《单片机与接口技术》

主讲人：李刚

03 指令系统及程序设计

第3章 指令系统及汇编语言程序设计

3.1 寻址方式

3.2 80C51指令系统

3.3 汇编语言程序设计

3.4 C51程序设计

3.1 寻址方式

3.1.1 指令的基本概念

指令：计算机用于控制各功能部件完成指定动作的指示和命令，不同功能指令的有序组合就构成了程序。

80C51系列单片机的指令系统共有111条指令。

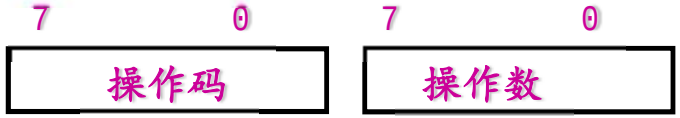
1、指令的格式

指令按字节长度可以分为三种：

单字节指令：

例如： MOV A, R0

双字节指令：

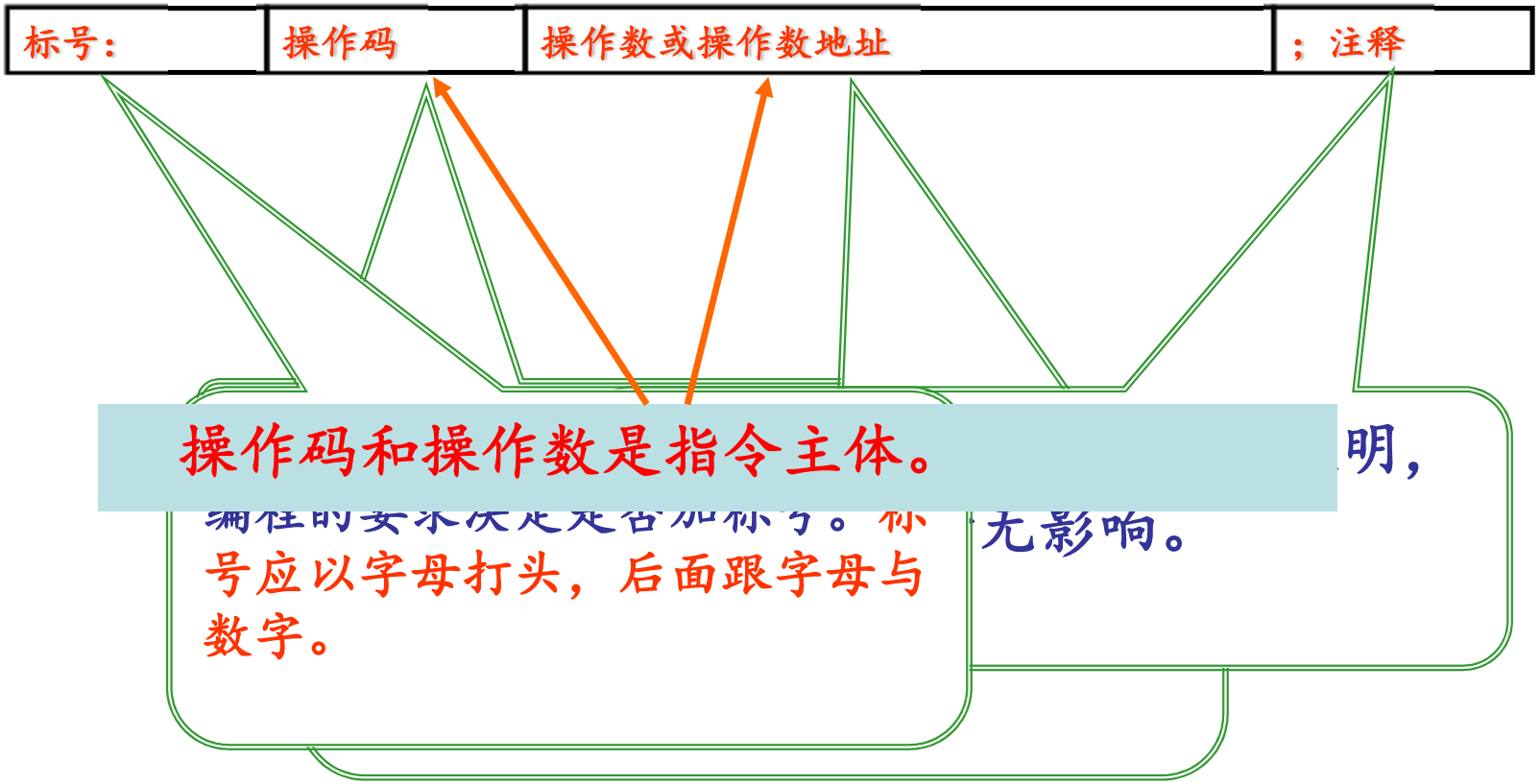


例如：MOV A, 30H

三字节指令：

例如：MOV 30H, #20H

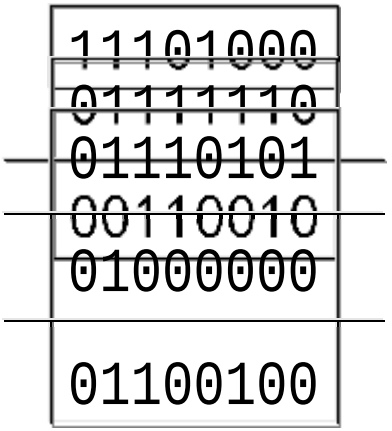
指令的格式：



操作码和操作数是指令主体，称为指令可执行部分，指令表中可查出对应指令代码。

举例：

汇编语言：	机器语言：
MOV A, R0	E8H
MOV R6, #32H	7E 32H
MOV 40H, #100	75 40 64H



2、指令的分类

1、功能分

2、数据传送指令：29条　　例：MOV A,20H

3、算术运算指令：24条　　例：ADD A,20H

逻辑运算指令：24条　　例：ANL A,20H

控制转移指令：17条　　例：SJMP LOOP

位操作指令：17条　　例：CPL P1.0

单字节指令：49条　　例：MOV A,R0

双字节指令：46条　　例：MOV A,20H

三字节指令：16条　　例：MOV 40H,20H

四周期指令：2条　　例：MUL AB

单周期指令：64条

双周期指令：45条

3、 符号约定

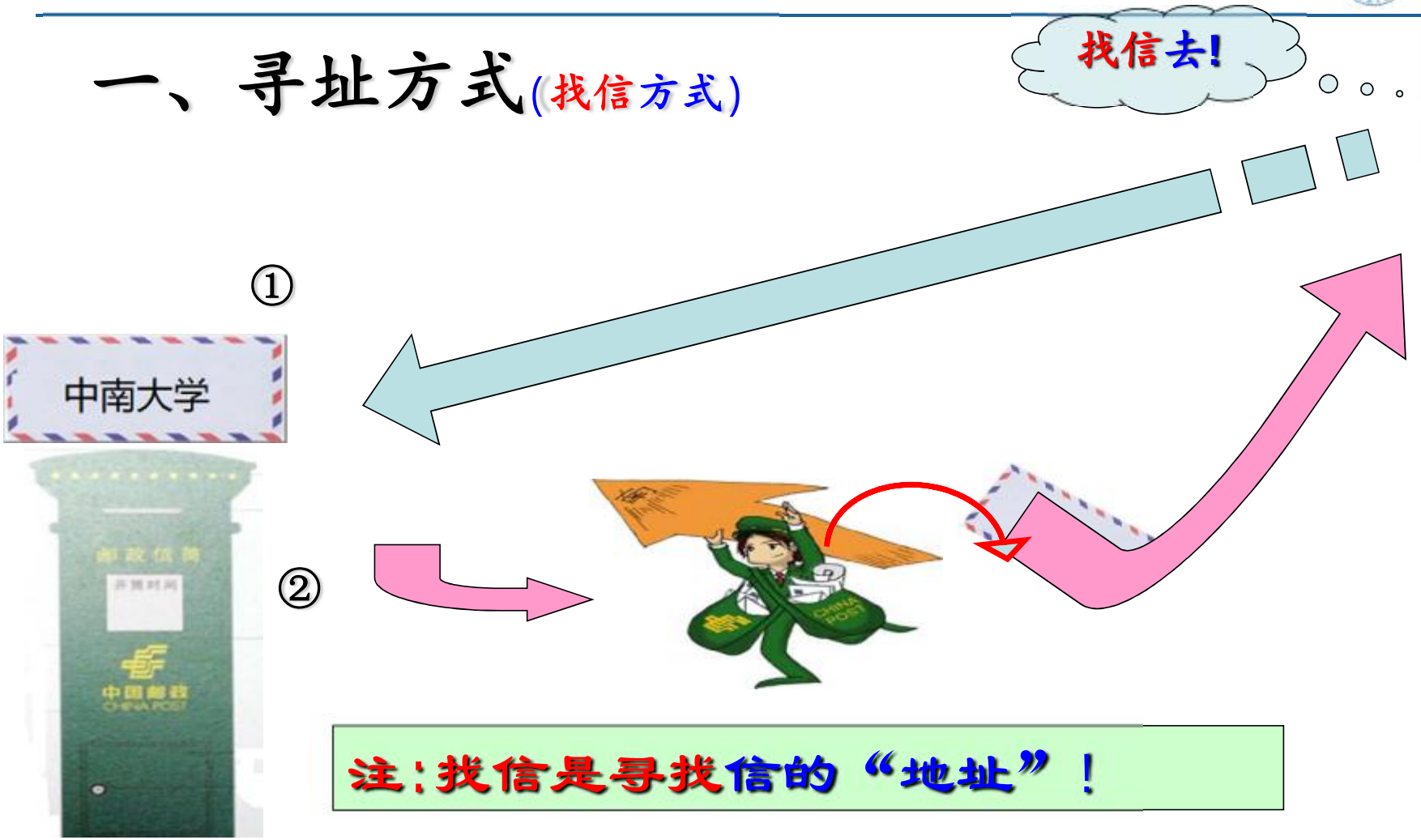
符 号	含 义
Rn	表示当前选定寄存器组的工作寄存器R0~R7
Ri	表示作为间接寻址的地址指针R0~R1
#data	表示8位立即数，即00H~FFH
#data16	表示16位立即数，即0000H~FFFFH
addr16	表示16位地址，用于64K范围内寻址
addr11	表示11位地址，用于2K范围内寻址
direct	8位直接地址，可以是内部RAM区的某单元或某专用功能寄存器的地址
Rel	带符号的8位偏移量（-128~+127）
Bit	位寻址区的直接寻址位
（X）	X地址单元中的内容，或X作为间接寻址寄存器时所指单元的内容
←	将 ← 后面的内容传送到前面去

3.1.2 寻址方式

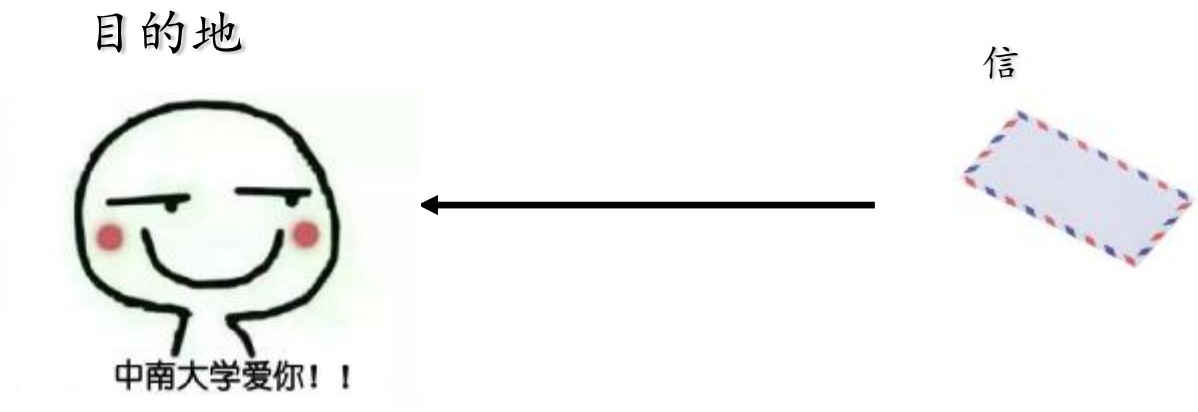
取得操作数的地址的方法叫做寻址方式。寻址方式与计算机的存储空间结构是密切相关的。灵活运用各种寻址方式，可以大大的提高程序的运行效率。

直接寻址
立即寻址
寄存器寻址
寄存器间接寻址
相对寻址
变址寻址
位寻址

一、寻址方式(找信方式)



