

机器语言、汇编语言和高级语言

机器语言：用二进制代码表示指令和数据，CPU可直接识别。

汇编语言：用助记符表示指令操作功能，直接面向机器硬件。用汇编语言编写的程序称为汇编语言程序。

把汇编语言转换为机器语言的过程称为“汇编”，完成这种转换的程序称为汇编程序。汇编得到的机器语言称为目标程序。

高级语言：独立于具体的机器，面向过程，接近自然语言和数学表达式。

把高级语言转换为机器语言的过程称为“编译”，完成这种转换的程序称为编译程序。

汇编语言语句的种类和格式

MCS-51单片机汇编语言，包含两类指令语句。

(1) **指令语句**：是机器能够执行的指令，每一条指令都有对应的机器码。

(2) **伪指令语句**：汇编时用于控制汇编的指令，是机器不执行的指令，无机器码。

汇编语言的语句格式

汇编语言源程序是由汇编语句组成的，汇编语言语句一般由四部分组成。

标号：	操作码	操作数	； 注释
START:	MOV	A, 30H	； A ← (30H)

汇编语言语句的种类和格式

标号段 —— 标号是用户根据需要设定的符号地址。通常是在一段程序的入口或程序的转向点设置一个标号。

- 标号由英文字母开头的1~8个字母数字串组成；
- 标号以“:”结束，但第一个字符必须是字母；
- 同一个标号名在一个程序中只能使用一次，不能重复定义；
- 不能使用汇编语言已经定义的符号，如指令助记符、伪指令以及寄存器名称等；
- 语句中标号不是一定要有的，可以没有标号。

操作码段 —— 即指令系统中的助记符。它规定了语句执行的操作属性，是唯一不可缺少的部分。



汇编语言语句的种类和格式

操作数段 —— 操作数用于给指令的操作提供具体数据和地址。

操作数可以是一项或用逗号分开的两项、三项，也可以是空白。

- 十六进制数需后缀加“H”；二进制数需后缀加“B”；十进制数需后缀加“D”，也可以省略。
- 若十六进制数以A—F开头，前面需要加一个0，如：0B5H；
- 允许采用工作寄存器符号、SFR的符号、位符号来表示操作数，也可以用它们的地址来表示操作数。如：MOV 0E0H, #05H和MOV A, #05H是一样的操作。
- 标号可以作为地址操作数，伪指令定义的操作数符号可以表示操作数；
- 符号“\$”表示当前指令所在的地址。

注释段 —— 用于对语句或一段程序进行解释和说明。注释与操作数之间用“；”分开。

伪指令

1. ORG ——汇编起始地址伪指令

指令格式: ORG m

m为十六进制或十进制数。它规定了下面的程序或数表应从ROM的m地址处开始存放。

一个汇编语言源程序中，可以多次使用ORG命令，地址一般应从小到大，且不能使各程序段出现重叠现象。

例如:

```
ORG 0000H
LJMP MAIN      ; 本指令从0000H开始存放
      |
ORG 0030H
MAIN: MOV R0, #40H ; 本指令从0030H开始存放
```

伪指令

2. END —— 汇编结束伪指令

指令格式: END

END指令用于终止源程序的汇编工作。一般一个源程序只能有一个END，且位于程序的最后。

3. EQU —— 赋值伪指令

指令格式: 标号名称 EQU 汇编符号或数

EQU右边的数或汇编符号（地址或常数）赋给左边的标号名称，在整个程序有效，必须先定义后使用。

```
LONG EQU 50H
```

```
ZZ EQU R0
```

```
MOV A, @ZZ ; R0间接寻址单元的内容送A
```

```
MOV R1, #LONG ; 立即数50H送R1
```

伪指令

4. DB —— 定义字节伪指令

指令格式: 标号 DB 字节常数或数表。

表明从该标号地址单元开始定义一个或若干个字节的数。10进制数自动转换为16进制数，字母按ASCII码存储。

例如: ORG 1000H
 TAB: DB 30H, 40H, 24, “C”, “B”

表示从1000H单元开始存放数，汇编后:

(1000H) = 30H

(1001H) = 40H

(1002H) = 18H (10进制数24)

(1003H) = 43H (字符“C”的ASCII码)

(1004H) = 42H (字符“B”的ASCII码)

伪指令

5. DW —— 定义字伪指令

指令格式: 标号: DW 字常数或字数表

类似DB, DW指从该标号地址单元开始, 存放一个或若干个字的数, 高8位在前。

例如: ORG 2000H

TAB1: DW 1234H, 9AH, 10

伪指令DW定义2000H~2005H单元的内容依次为:

(2000H) = 12H, (2001H) = 34H

(2002H) = 00H, (2003H) = 9AH

(2004H) = 00H, (2005H) = 0AH

伪指令

6. DS —— 预留空间伪指令

指令格式: 标号 DS 表达式

DS指定从标号地址单元开始，保留若干字节单元备用。

例如: TAB2: DS 100

通知汇编程序从TAB2开始保留100个字节单元，以备源程序另用。

7. BIT —— 位地址符号伪指令

指令格式: 标号名称 BIT 位地址

一般用来将位地址赋给标号名称，以用户编程和程序阅读。

例如: M0 BIT 20H.0

 MOV C, M0

汇编语言程序设计步骤

用汇编语言编写程序，一般可分为以下几个步骤：

- (1) 分析问题，确定算法；
- (2) 根据算法，设计程序框图以及流程图；
- (3) 确定数据结构，合理地选择和分配内存单元、工作寄存器、I/O、中断、定时器等资源；
- (4) 编写源程序；
- (5) 上机调试程序。

