

# 微型计算机原理与应用

材料学院 文世峰

Roya\_wen@hust.edu.cn

学时：  $32 = 28$  (授课) + 4 (实验)

考核方式：考试 + 实验 + 作业

使用教材：

《单片微型计算机原理与应用》，胡乾斌等，  
华中科技大学出版社

# 有关本课程学习的几点建议

- 先导课程：数字电路、计算机基础、模拟电路  
这方面知识掌握得不够好的请自己补上
- 本课程是一门实践性、应用性很强的学科  
仅仅听懂还不够,重在培养动手能力
- 硬件/软件同样重要、不可偏废  
硬件是骨架，软件是灵魂和思想
- 预习、听课、复习、作业、实验环环都重要  
用科学的方法学习

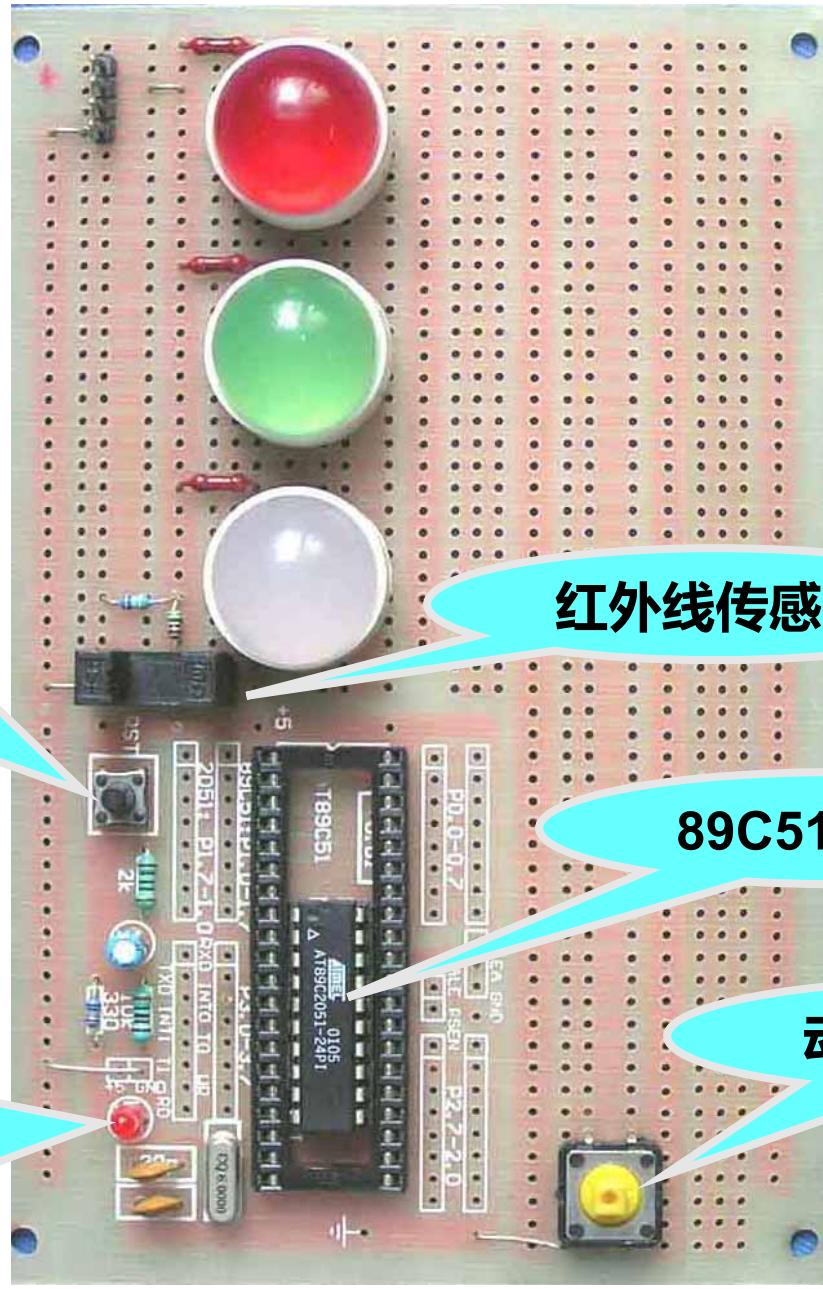
# 授课内容与课时分配

- 1、单片微型计算机概述 (2学时)
- 2、MCS-51单片机的内部结构 (4学时)
- 3、MCS-51的指令系统 (4学时)
- 4、汇编语言程序设计 (4学时)
- 5、存储器 (2学时)
- 6、中断系统 (2学时)
- 7、输入和输出 (2学时)
- 8、定时器/计数器 (2学时)
- 9、串行通信及其接口 (2学时)
- 10、D/A和A/D转换接口 (2学时)
- 11、显示器、键盘、打印机接口 (2学时)

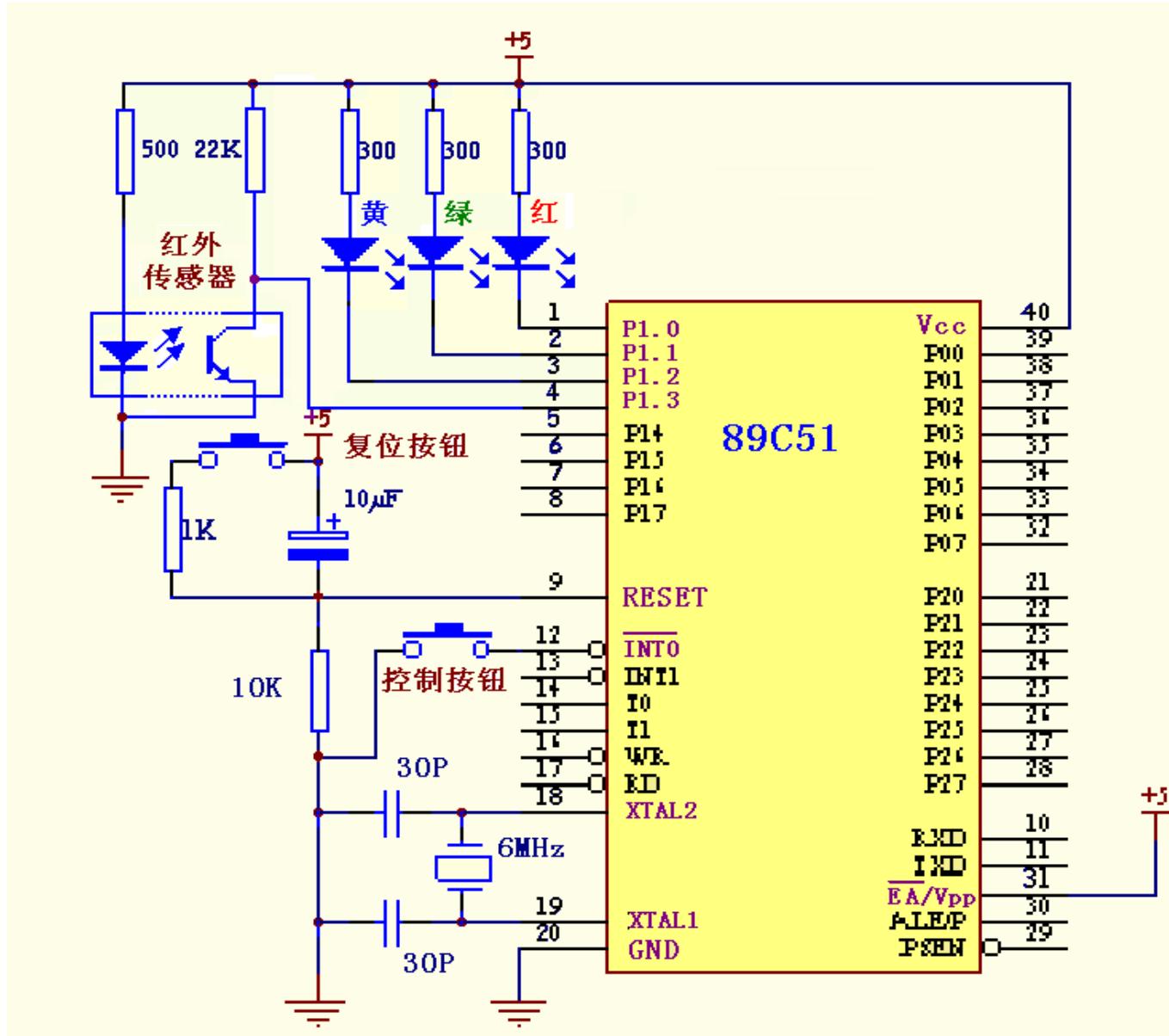
## § 1.1 概 述

在工业、农业、军事、安保、金融、仪器仪表、航空航天、医疗、通讯、办公设备、娱乐休闲、健身、体育竞赛、服务领域……微型计算机应用技术已经无处不在，正迅速改变着人们传统的生产和生活方式。