立即数寻址



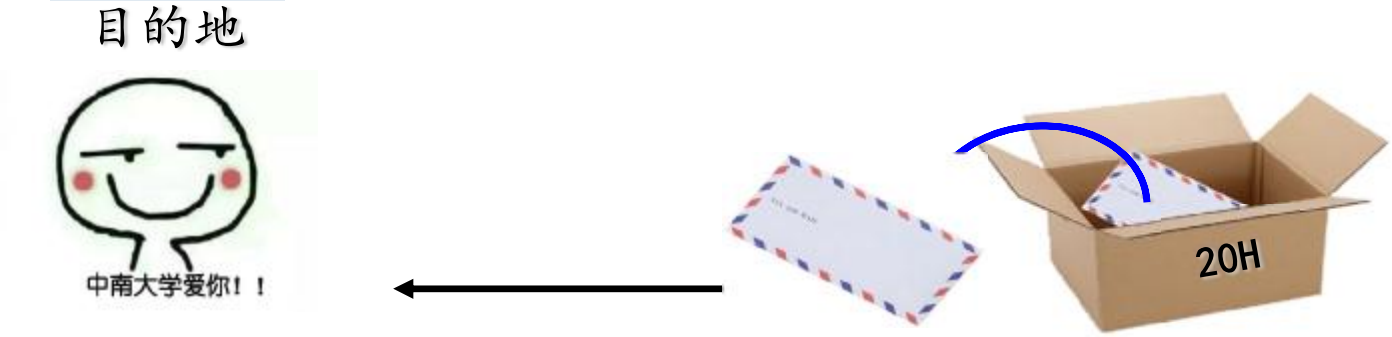
目的地

操作数

#55H

```
MOV  P1, #55H
MOV  A,  #01H
```

直接寻址



注：寻址是寻操作数的“地址”！

直接给出地址
20H——直接寻址

操作数

内部RAM
或SFR区
XXH

20H

P1是符号地址
也是直接寻址

MOV P1, 20H

寄存器寻址

目的地

R0-R7
A
B
DPTR

操作数

寄存器
XXH

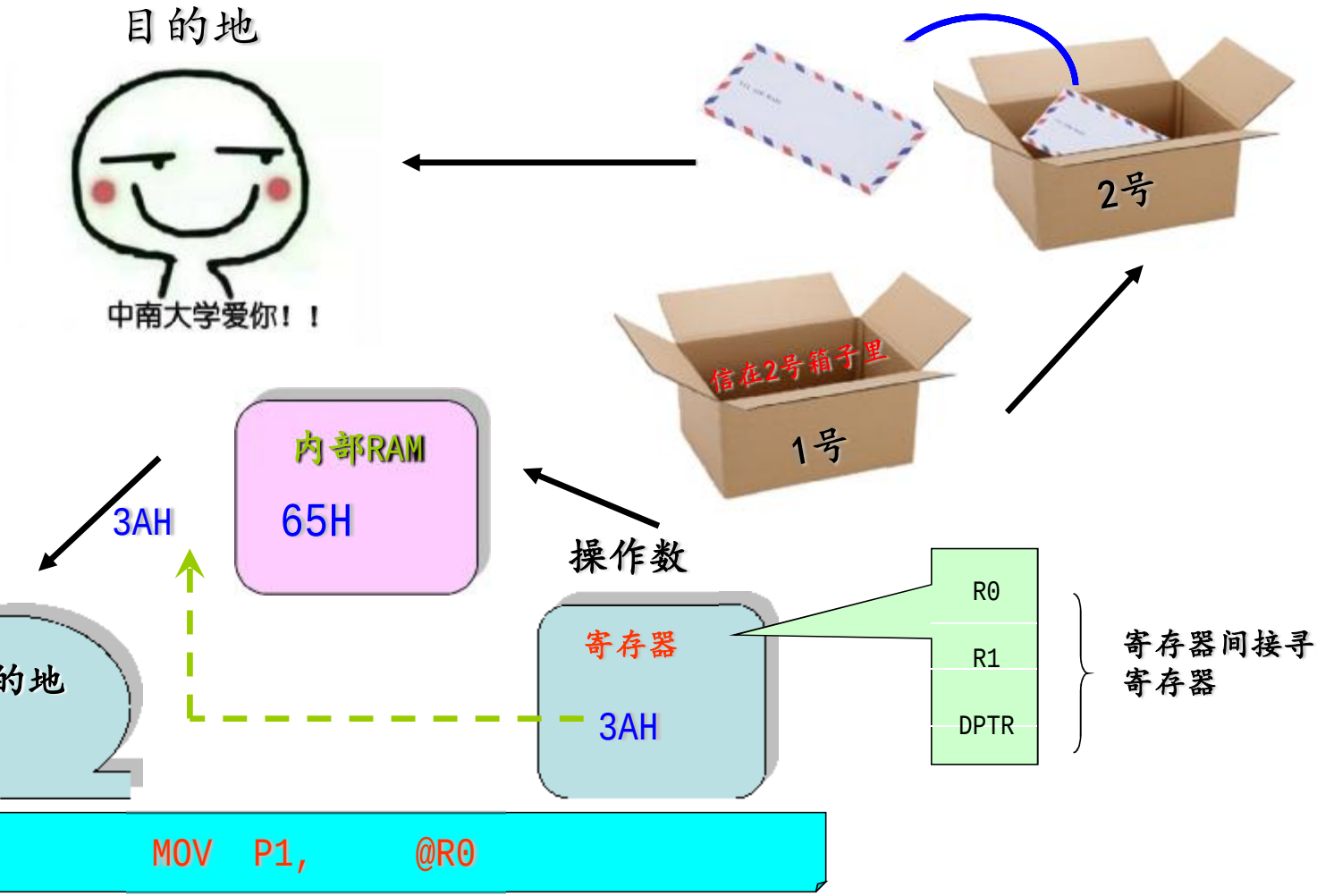
目的地

目的地址也可以是寄存器。

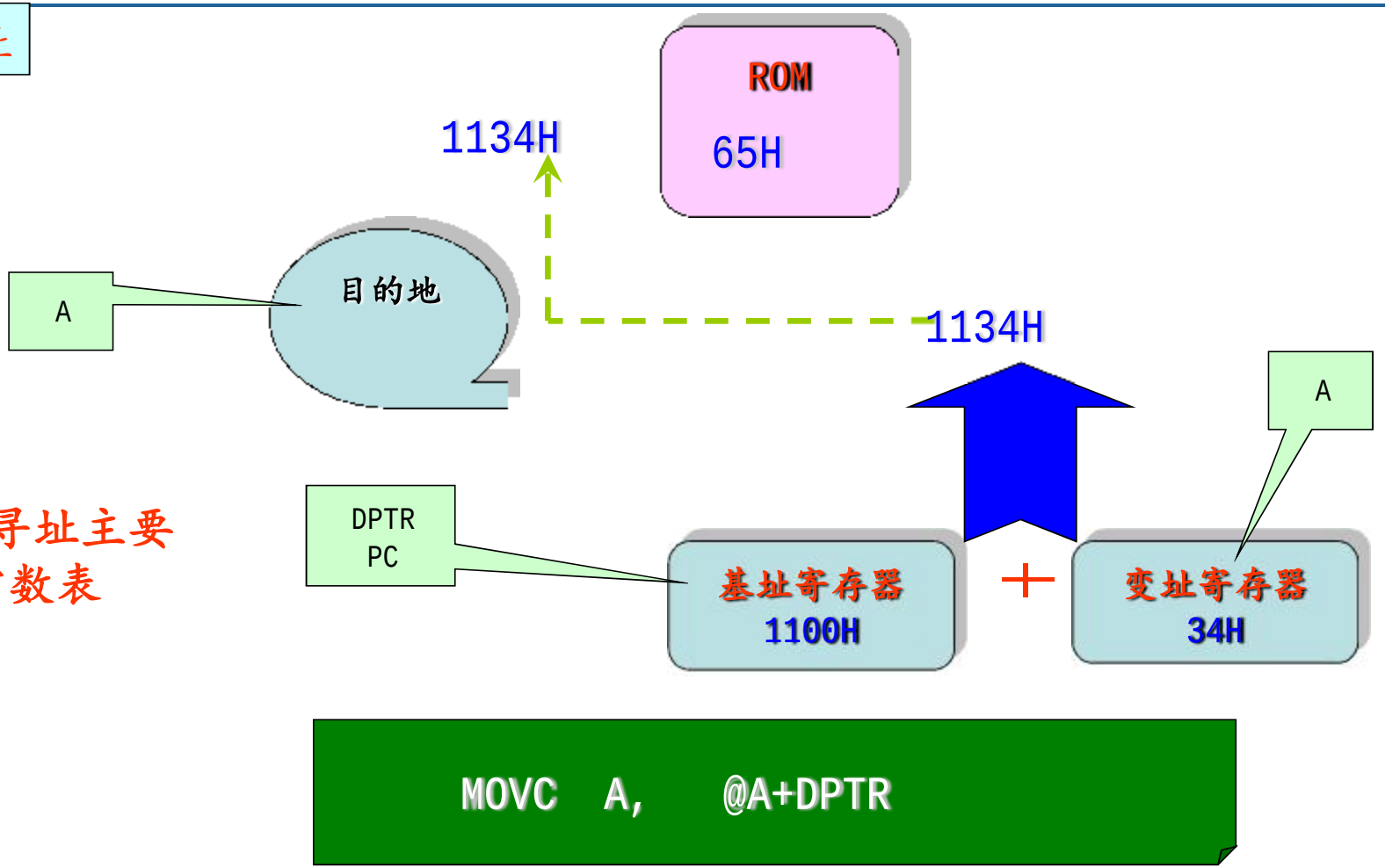
```
MOV  P1,  A
MOV  R3,  #1
```

寄存器间接寻址

寄存器间接寻址适合处理大批同类型的数据



变址寻址



变址寻址主要
用于查常数表

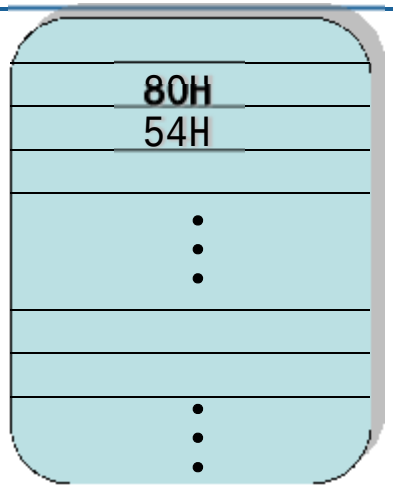
相对寻址

当前PC值

PC ☞ 2002H

转移到目的地址

PC ☞ 2056H



2002H + 54H =2056H

SJMP 54H

位寻址

单元地址	MSB ◄---------- 位地址 ----------► LSB							
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00

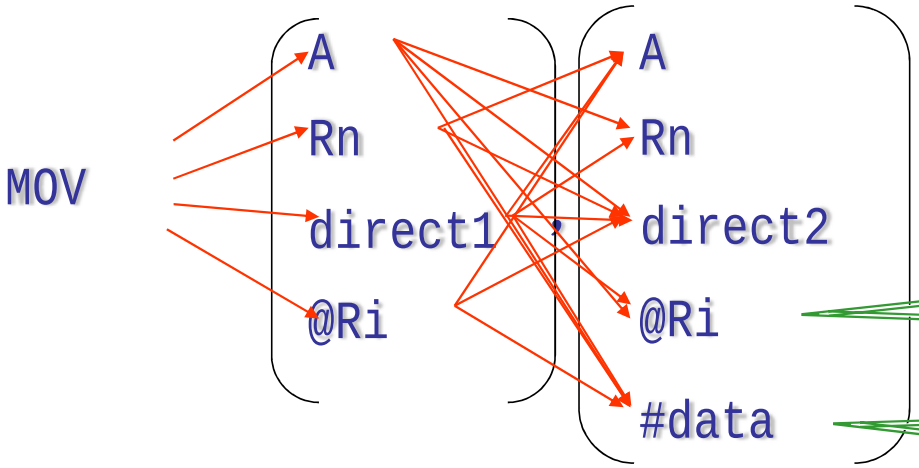
SETB 3DH

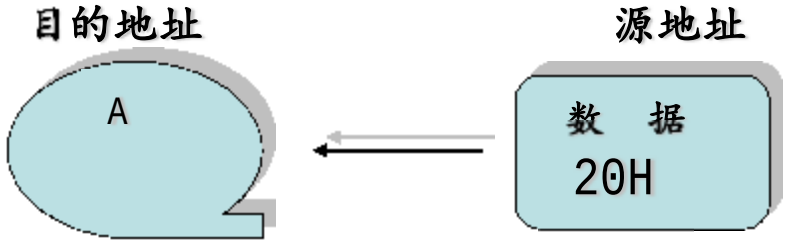
3.2 MCS-51单片机指令系统

3.2.1 数据传送指令

——8位数据传送指令(15条)

MOV <目的操作数>,<源操作数>





MOV

源操作数与目的操作数不能同时与通用寄存器相关

立即数只能做源操作数

MOV R1,R2

MOV R0,@R1

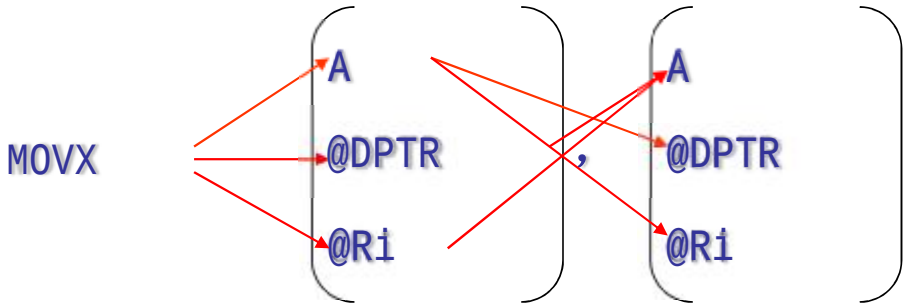
——16位数据传送指令（1条）

MOV DPTR, #data16

例： MOV DPTR, #2000H

——外部数据传送指令(4条)