汇编语言源程序的汇编

- **手工汇编**：人工将指令翻译为机器代码。
- **机器汇编**：利用专门的汇编程序生成及其代码。

机器汇编中，汇编语言程序必须经过汇编、链接，生成机器语言程序方可被单片机执行。有关文件的后缀为：

- *.ASM -- 汇编语言源程序，ASCII码文件
- *.OBJ -- 汇编语言源程序经过汇编后形成的浮动目标代码文件。
- *.HEX -- 浮动目标代码经过链接后形成的绝对地址目标文件，**可以被大多数编程器所读取，直接写入到EPROM、EEPROM和单片机中，也就是机器代码。**
- *.LST -- 汇编语言源程序经过汇编后形成的列表文件。

汇编语言程序的基本结构形式

汇编语言程序的基本结构：顺序结构、分支结构和循环结构、子程序、中断服务子程序。

1) 顺序结构；

2) 分支结构；

程序中含有转移指令，分为：**无条件分支**，**有条件分支**。

有条件分支又分为：**单分支结构**和**多分支结构**。

3) 循环结构；

4) 子程序；

5) 中断服务子程序。

子程序的设计

一、子程序设计原则和应注意的问题

一种能完成某一特定任务的程序段。其资源要为所有调用程序共享。因此，子程序在结构上应具有独立性和通用性，在编写子程序时应注意以下问题：

1) 子程序的第一条指令的地址称为子程序的入口地址。该指令前必须有标号。

2) 主程序通过调用指令调用子程序。

(1) 短调用指令： **ACALL addr11**

(2) 长调用指令： **LCALL addr16**

子程序的设计

- 3) 注意设置堆栈指针和现场保护。
- 4) 最后一条指令必须是RET指令。
- 5) 子程序可以嵌套，即子程序可以调用子程序。
- 6) 在子程序调用时，还要注意参数传递的问题。

二、子程序的基本结构

```
MAIN:      ; MAIN为主程序或调用程序标号
           |
           |
LCALL SUB   ; 调用子程序SUB
           |
```



子程序的设计

SUB: PUSH PSW ; 现场保护

PUSH ACC

子程序处理程序段

POP ACC ; 现场恢复

POP PSW

RET ; 最后一条指令必须为RET

例 单字节有符号数的加减法子程序

本例中程序功能是 $(R2) \pm (R3)$ ，R2、R3是有符号数，R7存放计算结果。

参数传递是通过累加器A完成的，主程序将被转换的数送到A中，子程序将A中的有符号数求补后再存回A中。

若需要，该子程序需要补充保护现场。



子程序的设计

```
MAIN:      ; 主程序入口
           ; 调用减法子程序
LCALL SUB1
           ; 调用加法子程序
LCALL ADD1
           ;
```

```
SUB1:      MOV     A, R3
           CPL     ACC.7    ; 符号位取反
           MOV     R3, A
ADD1:      MOV     A, R3
           LCALL   CMPT    ; 求补子程序
           MOV     R3, A
           MOV     A, R2
           LCALL   CMPT    ; 求补子程序
           ADD     A, R3
           JB      OV, OVER
           LCALL   CMPT    ; 求补子程序
           MOV     R7, A
OVER:      RET
```


查表程序设计

数据补偿、修正、计算、转换等各种功能，具有程序简单、执行速度快等优点。

查表就是根据自变量 x ，在表格中寻找 y ，使 $y=f(x)$ 。

在MCS-51的指令系统中，给用户提供了两条极为有用的查表指令：

1) **MOVC A, @A+DPTR**

2) **MOVC A, @A+PC**

指令“**MOVC A, @A+DPTR**”完成把A中的内容作为一个无符号数与DPTR中的内容相加，所得结果为某一程序存储单元的地址，然后把该地址单元中的内容送到累加器A中。

该指令执行完后，DPTR的内容不变。



查表程序设计

指令“**MOVC A, @A+PC**”以PC作为基址寄存器，PC的内容和A的内容作为无符号数，相加后所得的数作为某一程序存储器单元的地址，根据地址取出程序存储器相应单元中的内容送到累加器A中。

- 指令执行完，PC的内容不发生变化，仍指向查表指令的下一条指令。
- 优点：预处理较少且不影响其它特殊功能寄存器的值，所以不必保护其它特殊功能寄存器的原先值。
- 缺点：该表格只能存放在这条指令的地址之后的00~FFH字节范围之内。表格所在的程序空间受到了限制。

查表程序设计

例 子程序的功能为：根据累加器A中的数x（0~9之间）查x的平方表y，根据x的值查出相应的平方y。x和y均为单字节数。

地 址	子程序
Y3Y2Y1Y0	ADD A, #01H
Y3Y2Y1Y0+2	MOVC A, @A+PC
Y3Y2Y1Y0+3	RET
Y3Y2Y1Y0+4	DB 00, 01, 04, 09, 16 DB 25, 36, 49, 64, 81

第1条指令 ADD A, #01H 的作用是加上偏移量，可以根据A的内容查出x对应的平方。

查表程序设计

MOVC A, @A+DPTR 这条指令的应用范围较为广泛，使用该指令时不必计算偏移量，表格可以设在64K程序存储器空间内的任何地方，而不像 **MOVC A, @A+PC**那样只设在PC下面的255个单元中，使用较方便。

前例的程序可改成如下形式：

```
PUSH  DPH                ; 保存DPH
PUSH  DPL                ; 保存DPL
MOV   DPTR, #TAB1
MOVC  A, @A+DPTR
POP   DPL                ; 恢复DPL
POP   DPH                ; 恢复DPH
RET
```

```
TAB1:  DB 00, 01, 04, 09, 16
        DB 25, 36, 49, 64, 81
```

