



第三章 单片机指令系统

信息科学与工程学院自动化系

本章是全书的重点之一， 是汇编语言程序设计的基础。

学习指令系统重点要掌握以下几点：

- 1、指令功能
- 2、寻址方式
- 3、操作数的位数和存储结构
- 4、对PSW的影响
- 5、指令与存储区的对应关系
- 6、各类指针及其指向的地址范围
- 7、转移指令的转移范围
- 8、指令字节数和机器周期数



- Ø 本章主要介绍单片机的寻址方式及指令系统，
是必须掌握的内容。
- Ø 一台计算机所有指令的集合，
称为该计算机的指令系统。
- Ø 各种计算机都有专用的指令系统。

第三章 单片机的指令系统

- 3.1 指令系统概述
- 3.2 寻址方式
- 3.3 MCS-51 单片机的指令系统



3.1 指令系统概述

3.1.1 指令格式

3.1.2 指令字长和指令周期

3.1.3 指令分类



✓ 机器语言：

二进制数表示的机器指令，能被计算机直接识别并执行，但是不便记忆和理解，书写易出错；

✓ 汇编语言：

用助记符来表示机器指令，与机器码一一对应，易于掌握和使用，比高级语言的程序结构紧凑，节省存储空间，执行效率高。

✓ 高级语言

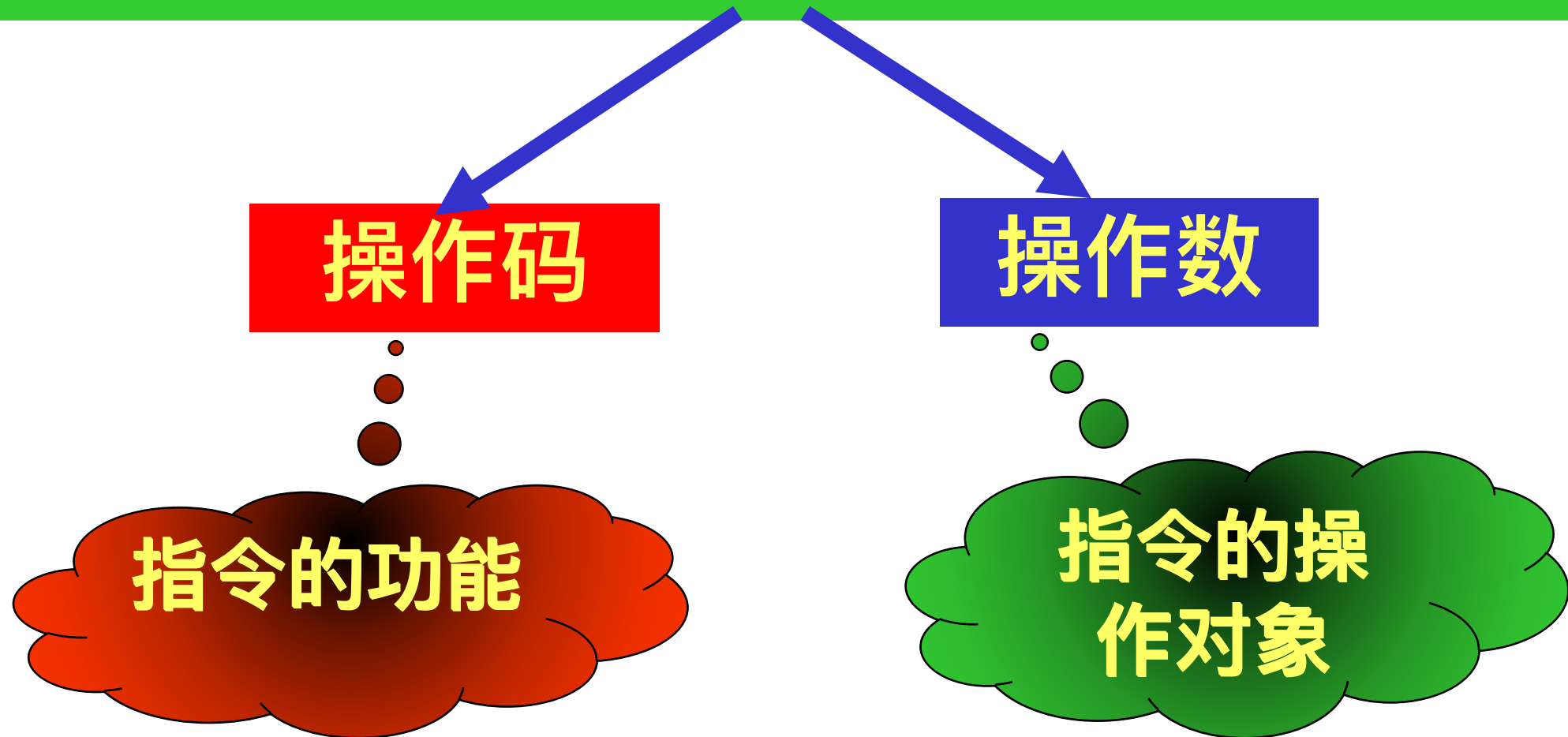
3.1.1 指令格式

—汇编语言指令格式
—机器语言指令格式



汇编语言指令

一条用助记符表示的汇编语言指令



ADD A , #10H ; ADD为操作码,

A及#10H为操作数

汇编语言指令格式

[标号]: 操作码助记符 [目的操作数], [源操作数]; [注释]

例如:

Loop: MOV A, R0; (R0) --->(A)

INC A;

RETI ; 中断返回指令, 无操作数



机器语言指令格式

单字节：

操作码

或

操作码	操作数或寻址方式
-----	----------

双字节：

操作码

操作数或寻址方式

三字节：

操作码

操作数或寻址方式

操作数或寻址方式

例： **ADD A , #10H**

机器码

0 0 1 0 0 1 0 0
0 0 0 1 0 0 0 0

操作码 24H

操作数 10H



3.1.2 指令字长和指令周期

51单片机采用变字长存储机器指令的方式

指令字长有三种：

单字节

RET

双字节

MOV A, #68H

三字节

MOV 30H, 46H

指令周期是指执行一条指令所需要的时间

1 机器周期指令

2 机器周期指令

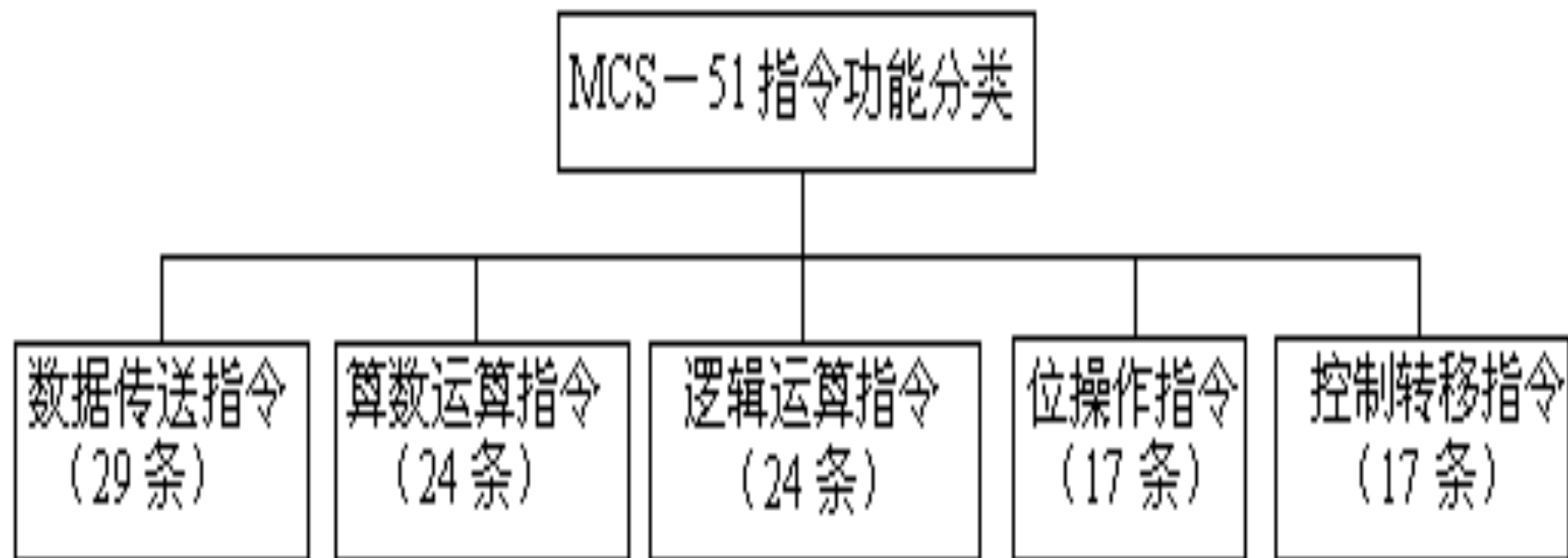
4 机器周期指令

P301附录B

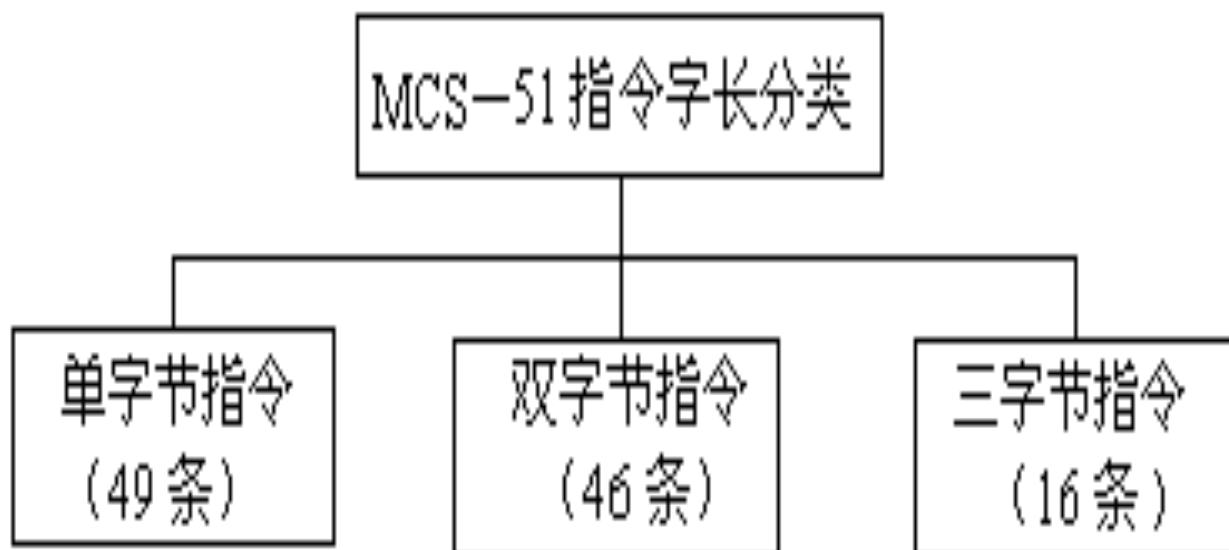


3.1.3 指令分类

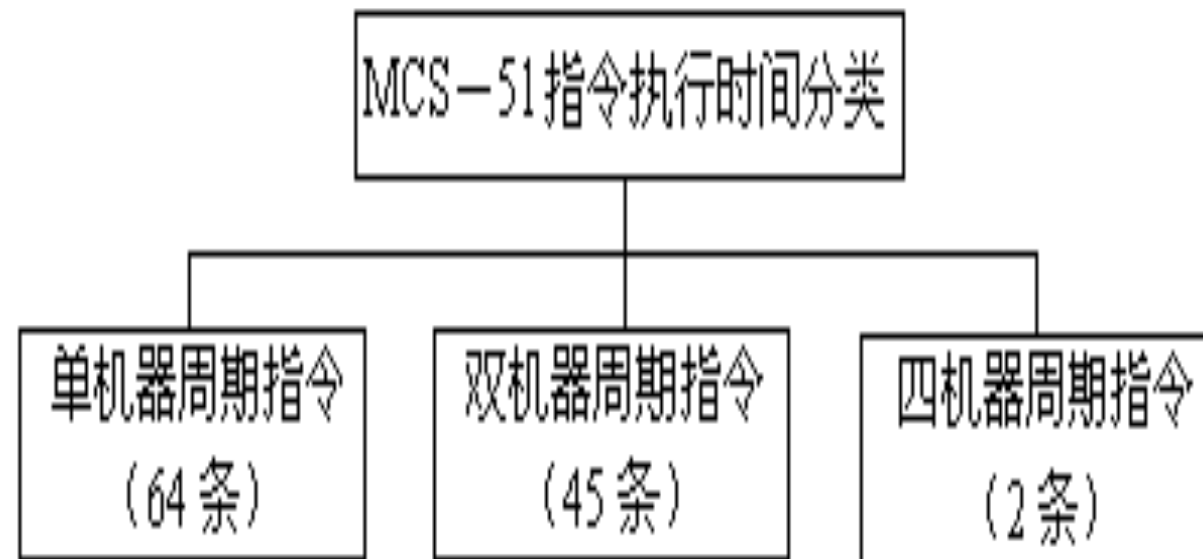
按指令功能分类



按指令字长分类



按指令执行时间分类



3.2 寻址方式

寻址方式是CPU寻找操作数或操作数地址的方法。

寻址方式包含两方面内容：

一是操作数的寻址，二是操作数地址的寻址。

MCS-51单片机的寻址方式有七种。

寻址方式越多，计算机指令功能越强，灵活性越大。

• 寻址方式：7种

- 寄存器寻址
- 直接寻址
- 寄存器间接寻址
- 立即寻址
- 变址间接寻址
- 相对寻址
- 位寻址
- 寻址方式与寻址空间
- MCS-51单片机的两个突出特点

寻址：

1、CPU寻找操作数或操作数地址的方法

2、为PC指针寻找目标地址



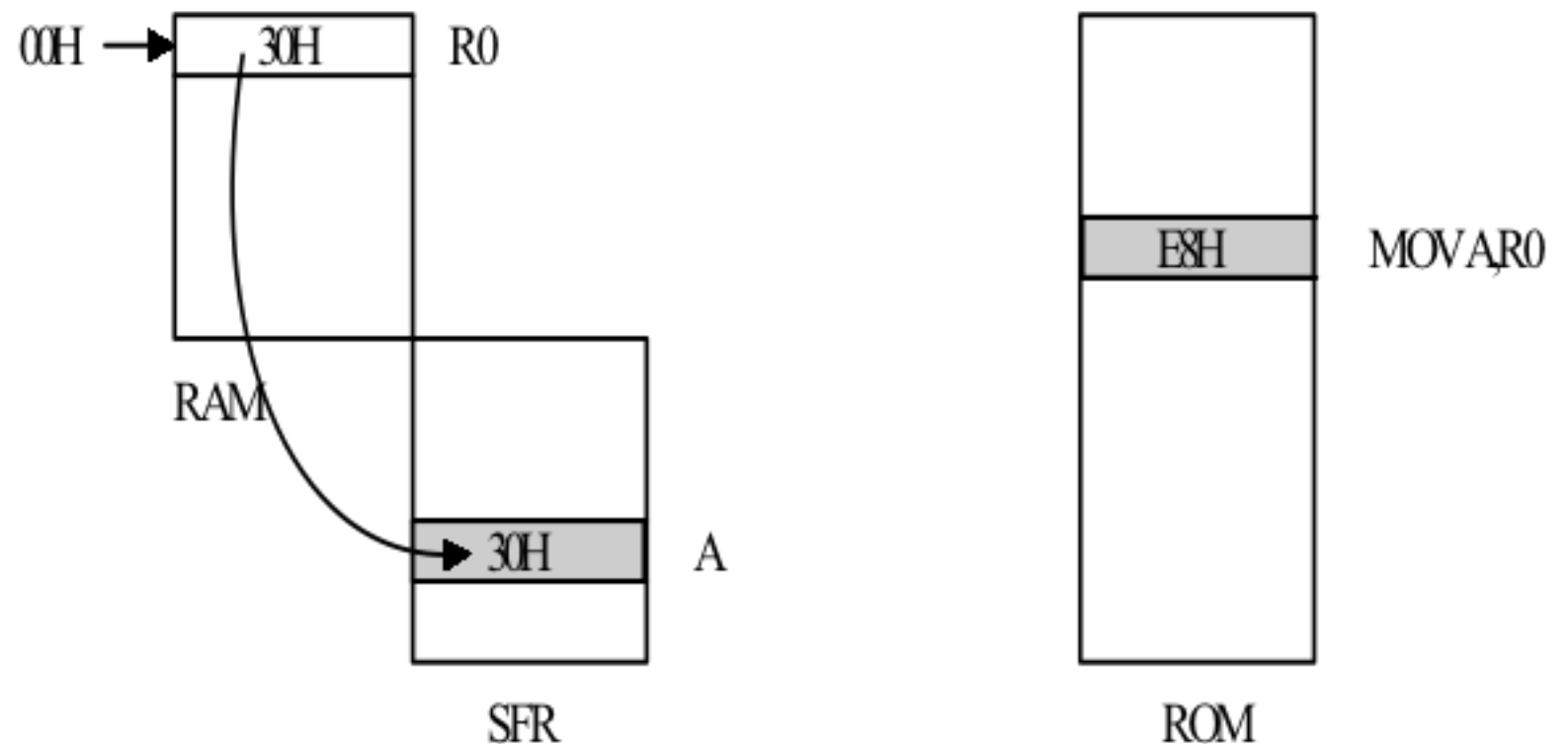
寄存器寻址——操作数存放在寄存器中

指令中直接给出该寄存器名称的寻址方式称为寄存器寻址。

采用寄存器寻址可以获得较高的传送和运算速度。

如：

MOV A, R0



寻址范围:

1. 四组工作寄存器: **R0~R7共32个工作寄存器**, 由PSW中的RS1、RS0两位状态来进行当前寄存器组的选择;
2. 特殊功能寄存器: **累加器A** (注: 使用符号ACC表示累加器时属于直接寻址); **寄存器B** (以AB寄存器对形式出现); **数据指针DPTR和Cy**。

MUL AB; **寄存器寻址**

MOV A, B; **直接寻址**

寄存器为 R0~R7, A, B, DPTR, C

MOV A,R0

MOV R0,#01001111B

SETB RS0

MOV R3,#56H

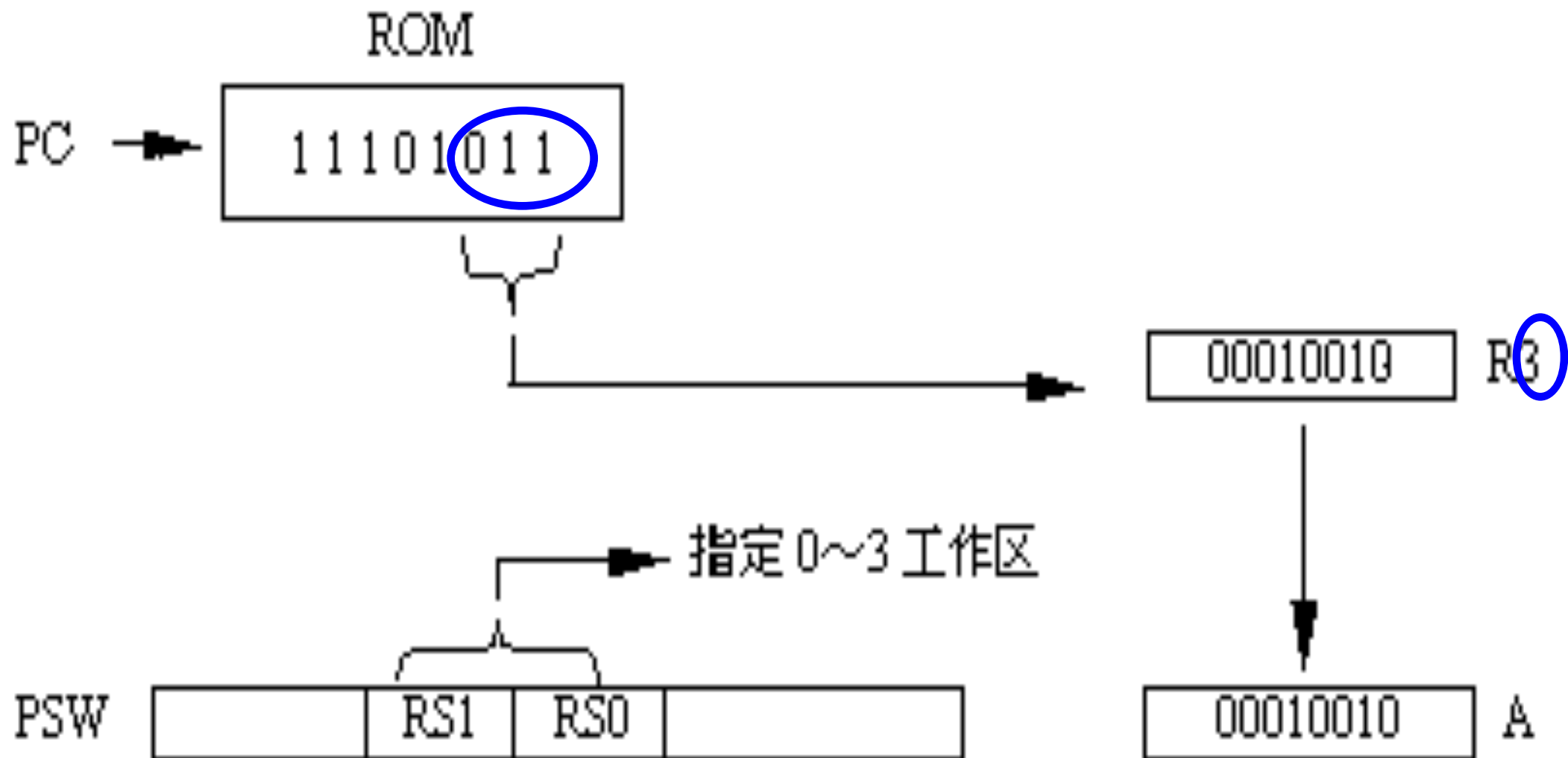
工作寄存器 (00H—1FH)

RS1	RS0	寄存器组	片内RAM地址	寄存器
0	0	第0组	00H~07H	R0~R7
0	1	第1组	08H~0FH	R0~R7
1	0	第2组	10H~17H	R0~R7
1	1	第3组	18H~1FH	R0~R7

例如：MOV A, R3 ;机器码为 **0EBH**

指令功能是把当前R3中的操作数送累加器A。

指令执行示意图如图3-3所示。设 (R3) = 12H

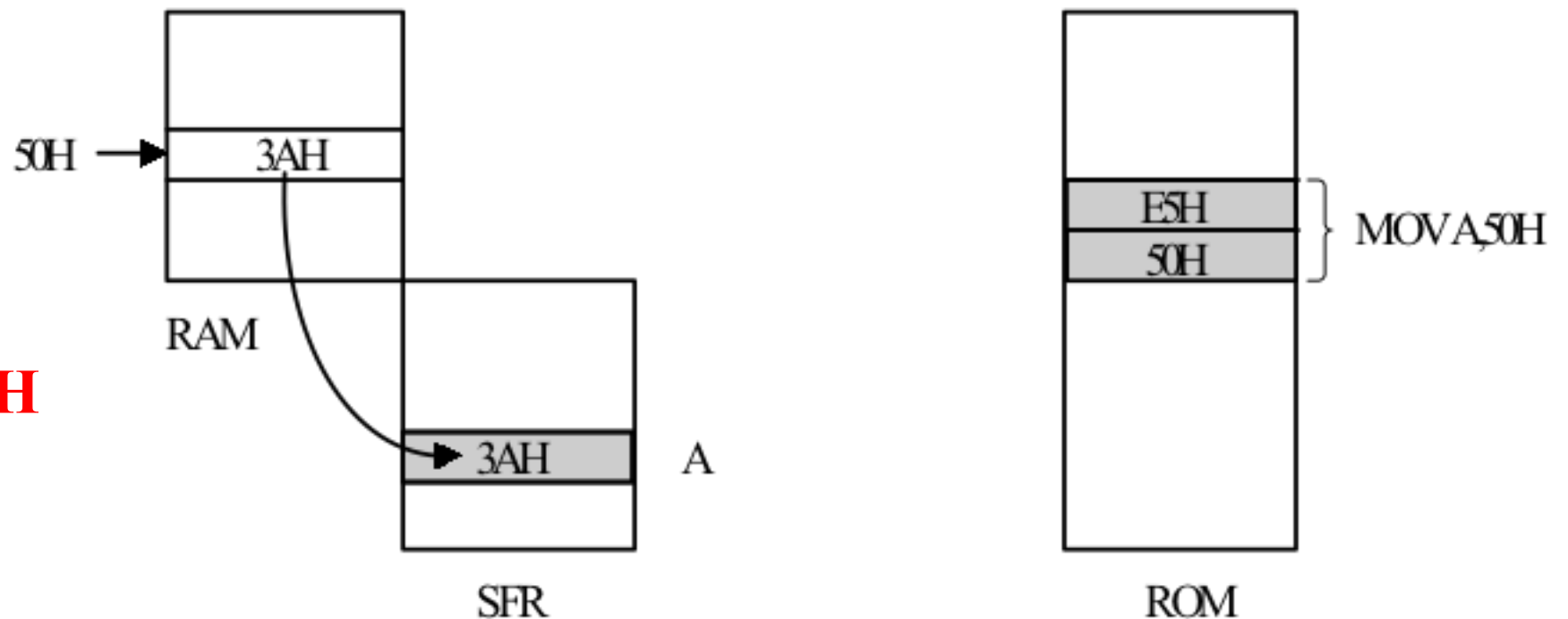


直接寻址——指令中直接给出操作数的地址

指令操作码之后的字节存放的是操作数的地址，操作数本身存放在该地址指示的存储单元中的寻址方式称为直接寻址。

如：

MOV A, 50H



功能最强，可访问 3 种地址空间

- 内部数据存储器地址空间 (R A M) :

0 0 H – 7 F H (直接以单元地址形式给出)

MOV A, 00H

MOV 30H, 20H

- 特殊功能寄存器 (SFR) 地址空间, 唯一方式

8 0 H – 0 F F H (以单元地址或寄存器的符号形式表示 (A、AB、DPTR除外))

MOV A, 90H

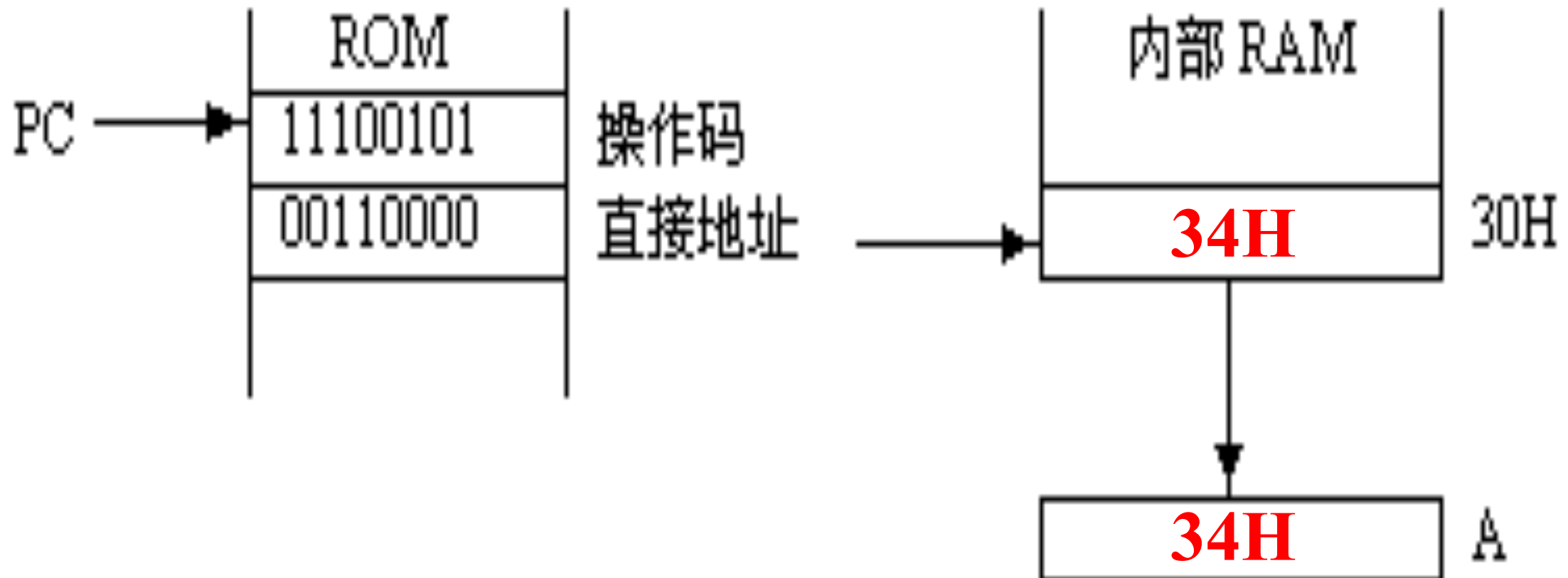
MOV A, B

- 位地址空间 (内部RAM可位寻址的20H—2FH单元对应的128个位地址和SFR中的11个可位寻址的寄存器中的位地址)

0 0 H – 0 F F H MOV C, 00H

例：MOV A , 30H；机器码为E530H

指令功能是把直接地址30H单元的内容送累加器A，即(30H) → (A)如图3-1所示。



在访问特殊功能寄存器SFR**只能**采用直接寻址方式的**原因**：SFR分布在80H~0FFH范围内，而52系列单片机有256字节的片内RAM,其中的80H~0FFH的RAM与SFR所占**地址重叠**。

规定：80H~0FFH范围内的RAM只能用**寄存器间接寻址**方式，而SFR只能用**直接寻址**方式。从而解决了地址冲突的问题。例如：

★MOV A, 90H 等效于 MOV A, P1 属直接寻址

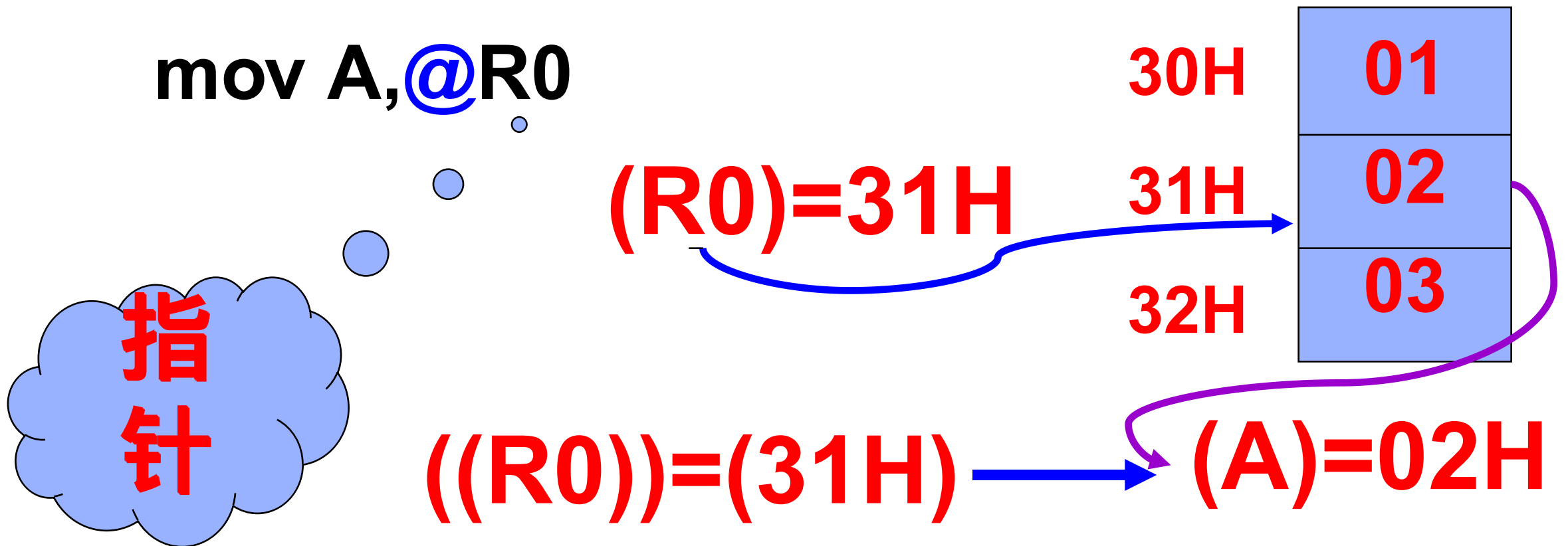
★MOV A, @R0 ; [事先已知 (R0) = #90H]

