

第四章 汇编语言程序设计



信息科学与工程学院自动化系



本章重点

教学目标：

本章内容是软件编程的关键，地位重要。

- ☐ 程序结构（顺序、分支、循环、子程序）；
- ☐ 应用实例；
- ☐ 熟练掌握程序设计的思路方法和技巧；
- ☐ 要求掌握典型算法；
- ☐ 找到分析问题和解决问题的着眼点；
- ☐ 学会抓住不同问题的规律性；
- ☐ 举一反三，独立思考，有创意，有新意，独到。

汇编语言程序设计的要点：

一、分清可执行指令和非执行指令及其功能

二、将高级语言程序设计方法迁移到汇编语言程序设计中来

三、特别注意汇编语言是面向机器的， 要记住

❑ CPU的资源；

❑ 存储器结构与寻址方式；

❑ I/O口、定时/计数器、中断系统等关键的参数；

❑ 寄存器间接寻址中指针的选择和使用技巧。

四、从宏观上看，MCS-51汇编语言程序的整体结构

从微观上看，微机中各组成部分是如何通过程序联系起来。

第四章 汇编语言程序设计

4. 1 汇编语言程序设计基础

4. 2 单片机汇编语言程序设计



4. 1 汇编语言程序设计基础

4. 1. 1 汇编语言与机器语言

4. 1. 2 汇编语言的格式

4. 1. 3 伪指令

4. 1. 4 汇编语言源程序的汇编

4. 1. 5 汇编语言程序设计的一般步骤



4. 1. 1 汇编语言与机器语言

- **机器语言**

- 计算机唯一能够识别和执行的语言，面向机器，二进制。

- **汇编语言**

- “符号语言”，用指令助记符代表机器语言指令，面向机器。
- 要经过汇编。

低级语言适合开发实时控制程序

- **高级语言**

- 面向算法、过程、对象，类似自然语言，可移植性好，须经解释或翻译后才能被执行。



4. 1. 2 汇编语言的格式

指令格式:

标号段： LABEL	操作码段 OPCODE	操作数段 OPRAND	； 注释段 COMMENT
---------------	----------------	----------------	------------------

•伪指令格式:

—名字 定义符 参数, ..., 参数; 注释或
—[标号:]定义符 项表 ; 注释

1. 标号段

标号段位于语句的开头，首地址，标号又称为**标号地址**，是**可选项**，只有需要时才设置。

```
LOOP: MOV  A,@R0
      .....
      DJNZ R2,LOOP
```

- 绝对不允许把指令的**保留字**、寄存器号及伪指令字符作为语句的标号；
- 标号在同一程序单位中只能出现一次。

2、操作码段

- 操作码段可以是**可执行指令的助记符**，操作码段用于指示计算机进行何种操作，因此，是任何一条语句中的**必选项**，汇编语言根据这一字段生成目标代码。
- 操作码段也可以是伪指令的助记符，对汇编程序下命令，在汇编时起作用。

3、操作数段

数

因指

- 常数：
 - 01010101B; 12D, 12; 0F1H, 59H
 - 67Q; 'A', 'a'
- 操作数：
 - A, B, DPTR
- 表达式
 - mov A,#(12H-03H)

(3) 标号地址 DJNZ R7,NEXT

(4) 带加、减运算符的表达式，例：MOV A,#100-1

4、注释段：注释指令或程序的含义，便于阅读程序、维护程序。

2021-5-10
必须用“;”隔开，续行时，也必须以“;”开头。



4. 1. 3 伪指令

- **定义：** 仅向汇编程序发出的，并仅由汇编程序在汇编过程中识别和执行的一种汇编控制命令，它本身在目标程序中不产生机器码。
- 汇编起始伪指令：ORG
- 汇编结束伪指令：END
- 定义字节伪指令：DB
- 定义字伪指令：DW
- 定义位伪指令：BIT
- 赋值伪指令：EQU DATA



汇编起始伪指令

- 格式: **ORG <起始地址>**
- 功能: **指定汇编源程序编译成机器语言程序的起始地址**

– 例如: **ORG 0000H**
LJMP 0100H
ORG 0100H
Start: MOV A,#5AH
SJMP \$
END

程序中可以有
多条ORG语句，但
定义的起始地址
既不要交叉，也
不要重叠。

必须从小地址向
大地址分配程序
所占空间。



汇编结束伪指令

- 格式： **END** [起始地址]
- 功能： 停止汇编。汇编程序遇到END伪指令后即结束汇编。处于END之后的程序，汇编程序将不处理。

示例： **ORG 0000H**

ljmp 0100H

ORG 0100H

Start: mov A,#5Ah

sjmp \$

END

一个汇编源程序可能有几个程序单元组成，包括主程序和若干个子程序，但只能有一个END语句。



定义字节伪指令

- 格式：<标号：> DB <项或项表>
- 功能：把项或项表的数值存入从标号开始的连续单元，其中项或项表可以是一个字节、数或以引号括起来的字符串。

ORG 1000H

Dat: DB 11h,-1,'A','BCD'
END

补码

1000H

ASCII码

程序存储器

11H

FFH

41H

42H

43H

44H

注意：

该指令只能为程序存储器赋初值，不能为其他存储器赋初值，尤其不能为内部数据存储器赋初值。



定义字伪指令

- 格式：<标号：> DW <项或项表>
- 功能：把项或项表的数值存入从标号开始的连续单元，其中项或项表是一个字(两字节)。

ORG 1000H

Dat: DW 1122h, 3344h, -1

END

1000H

程序存储器

11H

22H

33H

44H

FFH

FFH

只对程序存储器起作用



定义位伪指令

- 格式：<符号> BIT <位地址>
- 功能：定义位变量地址

U BIT 20H.0

V BIT 20H.1

W BIT 20H.2

X BIT P1.0

Y BIT P2.4

Z BIT P3.2

Q BIT TCON.2

“将右边的位地址赋给左边的字符名称”，被定义后，“字符名”是一个符号位地址。



等值(赋值)伪指令

- 格式: <符号> EQU <变量值>
- 功能: 定义符号变量值

X EQU 05H

Y EQU 06H

NEXT EQU 2000H

格式: <符号> DATA <变量值>

功能: 定义符号变量值(内部RAM), 数据
地址赋值伪指令

ONE DATA 30H (数据或地址)

TWO DATA ONE+1



DATA与EQU的区别:

- EQU定义的字符名必须“先定义后使用”，而DATA定义的“字符名”没有这种规定；
- DATA伪指令可以放在源程序的开头或结尾，也可以放在程序的其它位置，比EQU伪指令要灵活。



八、空间定义伪指令 DS

[标号:] DS 表达式

功能是从标号指定的地址单元开始，在程序存储器中保留由表达式所指定的个数的存储单元作为备用的空间，并都填以零值。例如：

ORG 3000H

BUF: DS 50

... ..

汇编后，从地址3000H开始保留50个存储单元作为备用单元。

4. 1. 4 汇编语言源程序的汇编

汇编可分成两种：

人工汇编

汇编语言源程序

机器汇编

汇编语言源程序

人工查指令机器码表

汇编

机器语言目标程序

汇编

机器语言目标程序

汇编程序

机器汇编是用机器自动把汇编语言源程序翻译成可以运行的目标代码程序，即二进制程序的过程，它是一种用机器来代替人脑的汇编，完成这一汇编过程的软件称为“汇编程序”。



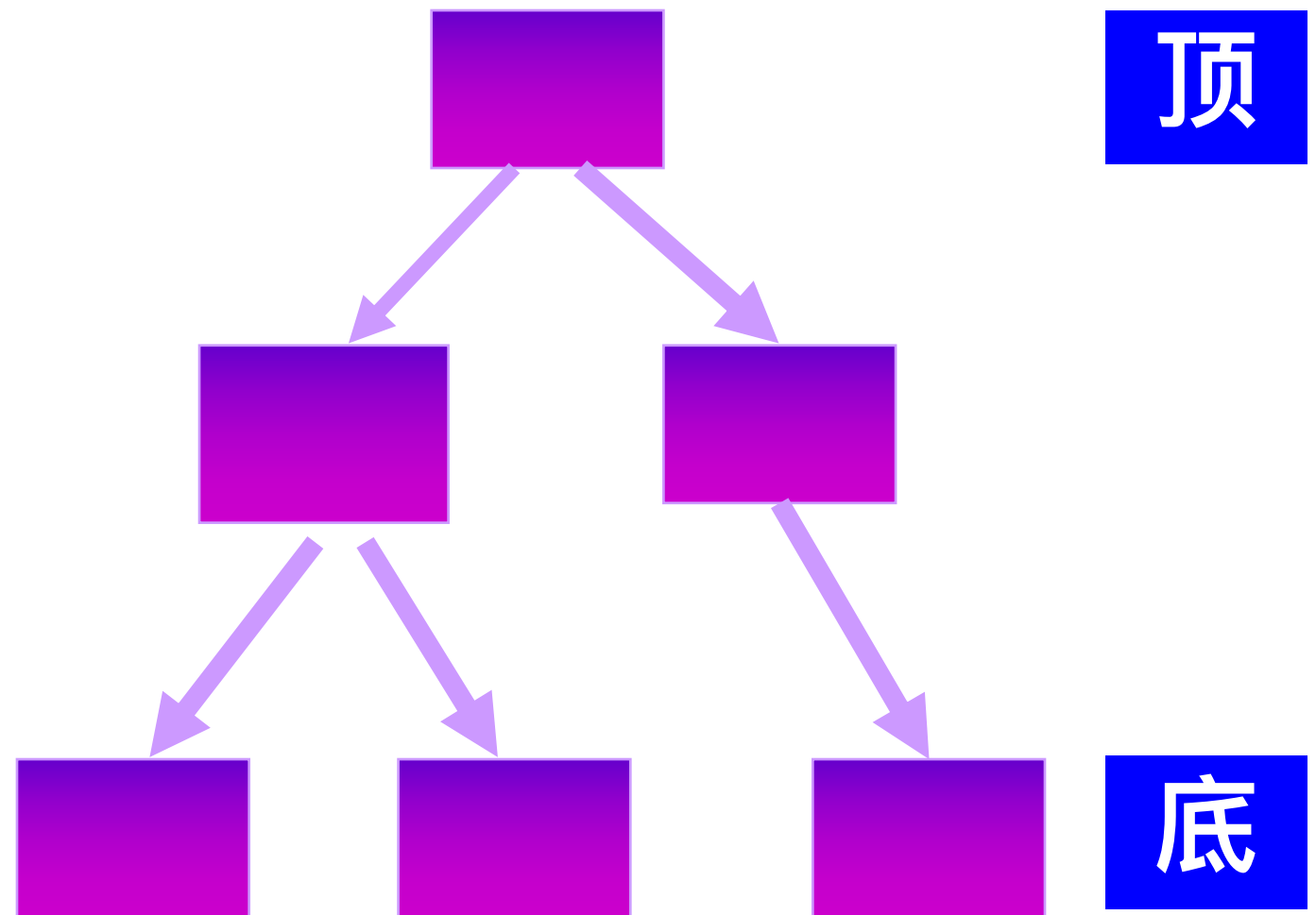
4. 1. 5 汇编语言程序设计的一般步骤

- 程序设计的方法
- 程序设计的步骤



程序设计的方法

- 结构化设计
- 模块化设计
- 开发方法
 - 自底向上开发
 - 先底层开发
 - 自顶向下开发
 - 先顶层开发
 - 混合方法



程序设计的步骤

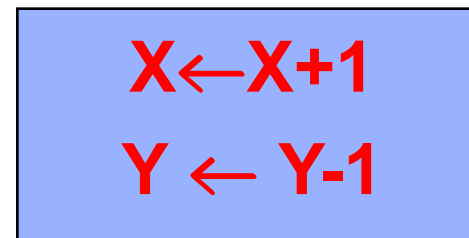
- 分析课题，确定算法和思路
- 根据算法和思路画出流程图
- 根据流程图编写程序
- 上机调试，排除错误