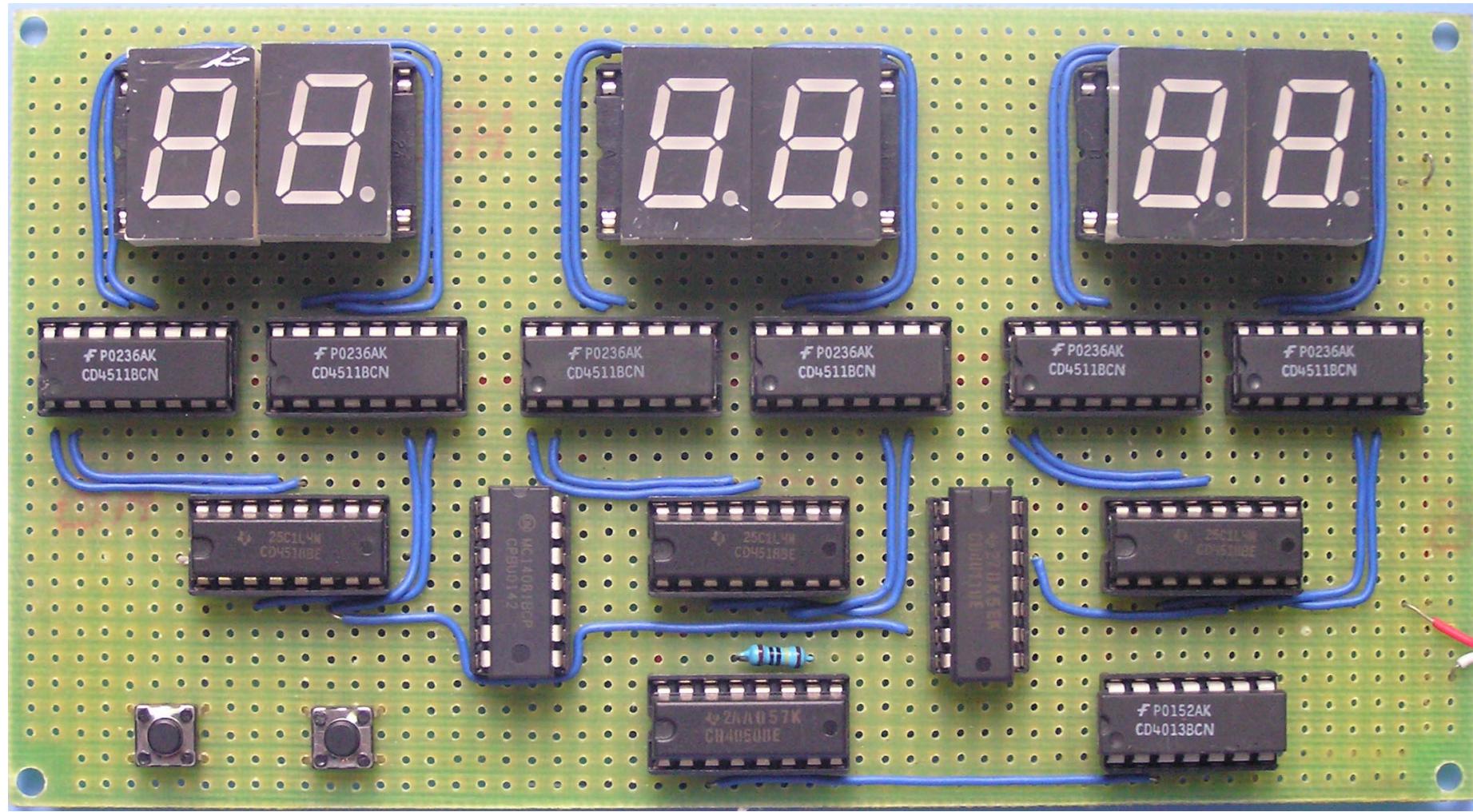
# 一个简单的测控应用实例



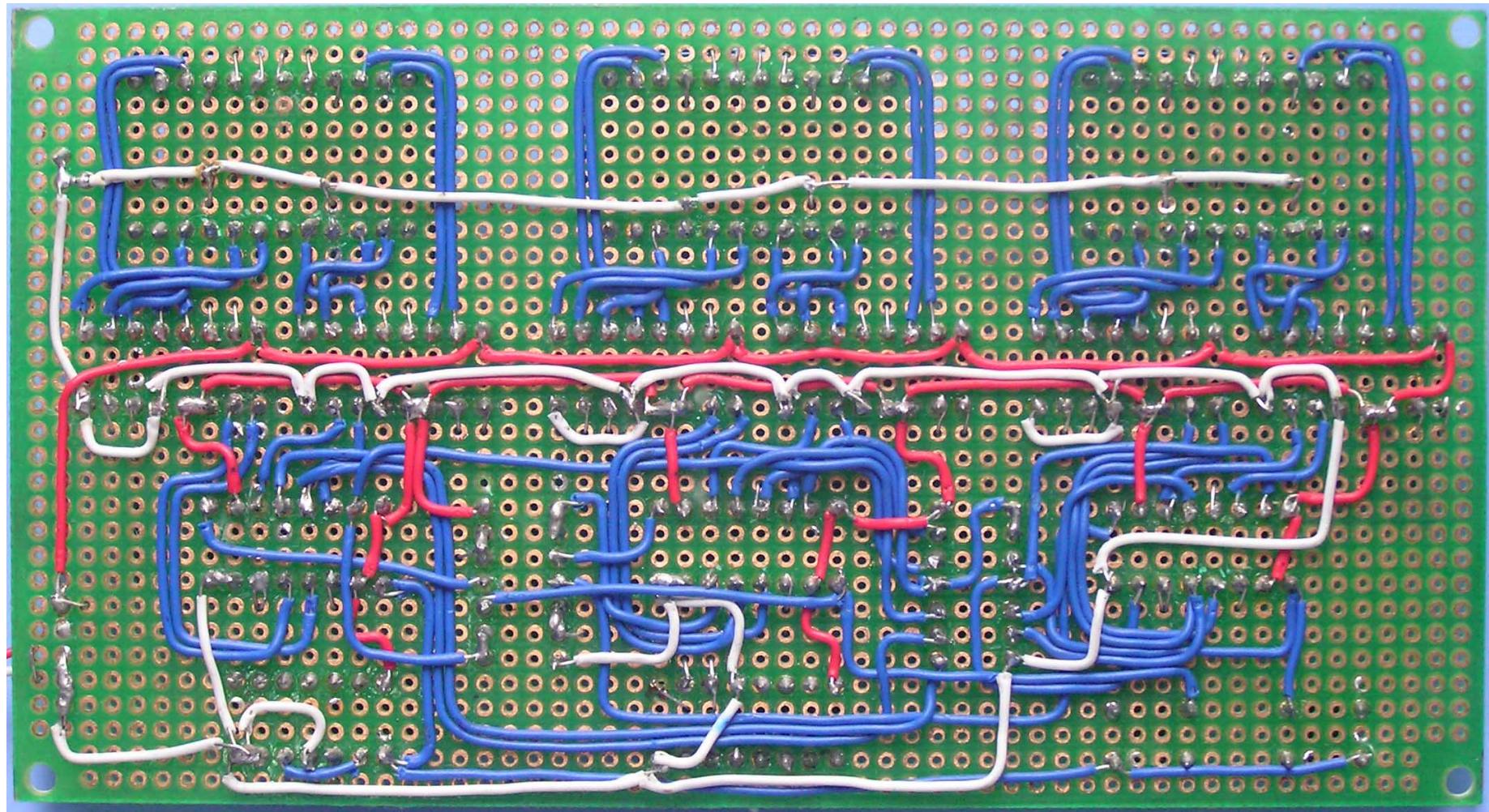
# 测控实例原理图



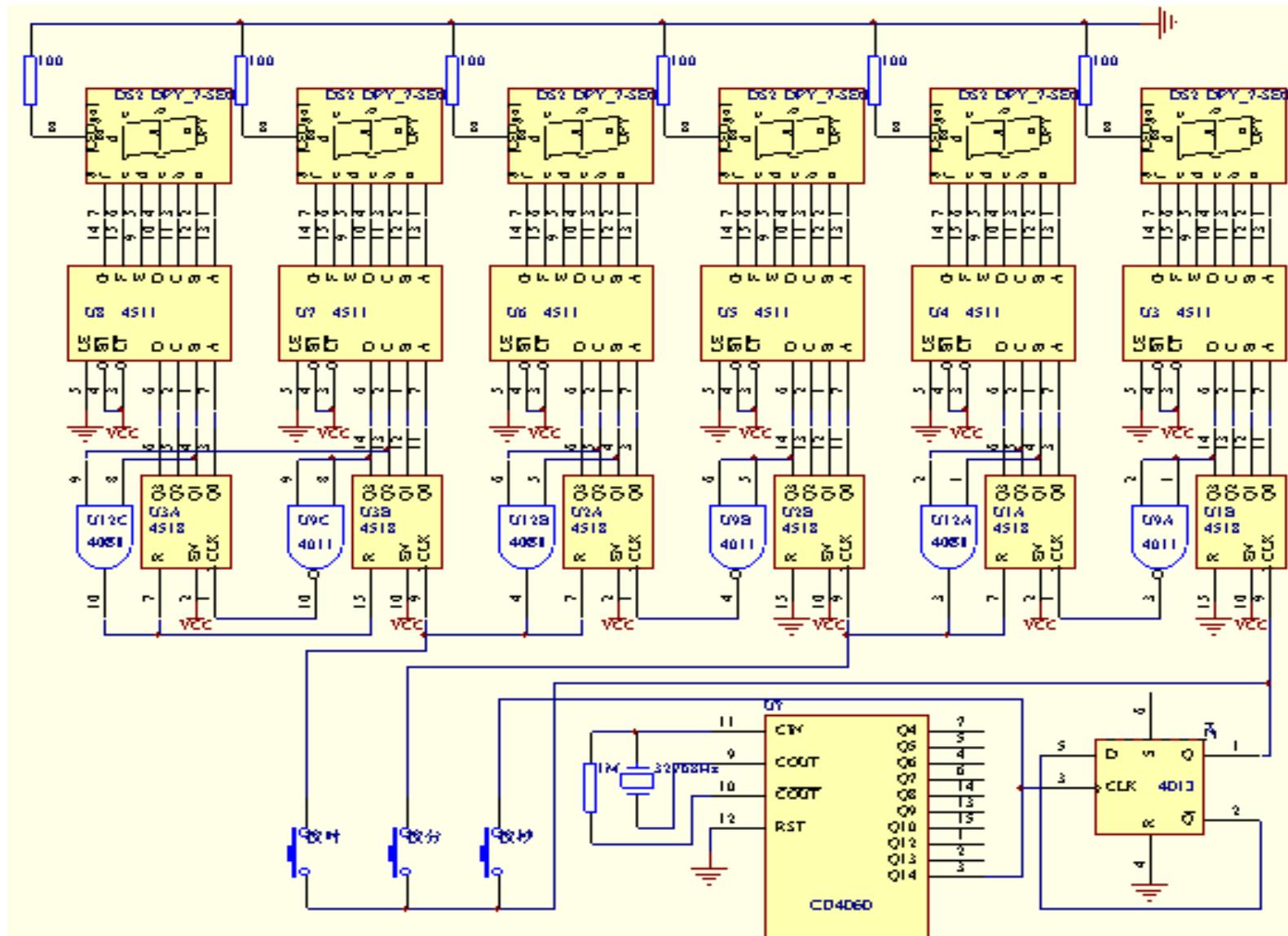
# 数字电路中由13片IC组成的数字钟



# 数字电路中由13片IC组成的数字钟



# 数字电路中由13片IC组成的数字钟

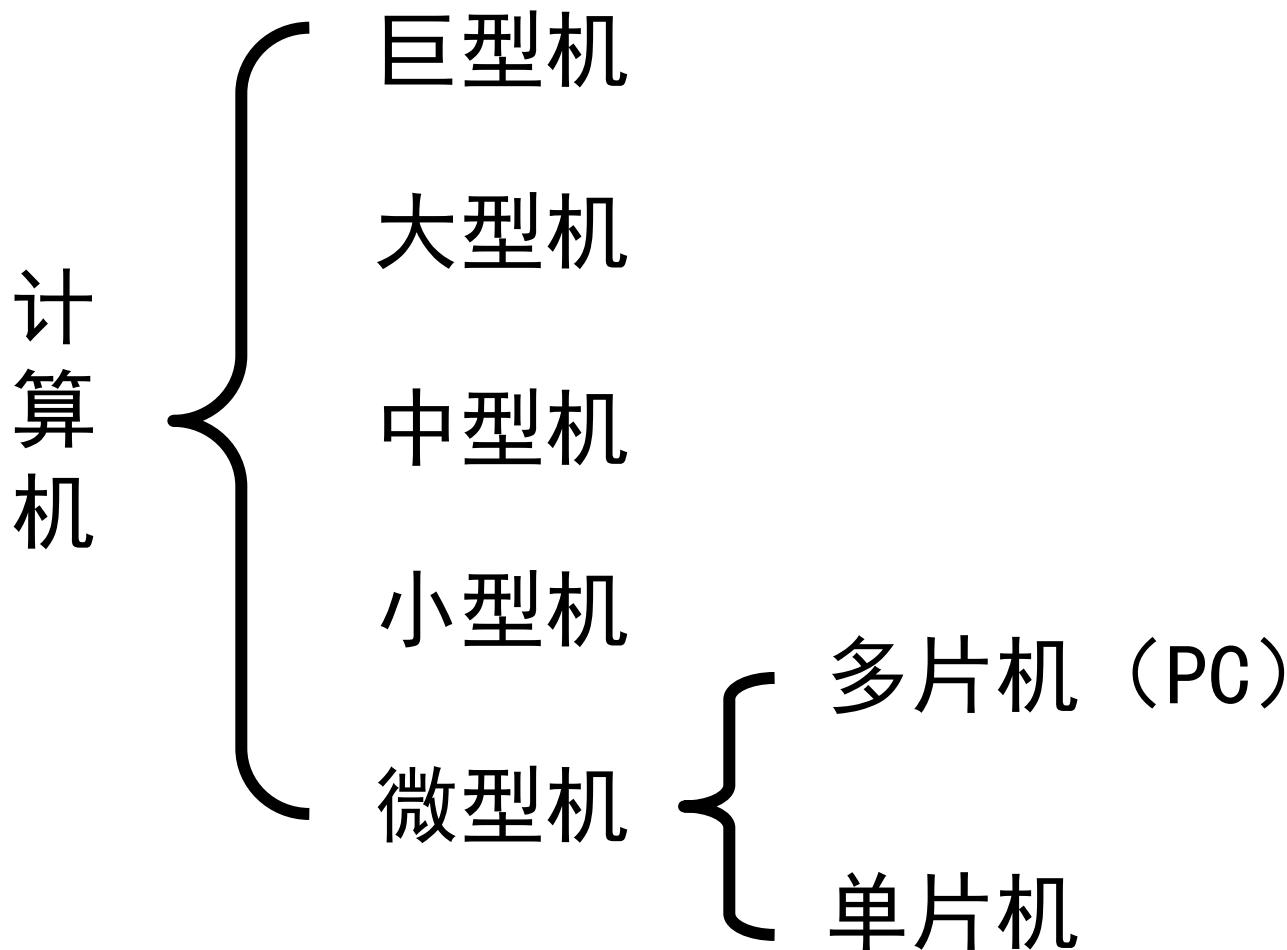


# 单片IC（单片机）电子钟



## § 1.2

# 微型计算机与单片机



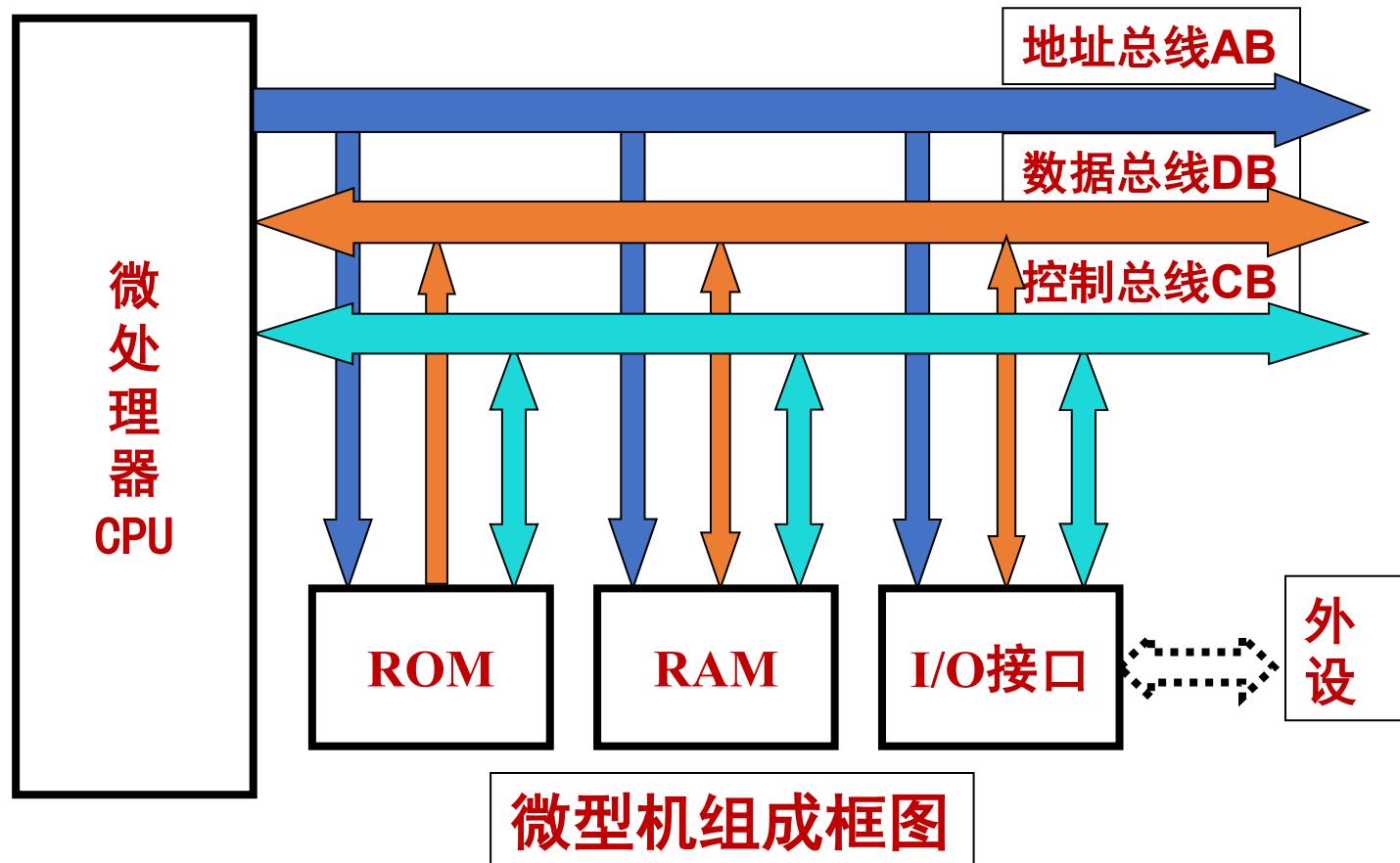
$\mu P$ ,  $\mu C$ ,  $\mu CS$

- 微处理器 (Microprocessor,  $\mu P$ )
  - 微型化的中央处理器——CPU或MPU
  - 组成：
    - ALU, 算术逻辑部件
    - 累加器和通用寄存器组
    - PC (程序计数器) , IR (指令寄存器) , ID (指令译码器)
    - 时序和控制电路
  - 功能：运算、译码、访问存储器和外设、定时和控制、中断响应

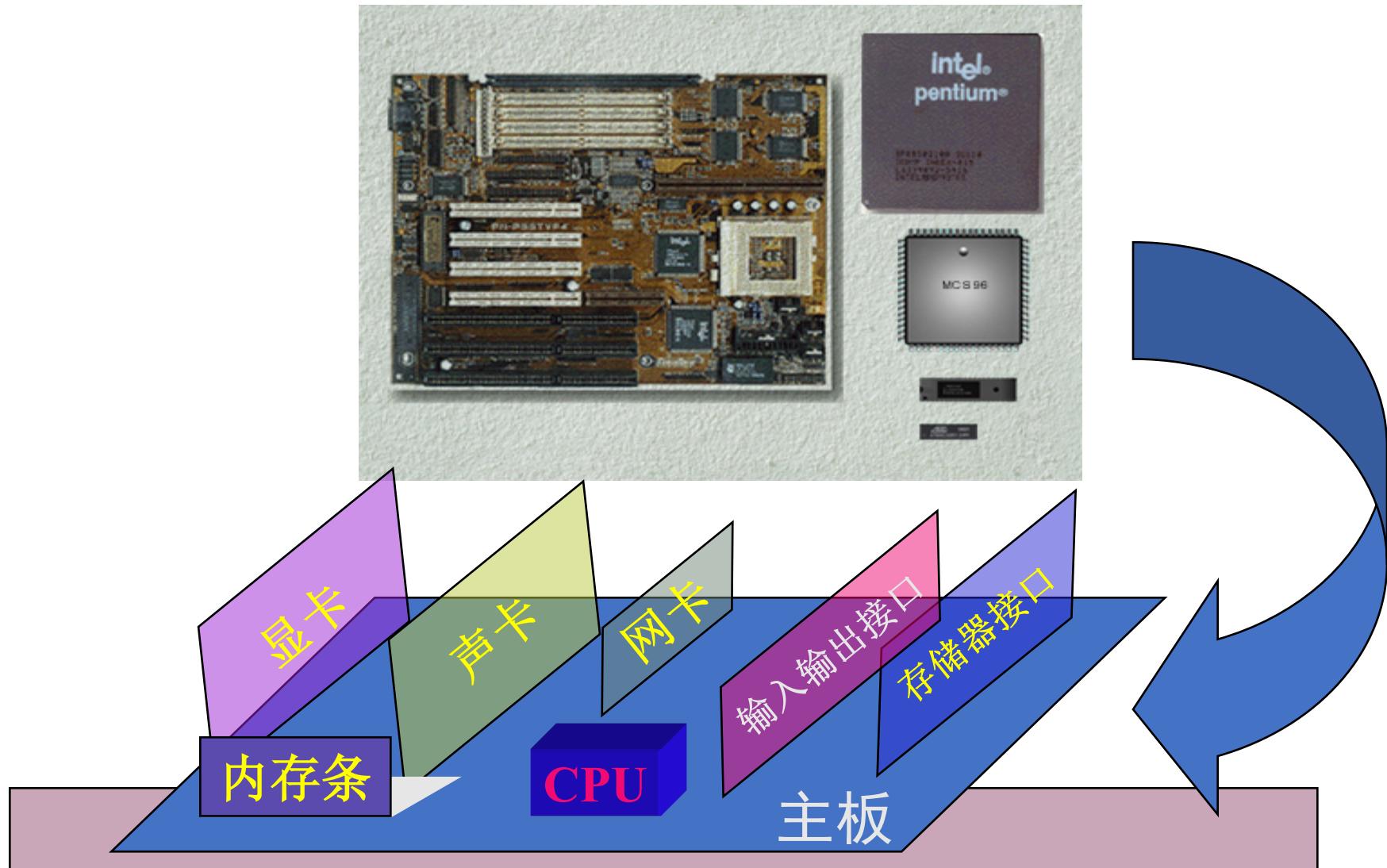
# $\mu P$ , $\mu C$ , $\mu CS$

- 微型计算机 (Microcomputer,  $\mu C$ )

- $\mu P + RAM/ROM + I/O + BUS$

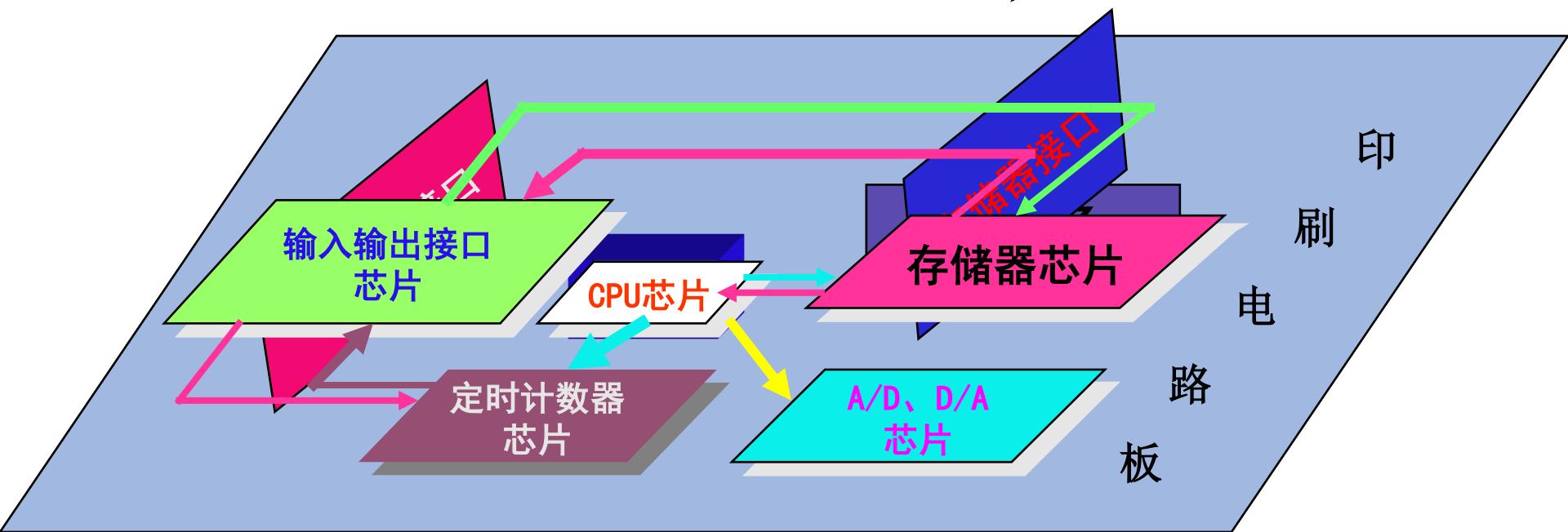


# 计算机系统通常由多块印刷电路板制成：

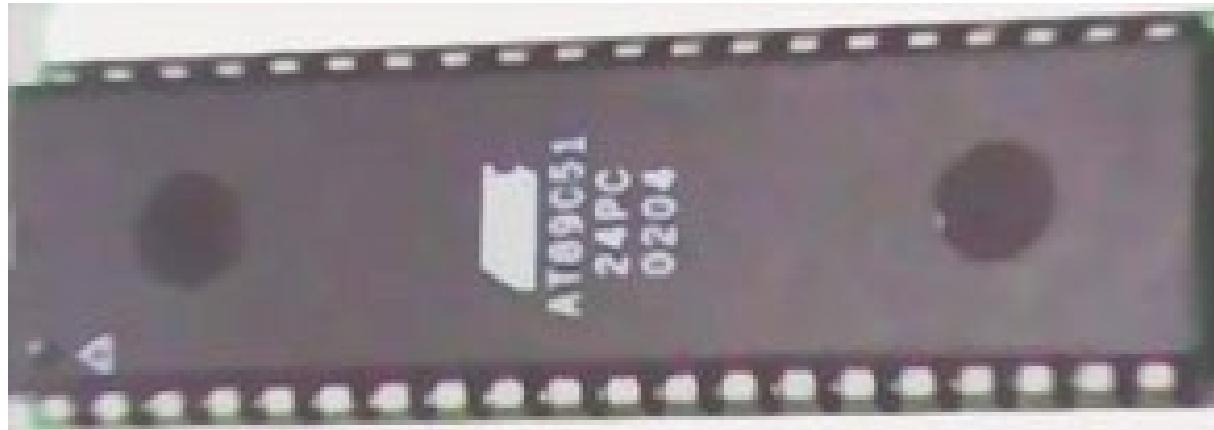
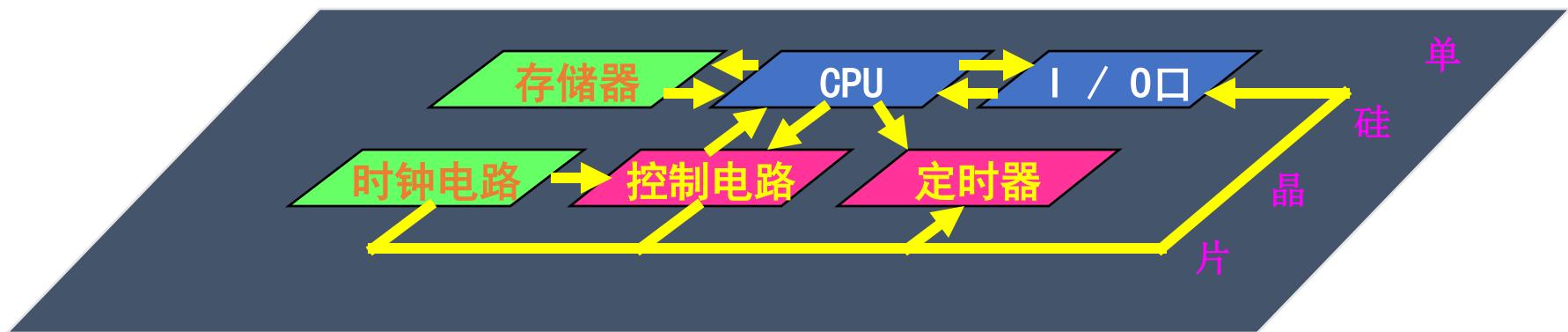