执行的操作：A ←(90H) 属寄存器间接寻址，

寄存器间接寻址

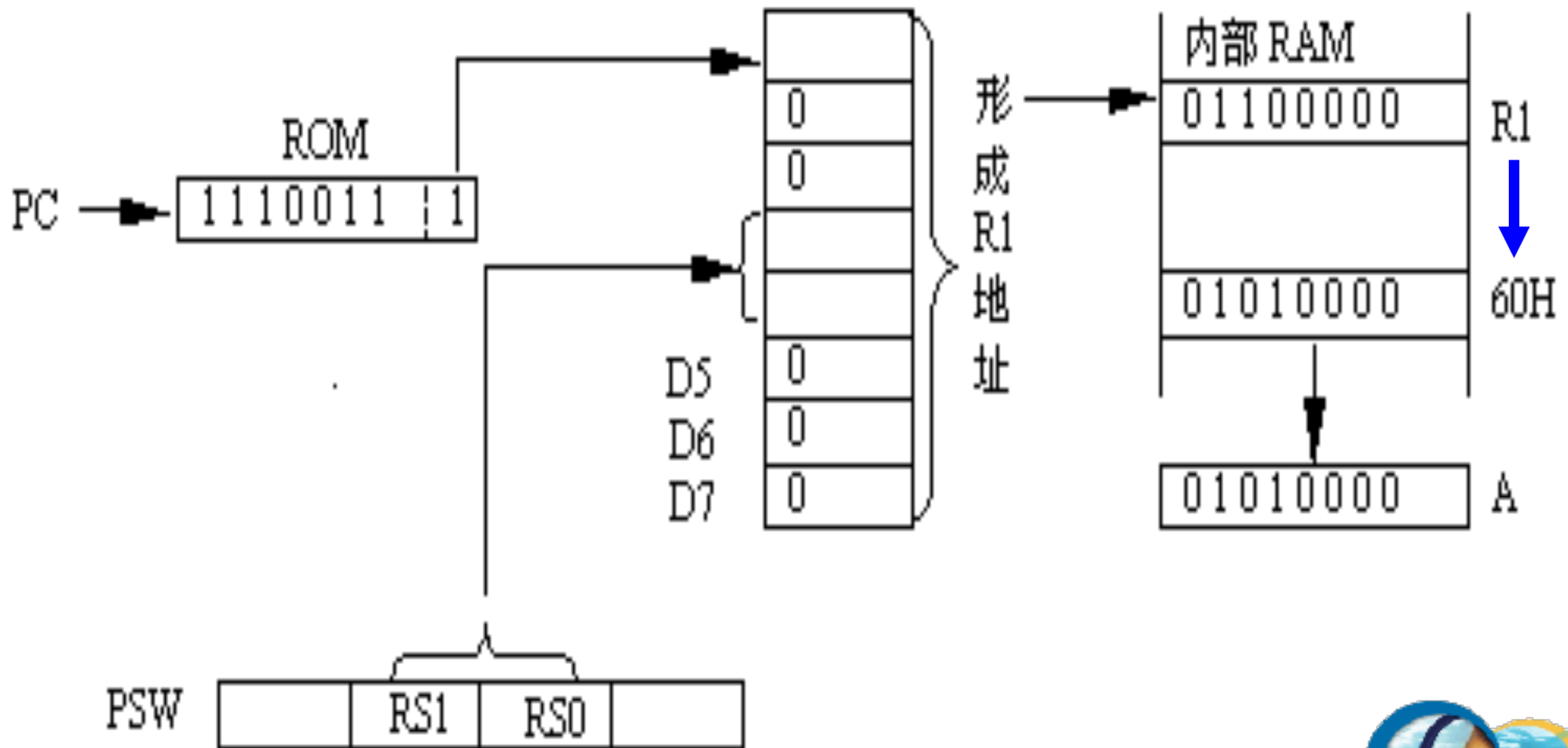
- 将指定的寄存器的内容为地址，由该地址所指定的单元内容作为操作数。

mov A, @R0



例如： MOV A, @R1 ;机器码 E7H

设 (R1) = 60H, (60H) = 50H, 执行结果 (A) = 50H, 该指令执行过程如图3-4所示。



寄存器间接寻址的存储空间为片内RAM或片外RAM:

- 片内RAM的数据传送采用“MOV”类指令，间接寻址寄存器采用R0或R1（堆栈操作时采用SP）；
- 片外RAM的数据传送采用“MOVB”类指令，这时间接寻址寄存器有两种选择：
 - ✓ 一是采用R0和R1作间址寄存器，这时R0或R1提供低8位地址（外部RAM多于256字节采用页面方式访问时，可由P2口未使用的I/O引脚提供高位地址）；
 - ✓ 二是采用DPTR作为间址寄存器。

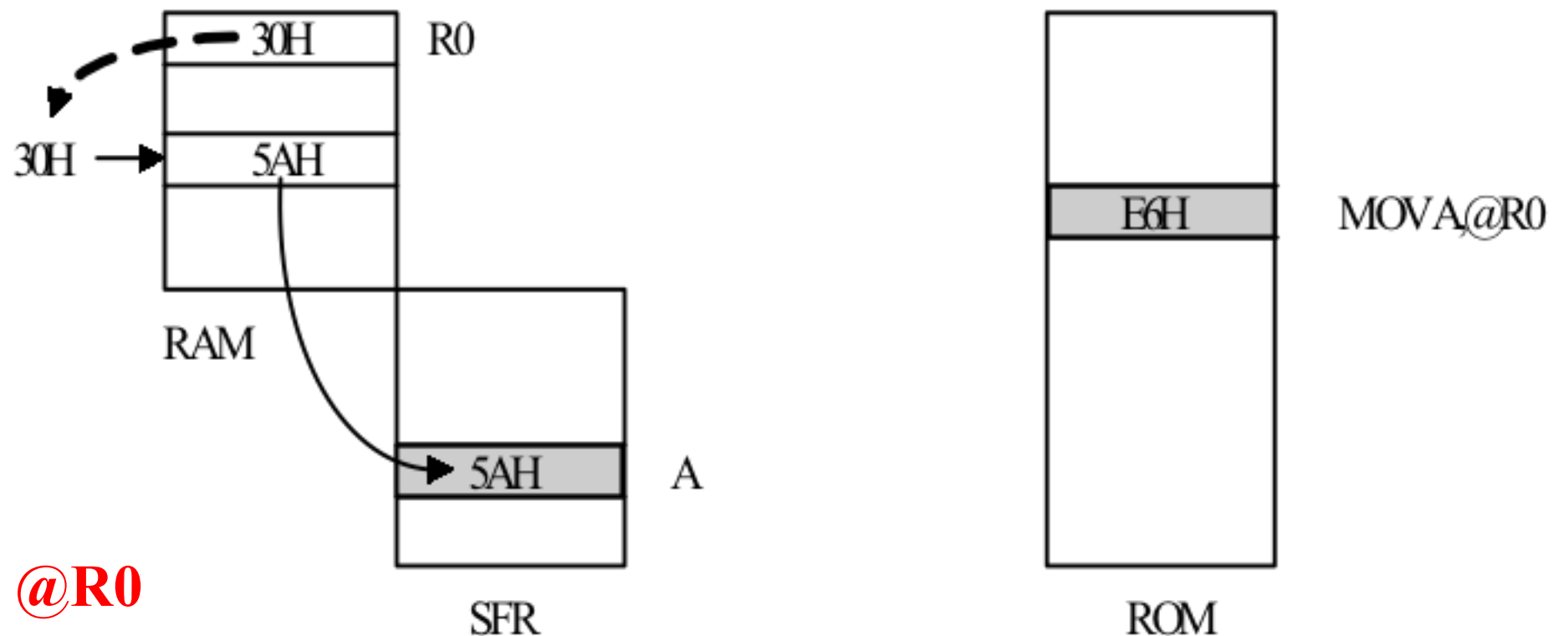
寄存器间接寻址对应的空间为：

片内RAM（采用@R0，@R1或SP）；

片外RAM（采用@R0，@R1或@DPTR）。

如：

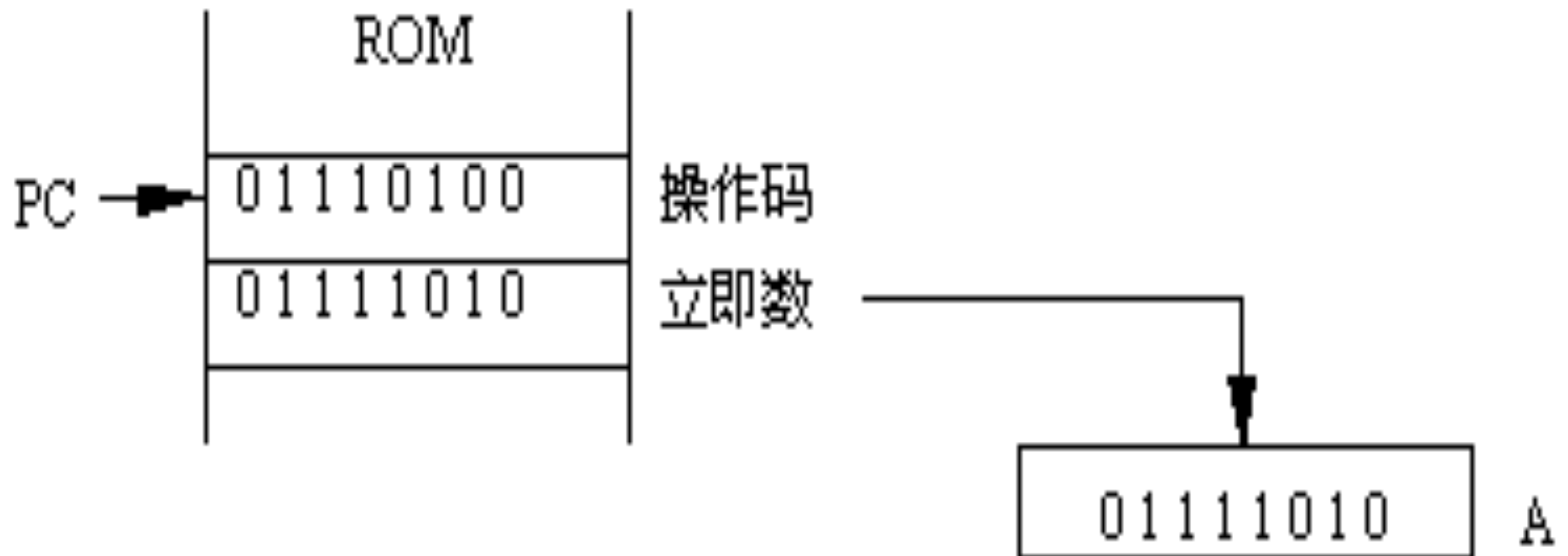
MOV A, @R0



立即寻址——在指令中直接给出操作数

例： **MOV A,#7Ah; 747AH**

把立即数7AH送累加器A，指令执行示意图如图3-2所示。



例: MOV DPTR, #1234h
(DPH)=12H
(DPL)=34H

注意: 立即数前加“#”号, 以区别直接地址。

例如: MOV A, #30H ; (A) ← 30H

MOV A, 30H ; (A) ← (30H)



变址间接寻址

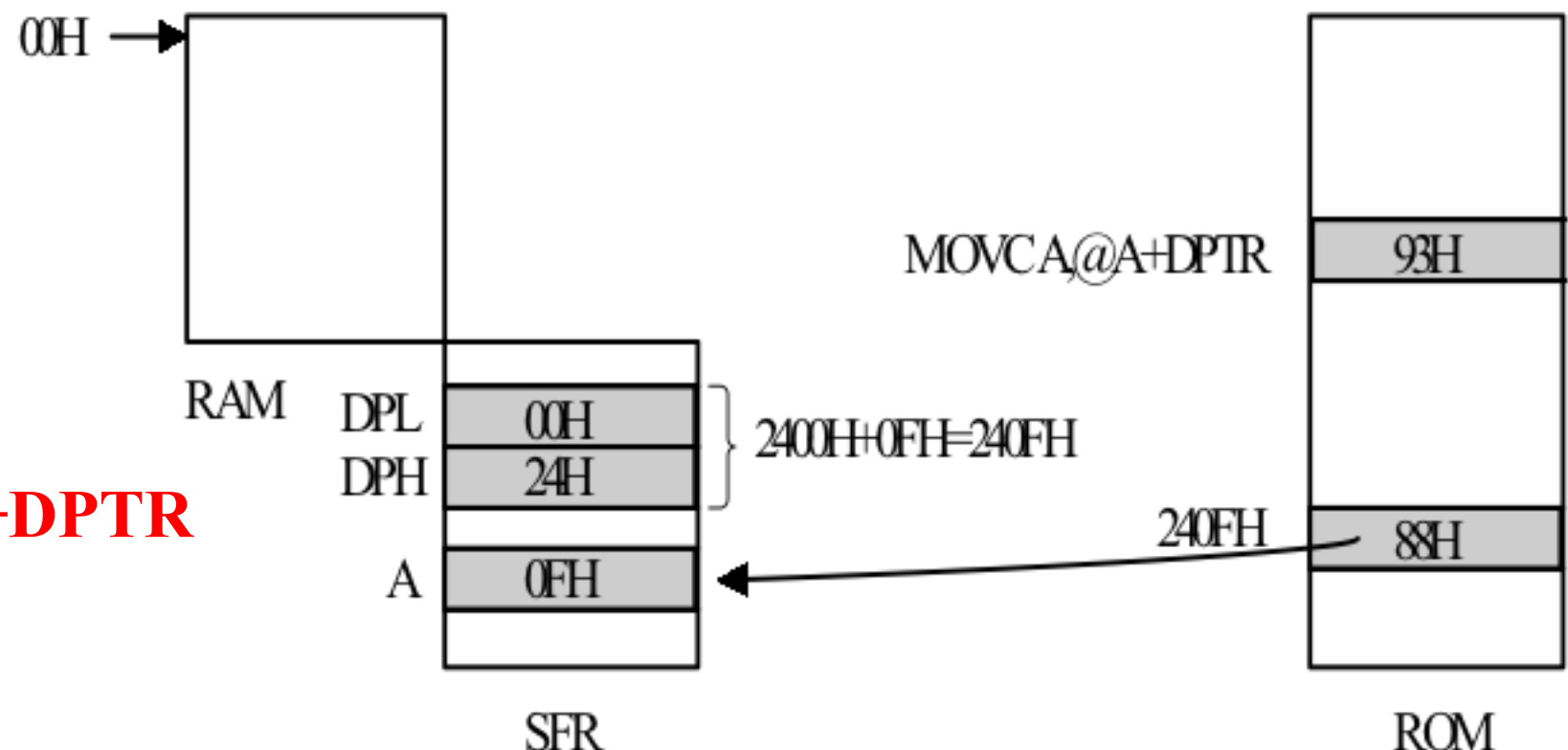
• 基址寄存器 + 变址寄存器的 **间接寻址方式**

以数据指针DPTR或程序指针PC作为基址寄存器，以累加器A作为变址寄存器，并以两者相加形成新的16位地址作为操作数地址，再寻址该地址，读取数据。

变址寻址所对应的**寻址空间**为：ROM

如：

MOVC A, @A+DPTR



- 只能对**程序存储器**进行寻址，用于读取数据，不能用于存放数据。

DPTR/PC A

MOVC A, @ A+DPTR

(A) ((DPTR) + (A))



MOVC A, @ A+PC

(A) ((PC) + (A))



