

# 第7章 单片机“起舞” ——指令的执行

欢迎访问 电路飞翔网

<http://www.circuitfly.com> 获取更多信息

- 7.1 汇编语言详谈
- 7.2 时钟与指令的执行
- 7.3 寻址方式
- 7.4 实例点拨——程序存储器和数据存储器的寻址

# 7.1 汇编语言详谈

- 汇编语言很原始，直接与计算机的机器码打交道。
- 高级语言让程序员减少出错率和提高工作效率。

（用汇编语言写成的）汇编程序

（汇编器）的汇编

（十六进制的）执行代码

（用高级语言写成的）高级语言程序

（编译器）的编译

（可执行的）文件

汇编与编译的过程

# 7.1 汇编语言详谈

## • 7.1.1 汇编器

几个定义：

- 指令（**instruction**）：控制单片机的命令行
- 汇编程序（**assembly program**）：控制目标系统（如单片机）实现特定功能的指令集合。
- 汇编语言（**assembly language**）：它是一种低级的计算机语言。
- 源文件（或源代码，**source code file**）：以.asm为后缀的文件，其中包含一条一条的指令。
- 目标文件（或目标代码，**object code**）：编译器/汇编器从源代码文件中生成的文件。
- 执行代码（**opcode or operation code**）：指定了操作如何执行，如.HEX文件中的十六进制代码。

# 7.1 汇编语言详谈

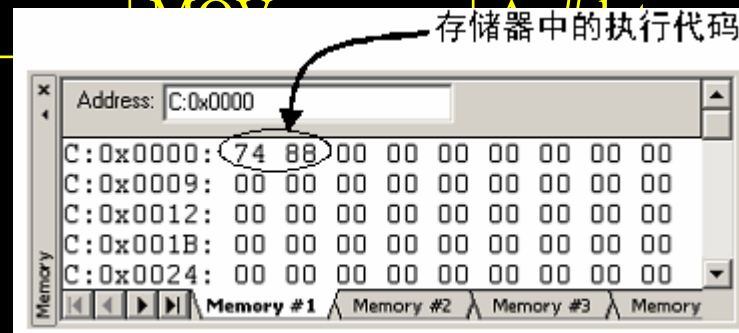
## 7.1.1 汇编器

汇编器的功能为：把（源文件中的）指令汇编成（目标文件中的）执行代码。汇编器还可以自动定义汇编程序中的标号，即自动在存储器中为标号分配地址。伪指令不会被汇编器汇编。

执行代码（十六进制）	字节数	助记符	操作数
.....	.....	.....	.....
73	1	JMP	@A+DPTR
74	2	MOV	A, #10H

```
ORG    00H
START:
        MOV    A, #88H
        END
```