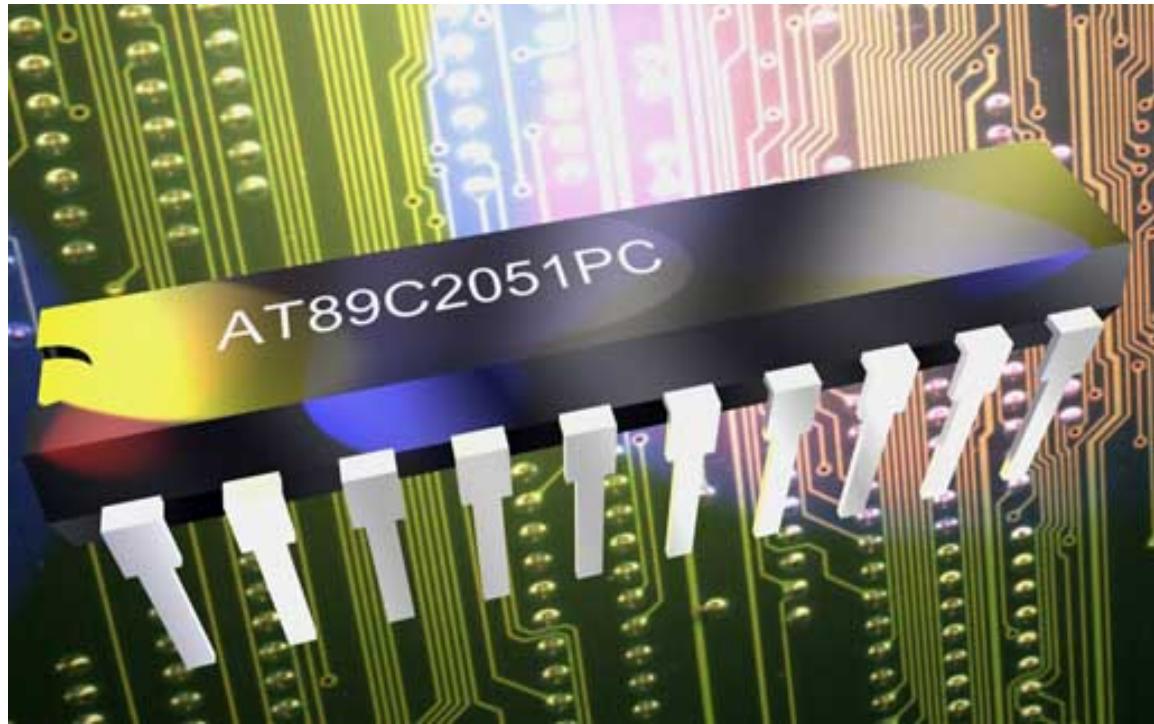
多板机 (PC)

# 单板机



单片机





- ✓ 2K字节的Flash **E<sup>2</sup>PROM**
- ✓ 128字节的**RAM**
- ✓ 15根**I/O引线** (P1. 0–P1. 7; P3. 0–P3. 5, P3. 7)
- ✓ 2个16位**定时器/计数器**
- ✓ 1个五向量两级**中断结构**
- ✓ 1个全双工**串行口**
- ✓ 1个精密**模拟比较器**
- ✓ 片内**振荡器**和**时钟**电路

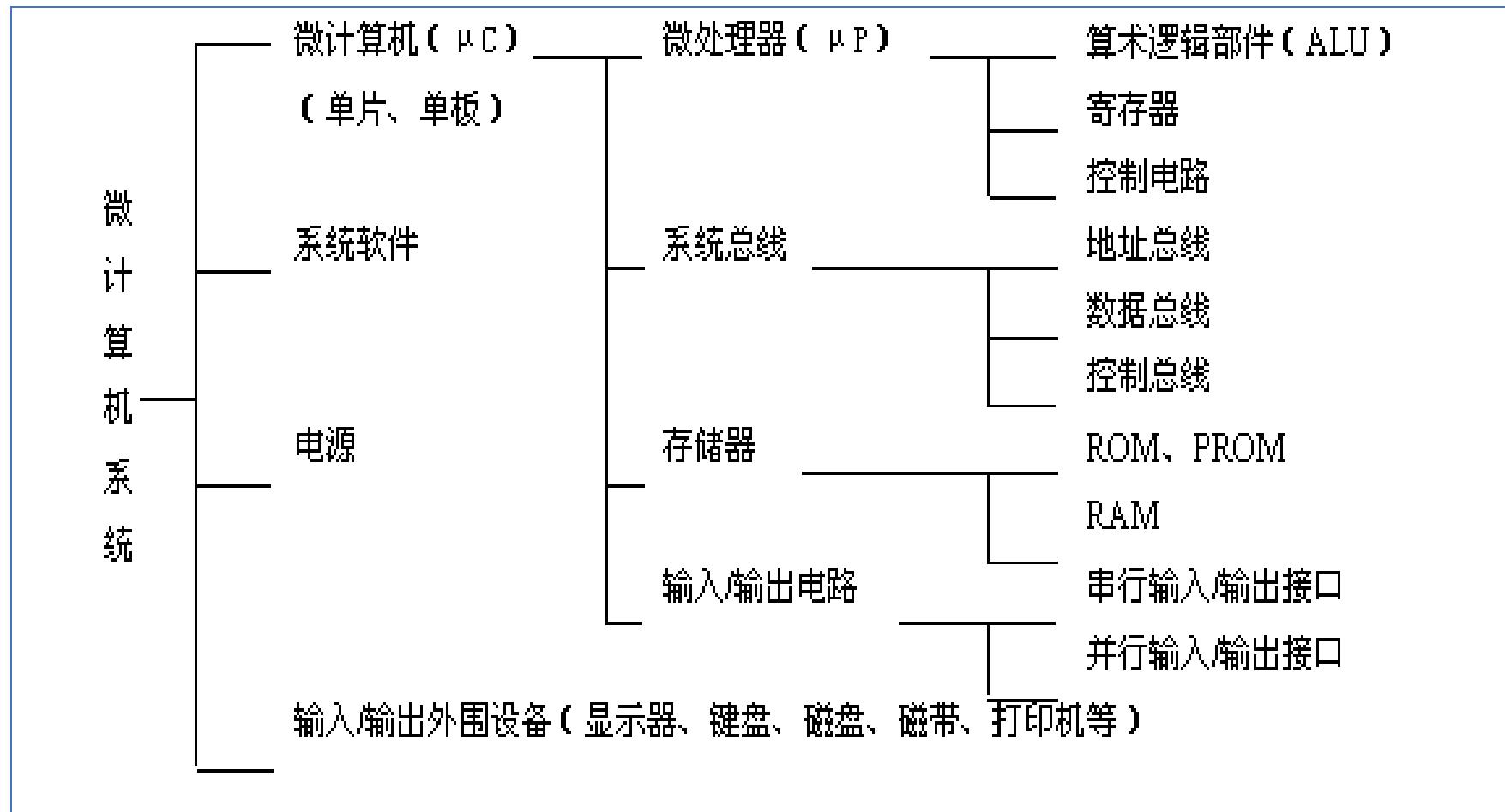
$\mu P$ ,  $\mu C$ ,  $\mu CS$

## 微型计算机系统 (Microcomputer System, $\mu CS$ )

组成：

- $\mu C$ : 微型计算机
- 软件: 操作系统软件 + 应用软件
- 外设: 数据的输入和输出

# $\mu P$ , $\mu C$ , $\mu CS$



微处理器、微型计算机和微型计算机系统的关系

# 单片机与PC机之异同 (1)

## 组成：

CPU(进行运算、控制)      RAM(数据存储器)

I/O口(串口、并口等)      ROM(程序存储器)

PC机： 上述部件以电路板形式安装在主板上。

单片机： 上述部件被集成到单芯片中。

通用PC包括： 键盘、显示器、鼠标、硬/软/光驱、  
音箱、打印机、扫描仪…等外设。

单片机只是一片集成电路。(……100、48、40、  
32、28、20、16、8条引脚)

# 单片机与PC机之异同 (2)

## **功能：**

**PC机：** 数据运算、采集、处理、存储、传输；

**单片机：** 控制（或受控于）外设。

通用计算机擅长于数据运算、采集、处理、存储和传输。

单片机的专长是测控，往往嵌入某个仪器/设备/系统中，使其达到智能化的效果。

# 单片机与PC机之异同 (3)

## 应用特点：

### PC机（微机）：

体积大，功耗大，价格高，用途较固定，属通用计算机。易于学习掌握和使用，但用于控制时必须制作或购买专用的接口卡，并编制专门的应用软件。

### 单片机：

体积小，功耗小，价格低，用途灵活，无处不在，属专用计算机。是一种特殊器件，需经过专门学习方能掌握应用，应用中要涉及专业的硬件和软件。

## § 1.3 单片机概述

★单片机SCMC (Single Chip MicroComputer)

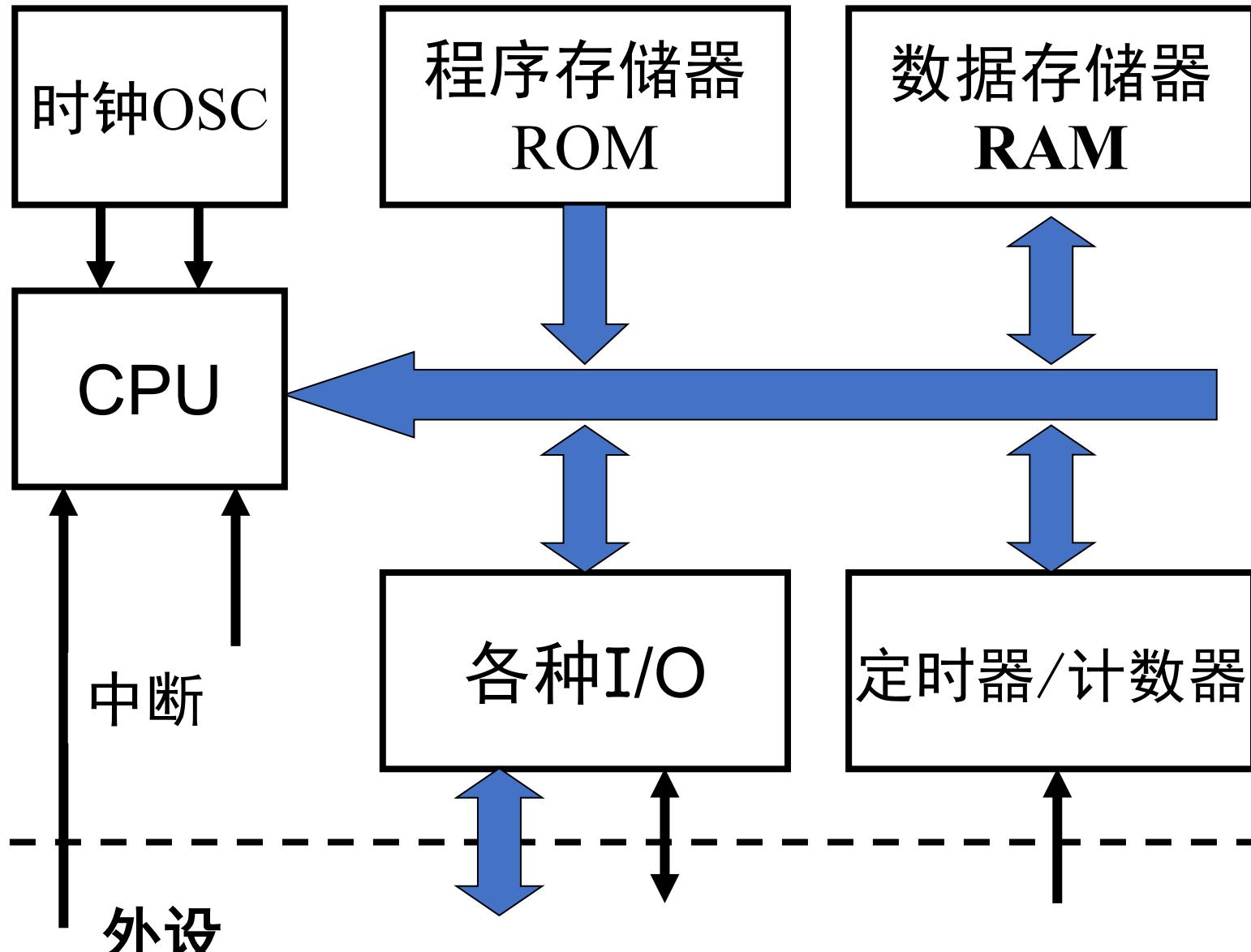
----属于微型机的一种

----具有一般微机的基本组成和功能

单片机是应工业测控的需要而诞生的，其结构与指令功能都是按照工业控制要求设计。也称之为：

- 单片微控制器 (Single Chip Microcontroller)
- 微控制器MCU (MicroController Unit )
- 嵌入式微控制器 (Embedded Microcontroller)





MCS-51单片机组件框图

# 一、单片机的特点 (集成度高、可靠性高、性价比高)

- (1) 优异的性能价格比。
- (2) 集成度高、体积小、有很高的可靠性。

单片机把各功能部件集成在一块芯片上，内部采用总线结构，减少了各芯片之间的连线，程序多采取固化形式，可提高单片机的可靠性与抗干扰能力。另外，其体积小，对于强磁场等环境易于采取屏蔽措施，适合于在恶劣环境下工作。

### (3) 控制功能强。

为了满足工业控制要求，一般单片机的指令系统中均有极丰富的转移指令、I/O口的逻辑操作以及位处理功能。单片机的逻辑控制功能及运行速度均高于同一档次的微机。

### (4) 单片机的系统扩展、系统配置较典型、规范，容易构成各种规模的应用系统。

## 二、单片机的应用

- ① **工业自动化方面**（过程控制、数据采集和测控技术、机器人技术、机械电子计算机一体化技术）
- ② **仪器仪表方面**（测试仪表和医疗仪器—数字化、智能化、高精度、小体积、低成本、便于增加显示报警和自诊断功能）
- ③ **家用电器方面**（冰箱、洗衣机、空调机、微波炉、电视机、音像设备等）
- ④ **信息和通信产品方面**（计算机的键盘、打印机、磁盘驱动器；传真机、复印机、手机、电话机、考勤机）
- ⑤ **军事方面**（飞机、大炮、坦克、军舰、导弹、火箭、雷达等）

### 三、单片机的发展

产生：与微处理器（1971年研制成功）大体相同

可分为四个发展阶段：

- ◆74~76年初级阶段：工艺限制，RAM小、CTC和ROM需外接，仙童的F8：8位CPU、64字节RAM和2个并行口+3851（1K ROM、定时器/计数器和2个并行I/O）
- ◆76~78年低性能阶段：中断处理比较简单（优先级少）、ROM和RAM寻址不大于4K，MCS-48
- ◆78~82年高性能阶段：中断系统、I/O口功能强，存储容量达64K，MCS-51, 52
- ◆82~现在：巩固发展8位，推出16位、32位单片机阶段，MCS-80960

国内外主流产品：MCS-51系列产品

## § 1.4 常用单片机系列

### 1、综述

自单片机诞生以来，已有70多个系列的近500个机种。国际上较有名、影响较大的公司及产品有：

- **Intel**公司的MCS-48、MCS-51、MCS-96系列；★
- **Motorola**公司的6801、6802、6803、6805、68HC11系列产品；
- **Zilog**公司的Z8、Super8系列产品；
- **仙童**（Fairchild）公司和Mostek公司的F8、3870系列产品；

- NEC公司的μCOM-87系列产品；
- Rockwell公司的6500、6501系列产品。
- Atmel、Philips、LG等公司的51兼容机。★

20世纪80年代中期以后，Intel公司以专利转让的形式将8051内核技术转让给许多半导体芯片生产厂家，如Atmel、Philips、analog等。这些厂家生产的芯片是MCS-51系列的兼容产品，即与MCS-51指令系统兼容的单片机。

## 2、MCS-51系列和AT89系列单片机

### （1）MCS-51系列

MCS-51是指由美国Intel公司生产的一系列单片机的总称，这一系列单片机包括了很多品种，如**8031**, **8051**, **8751**, **8951**, **8032**, **8052**, **8752**, **8952**等。其中**8051**是最早最典型的产品，该系列其它单片机都是在**8051**的基础上进行功能的增、减、改变而来。

# MCS-51系列单片机分类

资源 配置 子 系列	片内ROM形式				片 内 ROM 容 量	片 内 RAM 容 量	定时/ 计数器	中 断 源
	无	ROM	EPR OM	E <sup>2</sup> PR OM				
51子系列	8031	8051	8751	8951	4KB	128B	2×16	5
52子系列	8032	8052	8752	8952	8KB	256B	3×16	6

P1.0	1	40	Vcc
P1.1	2	39	P0.0
P1.2	3	38	P0.1
P1.3	4	37	P0.2
P1.4	5	36	P0.3
P1.5	6	35	P0.4
P1.6	7	34	P0.5
P1.7	8	33	P0.6
RST/V <sub>PD</sub>	9	32	P0.7
RXD/ P3.0	10	31	EA/Vpp
TXD/ P3.1	11	30	ALE/PROG
INT0/ P3.2	12	29	PSEN
INT1/ P3.3	13	28	P2.7
T0/ P3.4	14	27	P2.6
T1/ P3.5	15	26	P2.5
WR/ P3.6	16	25	P2.4
RD/ P3.7	17	24	P2.3
XTAL2	18	23	P2.2
XTAL1	19	22	P2.1
Vss	20	21	P2.0

8051

## (2) AT89系列

AT89C51是在我国比较流行的单片机，它是由美国Atmel公司开发生产的。虽然也是Intel公司授权的MCS-51的核心技术，但是功能或多或少都会改变，来满足不同的需求。

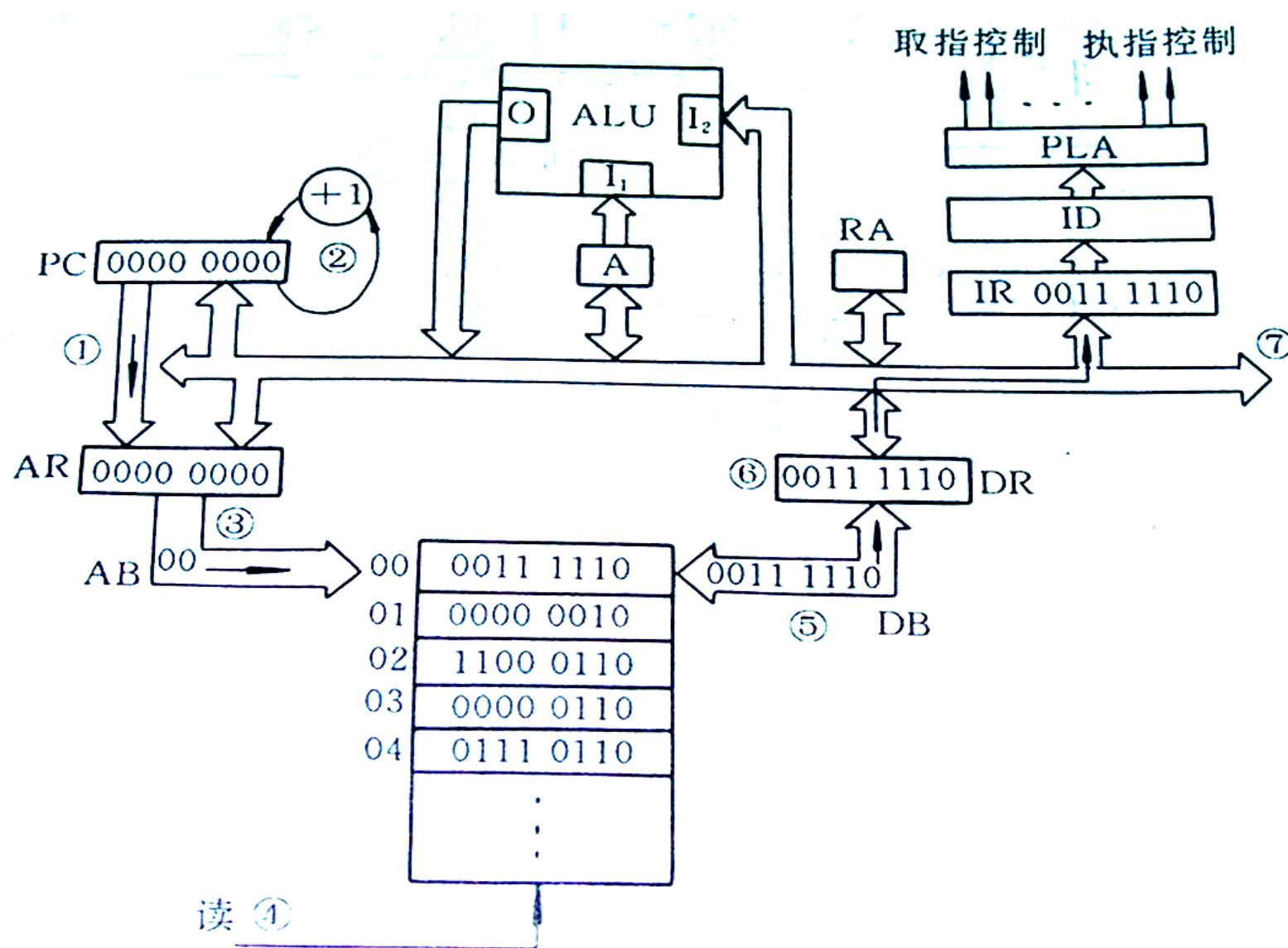
## § 1.5 单片机的工作过程

- 预存程序式的冯·诺依曼结构
- 程序：一系列指令的有序集合
- 指令：计算机执行某种操作的命令  
    指令 = 操作码 + 操作数
- 指令的执行包括两个阶段：
  - 取指令，取得操作码后译码
  - 执行指令，取出操作数对其进行操作