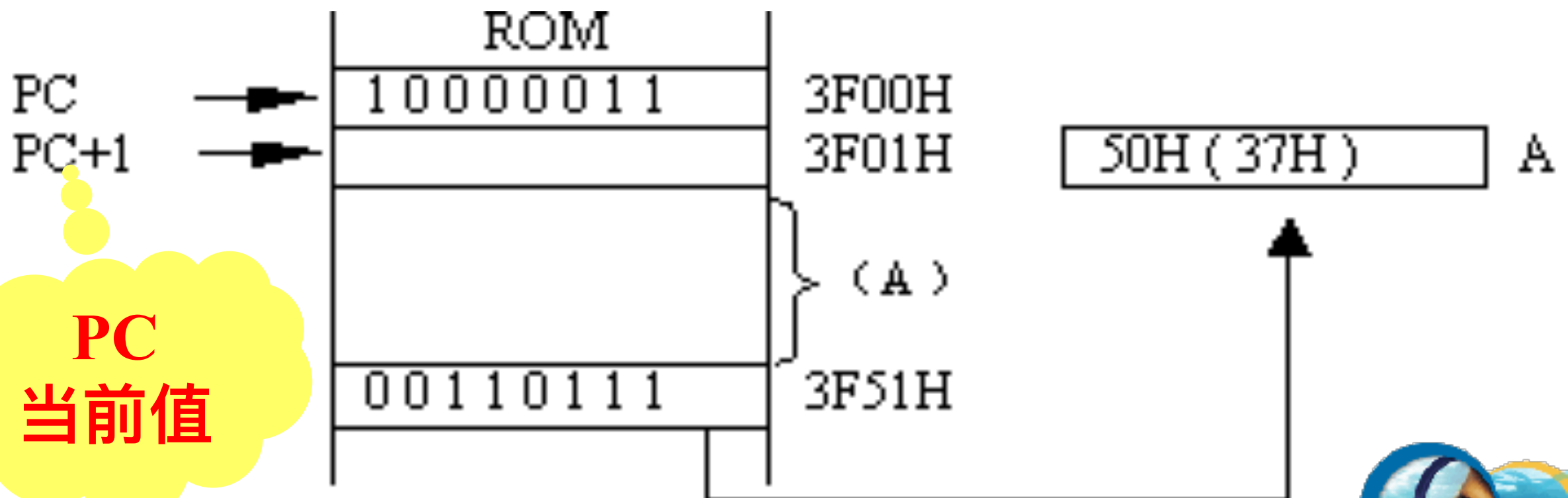
JMP @ A+DPTR

(PC)=(A)+(DPTR)

MOVC A , @A+PC ; 83H

设执行指令之前 (A) = 50H

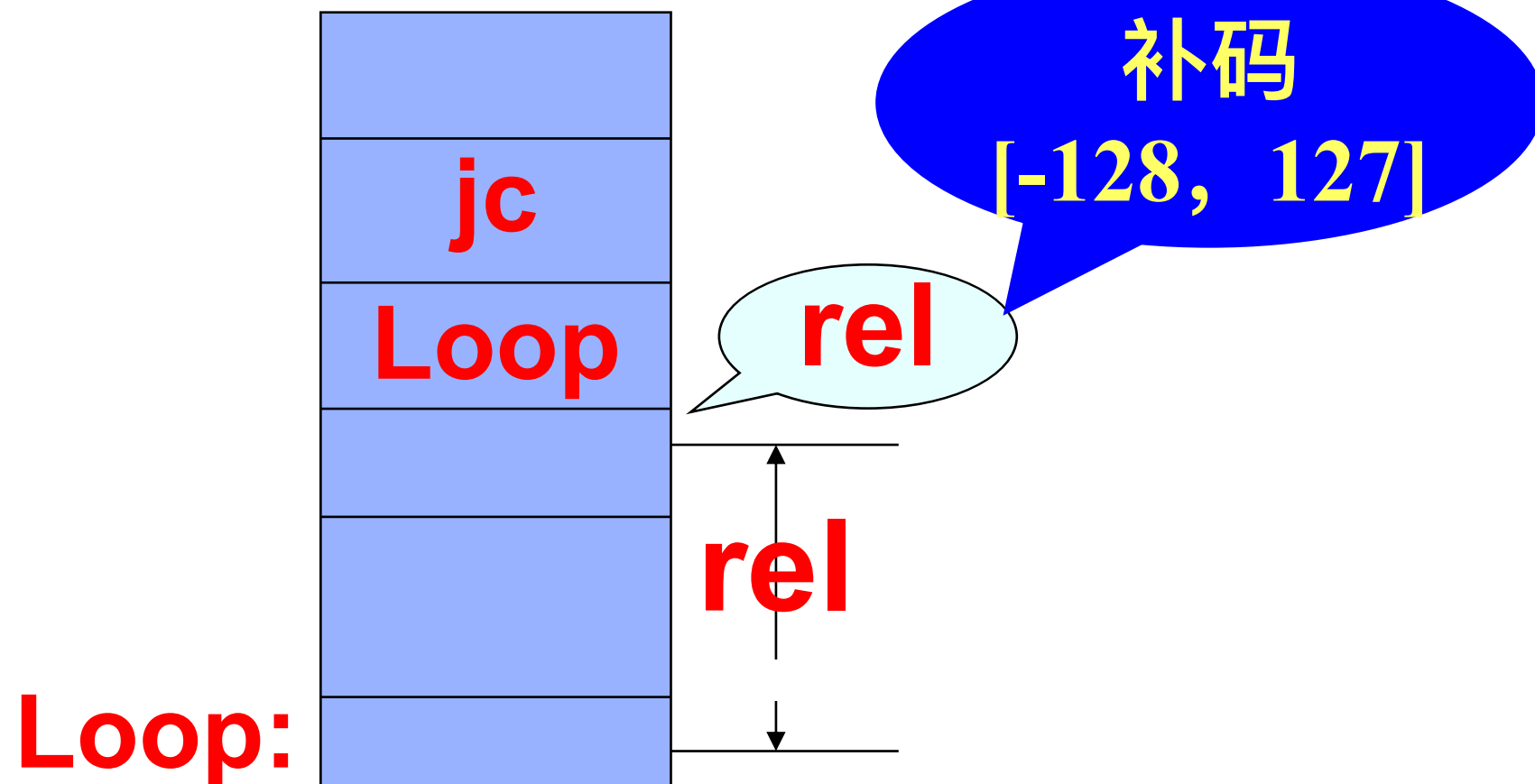
指令功能是把该指令**当前地址PC值**与A累加器内容相加形成操作码地址3F51H, 3F51H中的内容37H送A累加器。



相对寻址

- 以当前的PC值为基准，加上指令中给出的相对偏移量(rel)形成有效的转移地址。
- 用于访问程序存储器，只出现在转移指令中，“寻址”是要得到程序跳转地址PC值，不是操作数的地址。

jc Loop;



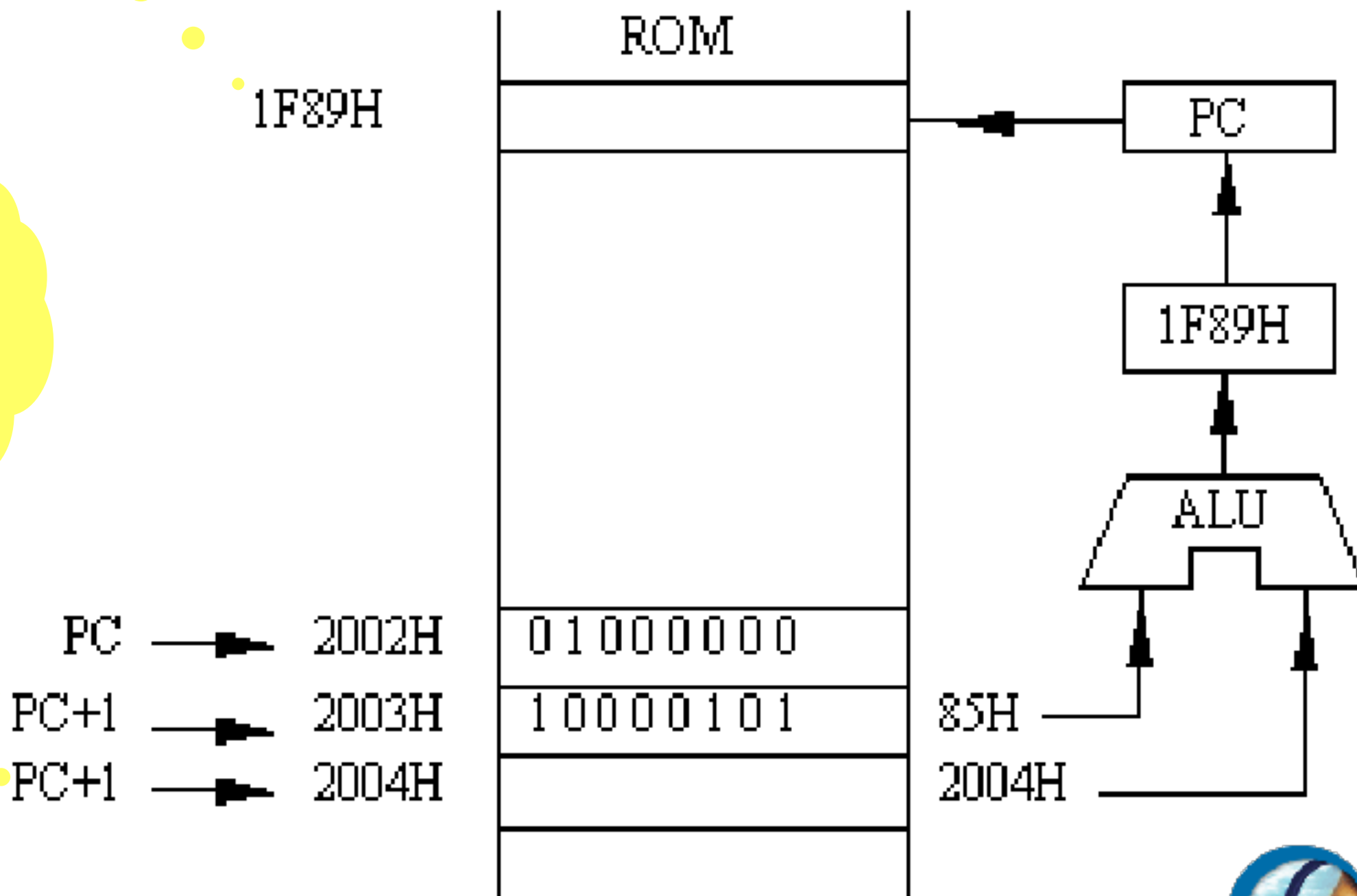
在实际编程中，“rel”通常用标号代替

JC rel; 4085H

设rel=85H, Cy=1

目标地址

PC
当前
值



2021-5-10

注: 85H=10000101B(补码) 其真值为-01111011B=-7BH



位寻址 bit

对位地址中的内容进行操作的寻址方式称为位寻址。采用位寻址指令的操作数是8位二进制数中的某一位。指令中给出的是位地址。位寻址方式实质属于位的直接寻址。

寻址空间为：片内RAM的20H~2FH单元中的128可寻址位；SFR的可寻址位。

习惯上，特殊功能寄存器的寻址位常用符号位地址表示。

位寻址范围：

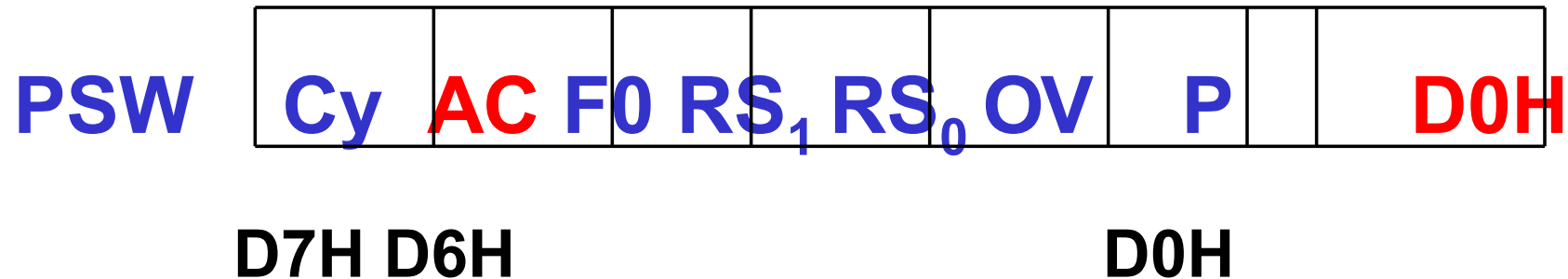
1、内部RAM的位寻址区，共16个单元的128位，字节地址为20H~2FH，位地址为00H~7FH。可用直接位地址或字节地址加位的表示方法。

例如： `MOV C, 7AH`

或 `MOV C, 2FH.2`

2、特殊功能寄存器SFR可供位寻址的专用寄存器共11个，实有位地址位83位。位地址有4种表达方式。

以对程序状态寄存器PSW辅助进位位AC进行操作为例：



1) 直接使用位地址

MOV C, 0D6H

2) 位名称表示法

MOV C, AC

3) 单元地址加位的表示法

MOV C, 0D0H.6

4) 专用寄存器符号加位的表示法

MOV C, PSW.6



寻址方式与寻址空间

寻址方式	寻址空间
寄存器寻址	R0~R7
	A、AB、CY、DPTR
直接寻址	内部RAM 00H~7FH
	特殊功能寄存器SFR 80H~0FFH
	内部RAM中20H~2FH单元的128个位地址
	SFR中83个有效位地址 80H~0FFH
寄存器间接寻址	内部RAM 00H~FFH (@R0、@R1、SP)
	外部RAM或外部I/O口 0000H~0FFFFH (@R0、@R1、@DPTR)
立即寻址	程序存储器
基址+变址 寄存器寻址	程序存储器 (@A+DPTR、@A+PC)
相对寻址	程序存储器 (PC+偏移量)
2021位寻址	内部RAM20H-2FH和部分特殊功能寄存器



T1 内部数据存储器与内部I/O口统一编址

1、内部数据存储器的寻址方式

1) **00H~1FH** 寄存器寻址 4组 **R0 ~R7** **Rn**

直接寻址 **direct**

寄存器间接寻址 **@R0,@R1**

例: (01H) \rightarrow (02H)

MOV 02H,01H ;源寻址和目的寻址均为直接寻址

MOV R2,01H ;源寻址为直接寻址, 目的寻址为寄存器寻址

MOV R0,#01H

MOV 02H,@R0; 源寻址为寄存器间接寻址,
目的寻址为直接寻址

2) 20H ~2FH 可位寻址区的寻址方式

字节寻址方式： 直接寻址 direct

寄存器间接寻址 @R0, @R1

位寻址： bit 直接寻址

例： MOV 26H,C ; 位寻址 (26H) 1位

MOV 26H,A ; 字节寻址 (26H) 8位

3) 30H ~ FFH 数据缓冲区的寻址方式

字节寻址方式： 直接寻址 direct

寄存器间接寻址 @R0, @R1

例：MOV 56H, A ; 字节寻址 (56H) 8位

MOV R0, #96H

MOV @R0, #56H

可以将用户堆栈设在该区内，堆栈指针SP

2、SFR 及I/O口的操作——只能直接寻址

例： P1 口 90H

MOV A,90H

MOV A,P1

MOV P1,A

统一编址

3、关于A累加器有两种寻址方式：

MOV **A**,#23H ; A寄存器寻址

PUSH **ACC** ; 直接寻址

POP **0E0H**

4、可做片内RAM的指针有：

R0, R1, 四个组共有8个

预先设置RS1、RS0, 以选定组。

SETB RS0

CLR RS1; 1组

MOV R0,#34H ; R0的地址是?

SETB RS1 ; 3组

MOV R0,#68H ; R0的地址是?

