第3章 MCS-51指令系统

主要内容：

- MCS-51指令系统概述
- 指令系统的寻址方式
- MCS-51指令系统

教学目的

- 了解MCS-51的寻址方式、指令系统
- 熟悉MCS-51的指令，掌握使用汇编语言进行程序设计。

3.1 MCS-51指令系统概述

3.1.1 指令概述

- **指令：**是规定计算机进行某种操作的命令
- **计算机指令系统：**一台计算机所能执行的指令集合
- **机器语言指令：**能被计算机直接识别、执行的二进制编码表示的指令
- **汇编语言指令：**以助记符表示的指令。助记符是用英文缩写来描述指令的功能。它便于记忆，也便于理解和分类。

3.1.1 指令概述

- AT89S51单片机使用MCS-51指令系统。
- MCS-51指令系统是复杂指令集（CISC），基本指令共有111条。
- 按执行时间来分，分三种：

（1）1个机器周期（12个时钟振荡周期）的指令64条；

（2）2个机器周期指令45条；

（3）4个机器周期——乘、除指令。

- 按所占字节分，分三种：

（1）单字节指令49条；

（2）双字节指令45条；

（3）三字节指令17条。

3.1.1 指令概述

- 按所占字节分，分三种：

（1）单字节指令，49条；

- 单字节指令：操作码和操作数同在一个字节中。

（2）双字节指令：45条；

- 一个字节为操作码，另一个字节是操作数。

（3）三字节指令：17条

- 操作码占一个字节，操作数占二个字节。

3.1.2 指令格式

- **指令格式：**指令的表示方法。
- 指令通常由两部分组成：**操作码和操作数。**
- **操作码**——指令进行什么操作。
- **操作数**——指令操作的对象。可能是一具体数据，也可能是指出到哪里取得数据的地址或符号。
- 指令长度不同，格式也就不同。

MCS51单片机汇编语言的指令格式如下

- **[标号：] <操作码> [操作数] [；注释]**

- **标号：** 指令的起始地址。由1—8个字符组成，第一个字符必须是字母，标号后跟分界符“：”
- **操作码：** 指令的助记符。
- **操作数：** 指令的操作对象。多个操作数之间用逗号“，”分隔。
- **注释：** 解释说明性的文字，用“；”开头，在本行“；”号后语句字符不再编译。

3.1.3 指令中常用符号说明

- Rn: 作寄存器组中的寄存器R0-R7之一；
- Ri: 地址指针的寄存器R0、R1；
- #data: 8位立即数；
- #data16: 16位立即数；
- direct: 内部RAM的8位地址；
- addr11: 11位目的地址；
- addr16: 16位目的地址；
- rel: 补码形式表示的8位地址偏移量；

3.1.3 指令中常用符号说明

- bit: 位寻址区或特殊功能寄存器的位地址;
- @: 间接寻址方式中间址寄存器的前缀标志;
- C: 进位标志位;
- /: 位操作数的前缀，表示对该位取反
- (x): 由x指定的寄存器或地址单元中的内容;
- ((x)): 由x寄存器的内容作为地址的存储单元的内容;
- $: 当前指令的地址;
- ←: 指令操作流程，将箭头右边的内容送到箭头左边的单元中。

3.2 MCS-51指令系统的寻址方式

- MCS-51的指令系统共使用了七种寻址方式：

1. 寄存器寻址
2. 直接寻址
3. 立即寻址
4. 寄存器间址
5. 基址寄存器加变址寄存器间址寻址方式
6. 相对寻址
7. 位寻址

1. 立即寻址

- 操作数直接出现在指令码中，即指令的操作数在指令中直接给出，**操作码**后面的就是实际的**操作数**（**立即数**）；
- 立即数前加“**#**”标志；
- 操作数可能是1字节，也可能是2字节。

例： MOV A ，#20H ；（A）←20H

功能：将20H这个立即数数送入累加器Acc中

例： MOV DPTR ，#2101H ；(DPH)←21H,(DPL)←01H

功能：将立即数2101H传送到指针寄存器DPTR中

2. 直接寻址

- 指令中直接给出操作数的单元地址，该单元地址中的内容就是操作数。
- 直接的操作数单元地址用“direct”表示。

例如：MOV　A，direct； “direct”就是操作数的单元地址。

具体：MOV　A，3AH ；源操作数3AH是片内RAM单元地址

功能：将片内RAM地址为3AH单元中的内容传送给累加器Acc。

例如：MOV　direct1，direct2

具体：MOV　42H，62H

功能：把片内RAM中62H单元的内容送到片内RAM中的42H单元中。

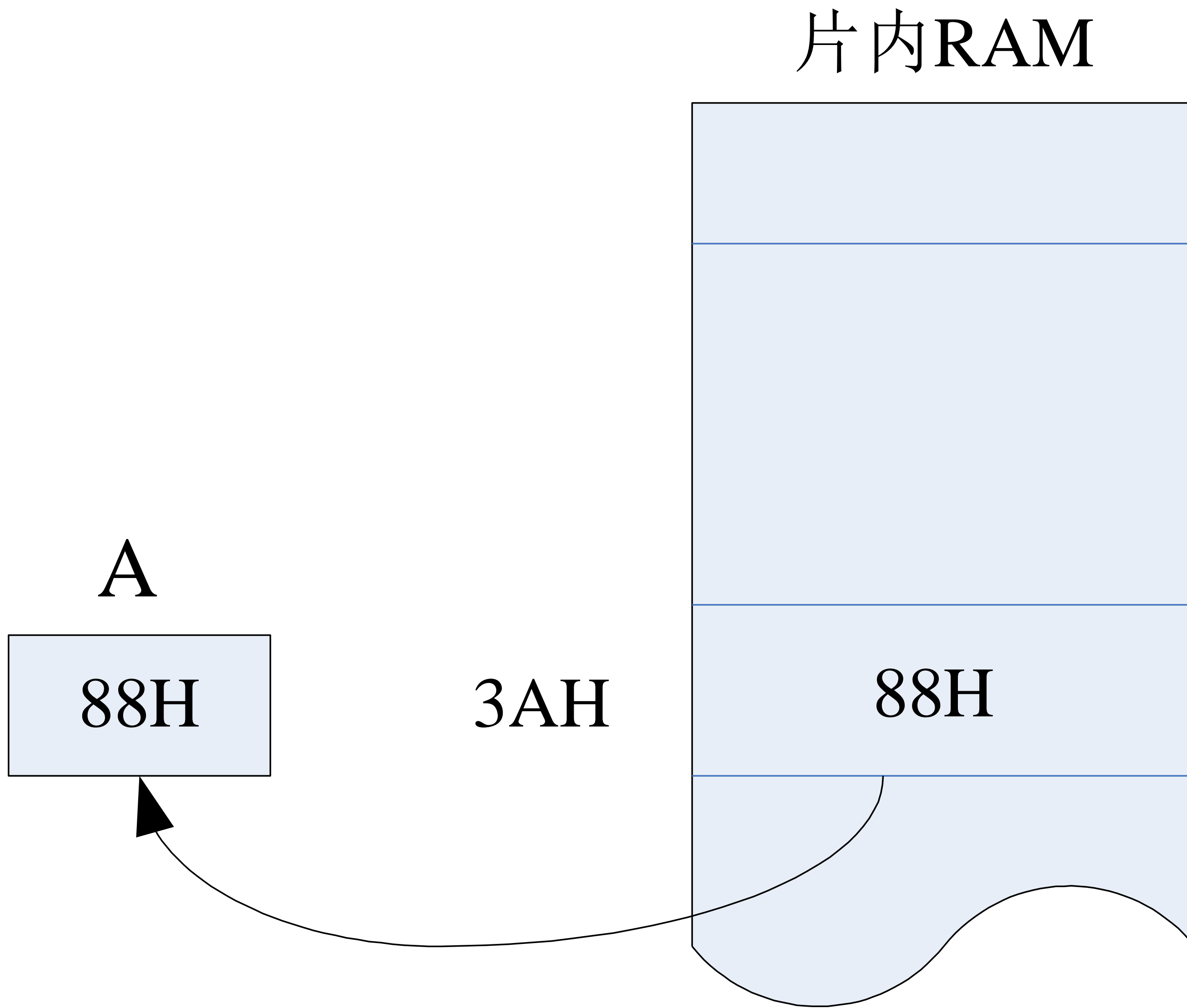


图3.1　直接寻址示意图

◆ 直接寻址方式可访问以下存储空间：

①**内部RAM低128个字节单元**，在指令中直接地址以单元地址的形式给出

②**特殊功能寄存器**

- 对于特殊功能寄存器，其直接地址还可以用特殊功能寄存器的符号名称来表示
- 访问特殊功能寄存器只能使用直接寻址方式

3. 寄存器寻址

◆ 在指令中指定寄存器，寄存器的内容作为操作数。

例如：MOV　A，Rn　　　；(Rn)→A，n=0～7

例：　MOV　A，R0　　　；（A）←（R0）

功能：将R0的内容传送到累加器A

例：　MOV　R2 ，A　　　；（R0）←（A）

功能：把累加器A中的内容传送到R2寄存器中。

◆ 能实现寄存器寻址方式的寄存器有：

- R0~R7、A、B寄存器和数据指针DPTR

4. 寄存器间接寻址

- 寄存器间接寻址就是以寄存器中的内容作为RAM地址，该地址中的内容才是操作数。
- 寄存器名称前必须加前缀标志“@”，表示寄存器间接寻址。

例如：MOV A，@Ri ；i=0或1

具体：MOV A，@R0 ；（A）←（（R0））

具体：MOVX A，@DPTR ；片外RAM((DPTR)）→A

如果DPTR＝3456H，片外RAM（3456H）＝99H

则（A）＝99H

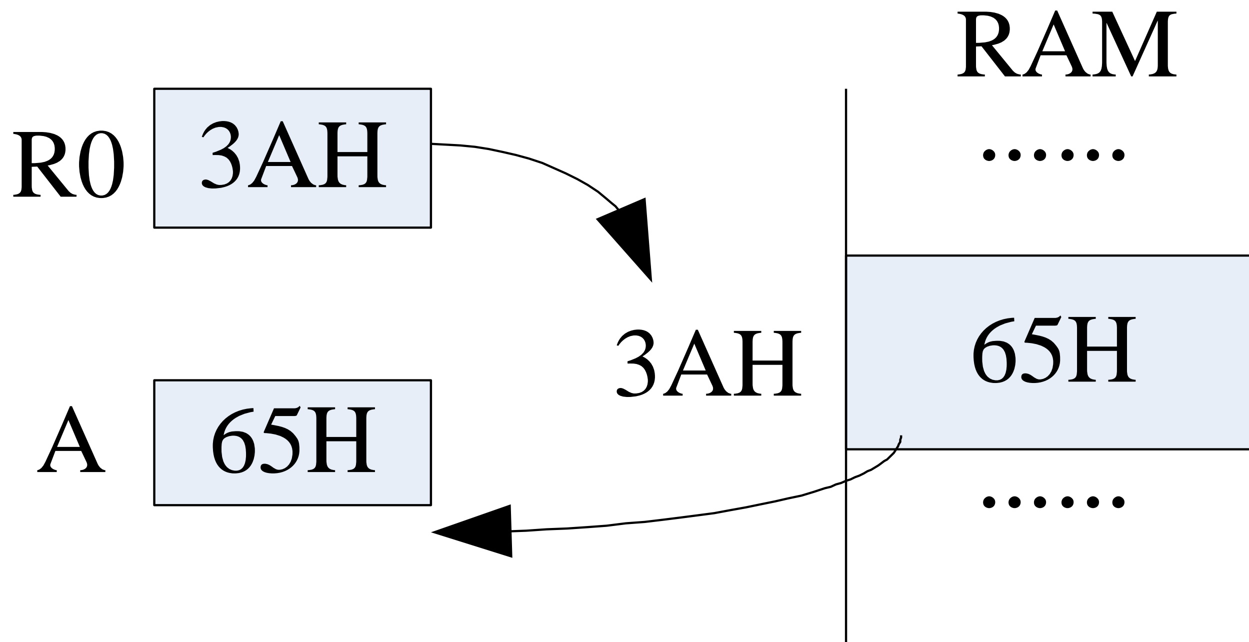


图3.2 寄存器简介寻址示意图

- R0寄存器的内容3AH是操作数地址
- 片内RAM的3AH单元的内容65H才是操作数，并把该操作数传送到累加器Acc，结果（A）＝65H。

■ 寄存器间接寻址的使用范围

- ◆ MSC51系列单片机规定只能用寄存器R0、R1、DPTR作为间接寻址的寄存器。
- ◆ 间接寻址可以访问的存储空间为片内RAM和片外RAM。
- · 片内RAM的低128个单元采用R0、R1作为间址寄存器，可寻址范围为00H～7FH单元。
- · **片外RAM的寄存器**间接寻址有**两种形式：**