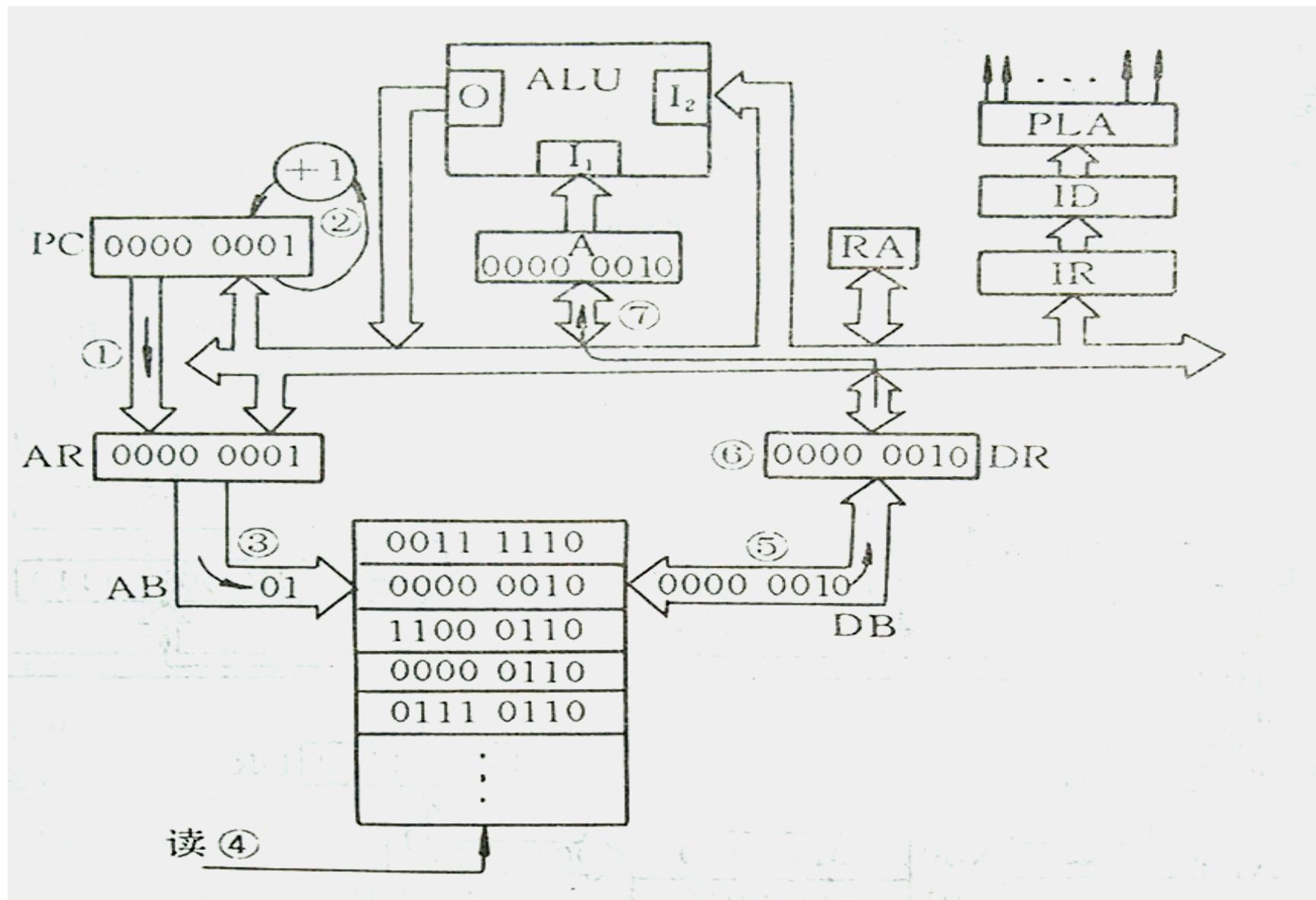
举例：

LD A,2	0011 1110
	0000 0010
ADD A,6	1100 0110
	0000 0110
HALT	0111 0110

# 取第一条指令过程



# 执行第一条指令过程



## § 1.6 单片机的开发过程

假设已设计并制作好硬件，开发过程就是编写软件的工作。在编写软件之前，首先要确定一些常数、地址（在硬件设计阶段已被直接或间接地确定下来，如当某器件的连线设计好后，其地址就被确定，当器件的功能被确定下来后，其控制字被确定）。用文本编辑器编写软件，编写好后，用编译器对源程序文件编译、查错，直到没有语法错误。一般应借助仿真机对软件进行调试，直到程序运行正确为止。

运行正确后，就可以写片（将程序固化在EPROM中）。

# 源程序

```
ORG 0000H
LJMP START
ORG 040H
MOV SP, #5FH ;设堆栈
NOP
LJMP LOOP ; 循环
END ; 结束
```

目标文件即最终写入EPROM的文件：

## § 1.7 仿真、仿真器

仿真是单片机开发过程中非常重要的一个环节，除了一些极简单的任务，一般产品开发过程中都要进行仿真。

仿真分软件模拟仿真和利用仿真器仿真两类。仿真的主要目的是进行软件调试，如果借助仿真器，也能进行一些硬件排错。

一块单片机应用电路板包括单片机部分及为达到使用目的而设计的应用电路，仿真可以利用仿真器来代替目标机的单片机部份，对应用电路部分进行测试、调试。

## 软件模拟仿真

是指用仿真软件来模拟单片机运行情况，一般常用这种方式，但不能进行硬件系统的调试和故障诊断。

## 利用仿真器仿真

利用仿真器以及微机可以进行软硬件系统的调试和故障诊断。

在仿真调试过程中，能够以各种运行方式运行程序（断点、单步、跟踪），还可以观察到单片机内部存储器、寄存器等的状态。

## § 1.8 数制与编码

### 1.8.1 数制的表示

#### 1. 常用数制

##### (1) 十进制数

十进制数有两个主要特点：

- 有十个不同的数字符号：0、1、2、...、9；
- 低位向高位进、借位的规律是“逢十进一”“借一当十”

的计数原则进行计数。十进制数用D结尾表示。

例如：

$$(1234.45)D = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}$$

式中的10称为十进制数的基数,  $10^3$ 、 $10^2$ 、 $10^1$ 、 $10^0$ 、 $10^{-1}$ 称为各数位的权。

## 1.8.1 数制的表示

### (2) 二进制数

在二进制中只有两个不同数码：0和1，进位规律是“逢二进一”“借一当二”的计数原则进行计数。二进制数用B结尾表示。

例如，二进制数11011011.01可表示为：

$$(11011011.01)B = 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

## 1.8.1 数制的表示

### (3) 十六进制数

在十六进制中有0、1、2...、9、A、B、C、D、E、F共十六个不同的数码，采用“逢十六进一”“借一当十六”的计数原则进行计数。十六进制数用H结尾表示。

例如，十六进制数(4E9.27)H可表示为

$$(4E9.27)H = 4 \times 16^2 + 14 \times 16^1 + 9 \times 16^0 + 2 \times 16^{-1} + 7 \times 16^{-2}$$

## 1.8.1 数制的表示

### 各种进位制的对应关系

十进制	二进制	十六进制	十进制	二进制	十六进制
0	0	0	9	1001	9
1	1	1	10	1010	A
2	10	2	11	1011	B
3	11	3	12	1100	C
4	100	4	13	1101	D
5	101	5	14	1110	E
6	110	6	15	1111	F
7	111	7	16	10000	10
8	1000	8	17	10001	11

## 1.8.1 数制的表示

### (1) 二、十六进制数转换成为十进制数

根据各进制的定义表示方式，按权展开相加，即可转换为十进制数。

【例】将(10101)B, (49)H转换为十进制数。

$$(10101)_B = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 37$$

$$(49)_H = 4 \times 16^1 + 9 \times 16^0 = 73$$

### (2) 十进制数转换为二进制数

十进制数转换二进制数，需要将整数部分和小数部分分开，采用不同方法进行转换，然后用小数点将这两部分连接起来。

## 1.8.1 数制的表示

### ①整数部分：除2取余法

具体方法是：将要转换的十进制数除以2，取余数；再用商除以2，再取余数，直到商等于0为止，将每次得到的余数按**倒序**的方法排列起来作为结果。

【例】将十进制数25转换成二进制数

所以  $(25)D=11001B$

## 1.8.1 数制的表示

### ②小数部分：乘2取整法

具体方法是：将十进制小数不断地乘以2，直到积的小数部分为零（或直到所要求的位数）为止，每次乘得的整数依次**顺序**排列即为相应进制的数码。最初得到的为最高有效数位，最后得到的为最低有效数位。

【例】将十进制数0.625转换成二进制数。



所以  $(0.625)D=0.101B$

## 1.8.1 数制的表示

【例】将十进制数25.625转换成二进制数，  
只要将上例整数和小数部分组合在一起即可

$$(25.625)D = (11001.101)B$$

## 1.8.1 数制的表示

### (4) 二进制与十六进制之间的相互转换

由于 $2^4=16$ ，故可采用“**合四为一**”的原则，即从小数点开始向左、右两边各以4位为一组进行二-十六转换，若不足4位的以0补足，便可以将二进制数转换为十六进制数。反之，每位十六进制数用四位二进制数表示，就可将十六进制数转换为二进制数。

## 1.8.1 数制的表示

【例】将(111111000111.100101011)B转换为十六进制数。

0001 1111 1100 0111 . 1001 0101 1000  
1 F C 7 . 9 5 8

即 (111111000111.100101011)B = (1FC7.958)H

【例】将(79BD.6C)H转换为二进制数。

7 9 B D . 6 C  
0111 1001 1011 1101 . 0110 1100

即 (79BD.6C)H = (111100110111101.011011)B

在计算机中，数字、字母、符号都是用二进制编码进行表示。

编码：按照一定的规则组合而成的若干位二进制代码。

## 1.8.2 常用的信息编码

### 二-十进制BCD码（Binary-Coded Decimal）

二-十进制BCD码是指每位十进制数用4位二进制数编码表示。由于4位二进制数可以表示16种状态，可丢弃最后6种状态，而选用0000~1001来表示0~9十个数符。这种编码又叫做8421码。既有二进制数的形式和特点，又有十进制数“逢十进一”的特点。

## 1.8.2 常用的信息编码

### 十进制数与BCD码的对应关系

十进制数	BCD码	十进制数	BCD码
0	0000	10	00010000
1	0001	11	00010001
2	0010	12	00010010
3	0011	13	00010011
4	0100	14	00010100
5	0101	15	00010101
6	0110	16	00010110
7	0111	17	00010111
8	1000	18	00011000
9	1001	19	00011001

## 1.8.2 常用的信息编码

【例】将69.25转换成BCD码。

6	9	.	2	5
0110	1001	.	0010	0101

结果为69.25=(01101001.00100101)BCD

【例】将BCD码100101111000.01010110转换成十进制数。

1001	0111	1000	.	0101	0110
9	7	8	.	5	6

结果为 (100101111000.01010110) BCD=978.56

## 2. 字符编码（ASCII码）

计算机使用最多、最普遍的是ASCII（American Standard Code For Information Interchange）字符编码，即美国信息交换标准代码。

## 1.8.2 常用的信息编码

七位ASCII代码表

$d_3\ d_2\ d_1\ d_0$ 位	0 $d_6\ d_5\ d_4$ 位							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	0	@	P	'	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	:	K	[	k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	↑	n	~
1111	SI	HS	/	?	O	←	o	DEL

## 1.8.2 常用的信息编码

ASCII码的每个字符用7位二进制数表示，其排列次序为  $d_6d_5d_4d_3d_2d_1d_0$ ， $d_6$ 为高位， $d_0$ 为低位。而一个字符在计算机内实际是用8位表示。最高位 $d_7$ 一般作为奇偶校验位（正常情况下，最高一位 $d_7$ 为“0”）。7位二进制数共有128种编码组合，可表示128个字符，其中有数字10个、大小写英文字母52个、其它字符32个和控制字符34个。

数字0~9的ASCII码为30H~39H。

大写英文字母A~Z的ASCII码为41H~5AH。

小写英文字母a~z的ASCII码为61H~7AH。

对于ASCII码表中的**0、A、a**的ASCII码**30H、41H、61H**应尽量记住，其余的数字和字母的ASCII码可按数字和字母的顺序以十六进制的规律写出。

## 1.8.2 常用的信息编码

### 奇偶校验码

奇偶校验码是在传送的代码上附加一个校验位，作为代码的比较校验。在接收方，先对信息代码按双方的校验规定求取奇偶校验码，然后再与收到的附加校验位作比较，若相等则认为接收的代码是正确的，否则为错。

奇偶校验就是鉴别代码中有奇数个“1”，还是有偶数个“1”。例如，有效信息为1011001，奇偶校验位在第一个数位，若采用偶校验码记为“0”的话，则有效代码为**0** 1011001；采用奇校验码记为“0”的话，有效代码则为**1** 1011001。当接收方收到这组代码后，便根据奇、偶校验的约定和有效代码中“1”的个数形成校验码，然后再与接收的校验位作比较。比较相等的话，说明接收的信息正确；反之，则认为出现了错误。

# 1.9 计算机中数的表示与运算

## 1.9.1 二进制数在计算机内的表示

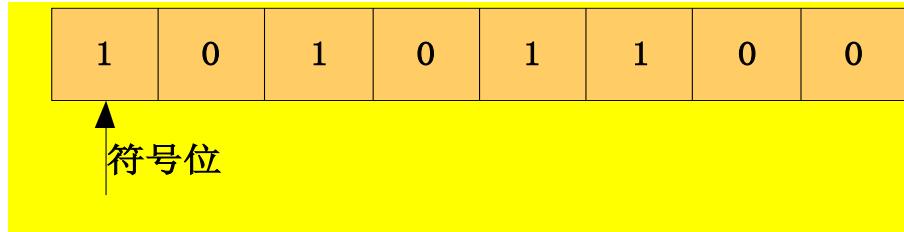
### 1. 机器数

在计算机中，只有“0”和“1”两种形式，因此数的正、负号也必须以“0”和“1”表示。通常把一个数的最高位定义为符号位，用0表示正，1表示负，称为数符，其余位仍表示数值。

在机器内存放的正、负号数码化的数称为机器数，机器外部由正、负号表示的数称为真值数。

## 1.9.1 二进制数在计算机内的表示

【例】真值为(-0101100)B的机器数为10101100，存放在计算机中，如图所示。



真值B在计算机中的存放

要注意的是，机器数表示的范围受到字长和数据的类型的限制。字长和数据类型确定，机器数能表示的数值范围也就确定。

例如，若表示一个整数，字长为8位，则最大的正数为01111111，最高位为符号位，即最大值为127。若数值超出127，就要“溢出”。最小负数为11111111，最高位为符号位，即最小值为-127。

## 1.9.1 二进制数在计算机内的表示

### 2. 数的定点和浮点表示

计算机内表示的数，主要分成定点小数、定点整数与浮点数三种类型。

#### （1）定点小数的表示法

定点小数是指小数点准确固定在数据某一个位置上的小数。一般把**小数点固定在最高数据位的左边**，小数点前边再设一位符号位。按此规则，任何一个数都可以写成：

$$N = N_s N_{-1} N_{-2} \cdots N_{-M}, \quad N_s \text{ 为符号位}$$

#### （2）整数的表示法

整数所表示的数据的最小单位为1，可以认为它是小数点定在数值最低位右面的一种表示法。整数分为带符号和不带符号两类。对带符号的整数，符号位放在最高位。可以写成：

$$N = N_s N_{n-1} \cdots N_2 N_1 N_0, \quad N_s \text{ 为符号位}$$

## 1.9.1 二进制数在计算机内的表示

一般定点数表示的范围和精度都较小，在数值计算时，大多采用浮点数。

### (3) 浮点数的表示方法

浮点表示法对应于科学(指数)计数法，浮点表示法中小数点的位置是浮动可变的，如数110.011可表示为：

$$N = 110.011 = 1.10011 \times 10^{+2} = 11001.1 \times 10^{-2} = 0.110011 \times 10^{+3}$$

一般任何一个二进制数N可以表示为  **$N = \pm 2^E \times M$**

其中，E为数N的阶码，取值为二进制整数，常用补码形式；

M为数N的尾数，一般均为纯小数；

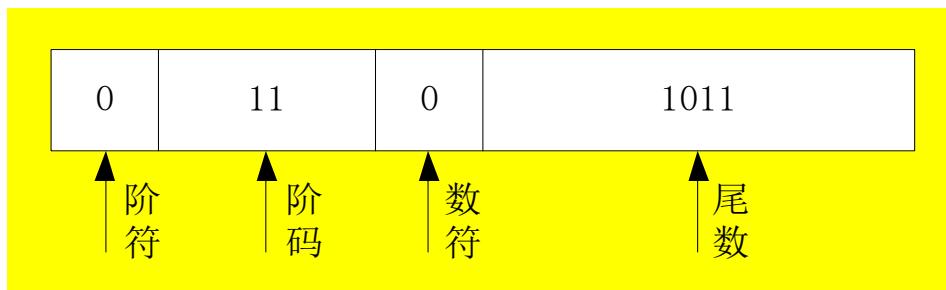
2为底，在微型机中它是约定好的，不用表示出来。

## 1.9.1 二进制数在计算机内的表示

在计算机中一个浮点数由两部分构成：阶码和尾数。阶码是指数E，尾数是纯小数M。其存储格式为：

阶符	阶码	数符	尾数
----	----	----	----

【例】设尾数为4位，阶码为2位，则二进制数  
 $N=2^{11} \times 1011$ 的浮点数表示形式为：



注意：浮点数的正、负是由尾数的数符确定，而阶码的正、负只决定小数点的位置，即决定浮点数的绝对值大小。

## 1.9.1 二进制数在计算机内的表示

### (4) 带符号数的表示

在计算机中，带符号数可以用不同方法表示，常用的有：  
**原码、反码和补码。**

#### ——原码

【例】当机器字长n=8时：

$[+1]_{\text{原}} = 0\ 0000001$ ,  $[-1]_{\text{原}} = 1\ 0000001$

$[+127]_{\text{原}} = 0\ 1111111$ ,  $[-127]_{\text{原}} = 1\ 1111111$

可以看出，在原码表示法中：

最高位为符号位，正数为0，负数为1，其余n-1位表示数的绝对值。

在原码表示中，零有两种表示形式，即：

$[+0] = 00000000$

$[-0] = 10000000$

## 1.9.1 二进制数在计算机内的表示

### ——反码

【例】当机器字长n=8时：

$[+1]_{\text{反}} = 00000001$ ,  $[-1]_{\text{反}} = 11111110$

$[+127]_{\text{反}} = 01111111$ ,  $[-127]_{\text{反}} = 10000000$

可以看出，在反码表示中：

正数的反码与原码相同，负数的反码只需将其对应的正数按位求反即可得到。

机器数最高位为符号位，0代表正号，1代表负号。

反码表示方式中，零有两种表示方法：

$[+0]_{\text{反}} = 00000000$

$[-0]_{\text{反}} = 11111111$

## 1.9.1 二进制数在计算机内的表示

### ——补码

【例】当机器字长n=8时，

$[+1]_{\text{补}} = 00000001$ ,  $[-1]_{\text{补}} = 11111111$

$[+127]_{\text{补}} = 01111111$ ,  $[-127]_{\text{补}} = 10000001$

可以看出，在补码表示中：

正数的补码与原码、反码相同，负数的补码等于它的反码加1。

机器数的最高位是符号位，0代表正号，1代表负号。

在补码表示中，0有唯一的编码：

$[+0]_{\text{补}} = [-0]_{\text{补}} = 00000000$

补码的运算方便，二进制的减法可用补码的加法实现，使用较广泛。

## 1.9.1 二进制数在计算机内的表示

【例】假定计算机字长为8位，试写出122的原码、反码和补码。

$$[122]_{\text{原}} = [122]_{\text{反}} = [122]_{\text{补}} = (0111\ 1010)_B$$

【例】假定计算机字长为8位，试写出-45的原码、反码和补码。

$$[-45]_{\text{原}} = (1010\ 1101)_B$$

$$[-45]_{\text{反}} = (1101\ 0010)_B$$

$$[-45]_{\text{补}} = (1101\ 0011)_B$$

对于用补码表示的负数，首先认定它是负数，而后用求它的补码的方法可得到它的绝对值，即可求得该负数的值。

例如，补码数  $(1111\ 0011)_B$  是一个负数，

反码为  $(1111\ 0010)_B$

原码为  $(1000\ 1101)_B$

故求出  $(1111\ 0011)_B$  为  $(-13)_D$ 。

## 1.9.2 补码的运算

在微处理机中，使用补码进行运算是十分方便的，它使同一个微处理机中既能运算带符号数又能运算不带符号的数。而且，在采用补码表示带符号数的情况下，两个数的减法可以用加法来实现。