MOVX <目的操作数>,<源操作数>



输出指令

MOVX @R0 @A

例：将片外RAM中2000H单元中的内容取 出，传送到片外2003H单元中去。

```
MOV   DPTR, #2000H
MOVX  A, @DPTR
MOV   DPTR, #2003H
MOVX  @DPTR, A
```

注意MOV和MOVX的使用区别

片内传送与片外传送分别用什么指令？

如何将片内30H单元内容传送至片外61H单元？

——交换类指令(5条)

1)字节交换指令(3条)

XCH A , direct / Rn / @Ri

A中内容与RAM内容互换

```
XCH   A , R5
```

2)低半字节交换指令(1条)

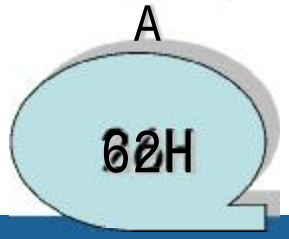
```
XCHD   A ,  @Ri
```

A中低4位与RAM低4位互换

```
XCHD   A , @R0
```

3)累加器A中高4位和低4位交换(1条)

```
SWAP   A
```



——查表类指令(2条)

MOVC A，@A+PC

MOVC A，@A+DPTR

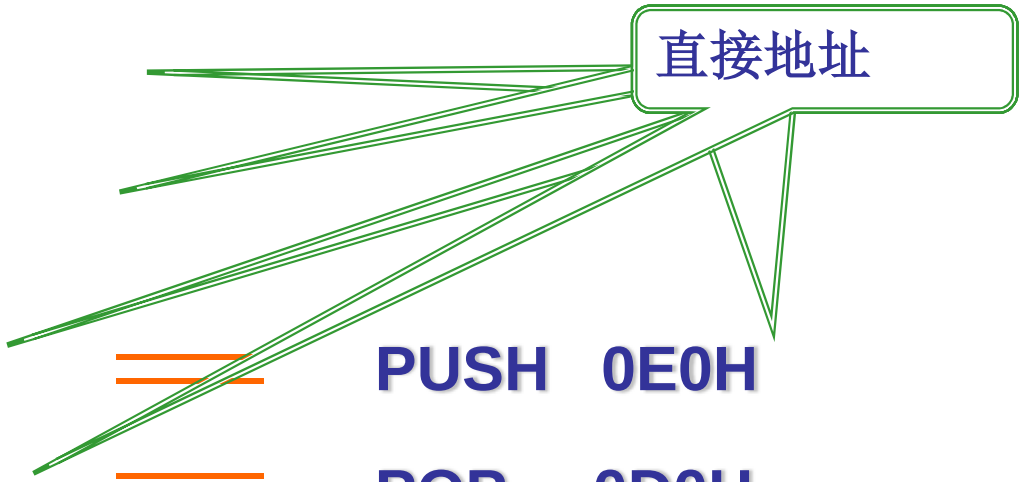
——堆栈操作指令(2条)

PUSH direct

POP direct

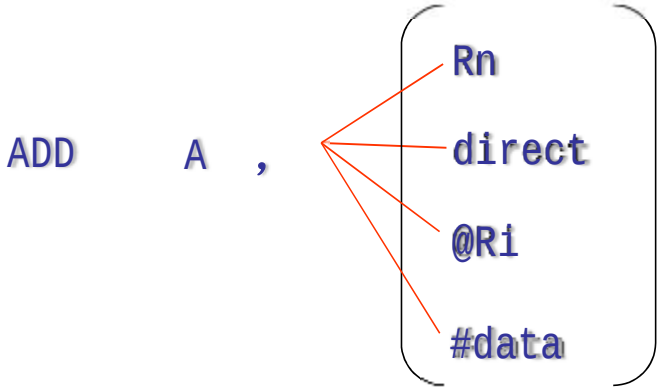
PUSH Acc = PUSH 0E0H

POP PSW = POP 0D0H



3.2.2 算术运算类指令（24条）

——普通加法指令（4条）



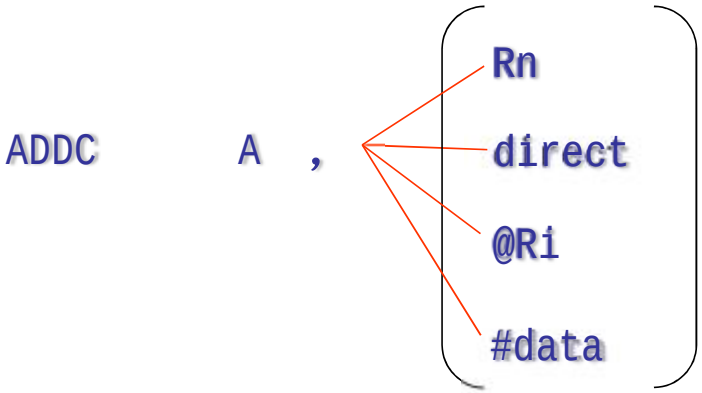

```
ADD A, Rn
ADD A, direct
ADD A,  @Ri
ADD A, #data
```

注意：1.本指令影响Cy，AC，Ov，P等标志。

2.求和操作既可看成是有符号数运算，也可看成是无符号数运算，完全由程序员编程时自行设定。运算结果在A中。

3. Cy 按照无符号数运算规则影响， Ov按照有符号数运算规则影响。

——带进位加法指令（4条）

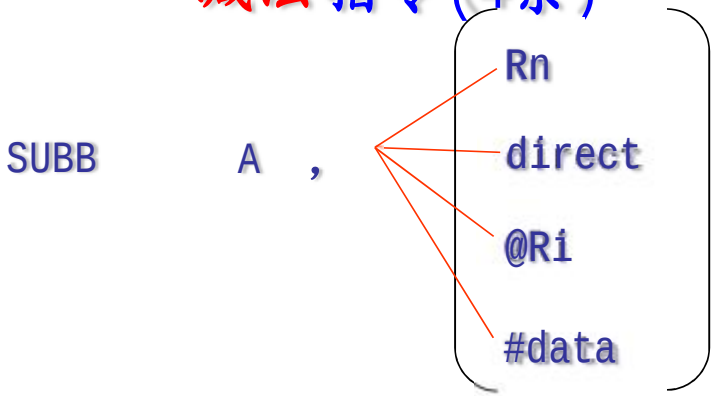



```
ADDC  A, Rn
ADDC A, direct
ADDC A,  @ Ri
ADDC A, #data
```

除相加时要考虑进位外，其余操作与前面ADD相同。

例：
```
MOV   A , #03H
ADD   A , #26H
```

——减法指令(4条)




```
SUBB  A, Rn
SUBB A, direct
SUBB A, @Ri
SUBB A, #data
```

1. 减法指令影响OV、CY、AC、P标志位。

2. 减法指令是带借位的，若不需要带借位减，可在减法指令前先清除借位位。

例：
```
MOV   A , #78H
CLR    C
SUBB A , #26H
```

——加1指令(5条)

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

注意：只有第1条指令影响P标志，其它指令对标志位无影响。

——减1指令(4条)

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

注意：1．只有第1条指令影响P标志，其它指令对标志位无影响。

2．DPTR只有加1指令，无减1指令

——乘法指令(1条)

MUL AB

两个8位操作数A、B相乘，形成16位的积，其中A为积的低位而B为积的高位。

若积大于255，溢出标志位OV置位，而Cy总是0。

——除法指令(1条)

DIV AB

两个8位操作数相除，其中A为被除数而B为除数。运算结果仍然在AB中，其中A为商而B为余数。

Cy和OV均清0。只有当除数B为0时，A和B中的内容为不确定值，此时OV置位。

——十进制调整指令(1条)

DA A

1. 此指令用来对BCD码的加法运算结果自动进行修正，但对BCD码的减法运算不能用此指令进行修正。

2. 在进行BCD码运算时，加法指令ADD或ADDC后面要紧跟一条十进制调整指令。

3. DA指令只影响进位标志CY。

例 ：设ACC中为压缩BCD码56H，R3中为压缩BCD码67H，且Cy=1，求ACC与R3的压缩BCD码之和。

```
MOV     A, #56H
MOV     R3, #67H
ADDC  A, R3
DA      A
```

```
(A) =01010110
(R3)=01100111
+)(Cy)=00000001
-----------------
和    =10111110
调整 +)  01100110
-----------------
1   00100100
```

结果BCD码为124

3.2.3逻辑及移位类指令

——逻辑与指令（6条）

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

例：ANL A, R4

	1	0	0	0	1	1	1	1	A
·)	1	0	0	1	0	1	1	0	R4
	1	0	0	0	0	1	1	0	A

1. 两个操作数按位进行与运算。

ANL P1, #11000000B

2. 指令前4条结果送入A中，执行后影响P。后2条指令结果送入直接地址单元中。

P1中高3位不变，低5位清0。

——逻辑或指令（6条）

例：ORL A, R4

```
ORL   A, Rn
ORL   A, @Ri
ORL   A, #data
ORL   A, direct
ORL   direct, A
ORL   direct, #data
```

	1	0	1	1	1	0	1	1	A
+)	1	0	1	1	0	0	0	1	R4
	1	0	1	1	1	0	1	1	A

1. 两个操作数按位进行或运算。

ORL P1, #11100000B

2. 指令前4条结果送入A中，执行后影响P。后2条指令结果送入直接地址单元中。

P1中高3位置1，低5位不变。

——逻辑异或指令（6条）

例：XRL A, R4

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

	1	0	1	0	0	1	1	1	A
⊕)	1	0	0	0	0	1	1	1	R4
	0	0	1	0	0	0	0	0	A

1. 两个操作数按位进行异或运算。

XRL P1, #11000000B

2. 指令前4条结果送入A中，执行后影响P。后2条指令结果送入直接地址单元中。

P1中高3位取反，低5位不变。

——累加器A清0和取反指令

求反指令（1条）

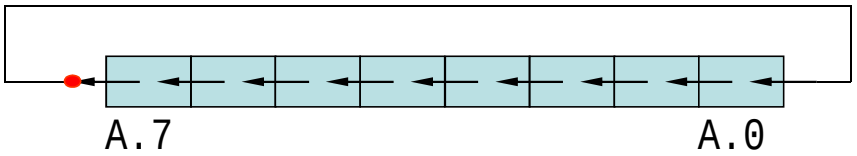
```
CPL   A          ；累加器中的内容按位取反
```

清零指令（1条）

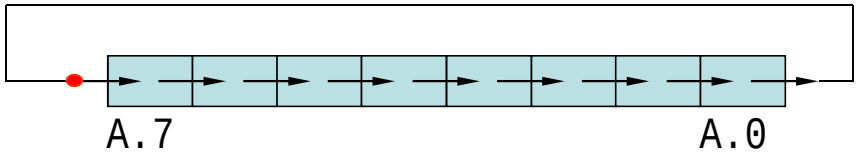
```
CLR   A          ；累加器中的内容清0
```

——循环移位指令

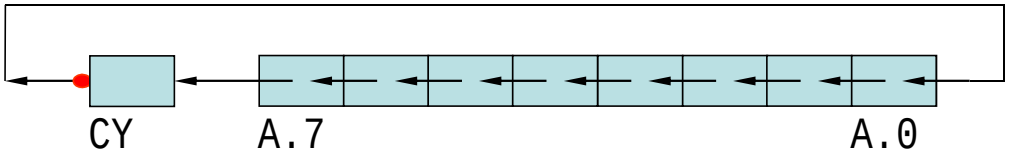
RL A



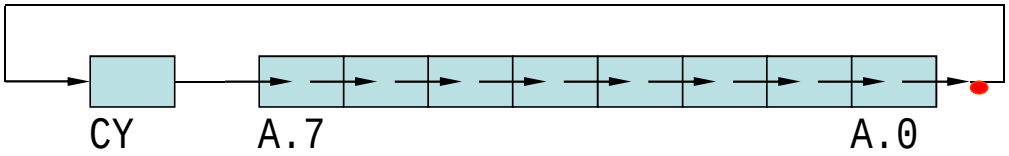
RR A



RLC A



RRC A



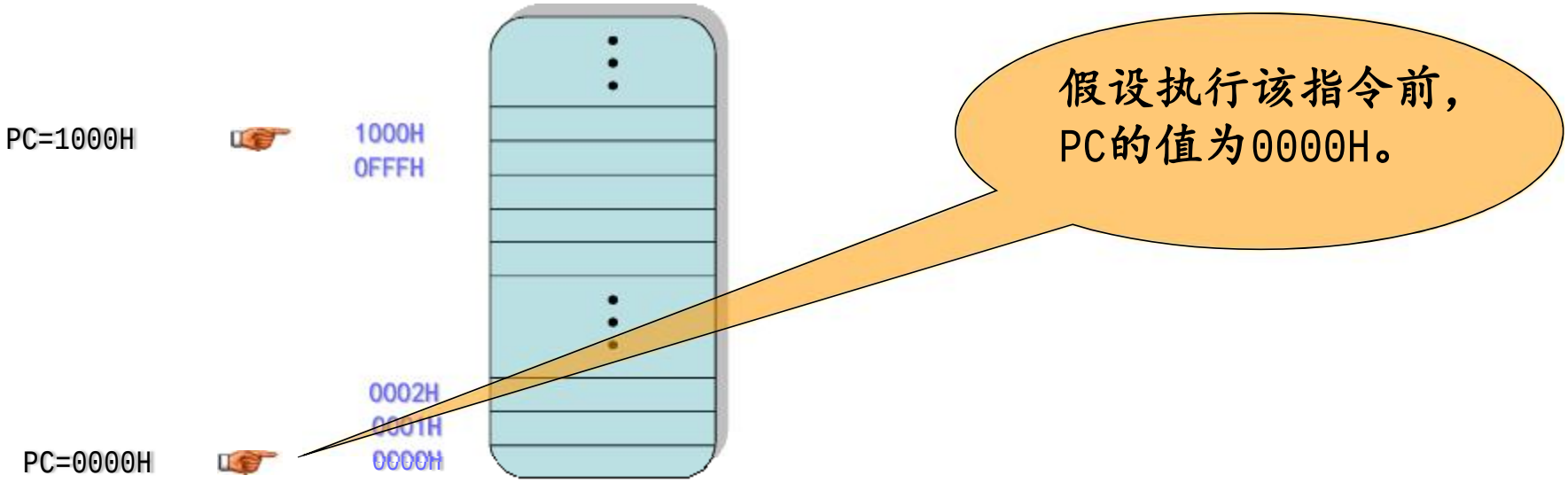
注意： 执行带进位的循环移位指令之前，必须根据程序要求给CY置位或清零。

3.2.4 控制转移类指令

——无条件转移指令(4条)