查表程序设计

例 在一个以MCS-51为核心的温度控制器中，温度传感器输出的电压与温度为非线性关系，传感器输出的电压已由A/D转换为10位二进制数。根据测得的不同温度下的电压值数据构成一个表，表中放温度值 y ， x 为电压值数据。设测得的电压值 x 放入R2R3中，根据电压值 x ，查找对应的温度值 y ，仍放入R2R3中。本例的 x 和 y 均为双字节无符号数。

程序如下：

查表程序设计

```
LTB2: MOV  DPTR, #TAB2
      MOV  A, R3   ;  $(R2R3) \times 2 \rightarrow (R2R3)$ 
      CLR  C
      RLC  A
      MOV  R3, A
      XCH  A, R2
      RLC  A
      XCH  R2, A
      ADD  A, DPL ;  $(R2R3) + (DPTR) \rightarrow (DPTR)$ 
      MOV  DPL, A
      MOV  A, DPH
      ADDC A, R2
      MOV  DPH, A
```

查表程序设计

CLR A

MOVC A, @A+DPTR

; 查第一字节

MOV R2, A

; 第一字节存入R2中

CLR A

INC DPTR

MOVC A, @A+DPTR

; 查第二字节

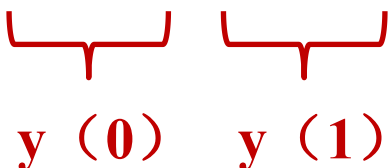
MOV R3, A

; 第二字节存入R3中

RET

TAB2: DW 0004H, 0026H,

; 温度值表


y (0) y (1)

查表程序设计

例 设有一个巡回检测报警装置，需对16路输入进行检测，**每路有一个最大允许值，为双字节数**。运行时，需根据测量的路数，找出每路的最大允许值。根据上述要求，编一个查表程序。

取路数为 x ， y 为最大允许值，放在表格中。设进入查表程序前，路数 x 已放于R2中，查表后最大值 y 放于R3R4中。本例中的 x 为单字节数， y 为双字节数。查表程序如下：

```
TB3:  MOV  A, R2
      ADD  A, R2          ; (R2)*2→(A)
      MOV  R3, A          ; 保存表内位置指针
      ADD  A, #6          ; 加偏移量
```


查表程序设计

```
MOVC A, @A+PC      ; 查第一字节
XCH  A, R3          ; 单字节指令
ADD  A, #3          ; 双字节指令
MOVC A, @A+PC      ; 查第二字节, 单字节指令
MOV  R4, A          ; 单字节指令
RET                ; 单字节指令
TAB3: DW 1520, 3721, 42645, 7580 ; 最大值表
      DW 3483, 32657, 883, 9943
      DW 10000, 40511, 6758, 8931
      DW 4468, 5871, 13284, 27808
```

关键字查找程序设计

关键字查找就是在表中查找关键字的操作，也称数据检索。主要有两种方式：顺序检索和对分检索。

一、顺序检索

从第1项开始逐项顺序查找，判断所取数据是否与关键字相等。

例 从50个字节的无序表中查找一个关键字“××H”。若找到，将关键字所在地址存入R2R3；若找不到，将0000H存入R2R3。

```
ORG 1000H
MOV  30H, #××H           ; 关键字××H送30H单元
MOV  R1, #50              ; 查找次数送R1
MOV  A, #22               ; 修正值送A
MOV  DPTR, #TAB4          ; 表首地址送DPTR
```

关键字查找程序设计

LOOP: PUSH ACC	
MOVC A, @A+PC	; 查表结果送A
CJNE A, 30H, LOOP1	; 不等于 (30H) 中的关键
	; 字则转LOOP1
MOV R2, DPH	; 已查到关键字, 把该字
	; 的地址送R2, R3
MOV R3, DPL	
??????????	; 少一条语句
DONE: RET	
LOOP1: POP ACC	; 修正值弹出
INC A	; A+1→A
INC DPTR	; 修改数据指针DPTR
DJNZ R1, LOOP	; R1≠0, 未查完, 继续查找

关键字查找程序设计

MOV R2, #00H	;	R1=0, 清 “0” R2 和 R3
MOV R3, #00H	;	表中50个数已查完
AJMP DONE	;	从子程序返回
TAB4: DB ..., ..., ...	;	50个无序数据表

二、对分检索（二分检索）

对分检索的前提是数据表已经排好序。检索方法如下：

- 取数据表中间位置的数与关键字比较，如相等，则查找到；
- 如果大于关键字，则下次对分检索范围是从数据表起点到本次取数点；
- 如果小于关键字，则下次对分检索范围是从本次取数点到数据表终点；
- 以此类推，逐步缩小检索范围，直至查找到关键字。