① 采用R0、R1作为间址寄存器，可寻址范围为00H～FFH单元；

② 采用16位的DPTR作为间址寄存器，可寻址片外RAM的全部64KB地址空间。

5. 基址寄存器加变址寄存器间址寻址方式

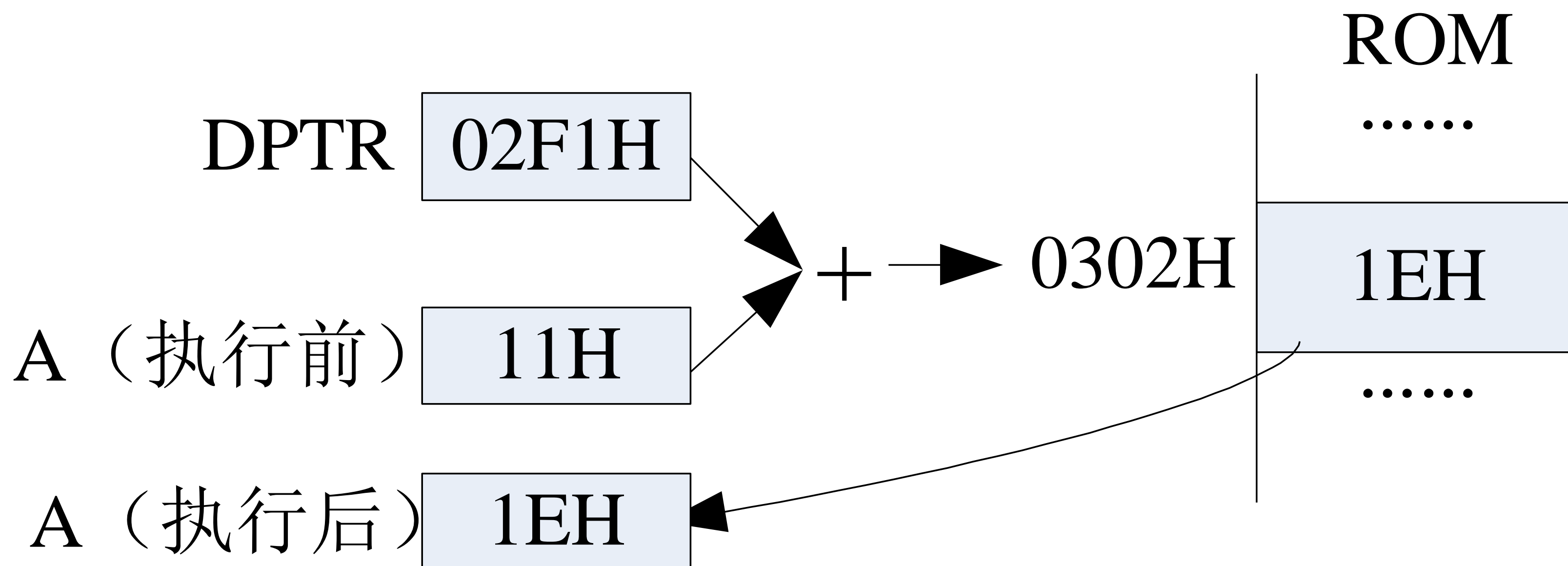
- 以DPTR或PC作为**基址寄存器**，以**累加器A**作为变址寄存器（**存放地址偏移量**），并以两者内容相加形成的16位地址作为操作数地址。
- 本寻址方式的指令有3条：

- MOVC A, @A+DPTR ；（A）←（（A）+DPTR）
- MOVC A, @A+PC ；（A）←（（A）+ PC）

※以上指令用于访问程序存储器中的数据表

- JMP A, @A+DPTR

※散转指令，A中内容为程序运行后的动态结果，可根据A中不同内容，实现跳向不同程序入口的跳转。



例：MOVC A，@A+DPTR

图3.3变址寻址示意图

6. 相对寻址

- 相对寻址方式用于转移指令，它是以该转移指令的地址（**PC值**）加上它的字节数（**程序计数器PC的当前值）**与指令中给出的**偏移量rel**相加，其结果形成新的转移目标地址送入**PC**中。
- 转移的目的地址用下式计算：
- **目的地址=转移指令所在的地址+转移指令字节数+rel**
- **rel**是一个**带符号的8位二进制数**，取值范围是-128~+127，故**rel**给出了相对于**PC当前值**的跳转范围。

6. 相对寻址

例如：SJMP　54H　　；“SJMP”是无条件相对转移指令，双字节。

说明：现假设此指令所在地址为2000H，执行此指令时，PC当前值为2000H+02H，则转移地址为2000H+02H+54H＝2056H。故指令执行后，PC=2056H，程序的执行发生了转移。

例如：LJMP　rel　　；LJMP是长程转移指令，3字节

说明：程序要转移到该指令的PC值加3再加上rel的目的地址处。编写程序时，只需在转移指令中直接写要转向的地址标号。

例如：LJMP　LOOP

说明：“LOOP”为目的地址标号。汇编时，由汇编程序自动计算和填入偏移量。但手工汇编时，偏移量的值由手工计算。

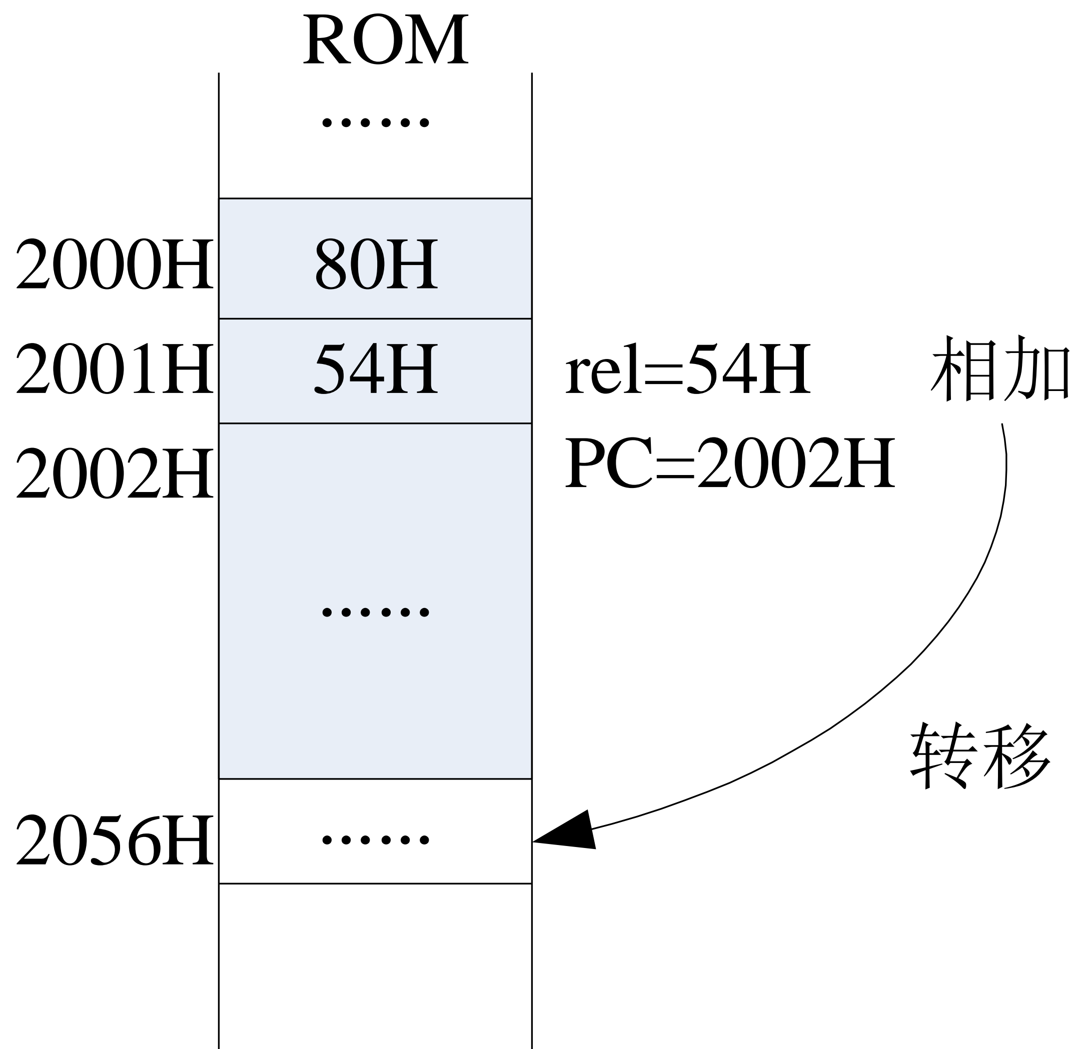


图3.4 相对寻址示意图

7. 位寻址

- 51系列单片机可对寻址的位单独进行操作，相应的在指令系统中有一类位操作指令，采用位寻址方式。该方式可对**内部RAM**和**特殊功能寄存器**具有位寻址功能的某位内容进行置1和清0操作。
- 位地址一般以直接位地址给出，位地址符号为“**bit**”。

例如：MOV　C，bit

具体：MOV　C，30H

功能：把位地址30H中的值传送到位累加器CY中

◆ 位寻址区

- 位寻址区20H~2FH单元的128位；
- 字节地址能被8整除的特殊功能寄存器相应位
- 位地址的表示方式：

① **直接使用位地址**。对于20H~2FH共16个单元的128位。其位地址是00H-7FH。

② **用单元地址加位序号**表示。如25H.5表示25H单元的D5位（位地址是2DH）。

③ **用位名称表示**。**特殊功能寄存器**中的可寻址位均有位名称，可以用位名称来表示该位，如可用RS0表示PSW中的0D0H.3。

④ **特殊功能寄存器**直接用寄存器符号加位序号表示。如PSW中的D3，又可表示为PSW.3。

表3-1　7种寻址方式及其寻址空间

序号	寻址方式	寻址空间
1	寄存器寻址	R0～R7、A、B、C(位)、DPTR等
2	直接寻址	内部128字节RAM、特殊功能寄存器
3	寄存器间接寻址	片内数据存储器、片外数据存储器
4	立即数寻址	程序存储器中的立即数
5	基址寄存器加变址寄存器间接寻址	读程序存储器固定数据和程序散转
6	相对寻址	程序存储器相对转移
7	位寻址	内部RAM中的可寻址位、SFR中的可寻址位

3.3 MCS-51指令系统

- AT89S51单片机使用MCS-51指令系统
- MCS-51指令系统共有111条指令，按功能划分为五大类：

- 数据传送类指令 29条
- 算术运算类指令 24条
- 逻辑运算及移位类指令 24条
- 控制转移类指令 17条
- 位操作类指令 17条

3.3.1 数据传送类指令

- 数据传送类使用最频繁，其功能是把源操作数传送到目的操作数;
- 指令执行后，源操作数不变，目的操作数被源操作数所代替。
- 主要用于数据的传送、保存及交换数据等场合。
- 本类指令不影响标志位:Cy、Ac和OV，但不包括奇偶标志位P。
- 一般数据传送类指令的助记符为“MOV”。指令格式为：

※ MOV <dest>,<src>

※ <src>为源操作数，<dest>为目的操作数

（1）片内RAM数据传送指令（16条）

- 内部RAM的数据传送类指令共16条，包括累加器、寄存器、特殊功能寄存器、RAM单元之间的相互数据传送。
- 指令助记符为：MOV

立即寻址方式格式

```
• MOV   A, #data          ; (A) ← #data
• MOV   Rn, #data         ; (Rn) ← #data
• MOV   @Ri, #data        ; ((Ri)) ← #data
• MOV   direct, #data     ; (direct) ← #data
```

※8位立即数可直接传送到片内RAM的各个位置，包括内部的80H-FFH单元。

```
• MOV   DPTR, #data16     ; DPTR← #data
```

※16位立即数的高8位送DPH，低8位送DPL。

片内RAM数据传送指令（立即寻址方式）举例

```
MOV    A, #80H
MOV    R0, #30H
MOV    R5, #0FAH
MOV    @R0, #70H
MOV    30H, #60H
```

```
MOV    DPTR, #1234H
```

等价于

```
MOV    DPH, #12H
MOV    DPL, #34H
```

□ 直接寻址方式形式

- MOV A, direct ；(A) ← (direct)
- MOV direct, A ；(direct) ← (A)
- MOV Rn, direct ；(Rn) ← (direct)
- MOV @Ri, direct ；((Ri)) ← (direct)
- MOV direct2, direct1 ；(direct2) ← (direct1)

※ 功能：将地址所指定片内RAM单元内容传送到累加器A、寄存器Rn和片内RAM单元。

例：

```
MOV  A, 80H
MOV  31H, A
MOV  R5, 70H
MOV  @R1, 71H
MOV  0E0H, 78H
MOV  @R1, P1
```

□ 间接寻址形式

- MOV @Ri , A ; ((Ri)) ← (A)
- MOV A, @Ri ; (A) ← ((Ri))
- MOV direct, @Ri ; (direct) ← ((Ri))

※ 通过址传送操作数到A和传送到直接地址

例：

- MOV @R0, A
- MOV A, @R0
- MOV 80H, @R1

□ 寄存器寻址

- MOV Rn，A   ； （Rn）← （A）
- MOV A，Rn   ； （A）← （Rn）
- MOV direct，Rn   ； （direct）← （Rn）
- MOV Rn，direct   ； （Rn）← （direct）

※ 工作寄存器的内容直接传送到累加器A、内部RAM的低128个单元及各个特殊功能寄存器。

例：

- MOV R4，A
- MOV A，R6
- MOV 80H，R3

※注:

- A是一个很重要的8位寄存器，可以与direct，@Ri，Rn任意交换数据;
- 立即数#data可以传给A，direct，Rn，@Ri;
- direct直接寻址可以与direct，@Ri， A， Rn任意交换数据;

※Rn与@Ri不能交换数据。

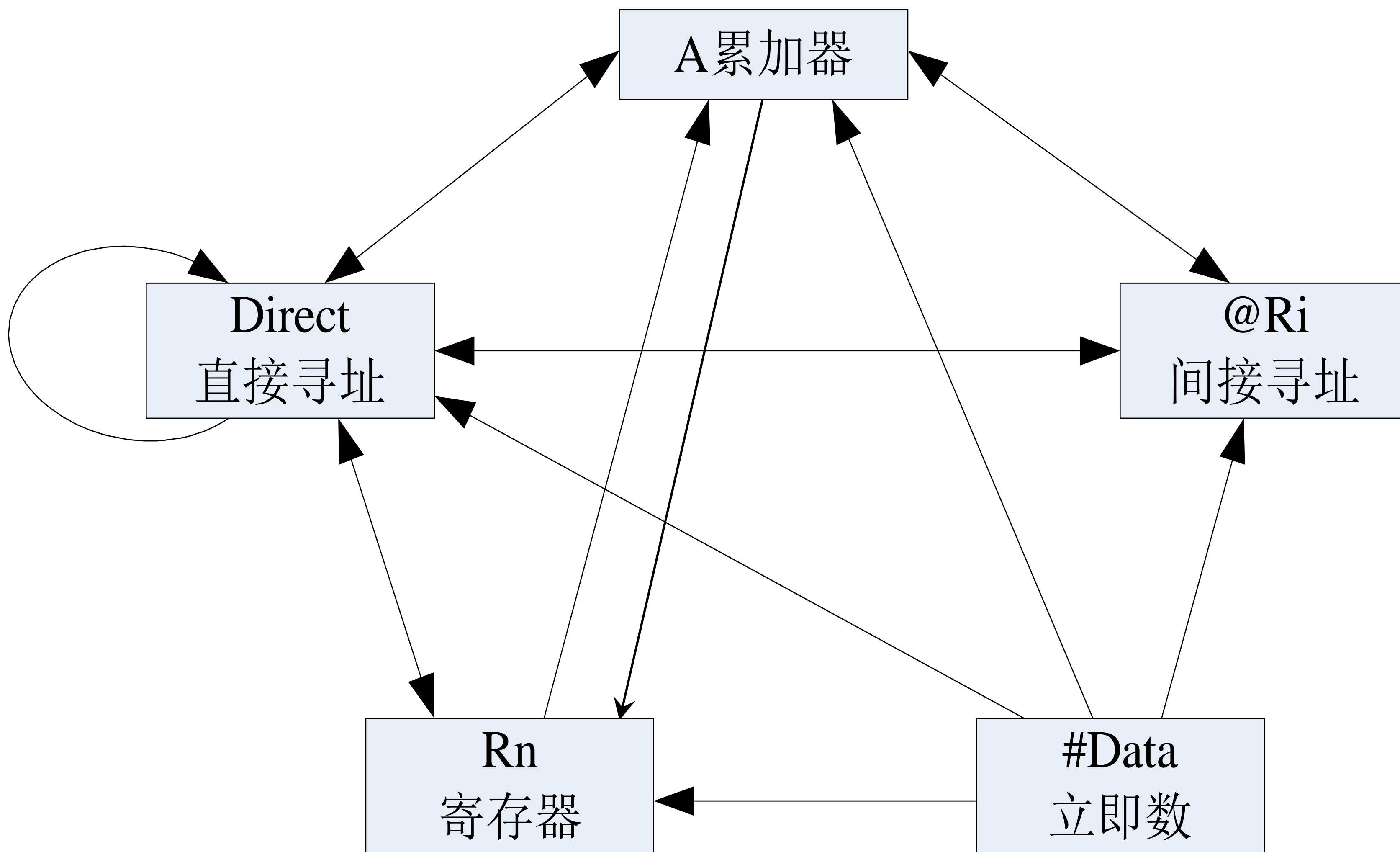


图3.5　数据传送指令传送规定示意图

※数据传送指令举例

例题：设（70H）＝60H，（60H）＝20H，P1为输入口，当前输入状态为0B7H，执行指令后，

- MOV R0，#70H；
- MOV A，@R0；
- MOV R1， A；
- MOV A，@R1；
- MOV @R0，P1；

- 运行结果：
- (70H) = 0B7H
- (60H) = 20H
- (A) = 20H
- (R0) = 70H
- (R1) = 60H
- P1 = 0B7H

（2）片外RAM的数据传送指令（4条）

- CPU与**片外RAM**或**I/O口**进行数据传送，必须采用寄存器间接寻址，并通过累加器A来传送。
- **片外RAM**的数据传送指令助记符为：**MOVX**；格式为

- MOVX A , @DPTR ； （A）←（（DPTR））
- MOVX @DPTR , A ； （（DPTR））←（A）

※ 这两条指令是以DPTR作为间址寄存器，其功能是将DPTR所指定的外部RAM单元与累加器A之间传送数据。

- MOVX A, @Ri ； （A）←（（Ri））
- MOVX @Ri, A ； （（Ri））←（A）

※ 这两条指令是以R0或R1作为间址寄存器，其功能是将R0或R1所指定的外部RAM单元与累加器A之间传送数据。

※ 指令说明

- **片外扩展的RAM**和**I/O口**统一编址，共同使用**64KB**。
- 因为**DPTR**是**16**位，所以用**DPTR**间接寻址可以访问**64KB任意单元**。
- 因为**Ri**是**8位寄存器**，只能寻**256B**空间，当**片外RAM小于256B**时，可直接采用这种方式；当**片外RAM大于256B**时，利用**P2口**输出**高8位地址（页地址）**作为一页，用**@Ri**进行**页内寻址**。

※ 片外RAM的数据传送指令举例

例题：向外部RAM的2007H单元写入数据60H

```
MOV     A, #60H
MOV     DPTR, #2007H
MOVX    @DPTR, A
```

或

```
MOV     A, #60H
MOV     P2, #20H
MOV     R0, #07H
MOVX    @R0, A
```

例题 试编程，将片外RAM的2000H单元内容送入片外RAM的0200H单元中。

解：片外RAM与片外RAM之间不能直接传送，需通过累加器A，另外，当片外RAM地址值大于FFH时，需用DPTR作间址寄存器。编程如下：

```
MOV     DPTR , #2000H      ; 源地址送DPTR
MOVX    A, @DPTR           ; 外部RAM中取数送A
MOV     DPTR, #0200H       ; 目的地址送DPTR
MOVX    @DPTR, A           ; A中内容送外部RAM
```

（3）程序存储器向累加器A传送数据指令

- MOVC A，@A+DPTR ；（A）←（（A）+（DPTR））
- MOVC A，@A+PC ；（A）←（（A）+（PC））

※ 该指令适合于查阅在ROM中建立的数据表格，故称做查表指令。

※ 前一条指令是采用DPTR作为基址寄存器。在使用前，表格首地址送入DPTR，实现在整个64KB ROM空间向累加器A的数据传送。即数据表格可以存放在64KB程序存储器的任意位置，称为远程查表指令。

※ 后一条指令是以PC作为基址寄存器。在程序中，执行该查表指令时PC值为下一条指令的地址，而不是表格首地址。在使用该查表指令之前，必须用一条加法指令进行地址调整，通过对累加器A的内容进行调整，使得A+PC和所读ROM单元地址保持一致。

※ 累加器A中的内容为8位无符号数，该指令只能查找指令所在地址以后256 B范围内的数据，称之为近程查表指令。

例：设（A）＝30H

执行　1000H：MOVC　A，@A+PC

因为该指令为单字节，故PC当前值为1001H，

1001H＋30H＝1031H

然后将程序存储器1031H单元的内容送入累加器A中。

例:　(DPTR)=0300H，（A）＝02H，

ROM中　(0302H)=55H

执行：MOVC　A,　@A+DPTR

结果：(A)=55H

例题： 编制根据A中数（0～9），查其平方表的程序。

平方表用**伪指令DB**存放在ROM中，把表的首地址置入**DPTR**中，把数0～9存放在变址寄存器A中。程序段如下：

```
        MOV     DPTR，#1000H
        MOVC    A，@A+DPTR
        ……
1000H： DB00H，01H，04H，09H，10H
        DB19H，24H，31H，40H，51H
```

例题：上例采用“MOVC A，@A+PC”

地址	源程序	
	ORG	0000H
0000H	ADD	A，#03H
0002H	MOVC	A，@A+PC
…		
0006H	DB	00H
0007H	DB	01H
0008H	DB	04H
0009H	DB	09H
000AH	DB	10H
000BH	DB	19H
000CH	DB	24H
000DH	DB	31H
000EH	DB	40H
000FH	DB	51H

（4） 数据交换指令

- 该指令可完成累加器和片内RAM单元之间的字节或半字节交换。
- 指令助记符为：XCH；整字节交换指令格式：

```
• XCH  A, Rn       ;  (A) ←→ (Rn)
• XCH  A, direct   ;  (A) ←→ (direct)
• XCH  A, @Ri      ;  (A) ←→ ((Ri))
```

例如：(A)=80H，(R7)=97H

执行：XCH A，R7

结果：(A)=97H，(R7)=80H

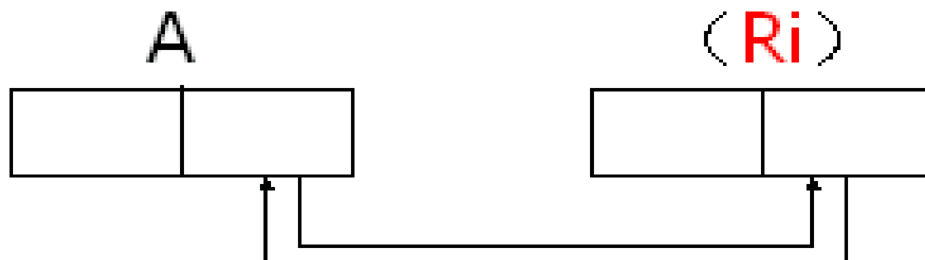
例：将片内RAM 60H单元与61H单元的数据交换。

```
        XCH  60H, 61H     ←对吗？
改正：  MOV  A, 60H
        XCH  A, 61H
        MOV  60H, A
```

■ 半字节交换指令格式

- XCHD A ，@Ri ；(A)$_{3\sim0}$ ←→ ((Ri))$_{3\sim0}$

※ （A）低4位与((Ri))低4位交换，高4位不变



■ 累加器高低半字节交换指令

- SWAP A ；(A)$_{7\sim4}$ ←→ （A）$_{3\sim0}$

例如：(R0)=60H， (60H)=3EH， (A)=59H

执行： XCHD A，@R0

结果： （A） = 5EH， （60H） = 39H

例如：(A)=3AH

执行：SWAP A

结果：(A)=0A3H

例题 试编程，将外部RAM1000H单元中的数据与内部RAM 6AH单元中的数据相互交换。

【解】数据交换指令只能完成累加器A和内部RAM单元之间的数据交换，要完成外部RAM与内部RAM之间的数据交换，需先把外部RAM中的数据取到A中，交换后送回到外部RAM中。编程如下：

```
MOV  DPTR , #1000H   ；外部RAM地址送DPTR
MOVX A, @DPTR        ；从外部RAM中取数送A
XCH  A, 6AH          ；A与6AH地址中的内容进行交换
MOVX @DPTR, A        ；交换结果送外部RAM
```

（5） 堆栈操作指令

- **进栈指令**助记符为“PUSH”，格式如下

• PUSH direct； （SP）← （SP）+ 1， （SP）←（direct）

※ 进栈指令功能是先将栈指针SP的内容加1，使它指向栈顶空单元，然后将直接地址direct单元或专用寄存器的内容送入栈顶空单元。

- **出栈指令**：助记符为“POP”

• POP direct； （direct） ←（SP），（SP）←（SP）- 1

※ 出栈指令功能是将SP所指的单元的内容送入直接地址所指出的单元或专用寄存器，然后将栈指针SP的内容减1，使之指向新的栈顶单元。

◆ 必须使用直接地址，只针对片内128BRAM或专用寄存器，所用的指针是SP

例：（SP）= 60H，（A）= 30H，（B）= 70H

执行：PUSH ACC ；(SP)←(SP)+1＝61H，((SP))←(ACC)

PUSH B ；(SP)←(SP)+1＝62H，((SP))←(B)

结果：（61H）＝ 30H，（62H）＝ 70H，（SP）＝ 62H

例：（SP）= 62H，（62H）= 70H （61H）= 30H

执行：POP DPH

POP DPL

结果：（DPTR）= 7030H，（SP）= 60H

3.3.2 算术运算类指令

- 80C51系列单片机的算术运算类指令共有24条
- 全部指令都是8位数运算指令；
- 算术运算类指令大多数要影响到程序状态字寄存器PSW中的标志CY,OV,AC,P；