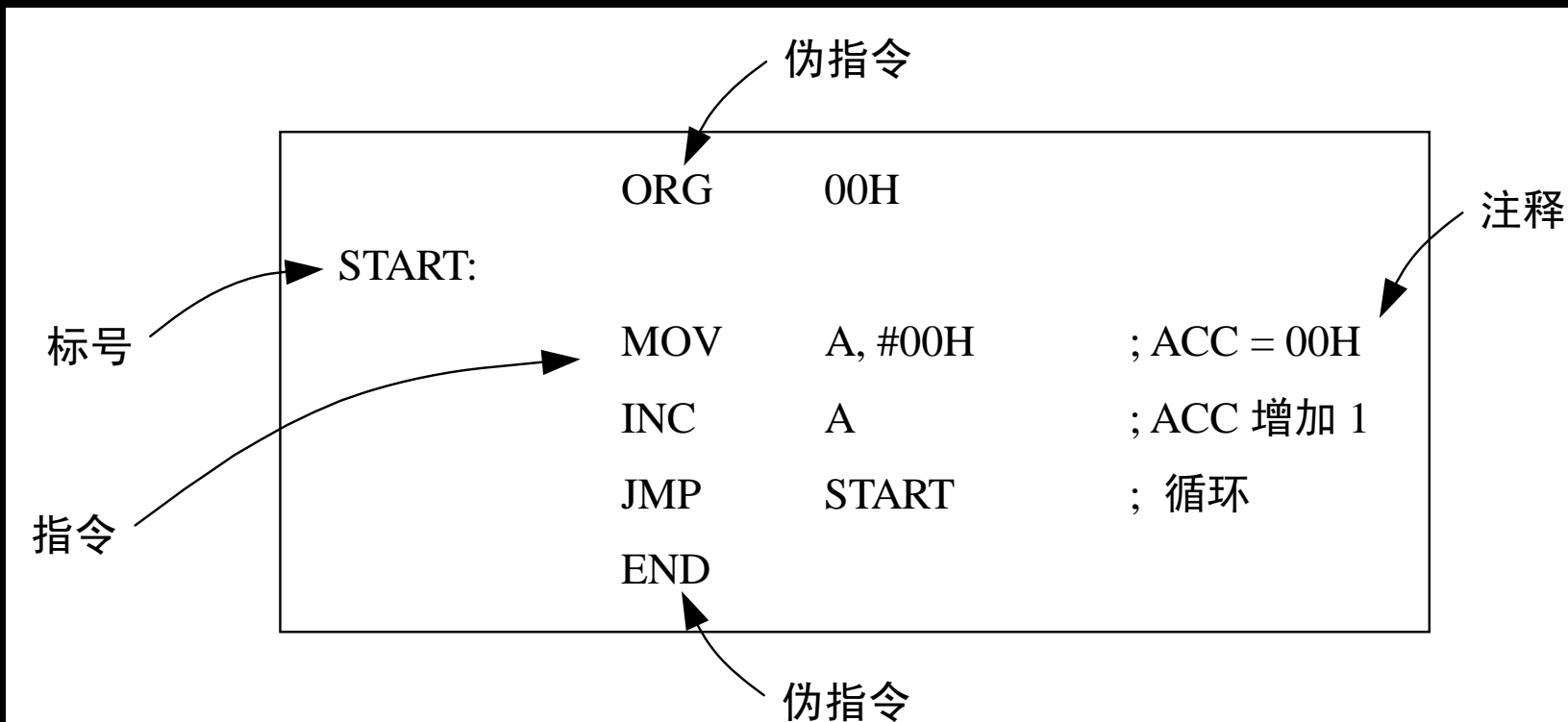
(a) 汇编程序



(b) 程序存储器中的执行代码

# 7.1 汇编语言详谈

## • 7.1.2 汇编程序书写格式



## 汇编程序书写格式

## 7.1 汇编语言详谈

### • 7.1.3 伪指令

伪指令（pseudo opcode）不被汇编器汇编，没有执行代码，但却告知汇编器一些汇编信息。

伪指令简介：

✓ BIT——用于汇编程序的一开始创建一个常量。

FLASH\_COUNT                      BIT                      3EH

； 创建一个名为FLASH\_COUNT的常量，并把立即数3EH赋给这个常量，在程序中就可以直接把FLASH\_COUNT等同于立即数3EH进行操作。

# 7.1 汇编语言详谈

## • 7.1.3 伪指令

伪指令简介：

- ✓ **DATA**——定义一个指向特殊功能寄存器区地址的变量。

**DPTRSW            DATA            0A2H**

； **DPTRSW**指向特殊功能寄存器**0A2H**地址上

- ✓ **DB**——用于汇编程序中定义若干个长度为**1**个字节的字，这若干个字使用逗号分隔开，如果逗号之间没有数据，汇编器默认为**00H**。

**DB            10H, 11H, , 3FH, 20H**

； 在目标文件中生成**10H, 11H, 00H, 3FH, 20H**

## 7.1 汇编语言详谈

### • 7.1.3 伪指令

伪指令简介：

- ✓ **DS**——用于保留一块存储器空间给程序变量使用或别的用途。

**STORAGE      DS      10**

；保留一块名叫“STORAGE”的10字节存储空间

- ✓ **DW**——定义若干个长度为两个字节的字，这若干个字使用逗号分隔开，如果逗号之间没有数据，汇编器默认为0000H。

**DW      0FFFEH, , 0102H**

；在目标文件中生成代码：FFH，FEH，00H，00H，01H，02H



# 7.1 汇编语言详谈

## • 7.1.3 伪指令

伪指令简介：

- ✓ **END**——该伪指令告诉汇编器程序的结束点。
- ✓ **EQU**——定义某一个符号的值，一旦一个符号被定义后，就不能被另一个**EQU**或**SET**指令重复定义。

**BEEP\_COUNT EQU 1+1**

；表达式把2定义给符号**BEEP\_COUNT**

# 7.1 汇编语言详谈

## • 7.1.3 伪指令

伪指令简介：

- ✓ **IF、ELSE、ENDIF**——这3个伪指令是条件选择语句，它们告诉汇编器根据表达式的值，是否汇编某一块程序，没有汇编的块在目标文件中是不存在对应的执行代码的。