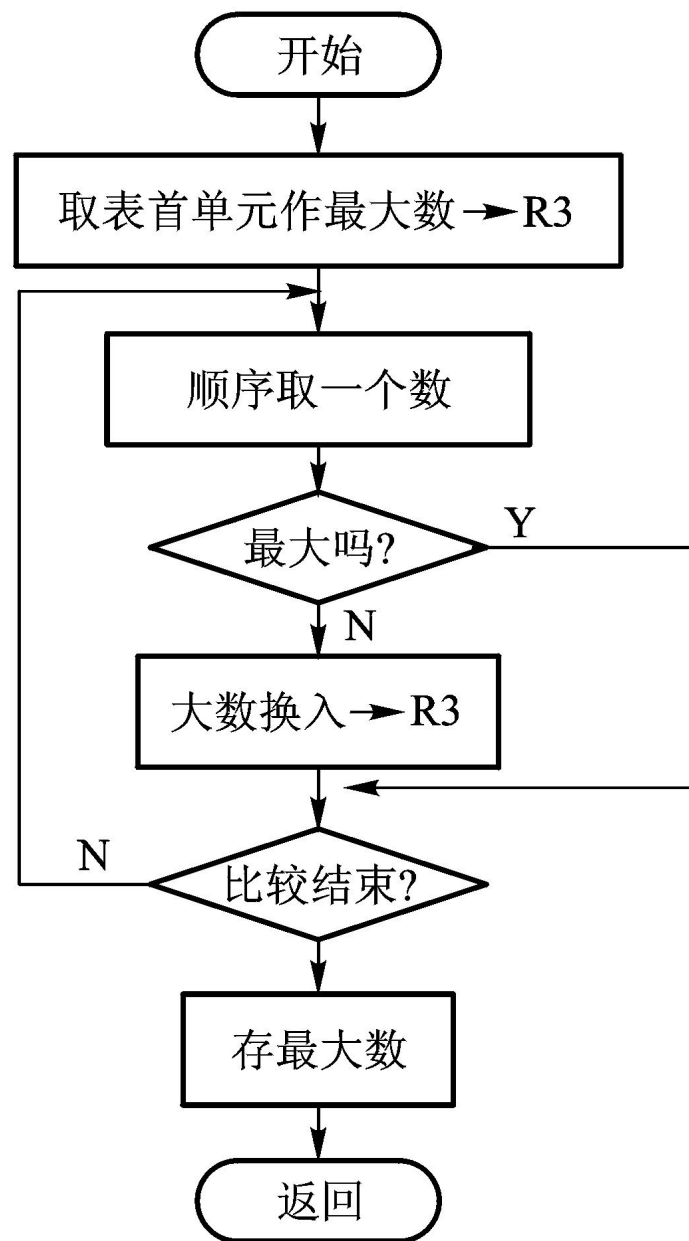
数据极值查找程序设计

在指定的数据区中找出最大值（或最小值）。

进行数值大小的比较，从这批数据中找出最大值（或最小值）并存于某一单元中。

例 片内RAM中存放一批数据，查找出最大值并存放于首地址中。设R0中存首地址，R2中存放字节数，程序框图如图4-1所示。

程序如下：



数据极值查找程序设计

	MOV R2, #n	； n为要比较的数据字节数
	MOV A, R0	； 存首地址指针
	MOV R1, A	
	MOV A, @R1	
	DEC R2	； 循环次数=字节数-1
LOOP:	MOV R3, A	； R3存放的是当前最大的数
	INC R1	； 地址上移
	CLR C	
	SUBB A, @R1	； 两个数比较
	JNC LOOP1	； Cy=0, A中的数大, 跳LOOP1
	MOV A, @R1	； Cy=1, 则大数送A
	SJMP LOOP2	
LOOP1:	MOV A, R3	
LOOP2:	DJNZ R2, LOOP	； 是否比较结束?
	MOV @R0, A	； 存最大数
	RET	

作业4-1

- 一、用于程序设计的语言分为哪几种？它们各有什么特点？
- 二、试编写程序，查找在内部RAM的20H~50H单元中出现“55H”这一数据的次数，并将查找到的结果存入51H单元。
- 三、课件PPT-P26页的例子：从50个字节的无序表中查找一个关键字“××H”，查表语句改用：

MOVC A, @A+DPTR

请重新编写该程序。

数据排序程序设计

升序排，降序排。仅介绍无符号数据升序排。

冒泡法：相邻数互换的排序方法，类似水中气泡上浮。排序时从前向后进行相邻两个数的比较，次序与要求的顺序不符时，就将两个数互换；顺序符合要求不互换。

假设有7个原始数据的排列顺序为：6、4、1、2、5、7、3。第一轮冒泡的过程是：

6、4、1、2、5、7、3	； 原始数据的排列
<u>4</u> 、 <u>6</u> 、1、2、5、7、3	； 逆序，互换
4、 <u>1</u> 、 <u>6</u> 、2、5、7、3	； 逆序，互换
4、1、 <u>2</u> 、 <u>6</u> 、5、7、3	； 逆序，互换
4、1、2、 <u>5</u> 、 <u>6</u> 、7、3	； 逆序，互换