※进位（借位）标志CY，可进行多字节无符号整数的加、减运算

※溢出标志OV可对带符号数进行补码运算

※辅助进位标志AC则用于BCD码运算的调整

※奇偶标志位P主要用于串行通信

（1）加法指令（4条）

■ **加法指令**共4条，指令助记符是“ADD”，格式如下：

- ADD A，# data ；（A）← （A）+data
- ADD A，direct ；（A）← （A）+（direct）
- ADD A，Rn ；（A）← （A）+（Rn）
- ADD A，@Ri ；（A）← （A）+（（Ri））

※ 功能是把源操作数所指出的内容与累加器A的内容相加，其结果存放在A中。

※ 源操作数的寻址方式分别为立即寻址、直接寻址、寄存器寻址和寄存器间接寻址。

■ 运算结果对程序状态位CY、AC、OV和P的影响：

- 进位标志CY：在加法运算中，如果D7位向上有进位，则CY＝1；否则，CY=0。
- 半进位标志AC：在加法运算中，如果D3位向上有进位，则AC＝1；否则，AC＝0。
- 溢出标志OV：在加法运算中，如果D7、D6位只有一个向上有进位时，OV＝1；如果D7、D6位同时有进位或同时无进位时，OV＝0；
- 奇偶标志P：当A中“1”的个数为奇数时，P＝1；为偶数时，P＝0。

例：设A =94H，（30H）=8DH，

执行指令 ADD A ，30H，操作如下：

```
          1001 0100
+         1000 1101
-------------------
1         0010 0001
```

结果：（A）= 21H，（CY）= 1，（AC）= 1，

（OV）=1，（P）=0

(2) 带进位加法指令

■ **带进位加法指令**共4条，指令助记符是“ADDC”，格式如下：

- ADDC A ，# data ；(A)← (A)+#data +(CY)
- ADDC A ，direct；(A)← (A)+(direct) +(CY)
- ADDC A ，Rn ； (A)←(A)+(Rn)+(CY)
- ADDC A ，@Ri ；(A)←(A)+((Ri)) +(CY)

※ 功能是把源操作数所指出的内容与累加器A的内容相加、再加上进位标志CY的值，其结果存放在A中。

※ 源操作数的寻址方式分别为立即寻址、直接寻址、寄存器寻址和寄存器间接寻址。

※ 运算结果对PSW标志位的影响与ADD指令相同。

例：设（A）=AEH，（R1）=81H，（CY）=1

执行指令 ADDC A ，R1，则操作如下：

```
      1010 1110
      1000 0001
+             1 ←（CY）
----------------
    1 0011 0000
```

结果（A）=30H，（CY）=1，（OV）=1，

（AC）=1，（P）=0

例题：设有两个无符号16位二进制数，分别存放在30H、31H单元和40H、41H单元中（低8位先存），写出两个16位数的加法程序，将和存入50H、51H单元。（设和不超过16位）。

【解】由于不存在16位数的加法指令，所以只能先加低8位，后加高8位，而在加高8位时要连低位相加的进位一起相加，编程如下：

```
MOV     A , 30H       ；取一个加数的低字节送A中
ADD     A , 40H       ；两个低字节数相加
MOV     50H , A       ；结果送50H单元
MOV     A , 31H       ；取一个加数的高字节送A中
ADDC    A , 41H       ；高字节数相加，同时加低字节产生的进位
MOV     51H , A       ；结果送51H单元
```

（3） 带借位减法指令

■ **带借位减法指令**有4条，指令助记符是“SUBB”，格式如下：

- SUBB A , # data ; (A) ← (A)-#data -(CY)
- SUBB A , direct ; (A)←(A)-(direct) -(CY)
- SUBB A , Rn ; (A)← (A)-(Rn)-(CY)
- SUBB A , @Ri ; (A) ← (A)-((Ri)) -(CY)

※ 功能是将累加器A中的数减去源操作数所指出的数和进位位CY，其结果存放在累加器A中。

※ 减法运算只有带借位减法指令，而没有不带借位的减法指令。

※ 若要进行不带借位的减法运算，应该先用指令将CY清零，然后再执行SUBB指令。

- 运算结果对PSW中各标志位的影响情况如下：

- 借位标志CY：如果D7位向上需借位，则CY=1；否则，CY=0。
- 半借位标志AC：如果D3位向上需借位，则AC＝1；否则，AC＝0。
- 溢出标志OV：如果D7、D6位只有一个向上有借位时，OV＝1；如果D7、D6位同时有借位或同时无借位时，OV＝0。
- 奇偶标志P：当A中“1”的个数为奇数时，P＝1；为偶数时，P=0。

（4） 加1指令

■ **加1指令**助记符是“INC”，指令共有5条，格式如下：

- INC A ；(A)←(A)+1
- INC @Ri ；((Ri))←((Ri))+1
- INC direct ；(direct)←(direct)+1
- INC Rn ；(Rn)←(Rn)+1
- INC DPTR ；(DPTR)←(DPTR)+1

※ 功能是将操作数所指定单元的内容加1。

※ 除“INC A”指令影响P标志外，其余指令均不影响标志位。

※ 加1指令常用来修改操作数的地址，以便于使用间接寻址方式。

（5） 减1指令

- **减1指令**共4条，指令助记符为“DEC”，格式如下：

- DEC A ; (A)←(A)－1
- DEC Rn ; (Rn)←(Rn)－1
- DEC direct ; (direct)←(direct)－1
- DEC @Ri ; ((Ri))←((Ri))－1

※ 功能是将操作数所指定单元的内容减1。

※ 除“DEC A”指令影响P标志外，其余指令均不影响PSW标志。

※ DPTR没有减1指令，只有加1指令。

（6） 乘除指令

- **乘法指令**助记符为“**MUL**”，格式如下
- 乘法： MUL AB
- 操作： (A)×(B) 低8位→A 高8位→B

※ 功能是把累加器A和寄存器B中的两个8位无符号数相乘，所得16位乘积的低8位放在A中，高8位放在B中。

※ (A)←被乘数，(B)←乘数；(A)←乘积低字节，(B)←乘积高字节

※ 上述数均为无符号数

※ 若乘积大于0FFH，则OV=1，否则OV=0。

※ CY总是清零

例：已知（A）＝ 080H，（B）＝ 32H，

执行:　　MUL AB

结果:　　（A）＝ 00H，（B）＝ 19H，OV＝1，

CY＝0，P＝0

■ **除法指令**助记符为“DIV”，格式如下：

- 除法：DIV AB
- 操作：(A)÷(B)，商→(A)，余数→(B)

※ 功能是将两个8位无符号数进行除法运算，其中被除数存放在累加器A中，除数存放在寄存器B中。指令执行后，商存于累加器A中，余数存于寄存器B中。

※ (A)←被除数，(B)←除数，(A)←商，(B)←余数

※ 以上数均为无符号数。

※ CY总是清零；若(B)＝0，则(OV)＝1，否则(OV)＝0

※ 本指令范围小（0～255），故用处不大

例：已知（A）= 87H，（B）= 0CH，

执行指令　　DIV AB

结果：（A）＝0BH，（B）＝03H，OV＝0，CY＝0，P=1

（7）十进制调整指令

- 本指令完成对BCD码加法结果做调整
- 指令助记符为“DA”，格式如下：
- DA　　A

※ 功能是两个BCD码的加法结果进行修正。

※ 该指令只影响进位标志CY。

- 该指令调整方法为：
- 若在加法过程中低4位向高4位有进位（即AC＝1）或累加器A中低4位大于9，则累加器A做加6调整；
- 若在加法过程中最高位有进位（即CY＝1）或累加器A中高4位大于9，则累加器A做加60H调整（即高4位做加6调整）。

例：求69D和59D的和。

先看作 69H＋59H，再调整

```
MOV  A, #69H
ADD  A, #59H
```

执行结果：　(A)=C2H, 并不是 128H，BCD码结果不正确。

作调整：　DA　A

调整结果：　(A)＝28H，CY=1

128H 为压缩BCD码，表示百位是1，十位为2，　个位为8

完成十进制数（BCD码）相加时，在ADD和ADDC后各加上一条“DA　A”即可

例题：试编写程序，实现95+59的BCD码加法、并将结果存入30H、31H单元。

编程如下：

```
MOV    A，#95H      ；95的BCD码数送A中
ADD    A，#59H
DA     A            ；对相加结果进行十进制调整
MOV    30H，A       ；A中的和存入30H
MOV    A，#00H      ；A清零
ADDC   A，#00H      ；加进位
DA     A            ；BCD码调整
MOV    31H，A       ；存进位
```

3.3.3 逻辑运算及移位类指令

- 逻辑运算的特点是按位进行。
- 逻辑运算及移位类指令共有24条。
- 逻辑运算包括与、或、异或三类，每类都有6条指令。
- 此外还有移位指令及对累加器A清零和求反指令。

（1）逻辑与运算指令

- 与操作指令有6条，指令助记符为“ANL”，格式如下：

- ANL A，#data
- ANL A，direct
- ANL A，Rn
- ANL A，@Ri
- ANL direct，A
- ANL direct，#data

※ 逻辑与运算指令常用于将某些位屏蔽（即使之为零）。

※ 方法是将要屏蔽的位和“0”相与，要保留的位同“1”相与。

（2）逻辑或运算指令

- 逻辑或指令有6条，指令助记符为“ORL”，格式如下：
- ORL A，#data
- ORL A，direct
- ORL A，Rn
- ORL A，@Ri
- ORL direct，A
- ORL direct，#data

※ 逻辑或运算指令常用于将某些位置位（即使之为1）。

※ 方法是将要置位的位和“1”相或，要保留的位同“0”相或。

例题：将累加器A的低4位送到P1门的低4位输出，而P1的高4位保持不变。

【解】这种操作不能简单地用MOV指令实现，而可以借助与、或逻辑运算。程序如下：

```
ANL    A , #0FH      ；屏蔽A的高4位，保留低4位
ANL    P1 , #0F0H    ；屏蔽P1的低4位，保留高4位
ORL    P1 , A        ；通过或运算，完成所需操作
```

（3）逻辑异或运算指令

- 逻辑异或运算指令有6条，指令助记符为“XRL”，格式如下：

- XRL A，#data
- XRL A，direct
- XRL A，Rn
- XRL A，@Ri
- XRL direct，A
- XRL direct，#data

※ 逻辑异或运算指令常用于将某些位取反。

※ 方法是将要求反的位同“1”相异或，要保留的位同“0”相异或。

（4） 累加器清零、取反指令

- 累加器清零指令1条，指令助记符为“CLR”，格式如下：
- CLR A
- 累加器按位取反指令1条，指令助记符为“CPL”，格式如下：
- CPL A

※ 80C51系列单片机只有对A的取反指令，没有求补指令。

※ 若要进行求补操作，可按“求反加1”来进行。

（5） 循环移位指令

- 80C51系列单片机的移位指令只能对累加器A进行移位
- 共有循环左移、循环右移、带进位的循环左移和右移4种：
- **循环左移：** RL A
- **循环右移：** RR A
- **带进位循环左移：** RLC A
- **带进位循环右移：** RRC A