**IF P1.0;** 如果**P1.0=1**，就汇编下一行

**DB 01H, 02H, 03H;** 在存储器中定义字**01H、02H、03H**

**ENDIF;** 条件选择结束，如果**P1.0≠1**，上一行不被汇编

# 7.1 汇编语言详谈

## • 7.1.3 伪指令

伪指令简介：

- ✓ **INCL**——该伪指令用于在汇编时把其他文件与当前文件结合在一起汇编。

**INCL**                   "const.def"

； 即把文件“const.def”与当前文件结合在一起汇编。

- ✓ **ORG**——该伪指令用于设置程序计数器PC的初始值。

**ORG**                   0000H

； 指令的执行代码在单片机的程序存储器中从0000H开始存储（也可简单写成00H）。

## 7.1 汇编语言详谈

### • 7.1.3 伪指令

伪指令简介：

- ✓ **SET**——该伪指令类似**EQU**，但不同的是**SET**可以通过另一个**SET**伪指令重复定义变量的值。

```
COUNT          SET      3  
COUNT          SET      1 ; 最终COUNT=1
```

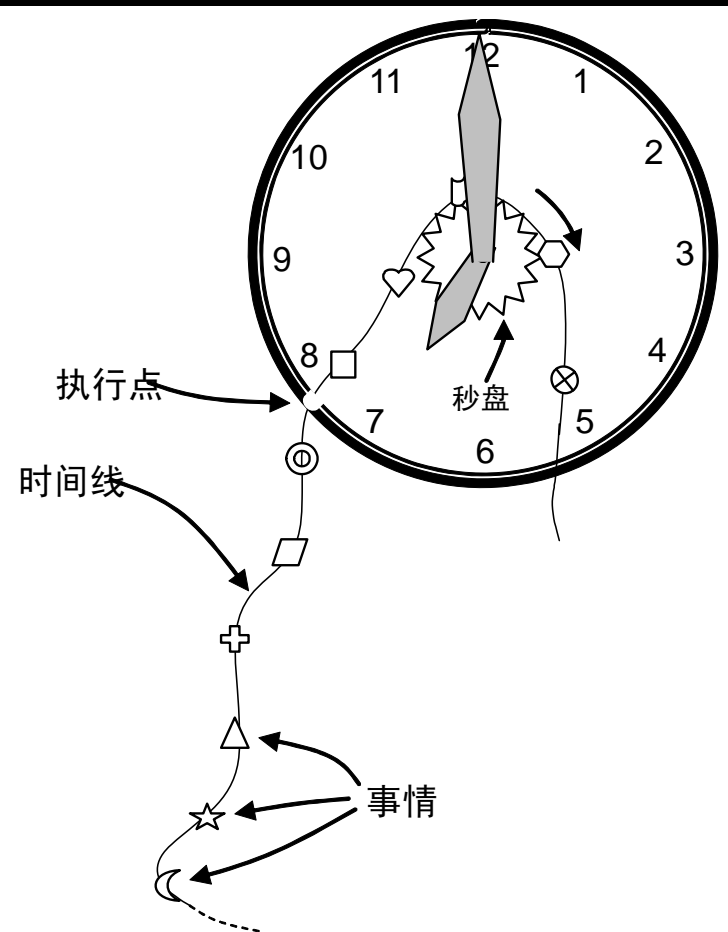
- ✓ **\$**——美元符号表示当前地址，意味着程序计数器**PC**的值不变，在程序中表示“本行程序”。

```
DJNZ            R5,    $
```