流程图图例

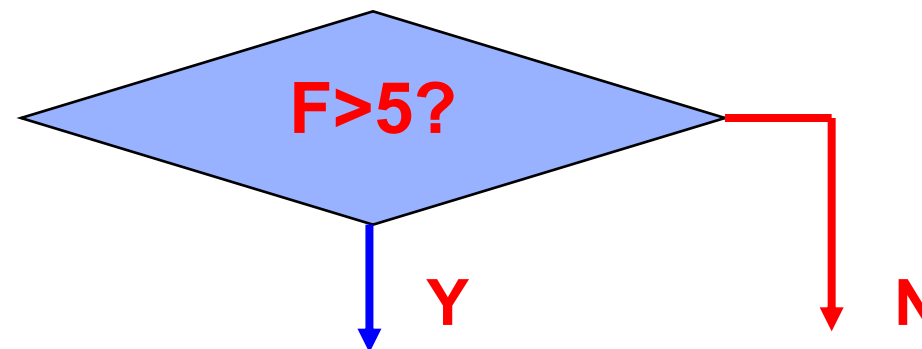
- 起止框



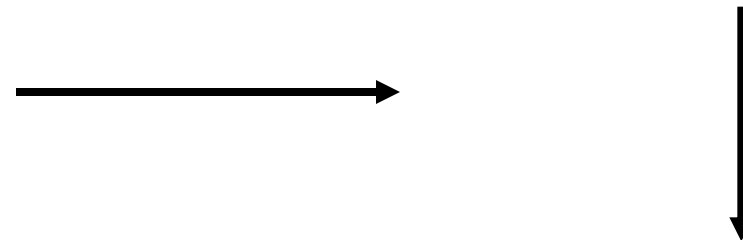
- 处理框



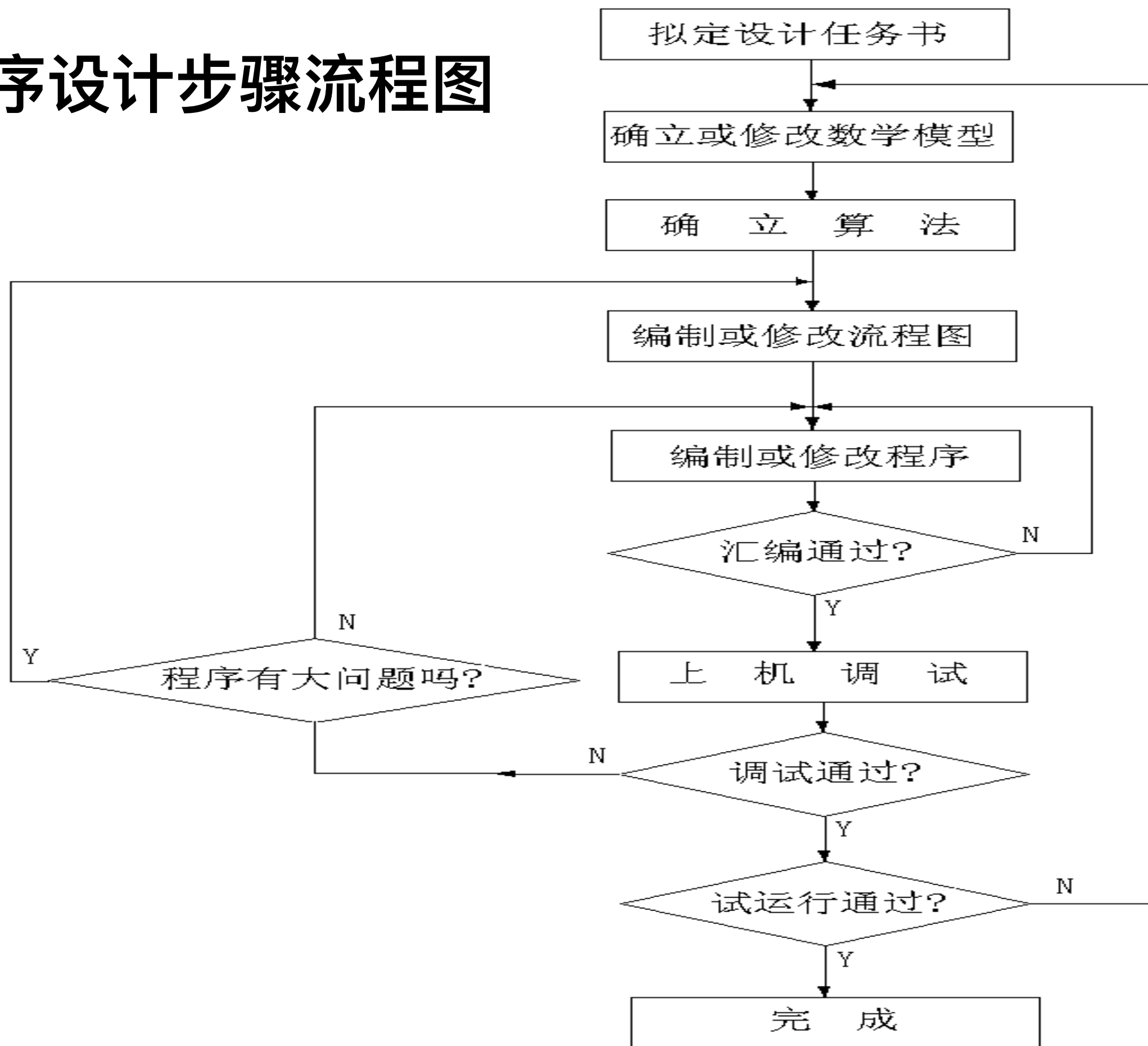
- 判断框



- 连线



程序设计步骤流程图



4. 2 单片机汇编语言程序设计

4. 2. 1 简单程序设计

4. 2. 2 分支程序设计

4. 2. 3 循环程序设计

4. 2. 4 查表程序设计

4. 2. 5 散转程序设计

4. 2. 6 子程序设计

4. 2. 7 运算程序设计



4. 2. 1 简单程序设计

Ø **特点：**从第一条指令开始依次执行每一条指令，直到程序执行完毕，中间没有转移指令，没有分支。

只有一个入口一个出口。

1、[例4-2] 将一个字节内的两位压缩的BCD码拆开并转换成相应的ASCII码，存入两个单元中。

算法1 ANL 拆字、ORL #30H 拼字

算法2 DIV 拆字、ORL #30H 拼字

存储 两位BCD数指压缩的BCD码占一个单元
相应的ASCII码占二个单元

2、[例4-4] 将8位无符号二进制数转换成三位BCD码，并存放在三个单元中。

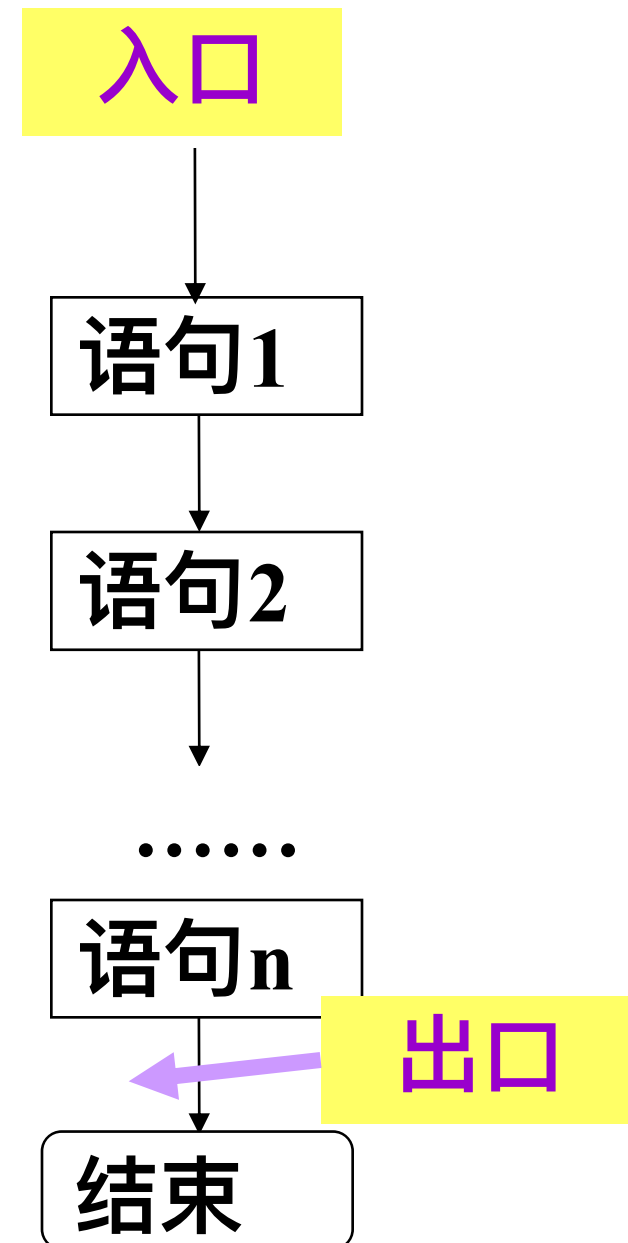
算法1 DIV 100 得百位，再DIV 10 的十位和个位

算法2 DIV 10 的个位，再DIV 10 的十位和百位

存储 8位无符号二进制数占一个单元

2021-10-10 十位和百位各占一个单元

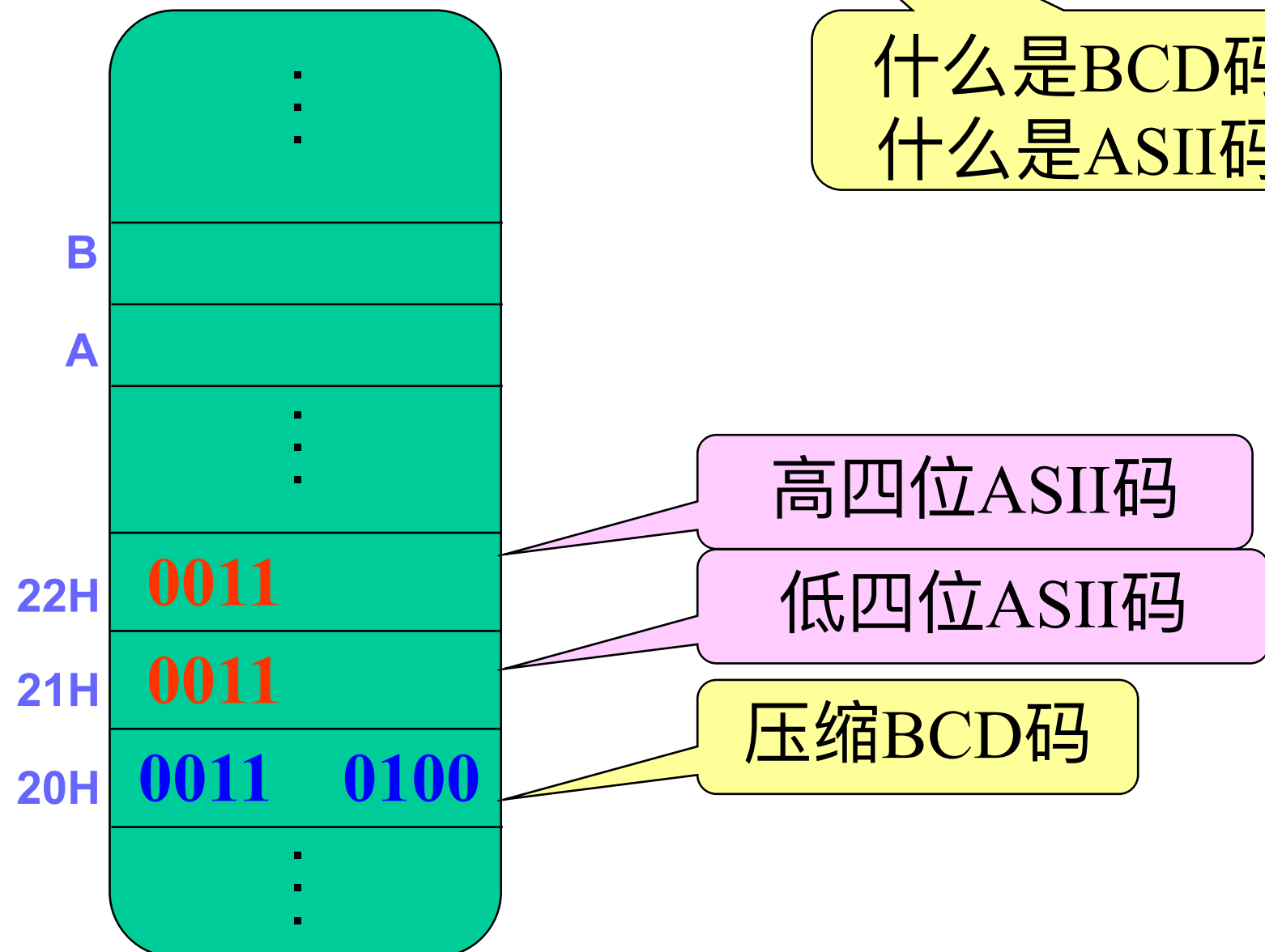
```
MOV A, R1
ANL A, #0FH
ORL A, #30H
MOV 31H, A
MOV A, R1
MOV B, 10H
(00010000B)
DIV AB
ORL A, #30H
MOV 32H, A
SJMP $
```



简单程序设计

结构特点： 按指令的先后顺序依次执行。

例1： 将20H单元的压缩BCD码拆开变成ASCII码，存入21H、22H单元。(假设20H中的BCD码为00110100)