R0/R1指向的地址范围：00H~FFH

T2 片外数据存储区和外部扩展的I/O口的寻址方式

指针: R0,R1 8位

DPTR 16位

只能寄存器间接寻址

指令助记符: MOVX

例: MOV DPTR,#2000H

MOV A,#34H

MOVX @DPTR,A

或

MOV P2,#20H

MOV R0,#00H

MOV A,#34H

MOVX @R0,A

外部数据存储器的地址, 或
I/O口的地址 16位

高8位地址

低8位地址

统一
编址

R0/R1指向的地址范围:

00H~0FFH



3.3 单片机的指令系统

指令描述符号简介

- **指令分类:**

<u>数据传送类指令</u>	(29)	<u>传送类指令举例</u>
— <u>算术运算类指令</u>	(24)	
— <u>逻辑运算类指令</u>	(24)	
— <u>控制转移类指令</u>	(17)	
— <u>布尔处理类指令</u>	(17)	



指令描述符号 (1) P52

- **Rn--R0~R7** 工作寄存器R₀~R₇, n=0~7
- **direct--** 8位直接地址,表示直接寻址方式
- **@Ri--** 只能是R0或R1, 所以i=0, 1
- **#data --** 8位立即数, 数据范围00H~FFH
- **#data16 --** 16位立即数, 数据范围0000H~FFFFH
- **addr16 --** 16位目标地址
- **addr11 --** 低11位目标地址

指令描述符号 (2)

rel 8位带符号地址偏移量, $[-128, 127]$ 补码

bit 位地址

\$ 当前指令地址

← 数据传送

↔ 数据交换

(X) X单元的内容

((X)) 以X单元的内容为地址的单元的内容



数据传送类指令（5种/29条）

传送类指令占有较大的比重。数据传送是进行数据处理的最基本的操作，这类指令一般**不影响标志寄存器PSW的状态**。

传送类指令可以分成两大类。

一是采用MOV操作符，称为一般传送指令；

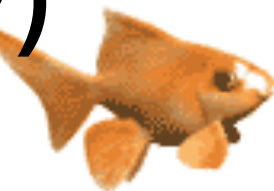
二是采用非MOV操作符，称为特殊传送指令，如：

MOVC、MOVX、PUSH、POP、XCH、XCHD及SWAP。

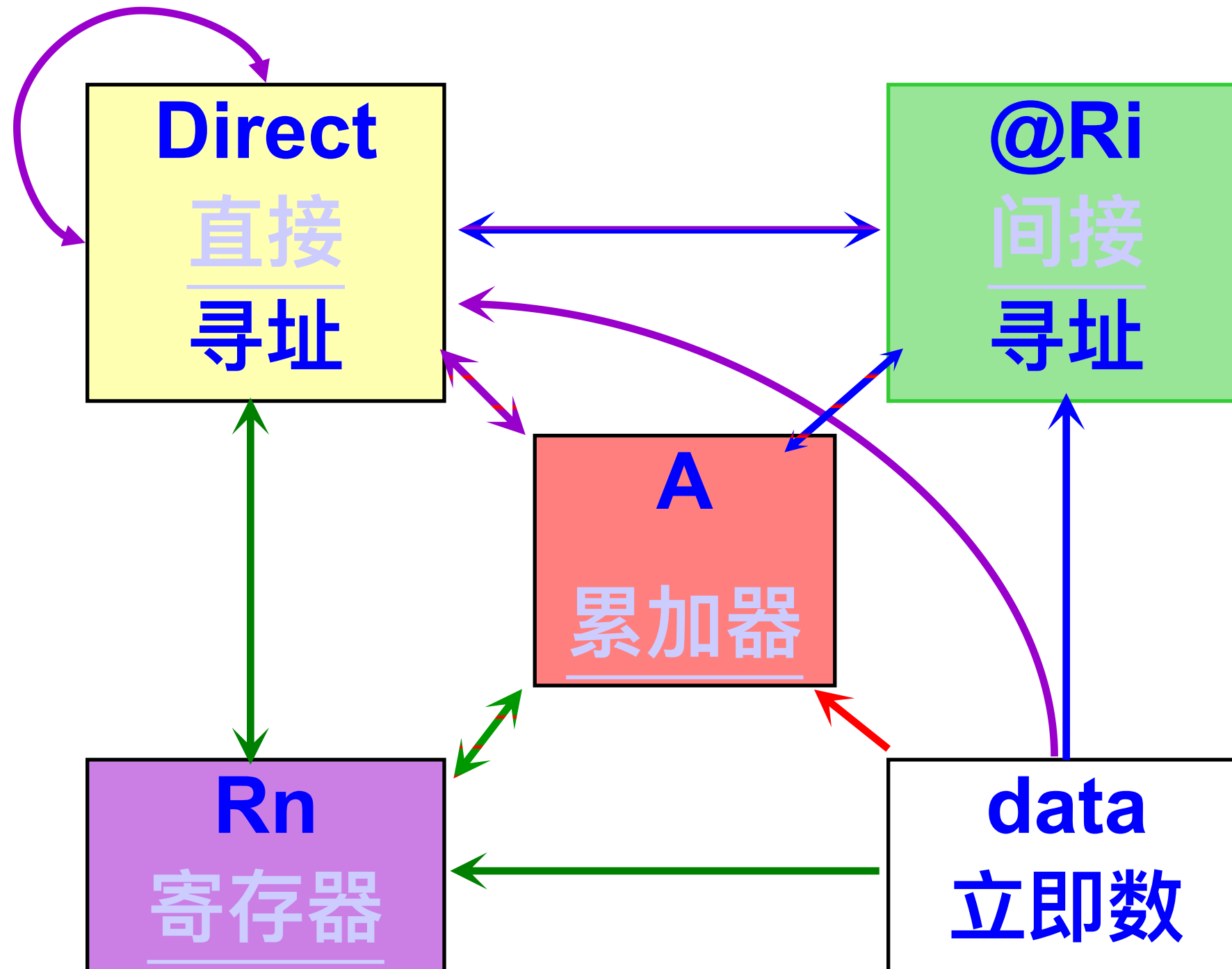
使用8种助记符：**MOV、MOVX、MOVC、XCH、XCHD、SWAP、PUSH及POP**。

数据传送类指令（5种/29条）

- 内部存储器间传送：（MOV——16条）
- 外部数据存储器与累加器间传送：
（MOVX——4条）
- 程序存储器向累加器传送：
（MOVC——2条）
- 数据交换：（XCH, XCHD,
SWAP——5条）
- 堆栈操作：（PUSH, POP——2条）



数据传送类指令(1) 内部



数据传送类指令(2)

内部数据存储器与累加器A之间的传送

- **mov A,Rn ;** $(A) \leftarrow (Rn) \quad n=0\sim7$
- **mov A,direct ;** $(A) \leftarrow (\text{direct})$
- **mov A, @Ri ;** $(A) \leftarrow ((Ri)) \quad i=0,1$
- **mov A,#data ;** $(A) \leftarrow \#data$
- **mov Rn,A ;** $(Rn) \leftarrow (A)$
- **mov Rn,direct ;** $(Rn) \leftarrow (\text{direct})$
- **mov Rn,# data;** $(Rn) \leftarrow \#data$

注意：以A为目的寄存器的传送指令影响PSW中的奇偶标志位。



数据传送类指令(3)

内部RAM中Rn、SFR与片内RAM之间的传送

MOV @R1,@R0

– mov **direct**,A ; (direct) ← (A)

MOV R3,@R0

– mov **direct**,Rn ; (direct) ← (Rn) n=0~7

– mov **direct**,direct ; (direct) ← (direct)

MOV @R1,R0

– mov **direct**,@Ri ; (direct) ← ((Ri)) i=0,1

MOV R1,R0

– mov **direct**,#data ; (direct) ← #data

– mov @Ri,A ; ((Ri)) ← (A)

– mov @Ri,direct; ((Ri)) ← (direct)

– mov @Ri,# data ; ((Ri)) ← #data



数据传送类指令(4) 外部

外部
数据

- **movx** A, @Ri ; $(A) \leftarrow ((Ri)) \ i=0,1$
- **movx** A,@DPTR ; $(A) \leftarrow ((DPTR))$
- **movx** @Ri,A; $((Ri)) \leftarrow (A)$
- **movx** @DPTR,A ; $((DPTR)) \leftarrow (A)$

外部
程序

- **movc** A,@A+DPTR ; $(A) \leftarrow ((A)+(DPTR))$
- **movc** A,@A+PC; $(PC) \leftarrow (PC)+1 ,$
 $(A) \leftarrow ((A)+(PC))$

16位

mov DPTR,#data16; (DPTR) \leftarrow #data16



数据传送类指令(5) 交换指令

- **xch A, Rn ;** $(A) \Leftrightarrow (Rn) \text{ } n=0\sim7$
- **xch A,direct ;** $(A) \Leftrightarrow (\text{direct})$
- **xch A, @Ri ;** $(A) \Leftrightarrow ((Ri)) \text{ } i=0,1$
- **xchd A,@Ri ;** $(A_{0\sim3}) \Leftrightarrow ((Ri)_{0\sim3})$
- **swap A;** $(A_{0\sim3}) \Leftrightarrow (A_{4\sim7})$

数据传送类指令(6) 堆栈操作

—pop direct ; $(\text{direct}) \leftarrow ((\text{sp})) , (\text{sp}) \leftarrow (\text{sp})-1$

—push direct ; $(\text{sp}) \leftarrow (\text{sp})+1, ((\text{sp})) \leftarrow (\text{direct})$

1、堆栈 一种数据结构，是“先进后出”线性表。

2、堆栈操作： 压入 PUSH， 弹出 POP

3、堆栈区： 占片内RAM 中连续的存储单元

复位后，系统自动将SP指针指向07H

用户可将堆栈区设在30H~7FH数据缓冲区内，

MOV SP,#5FH

堆栈有两种类型：向上生长型和向下生长型两种。 **向上生长型堆栈**，栈底在**低地址**单元。随着数据进栈，地址递增，SP 的内容越来越大，指针上移；反之，随着数据的出栈，地址递减，SP 的内容越来越小，指针下移。如(b)图所示。

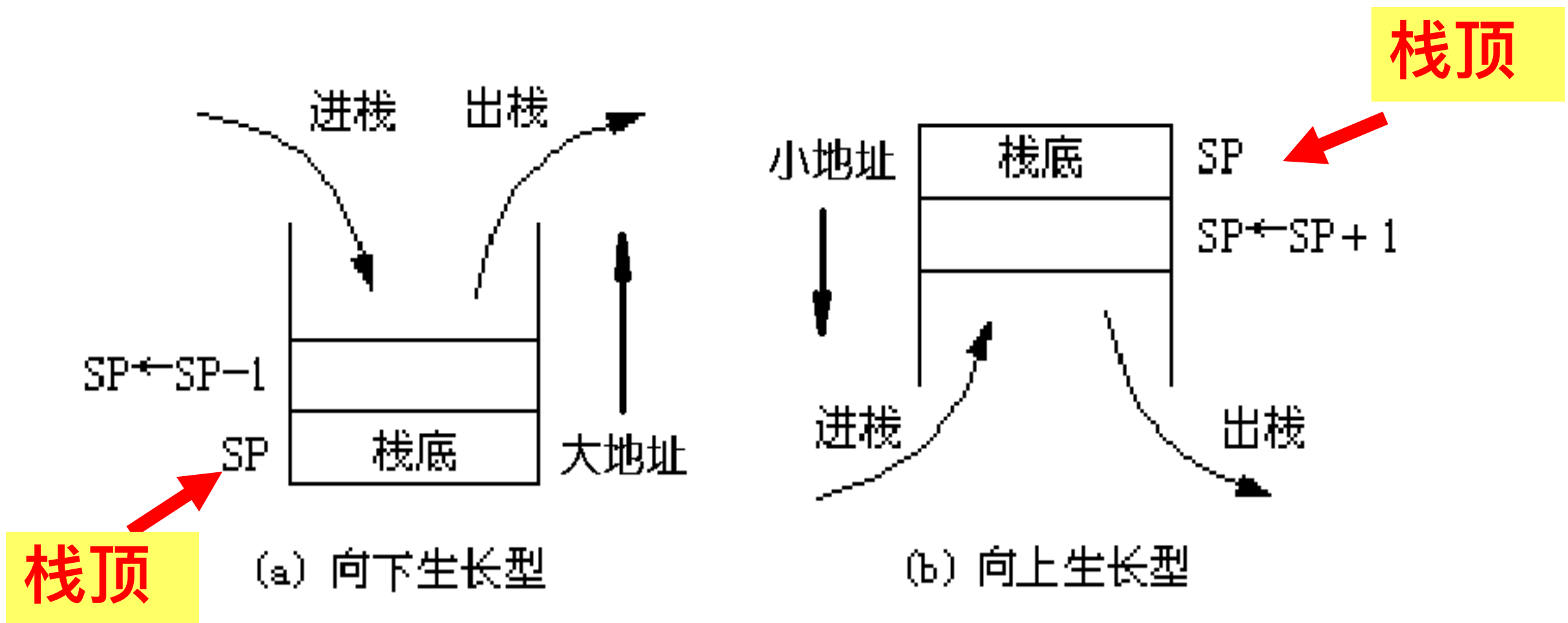


图 2-5 堆栈类型

MCS-51属向上生长型堆栈，这种堆栈的操作规则如下：

进栈操作：先SP加1，后写入数据。

出栈操作：先读出数据，SP减1。

向下生长型堆栈，栈底设在高地址单元。随着数据进栈，地址递减，SP内容越来越小，指针下移；反之，随着数据的出栈，地址递增，SP内容越来越大，指针上移。其堆栈操作规则与向上生长型正好相反。如（a）图所示。