1) 长转移指令：LJMP addr16 ； addr16→PC

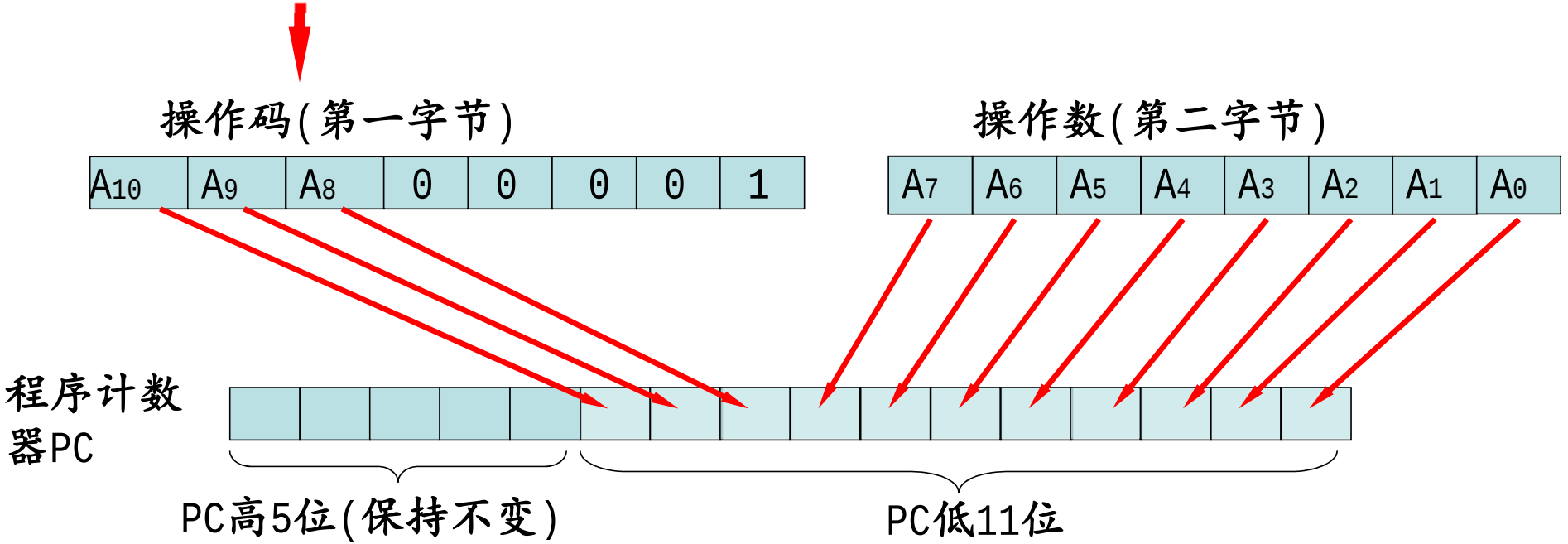
例：LJMP 1000H



注意：该指令可以转移到64 KB程序存储器中的任意位置。

2)绝对转移指令

AJMP addr11 ； PC+2→PC， addr11 →PC.10~PC.0



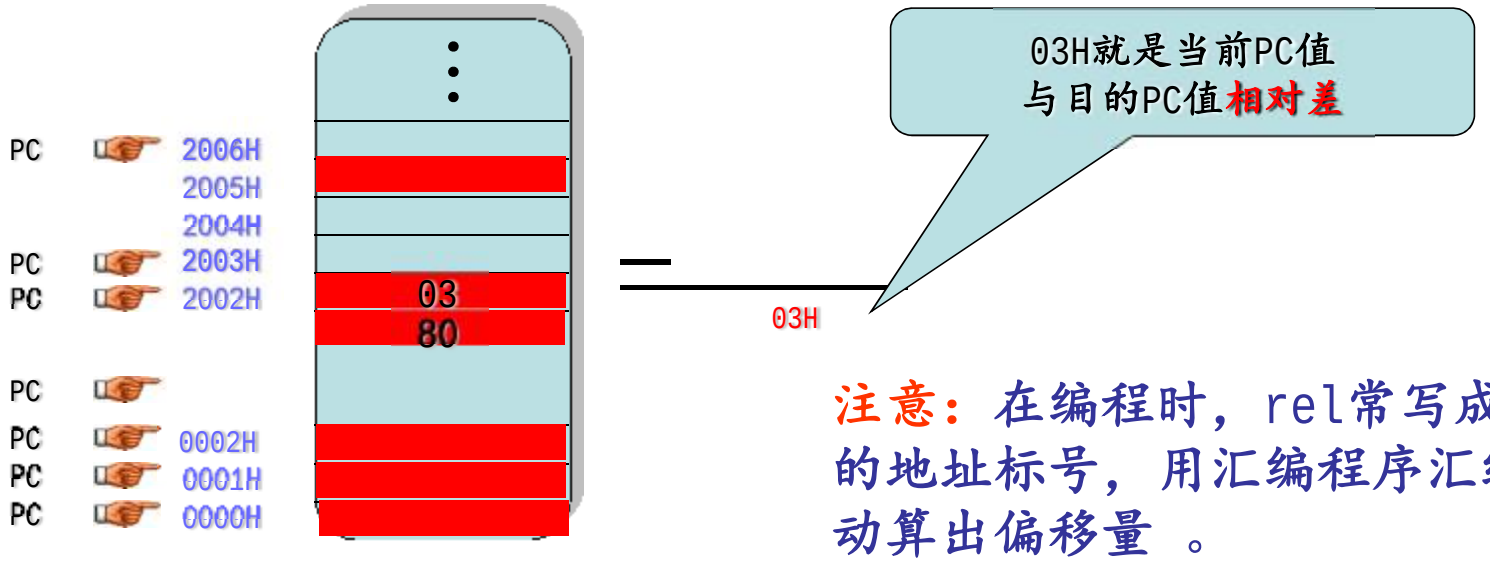
11位转移地址的形成示意图

注意： 高5位不变，则意味着只能在该转移指令的下一条指令所在的2K地址范围内转移。

3)相对转移指令

SJMP rel

例： SJMP 03H → 机器码 80 03



注意：在编程时，rel常写成转移后的目的地址标号，用汇编程序汇编时，能自动算出偏移量 。

例：SJMP LOOP

4)散转指令

助记符格式	机器码(B)	相应操作	机器周期
JMP @A+DPTR	01110011	PC ← A+DPTR	2

例： 根据累加器A中的键值，设计命令键操作程序入口跳转表。程序如下：

```
        CLR   C
        RLC   A
        MOV   DPTR, #JPTAB
        JMP   @A+DPTR
JPTAB:  AJMP  CCS0
        AJMP  CCS1
        AJMP  CCS2
```

——条件转移指令(8条)

1)累加器A判0指令(2条)

助记符格式	机器码(B)	相应操作	机器周期
JZ rel	0110000	若A=0，则PC←PC+rel，否则程序顺序执行	2
JNZ rel	01110000	若A≠0，则PC←PC+rel，否则程序顺序执行	2

2)减1非零转移指令(2条)

助记符格式	机器码(B)	相应操作	机器周期
DJNZ Rn,rel	11011rrr rel	Rn←Rn-1，若Rn≠0，则PC←PC+rel，否则顺序执行	2
DJNZ direct,rel	11010101 direct rel	（direct）←（direct）-1，若（direct）≠0，则PC←PC+rel，否则顺序执行	2

3)比较转移指令(4条)

助记符格式	机器码(B)	相应操作	机器周期
CJNE A,#data,rel	10110100 data rel	若A≠#data，则PC←PC+rel，否则顺序执行； 若 A<#data，则CY=1，否则CY=0	2
CJNE Rn,#data,rel	10111rrr data rel	若 Rn≠#data，则PC←PC+rel，否则顺序执行； 若Rn<#data，则CY=1，否则CY=0	2
CJNE @Ri,#data,rel	1011011i data rel	若(Ri)≠#data，则PC←PC+rel，否则顺序执行；若(Ri)<#data,则CY=1，否则CY=0	2
CJNE A,direct,rel	10110101 direct rel	若A≠(direct)，则PC←PC+rel，否则顺序执行；若A<(direct)，则CY=1，否则CY=0	2

——调用和返回指令(8条)

1）绝对调用指令（1条）

助记符格式	机器码(B)	相应操作	机器周期
ACALL addr11	$a_{10}a_9a_8$10001 $addr_{7\sim0}$	PC ← PC+2 SP ← SP+1,(SP)← $PC_{0\sim7}$ SP ← SP+1,(SP)← $PC_{8\sim15}$ $PC_{0\sim10}$ ← addr11	2

2）长调用指令（1条）

助记符格式	机器码(B)	相应操作	机器周期
LCALL addr16	00010010 $addr_{15\sim8}$ $addr_{7\sim0}$	PC← PC+3 SP← SP+1,SP← $PC_{0\sim7}$ SP← SP+1,SP← $PC_{8\sim15}$ PC← addr16	2

3)返回指令

助记符格式	机器码(B)	相应操作	机器周期
RET	00100010	$PC_{8\sim15} \leftarrow (SP)$, SP ← SP-1 $PC_{0\sim7} \leftarrow (SP)$, SP ← SP-1 子程序返回指令	2
RETI	00110010	$PC_{8\sim15} \leftarrow SP$, SP ← SP-1 $PC_{0\sim7} \leftarrow SP$, SP ← SP-1 中断返回指令	2

4)空操作指令

助记符格式	机器码(B)	相应操作	指令说明
NOP	00000000	空操作	消耗1个机器周期

3.2.5 位操作类指令

1.位传送指令

助记符格式	机器码(B)	相应操作	指令说明	机器周期
MOV C,bit	10100010	CY¬ bit	位传送指令，结果影响CY标志	2
MOV bit,C	10010010	bit¬ CY	位传送指令，结果不影响PSW	2

2.位置位和位清零指令

助记符格式	机器码(B)	相应操作	指令说明	机器周期
CLR C	11000011	CY¬ 0	位清0指令，结果影响CY标志	1
CLR bit	11000010 bit	bit¬ 0	位清0指令，结果不影响PSW	1
SETB C	11010011	CY¬1	位置1指令，结果影响CY标志	1
SETB bit	11010010 bit	bit¬ 1	位置1指令，结果不影响PSW	1

3.位运算指令

助记符格式	机器码(B)	相应操作	指令说明	机器周期
ANL C,bit	10000010 bit	CY¬ CY∧bit	位与指令	2
ANL C,/bit	10110010 bit	CY¬ CY∧$\overline{bit}$	位与指令	2
ORL C,bit	01110010 bit	CY¬ CY∨bit	位或指令	2
ORL C,/bit	10100010 bit	CY¬ CY∨$\overline{bit}$	位或指令	2
CPL C	10110011	CY¬ $\overline{CY}$	位取反指令	2
CPL bit	10110010	bit¬ $\overline{bit}$	位取反指令，结果不影响CY	2

4.判位转移指令

助记符格式	机器码(B)	相应操作	机器周期
JB bit,rel	00100000 bit rel	若bit=1，则PC← PC+3+rel，否则顺序执行	2
JNB bit,rel	00110000 bit rel	若bit=0，则PC← PC+3+rel，否则顺序执行	2
JBC bit,rel	00010000 bit rel	若 bit=1，则PC← PC+3+rel，bit← 0，否则顺序执行	2