方法1: 将BCD码除以10H，恰好是将BCD码分别移到了A、B的低4位。然后再各自与30H相或，即成为ASCII码。

方法2: 利用半字节交换指令来实现。

简单程序例1——方法1

源程序如下: PC 🖱️ ORG 0000H

PC 🖱️ MOV A, 20H

PC 🖱️ MOV B, #10H

PC 🖱️ DIV AB

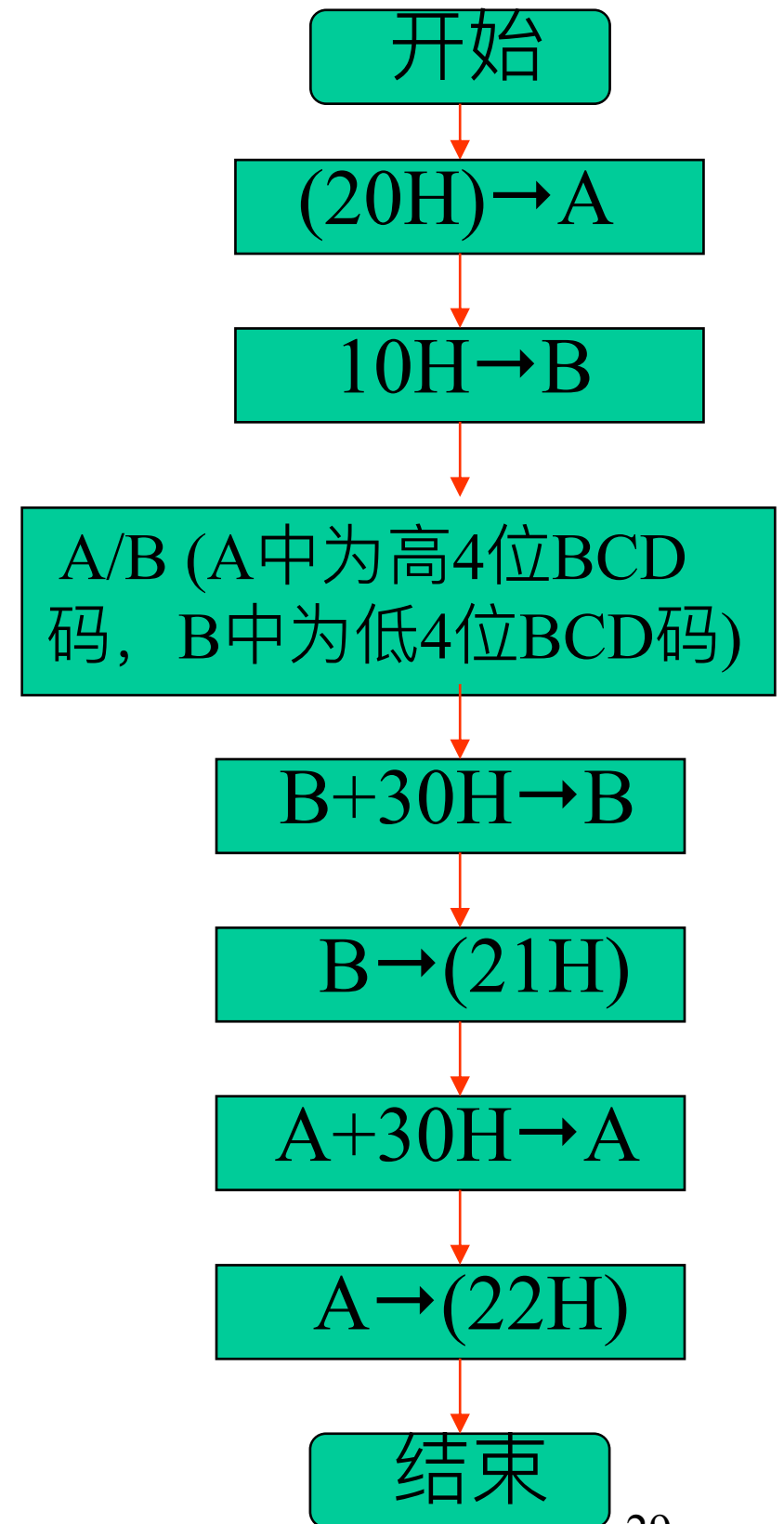
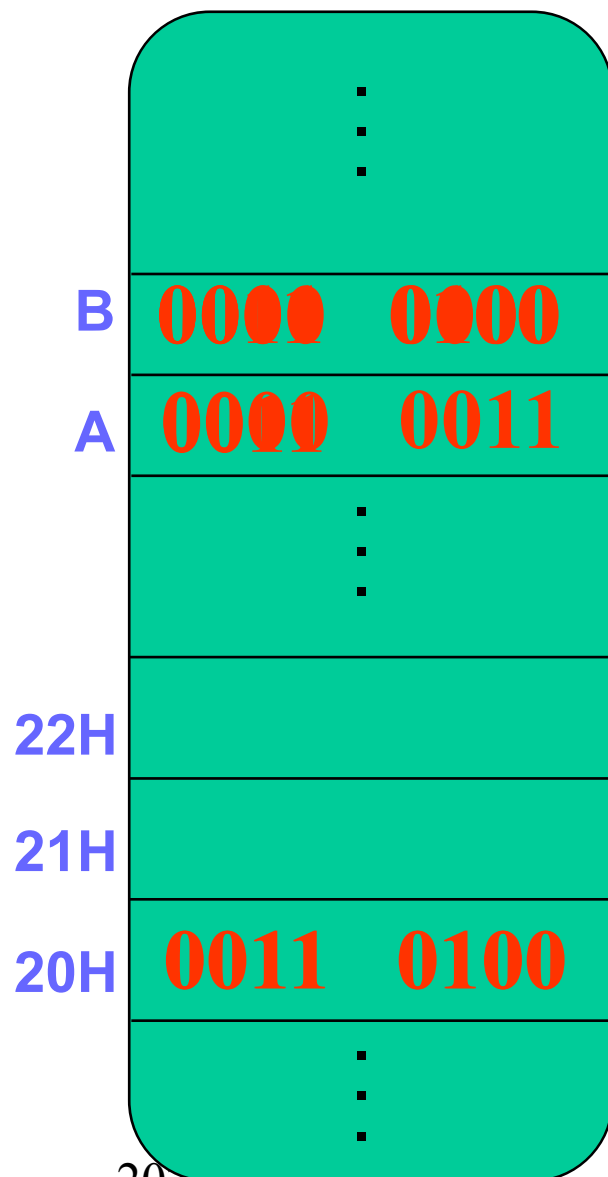
PC 🖱️ ORL B, #30H

PC 🖱️ MOV 21H, B

PC 🖱️ ORL A, #30H

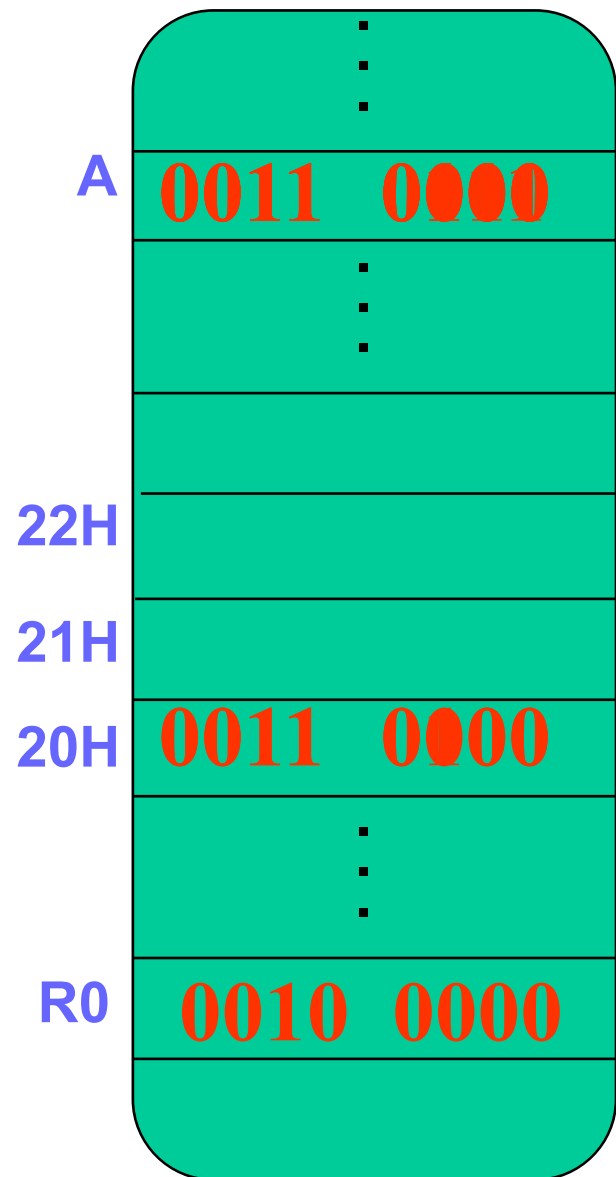
PC 🖱️ MOV 22H, A

PC 🖱️ SJMP \$



简单程序例1——方法2

源程序如下：



PC ORG 0000H

PC MOV R0, #20H

PC MOV A, #30H

PC XCHD A, @R0

PC MOV 21H, A

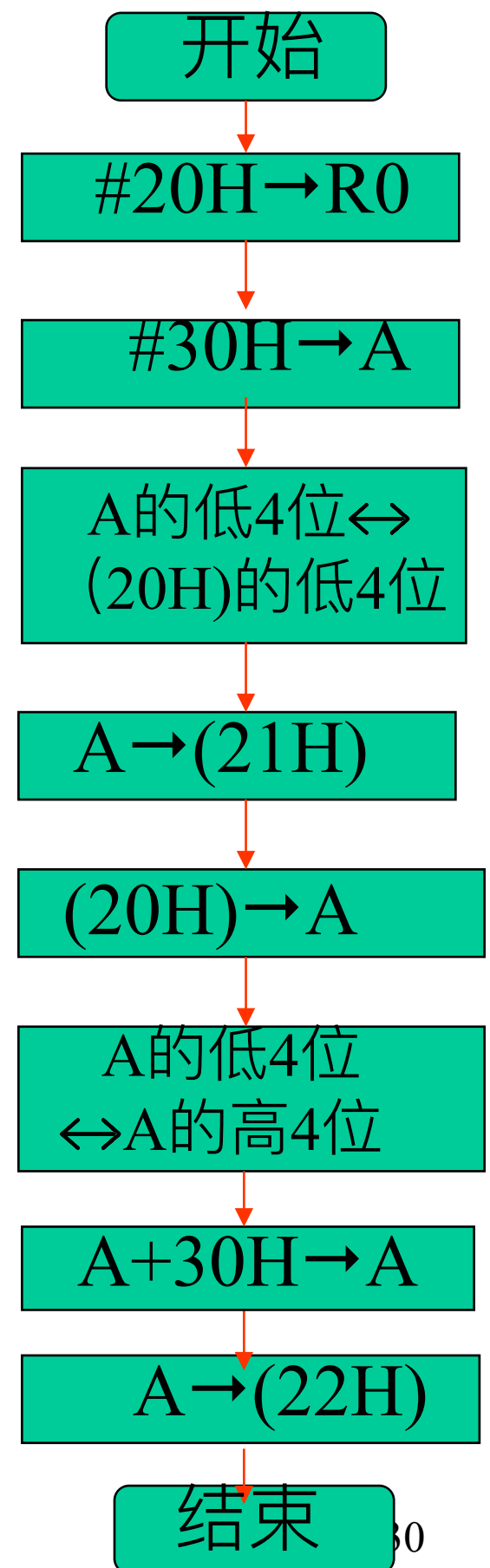
PC MOV A, @R0

PC SWAP A

PC ORL A, #30H

PC MOV 22H, A

PC SJMP \$



4. 2. 2 分支程序设计

- 用条件转移语句实现二分支
- 用cjne实现三分支
- 例4-5 多分支
- 用 jmp @A+DPTR 实现多分支

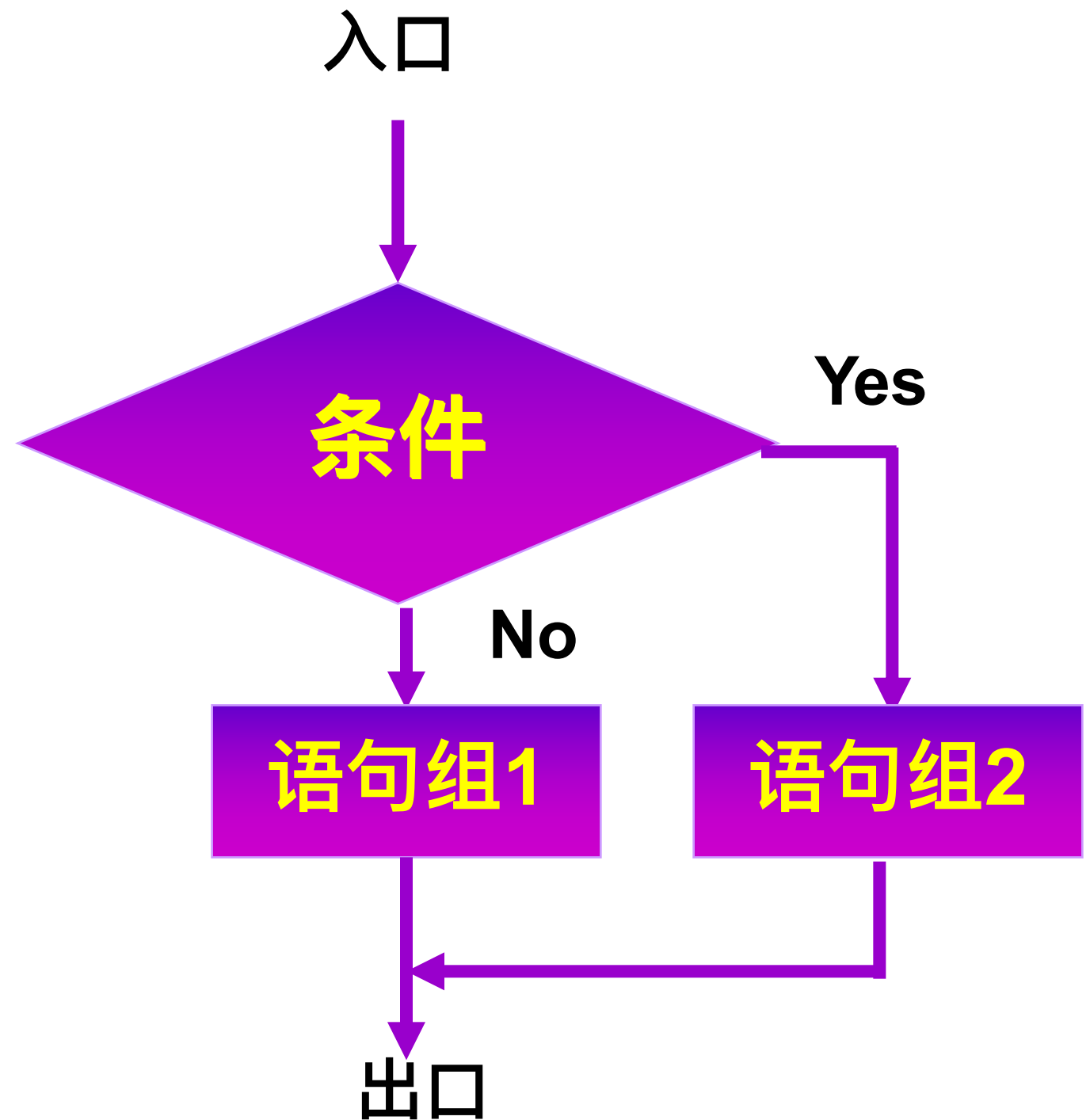


用条件转移语句实现二分支

根据不同的条件，执行不同的程序段。

JZ、JNZ、
JC、JNC、JB、
JNB、CJNE、
DJNZ

正确合理地运用



用cjne实现三分支

编程要点:

cjne = 等于

≠不等于则:

jc < 小于

或

jnc > 大于

如何实现 ≤?

≥?