数据排序程序设计

4、1、2、5、6、7、3

；正序，不互换

4、1、2、5、6、3、7

；逆序，互换，第一轮冒泡结束

如此进行，各轮冒泡的结果如下：

第1轮冒泡结果：4、1、2、5、6、3、7

第2轮冒泡结果：1、2、4、5、3、6、7

第3轮冒泡结果：1、2、4、3、5、6、7

第4轮冒泡结果：1、2、3、4、5、6、7 ；已完成排序

第5轮冒泡结果：1、2、3、4、5、6、7

第6轮冒泡结果：1、2、3、4、5、6、7

对于n个数，理论上应进行 (n-1) 轮冒泡，有时不到 (n-1) 轮就已完成排序。



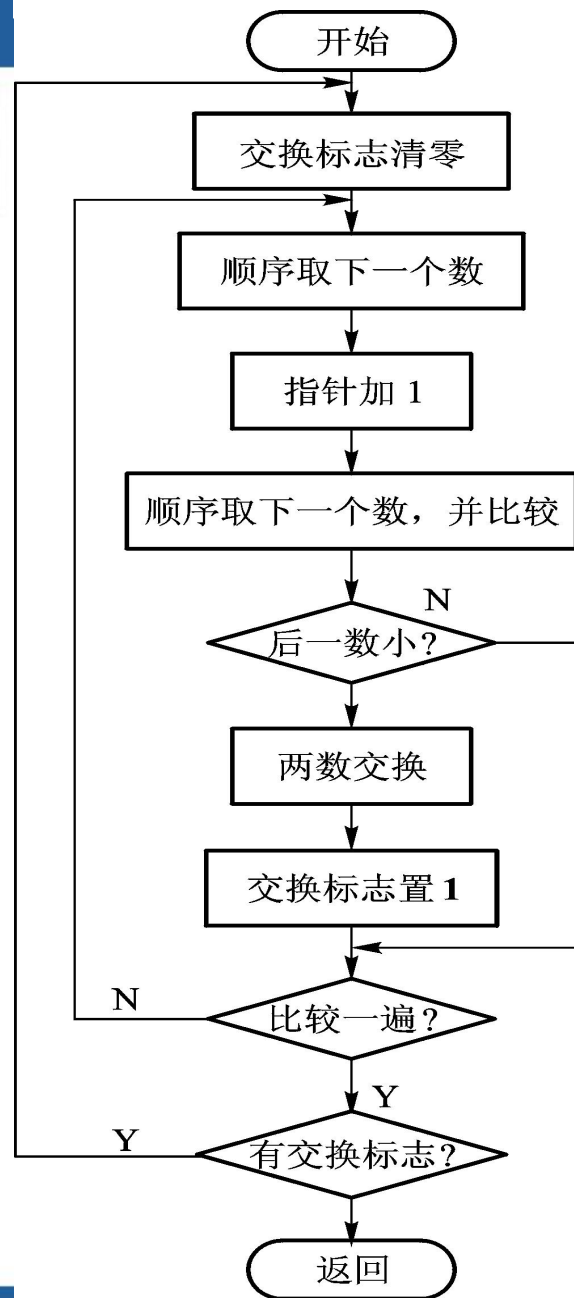
数据排序程序设计

如何判定排序是否已完成，看各轮冒泡中是否有互换发生，如果有数据互换，则排序还没完成。

在程序设计中，常使用设置互换标志的方法，该标志的状态表示在一轮冒泡中是否有互换进行。

例 一批单字节无符号数，以R0为首地址指针，R2中为字节数，将这批数进行升序排列。**程序框图如图4-2所示。**

程序如下：



数据排序程序设计

SORT:	MOV A, R0	
	MOV R1, A	; R1作为取数据的地址指针
	MOV A, R2	; 字节数送入R5
	MOV R5, A	
	CLR F0	; 互换标志位F0清零
	MOV A, @R1	; 取冒泡比较的第1个数
	DEC R5	; 每一轮比较次数=字节数-1
LOOP:	MOV R3, A	
	INC R1	; 指向冒泡比较的第2个数
	CLR C	
	MOV A, @R1	; 比较大小
	SUBB A, R3	;
	JNC LOOP1	; \geq 跳转
	SETB F0	; 第2个数小于第1个数, 置标志位F0

数据排序程序设计

	MOV A, R3	;
	XCH A, @R1	; 第1个数放到第2个数的位置
		; 第2个数暂存到A
	DEC R1	; 地址指针指向第1个数
	MOV @R1, A	; 第2个数放到第1个数的位置
	INC R1	; 取数指针指回第2个数
LOOP1:	MOV A, @R1	; 取下一轮冒泡比较的第1个数
	DJNZ R5, LOOP	
	DEC R2	; 本轮最大值已拍到最后, 排序数减1
	JB F0, SORT	; F0若为1, 冒泡尚未结束, 继续
	RET	

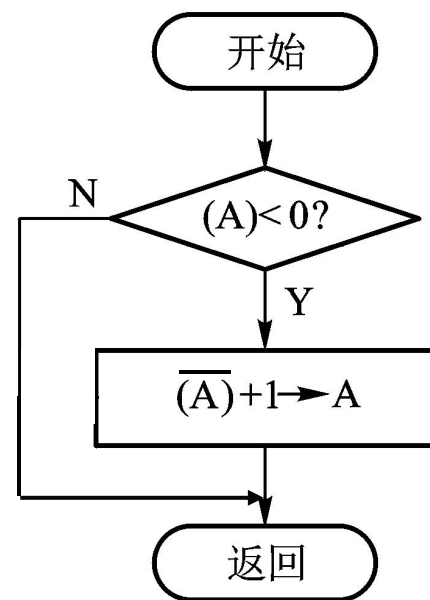
分支转移程序设计

分支转移程序的特点是程序中含有转移指令，根据指令转移条件可分为**无条件分支转移程序**和**有条件分支转移程序**。有条件分支转移程序又分为**单分支转移结构**和**多分支转移结构**。

一、单分支转移结构：仅有两个出口，两者选一。

例 求单字节有符号数的二进制补码。参考程序如下：

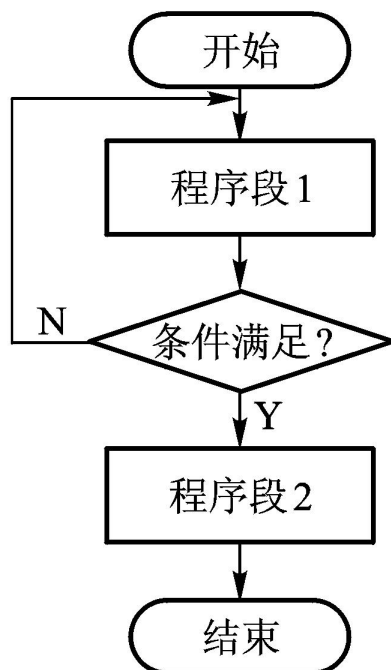
CMPT: JNB	Acc.7, RTN	; (A) > 0, 不需转换
MOV	C, Acc.7	; 符号位保存
CPL	A	; (A) 求反, 加1
ADD	A, #1	
MOV	Acc.7, C	; 符号位存A的最高位
RTN:	RET	