工作寄存器**R5**减1，如果不等于0就跳回本行——直到**R5**减至0为止，执行下一条指令。

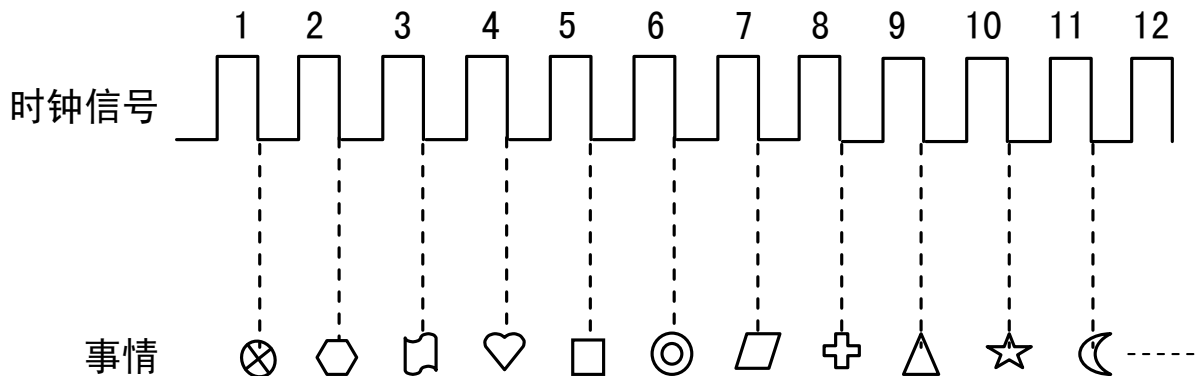
## 7.2 时钟与指令的执行

### • 7.2.1 时钟究竟是什么



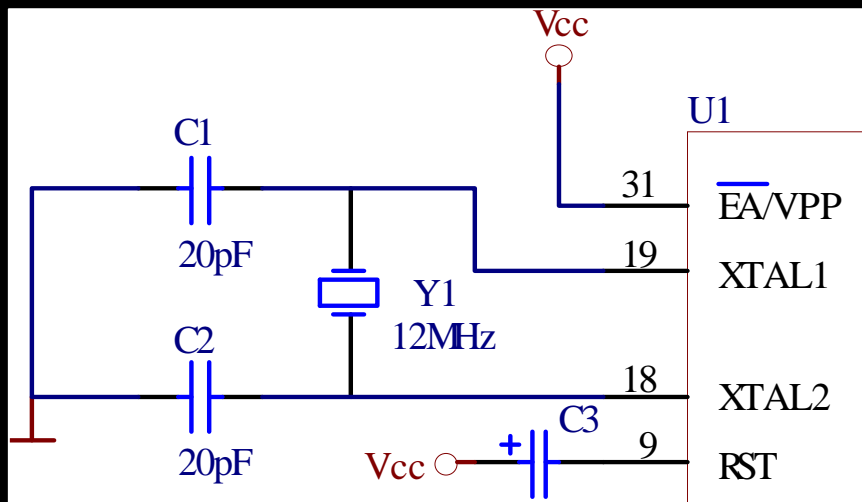
◁ 事情与时间线

▽ 时钟信号波形与事情

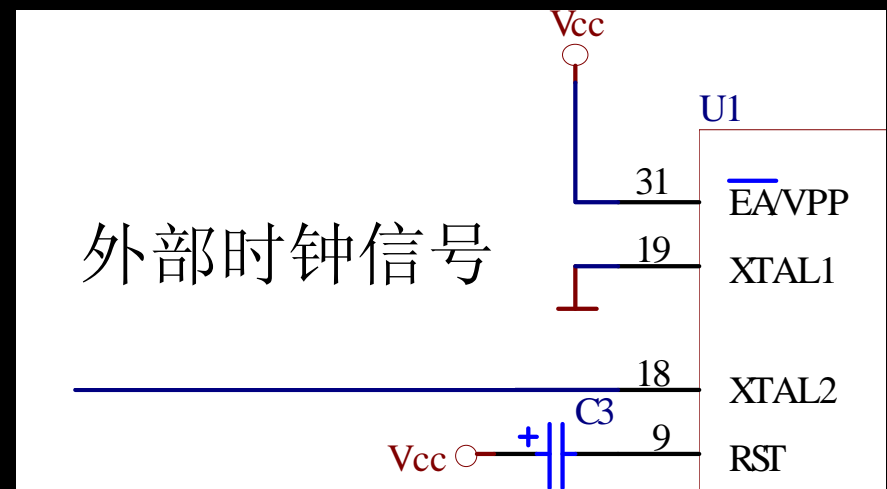


## 7.2 时钟与指令的执行

### • 7.2.1 时钟究竟是什么(单片机的时钟信号源)



内部时钟



外部时钟

## 7.2 时钟与指令的执行

- 7.2.1 时钟究竟是什么(有关时钟信号周期的概念 )
  - ✓ 振荡周期——振荡器产生的时钟信号的周期。
  - ✓ 机器周期——1个机器周期=12个振荡周期。
  - ✓ 指令周期——执行一条指令所占用的全部时间，一般为1~4个机器周期。