ORG 0000H

Cjne R3,#15,NEQ

sjmp L1 ;=15

NEQ: jnC L1 ;>15

sjmp L2 ;<15

L1: nop ;≥15

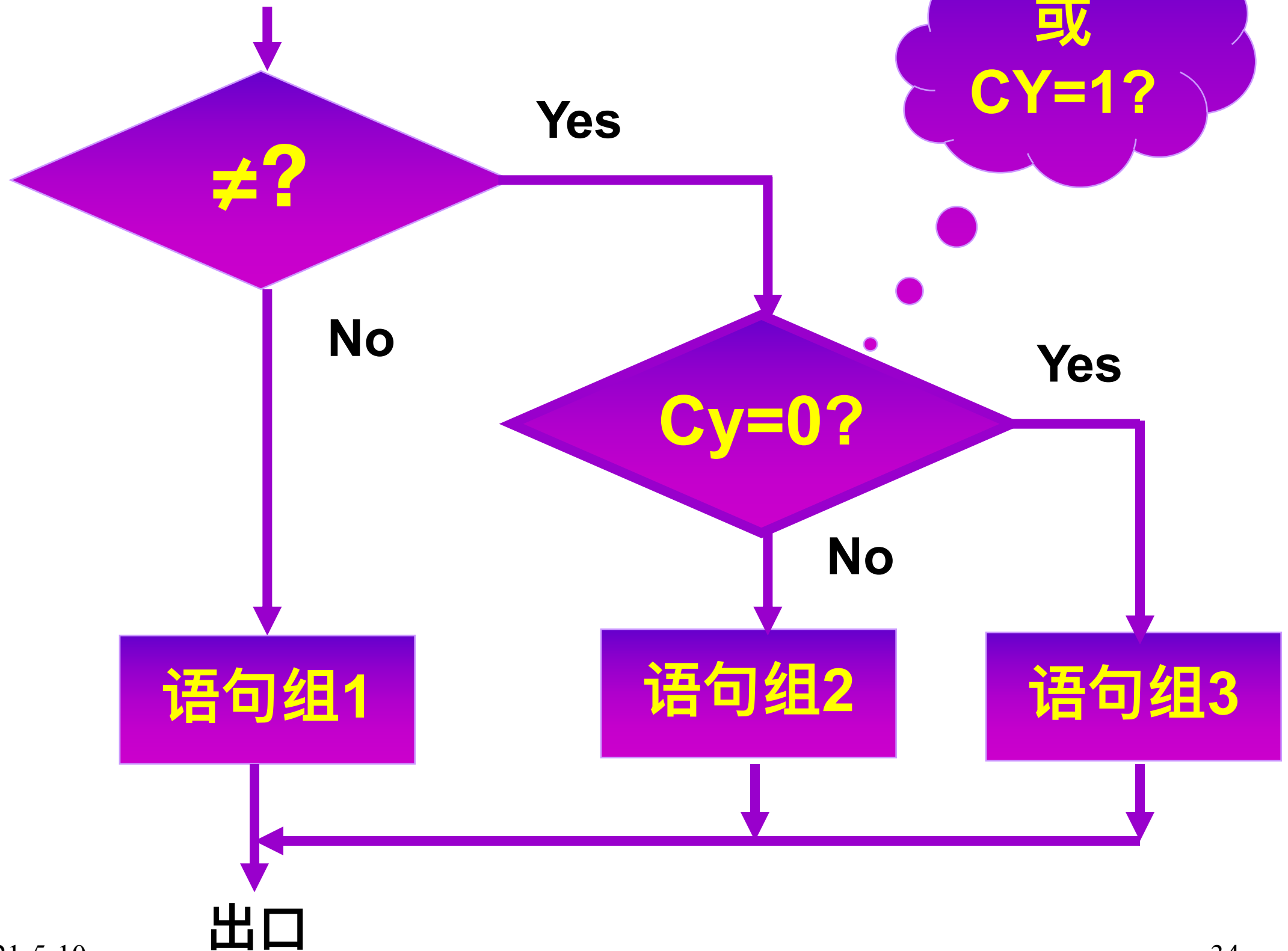
.....

L2:

比较指令相等于减法运算!

CJNE

入口



分析: $y = \begin{cases} +1 & x > 0 \\ -1 & x < 0 \\ 0 & x = 0 \end{cases}$ $(R0) \leftarrow x$
 $(R1) \leftarrow y$

1、判零

CJNE R0,#0,NZERO

MOV A,R0
JZ ZERO

CLR C
MOV A,R0
SUBB A,#0
JZ ZERO

其它方法?

2、判正负

MOV A,R0

JNB ACC.7, positive

CJNE R0, #80H,DO1

SJMP negative

DO1: JC positive

..... ; negative

Positive :....

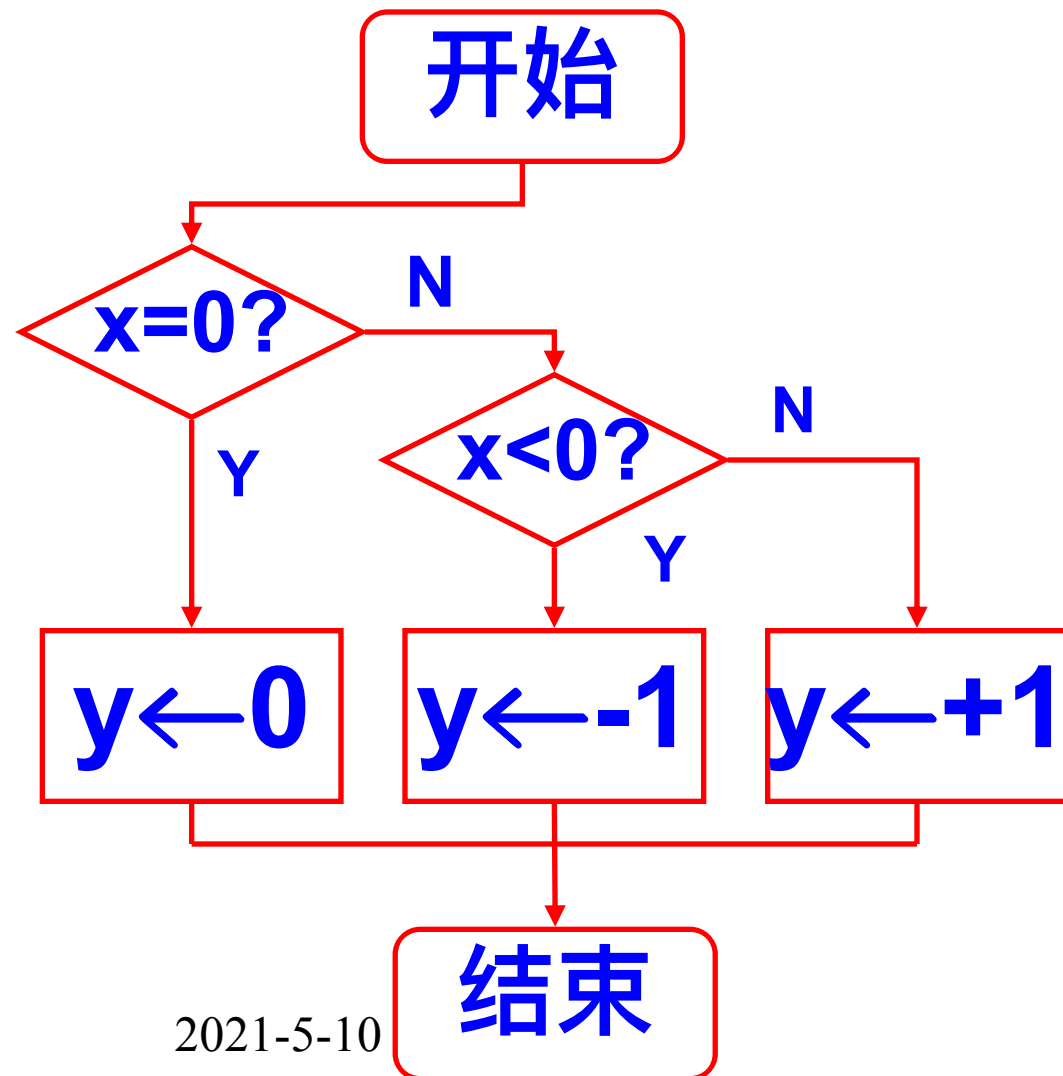
Negative:...

∴ 0FFH~80H negative

00H~7FH positive

用cjne实现三分支

$$y = \begin{cases} +1 & x > 0 \\ -1 & x < 0 \\ 0 & x = 0 \end{cases} \quad \begin{array}{l} (R0) \leftarrow x \\ (R1) \leftarrow y \end{array}$$



cjne R0,#0,MP1

mov R1,#0

sjmp MP3

MP1:mov A,R0

jnb ACC.7,MP2

mov R1,#0FFh

sjmp MP3

MP2:mov R1,#01h

MP3:sjmp \$

end

判零

判正负

分支间
隔离



[例4-5] ONE和TWO单元中的两个带符号数比较大小，将较大者存入MAX单元中。两数相等则任一个存入MAX即可。

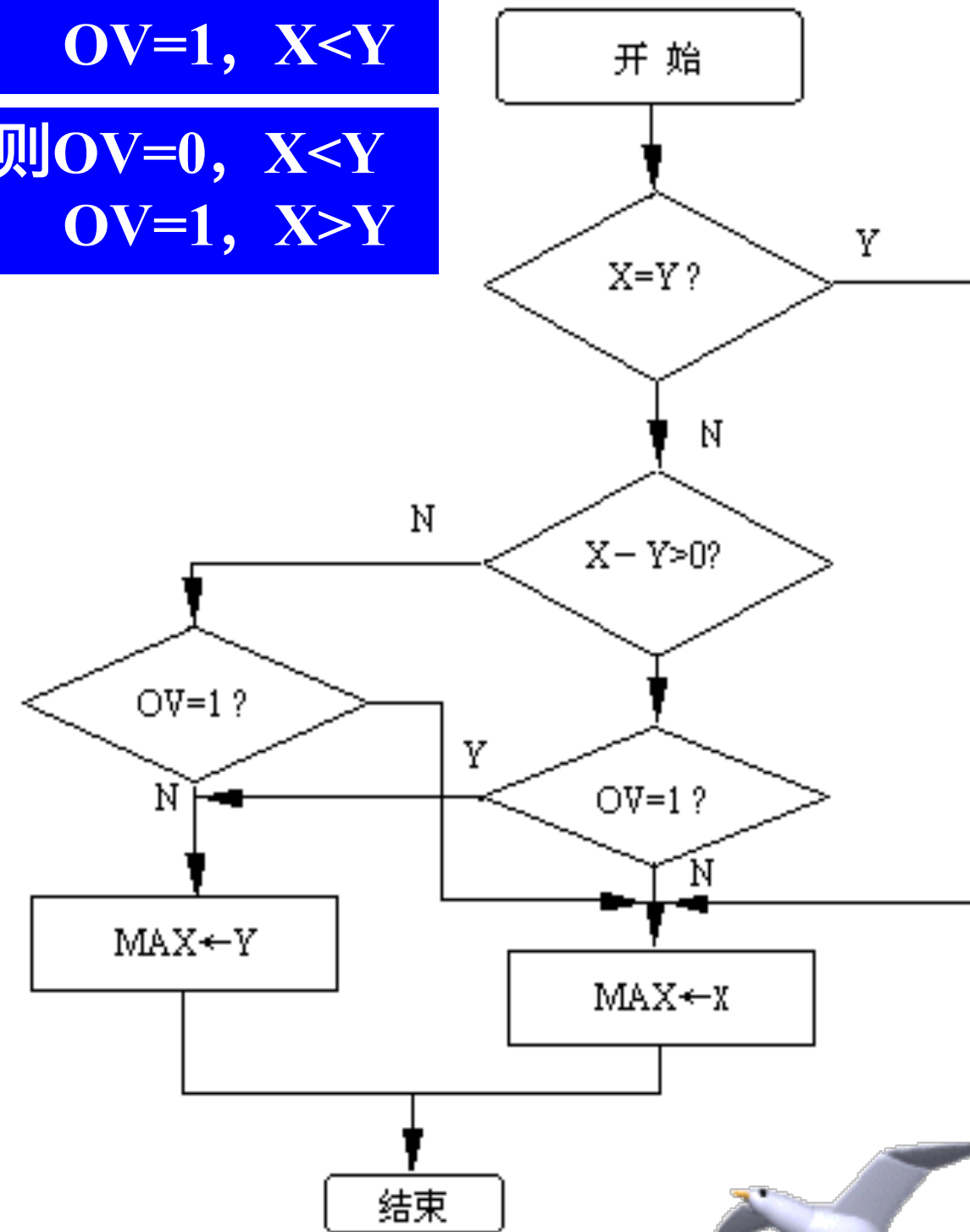
若 $X-Y$ 为正，则 $OV=0$ ， $X>Y$

$OV=1$ ， $X<Y$

若 $X-Y$ 为负，则 $OV=0$ ， $X<Y$

$OV=1$ ， $X>Y$

```
ORG 1000H
ONE DATA 30H
TWO DATA 31H
MAX DATA 32H
CLR C
MOV A , ONE
SUBB A , TWO
JZ XMAX
JB ACC. 7 , NEG
JB OV , YMAX
SJMP XMAX
NEG: JB OV , XMAX
YMAX: MOV A , TWO
      SJMP RMAX
XMAX: MOV A , ONE
RMAX: MOV MAX , A
SJMP $
END
```



用 jmp @A+DPTR 实现多分支

KeyB5: **mov DPTR,#JMPTBL**
 clr C
 subb A,#0Ah
 rl A ; (A) = (A) X2
 (占2个子节)

JMP @A+DPTR

JMPTBL: **ajmp AAA**
 ajmp BBB
 ajmp CCC
 ajmp DDD

AAA: ...

...

BBB: ...