5.判CY标志转移指令

助记符格式	机器码(B)	相应操作	机器周期
JC rel	01000000 rel	若CY=1，则PC← PC+2+rel，否则顺序执行	2
JNC rel	01010000 rel	若CY≠1，则PC← PC+2+rel，否则顺序执行	2

3.2.6 伪指令

1. 什么是伪指令

- 伪指令又称汇编程序控制译码指令，属说明性汇编指令。
- “伪”字体现在汇编时不产生机器指令代码，不影响程序的执行，仅产生供汇编时用的某些命令，在汇编时执行某些特殊操作。
- 如END表示编译到此结束。

（1）定位伪指令

ORG 地址表达式

例如：ORG 1000H ；

MOV A，#20H

规定程序块或数据块存放的起始位置。

表示指令

MOV A，#20H

存放于1000H开始的单元。

（2）定义字节数据伪指令

DB 字节数据表

表示从1000H单元开始的地方存放数据2BH，0A0H，41H,42H,43H（字串ABC的ASCII码），08H

例如：

```
        ORG   1000H
TAB:  DB     2BH, 0A0H, ‘ABC’, 2*4 ;
```

（3）定义字数据伪指令

DW 字数据表

例如：

```
        ORG 1000H
DATA: DW  324AH, 3CH
```

表示从1000H单元开始的地方存放数据32H，4AH，00H，3CH（3CH以字的形式表示为003CH）

（4）定义空间伪指令

DS 表达式

例如：

```
        ORG  1000H
BUF:  DS  50
TAB:  DB  22H
```

从指定的地址开始，保留若干个间。

从1000H开始的地方预留50（1000H~1031H）个存储字节空间。

22H存放在1032H单元

（5）符号定义伪指令

符号名 EQU 表达式

例如：

```
LEN  EQU  21H
     …
     MOV  A, #LEN
```

将符号定义为表达式的值，只能定义单字节数据，且必须先定义后使用。

执行指令后，累加器A中的值为21H

（6）位定义伪指令

符号名 BIT 位地址

例如：

```
SDA  BIT  P1.0
```

（7）汇编结束伪指令

```
END
```

汇编语言源程序结束标志，用于整个汇编语言程序的末尾处。

3.3 汇编语言程序设计

3.3.1 程序编制的基本知识

1、程序设计语言

单片机程序设计语言有机器语言、汇编语言和高级语言。

机器语言（Machine Language）是指直接用机器码编写程序、能够为计算机直接执行的机器级语言。CPU最终执行的全部是机器语言。

汇编语言（Assembly Language）是指用指令助记符代替机器码的编程语言。汇编语言程序结构简单，执行速度快，程序易优化，编译后占用存储空间小，是单片机应用系统开发中最常用的程序设计语言。

高级语言（High-Level Language）是在汇编语言的基础上用自然语言的语句来编写程序，例如PL/M-51、Franklin C51、MBASIC 51等，程序可读性强，通用性好，适用于不熟悉单片机指令系统的的用户。

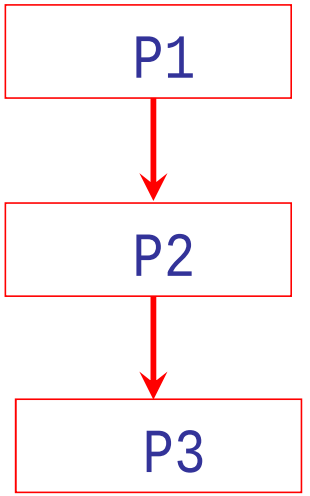
单片机汇编语言程序设计的基本步骤如下：

1. 题意分析
2. 画出程序流程图
3. 分配内存工作区及有关端口地址
4. 编制汇编源程序
5. 仿真、调试程序
6. 固化程序

3.3.2 基本程序结构

1、顺序程序

顺序程序是一种简单程序，它是最简单、最基本的程序结构，其特点是按指令的排列顺序一条条地执行，直到全部指令执行完毕为止。不管多么复杂的程序，总是由若干顺序程序段所组成的。

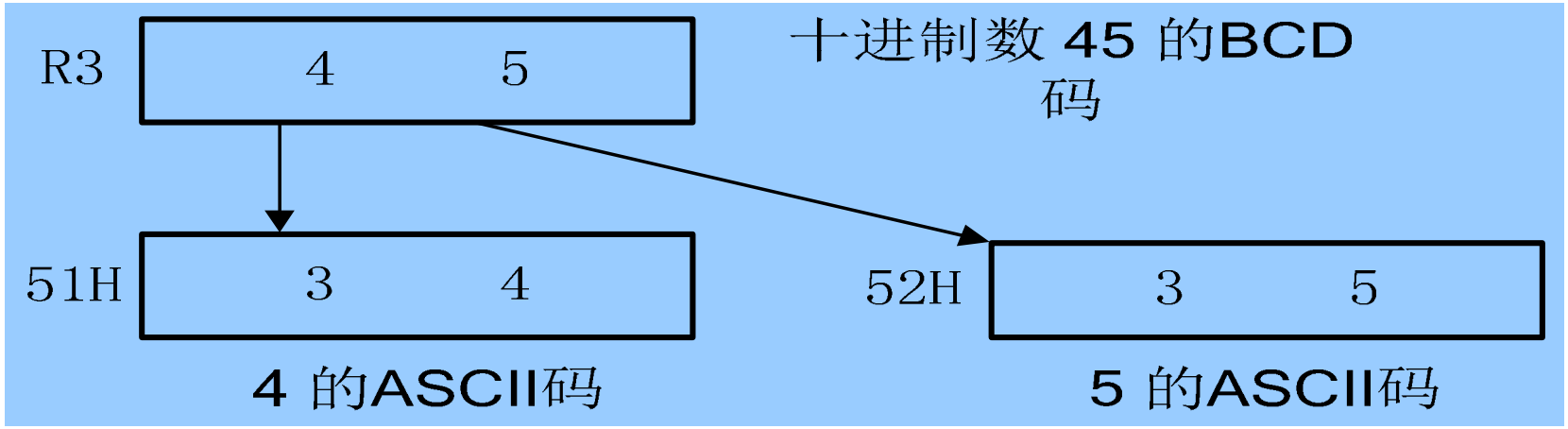


例1 ： 将R3中存放的BCD码拆开并变成相应的ASCII码，分别存放到51H和52H单元中。

数字与ASCII码之间的关系是：高4位为0011H，低4位即为该数字的8421码。

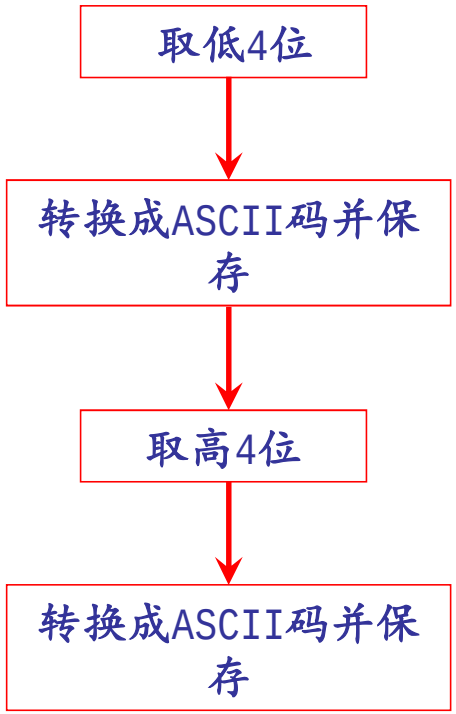
(1) 题意分析。

题目要求如下图所示。

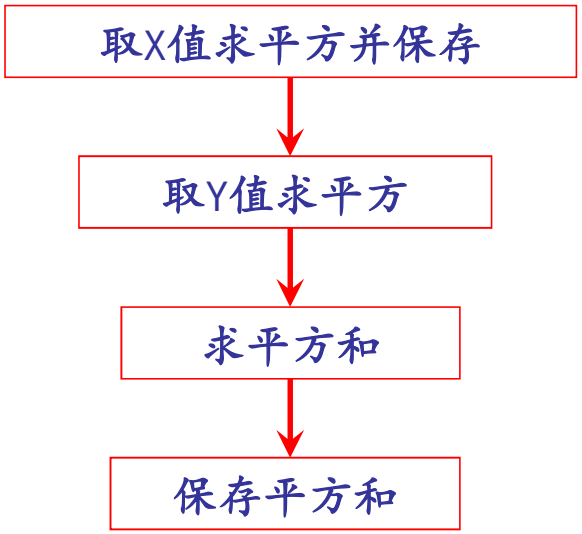


汇编语言源程序

```
        ORG   0000H
        MOV   A,R3
        ANL           A,#0FH
        ORL          A,#30H
        MOV   52H,A
        MOV   A,R3
        SWAP  A
        ANL      A,#0FH
       ORL        A,#30H
       MOV    51H,A
       END
```



例2 ： 设X、Y两个小于10的整数分别存于片内30H、31H单元，试求两数的平方和并将结果存于32H单元。



两数均小于10，故两数的平方和小于100，可利用乘法指令求平方。

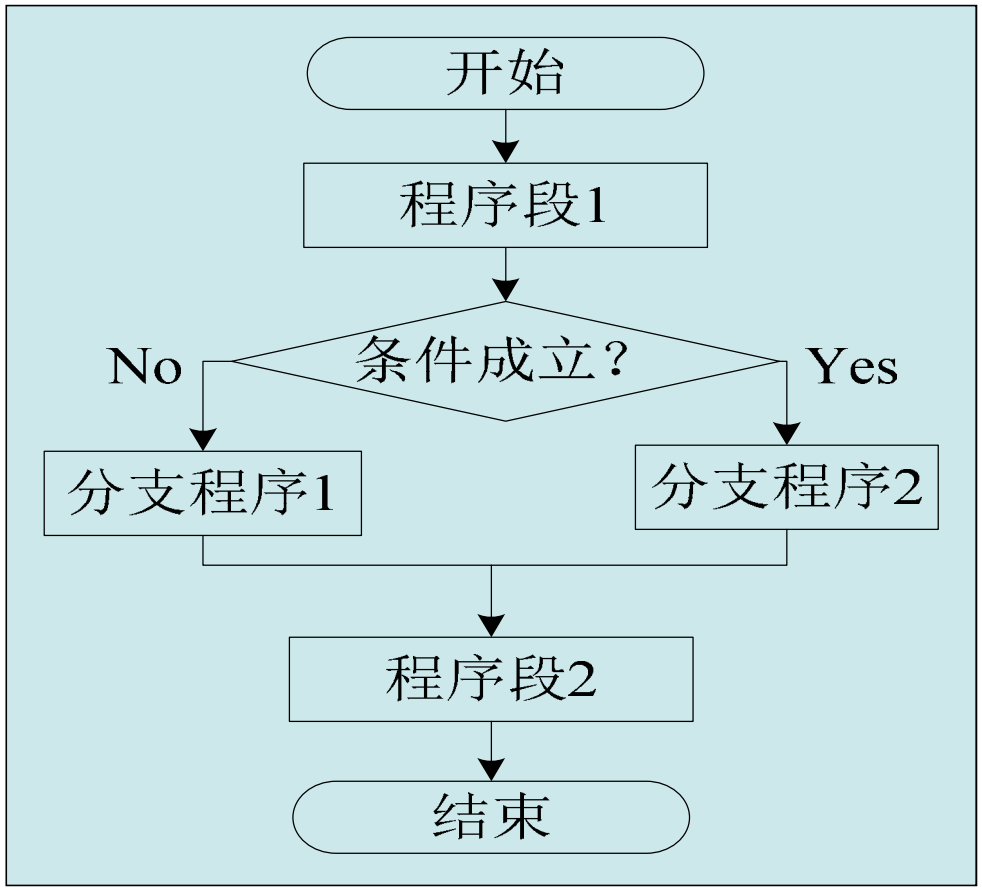
```
ORG   2000H
MOV   A, 30H
MOV   B, A
MUL   AB
MOV   R1, A
MOV   A, 31H
MOV   B, A
MUL   AB
ADD   A, R1
MOV   32H, A
```