例：使用的是内部时钟，晶振频率为12MHz，则有：

振荡周期=1/晶振频率=1/12μs

机器周期=12×振荡周期=1μs

“DJNZ R4,DELAY”的指令周期为两个机器周期，执行时间为2×机器周期=2×1μs =2μs

## 7.2 时钟与指令的执行

- 7.2.2 程序计数器PC的角色

程序计数器PC，它是一个两个字节长度的寄存器，这个寄存器中的数值告诉单片机要到哪一个地址去找下一条指令来执行。

程序一开始定义“ORG 00H”，程序计数器PC从0000H开始，当第一条指令执行后，PC自增1，指示下一条指令从0001H中拿出执行。

程序计数器PC是不能被直接修改的，但可以使用跳转指令来改变PC的值，如指令“LJMP 1234H”执行后，PC= 1234H。

没有指令能够读PC的值。



## 7.2 时钟与指令的执行

### • 7.2.3 指令的执行

时钟信号提供时序，程序计数器PC指示执行地址，单片机便可以顺利地执行指令。  
单片机的指令执行大致分为以下5个步骤。

- ① 抓取指令——单片机通过总线向程序存储器读取执行代码和等待处理的过程。
  - ② 单片机内部数据处理——将抓取到的数据和某个寄存器做运算。
  - ③ 逻辑或标志处理——运算完成后，对逻辑或标志进行处理。
  - ④ 转移判断——依照判断的结果进行转移，条件为真时如何动作，条件为假时如何动作。
  - ⑤ 返回——将结果保存到其他寄存器或存储器中。
- 抓取数据 → 处理数据 → 存储数据