键入 10, 转AAA

11, 转BBB

12, 转CCC

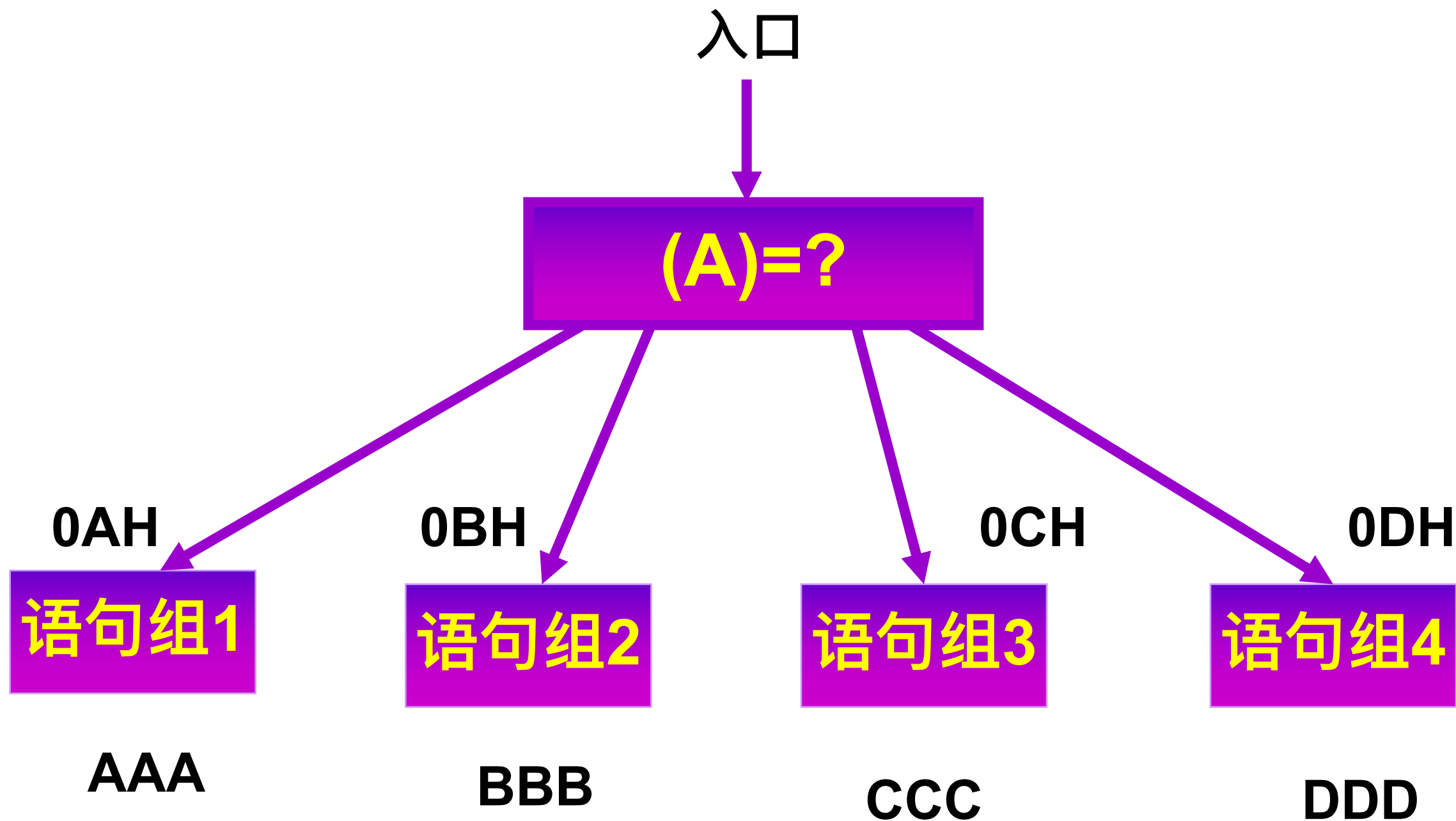
13, 转DDD

CCC: ...

...

DDD: ...

...



4. 2. 3 循环程序设计

- 循环程序结构
- 循环程序通常有两种编制方法
- 循环问题的类型
- 循环程序设计举例



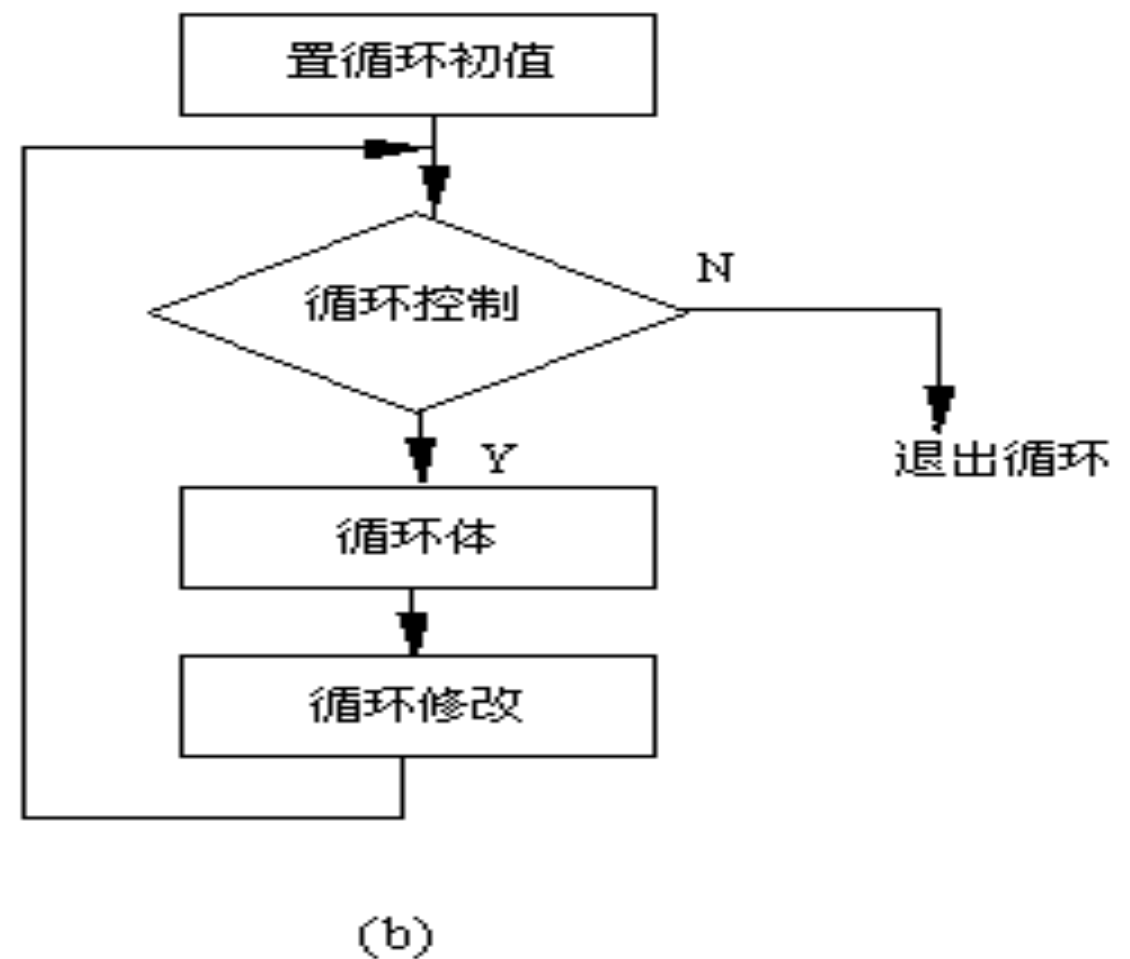
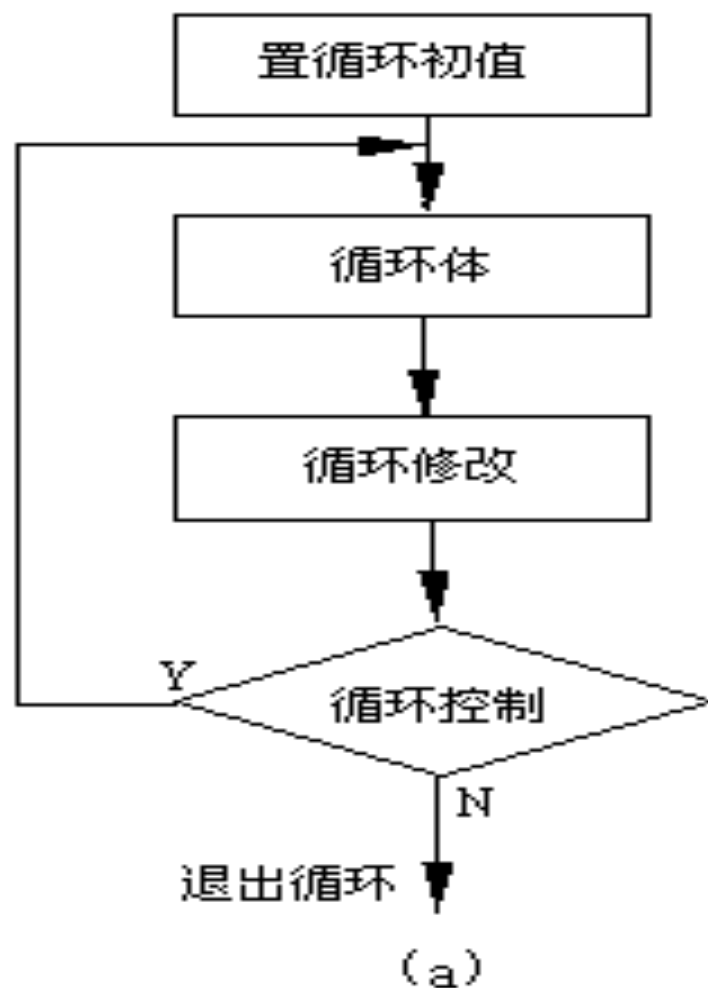
循环程序结构

- **循环初始化：** 循环控制变量的初始化、数据变量的初始化
- **循环工作部分：** 循环主体，重复执行的部分
- **循环控制部分：** 循环变量的修改、终止控制
- **循环结束：** 这部分程序用于存放执行循环程序所得结果以及恢复各单元的初值。



循环程序通常有两种编制方法

1、先循环处理后循环控制，称为直到型循环；



2、先循环控制后循环处理，称为当型循环。



循环问题的类型

1、计数型

循环次数已知，用计数方法控制循环的终止。

2、条件型

循环次数未知，根据某种条件判断是否终止循环。

[例4－7] 内部RAM块传，遇到“#”字符结束

3、计数型+条件型

[例4－8] 8031外部RAM块传及冒泡排序 重点



循环程序设计举例

•单循环程序

- 例1：多个单字节数求累加和 **计数型**
- 例2：内部数据区清零 **计数型**
- [例4—7] 内部RAM块传 **条件型**

•多重循环程序

- 例3：50ms延时程序 **计数型**
- [例4—7] 内部RAM块传 **计数型+条件型**
- [例4—8] 外部RAM块冒泡排序 **重点 计数型+条件型**



例1：多个单字节数求累加和

ORG 0000H

SUM: mov R3,#0

mov R4,#0

mov R0,#50H

mov R2,#5

Loop: mov A,R4

add A,@R0

mov R4,A

inc R0

clr A

addc A,R3

累加和
单元要
先清零

Σ

mov R3,A

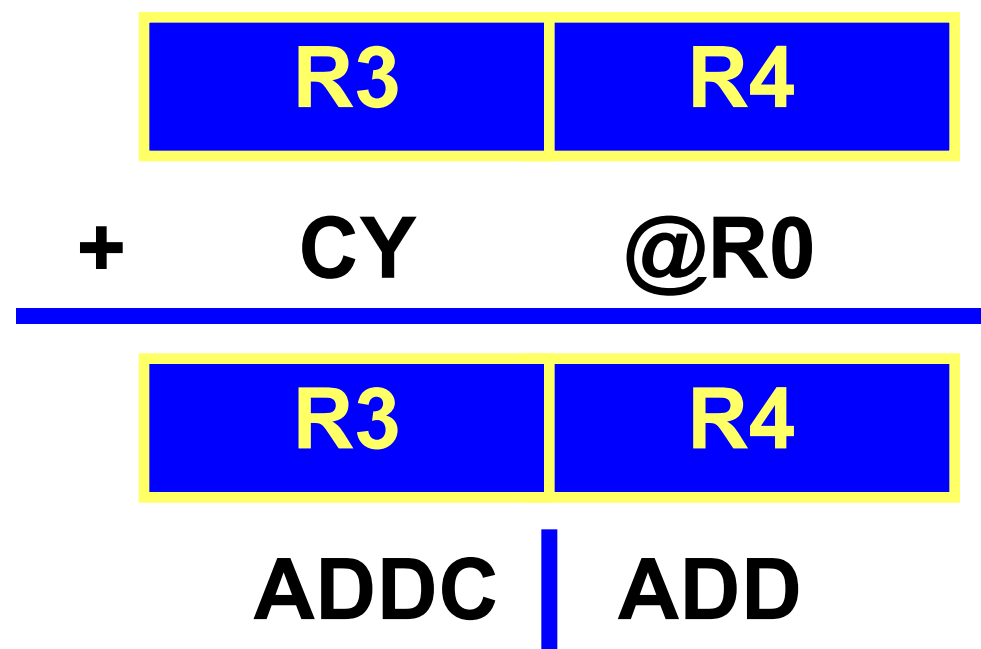
djnz R2,Loop

sjmp \$

END

内部RAM

R0 → 50H



23H
98H
0A8H
0FDH
6DH



例2：内部数据区清零

ORG 0000H

Zero: mov R0,#30H

mov R7,#10

mov A,#00H

Loop: mov @R0,A

inc R0

djnz R7,Loop

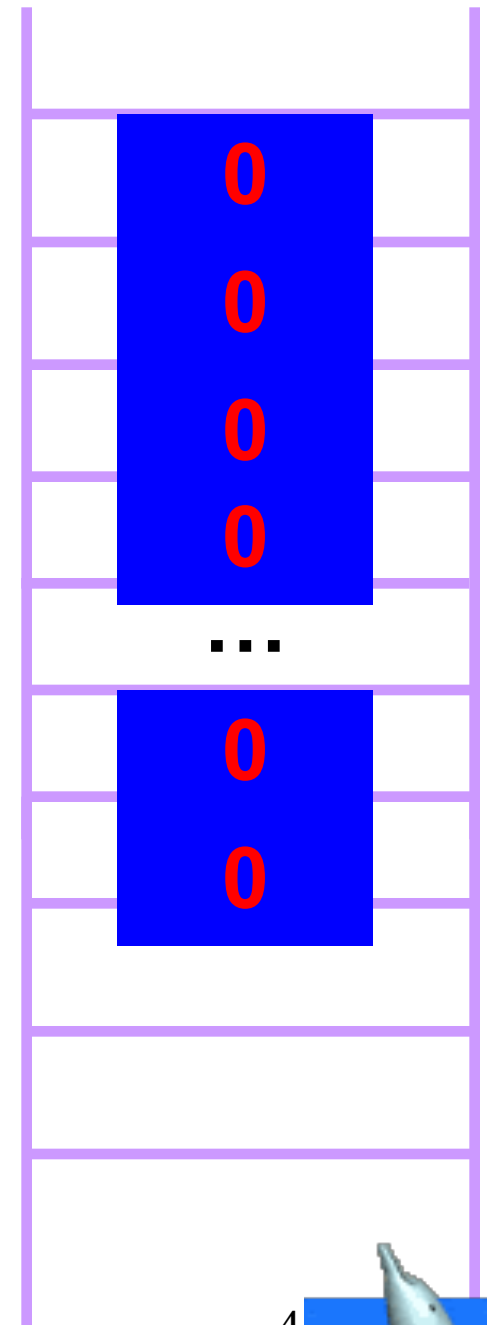
sjmp \$

END

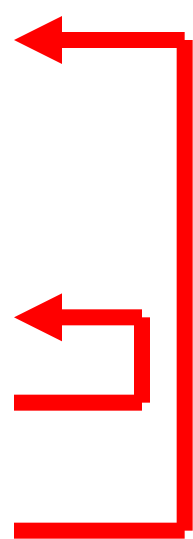
R0 → 30H

10个

内部RAM



例3：50ms延时程序

Delay: mov R7,#200		;1us
Del1: mov R6,#123		;1us
nop		;1us
Del2: djnz R6,Del2		;2us
djnz R7,Del1		;2us
sjmp \$; (不计入)		

$$T = [200 \times (1 + 1 + 123 \times 2 + 2) + 1] \times 1\mu s$$
$$= 50001\mu s = 50ms$$