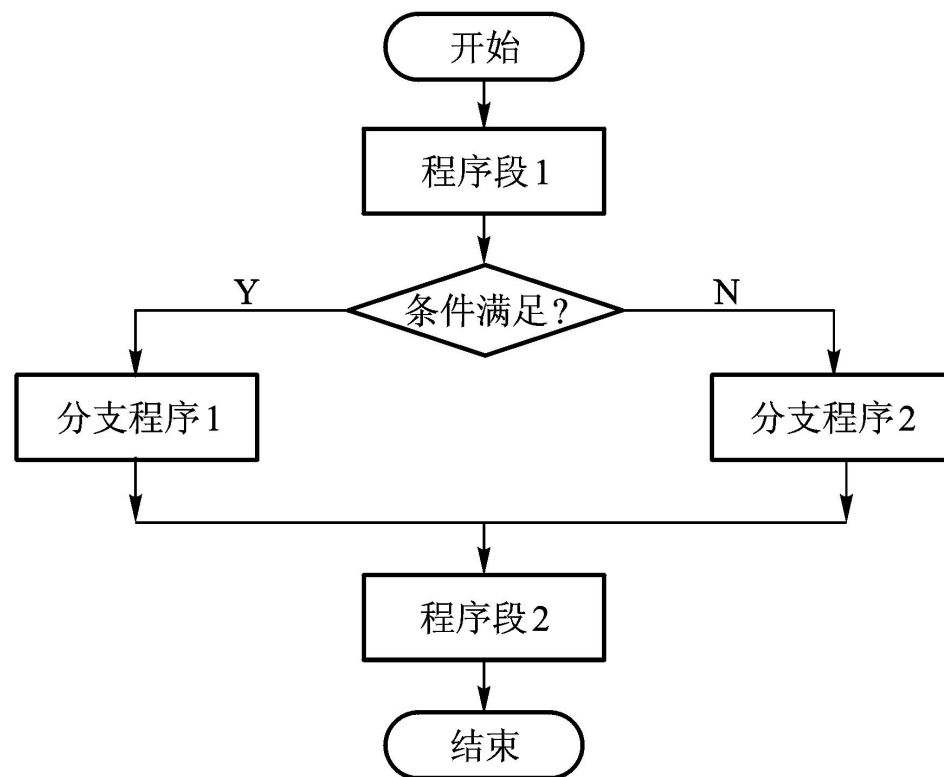
4-3

分支转移程序设计

此外，单分支选择结构还有如图4-4、图4-5等所示的几种形式：



4-4



4-5

分支转移程序设计

二、多分支转移结构：程序的判别部分有两个以上的出口流向。

多分支转移结构一般使用以下两种多分支选择指令实现：

- (1) 间接转移指令: `JMP @A+DPTR;`
- (2) 比较不相等转移指令:
`CJNE A, direct, rel;`
`CJNE A, #data, rel;`
`CJNE Rn, #data, rel;`
`CJNE @Ri, #data, rel;`

第(1)种多分支转移方式是由DPTR指针决定转移程序的首地址，由A的内容选择对应的分支程序。第(2)多分支转移方式是通过比较内容，根据比较结果选择对应的分支程序。

思考：这两种方式各有什么优缺点。

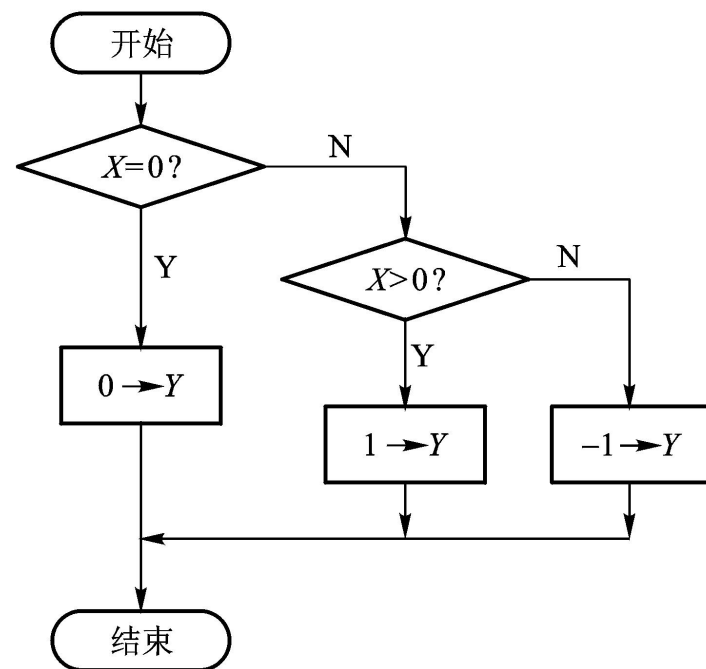
分支转移程序设计

多分支转移结构常见的两种形式。如图4-6和图4-7

例：求如下符号函数的值，X存放在40H，Y存放在41H：

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

```
SIGNFUC:  MOV  A, 40H
          CJNE A, #00H, NZEAR
          AJMP: NEG T
NZEAR:    JB  ACC.7, POSI
          MOV  A, #01H
          AJMP NEG T
POSI:     MOV  A, #81H      ; A=-1
NEG T:    MOV  41H, A
          END
```