## 7.2 时钟与指令的执行

### • 7.2.3 指令的执行(详解指令执行)

“MOV A, 32H”其执行代码为“E5 32”。

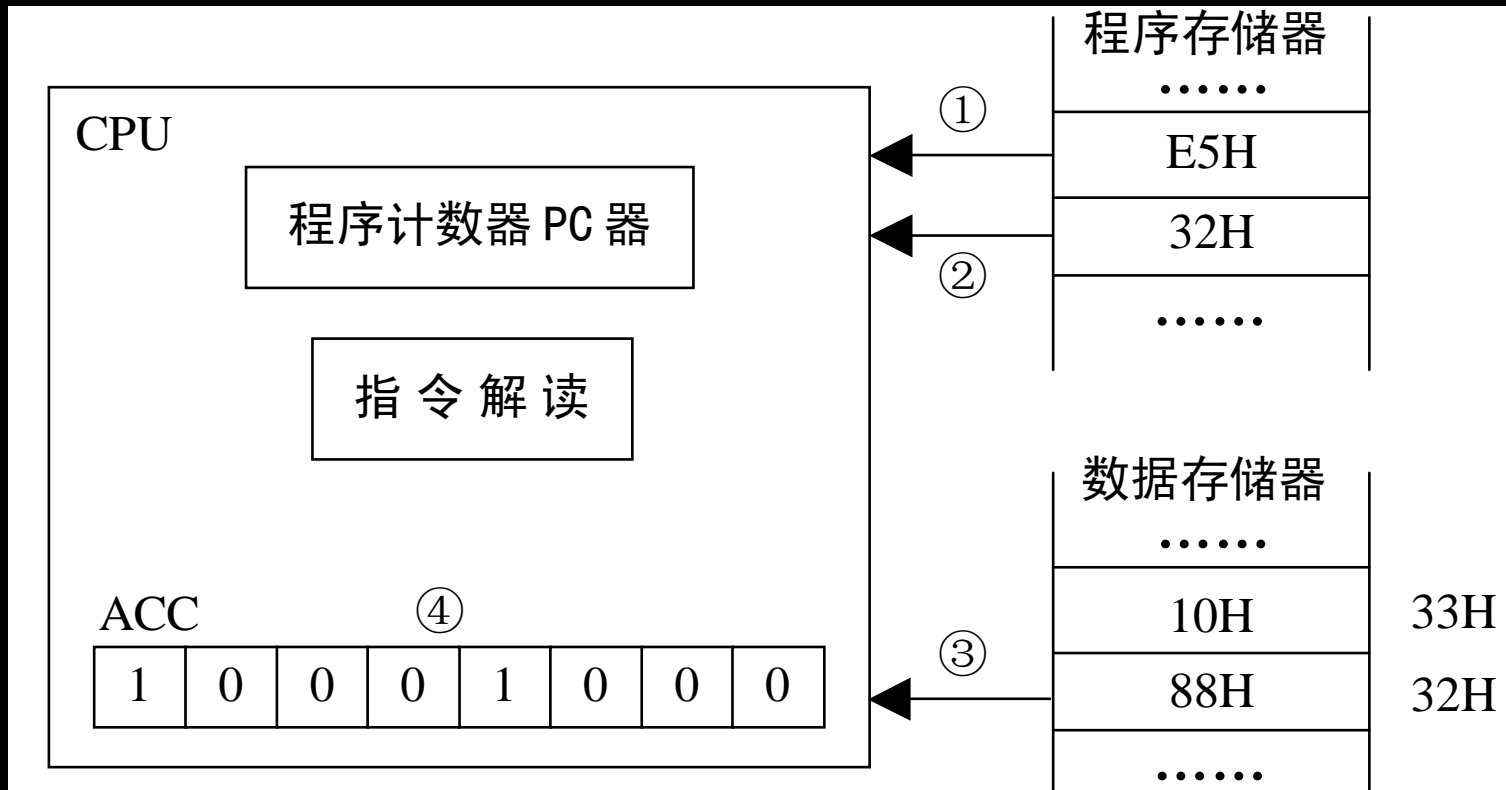
理解：

- ① 当单片机抓取“E5”时，就知道要执行的操作是把下一个执行代码所指向的数据存储器地址中的数据载入ACC。
- ② 于是单片机往下抓取下一个执行代码“32”。单片机根据这个值知道要去取数据存储器地址32H中的内容。
- ③ 抓取数据存储器32H中的内容“88H”后。
- ④ 再把88H载入到ACC中，于是最终（A）=88H。

抓取数据→处理数据→存储数据

## 7.2 时钟与指令的执行

### • 7.2.3 指令的执行(详解指令执行)



- ① 读取执行代码E5；② 获知目标地址是32H；  
③ 取目标地址中的内容；④ 数据载入累加器ACC中。

寻址示意图

## 7.3 寻址方式

### 6种寻址方式

在指令中的地址或立即数如果以十六进制中的A、B、C、D、E或F开头需要在地址或立即数前加“0”。

寻 址 方 式	例 子
直接寻址	MOV A, 30H
间接寻址	MOV A, @R0
寄存器寻址	MOV A, R0
寄存器特征寻址	INC A
立即寻址	MOV A, #8FH
变址寻址	MOVC A, @A+DPTR

## 7.3 寻址方式

### • 7.3.1 直接寻址

直接寻址（**direct addressing**）是指一个直接地址的内容载入一个寄存器中或相反。单片机片内数据存储器的**00H~7FH**（但常用**30H~7FH**）以及特殊功能寄存器**SFR**能被直接寻址。

直接寻址的内容不包含在执行代码中，而是在单片机的数据存储器中，所以执行直接寻址指令的速度还是很快的。

```
MOV  A, 30H    ;把30H地址的内容载入累加器ACC
MOV  50H, B     ;把B寄存器的值载入50H地址内
ADD  A, 60H     ;ACC和60H地址的内容相加，结果存回ACC
```

## 7.3 寻址方式

### • 7.3.2 间接寻址

- ✓ 间接寻址（indirect addressing）是用R0、R1、SP、DPTR中的某一寄存器来代替直接寻址中的直接地址来寻址。例如，直接寻址“MOV A, 32H”与间接寻址是相同的。

MOV R1, #32H ; R1=32H

MOV A, @R1; R1所指向的地址的内容载入ACC中

- ✓ 间接寻址可以访问片内和片外数据存储器。

MOV A, @R0 ; 取得R0所指的存储器地址的内容，并载入ACC

MOVBX A, @DPTR ; 取得DPTR所指的片外数据存储器的内容，并载入ACC

## 7.3 寻址方式

### • 7.3.3 寄存器寻址

寄存器寻址（register instructions）是指与工作寄存器R0～R7有关的寻址指令。程序状态字PSW中的RS1和RS0决定使用AT89S51单片机4组R0～R7寄存器中的哪一组。