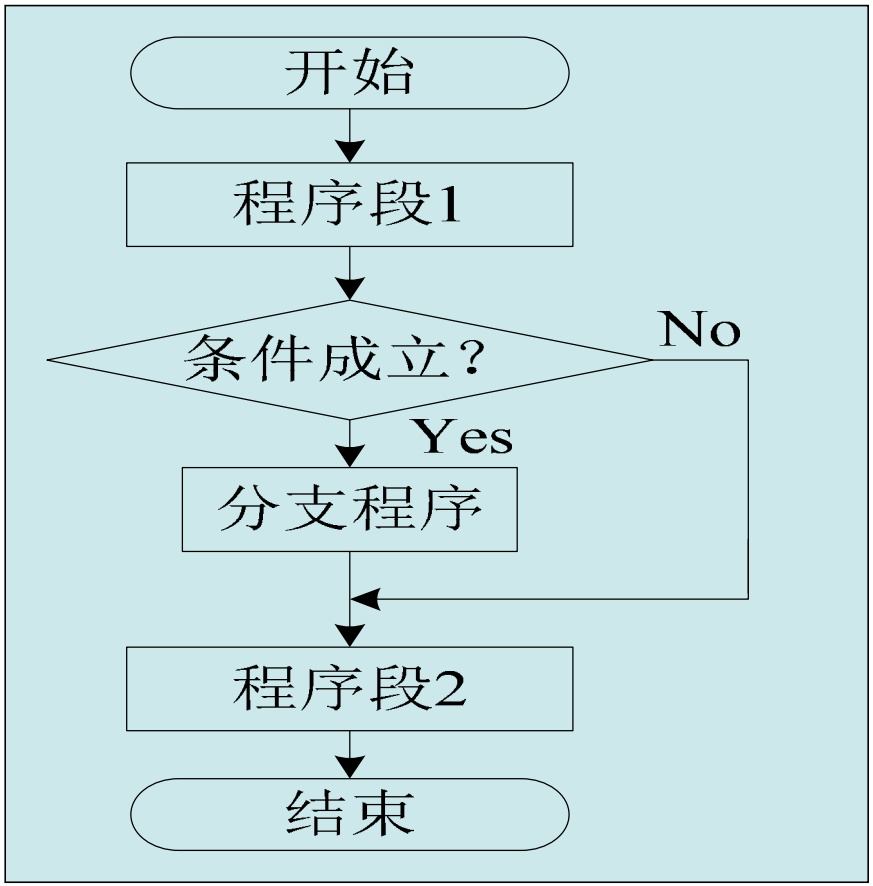
2、分支程序设计

1）分支程序设计的基本形式

（1）单分支程序结构

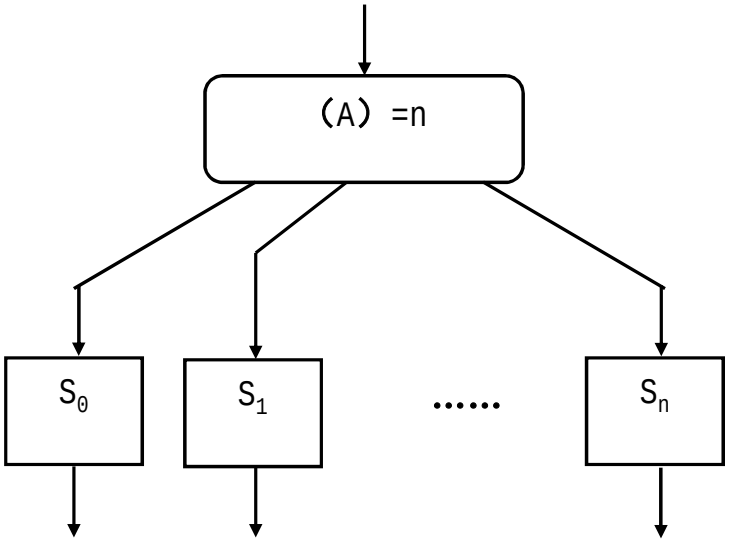
- 只有一个入口
- 两个出口
- 条件中只有两种可能性出现
- 条件一般由运算或检测的状态标志提供
- 通常使用条件判断指令实现


开始
程序段1
条件成立？
No
Yes
分支程序1
分支程序2
程序段2
结束


开始
程序段1
条件成立？
No
Yes
分支程序
程序段2
结束

（2）多分支程序结构

- 只有一个入口
- 多个出口
- 条件有多种可能性出现
- 通常使用散转指令实现
- JMP @A+DPTR

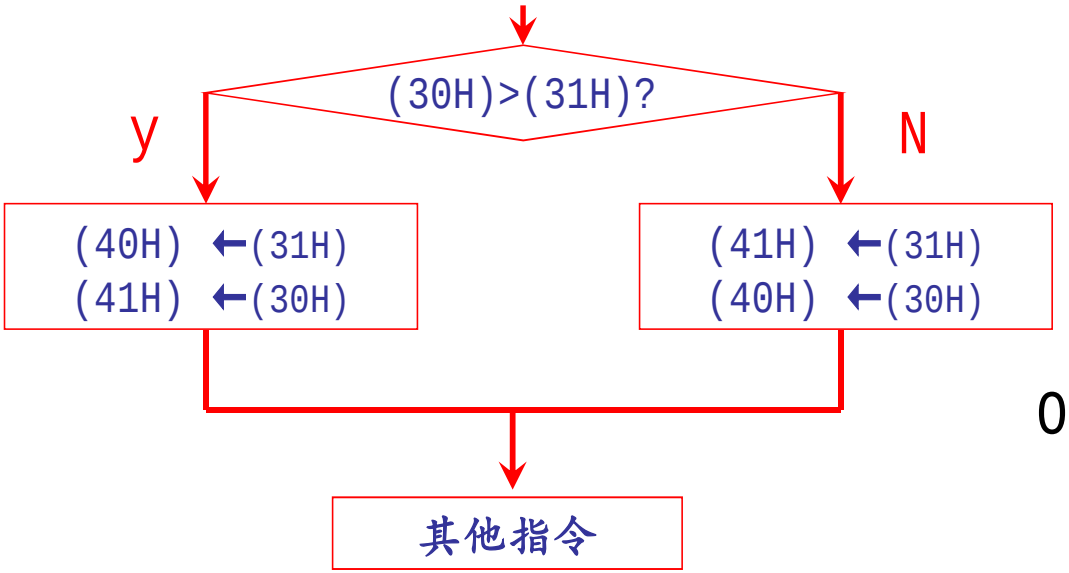


（3）分支程序的设计要点

- 先建立可供条件转移指令测试的条件
- 选用合适的条件转移指令
- 在转移的目的地址处设定标号

例3 ： 设内部RAM30H、31H单元中存有两个无符号数，比较其大小，并将大数存于41H单元，小数存于40H单元。

无符号数比较其大小，先做减法，再进位位判断。




```
        CLR  C
        MOV  A,30H
        SUBB A,31H
        JC   OUT1
        MOV  41H,30H
        MOV  40H,31H
        SJMP OUT
OUT1:   MOV  40H,30H
        MOV  41H,31H
OUT:    SJMP $
```

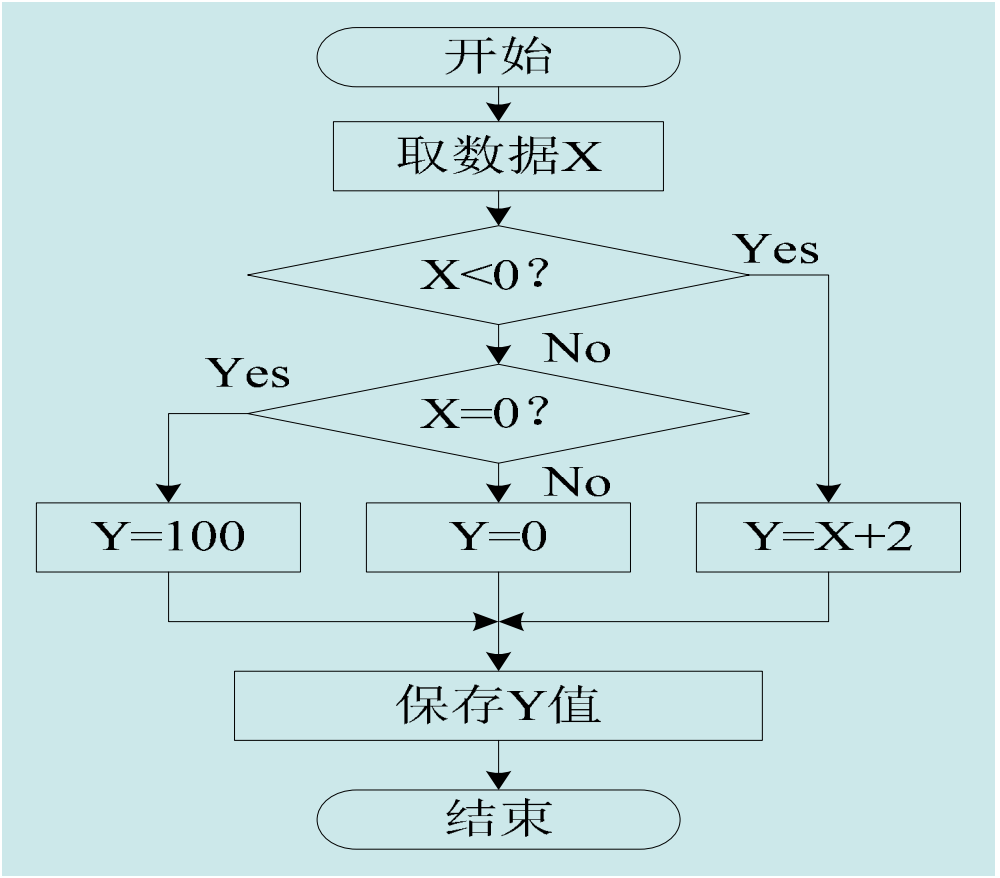
例4： 请编写计算下述函数式的程序，设X存在30H单元中，求出的Y值存入31H单元。

$$y=\begin{cases} x+2 & X<0 \\ 100 & X=0 \\ 0 & X>0 \end{cases}$$

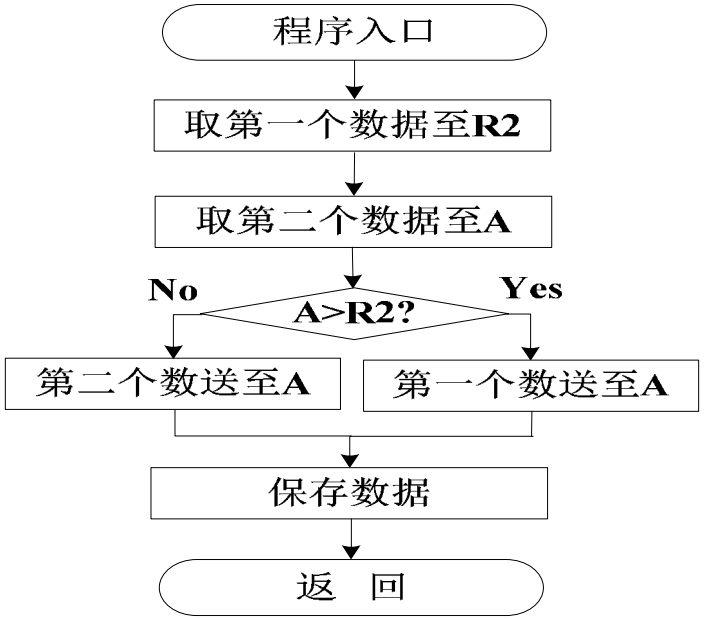
题意分析：

根据数据的符号位判别该数的正负，最高位为1，该数为负数；若最高位为0，再判别该数是否为0。

```
        ORG   0100H
        MOV   A,30H
        JB     ACC.7, L1
        JZ     L2
        CLR   A
        SJMP L3
L1:     ADD   A, #2
        SJMP L3
L2:     MOV   A,#100
L3:     MOV   31H, A
        SJMP $
        END
```



例5 ： 设片外RAM的2000H、2001H单元中存有两个无符号数，比较其大小，并将大数存于2002H单元中。




```
START:  CLR   C
        MOV   DPTR,#2000H
        MOVX  A,@DPTR
        MOV   R2,A
        INC   DPTR
        MOVX  A,@DPTR
        SUBB  A,R2
        JNC   LOOP1
        MOV   A,R2
LOOP0:  INC   DPTR
        MOVX  @DPTR,A
        RET
LOOP1:  MOVX  A,@DPTR
        SJMP  LOOP0
```

无符号数比较其大小，先做减法，再进位位判断。

3.3.3 循环程序设计

重　　点： 循环程序的结构及设计。

难　　点： 循环控制的条件。

教学要求： 灵活掌握几种常用类型循环程序的结构及其设计方法。

优点：速度快
缺点：代码长

优点：代码短
缺点：速度慢

导 设程序

将片内RAM中从31H开始的50个单元的内容全部清0。

顺序程序

```
ORG    0030H
MOV   31H, #00H
MOV   32H, #00H
MOV   33H, #00H
… …
MOV   61H, #00H
MOV   62H, #00H
… …
END
```

循环程序

```
        ORG     0050H
        MOV    R0, #31H      ;目的操作数首地址
        MOV    R7, #32H      ;预置循环次数50
LP:  MOV    @R0,#00H
        INC      R0                  ;修改目的地址
        DJNZ   R7,   LP          ;循环未达50次则继续
        … …
        END
```

一．循环程序的概念

1．循环程序的定义：按照某种控制规律重复执行程序中的一部分指令，这样的程序结构称为循环程序。

2．循环程序的结构特点：程序中含有可以重复执行的程序段（即循环体）。

3．循环程序的组成：

```
        ORG     0050H
        MOV     R0, #31H
        MOV     R7, #32H
LP:  MOV   @R0, #00H
        INC      R0
        DJNZ   R7,   LP
        … …
        END
```