4-6

分支转移程序设计

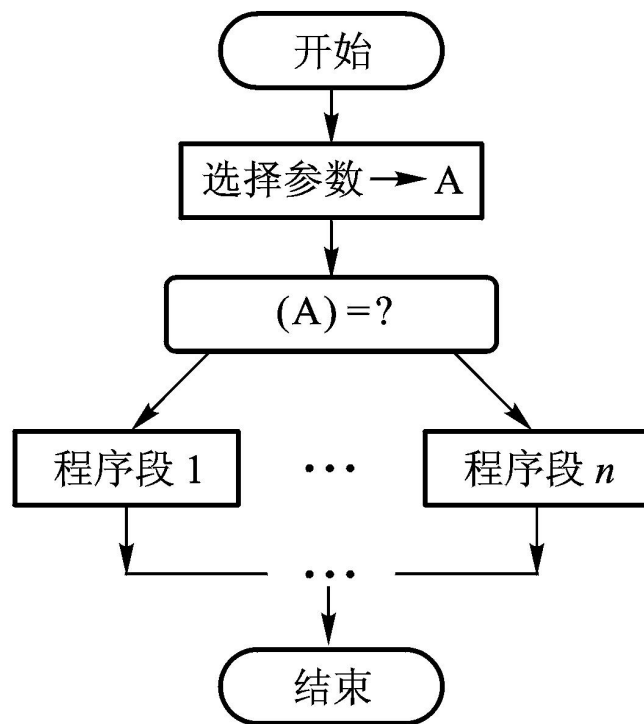
例：根据寄存器R2的内容，转向各个处理程序PRG_x（x=0~n）。

(R2) = 0, 转PRG₀;

(R2) = 1, 转PRG₁;

.....

(R2) = n, 转PRG_n;



4-7

根据某一单元的内容是0, 1,, n, 来分别转向处理程序0, 处理程序1,, 处理程序n。对于这种情况, 可用直接转移指令 (LJMP或AJMP指令) 组成一个转移表, 然后把该单元的内容读入累加器A, 转移表首地址放入DPTR中, 再利用间接转移指令实现分支转移。

分支转移程序设计

程序如下:

JMP6:	MOV DPTR, #TAB5	； 转移表首地址送DPTR
	MOV A, R2	； 分支转移参量送A
	MOV B, #03H	； 乘数3送B
	MUL AB	； 分支转移参量乘3
	MOV R6, A	； 乘积的低8位暂存R6
	MOV A, B	； 乘积的高8位送A
	ADD A, DPH	； 乘积的高8位加到DPH中
	MOV DPH, A	
	MOV A, R6	
	JMP @A+DPTR	； 多分支转移选择
	
TAB5:	LJMP PRG0	； 多分支转移表，LJMP为3字节指令
	LJMP PRG1	
	
	LJMP PRGn	

分支转移程序设计

程序如下:

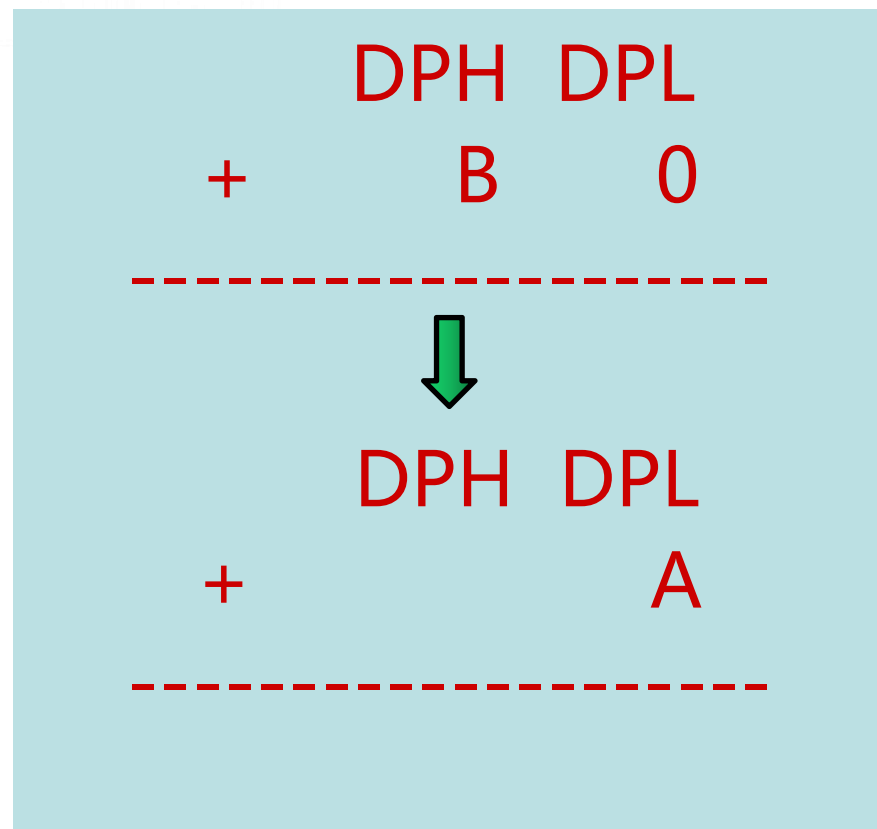
```
JMP6:    MOV  DPTR, #TAB5
          MOV  A,  R2
          MOV  B, #03H
          MUL  AB
          MOV  R6, A
          MOV  A, B
          ADD  A, DPH
          MOV  DPH, A
          MOV  A, R6
          JMP  @A+DPTR
```