在进行带符号数的加减运算时，应把参与运算的数据转换成补码形式进行运算。当使用8位二进制数表示带符号的数时，它所能表示的数值范围在(-128)D～(+127)D之间，如果相加结果超出了这个范围，就会导致错误发生。

$$[X+Y]_{\text{补}} = [X]_{\text{补}} + [Y]_{\text{补}}$$

$$[X-Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}}$$

## 1.9.2 补码的运算

【例】两个带符号的数  $(01000001)B$  与  $(01000011)B$  的相加情况。

$$\begin{array}{r} 01000001 \\ + 01000011 \\ \hline 10000100 \end{array}$$

两个正数相加，结果却是一个负数——符号位为1。显然这个结果是错误的，原因在于这两个数相加的结果超过了8位二进制带符号数所能表示的数值范围。

【例】两个负数  $(10001000)B$  与  $(11101110)B$  的相加情况。

$$\begin{array}{r} 10001000 \\ + 11101110 \\ \hline 1 01110110 \end{array}$$

数值部分相加没有进位，但符号位相加有进位，仍然有溢出。

# 本章小结

微型计算机的发展和特点

微处理器、微型计算机和微型计算机系统

微型计算机的工作过程 ★

计算机中的数制与编码 ★

# 第二章 MCS-51单片机的内部结构

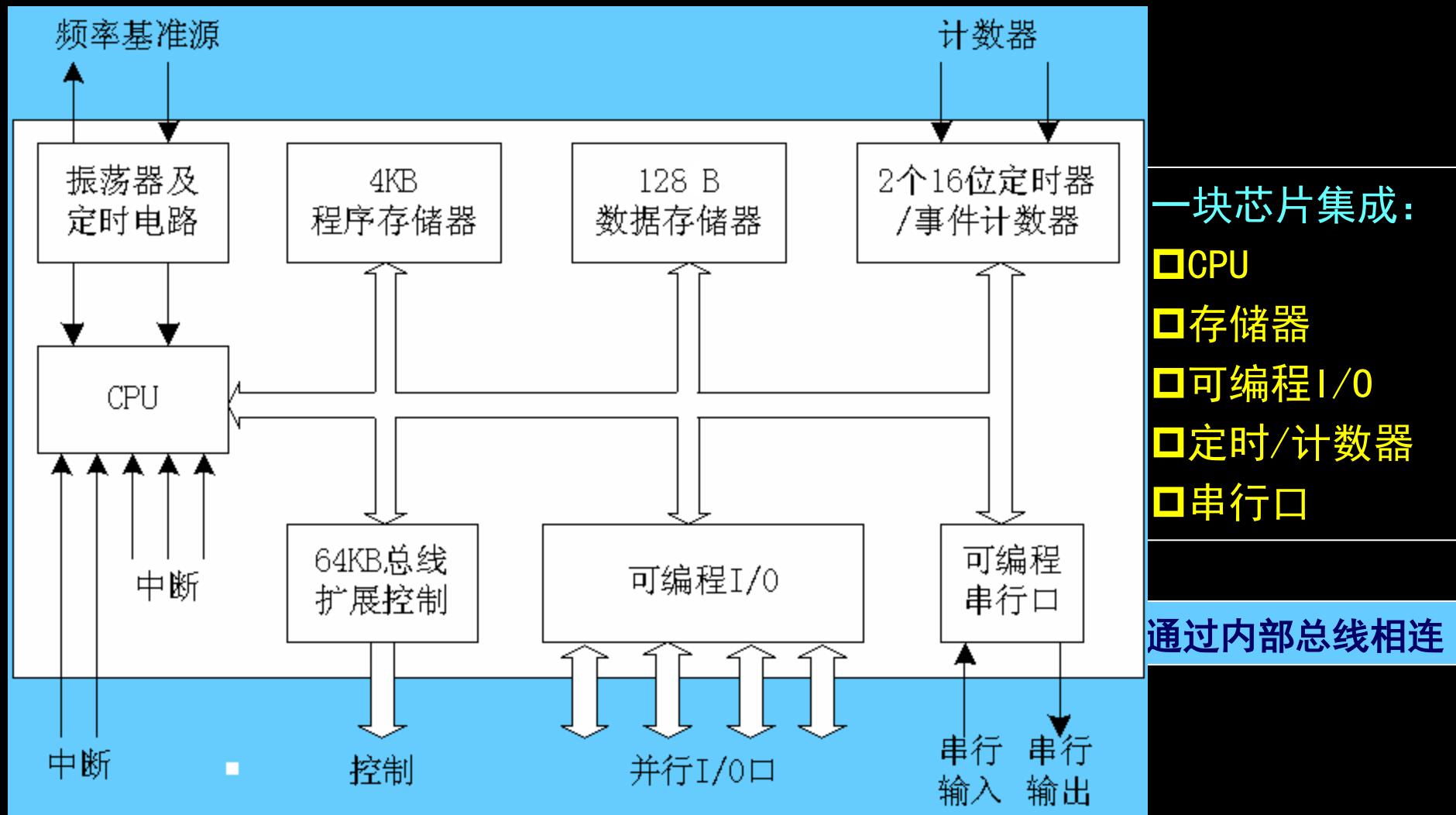
1. 总体和内部结构
2. 微处理器**CPU**
3. 引脚和片外总线
4. **CPU**时序和辅助电路
5. 存储器结构

MCS-51的典型产品是**8051、8031、8751**。  
8051是ROM型单片机，内部有4K ROM；8031无  
片内ROM；8751片内有4K EPROM；除此以外，  
它们的内部结构及引脚完全相同。

以**MCS-51**系列的**8051**为典型例子，详细介绍  
单片机的内部结构、性能、存储器结构及工作原理  
等内容。

## § 2.1 MCS-51单片机的结构

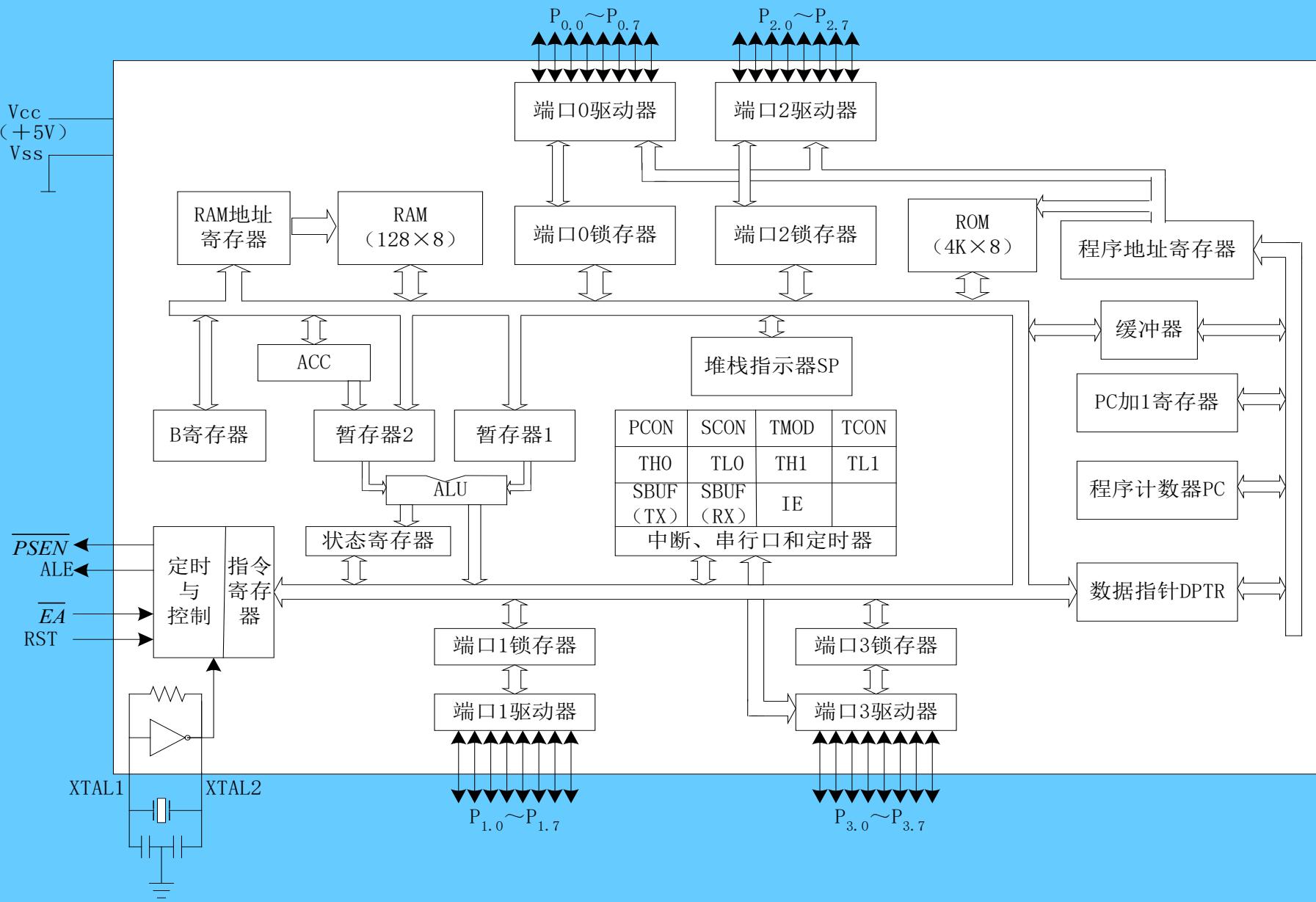
以8051为例给出的单片机总体结构框图如下：



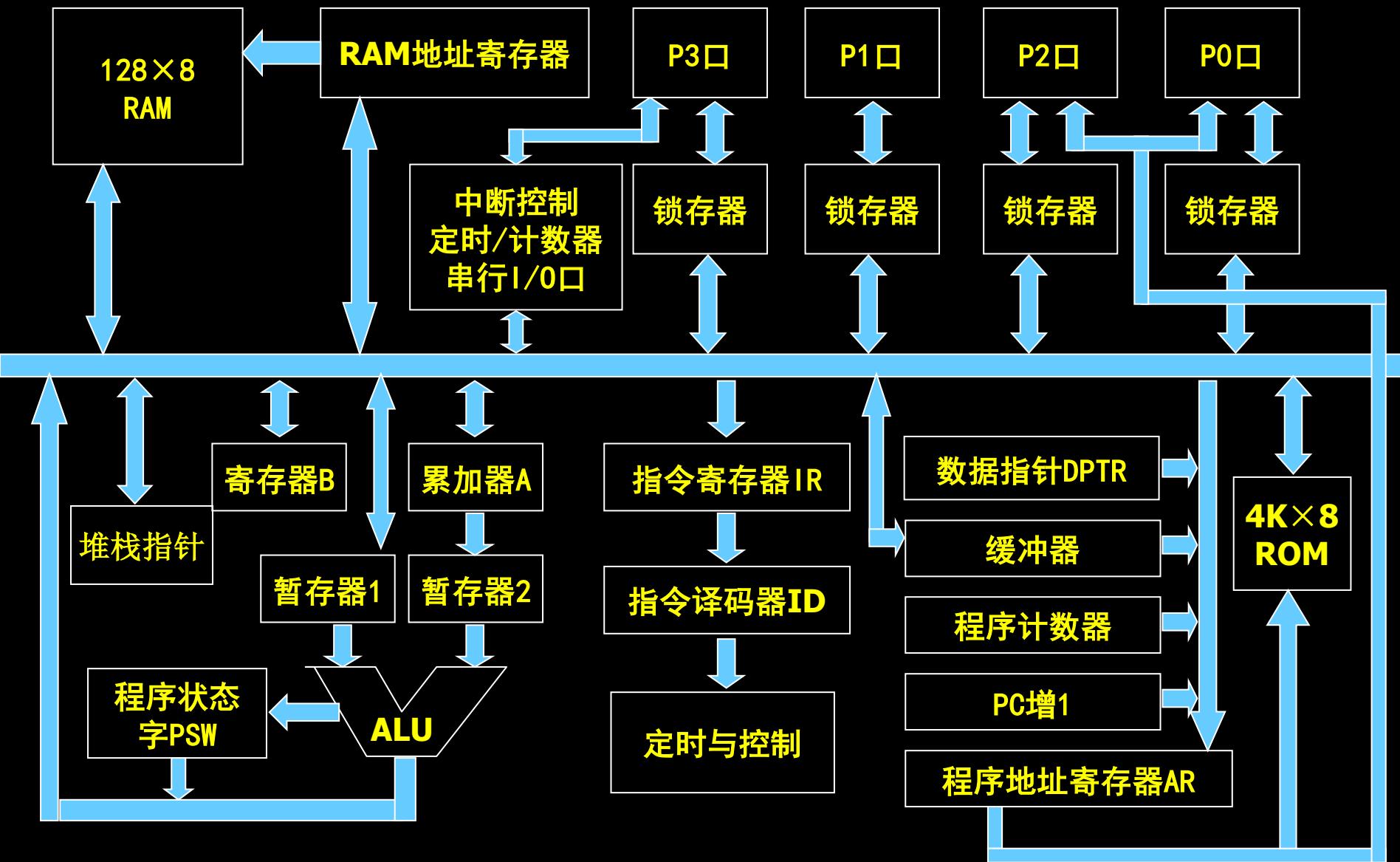
# 8051单片机基本结构：

- ◆ 面向控制的一个8位 CPU和指令系统；
- ◆ 一个片内时钟振荡器和时钟电路；
- ◆ 4Kbyte 程序存储器（ROM/PROM）；
- ◆ 128byte 数据存储器（RAM）；
- ◆ 两个16位 定时/计数器；
- ◆ 64Kbyte 总线扩展控制电路；
- ◆ 四个8-bit 并行I/O端口；
- ◆ 一个可编程串行接口；
- ◆ 五个中断源，其中包括两个优先级嵌套中断

# MCS-51单片机芯片内部结构展开图



# 8051的内部结构展开图



## § 2.2 MCS-51单片机的CPU

8位微处理器，是单片机的核心部分

- 运算器
  - 算术逻辑部件 (**ALU, Arithmetic Logic Unit**)
  - 寄存器：累加器ACC、B寄存器、暂存器TMP1、TMP2、程序状态字寄存器PSW、BCD码运算调整电路等
- 控制器
  - 程序计数器**PC**，堆栈指针**SP**，数据指针**DPTR**，指令寄存器**IR**
  - 指令译码器
  - 时钟发生器
  - 定时控制逻辑

## § 2.2.1 运算器

### ■ 算术逻辑部件**ALU**

- 组成：加法器+其它逻辑电路
- 算术运算：8位数据的+ - × ÷
- 逻辑运算：与、或、取反、异或、循环移位、位操作等

### ■ 与ALU相关的特殊功能寄存器**SFR**

(Special Functional Register)

- ACC
- B
- PSW

# 什么是特殊功能寄存器(SFR)？

## ▼ 特殊功能寄存器**SFR** (专用寄存器)

专用于控制、选择、管理、存放单片机内部各部分的工作方式、条件、状态、结果的寄存器。

▼ 不同的**SFR**管理不同的硬件模块，负责不同的功能——各司其职

换言之：要让单片机实现预订的功能，必须有相应的硬件和软件，而软件中最重要的一项工作就是对**SFR**写入指令。

## 与ALU相关的 (3个)

### ■ A Register (Accumulator) :

累加器，通常用 A 或 ACC 表示。可字节寻址 (E0H) ,  
也可位寻址 (E0H~E7H)

它是一个寄存器，而不是一个做加法的部件。

在运算器做运算时其中一个数一定是在累加器 A 中，  
运算结果又存于累加器 A 中。

### ■ B Register :

暂存寄存器。在做乘、除法时放乘数或除数及相关结果。

运算前：乘数或除数

运算后：乘积的高位字节或商的余数部分

### ■ PSW (Program Status Word) :

程序状态字寄存器

## PSW (Program Status Word) :

PSW是8位寄存器，用于存放程序运行状态的标志

按照**功能**来分，PSW的标志有两类。一类是状态标志，当CPU进行各种逻辑操作或算术运算时，为反映操作或运算结果的状态，把相应的标志位置1或清0。它为计算机确定程序的下一步操作提供依据。

另一类是**用户设定的标志位**，用来选择CPU当前使用的工作寄存器组，或者用户在程序设计中作为某种特定的标志位。

它的各位功能如下：

PSW位地址	D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H
	CY	AC	F0	RS1	RS0	OV	(F1)	P

PSW位地址

D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H
<b>CY</b>	<b>AC</b>	F0	RS1	RS0	OV		P

- **CY**: 进位标志, 有时表示为**C**。

加减运算时, 最高位进位或借位时置1, 否则为0;  
进行位操作时, 作为位累加器C, 布尔累加器;  
循环移位和比较转移指令时也影响该位。

- **AC**: 半进位标志

可用于校正BCD码加减运算结果 (DA A)

例: 78H+97H

$$\begin{array}{r}
 0111 1000 \\
 +1001 0111 \\
 \hline
 1 0000 1111
 \end{array}$$

有进位  
**CY=1**

没有半进位  
**AC=0**

PSW位地址

D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H
CY	AC	F0	RS1	RS0	OV		P

■ **RS1,RS0:** 工作寄存器组选择位

- |     |           |
|-----|-----------|
| 0 0 | 选择工作寄存器0组 |
| 0 1 | 选择工作寄存器1组 |
| 1 0 | 选择工作寄存器2组 |
| 1 1 | 选择工作寄存器3组 |

所选的4组工作寄存器

0组 (内部**RAM**地址 **00H~07H**)

1组 (内部**RAM**地址 **08H~0FH**)

2组 (内部**RAM**地址 **10H~17H**)

3组 (内部**RAM**地址 **18H~1FH**)