（1）**初始化**：用来完成循环前的准备工作，如设置地址指针、循环次数等；

（2）**循环体**：被重复执行的指令或程序段，它是循环程序的工作程序。

（3）**循环控制**：用于控制循环的执行和结束。每执行完一次循环体后，对地址指针和循环次数进行修改，并据此判断是否继续循环。

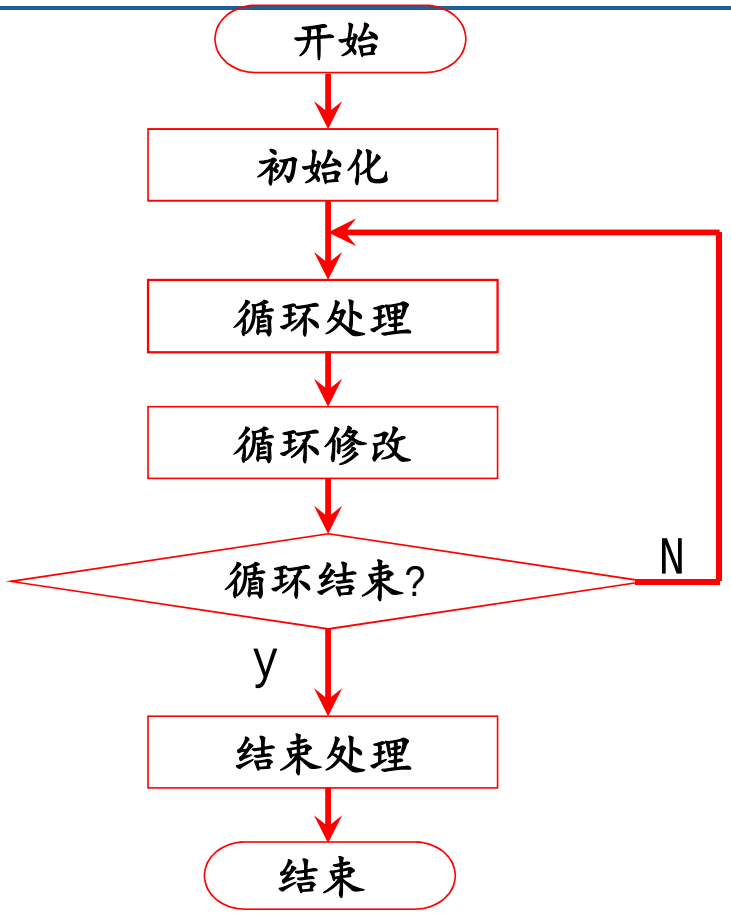
（4）**结束部分**：存放执行循环程序的结果或其它处理。

二. 循环程序分类:

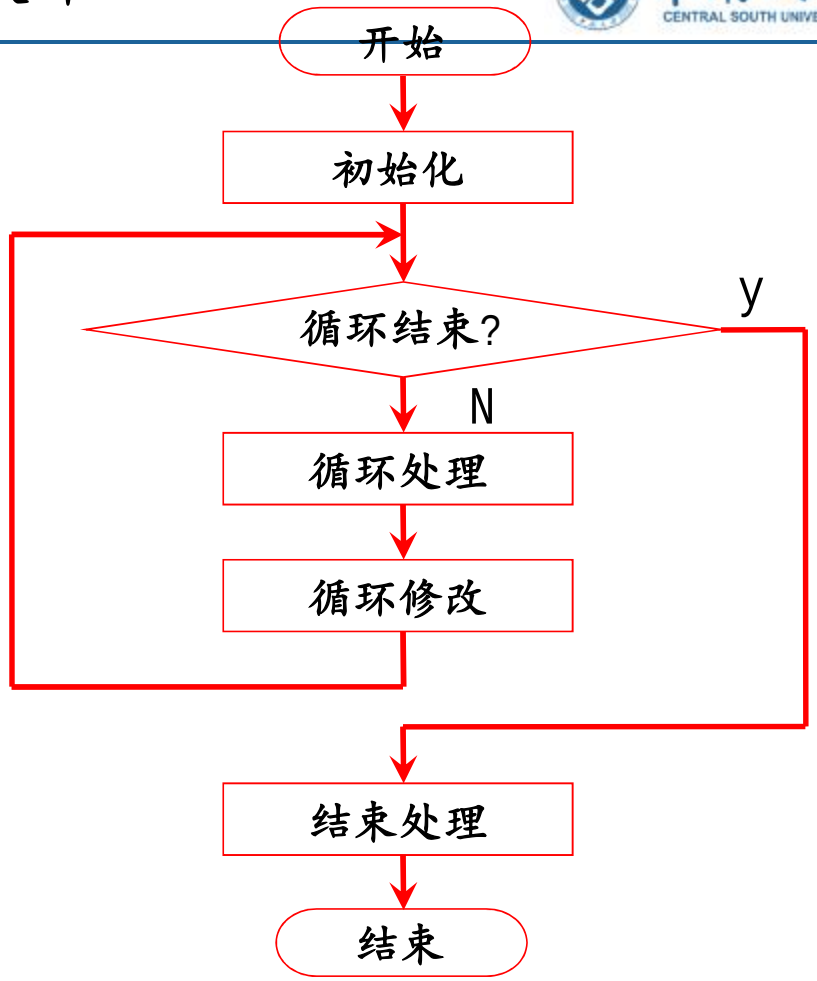
1. **按循环体的数量分为:** 单循环和多循环;

2. **按命题所给循环次数分为:** 已知循环次数和未知循环次数;

3. **按程序的编制方法分为:** 先处理后判断和先判断后处理。



先处理后判断



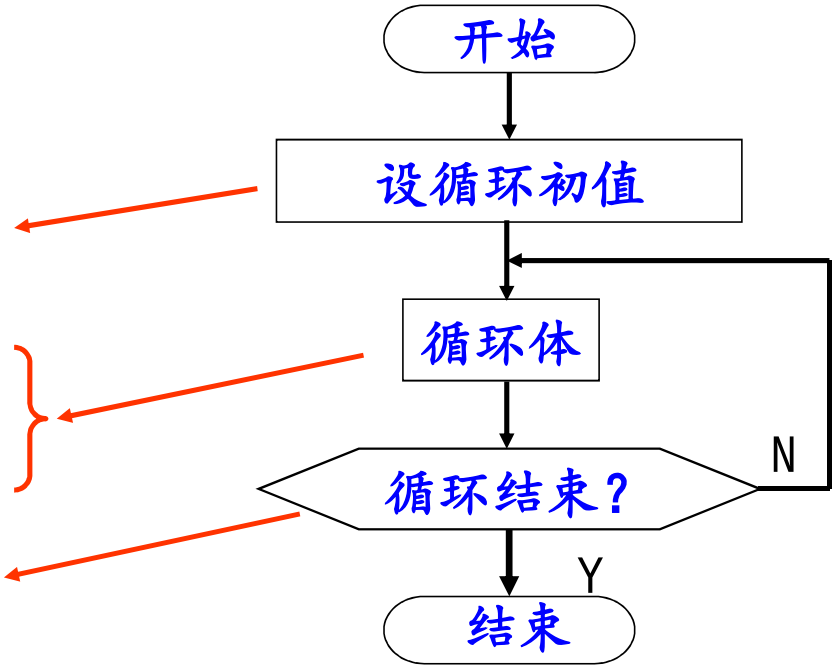
先判断后处理

三. 循环程序设计举例

1. 单循环程序

例1：软件延时500μs。

```
      ORG  0030H
      MOV  R1,#125  ; 1周期
LP:   NOP           ; 1周期
      NOP           ; 1周期
      DJNZ R1,LP    ; 2周期
      RET           ; 2周期
```



```
        ORG 0030H
        MOV  R1,#data ; 1
LP:NOP                ; 1
        NOP           ; 1
        DJNZ  R1,LP   ; 2
        RET           ; 2
```

设系统晶振频率为12MHz，则1个机器周期=1μS，执行该段程序所用的时间为：

$1+(1+1+2)\times data+2=500\ \mu S$

$data=125$

该程序可达到的最长延时时间为：

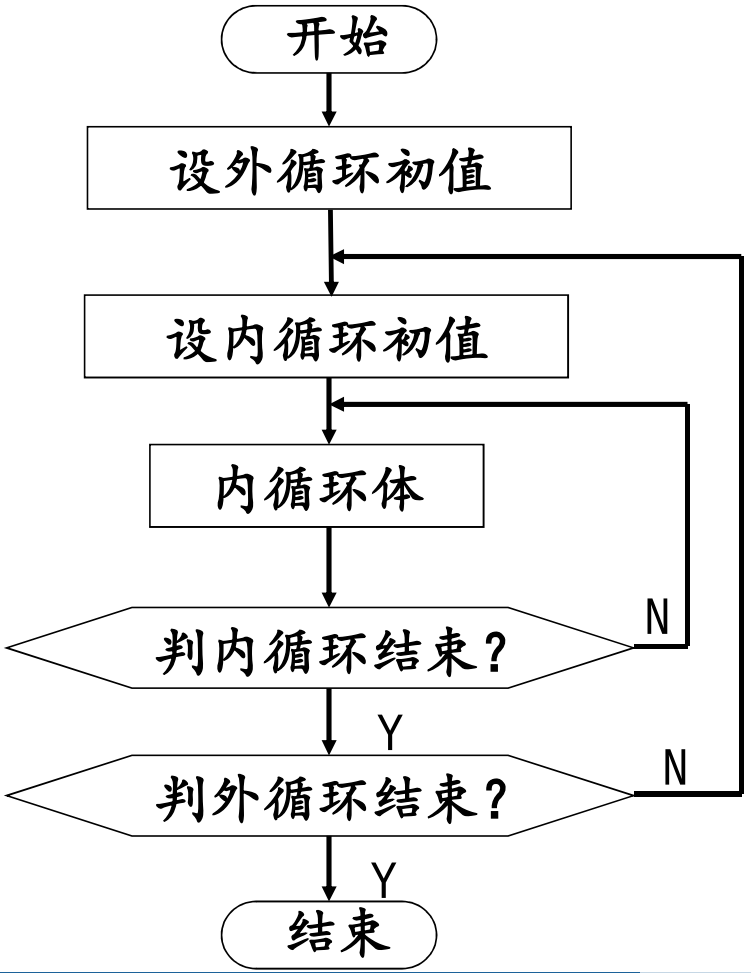
$1+(1+1+2)\times 256+2=1027\ \mu S$

如果需要更长的延时，则要增加循环体的数量。

2. 多重循环程序

例2：长延时程序。

```
      MOV  R1,#100   ; 1
LP1:MOV  R2,#10    ; 1
LP2:NOP            ; 1
      DJNZ R2,LP2    ; 2
      DJNZ R1,LP1    ; 2
      RET            ; 2
```



执行该段程序所用的时间是：

T=1+[1+（1+2）×10+2]×100+2=3303μS

其中圆括号内为内循环的机器周期数，方括号内为外循环的机器周期数，由此构成了两重循环嵌套。

3. 先处理后判断（如例1和例2）

4. 先判断后处理

例3：将[illegible]始的有限数据块[illegible]H开始的[illegible]’字符时停止传[illegible]

遇到“$”字符时停止传送

传送完“$”字符后停止传送

先判断后处理

```
        MOV   R0, #30H
        MOV   DPTR, #2000H
LP1:  MOV   A,@R0              ;取数
        CJNE   A, #24H, LP2     ;先判断
        SJMP   LP3
LP2:  MOVX   @DPTR, A      ;后处理
        INC   R0
        INC   DPTR
        SJMP   LP1
LP3:  END
```

先处理后判断

```
        MOV   R0,#30H
        MOV   DPTR,#2000H
TO1:  MOV   A,@R0              ;取数
        MOVX   @DPTR,A      ;先处理
        INC   R0
        INC   DPTR
        CJNE   A,#24H,TO1     ;后判断
        END
```

四．注意：

1．在多重循环中，只允许外重循环嵌套内重循环，不允许循环相互交叉；多个内重循环之间也不能相互交叉，但可以是并列关系。

2．选择控制循环的指令不同时，循环指针参数的设置也不相同，但应保证完成要求的循环任务：

1）命题中不能确定循环次数时，可用CJNE指令或其他判断指令根据检测条件控制循环的执行或结束；

2）如能够确定循环次数时，一般用DJNZ作为循环控制指令。程序中，循环初值预置为已知的循环次数，每循环一次，循环计数器减1。也可以用CJNE指令，但在程序的初始化部分循环初值应预置为0，每循环一次，循环计数器加1，再用CJNE指令对实际循环次数与给定的循环次数作比较，以判断循环继续或结束。

3．一定要有结束循环的条件，不要使程序进入死循环。

例4：不带符号的两个多字节数的减法程序：

```
        CLR     C
        MOV     R2,#N    ; 设定N字节
LOOP:MOV    A,  @R0  ; 从低位取出被减数1个字节
        SUBB    A,  @R1   ; 两位数相减
        MOV     @R0,  A  ; 保存差
        INC     R0    ; 修改指针
        INC     R1
        DJNZ    R2,  LOOP; 没减完则继续
        RET
```