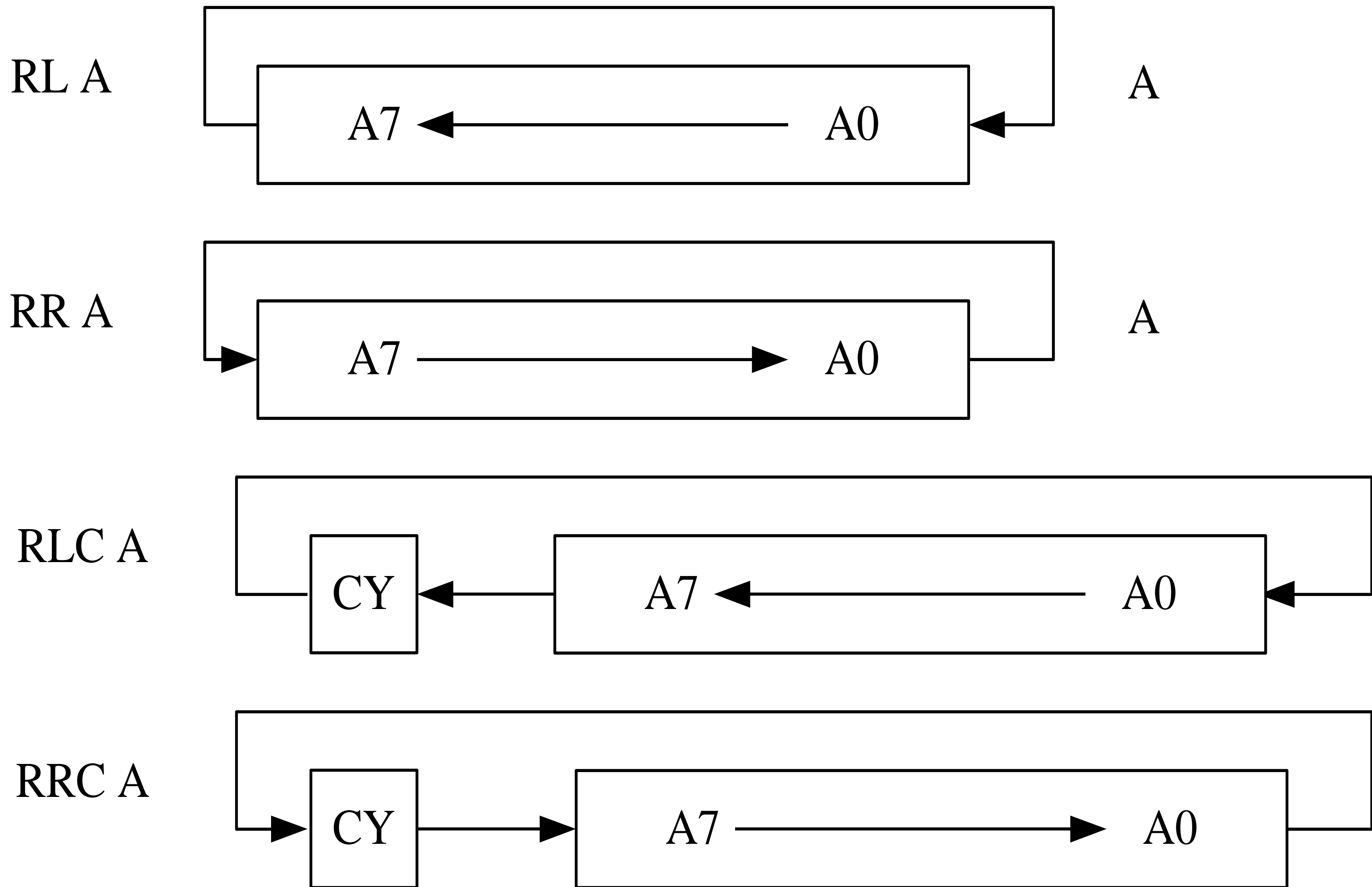


图3.6　循环移位指令图示

例题：设（A）＝08H，试分析下面程序执行结果。

（1） RL A ；A的内容左移一位，结果（A）=10H

RL A ；A的内容左移一位，结果（A）=20H

RL A ；A的内容左移一位，结果（A）=40H

- 即左移一位，相当于原数乘2（原数小于80H时）。

（2） RR A ；A的内容右移一位，结果（A）=04H

RR A ；A的内容右移一位，结果（A）=02H

RR A ；A的内容右移一位，结果（A）=01H

- 即右移一位，相当于原数除 2（原数为偶数时）。

例：（A）= 6CH =0110 1100B，（C） = 1

执行： RRC A

结果： (A)=1011 0110B=0B6H

例： （A）= 6CH =0110 1100B， （C） = 1

执行：RLC A

结果：(A)=1101 1001B=0D9H

3.3.4 控制转移类指令（17条）

- 通常情况下，程序的执行是按顺序进行的，这是由PC自动加1实现的，有时因任务要求，需要改变程序的执行顺序，这时就需要改变程序计数器PC中的内容，这种情况称做程序转移。
- 控制转移类指令都能改变程序计数器PC的内容。
- 控制转移类指令包括**无条件转移**指令，**条件转移指令**和**子程序调用**及**返回**指令，
- 这类指令一般不影响标志位。

（1）无条件转移指令（4条）

- 80C51系列单片机有4条无条件转移指令，提供了不同的转移范围和方式，可使程序无条件地转移到指令所提供的地址处。
- **长转移指令**：

• 指令助记符为“**LJMP**”

• 指令格式： LJMP　　addr16

※ 这是三字节指令，其功能是把指令中给出的**16位目的地址addr16**送入程序计数器**PC**，使程序**无条件转移到addr16处**。

※ **16位地址**可以寻址**64K**，该条指令可转移到**64KB**程序存储器的任何位置，故称为“**长转移**”。

■ 绝对转移（短跳转）指令

- 指令助记符是“AJMP”
- **指令格式：AJMP addr11**

※这是一条两字节指令，其指令代码格式为：

第一字节	a_{10}	a_9	a_8	0	0	0	0	1
第二字节	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0

※指令中提供了11位目的地址：

- 其中位a_7-a_0在第二字节，
- a_{10}-a_8则占据第一字节的高3位，
- 而00001是这条指令的操作码，占据第一字节的低5位。

※ **绝对转移指令的执行分为两步：**

① 取指令。此时PC加2（PC当前值），指向下一条指令的起始地址

② 用指令中给出的11位地址替换PC当前值的低11位，PC高5位保持不变，形成新的PC地址—即**转移的目的地址**。

＊ Addr11代表的寻址范围是$2^0 \sim 2^{11} = 2K$

＊ 转移可以向前也可以向后。

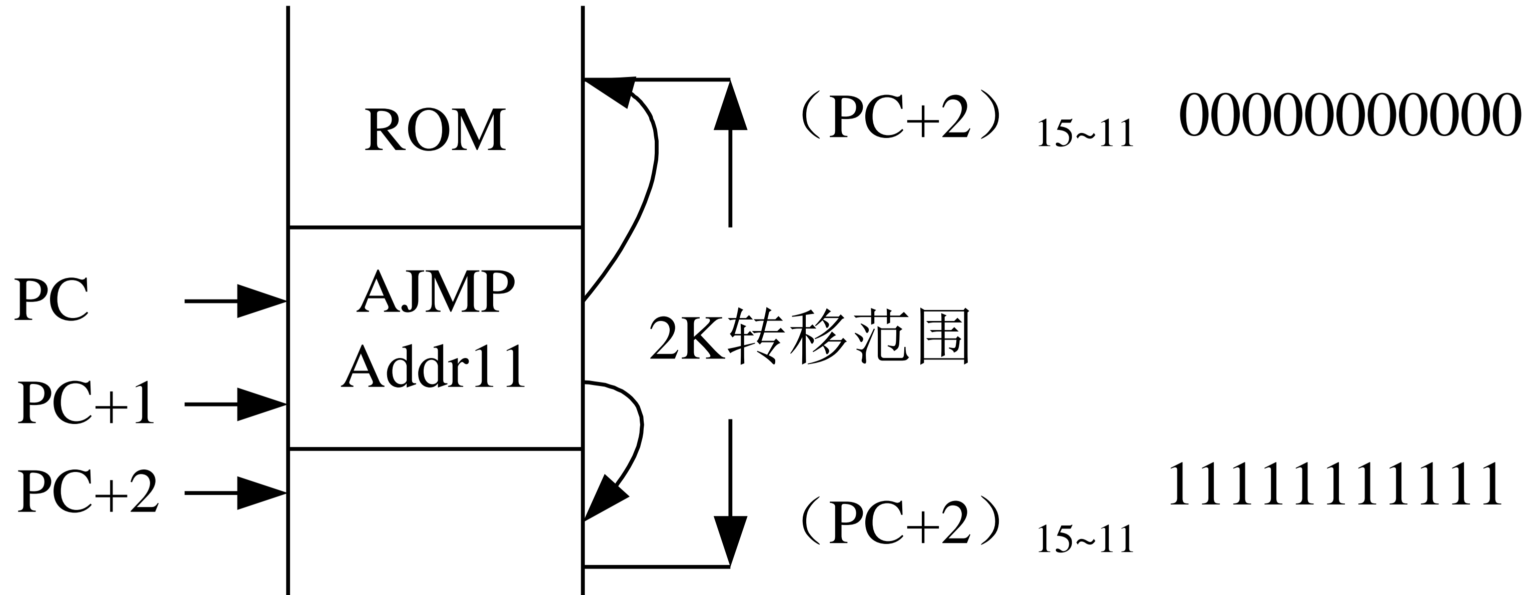
＊ **注意：转移到的位置是要与PC+2的地址在同一个2K区域，而不一定与AJMP指令的地址在同一个2K区域。**

* 64K的ROM分为32页，每页2K，范围分别是：

0000H~07FFH、 0800H~0FFFH、1000H~17FFH、⋯... F000H~F7FFH、F800H~FFFFH

* AJMP转移范围就是PC当前值所在的2K字节页面内，超出该寻址范围就出错。
* 该指令用于不太长的程序

AJMP指令转移范围图示



例：

```
07FEH      AJMP  K11        ；(PC)=(PC)+2=0800H

0800H      ........
..........
0E00H      K11：    ..........
.......
0F80H      K12：    ..........
.........
0FFEH      AJMP   K12       ；错误 (PC)=(PC)+2=1000H
                            ；K12不在1000H~17FFH页内
1000H      ..........
```

例题：分析下面绝对转移指令的执行情况。

1234H：AJMP 0781 H

分析：

- 在指令执行前，（PC）= 1234H；
- 取出该指令后，（PC）+ 2形成PC当前值，等于1236H；
- 指令执行过程就是用指令给出的11位地址11110000001B替换PC当前值的低11位。即：新的PC值为1781H，所以指令执行结果就是转移到1781 H处执行程序。

※ 应注意：只有转移的目的地址在2K范围之内时，才可使用AJMP指令。超出2K范围，应使用长转移指令LJMP。

■ 短转移（相对转移）指令

- 指令助记符是**“SJMP”**
- 指令格式：**SJMP rel**

※SJMP是无条件相对转移指令，rel是相对转移的偏移量。

※指令的执行分两步完成：

① **取指令。此时PC加2形成PC的当前值。**

② **将PC当前值与偏移量rel相加形成转移的目的地址。即：**

目的地址＝（PC）+2+rel

※rel是一个带符号的相对偏移量，其范围为：-128~+127，负数表示向后转移，正数表示向前转移。

例： 835AH SJMP 35H

目的地址：835AH+02H+35H=8391H

例： 835AH SJMP E7H ；E7H=－19H

目的地址：835AH+02H－19H=8343H

- 编程时，可用标号代替转移目的地址
- AJMP NEXT
- SJMP NEXT
- LJMP NEXT
- 原地踏步指令

HERE： SJMP HERE

常写成： SJMP $

■ 变址寻址转移指令

- 指令助记符是**“JMP”**
- 指令格式：JMP　　@A+DPTR

※ 指令功能是把累加器A中的8位无符号数与基址寄存器DPTR中的16位地址相加，其和作为目的地址送入PC。

※ 指令执行后不改变A和DPTR中的内容，也不影响任何标志位。

※ 指令的特点是转移地址可以在程序运行中加以改变。例如，在DPTR中装入多分支转移指令表的首地址，而由累加器A中的内容来动态选择该时刻应转向哪一条分支，实现由一条指令完成多分支转移的功能。

※ 该指令又称**散转指令**、**间接转移指令**。

例题： 设累加器A中存有用户从键盘输入的键值0~3，键处理程序分别存放在KPRG0、KPRG1、KPRG2、KPRG3处，试编写程序，根据用户输入的键值，转入相应的键处理程序。

编程如下：

```
MOV  DPTR ,  #JPTAB ；转移指令表首地址送入DPTR
RL   A       ；键值×2，因AJMP指令占2个字节
JMP  @A+DPTR   ；JPTAB十2倍键值，和送PC中，
     ；则程序就转移到表中某一位置执行指令。
JPTAB:
AJMP KPRG0
AJMP KPRG1
AJMP KPRG2
AJMP KPRG3
```

```
KPRG0：…
KPRG1：…
HPRG2：…
KPRG3：…
```

（2）条件转移指令

- 条件转移指令是指当某种条件满足时，转移才进行；而条件不满足，程序则顺序往下执行。
- 条件转移指令的共同特点是以下两点。

① 所有的条件转移指令都属于**相对转移指令**，转移范围相同，都在以PC当前值为基准的**256B**范围内（**-128～+127**）；

② 计算转移地址公式为：**转移地址＝PC当前值+rel**

■ 累加器到零转移指令（2条）

- **指令格式： JZ rel**

  JNZ rel

※以上指令以累加器A的内容是否为零作为是否判断转移的条件

※**JZ指令**功能是：累加器（A）＝0则转移；否则就按顺序执行；

※**JNZ指令**的操作正好与之相反。

■ 比较条件转移指令

- 比较条件转移指令共有4条，指令助记符是“CJNE”，其差别只在于操作数的寻址方式不同。

格式：CJNE　　A , #data, rel

操作：(A)≠#data　则转移，　(PC)←(PC)+3+rel

格式：CJNE　　A , direct, rel

操作：(A)≠(direct)　则转移，　(PC)←(PC)+3+rel

格式：CJNE　　Rn , #data, rel

操作：(Rn)≠#data　则转移，　(PC)←(PC)+3+rel

格式：CJNE　　@Ri , #data, rel

操作：((Ri))≠ #data　则转移，　(PC)←(PC)+3+rel

※指令功能：该组指令在执行时首先对两个规定的操作数进行比较，然后根据比较的结果来决定是否转移：

- 若左操作数＝右操作数，则程序顺序执行(PC)←(PC)+3，同时(CY)=0
- 若左操作数>右操作数，则转移(PC)←(PC)+3+rel，同时(CY)=0
- 若左操作数<右操作数，则转移(PC)←(PC)+3+rel，同时(CY)=1
- 为进一步的分支创造条件，通常在该组指令之后，选用以CY为条件的转移指令，则可以判别两个数的大小。