- **P:** 奇偶校验位, 它用来表示累加器**A**内容中二进制数位“1”的个数的奇偶性。若为奇数, 则**P=1**, 否则为0。
- 例: 某运算结果是**78H** (01111000), **P=0**。

PSW位地址

D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H
CY	AC	F0	RS1	RS0	OV		P

■F0：用户标志位。作为软件标志，由编程人员决定何时使用。

■OV：溢出标志位。

有符号数运算时，如果发生溢出，OV置“1”，否则置“0”

两个同号数相加或两个异号数相减时，可能会产生溢出。

## 举例：

$$\begin{aligned}25(00011001) + 125(01111101) &= 10010110 \\-121(10000111) - 120(10001000) &= 00001111\end{aligned}$$

$$OV = D_{7C} \oplus D_{6C}$$

- 一般用符号位相加的进位  $D_{7C}$  和数值部分的最高位相加的进位  $D_{6C}$  的状态来判断是否有溢出。
- $\oplus$  是异或运算，即两个数相同为0，不同为1。

## § 2.2.2 控制器

- 控制器是单片机的控制部件，其主要任务是识别指令，并根据指令的性质控制单片机各功能部件，从而保证单片机各部分能自动而协调地工作。
- 单片机执行指令是在控制器的控制下进行的：  
首先从程序存储器中读出指令，送指令寄存器保存；然后送指令译码器进行译码，译码结果送定时控制逻辑电路，定时控制逻辑产生各种定时信号和控制信号，再送到单片机的各个部件去进行相应的操作。
- 控制器主要包括程序计数器、堆栈指针、数据指针、指令寄存器IR、指令译码器ID、条件转移逻辑电路及时序控制逻辑电路。

# 1. 程序计数器PC

- **程序计数器PC (Program Counter)**

**PC**是16位的专用寄存器，内容为下一条要执行的指令的地址

- 特点：

- ▼ 按机器周期自动增1计数器
- ▼ 总指向下一条指令所在首地址(当前PC值)
- ▼ 一切分支/跳转/调用/中断/复位 等操作的本质就是改变 PC 值

## 2、堆栈指针SP：管理堆栈区

### ● 堆栈：

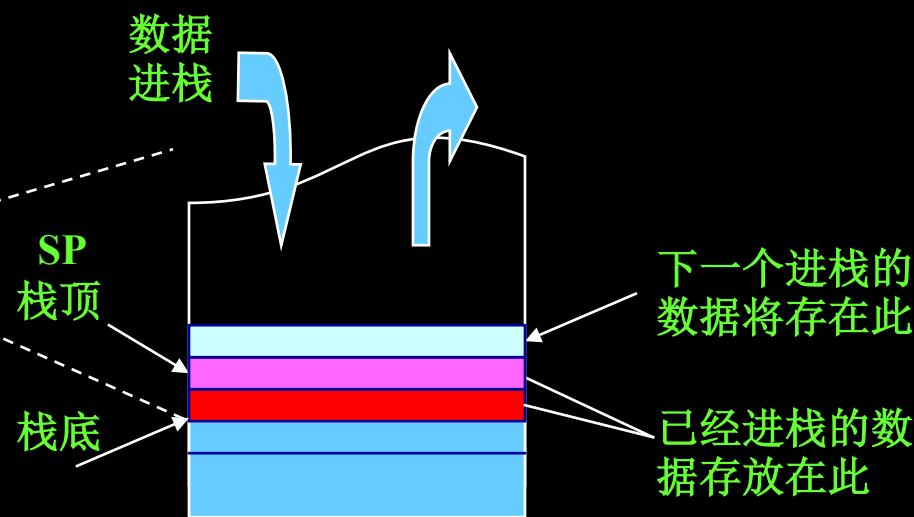
在片内RAM中，常常要指定一个专门的区域来存放某些特别的数据，它遵循先进后出或后进先出(LIFO/FILO) 的原则，这个RAM区叫堆栈。

### ● 功用：

- 1) 子程序调用和中断服务时CPU自动将当前PC值压栈保存，返回时自动将PC值退栈。
- 2) 中断时保护现场/恢复现场
- 3) 数据传输

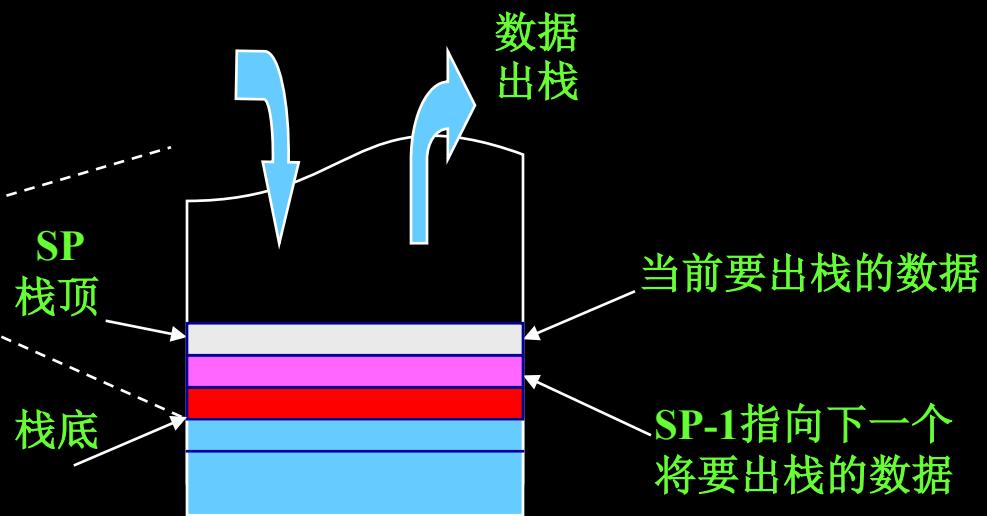
堆栈区可以安排在RAM区任意位置，一般不安排在工作寄存器区和可按位寻址的RAM区，通常放在RAM区靠后的位置。

复位后  $SP=07H$ 。数据进栈时：首先  $SP+1$ ，然后将数据压入  $SP$  所指单元，故使用默认  $SP$  值时，第一个放进堆栈的数据将放进  $08H$  单元，……



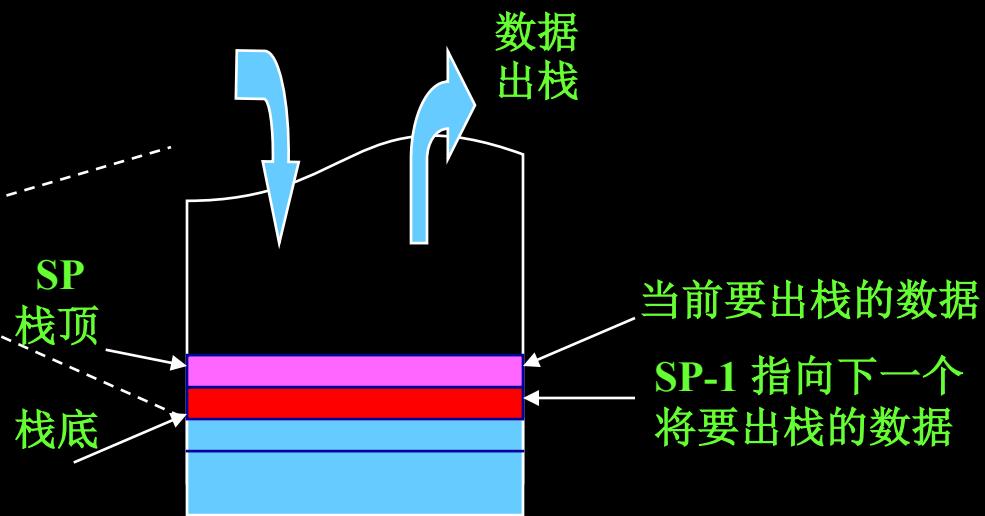


从堆栈取出数据时：取出的数据是最近放进去的一个数据(SP所指)，也就是当前栈顶的数据。然后SP再自动减1，仍指着栈顶.....





从堆栈取出数据时：取出的数据是最近放进去的一个数据，也就是当前栈顶的数据。然后SP再自动减1，仍指着栈顶.....



### 3、数据指针DPTR

- **16位地址寄存器**，用来存放地址指针
- 可指向**64K**范围的任一单元，既可以用  
来访问外部数据存储器中的任一单元，  
也可以作为通用寄存器来用。
- 由两个独立的**8位寄存器**组成
  - 高字节**DPH**
  - 低字节**DPL**

所有  
SFR  
寄  
存  
器  
(20个)

标识符号	字节地址	寄存器名称
ACC	0E0H	累加器
B	0F0H	B寄存器
PSW	0D0H	程序状态字
SP	81H	堆栈指针
DPTR	82H、83H	数据指针 (16位) 含DPL和DPH
IE	0A8H	中断允许控制寄存器
IP	0B8H	中断优先控制寄存器
P0	80H	I/O口0寄存器
P1	90H	I/O口1寄存器
P2	0A0H	I/O口2寄存器
P3	0B0H	I/O口3寄存器
PCON	87H	电源控制及波特率选择寄存器
SCON	98H	串行口控制寄存器
SBUF	99H	串行数据缓冲寄存器
TCON	88H	定时控制寄存器
TMOD	89H	定时器方式选择寄存器
TL0	8AH	定时器0低8位
TH0	8CH	定时器0高8位
TL1	8BH	定时器1低8位
TH1	8DH	定时器1高8位

## 与端口相关的 (7个)

- P0、P1、P2、P3：  
四个并行 I/O (输入/输出) 口的寄存器。其内容对应着管脚的输出。
- SCON (Serial Control Register)  
串行口控制寄存器
- SBUF (Serial Date Buffer)  
串行口数据缓冲器
- PCON (Power Control Register)  
电源控制寄存器

## 与定时/计数器相关的 (6个)

- TMOD (Timer/Counter Mode Register)  
定时器/计数器工作模式寄存器
- TCON (Timer/Counter Control Register)  
定时器/计数器控制寄存器
- TH0、TL0 、 TH1、 TL1：分别是T0、 T1的计数初值寄存器

## 与中断相关的 (2个)

- IP (Interrupt Priority Register)  
中断优先级控制寄存器
- IE (Interrupt Enable Register)  
中断允许控制寄存器

# PC与SFR复位状态表

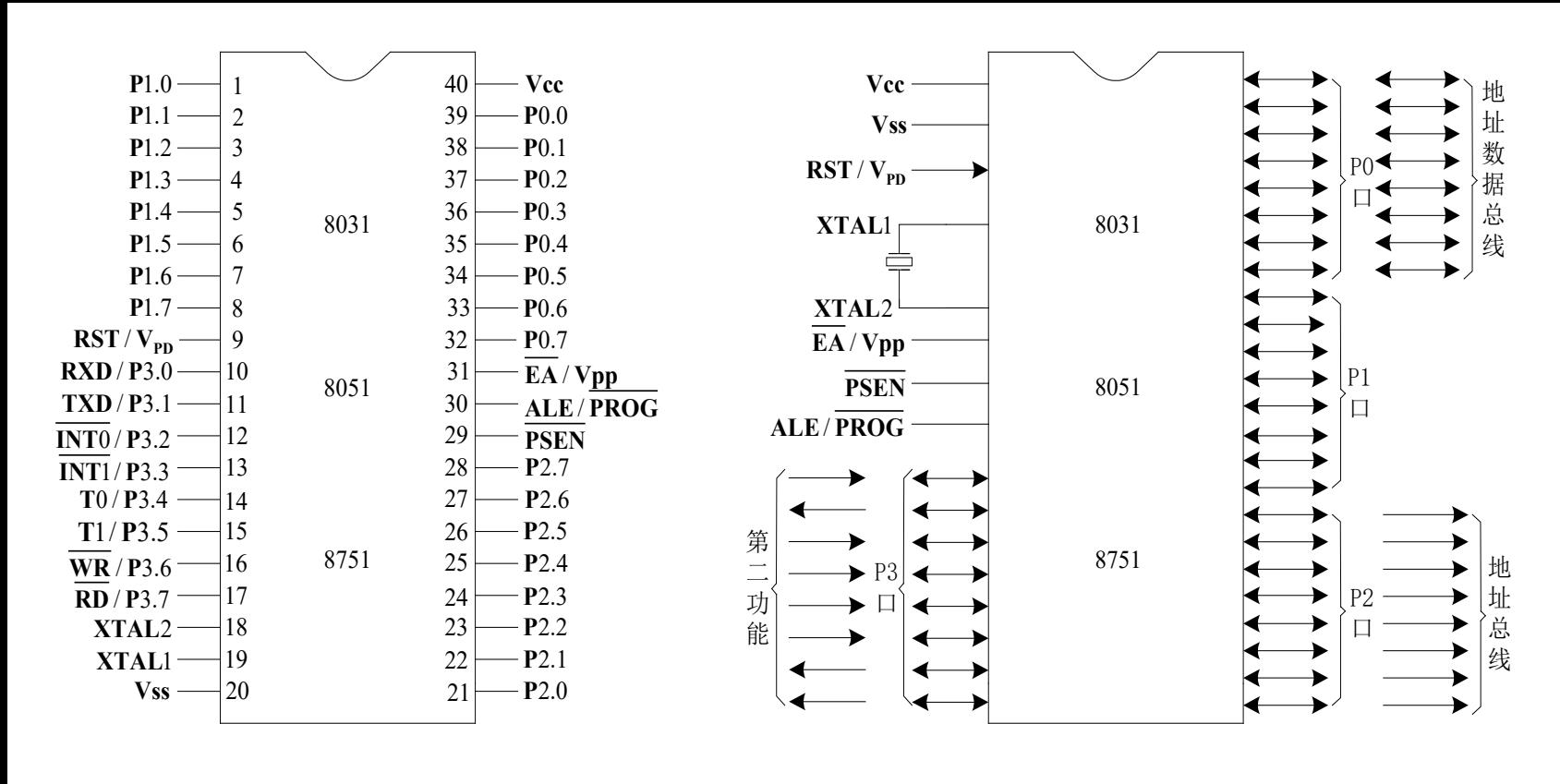
寄存器	复位状态	寄存器	复位状态
PC	0000H	TCON	00H
A	00H	TH0	00H
B	00H	TL0	00H
PSW	00H	TH1	00H
SP	07H	TL1	00H
DPTR	0000H	SCON	00H
P0~P3	FFH	SBUF	XXH
IP	XX000000B	PCON	0XXX0000B
IE	0X000000B		
TMOD	00H		

## § 2.3 MCS-51的引脚和片外总线结构

- MCS-51的引脚功能
- MCS-51外部总线结构

## § 2.3.1 引脚信号

MCS-51系列单片机为标准40脚DIP封装。



MCS-51系列单片机引脚图及逻辑符号

# 认识单片机的引脚

P1.0	1	40	VCC
P1.1	2	39	P0.0 (AD0)
P1.2	3	38	P0.1 (AD1)
P1.3	4	37	P0.2 (AD2)
P1.4	5	36	P0.3 (AD3)
P1.5	6	35	P0.4 (AD4)
P1.6	7	34	P0.5 (AD5)
P1.7	8	33	P0.6 (AD6)
RST	9	32	P0.7 (AD7)
(RXD) P3.0	10	31	EA/VPP
(TXD) P3.1	11	30	ALE/PROG
(INT0) P3.2	12	29	PSEN
(INT1) P3.3	13	28	P2.7 (A15)
(T0) P3.4	14	27	P2.6 (A14)
(T1) P3.5	15	26	P2.5 (A13)
(WR) P3.6	16	25	P2.4 (A12)
(RD) P3.7	17	24	P2.3 (A11)
XTAL2	18	23	P2.2 (A10)
XTAL1	19	22	P2.1 (A9)
GND	20	21	P2.0 (A8)

# MCS-51单片机40脚

Vcc, VSS (GND)	2
XTAL1, XTAL2	2
RESET	1
EA/Vpp	1
ALE/PROG	1
PSEN	1
P0.0—P0.7	8
P1.0—P1.7	8
P2.0—P2.7	8
P3.0—P3.7	8

# 单片机的引脚（电源端）

- **Vcc, VSS (GND) :**

主电源端 (+5V/3.3V/2.7V) 与接地端，不同的单片机可以允许不同的工作电压，不同的单片机表现出的功耗也不同。

# 单片机的引脚（晶振端）

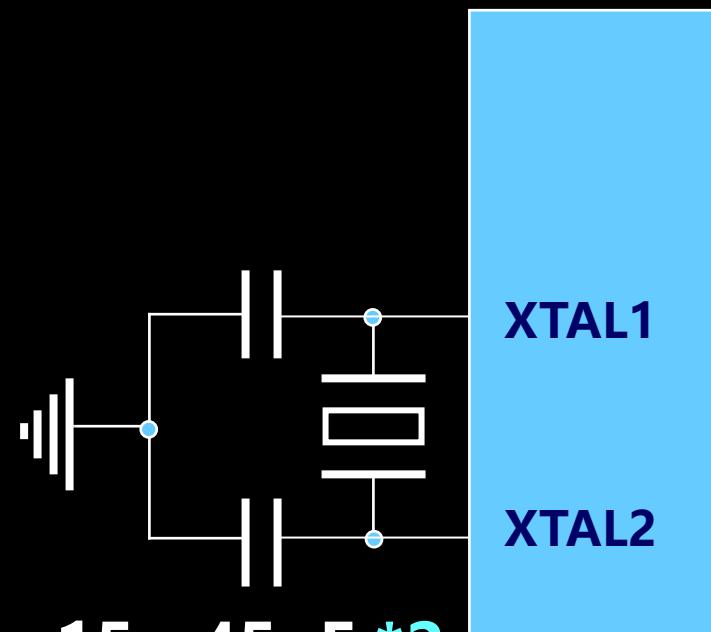
- **XTAL1, XTAL2**：片内振荡电路输入/输出端

若使用内部时钟，  
通常外接一个晶振  
和两个电容。

晶振频率：

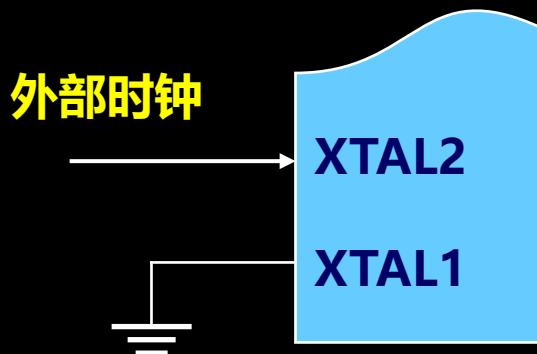
1.2 ~ 12MHz (MCS-51)

0 ~ 24MHz (Atmel-89C)

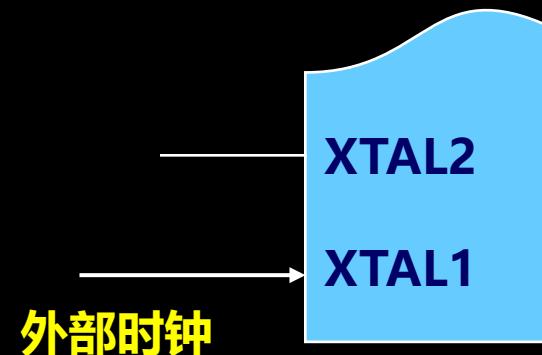


**MCS-51**系列的单片机一般采用**HMOS**和**CHMOS**两种工艺制造，**CHMOS**工艺具有**HMOS**的高速度和**CMOS**低功耗的特点。

若接外部时钟，对于**HMOS**单片机是由**XTAL2**端接入外部时钟，而将**XTAL1**接地。对于**CHMOS**单片机则是由**XTAL1**端接入外部时钟，而将**XTAL2**悬空。



**HMOS单片机**



**CHMOS单片机**

# 单片机的引脚**RST/V<sub>PD</sub>**

(复位端/备用电源输入端)