MOV       A, R0               ;将R0的值载入累加器ACC

ADD        A, R5               ;将ACC与R5的值相加，并把结果存回ACC

MOV        R7, A               ;将ACC的值载入R7中

## 7.3 寻址方式

### • 7.3.4 寄存器特征寻址

寄存器特征寻址（**register-specific instructions**）是一类与特定寄存器有关的寻址方式。指令不涉及用于指向的地址字节，不对其他地址和数据产生什么影响。例如：

**INC          A                          ;累加器ACC自增1**

**SWAP        A                          ;ACC的高4位与低4位互换**

**INC          DPTR                      ;数据指针寄存器DPTR自增1**



## 7.3 寻址方式

- 7.3.5 立即寻址

立即寻址（**immediate addressing**）是一类与立即数相关的寻址方式。当程序中需要向寄存器或地址中载入某个立即数，就可采用立即寻址。立即数的特征就是在常数前加一个“#”号。这类寻址方式比较简单。

例如：

**MOV A, #100** ;将立即数100（64H）载入累加器AC

**MOV 33H, #10H** ;将立即数10H载入数据存储器33H中

**MOV R0, #0FFH** ;R0=FFH

**MOV DPTR, #2046H** ;DPTR=2046H

## 7.3 寻址方式

### • 7.3.6 变址寻址

变址寻址（indexed addressing）针对的是程序存储器，只能从程序存储器中读数据。通常我们对程序存储器访问得较多的是数据表中的数据。在变址寻址中，使用程序计数器PC或数据指针寄存器DPTR作为间接地址，有时还加上累加器ACC。

例如：

**MOVC A, @A+DPTR** ;查表操作：累加器ACC和数据指针寄存器DPTR的值相加得到一个程序存储器上的地址，将该地址的内容载入ACC中

**MOVX @DPTR, A** ;访问片外存储器：将ACC的值载入DPTR所指的片外存储器的地址上

## 7.4 实例点拨

### ——程序存储器和数据存储器的寻址

#### • 7.4.1 把程序存储器中的数据载入工作寄存器中

例：将程序存储器中的200H~202H上的“C”、“A”、“T”3个字节数据载入工作寄存器R0、R1、R2中。

```
MOV    DPTR, #200H           ; DPTR = 0200H
CLR     A                    ; ACC 清 0
MOVCA,  @A+DPTR              ; 200H 上的数据载入 ACC
MOV     R0,  A                ; R0 = ACC
INC     DPTR                  ; DPTR + 1
```

## 7.4 实例点拨

### ——程序存储器和数据存储器的寻址

#### • 7.4.1 把程序存储器中的数据载入工作寄存器中

```
INC      DPTR                ; DPTR + 1
CLR      A                   ; ACC = 0
MOVCA,   @A+DPTR             ; 201H 上的数据载入 ACC
MOV      R1, A                ; R1 = ACC
INC      DPTR                ; DPTR + 1
CLR      A                   ; ACC = 0
MOVCA,   @A+DPTR             ; 202H 上的数据载入 ACC
MOV      R2, A                ; R2 = ACC
JMP      $                   ; 停机
```

## 7.4 实例点拨

### ——程序存储器和数据存储器的寻址

- 7.4.1 把程序存储器中的数据载入工作寄存器中

ORG 0200H; 数据表TABLE的起始地址定义为0200H

TABLE:

DB "CAT"; 数据表中的数据“CAT”