※ 在使用CJNE指令时应注意以下几点：

① 比较条件转移指令都是三字节指令，此PC当前值＝PC+3 （PC是该指令所在地址），转移的目的地址应是PC加3以后再加偏移量rel。

② 比较操作实际就是做减法操作，只是不保存减法所得到的差（即不改变两个操作数本身），而将结果反映在标志位CY上。

③ CJNE指令将参与比较的两个操作数当做无符号数看待、处理并影响CY标志。因此CJNE指令不能直接用于有符号数大小的比较。

④ 若进行两个有符号数大小的比较，则应依据符号位和CY位进行判别比较。

■ 减1条件转移指令（2条）

- 指令助记符是“DJNZ”
- 指令格式： DJNZ Rn , rel

  DJNZ direct , rel

※功能：先将操作数（Rn或direct）内容减1，并保存结果；

※如果减1以后操作数不为零，则进行转移；

※如果减1以后操作数为零，则程序按顺序执行。

例题： 试编写程序，将内部RAM从DATA为起始地址的10个单元中的数据求和，并将结果送入SUM单元。设和不大于255。

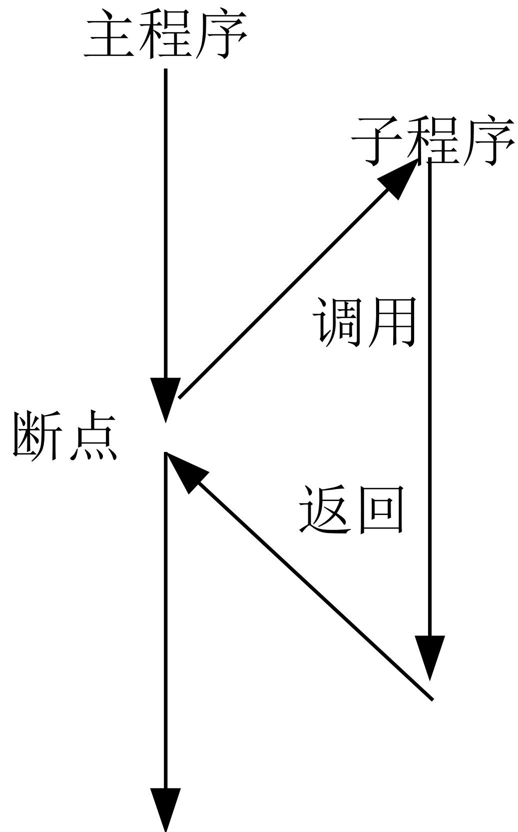
编程如下：

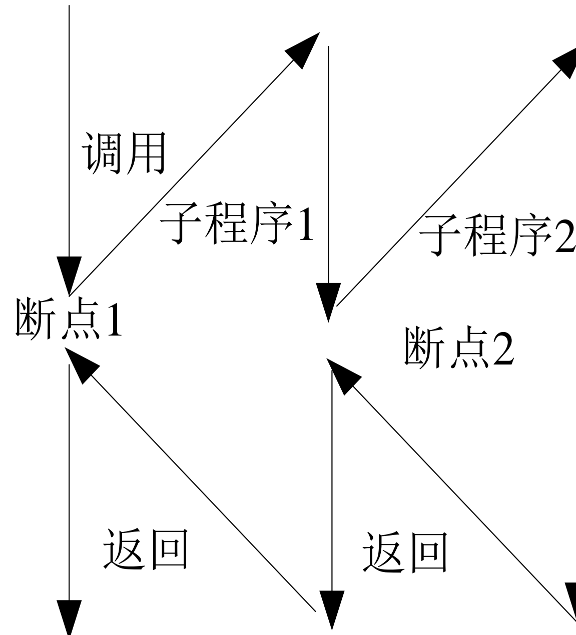
```
MOV   R0, #DATA        ；首地址送入间址寄存器R0
MOV   R7, #0AH         ；计数器R7送入计数初值
CLR   A                ；累加器A存放累加和，先清零
LOOP:
ADD   A, @R0           ；加一个数
INC   R0               ；地址加1，指向下一个单元
DJNZ  R7, LOOP         ；循环
MOV   SUM, A           ；累加和存入指定单元
SJMP  $                ；结束
```

（3）子程序调用及返回指令

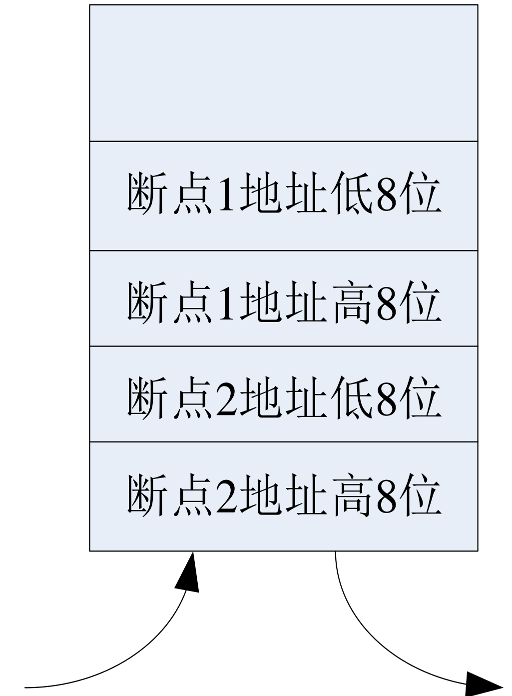
- 在程序设计中，常常出现几个地方都需要进行功能完全相同的处理，如果重复编写这样的程序段，会使程序变得冗长而杂乱。对此，可以采用子程序，即把具有一定功能的程序段编写成子程序，通过主程序调用来使用它，这样不但减少了编程工作量，而且也缩短了程序的长度。

如果在子程序中还调用其他子程序，称为子程序嵌套；





（a）子程序嵌套示意图



（b）堆栈操作示意图

子程序调用示意图

子程序嵌套及断点存放示意图

■ 子程序调用指令（2条）

① 长调用指令

- **格式：LCALL　addr16**
- **操作：(PC)←(PC)+3，(SP)←(SP)+1，((SP))←($PC_{7\sim0}$)**
  (SP)←(SP)+1，((SP))←($PC_{15\sim8}$)，(PC)←addr16

※**LCALL**指令称为**长调用指令**，指令的操作数部分给出了子程序的16位地址。

※该指令功能是：先将PC加3，指向下条指令地址（即断点地址），然后将断点地址压入堆栈，再把指令中的16位子程序入口地址装入PC。以使程序转到子程序入口处。

例：(SP)=60H，标号STRT值为0100H，标号DIR值为8100H。

```
STRT：LCALL  DIR
```

或

```
STRT：LCALL  8100H
```

结果：(SP)=62H；(61H)=03H；(62H)=01H；(PC)=8100H

② 绝对调用指令

- 格式：ACALL addr11
- 操作：(PC)←(PC)+2,(SP)←(SP)+1,((SP))←(PC7～0)

  (SP)←(SP)+1,((SP))←(PC15～8)，(PC)←addr11

※目的地址与当前PC值必须位于2K范围的同一页面内。

※编程时，可用标号代替转移目的地址，addr11，addr16由编译程序计算

例： LCALL DIR

ACALL DIR

※ ACALL指令为两字节，其代码格式为：

第一字节	a_{10}	a_9	a_8	1	0	0	0	1
第二字节	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0

※ 指令的操作数部分提供了子程序的低11位入口地址，其中a7-a0在第二字节、a10-a8则占据第一字节的高3位，而10001是这条指令特有的操作码，占据第一字节的低5位。

※ 绝对调用指令的功能是：先将PC加2，指向下条指令地址（即断点地址），然后将断点地址压入堆栈，再把指令中提供的子程序低11位入口地址装入PC的低11位上，PC的高5位保持不变。使程序转移到对应的子程序入口处。

■ 返回指令（2条）

- 格式：RET

  操作：$(PC_{15\sim8}) \leftarrow ((SP))$，$(SP) \leftarrow (SP)-1$；

  $(PC_{7\sim0}) \leftarrow ((SP))$，$(SP) \leftarrow (SP)-1$

- 格式：RETI

※RET指令被称为**子程序返回**指令，放在子程序的末尾。其功能是从堆栈中自动取出断点地址送入程序计数器PC，使程序返回到主程序断点处继续往下执行。

※RETI指令是**中断返回**指令，放在中断服务子程序的末尾。其功能也是从堆栈中自动取出断点地址送入程序计数器PC，使程序返回到主程序断点处继续往下执行。

例： 若 (SP)=62H，(62H)=07H；(61H)=30H

执行RET

则：(SP)=60H；(PC)=0730H

■ 空操作指令

- **格式：NOP**

※**单字节指令**，不产生任何操作，只是使PC的内容加1，然后继续执行下一条指令；

※它又是一条**单周期指令**，执行时在时间上**消耗一个机器周期**，因此，NOP指令常用来实现等待或延时。

3.3.5 位操作类指令

- 开关变量以位（bit）为单位来进行运算和操作的。
- 位操作类指令的操作对象：

① 内部RAM中的位寻址区，即20H~2FH中的128位（位地址00H～7FH）；

② 特殊功能寄存器中可以进行位寻址的各位。

- 位地址在指令中都用bit表示，bit有四种表示形式：

① 采用直接位地址表示；

② 采用字节地址加位序号表示；

③ 采用位名称表示；

④ 采用特殊功能寄存器加位序号表示。

- 进位标志CY在位操作指令中直接用C表示。

（1）位变量传送指令

- 格式：　MOV　C，bit　；　(CY)←(bit)

　　　　　MOV　bit，C　；　(bit)←(CY)

※ 功能：是在以bit表示的位和位累加器CY之间进行数据传送，不影响其他标志。

※ 两个可寻址位之间没有直接的传送指令，若要完成这种传送，需要通过CY来进行。

例：

```
MOV   C , 06H
MOV   P1.0 , C
```

（2）清零置位指令

- 格式：CLR C ；（CY）←0

  CLR bit ；（bit）←0

  SETB C ；（CY）←1

  SETB bit ；（bit）←1

※ 功能：对CY及可寻址位进行清零或置位操作，不影响其他标志。

例：

```
CLR   C
CLR   27H
SETB  P1.0
```

（3）位逻辑运算指令

- 与：ANL C , bit ； (CY)←(CY)∧(bit)

  ANL C , /bit ； (CY)←(CY)∧(/bit)
- 或：ORL C , bit ； (CY)←(CY)∨(bit)

  ORL C , /bit ； (CY)←(CY)∨(/bit)
- 非：CPL C ； (CY)←(/CY)

  CPL bit ； (bit)←(/bit)

※ 前4条指令的功能是将位累加器CY的内容与位地址中的内容进行逻辑与、或操作，结果送入CY中；

※ 后两条指令的功能是把位累加器CY或位地址中的内容取反。

例题： 设E，B，D都代表位地址，试编写程序完成E、B内容的异或操作，并将结果存入D中。

解：

```
MOV  C , B           ；从位地址中取数送CY
ANL  C , /E          ；（CY）←C∧/（E）
MOV  D , C           ；暂存
MOV  C , E           ；取另一个操作数
ANL  C , /B          ；（CY）←E∧/（C）
ORL  C , D           ；进行运算
MOV  D , C           ；操作结果存D位
```