**RST:**

该信号高电平有效，在输入端保持两个机器周期的高电平后，就可以完成复位操作

**V<sub>pd</sub>:**

可外接+5V备用电源，当主电源V<sub>cc</sub>掉电时该引脚向片内RAM供电

# 单片机的引脚**ALE/PROG**

地址锁存使能输出 / 编程脉冲输入端

当CPU访问外部存储器时，ALE的输出作为外部锁存地址的低位字节的控制信号；当不访问外部存储器时，以1/6的时钟振荡频率固定地输出正脉冲，可以用来定时或者当作对外输出的时钟。

在对8751片内EPROM编程（固化）时，此引脚输入编程脉冲（50ms负脉冲）。

# 单片机的引脚 ( $\overline{EA}$ / $V_{PP}$ 端)

## 外部访问允许/编程电源输入端

当 $\overline{EA}$ 为高电平时，CPU执行片内程序存储器指令，但当PC中的值超过 $0FFFH$ 时，将自动转向执行片外程序存储器指令。当 $\overline{EA}$ 为低电平时，CPU只执行片外程序存储器指令。

该引脚低电平有效，片内无ROM时必须接地  
片内有ROM时应当接高电平

对片内ROM编程时，编程正电源加到此端，  
8751编程时接 $+21V$ 的编程电压 $V_{PP}$ 。

# 单片机的引脚 (PSEN端)

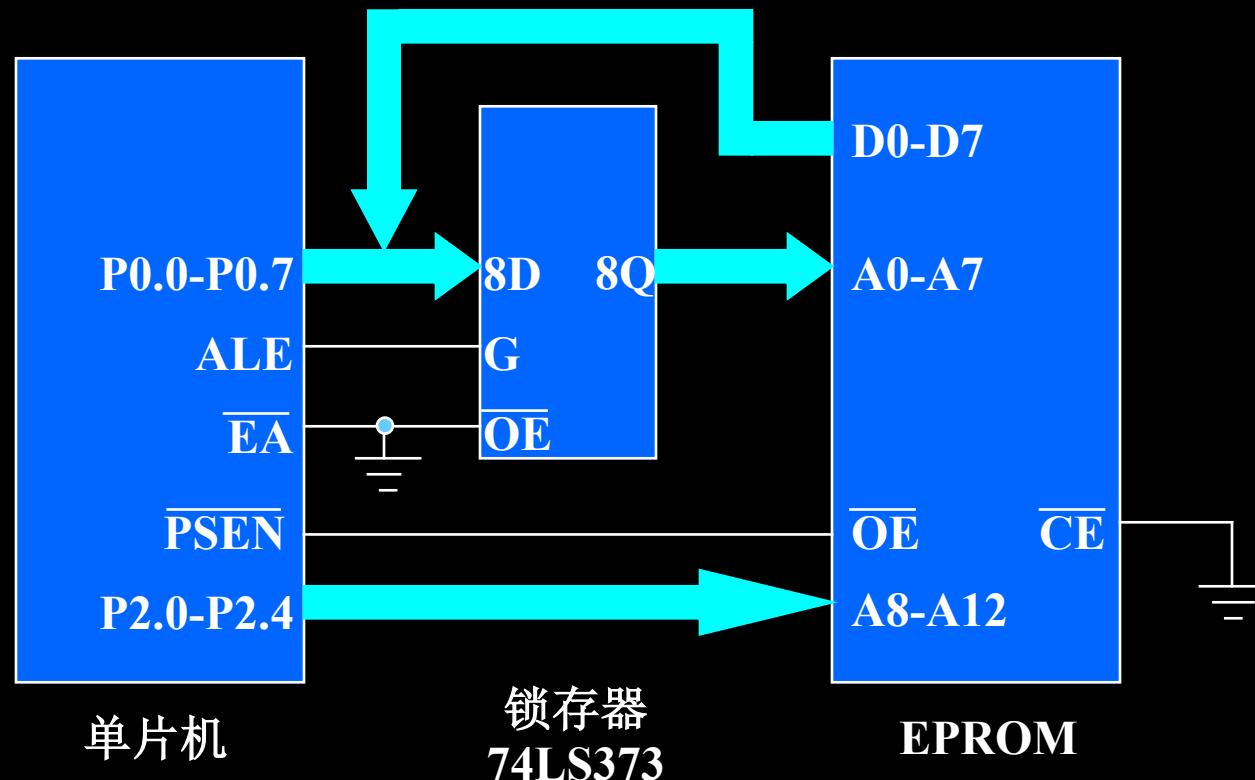
## ■ PSEN: 外部程序存储器读选通信号

CPU访问外部程序存储器时，每个机器周期中，  
PSEN信号两次有效。在执行片内程序存储器取指令和  
访问外部数据存储器时，这两次有效的PSEN信号不出  
现。

用于区别外部程序存储器和数据存储器的访问。

# 单片机的引脚 (PSEN端)

- PSEN: 寻址外部程序存储器时, 与外部EPROM的读控制端 (OE) 相连, 低电平有效。



# 输入/输出口引脚P<sub>0</sub>、P<sub>1</sub>、P<sub>2</sub>和P<sub>3</sub>

P<sub>0</sub>口 (P<sub>0.0</sub>~P<sub>0.7</sub>) :

- ◆ 8位双向 I/O 口, 负载能力为8个LSTTL负载
- ◆ 功能:
  1. 不带片外存储器时,通用I/O口,传送CPU的输入输出数据
  2. 带片外存储器时,它为低8位地址线和8位数据线的复用端口
  3. 8751内部EPROM编程时,传送编程机器码或读出校验码:  
在EPROM编程时, 从P0口输入指令; 验证时, 输出指令

## 输入/输出口引脚P<sub>0</sub>、P<sub>1</sub>、P<sub>2</sub>和P<sub>3</sub>

P<sub>1</sub>口 (P<sub>1.0</sub>~P<sub>1.7</sub>) :

- ◆ 一个内部带上拉电阻的8位准双向I/O口，P<sub>1</sub>口的驱动能力为4个LSTTL负载
- ◆ 功能：
  1. 通用I/O口
  2. 8751内部EPROM编程时，传送低8位地址

# 输入/输出口引脚P<sub>0</sub>、P<sub>1</sub>、P<sub>2</sub>和P<sub>3</sub>

P<sub>2</sub>口 (P<sub>2.0</sub>~P<sub>2.7</sub>) :

◆ 一个内部带上拉电阻的8位准双向I/O口，P<sub>2</sub>口的驱动能力为4个LSTTL负载

◆ 功能：

1. 通用I/O

2. 带片外存储器时，作存储器的高8位地址线（与P<sub>0</sub>口配合使用）

3. 8751内部EPROM编程时，输入高8位地址（与P<sub>1</sub>口配合使用）

## 输入/输出口引脚P<sub>0</sub>、P<sub>1</sub>、P<sub>2</sub>和P<sub>3</sub>

P<sub>3</sub>口 (P<sub>3.0</sub>~P<sub>3.7</sub>) :

- ◆ 内部带上拉电阻的8位准双向I/O口
- ◆ 功能：
  1. 作为一般的I/O口使用
  2. 还具有第二功能

## P3口具有的第二功能

口线	特殊功能	信号名称
<b>P3.0</b>	<b>RXD</b>	串行输入口
<b>P3.1</b>	<b>TXD</b>	串行输出口
<b>P3.2</b>	<u>INT0</u>	外部中断 <b>0</b> 输入口
<b>P3.3</b>	<u>INT1</u>	外部中断 <b>1</b> 输入口
<b>P3.4</b>	<b>T0</b>	定时器 <b>0</b> 外部输入口
<b>P3.5</b>	<b>T1</b>	定时器 <b>1</b> 外部输入口
<b>P3.6</b>	<u>WR</u>	写选通输出口
<b>P3.7</b>	<u>RD</u>	读选通输出口

# 引脚说明举例---8031与片外存储器的连接

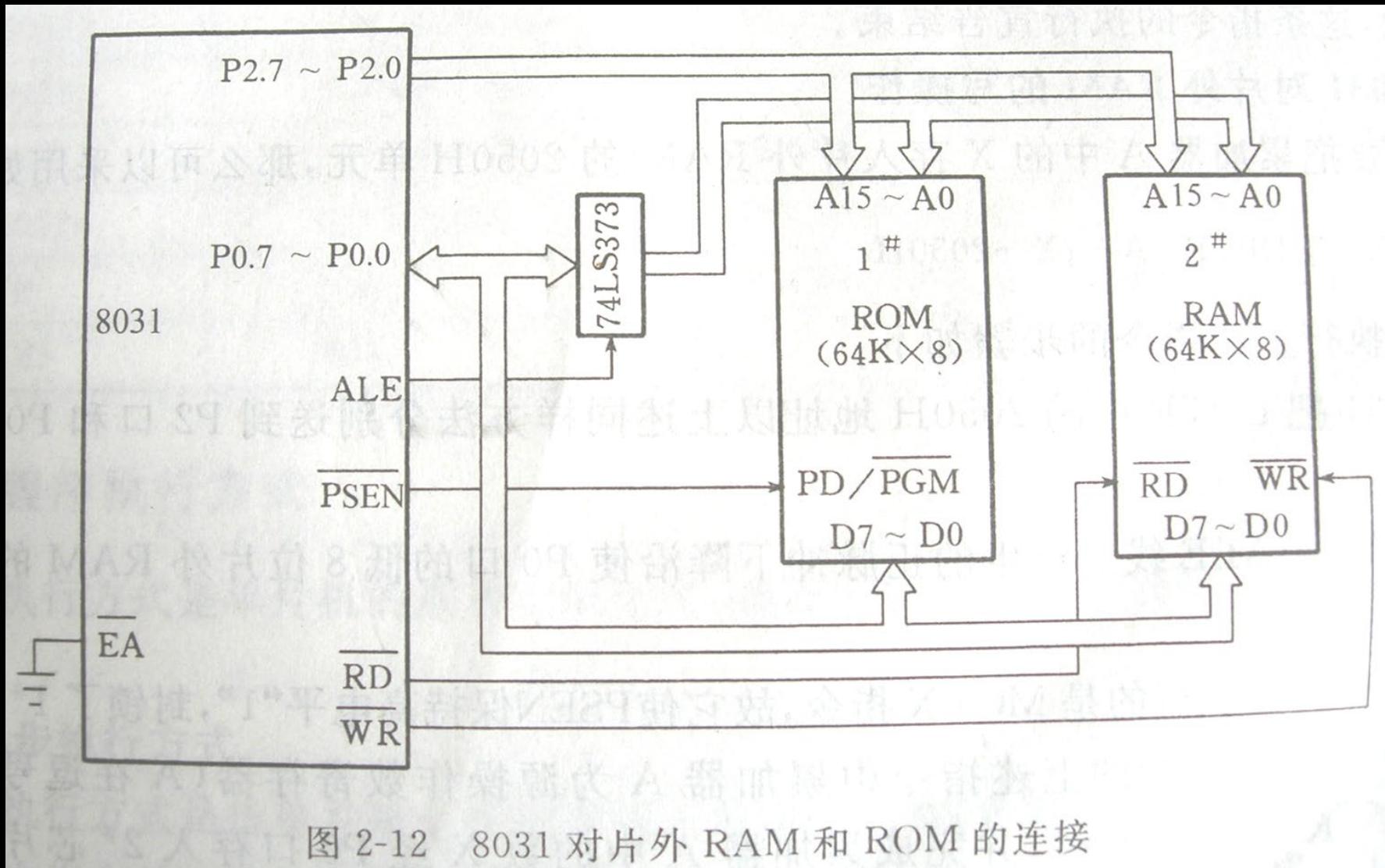


图 2-12 8031 对片外 RAM 和 ROM 的连接

片外ROM的读操作：

**MOV C A, @A+DPTR(=2050H)**

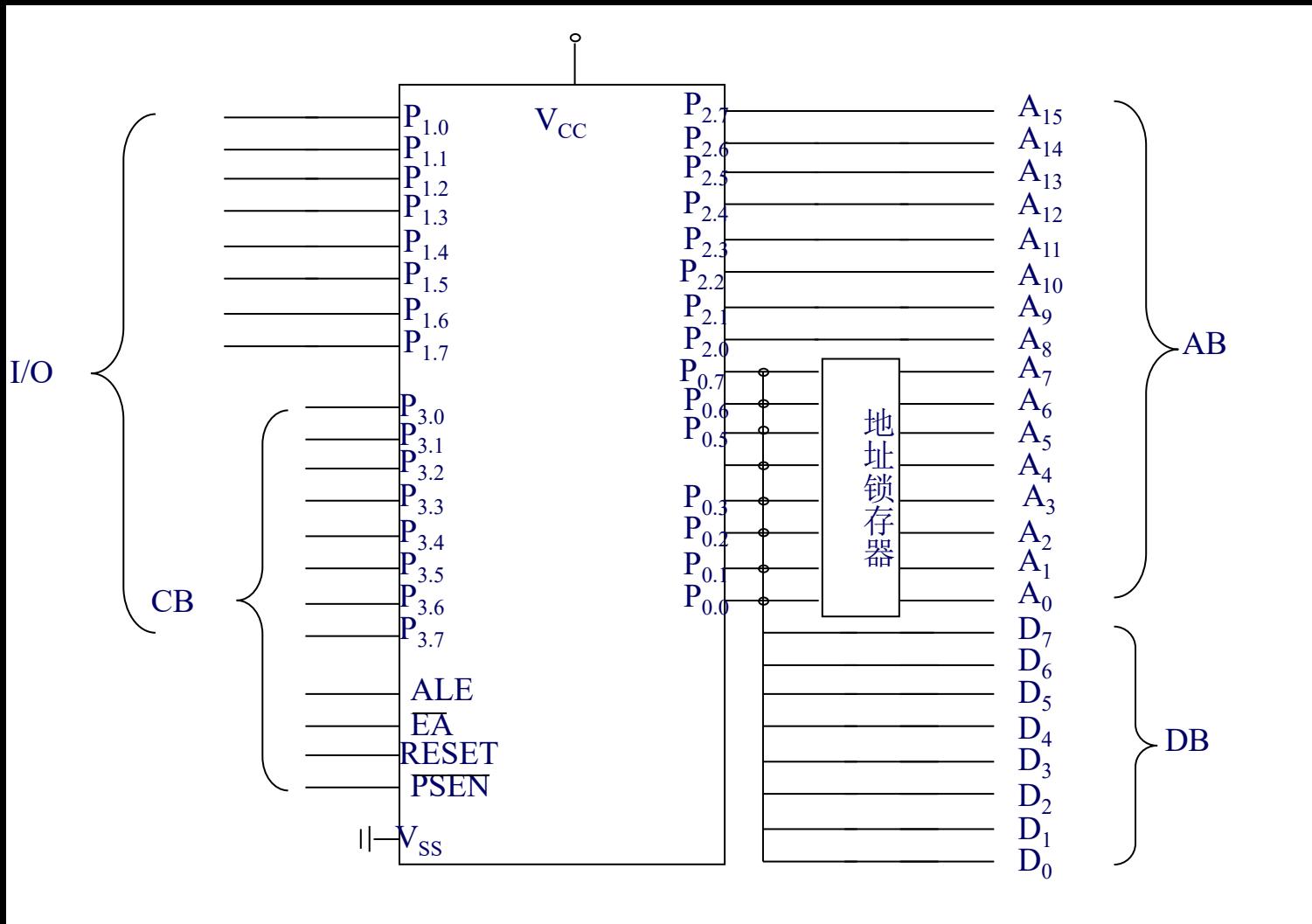
- 1 20H送到P2.7~P2.0, 50H送至P0.7~P0.0
- 2 一旦P0口上的地址数据50H稳定, ALE正脉冲下降沿使P0口上的地址数据被送至地址锁存器锁存
- 3 外部程序存储器读选通信号 $\overline{PSEN}$ 变为低电平,  $\overline{RD}$ 和 $\overline{WR}$ 保持高电平, 选择ROM存储器
- 4 ROM芯片读出2050H单元内容, 并送至P0口, CPU再把它送至累加器A

片外RAM的写操作:

**MOVX @DPTR(=2050H), A**

- 1 DPH中的20H送到P2.7~P2.0, DPL中的50H送至P0.7~P0.0
- 2 一旦P0口上的地址数据50H稳定, ALE正脉冲下降沿使P0口上的地址数据被送至地址锁存器锁存
- 3  $\overline{\text{PSEN}}$ 变为高电平,  $\overline{\text{RD}}=1$ ,  $\overline{\text{WR}}=0$ , 选择RAM存储器
- 4 CPU把A的内容经过P0口送至RAM的2050H单元

## § 2.3.2 外部总线结构

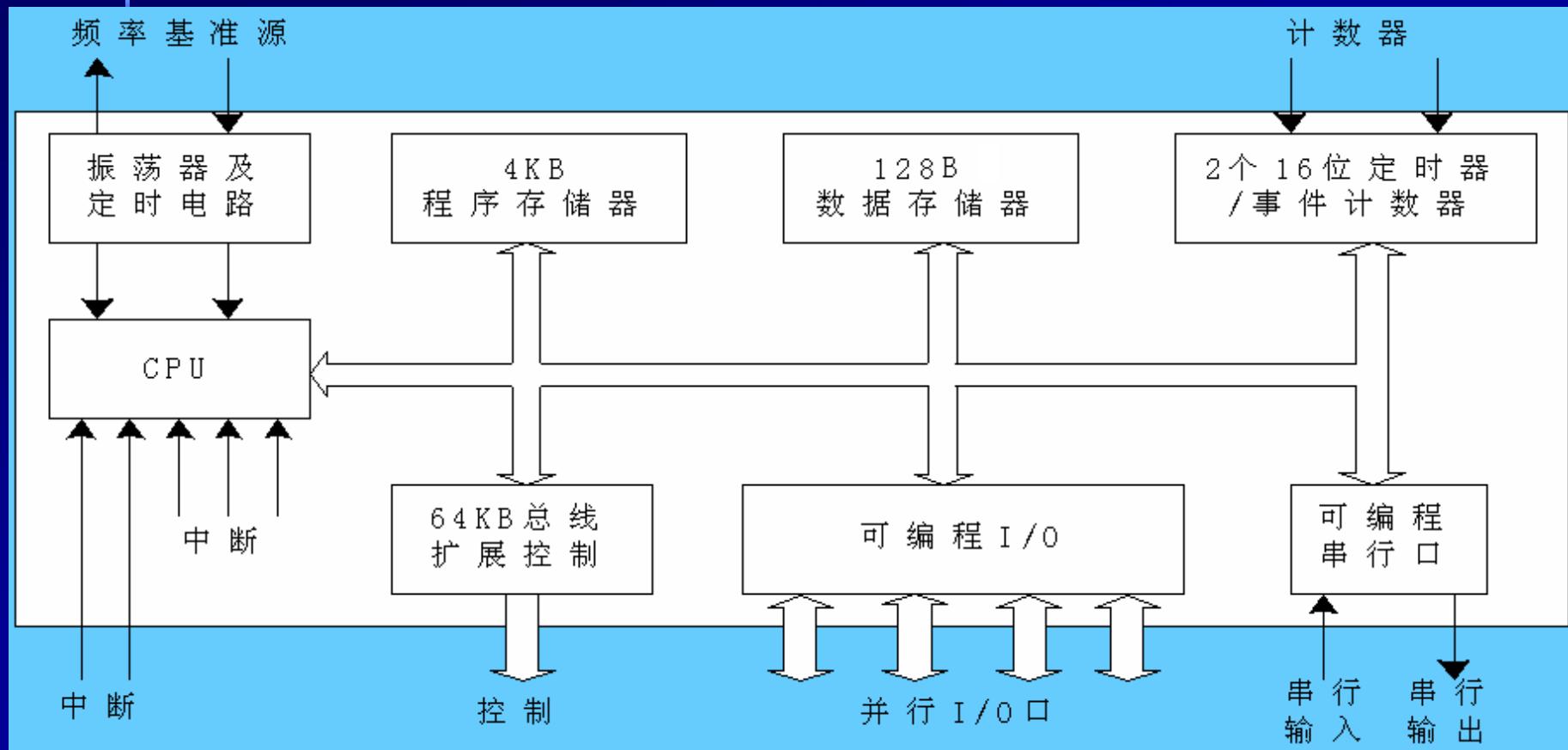


微型计算机中的总线通常分为：

- 地址总线（**AB**）：地址总线宽度为**16**位，由**P0**口经地址锁存器提供低**8**位地址（**A<sub>0</sub>-A<sub>7</sub>**）；**P2**口直接提供高**8**位地址（**A<sub>8</sub>~A<sub>15</sub>**）。地址信号是由**CPU**发出的，故地址总线是单方向的。
- 数据总线（**DB**）：数据总线宽度为**8**位，用于传送数据和指令，由**P0**口提供。
- 控制总线（**CB**）：控制总线随时掌握各种部件的状态，并根据需要向有关部件发出命令，包括**4**根独立的控制线（**RST**、**EA**、**ALE**、**PSEN**）和**P3**口的第二功能。