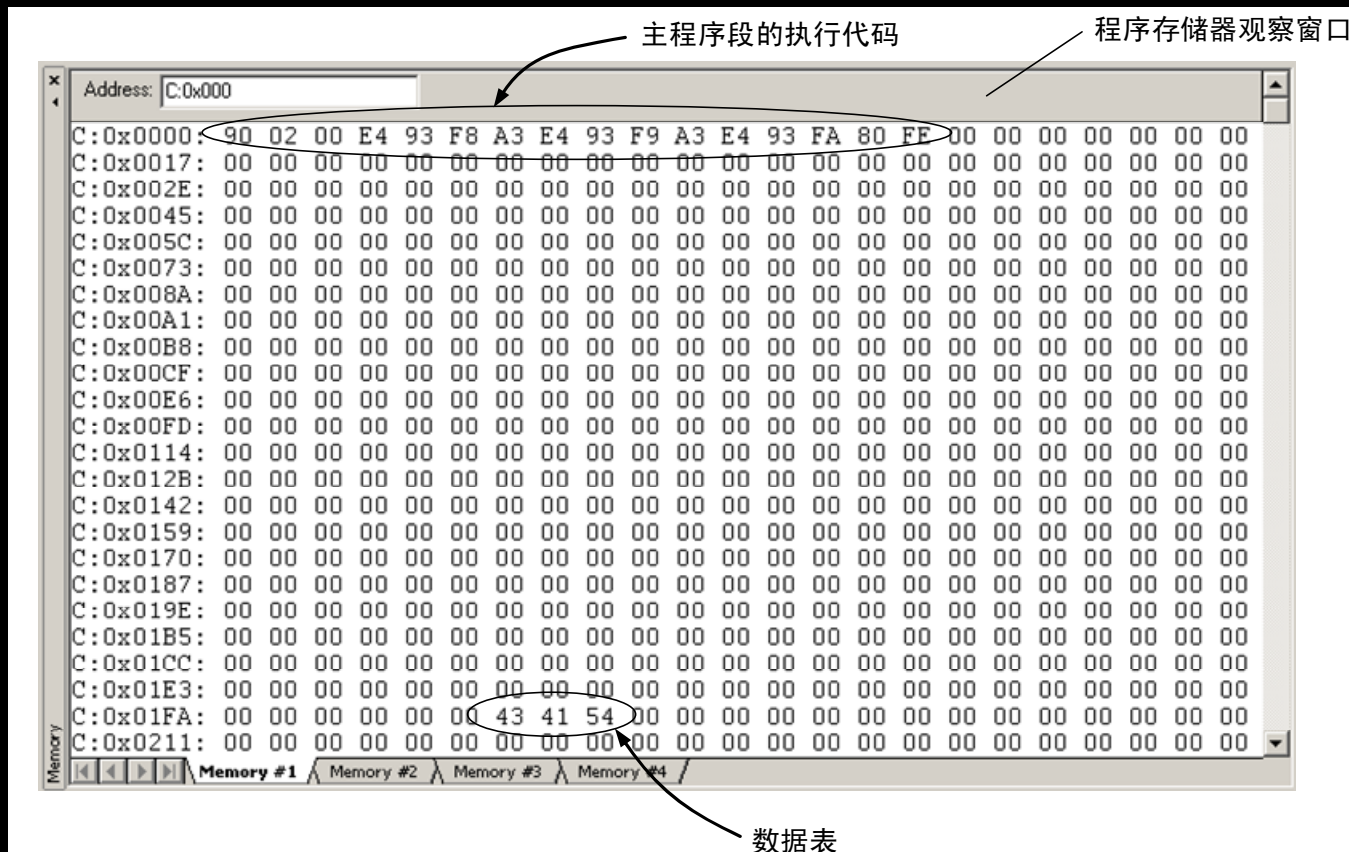
END

“MOV DPTR, #200H”用数据指针寄存器DPTR指向数据表TABLE的表头，也可以写成“MOV DPTR, #TABLE”

## 7.4 实例点拨

### ——程序存储器和数据存储器的寻址

#### • 7.4.1 把程序存储器中的数据载入工作寄存器中



伪指令定义数据表的起始地址

## 7.4 实例点拨

### ——程序存储器和数据存储器的寻址

#### • 7.4.2 把程序存储器中的数据载入数据存储器中

把程序存储器中数据表中的一个单词“CHINA”装载到数据存储器的30H~34H中。

```
ORG      000H                      ; 主程序的起始地址定义为 0000H

MOV      DPTR, #TABLE              ; DPTR 指向 TABLE
MOV      R0, #30H                  ; R0 = 30H

REPEAT:

CLR      A                        ; ACC 清 0
MOVCA, @A+DPTR                    ; 200H 上的数据载入 ACC
JZ       FINISH                   ; 如果 ACC = 0, 则跳到 FINISH
MOV      @R0, A                   ; ACC 的值载入 R0 所指的地址上
```

指令“MOV @R0, A”将ACC中的数据装到R0所指的数据存储器。

## 7.4 实例点拨

### ——程序存储器和数据存储器的寻址

#### • 7.4.2 把程序存储器中的数据载入数据存储器中

```
INC    DPTR                ; DPTR + 1
INC    R0                  ; R0 + 1, 即间接地址增加 1
JMP    REPEAT              ; 跳至 REPEAT

FINISH:
JMP    $                   ; 停机

ORG    250H                ; 定义数据表的起始地址为 250H

TABLE:
DB     "CHINA", 0          ; 数据表中的数据, "0" 为结束标志
```