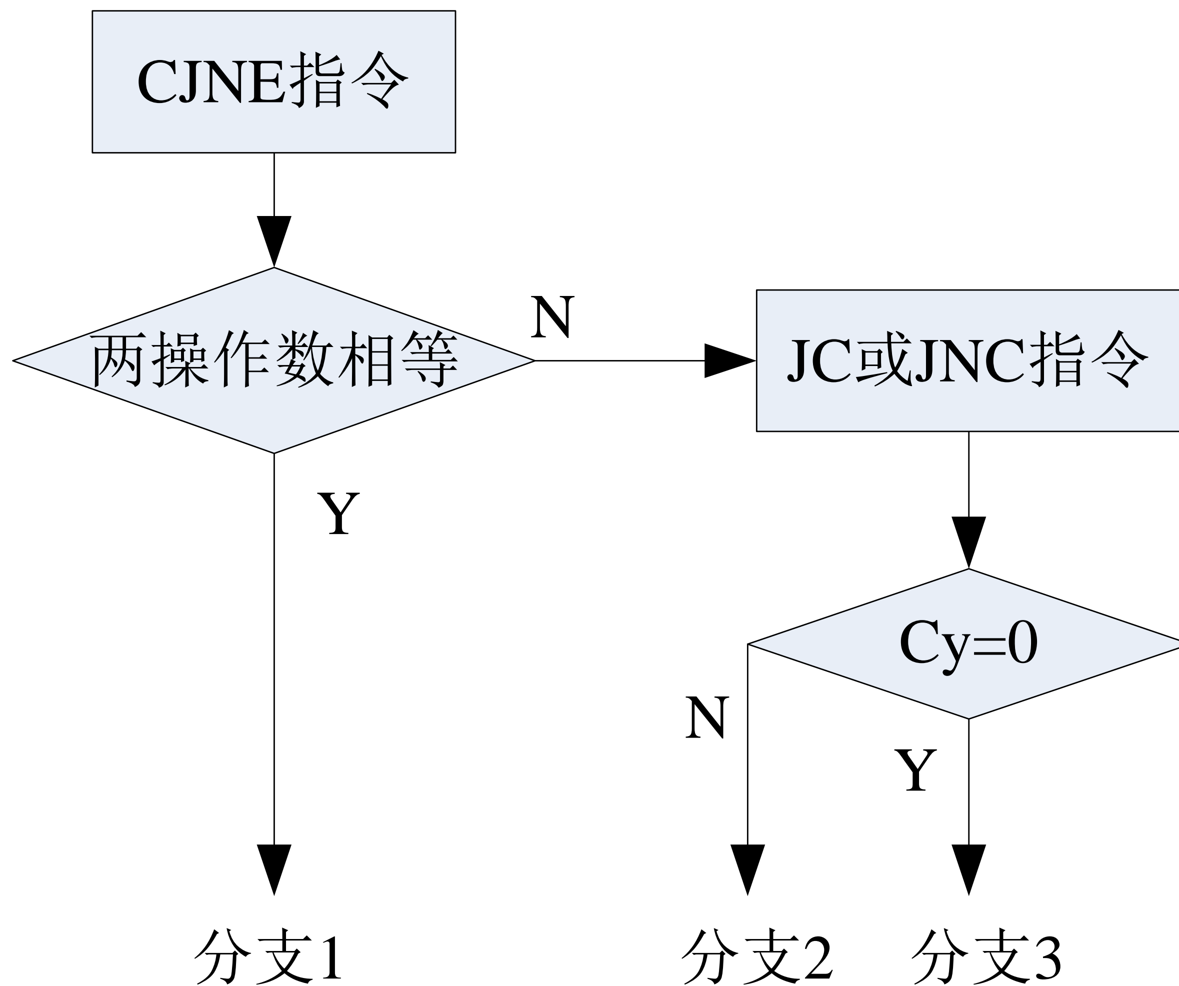
（4）位控制转移指令

- 位控制转移指令是条件转移指令，即以进位标志**CY**或地址**bit**的内容作为是否转移的条件。

以CY为条件的转移指令

- 格式：**JC rel**
- 操作：(CY)=1则转移，(PC)←(PC)+2+rel；否则，程序继续执行，PC)←(PC)+2
- 格式：**JNC rel**
- 操作：(CY)=0则转移，(PC)←(PC)+2+rel；否则，程序继续执行，(PC)←(PC)+2

※ 常和比较条件转称指令CJNE一起使用，先由CJNE指令判别两个操作数是否相等，若相等，就继续执行，若不相等，再根据CY中的值来决定两个操作数哪一个大，或者来决定如何进一步分支。从而形成三分支的控制模式。



CJNE和JC（或JNC）形成三个分支图示

例题：判别累加器A和30H单元内容的大小，若 A=（30H），转向LOOP1，若A>（30H），则转向LOOP2，若A<（30H），则转向LOOP3。设所存的都是补码数。

解：首先判断操作数的正负。可以将操作数和立即数80H相与，若结果为零，则为正数，否则，就为负数。然后再用CJNE指令和JC（JNC）指令形成三个分支。

```
MOV  R0 , A               ; 暂存
ANL  A , #80H             ; 判别A的正负
JNZ  NEG                  ; A<0则转至NEC
MOV  A , 30H
ANL  A , #80H             ; 判别（30H）的正负
JNZ  LOOP2                ; （30H）<0，A>（30H）
SJMP COMP                 ; （30H）>0，转向COMP
NEG:
MOV  A , 30H
ANL  A , #80H             ; 再次判别（30H）的正负
JZ   LOOP3                ; （30H）>0，A<（30H）
COMP:
MOV  A , R0               ; 取出原A值
CJNE A , 30H , NEXT       ; 比较A与（30H）
SJMP LOOP1                ; A=（30H）转LOOP1
NEXT:
JNC  LOOP2                ; A>（30H）转LOOP2
JC   LOOP3                ; A<（30H）转LOOP3
```

◆ 以位状态为条件的转移指令

- 格式：JB bit，rel
- 操作：(bit)=1则转移，(PC)←(PC)+3+rel；否则，程序继续执行，(PC)←(PC)+3
- 格式：JNB bit，rel
- 操作：(bit)=0则转移，(PC)←(PC)+3+rel；否则，程序继续执行，(PC)←(PC)+3
- 格式：JBC bit，rel
- 操作：(bit)=1则转移，(PC)←(PC)+3+rel，且(bit) ← 0；否则，程序继续执行，(PC)←(PC)+3

※注意，JB和JBC指令的区别：两者转移的条件相同，所不同的是JBC指令在转移的同时，还能将直接寻址位清零。

例题：从P1.1，P1.0输入开关量，高电平记为H(1)，低电平记为L(0)，当P1.1、P1.0状态为LL(00)、LH(01)、HL(10)、HH(11)，程序分别转到PRG0、PRG1、PRG2、PRG3处执行。程序段如下

```
        JB      P1.1 , K1
        JB      P1.0 , PRG1
        LJMP    PRG0
K1:     JB      P1.0 , PRG3
        LJMP    PRG2
```

3.3.6　某些指令的说明

1. I/O口的“读引脚”和“读锁存器”指令的区别

- P0口作输入口使用时，有两种读入方式：“读锁存器”和“读引脚”。
- 当CPU发出“读锁存器”指令时，锁存器的状态由Q端经上方的三态缓冲器BUF1进入内部总线；
- 当CPU发出“读引脚”指令时，锁存器的输出状态=1（即　端为0），而使下方场效应管截止，引脚的状态经下方的三态缓冲器BUF2进入内部总线。

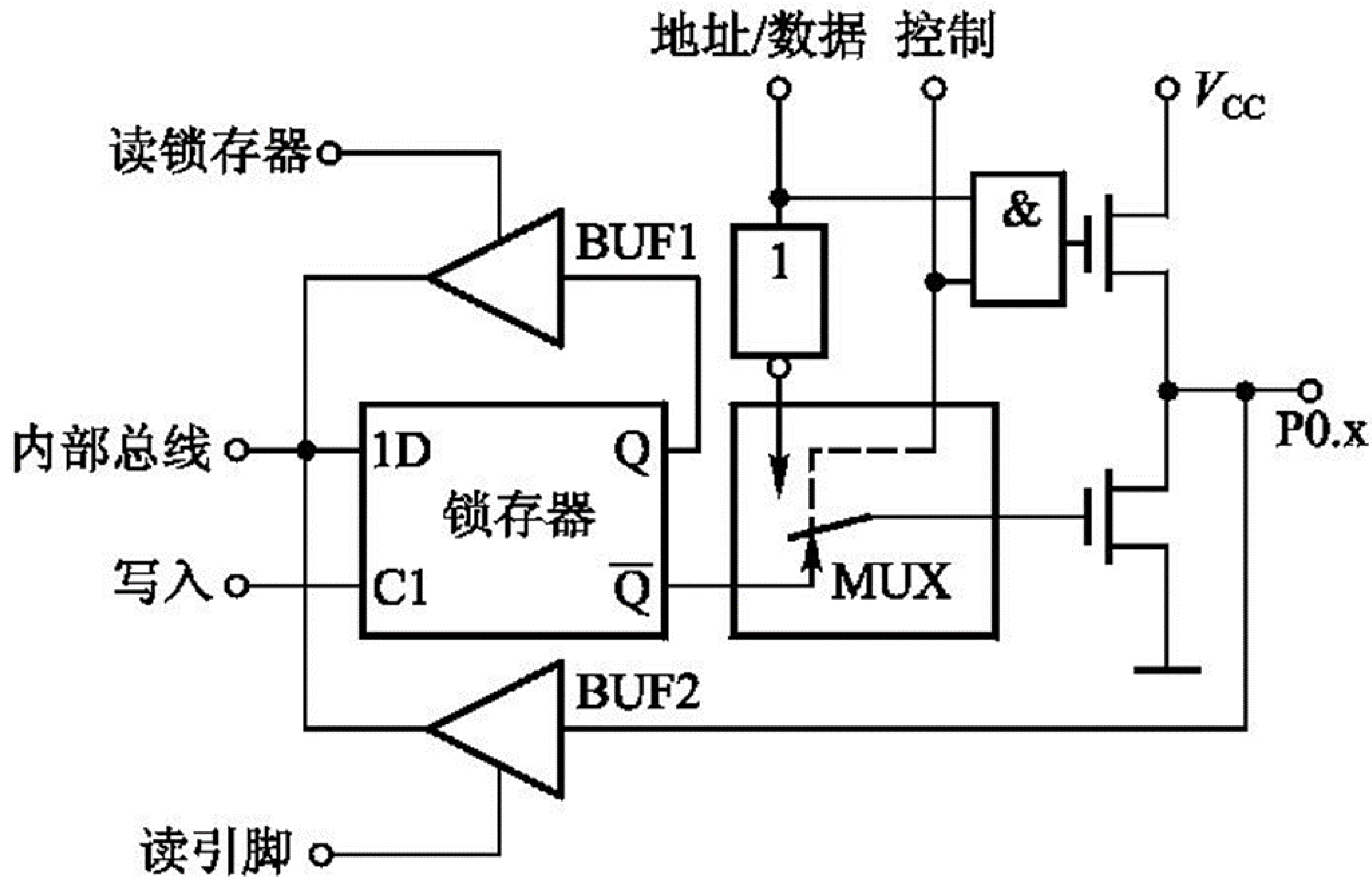


图 P0口某一位的位电路结构

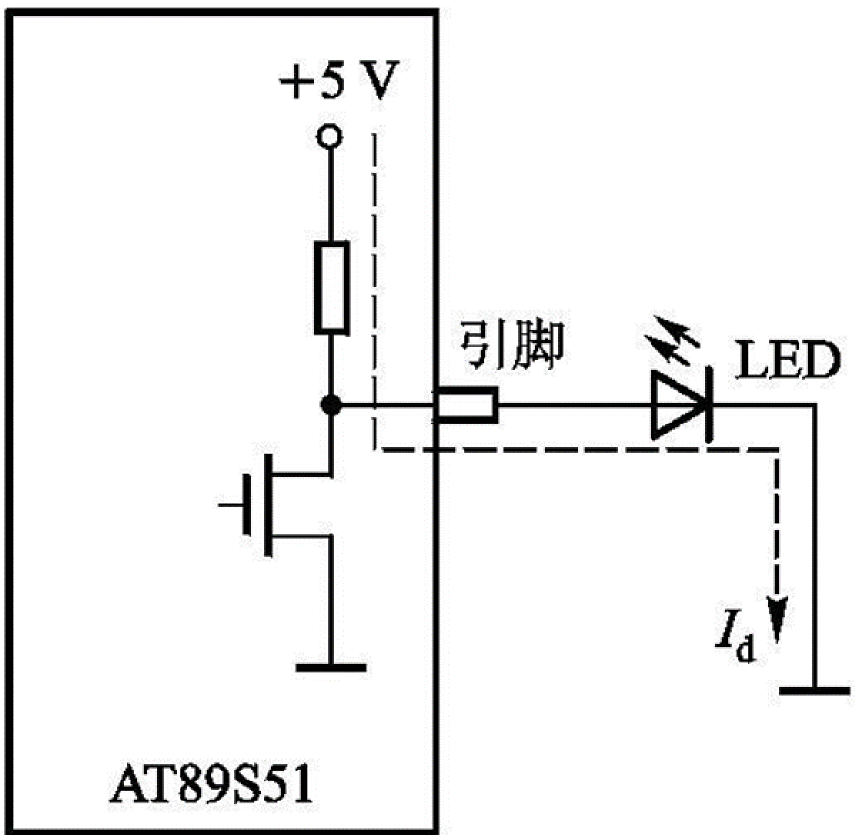
3.3.6 某些指令的说明

1. I/O口的“读引脚”和“读锁存器”指令的区别

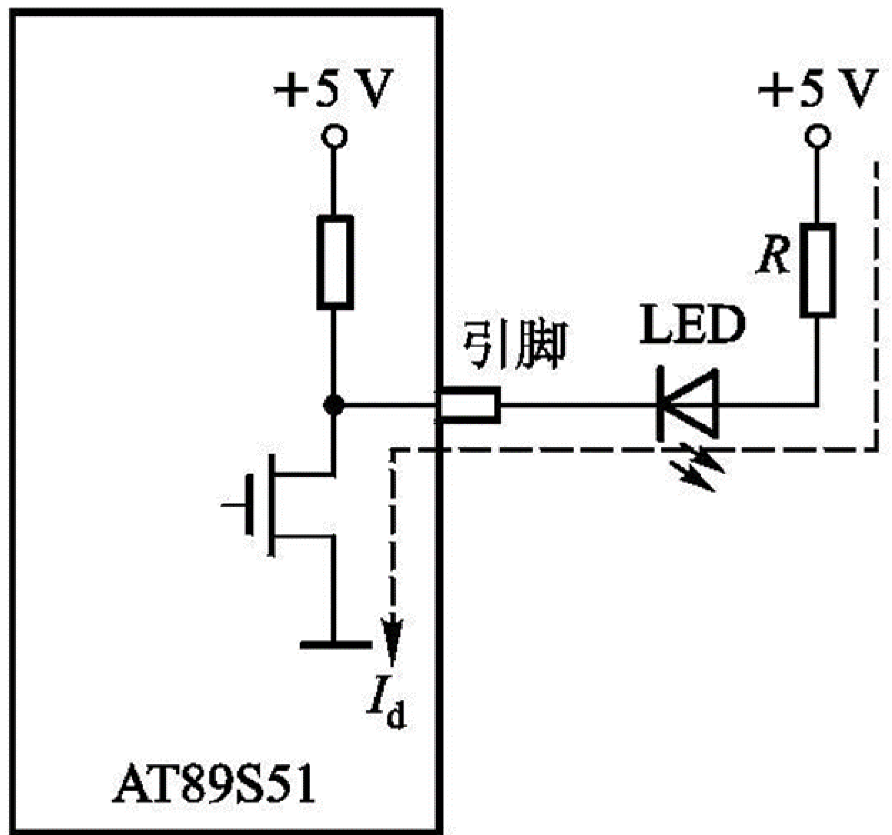
- 当P1口的P1.0引脚外接一个发光二极管LED的阳极，LED的阴极接地。若想查看一下单片机刚才向P1.0脚输出的信息是0还是1，不能直接从P1.0脚读取，因为单片机刚才向P1.0输出的信息如果是1的话，则LED导通点亮，此时P1.0引脚就为0电平，如果直接读引脚，结果显然错误。
- 正确的做法是读D锁存器的Q端状态，它储存的是前一时刻送给P1.0的真实值。
- 凡遇“读取P1口前一状态以便修改后再送出”的情形，都应当“读锁存器”的Q端信息，而不是读取引脚的信息。