.....

```
TAB5:    LJMP  PRG0
          LJMP  PRG1
```

.....

```
LJMP  PRGn
```



; 多分支转移表, LJMP为3字节指令

循环程序设计

循环程序的特点是程序中含有可以反复执行的程序段，该程序段通常称为循环体。

- 可大大缩短程序长度
- 使程序所占的内存单元数量少
- 使程序结构紧凑和可读性变好。

一、循环程序的结构

循环结构程序主要由以下四部分组成。

1. 循环初始化

循环初始化程序段用于完成循环前的的准备工作。例如，循环控制计数初值的设置、地址指针的起始地址设置、为变量预置初值等。

循环程序设计

2. 循环处理

循环程序结构的核心部分，完成实际的处理工作，是需反复循环执行的部分，故又称循环体。这部分程序的内容，取决于实际处理问题的本身。

3. 循环控制

在重复执行循环体的过程中，不断修改循环控制变量，直到符合结束条件，就结束循环程序的执行。循环结束控制方法分为循环计数控制法和条件控制法

4. 循环结束

这部分是对循环程序执行的结果进行分析、处理和存放。

循环程序设计

二、循环结构的控制

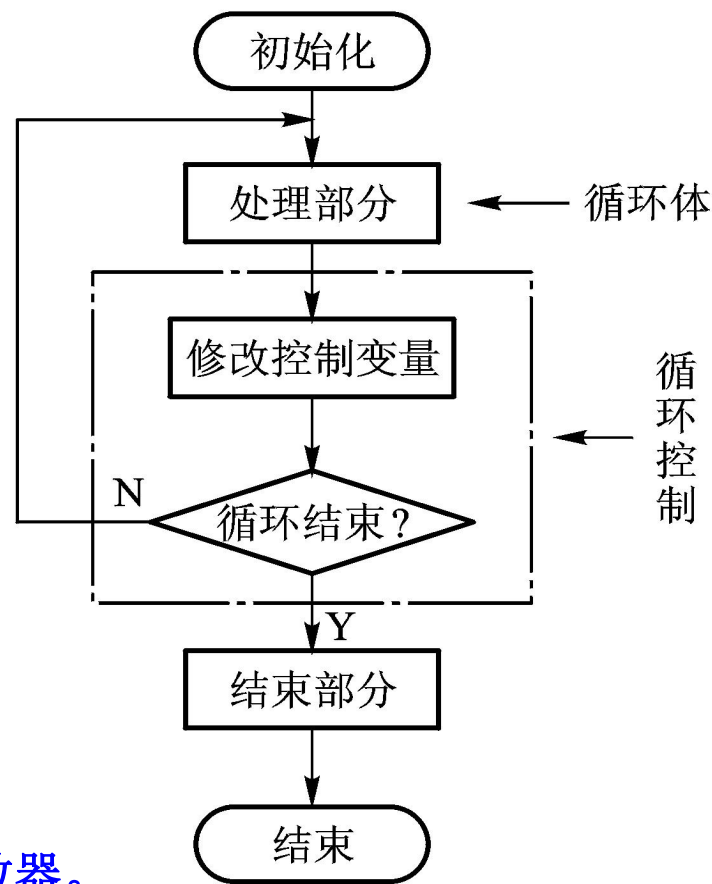
1. 计数循环结构

计数循环控制结构是依据计数器的值来决定循环次数，一般为减“1”计数器，计数器减到“0”时，结束循环。计数器的初值是在初始化时设定。

MCS-51提供的循环控制指令：

DJNZ Rn, rel ; 工作寄存器作控制计数器

DJNZ direct, rel ; 以直接寻址单元作控制计数器。



4-8

循环程序设计

例如： n 个单字节数 x_i ($i=1, \dots, n$)，按 i 顺序存放在内部RAM从50H开始的单元中， n 放在R2中 ($n \leq 255$)，现将它们的求和（双字节）放在R3R4中。

程序如下：

ADD1:	MOV R2, #n	； 循环次数=单字节数的个数
	MOV R3, #0	； 求和单元清零（高位）
	MOV R4, #0	； 求和单元清零（低位）
	MOV R0, #50H	； 置取数首地址
LOOP:	MOV A, R4	； 取求和值的低8位
	ADD A, @R0	； 当前的求和值加上一个单字节数
	MOV R4, A	
	CLR A	
	ADDC A, R3	
	MOV R3, A	
	INC R0	； 取数地址指针+1
TAB5:	DJNZ R2, LOOP	
	END	

循环程序设计

2. 条件控制结构

只有在循环次数已知的情况下才适用。对循环次数未知的问题，不能用循环次数来控制。往往需要根据某种条件来判断是否应该终止循环。

例：设有一串字符，依次存放在内部RAM从30H单元开始的连续单元中，该字符串以0AH为结束标志，编写测试字符串长度的程序。

本例采用逐个字符依次与“0AH”比较的方法。为此设置一个长度计数器和一个字符串指针。长度计数器用来累计字符串的长度，字符串指针用于指定字符。如果指定字符与“0AH”不相等，则长度计数器和字符串指针都加1，以便继续往下比较；如果比较相等，则表示该字符为“0AH”，字符串结束，长度计数器的值就是字符串的长度。

循环程序设计

程序如下:

```
      MOV R4, #0FFH      ; 长度计数器初值送R4
      MOV R1, #2FH        ; 字符串指针初值送R1
NEXT:  INC R4
      INC R1
      CJNE @R1, #0AH, NEXT ; 比较, 不等则进行下一个字符比较
      END
```

多重循环: 循环体中还包括其他循环。

例: 软延时子程序, 假设一个机器周期为1us。(前面举过的例子)

```
DELAY: MOV R5, #200          ; 1个机器周期指令
D1:    MOV R6, #250          ; 1个机器周期指令
      DJNZ R6, $              ; 2个机器周期指令
      DJNZ R5, D1             ; 2个机器周期指令
      RET
```