3.3.4 子程序设计

在解决实际问题时，经常会遇到一个程序中多次使用同一个程序段，例如延时程序、查表程序、算术运算程序段等功能相对独立的程序段。

为了节约内存，我们把这种具有一定功能的独立程序段编成子程序，例如延时子程序。当需要时，可以去调用这些独立的子程序。调用程序称为主程序，被调用的程序称为子程序。

① 子程序调用和返回过程：

调用指令　　LCALL　或　ACALL

返回指令　　RET

②子程序只需书写一次，主程序可以反复调用它。

③子程序的第一条语句必须有一个标号，代表该子程序第一个语句的地址，也称为子程序入口地址，供主程序调用。

④子程序需要从主程序中得到的参数称为子程序的入口参数；子程序向主程序传递的参数称为子程序的出口参数。

⑤子程序的说明

在查表子程序前，通常以程序注释的形式对子程序进行说明，说明内容如下：

(a) 子程序名：提供给主程序调用的名字。

(b) 子程序功能：说明子程序能完成的主要功能。

(c) 入口参数：主程序需要向子程序提供的参数。

(d) 出口参数：子程序向主程序返回的参数。

(e) 占用资源：该子程序中使用了哪些存储单元、寄存器等。

这些说明是写给程序员看的，供以后使用子程序时参考。

例：将20H开始的10个单元的16进制数转换成ASCII码存放在同一单元中。

子程序HEXB：将A中的16进制数转换成ASCII码存放在A中返回。

```
        ORG     0000H
MAIN:   MOV     R1,#20H
        MOV     R7,#10
LOOP:   MOV     A,@R1
         LCALL  HEXB
        MOV     @R1,A
        INC     R1
        DJNZ    R7,LOOP
        SJMP    $
HEXB:   ANL     A,#0FH
        ADD     A,#1
        MOVC A,@A+PC
        RET
TAB:    DB  30H,31H,32H,33H
        DB  34H,35H,36H,37H
        DB  38H,39H,41H,42H
        DB  43H,44H,45H,46H
            END
```

例：用单片机的P1口控制8个发光二极管，要求8个二极管每次点亮一个，点亮时间为50mS，循环左移，一个接一个地点亮，循环不止。

主程序 ：

```
                ORG 0000H
START:  MOV A , #FEH
LOOP1:   MOV P1 , A
                LCALL YS50MS
            RL  A
           SJMP LOOP1
```

50mS延时子程序

```
YS50MS: MOV R7 , #200
YS1:            MOV R6 , #123
          NOP
YS2:            DJNZ  R6 , $
                DJNZ  R7 , YS1
         RET
```

例：设X、Y两个小于10的整数分别存于片内30H、31H单元，试求两数的平方和并将结果存于32H单元。

解：两数均小于10，故两数的平方和小于100，可利用调子程序的方法求平方。

主程序：

```
MOV     A , 30H
LCALL   QPF
MOV     R1 , A
MOV     A , 31H
LCALL   QPF
ADD     A , R1
MOV     32H , A
SJMP    $
```

子程序：A←A^2

```
QPF: ADD A,#01H
     MOVC A , @A+PC
     RET
TAB:   DB 0, 1, 4, 9, 16          DB
25,36,49,64,81
```

3.3.5 查表程序设计

在单片机汇编语言程序设计中，查表程序的应用非常广泛，在LED显示程序和键盘接口程序设计中都用到了查表程序段。

例： 设计一个将16进制数转换成ASCII码的子程序。设16进制数存放在R0的低4位，转换后ASCII码放在R0中。

题意分析：

所谓表格是指在程序中定义的一串有序的常数，如平方表、字型码表、键码表等。因为程序一般都是固化在程序存储器（通常是只读存储器ROM类型）中，因此可以说表格是预先定义在程序的数据区中，然后和程序一起固化在ROM中的一串常数。查表程序的关键是表格的定义和如何实现查表。

汇编语言源程序1：

```
        ORG    0000H
HEXA:MOV    A,R0
        ADD    A,#2
        MOVC   A,@A+PC
        MOV    R0,A
        RET
TAB:    DB    30H,31H,32H,33H
        DB    34H,35H,36H,37H
        DB    38H,39H,41H,42H
        DB    43H,44H,45H,46H
              END
```

查表时PC的值为该条指令的地址。

查表时PC的值与表格首地址间的偏移量。

汇编语言源程序2：

```
        ADD    A,#2
        MOVC   A,@A+PC
        MOV    R0,A
        RET
TAB:    DB    30H,31H,32H,33H
        DB    34H,35H,36H,37H
        DB    38H,39H,41H,42H
        DB    43H,44H,45H,46H
              END
```

DPTR中通常为表格的首地址。

查表得到的值在A中。

16进制数的ASCII码表。

教学案例：烘箱温度控制系统显示数码拆分程序设计

任务：根据烘箱温度控制系统的当前状态，在正常控温时，将温度设定值SV、温度测量值PV的拆分成BCD数码放入显示缓冲区DISBUF中；在设定值修改状态时，将温度设定备份值SVTEMP、温度测量值PV拆分成BCD数码放入显示缓冲区DISBUF中。

分析：温度测量范围是0～250 ℃，拆分时可将温度数值除以100，得到的商即为百位BCD数码，余数除以10得到十位和个位的BCD数码。

程序名： CODECON

功能：将温度测量值PV与温度设定（备份）值SV拆分成BCD数码，存放至显示缓冲区中

占用资源：累加器A，状态寄存器PSW，寄存器B，通用寄存器00组（R0）

显示缓冲区：DISBUF（3AH～3FH）

测量值：测量值高位在PVH中，低位在PVL中

设定值：设定值整数位在SV中

状态标志：FLAG=0，正常状态；FLAG=1，设定值修改状态

```
PVH     EQU   34H      ; 测量值高位
PVL     EQU   35H      ; 测量值低位
SV      EQU   30H      ; 设定值
SVTEMP  EQU   32H      ; 设定备份值整数位
DISBUF  EQU   3AH      ; 显示缓冲区首地址
FLAG    BIT   F0
```

变量定义

```
CODECON:    MOV   A,PVL          ; 取测量值位，温度小于250℃
            MOV   B,#100
            DIV   AB
            MOV   R0, #DISBUF
            MOV   @R0,A ; 测量值百位
            INC   R0
            MOV   A,B
            MOV   B,#10
            DIV   AB
            MOV   @R0,A ; 测量值十位
            INC   R0
            MOV   @R0,B ; 测量值个位
```

测量值拆分

```
        MOV   A,SV          ; 取设定值整数
        JNB   FLAG,CODEC1; 正常状态转移
        MOV   A,SVTEMP      ; 取设定备份值整数
CODEC1: MOV   B,#100
        DIV   AB
        INC   R0
        MOV   @R0,A ; 设定值百位
        INC   R0
        MOV   A,B
        MOV   B,#10
        DIV   AB
        MOV   @R0,A ; 设定值十位
        INC   R0
        MOV   @R0,B ; 设定值个位
        RET
```

设定值拆分

3.4 C51程序设计

C语言是一种编译型程序设计语言，它兼顾了多种高级语言的特点，并具备汇编语言的功能。目前，使用C语言进行程序设计已经成为软件开发的一个主流。用C语言开发系统可以大大缩短开发周期，明显增强程序的可读性，便于改进和扩充。而针对8051的C语言日趋成熟，成为了专业化的实用高级语言。

C51的特点：

1．C语言作为一种非常方便的语言而得到广泛的支持，国内最通用的是Keil C51。

2．C语言程序本身不依赖于机器硬件系统，基本上不作修改就可将程序从不同的单片机中移植过来。

3．C提供了很多数学函数并支持浮点运算，开发效率高，故可缩短开发时间，增加程序可读性和可维护性。

C51编程优点：

1．对单片机的指令系统不要求了解，仅要求对8051 的存贮器结构有初步了解；

2．寄存器分配、不同存贮器的寻址及数据类型等细节可由编译器管理；

3．程序有规范的结构，可分成不同的函数，这种方式可使程序结构化；

4．具有将可变的选择与特殊操作组合在一起的能力，改善了程序的可读性；

5．提供的库包含许多标准子程序，具有较强的数据处理能力；

6．由于具有方便的模块化编程技术，使已编好程序可容易地移植。

3.4.1 C51数据存储类型

存储类型	寻址空间	数据长度	值域范围
data	片内直接寻址 **RAM**	**8**	**0~127**
idata	片内间接寻址 **RAM**	**8**	**0~255**
pdata	分页寻址 片外 **RAM**	**8**	**0~255**
xdata	片外数据存储 （**64K**）	**16**	**0~65535**
code	片内统一编址**ROM**（**64K**）	**16**	**0~65535**
bdata	片内可位寻址的**RAM**（**16byet**）	**1**	**32~47**

C51的数据声明一般格式：

[类型说明符][修饰符] 标识符[=初值]……

例：unsigned char code Flag = 0x0f;

声明一个无符号的char型变量Flag，并给他赋初值为0x0f；并且声明他所在的存储空间在片内统一编址的ROM中。

声明中，如果没有修饰符，则数据默认的存储空间为data型，也就是在片内RAM中。

3.4.2 C51的程序结构

与一般C语言的结构相同，以main()函数为程序人口，程序体中包含若干语句还可以包含若干函数。

C51函数的一般格式

```
类型  函数名（参数表）
{
    数据说明部分
    语句执行部分
}
```

3.4.2 C51的数据类型与变量定义

1、基本数据类型

类型	符号	关键字	所占位数	数的表示范围
整型	有	(signed) int	16	-32768~32767
		(signed) short	16	-32768~32767
		(signed) long	32	-2147483648~2147483647
	无	unsigned int	16	0~65535
		unsigned short int	16	0~65535
		unsigned long int	32	0~4294967295
实型	有	float	32	3.4e-38~3.4e38
	有	double	64	1.7e-308~1.7e308
字符型	有	char	8	-128~127
	无	unsigned char	8	0~255

2、C51数据类型扩充定义

sfr: 特殊功能寄存器声明

sfr16: sfr的16位数据声明

sbit: 特殊功能位声明

bit: 位变量声明

例：
```
sfr   SCON = 0X98;
sfr16 T2 = 0xCC;
sbit  OV = PSW^2;
```

3、C51变量定义

数据类型　存储类型　变量

例：

```
char data var1;
bit data flags;
unsigned char xdata vextor[10];
```

3.4.3 C51包含的头文件

reg51.h ：定义51的特殊功能寄存器和位寄存器。

reg52.h ：定义52的特殊功能寄存器和位寄存器。

absacc.h：包含了直接访问51不同存储区的宏定义（绝对地址）。

math.h ：定义常用数学运算。

ctype.h ：包含有关字符分类及转换的名类信息。

stdlib.h ：说明一些常用的子程序：转换子程序、搜索/ 排序子程序等。

3.4.4 C51的运算符

C51运算符与C语言基本相同：

+ - * / （加 减 乘 除）

> >= < <= （大于 大于等于 小于 小于等于）

== != （测试等于 测试不等于）

&& || ! （逻辑与 逻辑或 逻辑非 ）

>> << （位右移 位左移）

& | （按位与 按位或）

^ ~ （按位异或 按位取反）

3.4.5 C51的基本语句

与标准C语言基本相同：

if	选择语言
switch/case	多分支选择语言
while	循环语言
do-while	循环语言
for	循环语言

例：清零程序（将片外2000H—20FFH的内容清零）

★ 汇编语言程序

```
        ORG   0000H
SE01:   MOV   R0,#00H
        MOV   DPTR,#2000H     ;首地址送DPTR
LOOP1:  CLR   A
        MOVX  @DPTR,A         ;一个单元清0
        INC   DPTR            ;指向下一单元
        INC   R0              ;字节数加1
        CJNE  R0,#00H,LOOP1   ;未完继续
LOOP:   SJMP  LOOP
```

清零程序（将片外2000H—20FFH的内容清零）

★ C51程序

```
#include <reg51.h>
main( )
  {
     int   i;
     unsigned char xdata *p=0x2000;
                        /* 指针指向2000H单元 */
     for(i=0;i<256;i++)
     {*p=0; p++;}   /*清零2000H-20FFH单元*/
  }
```

例：查找零的个数（在2000H--200FH中查出有几个字节是零，把个数放在2100H单元中）

★ 汇编语言程序

```
        ORG  0000H
L00:    MOV  R0,#10H        ;查找16个字节
        MOV  R1,#0
        MOV  DPTR,#2000H
L01:    MOVX A,@DPTR
        CJNE A,#00H,L02     ;取出内容与00H相等吗?
        INC  R1             ;0的个数加1
L02:    INC  DPTR
        DJNZ R0,L01         ;未完继续
        MOV  DPTR,#2100H
        MOV  A,R1
        MOVX @DPTR,A        ;0的个数送2100H
L11:    SJMP L11
```

★查找零的个数C51程序

```
#include <reg51.h>
main ( )
{ unsigned char xdata *p=0x2000;/*指针p指向2000H单元*/
  int n=0,i;
  for(i=0;i<16;i++)
  { if(*p==0) n++;          /* 若该单元内容为零，则n+1 */
    p++;                    /* 指针指向下一单元  */
  }
  p=0x2100;                 /* 指针p指向2100H单元 */
  *p=n;                     /* 把个数放在2100H单元中 */
}
```

03 END

中南大學 CENTRAL SOUTH UNIVERSITY

THANKS

《单片机与接口技术》

主讲人：李刚