JNZ    NEXT    }  LOOP NEXT
```

其中，“**LOOP 标号**”为计数循环指令。

**功能：**16 位段：(CX) -1→CX，(CX) ≠0 转至标号处执行，(CX) =0 则结束循环，继续往下执行。32 位段使用 ECX。

**说明：**LOOP 后的标号为短标号，跳转范围为-128~+127 个字节。该指令的执行不影响标志位。

2. 条件控制：当循环次数未知时，通过比较所要求的条件是否满足，未满足继续循环，否则结束循环。(次数已知是一种特例)

```
NEXT:      |  
            |  
ADD   AX,10  
  
CMP   AX,2000  
  
JB     NEXT
```

## (一) 单循环设计

## 循环次数已知的循环程序设计

例：阅读下列程序，指出该程序的功能。

```
DATA    SEGMENT

BUF1    DB    80

        DB    ?

        DB    80    DUP ( ? )

DATA    ENDS

STACK   SEGMENT    STACK

        DB    200 DUP (0)

STACK   ENDS

CODE    SEGMENT

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

INPUT:  MOV    AX, DATA
```



```

MOV DS, AX
LEA DX, BUF1
MOV AH, 10
INT 21H
} 输入一串字符

MOVZX CX, BUF1+1 ; 循环次数 → CX
LEA SI, BUF1+2 ; 输入串首址 → SI
MOV AL, 'A'
MOV BH, 0 ; 计数器清 0

NEXT: CMP [SI], AL
      JNE INC1
      INC BH ; 是 A 字符, 计数器加 1
INC1: INC SI
      LOOP NEXT ; (CX) = 0? 直接回车时的问题

MOV AX, 4C00H
INT 21H

CODE ENDS

END INPUT

```

该程序的功能为：从键盘输入一串字符，统计其中所含字符 A 的个数 → BH。

思考：1. 将“CMP [SI], AL”改写为：“CMP [SI], 'A'”是否可以？

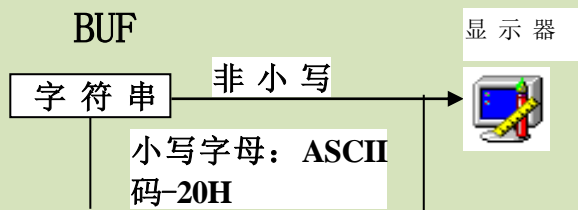
2. 如果要将输入的字符串再显示出来，还要做哪些工作？

### 循环次数未知的循环程序设计——用条件控制循环

例：在以 BUF 为首址的字节存贮区中存放着一个以 '\$' 作为结束标志的字符串，试编写程序在显示器上显示该字符串，并要求将小写字母以大写字母形式显示出来。

## 1. 分析解题方法

大写字母'A' ~'Z'的 ASCII 码为 41H~5AH, 小写字母'a' ~'z'的 ASCII 码为 61H~7AH, 即大小写对应字母 ASCII 码值相差 20H。



## 2. 确定输入输出数据的格式

输入用变量定义:

```
BUF DB 'ADD AX, BX', 0AH, 0DH
     DB 'SUB CX, 10', 0AH, 0DH
     DB 'MOV ax, 10', 0AH, 0DH
     DB 'MOV cx, 10', 0AH, 0DH, '$'
```

输出直接在当前行开始显示。

## 3. 寄存器使用分配

**BX:** 取 BUF 区数组元素指针;

**DL:** 中间寄存器。

**AH:** DOS 中断调用。

## 4. 画程序框图 (注意: 描述语言特点, 不能用汇编语句; 详简程度)

## 5. 编写源程序

DATA SEGMENT