2021-5 若晶振频率为12MHz，则一个机器周期为1μs。

1S延时程序

源程序：

```
DELAY:  MOV R2,  #10 ; 1 μs
DEL3:   MOV R3,  #200; 1 μs
DEL2:   MOV R4,  #125; 1 μs
DEL1:   NOP;          1 μs
        NOP;          1 μs
        DJNZ R4,  DEL1; 2μs
        DJNZ R3,  DEL2; 2μs
        DJNZ R2,  DEL3; 2μs
        RET
```

$$T = \{ 10 \times \{ 1 + 2 + 200 \times [1 + 2 + 125 \times (1 + 1 + 2)] \} + 1 \} \times 1 \mu s$$
$$= 1s$$

循
环
程
序
设
计



[例4-7]把内部RAM中起始地址为BLK1的数据块传送到外部RAM以BLK2为起始地址的区域，直到遇到“#”字符的ASCII码为止。**去掉块长度。**
参考程序如下：

```

                ORG    2000H
BLK1 EQU    30H
BLK2 EQU    1000H
                MOV    SP,#6FH
                MOV    R0  , #BLK1    ; BLK1数据块起始地址
                MOV    DPTR , #BLK2  ; BLK2数据块起始地址
XH:            CLR    C
                MOV    A  , @R0      ; 取数据
                PUSH   ACC
                SUBB   A  , #23H      ; 判是否为“#”字符
                JZ     STOP
                POP    ACC
                MOVX   @DPTR , A      ; 数据传送
                INC    R0
                INC    DPTR
                AJMP   XH              ; 循环控制
STOP:          SJMP   $
                END
```



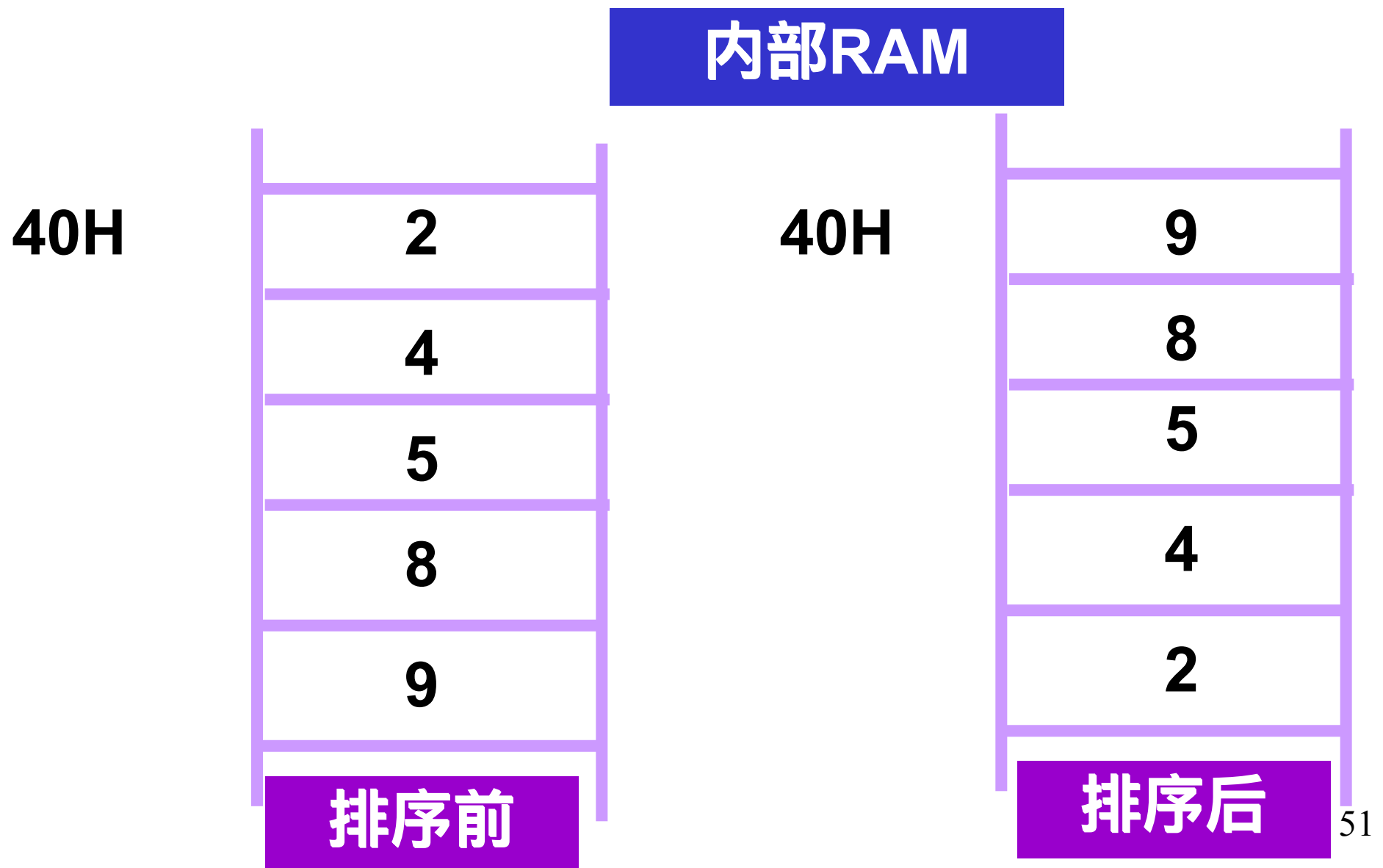
参考程序如下：

END

A thick blue line starts at the top left, goes right, then turns 90 degrees down, then right again. It then curves downwards and to the left, ending with an arrowhead pointing towards the bottom left.

[例4-8] 设单片机内部RAM从40H单元开始存放有100个无符号数，试编写程序能使它们按从**大到小的顺序排列**。（最大的数放在40H单元）

解：排序程序采用“冒气泡”的方法，其“冒气泡”的过程如下：（设N=5时）



2	4	4	4	4	4	5	5	5	5	8	8	8	8	9	9	9	9	2
4	2	5	5	5	5	8	4	8	8	9	9	9	4	9	9	9	9	4
5	5	2	8	8	8	9	8	4	9	9	9	9	9	9	9	9	9	9
8	8	8	2	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	2	9	9	9	9	9	9	9	9	9	9	9	9	9	9

第一轮
比较4次

第二轮
比较3次

第三轮
比较2次

第四轮
比较1次

可以
推知：

对 n 个数，则要进行 $n-1$ 轮扫描，
在第 i 轮扫描中要进行 $n-i$ 次比较。

若将原始数据改为 2 4 9 8 5，则排序过程如下：

2	4	4	4	4	4	9	9	9	9	9	9	9	9
4	2	9	9	9	9	9	4	8	8	8	8	8	8
9	9	2	8	8	8	8	8	4	5	5	5	5	5
8	8	8	2	5	5	5	5	5	4	5	5	5	5
5	5	5	5	2	5	5	5	5	5	5	5	5	5

第一轮比较4次

第二轮比较3次

第三轮比较2次

第四轮
比较1次

可以看出：

第三轮排序中没有发生交换，即第三轮结束后，已经排好了，应结束排序，不必再排第四轮。为此增加一个“**排好序标志位**”，预先将它清0，当产生交换时，将它置1，表示没排好，可以进行下一轮排序，否则，结束排序。

外循环		中间	内循环
计数		条件	计数
轮数		排好序标志	轮内比较次数
1	N-1	有交换置1继续	N-1次
2	N-2	无交换跳出结束	N-2次
3	N-3		N-3次

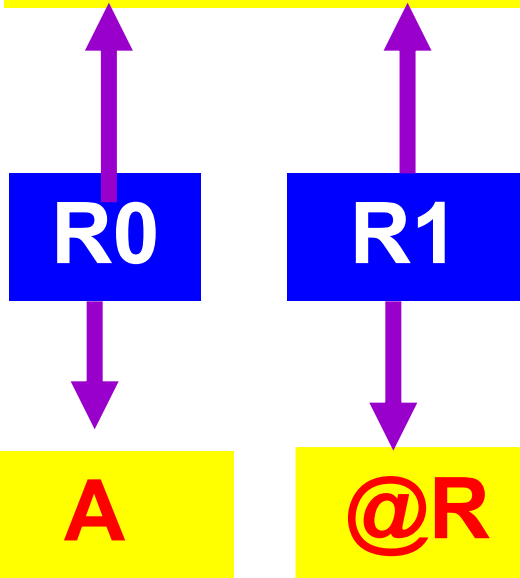
N-2	2		2次
N-1	1		1次

倒数计数

逻辑变量

倒数计数

40H 41H 42H 43H 44H 45H 46H



→

```
XCH A, @R1
MOV @R0, A
```

交换

冒泡法排序

ORG 0100h

mov PSW, #00h

mov R2, #100-1

Loop0: mov R0, #40h
mov R1, #41h
mov 03h, R2
clr F0; 排好序标志清0

Loop1: mov A, @R0
mov 30H, @R1
CJNE A, 30H, LOOP2
sjmp L1

Loop2: JNC L1 ; (A) > ((R1)) 则跳转

xch A, @R1

mov @R0, A

交换

setb F0; 排好序标志置1

L1: inc R0

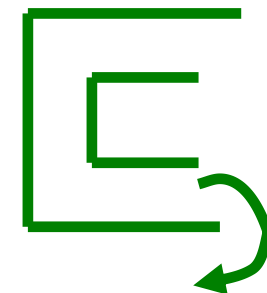
inc R1

djnz R3, Loop1; 轮内次数

jnb F0, Loop3; 排好序标志

djnz R2, Loop0; 轮数

Loop3: sjmp \$
END



4. 2. 4 查表程序设计

- 查表程序

- 以PC为指针
- 以DPTR为指针



查表程序设计

查表程序实现查表算法。

该方法把事先计算或实验数据按一定顺序编成表格，存于程序存储器内，然后根据输入参数值，从表中取得结果。

查表指令：

MOVC A , @A+DPTR

查表前

数据表格表头地址存入DPTR，

要查得的数在表中相对表头地址的**偏移量**送入累加器A，

最后MOVC A , @A+DPTR完成查表,可在64KB内查表

DPTR可以人为修改。

MOVC A , @A+PC

分为三步：

1) 用传送指令把**所查数据的项数**送入累加器A；

2) 使用ADD A , #data指令对累加器A进行修正，data值由下式确定：

PC当前值+data=**数据表头地址**

实际上data值等于查表指令和数据表格之间的字节数；

3) 用指令MOVC A , @A+PC完成查表

由于PC指针只能在A中提供的偏移地址的范围内查表，故查表范围在一页内。

MOV A, #2 ; 2

ADD A, #03H; 计算偏移量 2

MOVC A, @A+PC ; 查表 1

MOV R0, A; 存结果 1

SJMP \$; 2

ASCTAB: DB '0', '1', '2', '3'

PC

PC当前值

补偿值=表头地址-PC当前值

3B

PC当前值+3= ASCTAB +0

表中偏移量

PC当前值+3+2= ASCTAB +2

(A)

74
02
24
03
83
F8
80
FE
30
31
32
33

表头

0

1

2

3

例4-10 已知R0低4位有一个十六进制数（0~F中的一个），编写能把它转换成相应ASCII码并送入R0的程序。

	ORG 0100H	
	MOV A , R0; 取转换值	
	ANL A , # 0FH; 屏蔽高四位	
	ADD A , # 03H; 计算偏移量	
	MOVC A , @A+PC ; 查表	
	MOV R0 , A; 存结果	
	SJMP \$	
	ASCTAB: DB '0', '1', '2', '3'	
	DB '4', '5', '6', '7'	
	DB '8', '9', 'A', 'B'	
	DB 'C', 'D', 'E', 'F'	
	END	

PC当前值
 3B

↑

1B
 2B

]

ASCTAB

30
31
32
33
34
35
36
37
38
39
41
42

60

以PC为指针查表程序

```
TB1:mov A,R2
      add A,R2
      mov R3,A
      add A,#07H
      movc A,@A+PC
      xch A,R3
      add A,#03H+1
      movc A,@A+PC
      mov R4,A
      sjmp $
```

TAB1

1B

2B

1B

1B

2B

15
20
75
86
23
45
10
00
08
83
99
43

```
TAB1:DW 1520H,7586H
       DW 2345H,1000H
       DW 883H,9943H
       DW 4051H,6785H
       DW 4468H,5871H
       END
```



以DPTR为指针查表程序

```
ORG 0100H
ABC EQU 30H
L1: MOV DPTR, #TABL
    MOV A, ABC
    ADD A, ABC; 与双字节Y对应
    MOV R3, A
    MOVC A, @A+DPTR
    XCH A, R3
    ADD A, #01H
    MOVC A, @A+DPTR
    MOV R2, A
    SJMP $
```

```
TABL: DB 01, 00, 01, 00,
        02, 00, 06, 00, 24H,
        00, 20H, 01, 20H, 07,
        40H, 50H; 以BCD码存放
END
```

求 $Y=X!$ (X 在0~7范围),
设ABC单元存放 X , 双字节 Y 存放在寄存器R2R3中,
R3放低字节。



4. 2. 5 散转程序设

散转程序

分支程序的一种，根据某种输入或运算的结果，分别转向各个处理程序。

- 转移指令表
- 转向地址表



使用转移指令表的散转程序

ORG 0000H

mov R2,#2 ; 根据R2散转

Main:mov DPTR,#TAJ1

mov A,R2

add A,R2

jnc NADD ; $2*(R2) < 256$ 转NADD

inc DPH ; 修改DPH

NADD:jmp @A+DPTR

TAJ1:**ajmp PRG0**

ajmp PRG1

ajmp PRG2

ajmp PRG3

sjmp \$

PRG0:mov A,#0

sjmp Halt

PRG1:mov A,#1

sjmp Halt

PRG2:mov A,#2

sjmp Halt

PRG3:mov A,#3

Halt: sjmp \$

END

所有的处理程序入口
PRG0、PRG1、。。。和
散转表TAJ1都必须在同一
2kB范围内。



使用转向地址表的散转程序

```
JMP4: mov R2,#1
      mov DPTR,#TBL4
      mov A,R2
      add A,R2
      jnc NADD; 无进位
      inc DPH; 高位+1
NADD: mov R3,A
      movc A,@A+DPTR
           取转移地址高8位
      xch A,R3
      inc A
      movc A,@A+DPTR
           取转移地址低8位
      mov DPL,A
      mov DPH,R3
```

可以实现64kB的转移，但散转数应小于256。

```
      clr A
      jmp @A+DPTR
TBL4: DW PRG0
      DW PRG1
      DW PRG2
      DW PRG3
PRG0: mov R0,#0
      sjmp Halt
PRG1: mov R0,#1
      sjmp Halt
PRG2: mov R0,#2
      sjmp Halt
PRG3: mov R0,#3
Halt: sjmp $
      end
```

当转向范围
较大时



4. 2. 6 子程序设计

- 子程序设计方法
- 子程序的调用过程与子程序嵌套
- 调用过程中的参数传递
 - 通过寄存器传递
 - 通过堆栈传递
 - 通过数据指针 (DPTR) 传递

能完成某一确定任务，并能被其他程序反复调用的程序段称为**子程序**。



子程序设计方法

- 格式：
 - ；子程序名
 - ；子程序功能：
 - ；入口参数：
 - ；出口参数：
 - ；占用资源：子程序名： .