1. 并行I/O口的“读引脚”和“读锁存器”指令的区别

- 当P1口外接输入设备时，要想P1口引脚上反映的是真实的输入信号，必须先让该引脚内部的场效应管截止, 否则当场效应管导通时，P1口引脚上将永远为低电平，无法正确反映外设的输入信号。
- 让场效应管截止，就是用指令给P1口的相应位送一个“ 1 ”电平，这就是为什么读引脚之前，一定要先送出 1 的原因。



+5 V
引脚
LED
I_d
AT89S51



+5 V
+5 V
R
引脚
LED
I_d
AT89S51

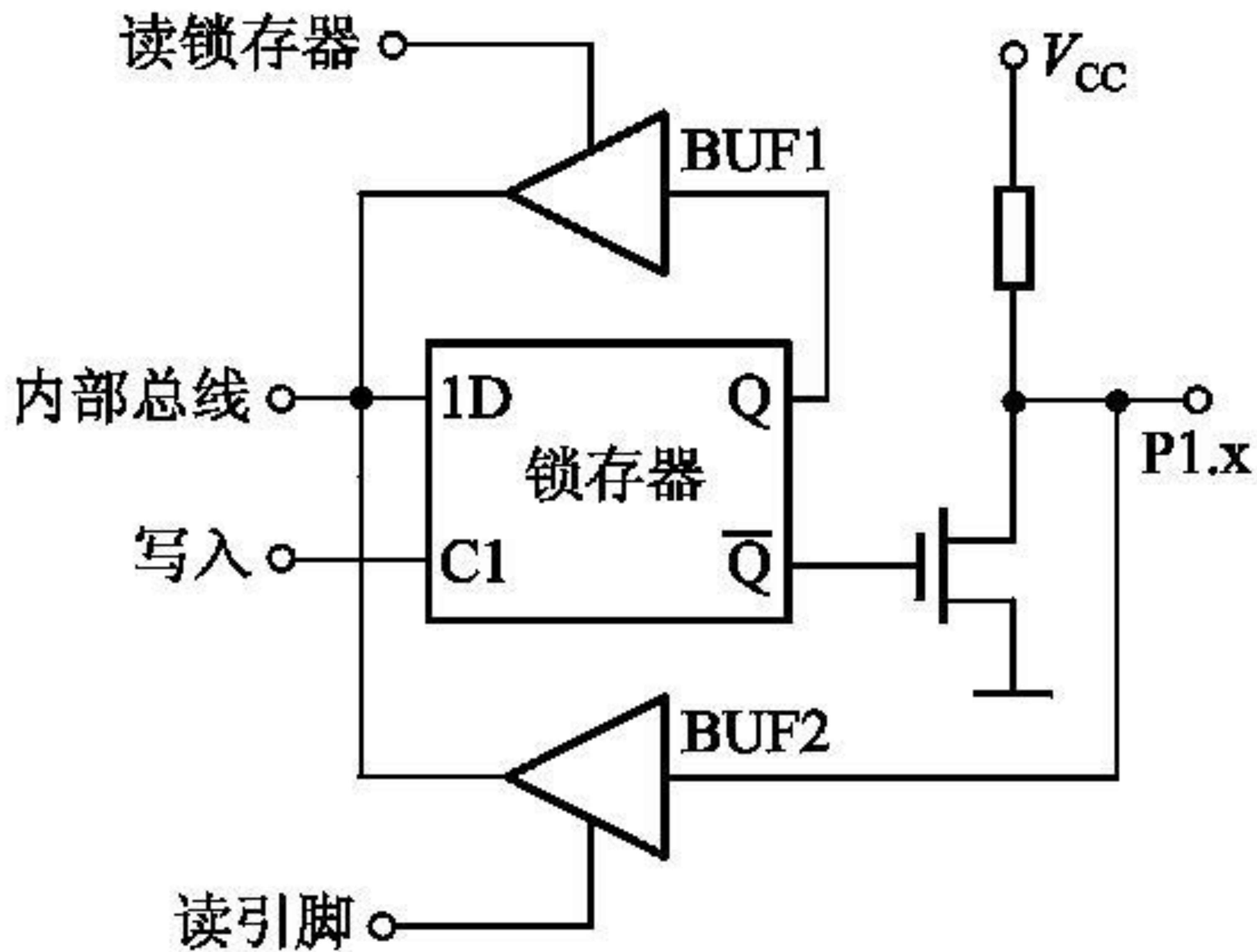


图　P1口某一位的位电路结构

※指令“MOVC，P1.0”读的是P1.0脚，同样，指令“MOV A，P1”也是读引脚指令。读引脚指令之前一定要有向P1.0写“1”的指令。

※指令“CPL P1.0”则是“读锁存器”，也即“读—修改—写”指令，它会先读P1.0的锁存器的Q端状态，接着取反，然后再送到P1.0引脚上。而指令“ANL P1，A”也是“读锁存器”命令。类似的“读—修改—写”

指令举例如下：

```
INC   P1
XRL   P3，A
ORL   P2，A
ANL   P1，A
CPL   P3.0
```

2. 关于操作数的字节地址和位地址的区分问题

- 如何区别指令中出现的字节变量和位变量?

- 例如指令“MOV C，40H”和指令“MOV A，40H”两条指令中源操作数“40H”都是以直接地址形式给出的，“40H” 是字节地址还是位地址？对于助记符相同指令，观察操作数就可看出。
- 显然前条指令中的“40H”肯定是位地址，因为目的操作数C是位变量。
- 后条指令的“40H” 是字节地址，因为目的操作数A是字节变量。

3. 关于累加器A与Acc的书写

- 累加器可写成A或Acc，区别是什么？

- Acc汇编后的机器码必有一个字节的操作数是**累加器的字节地址**E0H；A汇编后则隐含在指令操作码中。
- 例如："INC A"的机器码是04H。
- 写成"INC Acc"后，则成了"INC direct"的格式，对应机器码为"05H E0H"。

※在对累加器A直接寻址和累加器A的某一位寻址要用Acc，不能写成A。

例如，指令"POP Acc"不能写成"POP A"；

指令"SETB Acc.0"，不能写成"SETB A.0"

4. 书写2位十六进制数据前要加“0”

- 经常遇到必须在某些数据或地址的前面多填一个“前导”0，这是汇编语言的严格性和规范性的体现。
- 由于部分十六进制数是用字母来表示的，而程序内的标号也常用字母表示，为了将标号和数据区分开，汇编语言都规定，凡是以字母开头（对十六进制数而言，就是A～F开头）的数字量，应当在前面添加一个数字0。
- 至于地址量，它也是数据量的一种，前面也应该添加“0”。

例如:

```
MOV   A, #0F0H   ; “F0”以字母开头的数据量
MOV   A, 0F0H    ; “F0”以字母开头的地址量
```

- 如不加“前导”0，就会把字母开头的数据量当作标号来处理，从而出错以及不能通过汇编。

3.4 汇编语言程序的伪指令

- 对汇编过程进行说明和指导的一组命令。汇编时不产生目标程序代码，不影响程序的执行。

1. ORG（ORiGin）汇编起始地址伪指令

格式：[标号：] ORG 16位地址

功能：规定程序块或数据块存放的起始地址

```
例：      ORG    0030H
    ST：  MOV    A，R1
```

即规定标号ST代表地址为0030H开始。

※ 通常在源程序的起始处安排一条ORG指令来规定程序的起始存放地址，否则程序将从0000H开始。

※ 在一个程序中可多次用到ORG伪指令，但规定由小到大顺序排列，且不应使程序有交叉重叠。

例如：

```
ORG    2000H
……
ORG    2500H
……
ORG    3000H
……
```

这种顺序是正确的。

若按下面顺序的排列则是错误的，因为地址出现了交叉。

```
ORG    2500H
……
ORG    2000H
……
ORG    3000H
……
```

2. END(END of Assembly)汇编终止伪指令

格式：[标号：] END [表达式]

功能：结束汇编。

※ 是汇编语言源程序的结束标志

※ 汇编时遇到END就停止汇编，故该伪指令放在源程序结尾。

3. EQU（EQUate）赋值伪指令

格式： 符号名 EQU 表达式

或 符号名 EQU 寄存器名

功能：将一个数值或寄存器名赋给一个指定的符号名。

※ 表达式必须是一个是不带向前访问的表达式。

※ 用 EQU 指令赋值以后的字符名，可以用作数据地址、代码地址、位地址或者直接当做一个立即数使用。

例：　　TAL　EQU　6100H

　　　　MOV　DPTR，#TAL

等价于　MOV　DPTR，#6100H

例：　　JCC　EQU　R3

　　　　MOV　A，JCC

等价于　MOV　A，R3

例：　　S　EQU BX+SI

　　　　MOV　CX，[S]

那么上面两句代码相当于如下指令：

　　　　MOV CX，[BX+SI]

例：　ACCUM EQU A　　；定义ACCUM代替特殊汇编符号A（累加器）

HERE　EQU $　　；HERE为当前位置计数器的值

4. DB（Define Byte）字节数据定义伪指令

格式：［标号：］ DB 8位字节数据表

功能： 从标号指定的地址单元开始，将数据表中的字节数据按顺序依次存入。

```
例：      ORG   1000H
          DB    23H, 'ABC' , 25H
```

汇编后则从存储器1000H开始，依次存放：

23H，41H，42H，43H，25H

```
例：      ORG   2000H
          DB    30H，40H，24，"C"，"B"
```

汇编后： (2000H)=30H，(2001H)=40H，

(2002H)=18H(十进制数24)

(2003H)=43H(字符"C"的ASCII码)

(2004H)=42H(字符"B"的ASCII码)

5. DW（Define Word）字数据定义伪指令

格式：[标号：] DW 16位字数据表

功能：从标号指定的地址单元开始，将数据表中的字数据按从左到右的顺序依次存入。

※ 功能与DB相似，但DW定义的是16位数据

※ 汇编时DW按高字节在前（低地址单元），低字节在后（高地址单元）

※ 标号也可以，但必须事先赋值。

例：

```
        ORG     2800H
        DATA    EQU     2314H
OK:     DW      2100H，2150H
        DW      DATA，83H
```

汇编后，标号 OK即等于2800H，则从ROM的2800H地址开始，依次存入：

21H，00H，21H，50H，23H，14H，00H，83H

例：

```
        ORG     2000H
        DW      1246H，7BH，10
```

汇编后

```
(2000H)=12H          ；第1个字
(2001H)=46H
(2002H)=00H          ；第2个字
(2003H)=7BH
(2004H)=00H          ；第3个字
(2005H)=0AH
```

6. DS（Define Storage）定义存储区伪指令

格式：[标号：] DS 表达式

功能：从指定的地址单元开始保留由表达式指定的字节单元（不赋值）作为存储区，供程序运行使用。

```
例：  ORG   6000H
      DS    02H
      DB    88H
```

则6000H、6001H保留，即预留2个内存单元，而6002H放入88H

```
例：  ORG   2000H
      DS    10 H
```

表示从2000H地址开始，保留16个连续地址单元。

※ **注意：DB、DW、DS指令只能对程序存储器使用，而不能对数据存储器使用**

7. BIT位地址定义伪指令

格式：<字符名称> BIT <位地址>

功能：用于给字符名称赋以位地址，位地址可以是绝对位地址，也可是符号地址。

例：

```
A1    BIT    P1.0
A2    BIT    P1.5
```

功能是把P1.0的位地址赋给变量A1，把P1.5的位地址赋给变量A2。

伪操作指令例子

```
MATH    EQU  03H
        ORG  0030H
MAIN:   CLR  C
LOOP:   MOV  A , @RO
        MOV  R1, #MATH ；相当于 MOV  R1, #03H
NEXT:   SJMP $
        ORG  1100H
        DB   01H , 04H,  09H , 05H
        END
```

注：ROM中

地址	数据
1100H	01H
1101H	04H
1102H	09H
1103H	05H