堆栈的使用有两种方式：

一种是**自动方式**，即在调用子程序或断点时，断点地址自动进栈。程序返回时，断点地址再自动弹回PC。这种操作无需用户干预。

另一种是**指令方式**，即使用专用的堆栈操作指令，执行进出栈操作，其进栈指令为PUSH，出栈指令为POP。例如：

保护现场就是一系列指令方式的进栈操作；
而恢复现场则是一系列指令方式的出栈操作。

需要保护多少数据由用户决定。

保护现场：

PUSH ACC

PUSH PSW

PUSH 01H

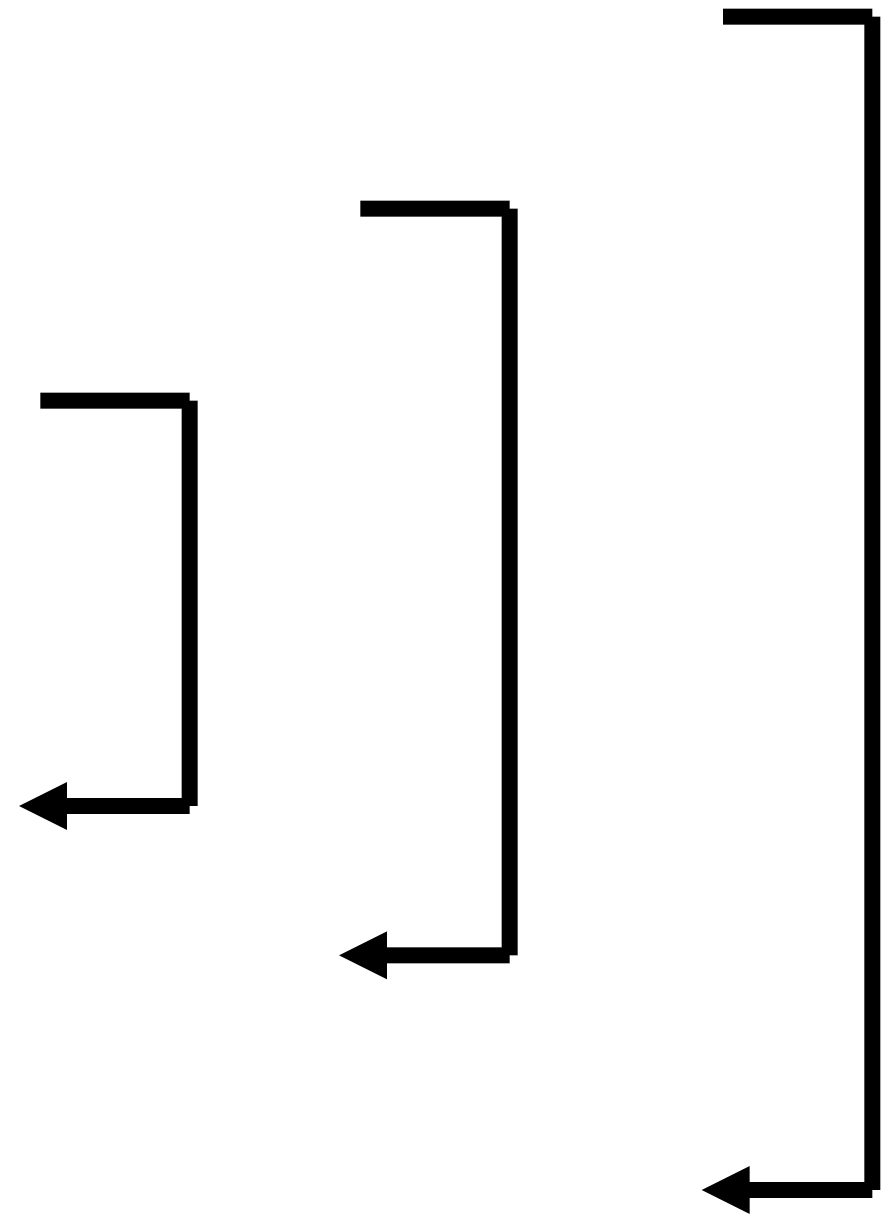
⋮

恢复现场：

POP 01H

POP PSW

POP ACC



系统复位时，SP的内容为07H。通常用户应在系统初始化时对SP重新设置。SP的值越小，堆栈的深度越深。

传送类指令举例：

[例3－1]已知 (R0) = 30H，问执行如下程序，A、R4、30H和31H单元的内容是什么。

MOV A , #10H (A) = 10H

MOV R4 , #36H (R4) = 36H

MOV @R0 , #7AH (30H) = 7AH

MOV 31H , #01H (31H) = 01H

解：8031执行上述指令后的结果为：

[例3－2]设内部RAM中30H单元的内容为40H，40H单元的内容为10H，P1口作输入口，其输入数据为0CAH，程序及执行后的结果如下：

MOV R0, #30H ; 单元地址30H送R0中

MOV A, @R0 ; R0 间址，将30H单元内容送A

MOV R1, A ; A送R1

MOV B, @R1 ; R1间址，将40H单元内容送B

MOV @R1, P1 ; 将P1内容送40H单元

MOV P2, P1 ; 将P1内容送P2

执行结果： (R0)=30H， (R1)=40H， (A)=40H， (B)=10H，
(P1)=0CAH (40H)=0CAH， (P2)=0CAH

[例3－3]已知**片外**RAM 的70H单元中的一个数X，需送到**片外**RAM的 1010H单元，试编写程序。

解： ORG 1000H

 MOV R0 , #70H

 MOV DPTR , #1010H

MOVX A , @R0

MOVX @DPTR , A

 SJMP \$

 END

**外部RAM之
间不能直接传送
数据,必须通过
累加器A传送**

~~MOVX @DPTR, @R0~~

[例3-5] 设 $(30H) = X$, $(40H) = Y$, 试利用堆栈区域实现30H和40H单元中的数据交换。

解：堆栈区是片内RAM的一个数据区，进栈和出栈的数据符合“先进后出”的原则。

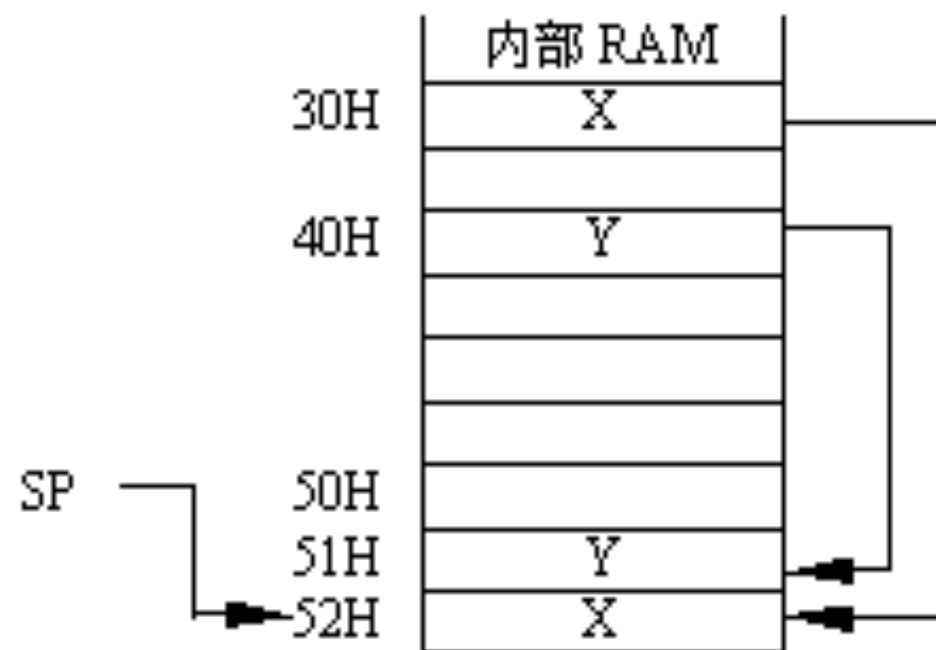
MOV SP , #50H; 设栈底 (栈底不存数)

PUSH 40H ; $(51H) \leftarrow (40H)$

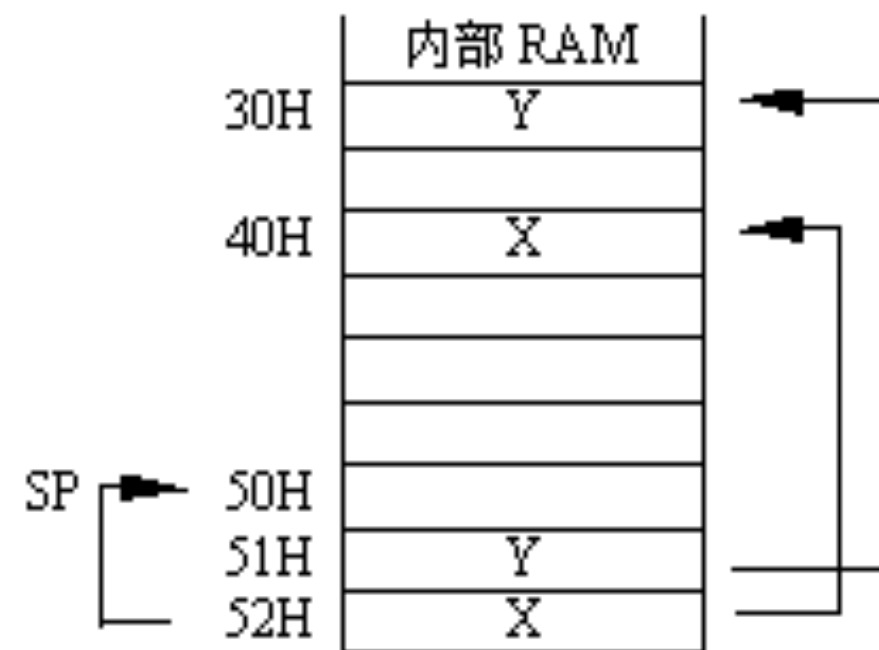
PUSH 30H ; $(52H) \leftarrow (30H)$

POP 40H ; $(40H) \leftarrow (52H)$

POP 30H ; $(30H) \leftarrow (51H)$



(a) 压入 Y , X



(b) 弹出 X , Y

[例3-6] 已知外部RAM 2020H单元中有一个数X， 内部RAM 20H单元一个数Y， 试编出可以使它们互相交换的程序。

解： MOV P2 , #20H
 MOV R1 , #20H
 MOVX A , @R1
 XCH A , @R1
 MOVX @R1 , A
 SJMP \$
 END

指向外部
RAM
2020H单元

指向内部
RAM
20H单元

[例3-7] 已知片内50H单元中有一个0~9的数，
试编程把它变为相应的ASCII码的程序。

解：因为0~9的ASCII码为30~39H

程序如下： **MOV** **R0** , #50H

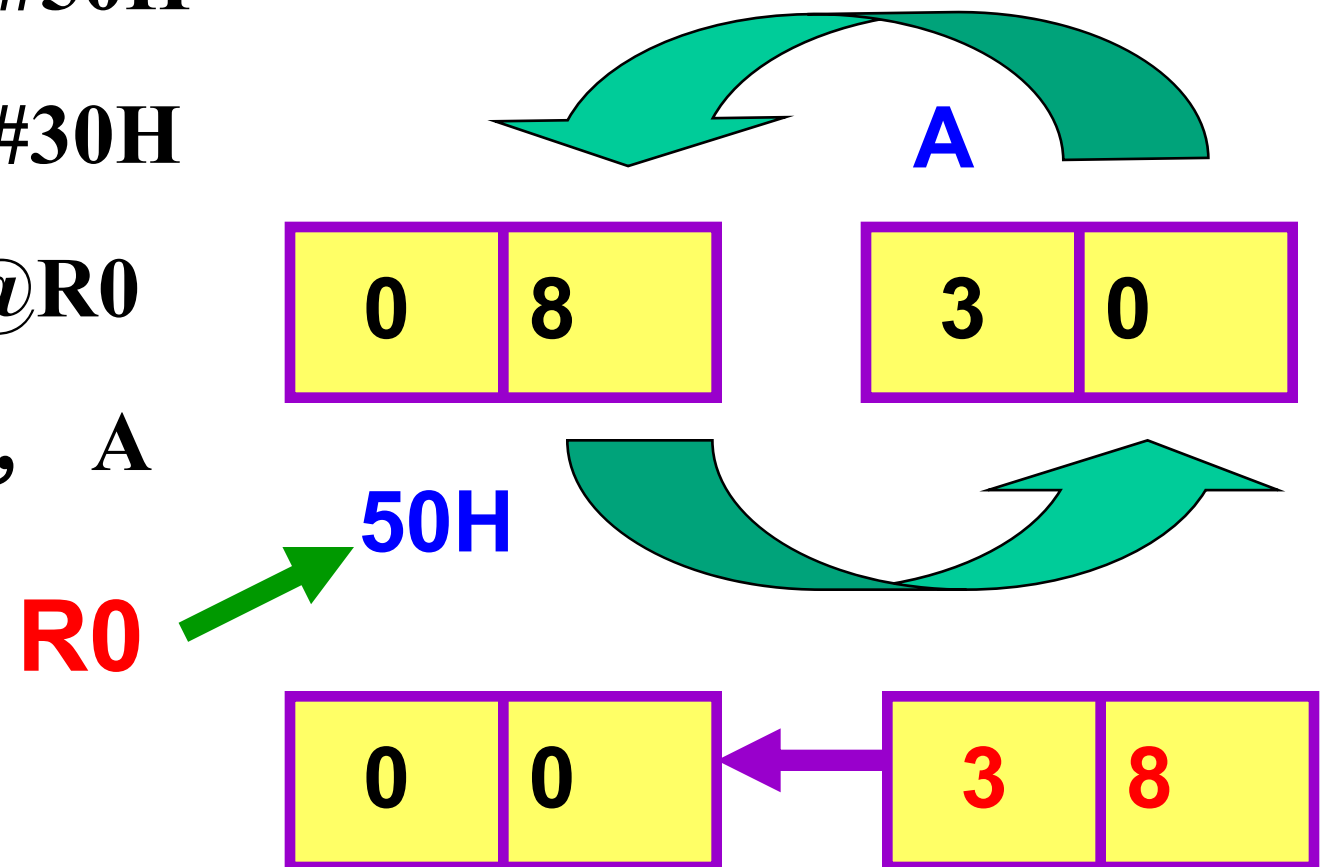
MOV **A** , #30H

XCHD **A** , @**R0**

MOV @**R0** , **A**

SJMP \$

END



[例3－8] 把01H单元内容送02H单元，有几种不同的实现方法。

- | | |
|------------------------|-----------------|
| ① MOV 02H , 01H | ； 直接寻址 3字节 2周期 |
| ② MOV A , 01H | ； 直接寻址 + 寄存器寻址 |
| MOV 02H , A | ； 4字节 2周期 |
| ③ MOV A , R1 | ； 寄存器寻址 2字节 2周期 |
| MOV R2 , A | |
| ④ MOV R0 , #01H | ； 4字节 3周期 |
| MOV 02H , @R0 | ； 间接寻址 |
| ⑤ PUSH 01H | ； 栈操作 4字节 4周期 |
| POP 02H | |

第三种方法占存储空间少，执行周期短。



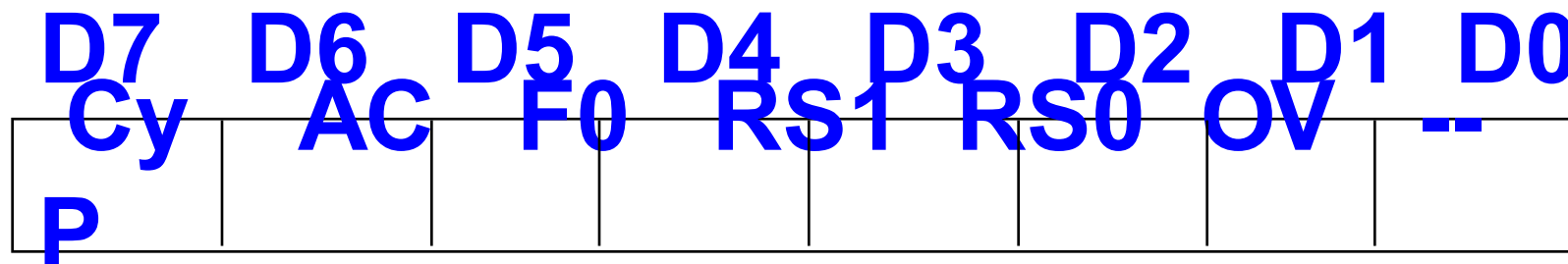
算术运算类指令（24条）

算术运算指令可以完成加、减、乘、除及加1和减1等运算。这类指令多数以A为源操作数之一，同时又使A为目的操作数。

- 程序状态字 PSW
- 加法指令
- 减法指令
- 乘/除指令



程序状态字 PSW



- **Cy**: 进位标志；布尔累加器
- **AC**: 辅助进位标志（半进位）
- **F0**: 用户标志
- **RS1/RS0**: 寄存器选择
- **OV**: 溢出标志
- **P**: 奇偶标志

进位（借位）标志CY为无符号整数的多字节加法、减法、移位等操作提供了方便；**溢出标志OV**可方便的控制补码运算；**辅助进位标志AC**用于BCD码运算。算术运算操作将影响PSW中的OV、CY、AC和P等。

指令 标志	ADD	ADDC	SUBB	DA	MUL	DIV
CY	√	√	√	√	0	0
AC	√	√	√	√	X	X
OV	√	√	√	X	√	√
P	√	√	√	√	√	√

注：符号√表示相应的指令操作影响标志；符号0表示相应的指令操作对该标志清0。符号X表示相应的指令操作不影响标志。另外，累加器加1（INC A）和减1（DEC A）指令影响P标志。



加法指令

- 不带进位的加法指令
- 带进位的加法指令
- 加 1 指令
- 二-十进制调整指令



不带进位的加法指令

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

带进位的加法指令

- **ADDC** A, Rn ;(A)<--(A)+(Rn)+(C)
- **ADDC** A, direct ;(A)<--(A)+(direct) +(C)
- **ADDC** A, @Ri ;(A)<--(A)+((Ri)) +(C)
- **ADDC** A, #data ;(A)<--(A)+#data +(C)

带进位加 常用于多字节加法运算。

加的进位标志**CY**的值是在该指令执行之前已经存在的进位标志的值，而不是执行该指令过程中产生的进位。

[例3-15] 设20H~21H单元存放一个16位二进制数X1（高8位存于21H单元），30H~31H单元存放另一个16位二进制数X2（高8位存于31H单元）。求X1+X2，和存于20H~21H设两数之和不超过16位。

解： ORG 2000H

MOV R0 , #20H

MOV R1 , #30H

MOV A , @R0 ; 取被加数低8位

ADD A , @R1 ; 求和的低8位

MOV @R0 , A ; 存和的低8位

INC R0 ; 指向被加数高8位

INC R1 ; 指向加数高8位

MOV A , @R0 ; 取被加数高8位

ADDC A , @R1 ; 求和的高8位

MOV @R0 , A ; 存和的高8位

SJMP \$; 停机

END

2021-5-10

(21H)	(20H)
+ (31H)	(30H)
Cy(21H)	(20H)
ADDC	ADD

运算结果

高8位存于21H单元，
低8位存于20H单元。



加 1 指令

- **INC A** ; **$(A) \leftarrow (A) + 1$**
- **INC Rn** ; **$(Rn) \leftarrow (Rn) + 1$**
- **INC direct** ; **$(direct) \leftarrow (direct) + 1$**
- **INC @Ri** ; **$((Ri)) \leftarrow ((Ri)) + 1$**
- **INC DPTR** ; **$(DPTR) \leftarrow (DPTR) + 1$**

指令的**功能**是把源操作数的内容加 1，结果再送回原单元。这些指令仅 **INC A** 影响 P 标志。其余指令都不影响标志位的状态。



二进制调整指令

- **DA A**

指令的功能是对累加器A中刚进行的两个BCD码的加法的结果进行十进制调整。

- **调整要完成的任务是：**

- (1) 当累加器A中的**低4位**数出现了非BCD码（1010~1111）或低4位产生进位（AC=1），则应在低4位加6调整，以产生低4位正确的BCD结果。
- (2) 当累加器A中的**高4位**数出现了非BCD码（1010~1111）或高4位产生进位（CY=1），则应在高4位加6调整，以产生高4位正确的BCD结果。