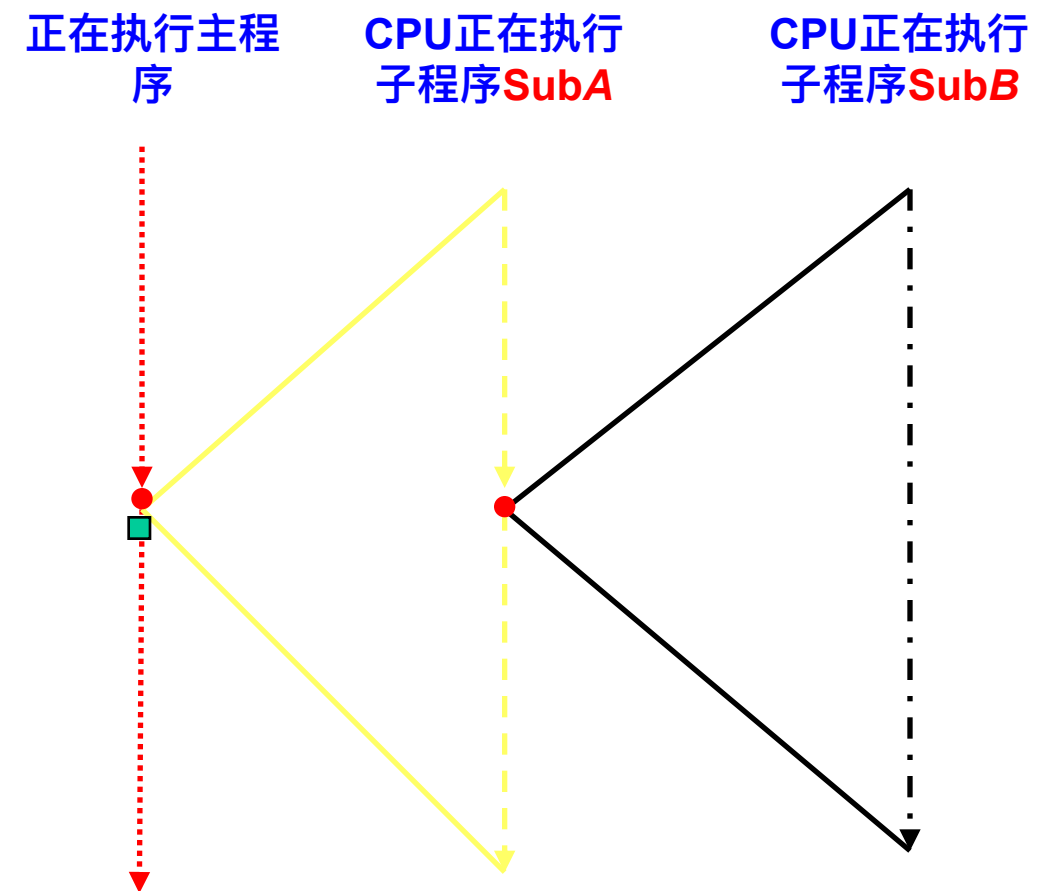
.

.



子程序的调用过程与子程序嵌套

- 主程序要调用子程序时通过call 指令
- 子程序执行完后通过ret 指令返回
- 防止自然进入子程序和自然退出子程序
- 子程序嵌套



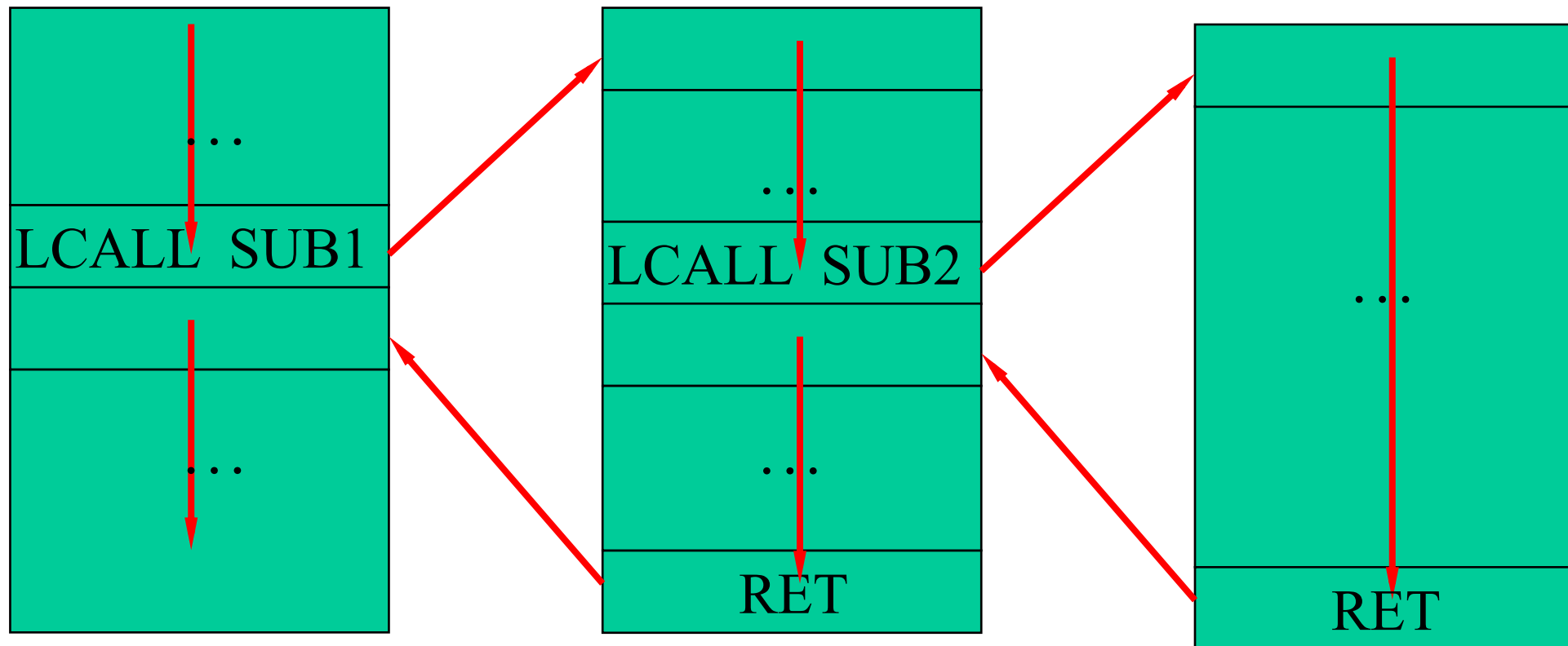
子程序嵌套

子程序嵌套(或称多重转子)是指在子程序执行过程中,还可以调用另一个子程序。

主程序MAIN

子程序SUB1

子程序SUB2



通过寄存器传递

- 方法简单、参数有限
- 例：

```
ORG 0000H  
Main: mov sp,#5FH  
      mov R0,#40H  
      mov R7,#10H  
      lcall Zero  
      sjmp $
```

```
;子程序名: Zero  
;功能:对内部数据区清零  
;入口参数: R0→内部数据区  
;          R7内部数据区长度  
;出口参数: 无  
;占用资源: A,R0,R7  
Zero:clr A  
Loop:mov @R0,A  
      inc R0  
      djnz,R7,Loop  
      ret  
END
```

通过堆栈传递

ORG 0000H

Main:MOV SP,#5FH

mov 70H,#40h

mov 71H,#10H

push 70h

push 71h

lcall Zero

sjmp \$

SP

5FH

40H

10H

61H

PCL

PCH

63H

;子程序名: Zero

;功能:对内部数据区清零

;入口参数:70h内部数据区地址

; 71h内部数据区长度

;出口参数: 无

;占用资源: ,R0,R7,70h,71h

Zero:pop DPH

pop DPL

pop 07h;R7

pop 00h;R0

clr A

Loop:mov @R0,A

inc R0

djnz,R7,Loop

push DPL

push DPH

ret

END

方法2:

ORG 0000H

Main:MOV SP,#5FH

mov 70H,#40h

mov 71H,#10H

push 70h

push 71h

lcall Zero

pop acc

pop acc

sjmp \$

SP

40H

10H

PCL

PCH

ZERO: MOV R0,SP;(SP)=63H

DEC R0

DEC R0

MOV 07H,@R0

DEC R0

MOV 01H,@R0

CLR A

LOOP:MOV @R1,A

INC R1

DJNZ,R7,LOOP

RET

END

5FH

61H

63H



通过数据指针 (DPTR) 传递

- 将待传递参数紧跟在调用指令之后

ORG 0000H

Main:MOV SP,#6FH

NOP

ACALL PRINT;

DB 'THIS IS AN';
(PC)- 保存表首地址

DB 'EXAMPLE'

DB 0AH,0DH,00H

NEXT: NOP

SJMP \$

PRINT:POP DPH

POP DPL;

把表地址放在DPTR中

PPP1:MOV A,#00H

MOVC A,@A+DPTR

INC DPTR

JZ PPPEND

PPP2:MOV P1,A

SJMP PPP1

PPPEND:**JMP @A+DPTR**

END



一般说来：

- 当相互传递的数据较少时，采用寄存器传递方式可以获得较快的传递速度；
- 当相互传递的数据较多时，宜采用存储器或堆栈方式传递；
- 如果是子程序嵌套时，最好是采用堆栈方式。



4. 2. 7 运算程序设计



• 算术运算程序

- 两个多字节数加法
- 多字节BCD码减法
- 单字节右移和加法 (乘法)
- 多字节乘法,用MUL指令
- 伪随机数发生程序

- 逻辑运算程序
- 数据的拼装
- 与逻辑简单实例
- 多字节左移一位



两个多字节数加法

```
ORG 0000H
JIA: MOV R0,#40H
      MOV R1,#50H
      MOV R2,#4
      CLR C
JIA1: MOV A,@R1
      ADDC A,@R0
      MOV @R0,A
      INC R0
      INC R1
      DJNZ R2,JIA1
      MOV F0,C
      SJMP $
      END
```

	2F	5B	A7	C3	H
	(43	42	41	40)	H
+	14	DF	35	B8	H
	(53	52	51	50)	H
<hr/>					
	44	3A	DD	7B	H
	(43	42	41	40)	H

(F0) ← (CY)