```
BUF DB 'ADD AX, BX', 0AH, 0DH
     DB 'SUB CX, 10', 0AH, 0DH
     DB 'MOV ax, 10', 0AH, 0DH
```

DB 'MOV cx, 10', 0AH, 0DH, '\$'

DATA ENDS

STACK SEGMENT STACK

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT

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

START: MOV AX, DATA

MOV DS, AX

LEA BX, BUF

**NEXT:** MOV DL, [BX]

CMP DL, '\$' ; 是否为结束符

JE EXIT

CMP DL, 'a' ; 是否为小写字母

JB OUT0

CMP DL, 'z'

JA OUT0

SUB DL, 20H ; 是, ASCII 码减 20H

**OUT0:** MOV AH, 2 ; 输出一个字符

INT 21H

INC BX

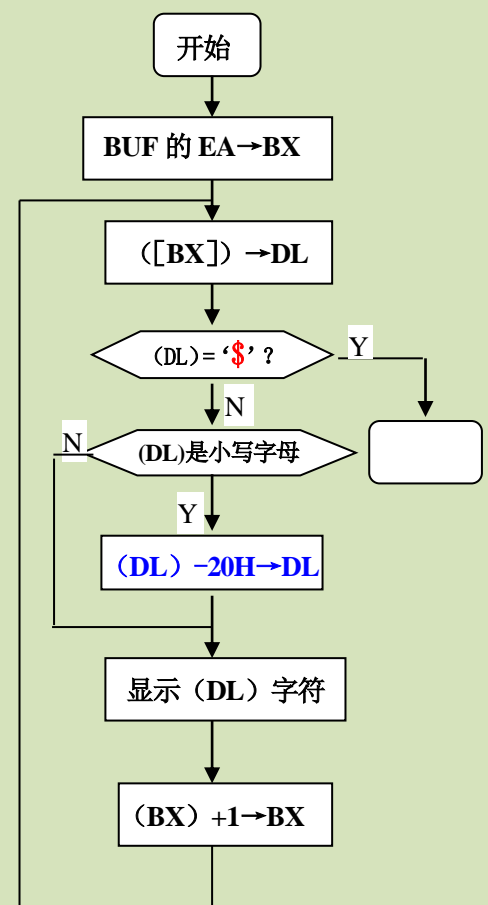
JMP **NEXT**

**EXIT:** MOV AH, 4CH

INT 21H

CODE ENDS

END START



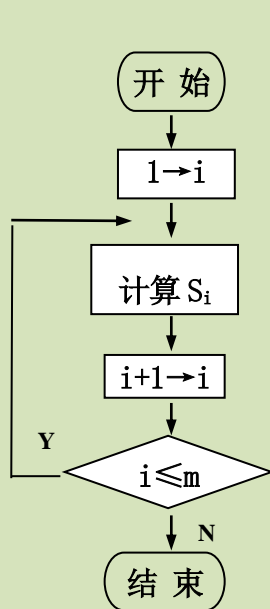
## （二）多重循环程序设计

例：（P114）已知  $m \times n$  矩阵  $A$  的元素  $a_{ij}$  按行序（即：先依次存放第一行的第一列至最后一列的元素，接着存放第二行的第一列至最后一列的元素，依次类推，直至最后一行的最后一列）存放在以 BUFA 为首址的字节存储区中，试编写程序，求每行元素之和  $S_i$ 。其中  $a_{ij}$  为 8 位有符号二进制数。

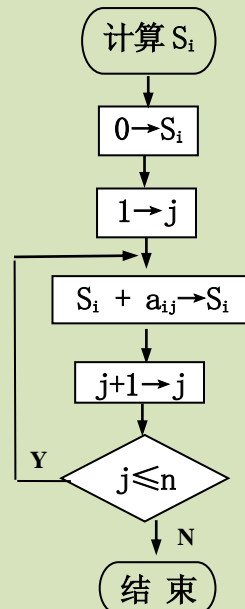
第  $i$  行元素之和  $S_i$  的计算公式为：

$$S_i = a_{i1} + a_{i2} + a_{i3} + \cdots + a_{in} \quad (i=1, 2, \cdots, n)$$

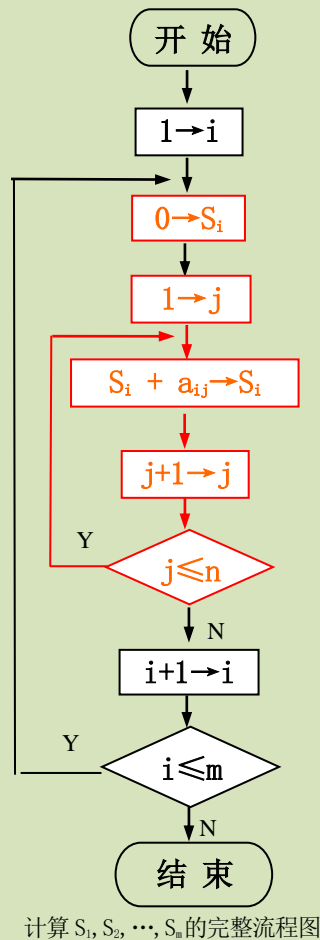
也即  $S_1, S_2, \cdots, S_m$  的计算过程完全一致，可用循环实现。



计算  $S_1, S_2, \cdots, S_m$  的流程图



计算  $S_i$  的流程图



图中的变量  $i, j, S_i, a_{ij}$  必须与计算机的存储单元或寄存器联系起来才能编制程序。如：

BX：用来存放外循环控制变量  $i$  的值，初值为 1，每循环一次加 1。

CX：用来存放内循环控制变量  $j$  的值，初值为 1，每循环一次加 1。

DX：用来累加一行元素之和  $S_i$ 。

AX：用来存放符号扩展后的  $a_{ij}$ 。

}