十进制调整指令执行后，PSW中的CY表示结果的百位值。

- **DA A**

- **调整原则:**

- 形式上非BCD码 需要加 06H、60H、66H调整
- 形式上是BCD码时:

CY	AC	调整原则
0	0	不调整
0	1	+06H
1	0	+60H
1	1	+66H

举例: BCD码加法



[例3－20] 利用DA指令作十进制加法调整。两个用单字节压缩BCD码表示的十进制数相加，借位标志存入C累加器。设20H、21H、22H分别存放被加数、加数以及和。

解：ORG 2000H

MOV A, 20H ; (A) ← 取被加数

ADD A, 21H ; (A) ← 被加数+加数

DA A ; 进行调整

L1: MOV 22H, A ; 存结果

SJMP \$;

END



减法指令

- **SUBB A, Rn ;(A)<--(A)-(Rn)-(C)**
- **SUBB A, direct ;(A)<--(A)-(direct) -(C)**
- **SUBB A, @Ri ;(A)<--(A)-((Ri)) -(C)**
- **SUBB A, #data ;(A)<--(A)-#data -(C)**

注：如何判断Cy, AC, OV

在减法之前，Cy需要先清0。



例 若 (A) =C9H, (R2) =54H, (CY) =1,
执行指令 SUBB A, R2 之后, 由于:

(A): ↵	1 1 0 0 1 0 0 1↵
— (CY): ↵	1↵
↵	1 1 0 0 1 0 0 0↵
— (R2): ↵	0 1 0 1 0 1 0 0↵
结果: ↵	0 1 1 1 0 1 0 0↵

即: (A) =74H, (CY) =0, (AC) =0, (OV)
=1 (位6有借位, 位7无借位), (P) =0。



减 1 指令

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

•这组指令的功能是把操作数的内容减 1 ， 结果再送回原单元。

•这组指令仅 DEC A 影响P标志。其余指令都不影响标志位的状态。



[例3－16] 试分析执行以下程序后，各有关单元的结果。

MOV R1 , #7FH

MOV 7EH , #00H

MOV 7FH , #40H

DEC @R1

DEC R1

DEC @R1

执行结果： (R1) =7EH (7EH) =0FFH
(7FH) =3FH



乘/除指令

- 无符号数乘法指令

- **mul AB ; $(A) \times (B) \rightarrow (B)(A)$**

- 若 $(B) \neq 0$ 则 $OV=1$, 否则 $OV=0$; $Cy=0$

- 无符号数除法指令

- **div AB ; $(A)/(B)=(A)$, 余数(B)**

- $Cy=0$; 若 $(B)=0$ 则 $OV=1$



逻辑运算类指令

- 单字节逻辑操作数运算指令
- 双字节逻辑操作数运算指令



单字节逻辑操作数运算指令

- 累加器A清“零”指令
- 累加器A取反指令
- 累加器A循环左移指令
- 累加器A连同进位位循环左移指令
- 累加器A循环右移指令
- 累加器A连同进位位循环右移指令



累加器A清“零”指令

- **CLR A;**
 - $(A) \leftarrow 0;$

累加器A清:

CLR A;

CLR C

SUBB A,0E0H

清零操作影响标志位P



双字节逻辑操作数运算指令

- 逻辑“与”指令
- 逻辑“或”指令
- 逻辑“异或”指令



逻辑“与”指令

- **ANL A, Rn** **;(A)←(A) • (Rn)**
- **ANL A, direct**
- **ANL A, @Ri**
- **ANL A, #data**
- **ANL direct, A**
- **ANL direct, #data**

逻辑“或”指令

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

逻辑“异或”指令

- **XRL A, Rn** $;(A) \leftarrow (A) \oplus (Rn)$
- **XRL A, direct**
- **XRL A, @Ri**
- **XRL A, #data**
- **XRL direct, A**
- **XRL direct, #data**

累加器A取反指令

- **CPL A**
 - (A)=55H
 - **CPL A**
 - (A)=0AAH
 - **CLR A**
 - **CPL A**
 - **INC A**

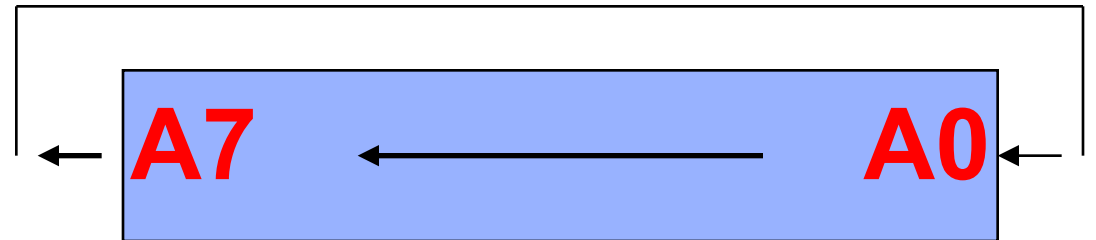


累加器A循环左移指令

- **RL A ; ROTATE LEFT**

- (A)=55H

- RL A; (A)=0AAH



- MOV A,#01H ; (A) ← 01H;

- RL A ; (A) ← 02H;

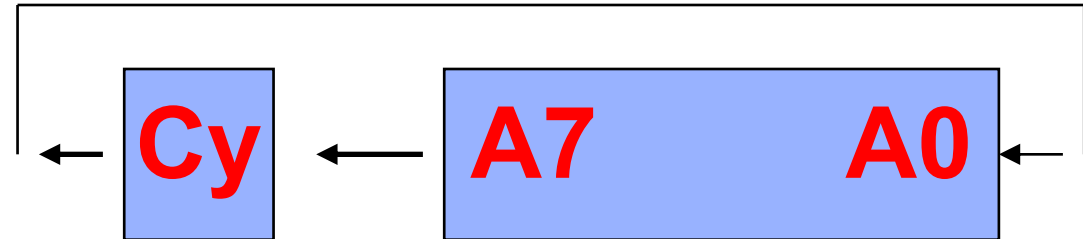
- RL A ; (A) ← 04H;

- RL A ; (A) ← 08H;



累加器A连同进位位循环左移指令

RLC A

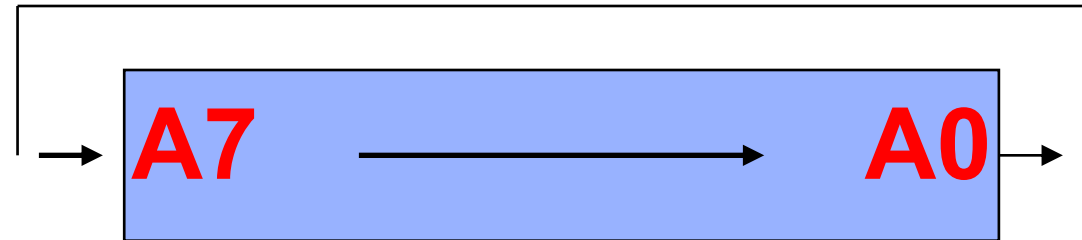


- $(A)=0$
- $(CY)=1$
- **RLC A** ; $(A) \leftarrow 01H, (CY)=0;$
- **RLC A** ; $(A) \leftarrow 02H, (CY)=0;$
- **RLC A** ; $(A) \leftarrow 04H, (CY)=0;$
- **RLC A** ; $(A) \leftarrow 08H, (CY)=0;$



累加器A循环右移指令

RR A ; ROTATE RIGHT



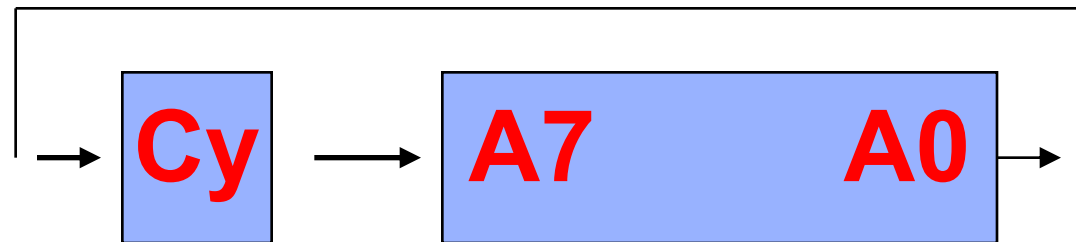
- **(A)=08H ; (A) ← 08H;**
- **RR A ; (A) ← 04H;**
- **RR A ; (A) ← 02H;**
- **RR A ; (A) ← 01H;**



累加器A连同进位位循环右移指令

RRC A;

- **(A)=0**
- **(CY)=0**
- **RRC A**
- **RRC A**
- **RRC A**
- **RRC A**



布尔处理类指令（17条）

位操作又称布尔操作，它是**以位为单位进行的各种操作**。位操作指令中的位地址有4种表示形式：

- 直接地址方式（如，0D5H）；
- 点操作符方式（如，0D0H.5、PSW.5等）；
- 位名称方式（如，F0）；
- 伪指令定义方式（如，MYFLAG BIT F0）。

以上几种形式表示的都是PSW中的位5。

与字节操作指令中累加器ACC用字符“A”表示类似的是，在位操作指令中，位累加器要用字符“C”表示（注：在位操作指令中CY与具体的直接位地址D7H对应）。



布尔处理类指令

- 布尔数据传送指令
- 布尔状态控制指令
- 位逻辑指令
- 位控制转移指令
- 布尔处理示例



布尔数据传送指令

- **mov C,bit ; (C)←(bit)**
- **mov bit,C ; (bit)←(C)**

•这两条指令可以实现指定位地址中的内容与位累加器CY的内容的相互传送。

例 若 (CY) =1, (P3) =1100 0101B, (P1) =0011 0101B。
执行以下指令：

MOV P1.3, C

MOV C, P3.3

MOV P1.2, C

结果为：(CY) =0, P3的内容未变, P1的内容变为 0011 1001B。



布尔状态控制指令

- 位清“0”指令

- CLR C
- CLR bit

例 若 (P1) = 1001 1101B

CLR P1.3

结果为: (P1) = 1001 0101B。

- 位置“1”指令

- SETB C
- SETB bit

例 若 (P1) = 1001 1100B

SETB P1.0

(P1) = 1001 1101B。

- 位取反指令

- CPL C
- CPL bit

例 若 (P1) = 1001 1101B

CPL P1.3

结果为: (P1) = 1001 0101B。

位逻辑指令

- 位逻辑与指令

- `anl C,bit;`
- `anl C,/bit;`

若 $(P1) = 1001\ 1100B$, $(CY) = 1$ 。

`ANL C, P1.0`

结果为：P1 内容不变，而 $(CY) = 0$ 。

- 位逻辑或指令

- `orl C,bit;`
- `orl C,/bit;`

若 $(P1) = 1001\ 1101B$, $(CY) = 0$ 。

`ORL C, P1.0`

结果为：P1 内容不变，而 $(CY) = 1$ 。

空操作指令

- **`nop ; (PC) ← (PC) + 1`**

不执行任何操作，消耗一个机器周期的时间。占用一个字节，常用来实现较短时间的延时。

- **`clr P3.7`**
- **`nop`**
- **`setb P3.7`**
- **`nop`**
- **`clr P3.7`**
- **`nop`**
- **`setb P3.7`**

位控制转移指令

- 判布尔累加器C转移指令
- 判位变量转移指令
- 判位变量并清零转移指令



判布尔累加器C转移指令

- 格式: JC rel; rel:8位相对偏移量

功能: $(PC) \leftarrow (PC) + 2,$

IF (C)=1 THEN $(PC) \leftarrow (PC) + \text{rel}$

ELSE go on

- 格式: JNC rel; rel:8位相对偏移量

功能: $(PC) \leftarrow (PC) + 2,$

IF (C)=0 THEN $(PC) \leftarrow (PC) + \text{rel}$

ELSE go on

判位变量转移指令

- 格式: **JB bit, rel;** **rel:8位相对偏移量**

功能: $(PC) \leftarrow (PC) + 3,$

IF (bit)=1 THEN $(PC) \leftarrow (PC) + rel$

ELSE go on

- 格式: **JNB bit, rel;** **rel:8位相对偏移量**

功能: $(PC) \leftarrow (PC) + 3,$

IF (bit)=0 THEN $(PC) \leftarrow (PC) + rel$

ELSE go on



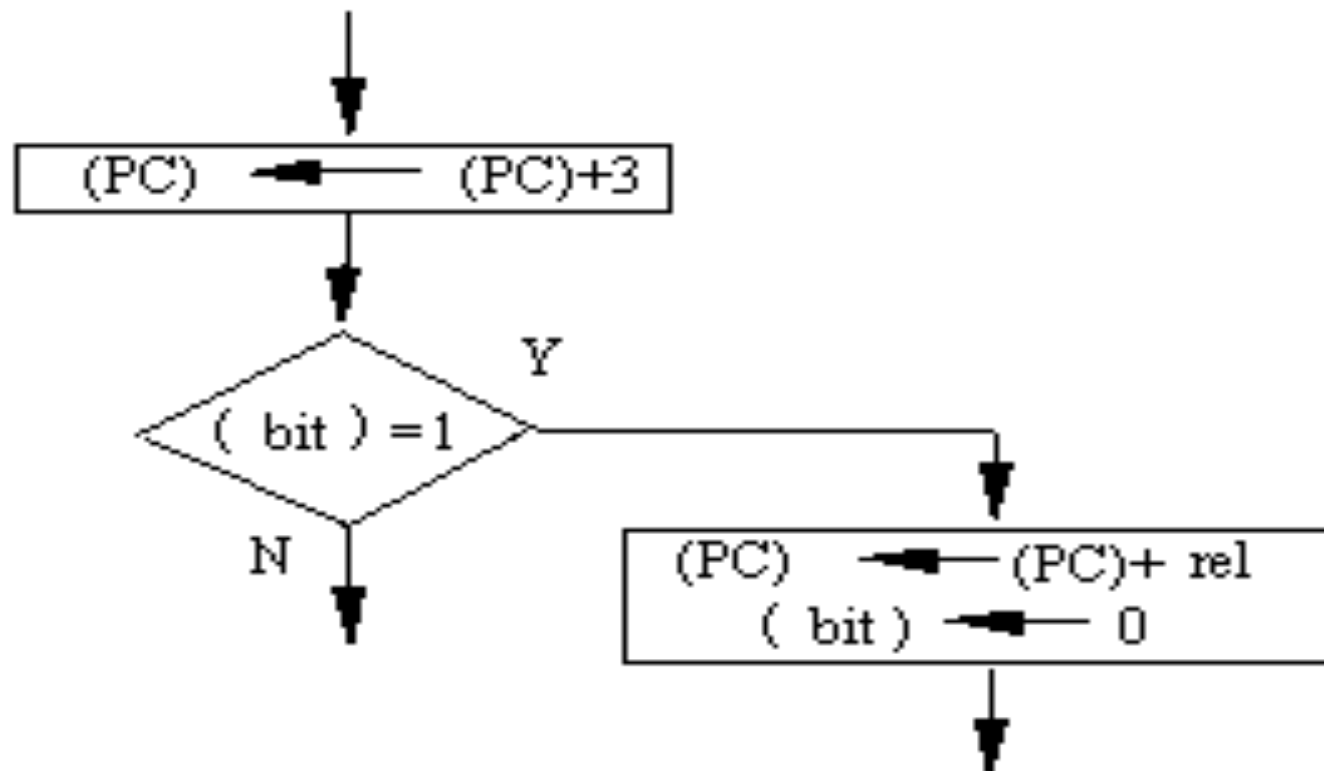
判位变量并清零转移指令

• 格式: JBC bit, rel; rel:8位相对偏移量

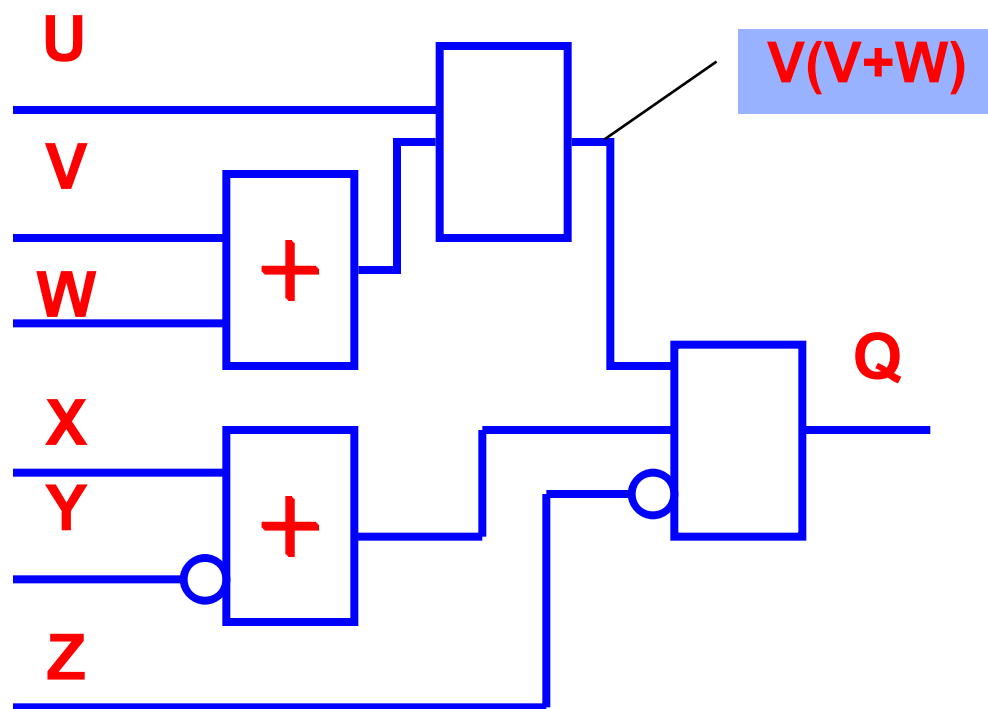
功能: $(PC) \leftarrow (PC) + 3,$

IF (bit)=1 THEN $(PC) \leftarrow (PC) + \text{rel}$ and (bit)=0

ELSE go on



布尔处理示例



$$Q = U \cdot (V + W) \cdot (X + \bar{Y}) \cdot \bar{Z}$$

U BIT 20H.0
V BIT 20H.1
W BIT 20H.2
X BIT 20H.3
Y BIT 20H.4
Z BIT 20H.5
Q BIT 20H.6
ORG 0000H

Start: mov C, V
 orl C, W
 anl C, U
 mov F0, C
 mov C, X
 orl C, /Y
 anl C, F0
 anl C, /Z
 mov Q, C
 sjmp \$
 END