用三个指针怎么做？



ORG 1000H

MOV R0,#40H

MOV R1,#50H

MOV R2,#4

SETB RS0

MOV R0,#60H

CLR RS0

CLR C

LOOP: MOV A,@R0

ADDC A,@R1

INC R0

INC R1

SETB RS0

MOV @R0,A

INC R0

CLR RS0

DJNZ R2,LOOP

MOV F0,C

SJMP \$

END

40H



← 0区R0

50H



← 0区R1

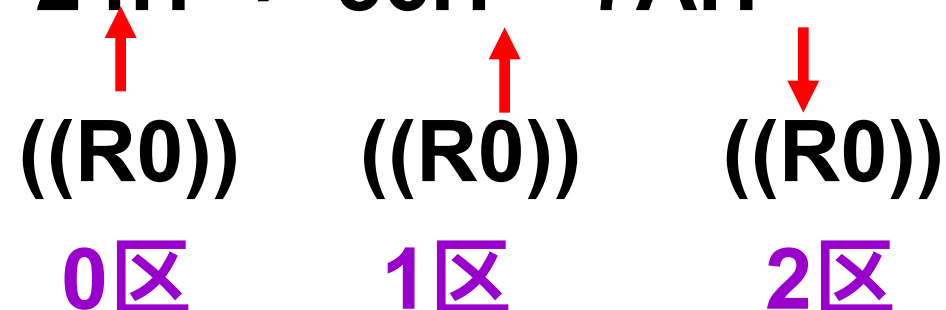


← 1区R0

60H

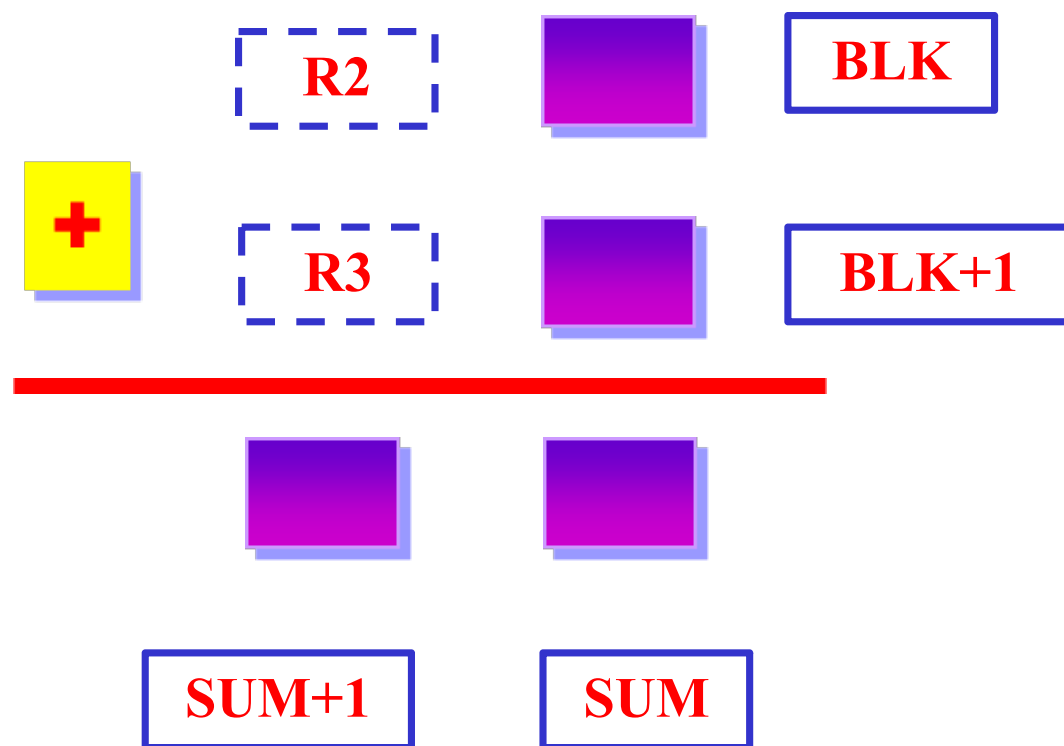
问：下例怎么设计？

例：24H + 56H = 7AH



八位二进制带符号数加法

[例] 两个八位二进制带符号数相加，和超过八位。被加数和加数分别存于BLK和BLK+1单元，和超过八位要占2个单元，设为SUM和SUM+1单元。



将两个8位二进制带符号数扩展成16位带符号数相加，若为8位正数，高8位扩展为00H，否则高8位扩展为0FFH。



```

    ORG 0100H
    BLK EQU 30H
    SUM EQU 40H
    MOV R0, # BLK;
                                R0指向被加数
    MOV R1, # SUM; R1指向和
    MOV R2, 0;      高八位先设为零
    MOV R3, 0
    MOV A, @R0;     取出被加数
    JNB ACC.7, N1;
                                若为正转到N1
    MOV R2, # 0FFH ;
                                若为负高八位为全1

```

```

N1: INC R0      ;修改指针
    MOV B, @R0;  取加数到B
    JNB B.7, N2 ; 若为正转N2
    MOV R3, # 0FFH
N2: ADD A, B;    低八位相加
    MOV @R1, A
    INC R1
    MOV A, R2
    ADDC A, R3;   高八位相加
    MOV @R1, A
    SJMP $
    END

```

多字节BCD码减法

9A9AH

99 88 77 66 H
(43 42 41 40) H
- 44 55 66 77 H
(53 52 51 50) H

ORG 0000H

Jia: mov R0,#40h
mov R1,#50h
mov R2,#4

clr C

Jia1:mov A,#9Ah
subb A,@R1
add A,@R0
da A

CPL C ;进位求反

mov @R0,A

inc R0

inc R1

djnz R2,Jia1

mov F0,C

sjmp \$

END

99999999AH
- 44556677H

???????????

9AH

-77H

23H

+66H

89H

CY=1

9AH

- 66H

- 1

33H

+77H

AAH

+66H

10H

CY=0

9AH

- 55H

45H

+88H

CDH

+66H

33H

CY=0

9AH

-44H

56H

+99H

EFH

+66H

55H

CY=0



数据的拼装

X7X6X5X4X3X2X1X0 (20H)

y7y6y5y4y3y2y1y0 (21H)

y2y1y0X4X3X2X1X0 (30H)

ORG 0000H

START: MOV 30H,20H

ANL 30H,#00011111B

MOV A,21H

SWAP A

RL A

ANL A,#11100000B

ORL 30H,A

SJMP \$

END



与逻辑简单实例

P1.7 P1.6 P1.5 P1.4 P1.3 P1.2 P1.1 P1.0



执行主程序的逻辑函数： $F = P1.6 \cdot P1.4 \cdot P1.2$

P1.2——准备就绪信号，为“1”表示就绪

P1.4——主回路工作正常信号，为“1”表示正常

P1.6——启动开关闭合信号，为“1”表示闭合

程序片段：

```
WAIT: MOV A,P1    ;(P1)=XXXXXXXXXB
```

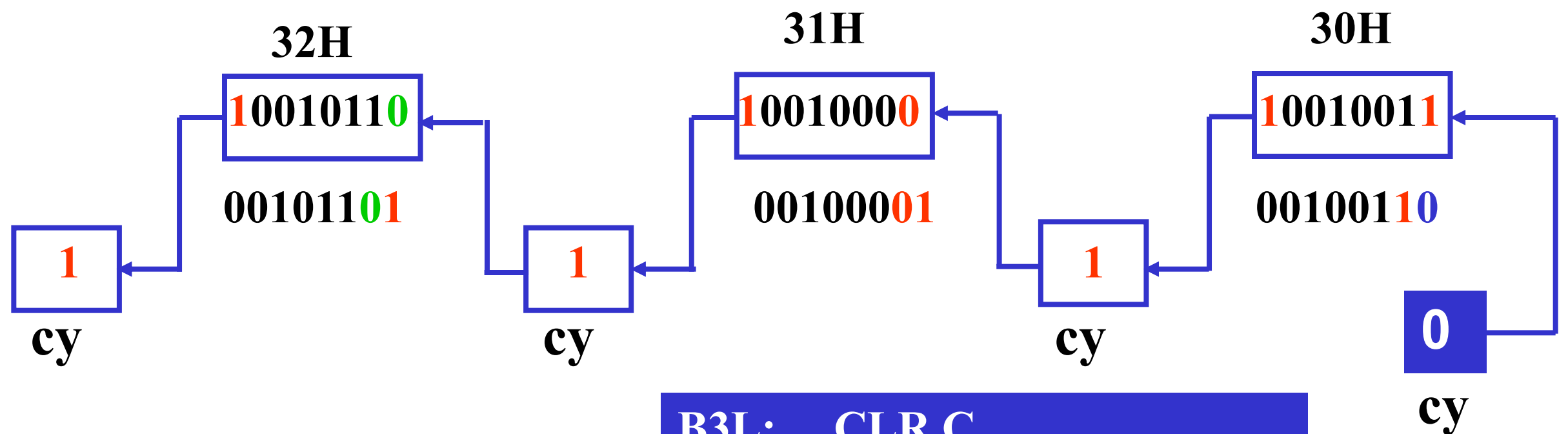
```
    ANL A,#54H;    01010100B
```

```
    CJNE A,#54H,WAIT; F=1?
```

```
MAIN:      .....    ;主程序
```



多字节左移一位



ORG 2000H

MAIN : MOV SP,#5FH

MOV R0,#30H

MOV R2,#3

ACALL B3L

SJMP \$

B3L: CLR C

LOOP: MOV A,@R0

RLC A

MOV @R0,A

INC R0

DJNZ R2,LOOP

RET

END



单个字节右移和加法 (乘法)

$$\begin{array}{r} 1011 \\ \times 1001 \\ \hline 1011 \\ +1011 \\ \hline 1100011 \end{array}$$

运算规则:

- ▶ 当乘数的当前位为1，加被乘数后右移1位；
- ▶ 当乘数的当前位为0，直接右移1位。

部分积	乘数
0000	100 <u>1</u>
+1011	
1011 →	
0101 →	<u>1100</u>
0010 →	<u>1110</u>
0001	<u>0111</u>
+1011	
1100 →	
0110	0011

乘积低位

用MUL指令做多字节乘法

例： 3344H*56H=?

	33	44H
	*	56H
	<hr/>	
	4456H	4456L
+	3356H	3356L
	<hr/>	
	(32H)	(31H) (30H)

ORG 3000H

MOV A,#44H

MOV B,#56H

MUL AB

MOV 30H,A ;(A)=4456L=(30H)

MOV 31H,B ;(B)=4456H=(31H)

MOV A,#33H

MOV B,#56H

MUL AB ;(A)=3356L

ADD A,31H ;4456H+3356L=(31H))

MOV 31H,A

CLR A

ADDC A,B ;(A)=3356H+CY

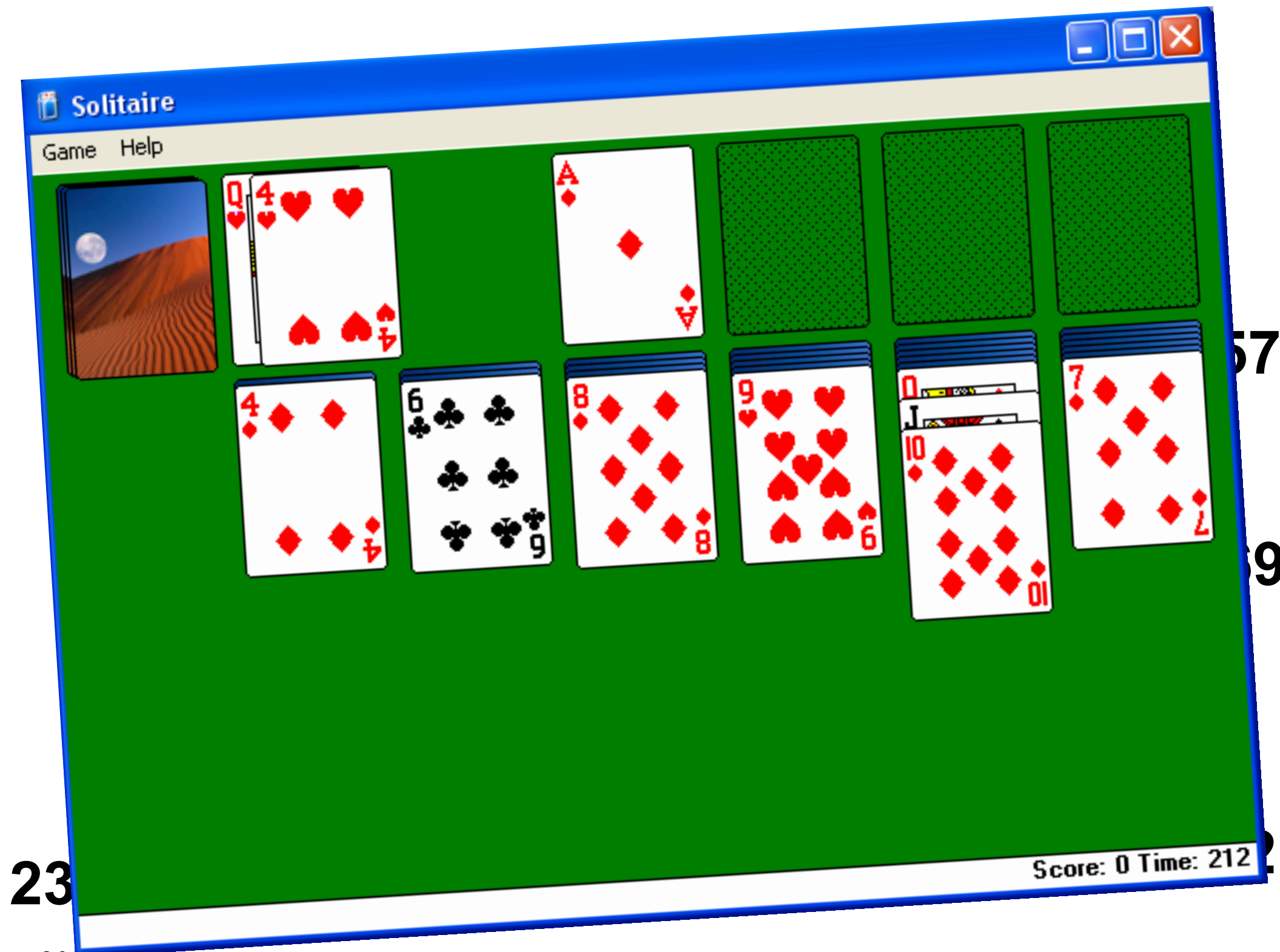
MOV 32H,A

SJMP \$

END



伪随机数



23

57

9

2

Testing of Digital Systems 数字系统

Random noise
随机噪声

Pseudo-random
numbers 伪随机数



伪随机数 (迭代线性同余法)

$$X_{i+1} = (A * X_i) \bmod m$$

Multiplier
(定常乘数)

209

Constant
(除数)

107

69

23

$X_0 = \text{Seed}$
(无符号整数)

150

230

189

11

132

伪随机数

$$X_0 = 4$$

$$X_{i+1} = (5 * X_i) \bmod 9$$

i	X_i	$5 * X_i$	X_{i+1}
0	4	20	2
1	2	10	1
2	1	5	5
3	5	25	7
4	7	35	8

伪随机数

$$X_{i+1} = (A * X_i) \bmod 256$$

1	1	0	0	0	1	0	0	1	1	0	1	1	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Divide a 16-bit number by 256

Modulus is low byte of the 16-bit number

余数就是原数中的低字节

$$X_{i+1} = \text{Low_byte}(A * X_i)$$

; Let Seed X(0) = 57h, multiplier A = 100 (64h)

SEED EQU 57h

MOV 40h, #SEED ; Seed into 40h

MOV R0, #40h

MOV A, @R0 ; Seed into A

LOOP: MOV B, #64h ; Multiplier = 100

MUL AB ; Multiplier * X(i)

当出现0
时?

用FFH代替0

INC R0

MOV @R0, A ; X(i+1) into RAM

CJNE R0, #70h, LOOP

SJMP \$

Pseudo-random numbers

i	X(i)	100*X(i)	Low_byte(100*X(i))	X(i+1)
0	57h	21FCh	FCh	FCh
1	FCh	6270h	70h	70h
2	70h	2BC0h	C0h	C0h
3	C0h	4B00h	00h	FFh
4	FFh	639Ch	9Ch	9Ch

23

91

130

230

189

11

132



第四章 结束

下次见