控制转移类指令（17条）

通常情况下，程序的执行是顺序进行的，但也可以根据需要**改变程序的执行顺序**，这种情况称作**程序转移**。

控制程序的转移要利用转移指令。
80C51的转移指令有**无条件转移、条件转移及子程序调用与返回等**。



控制转移类指令

- 无条件转移指令
- 条件转移指令
- 子程序调用与返回指令
- 空操作



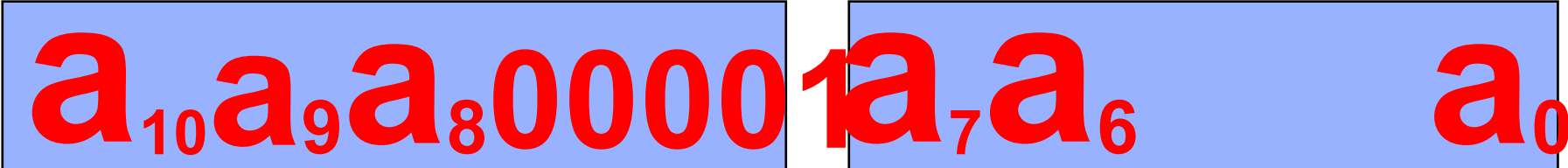
无条件转移指令

- 绝对转移指令
- 长转移指令
- 短转移指令
- 间接转移指令



绝对转移指令

- 格式: **AJMP addr11;**
- 功能: 无条件转向指令中提供的11位地址。

机器码: 

转移范围: 2K, PC_{15~11}不变

指令提供的11位地址与PC当前值的高5位组成16位目标地址，必须设置在包含AJMP指令后第一条指令的第一个字节在内的同一2kb范围。

例: **AJMP Start**

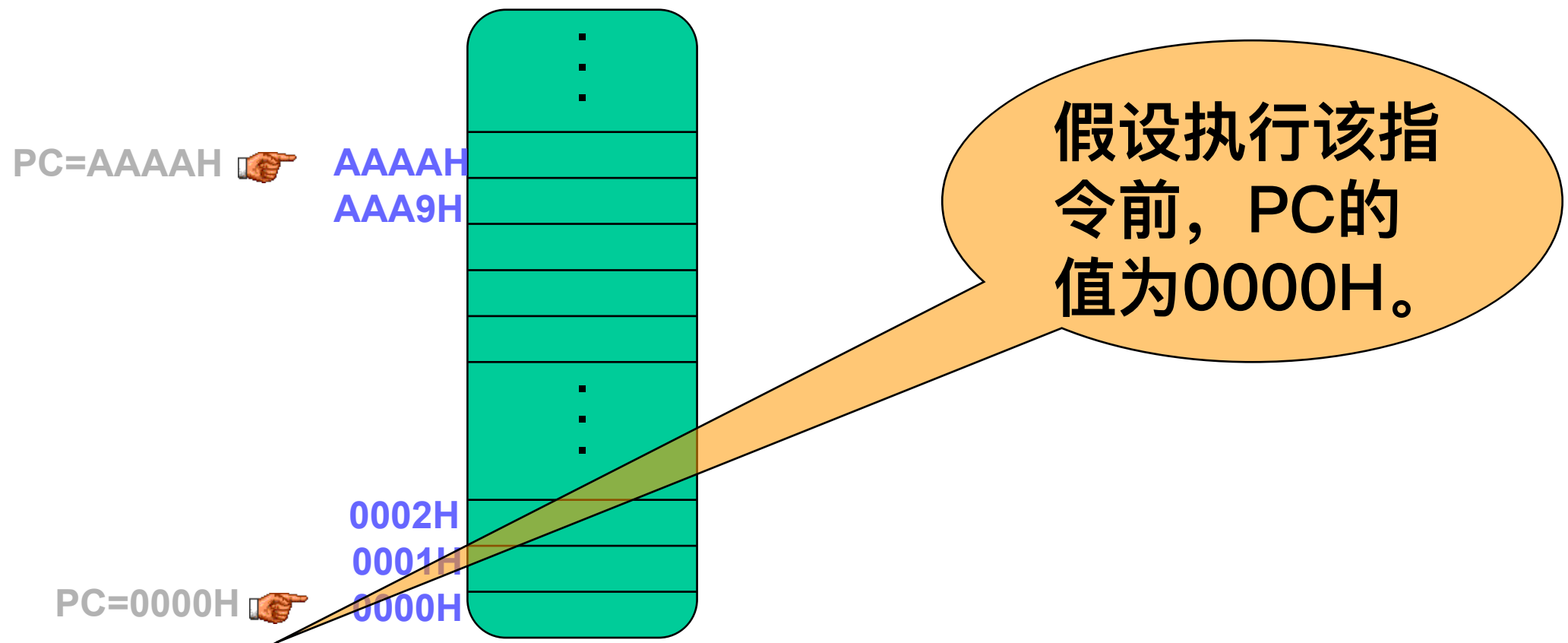
长转移指令

- 格式: **LJMP addr16; (PC)← addr16**
- 功能:无条件转向指令中提供的16位地址。

三字节双周期指令,后两个字节为转移的目标地址.转移范围: 64K

例: **LJMP Start**

长转移指令：LJMP AAAAH ; AAAAH→PC



注意:该指令可以转移到64KB程序存储器中的任意位置。



短转移指令

- 格式: **SJMP rel**; $(PC) \leftarrow (PC) + 2$,
 $(PC) \leftarrow (PC) + \text{rel}$
- 功能: 相对转向指令, rel 为一字节补码形式的相对偏移量.
- 转移范围: **-128 ~ +127**

当前PC前128字节, 或后127字节。

例: **sjmp Loop**

sjmp \$

间接转移指令(散转)

- 格式: **JMP @A+DPTR; (PC)←(A)+(DPTR)**
- 功能: 根据A与DPTR的内容转移, 转移的目标地址可变, 在程序运行时动态决定的。

- 例: **mov DPTR, #JMP_TBL**

jmp @A+DPTR

JMP_TBL:ajmp Lable1

ajmp Lable2

.

.

散转移

JMP @A+DPTR ; $PC \leftarrow (PC) + 1$, $PC \leftarrow (A) + (DPTR)$

该指令具有散转功能，可以代替许多判别跳转指令。其转移地址由数据指针DPTR的16位数和累加器A的8位数进行无符号数相加形成，并直接装入PC。该指令执行时对标志位无影响。

•例 有一段程序如下：

MOV DPTR, #TABLE

JMP @A+DPTR

TABLE: AJMP ROUT0

AJMP ROUT1

AJMP ROUT2

AJMP ROUT3

•当 (A) =00H时，程序将转到 ROUT0处执行；当 (A) =02H时，程序将转到 ROUT1处执行；其余类推。



条件转移指令

- 累加器判零转移指令
- 比较转移指令
- 循环转移指令
- 关于rel

举例：



累加器判零转移指令

- 格式: **JZ rel;** **rel:8位相对偏移量**

功能: **$(PC) \leftarrow (PC) + 2,$**

IF (A)=0 THEN $(PC) \leftarrow (PC) + rel$

ELSE go on

- 格式: **JNZ rel;** **rel:8位相对偏移量**

功能: **$(PC) \leftarrow (PC) + 2,$**

IF (A) \neq 0 THEN $(PC) \leftarrow (PC) + rel$

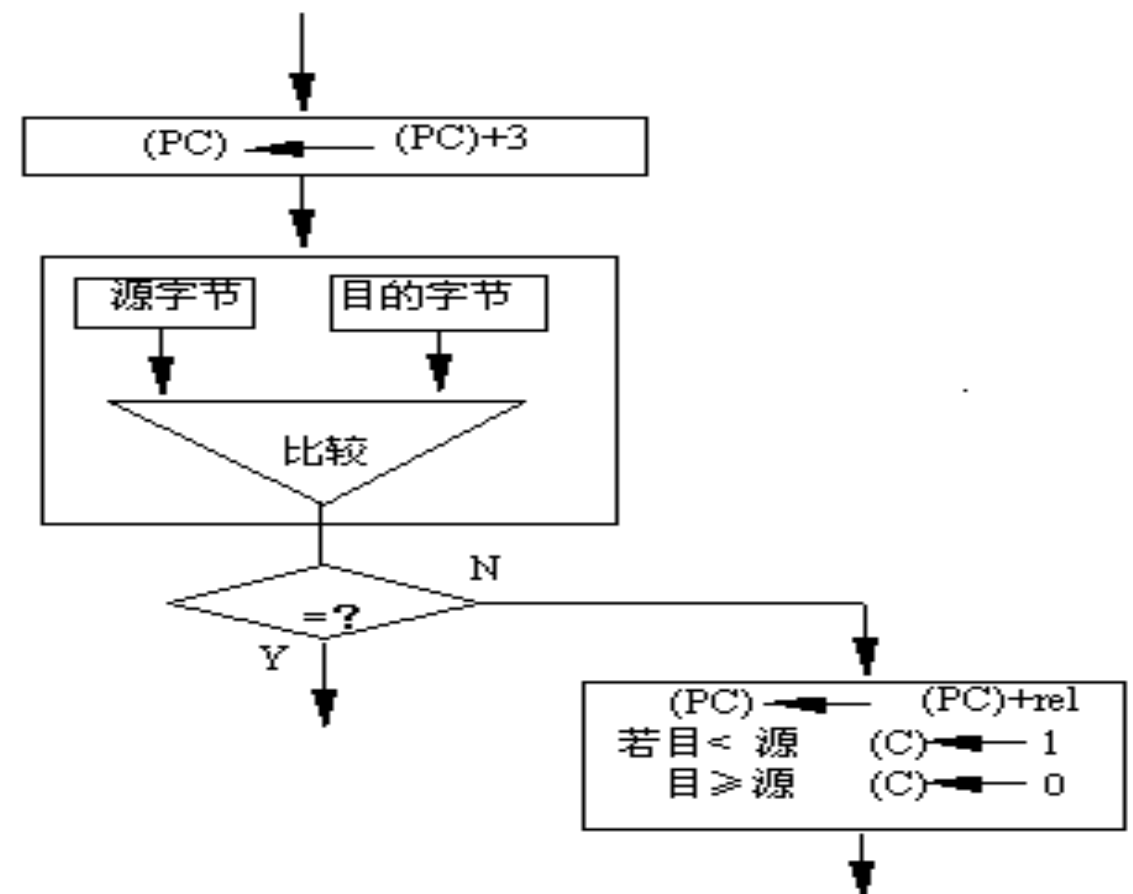
ELSE go on

比较转移指令

- 格式: **CJNE** (目的字节), (源字节), **rel**

功能: 目的字节与源字节比较, 不相等则转移, 同时, 若目的字节 \geq 源字节则(C)=0; 若目的字节 $<$ 源字节则(C)=1。相等则继续执行。

- **cjne A, direct, rel;**
- **cjne A, #data, rel;**
- **cjne Rn, #data, rel;**
- **cjne @Ri, #data, rel**



循环转移指令

- 格式: **DJNZ Rn, rel;** **rel:8位相对偏移量**

功能: $(PC) \leftarrow (PC) + 2$, $(Rn) \leftarrow (Rn) - 1$

IF (Rn) \neq 0 THEN $(PC) \leftarrow (PC) + rel$
ELSE go on

- 格式: **DJNZ direct, rel;** **rel:8位相对偏移量**

功能: $(PC) \leftarrow (PC) + 3$, $(direct) \leftarrow (direct) - 1$

IF (direct) \neq 0 THEN $(PC) \leftarrow (PC) + rel$
ELSE go on

例 有一段程序如下：

MOV 23H, #0AH

CLR A

LOOPX: ADD A, 23H

DJNZ 23H, LOOPX

SJMP \$

该程序执行后：

(A) = 10+9+8+7+6+5+4+3+2+1=37H



关于rel

机器语言中：rel 是 $[-128, +127]$ 内的补码

汇编语言中：rel的书写形式

1、jc loop ;loop是目标地址

2、jc \$-5 ; \$-5 也代表目标地址

3、jc -5 ;是相对偏移量

4、jc +5 ;

5、jc 85H ;



条件转移类指令范例——方案一

将00H~0FH这16个数顺序地置入片内RAM20H~2FH单元中。

```
MOV R0, #20H
MOV R7, #10(0F+1)H
CLR A
LOOP: MOV @R0, A
      INC A
      INC R0
      DJNZ R7, LOOP
      SJMP $
```

还有什么方法实现循环的终止？



条件转移类指令范例——方案二

MOV R0, #20H

MOV R7, #0FH

CLR A

LOOP: MOV @R0, A

INC A

INC R0

CJNE A, #10(0F+1)H, LOOP ; (A)=16?

SJMP \$



子程序调用与返回指令

- 绝对调用指令

- ACALL addr11; $(PC) \leftarrow (PC)+2$,

- $(SP) \leftarrow (SP)+1, ((SP)) \leftarrow (PC_{0\sim7})$,

- $(SP) \leftarrow (SP)+1, ((SP)) \leftarrow (PC_{8\sim15})$,

- $(PC_{0\sim10}) \leftarrow \text{addr11}, (PC_{11\sim15})$ 不变; 2kb地址范围

- 长调用指令

- LCALL addr16

- 返回指令

- RET; $(PC_{8\sim15}) \leftarrow ((SP)), (SP) \leftarrow (SP)-1$

- $(PC_{0\sim7}) \leftarrow ((SP)), (SP) \leftarrow (SP)-1$

- 举例:

例

若 $(SP) = 07H$ ，标号“XADD”表示的实际地址为 $0345H$ ，PC的当前值为 $0123H$ 。

执行指令 `ACALL XADD` 后，

$(PC) + 2 = 0125H$ ，其低8位的 $25H$ 压入堆栈的 $08H$ 单元，其高8位的 $01H$ 压入堆栈的 $09H$ 单元。

$(PC) = 0345H$ ，程序转向目标地址 $0345H$ 处执行。

子程序调用与返回指令举例

ORG 0000H

Start: mov A,#0

acall Sub1

mov A,#0

lcall Sub2

sjmp \$

Sub1:inc A

ret

Sub2:dec A

ret

END



第三單元結束