# 内容小结

## MCS-51单片机的系统结构框图



## 1. CPU

CPU是8位微处理器，是单片机的核心部件，它完成各种运算和控制操作，CPU由运算器和控制器两部分电路组成。

### （1）运算器

运算器包括ALU（算术逻辑部件）、ACC（累加器）、B寄存器、状态寄存器、暂存器1和暂存器2等部件，运算器的功能是进行算术运算和逻辑运算。

### （2）控制器

控制器包括程序计数器PC、堆栈指针SP、指令寄存器、指令译码器、数据指针DPTR、缓冲器以及定时与控制电路等。控制器完成指挥控制工作，协调单片机各部分正常工作。

## 2. 定时器/计数器

MCS-51单片机片内有两个16位的定时/计数器，即定时器T0和定时器T1。它们可以用于定时控制、延时以及对外部事件的计数和检测等。

## 3. 存储器

MCS-51系列单片机的存储器包括数据存储器和程序存储器，程序存储器和数据存储器的寻址空间相互独立，物理结构也不相同。

## 4. 并行I/O口

MCS-51单片机共有4个8位的I/O口（P0、P1、P2和P3），每一条I/O线都能独立地用作输入或输出。P0口为三态双向口，能带8个TTL门电路，P1、P2和P3口为准双向口，负载能力为4个TTL门电路。

## 5. 串行I/O口

MCS-51单片机具有一个采用通用异步工作方式的全双工串行通信接口，可以同时发送和接收数据。

## 6. 中断控制系统

8051共有5个中断源，即外部中断2个，定时/计数中断2个，串行中断1个。

## 7. 时钟电路

MCS-51芯片内部有时钟电路，但必须外接晶体振荡器和微调电容。时钟电路为单片机产生时钟脉冲序列，振荡器的频率范围为1.2MHz~12MHz，典型取值为6MHz。

## 8. 总线

以上所有组成部分都是通过总线连接起来，从而构成一个完整的单片机。系统的地址信号、数据信号和控制信号都是通过总线传送的，总线结构减少了单片机的连线和引脚，提高了集成度和可靠性。

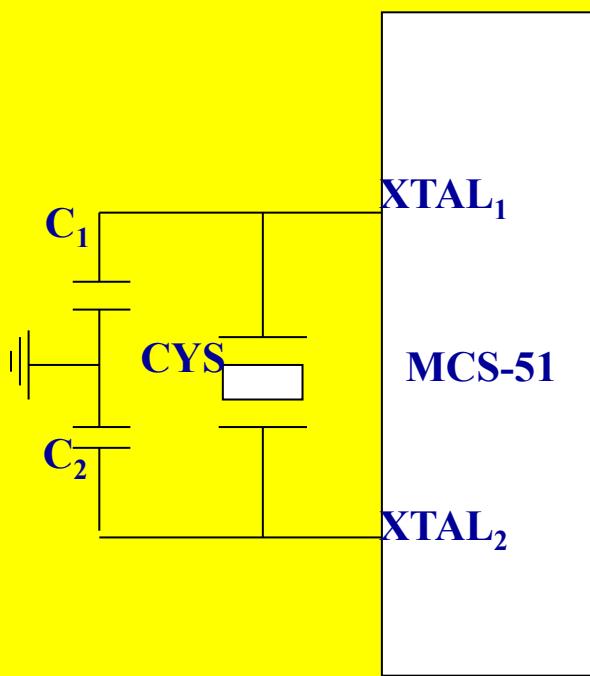
## 2.4 CPU时序及辅助电路

- 时钟电路
- 复位与复位电路
- CPU时序

## 2.4.1 单片机的时钟电路

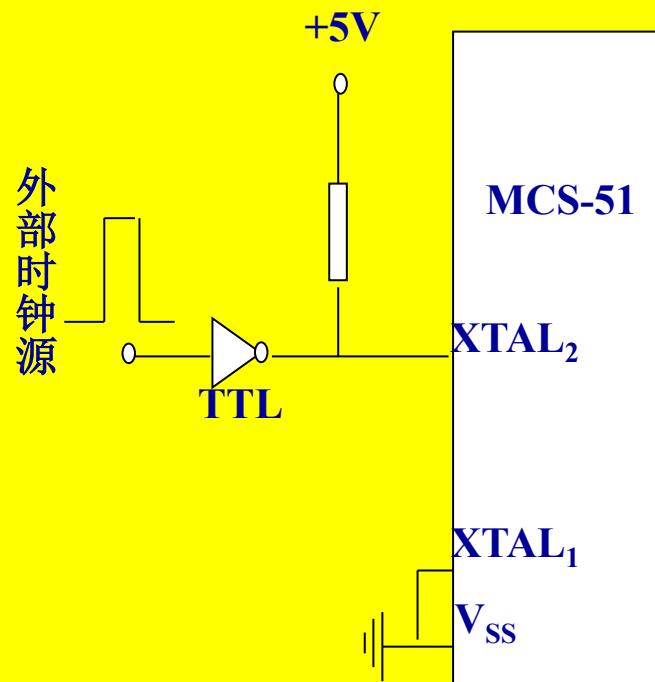
单片机时钟电路通常有两种形式：

- 1. 内部振荡方式：**MCS-51单片机片内有一个用于构成振荡器的高增益反相放大器，引脚**XTAL<sub>1</sub>**和**XTAL<sub>2</sub>**分别是此放大器的输入端和输出端。把放大器与作为反馈元件的晶体振荡器或陶瓷谐振器连接，就构成了内部自激振荡器并产生振荡时钟脉冲。
- 2. 外部振荡方式：**外部振荡方式就是把外部已有的时钟信号引入单片机内。



接入电容有利于振荡器起振  
外接石英晶体：电容值选30pF  
外接陶瓷谐振器：电容值47pF

内部时钟方式



外部时钟方式

## 2.4.2 MCS-51单片机复位及复位电路

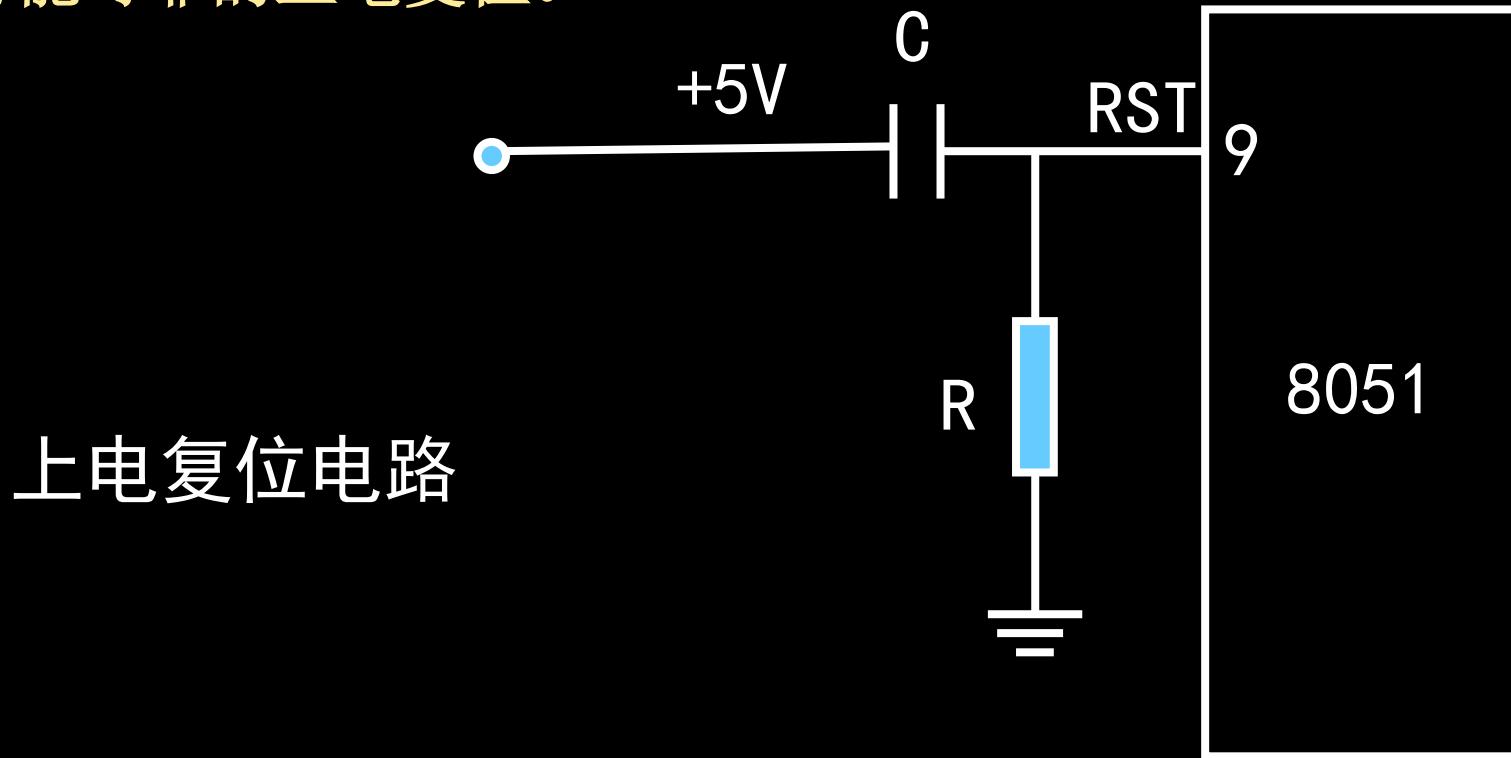
任何单片机在工作之前都要有个复位的过程，使CPU和系统中的其它部件都处于一个确定的初始状态。

如何进行复位？

在单片机的RST引脚上加上一定周期的高电平，复位需要不少于24个振荡周期（两个机器周期）的时间。

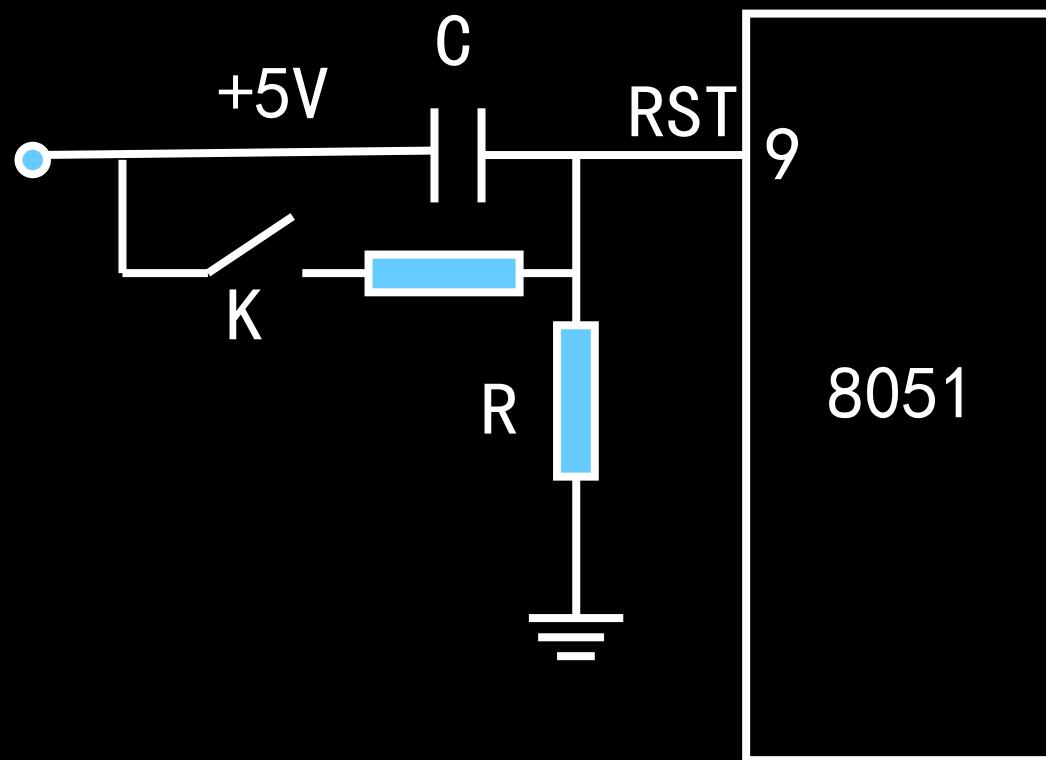
# 一、复位电路 (1)

上电复位电路。RST引脚是复位信号输入端，复位信号为高电平有效，其有效时间应持续24个振荡周期以上才能完成复位操作。在通电瞬间，由于RC的充电过程，在 RST端出现一定宽度的正脉冲，只要该正脉冲保持10ms以上，就能使单片机自动复位，在12MHz时钟时，通常C取10μF，R取8.2kΩ，这时能可靠的上电复位。

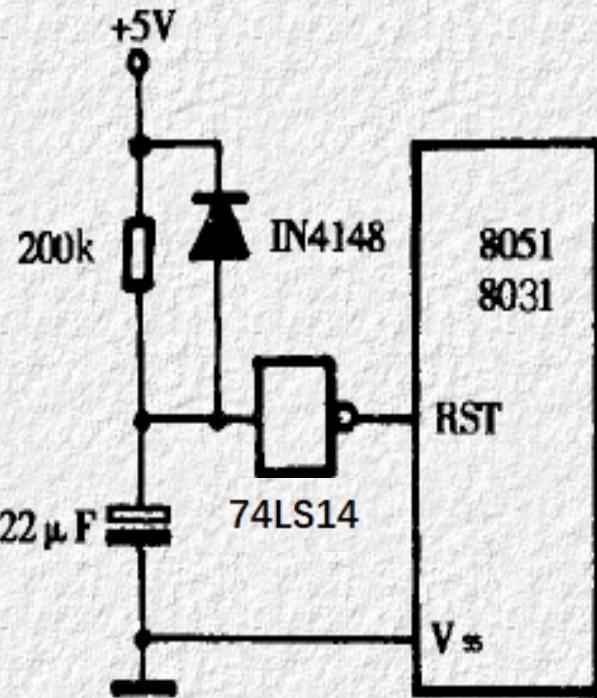


## 复位电路 (2)

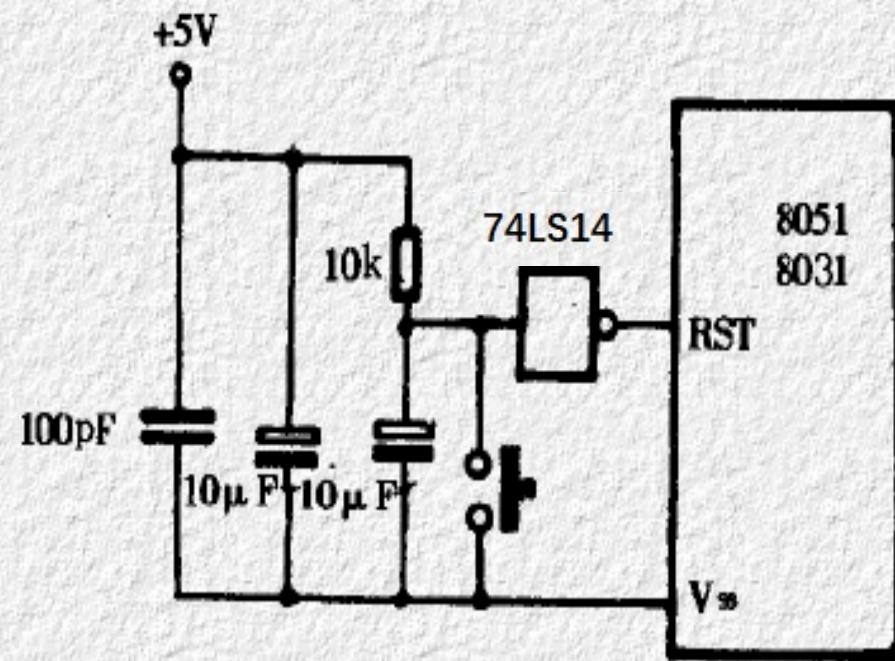
手动复位电路。多用在系统已经上电运行，期间需要进行复位时，就要通过手动按下按钮来进行复位。



# 两种实用的上电复位电路



(a) 上电复位电路



(b) 上电复位和按钮复位组合电路

实际应用时RC电路产生的信号一般需经施密特触发器整形，以保证可靠复位

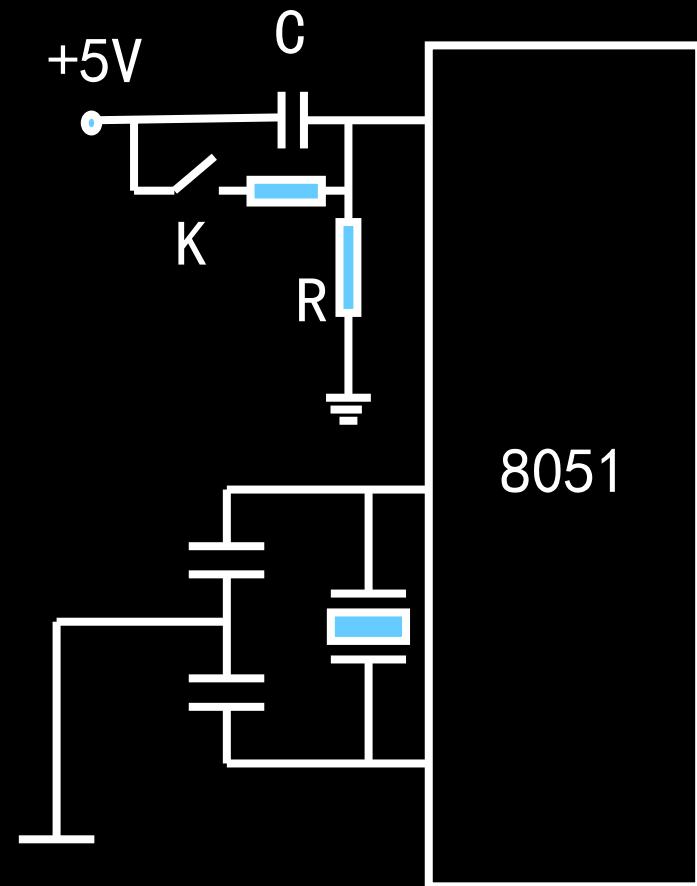
## 二、复位后单片机的状态

### 复位后各寄存器的状态

PC	0000H (程序入口)
P0、P1、P2、P3	0FFH (可以直接输入)
SP	07H (栈底已经设好)
PSW	00H (选择0组寄存器)
其余大部分都是	00H

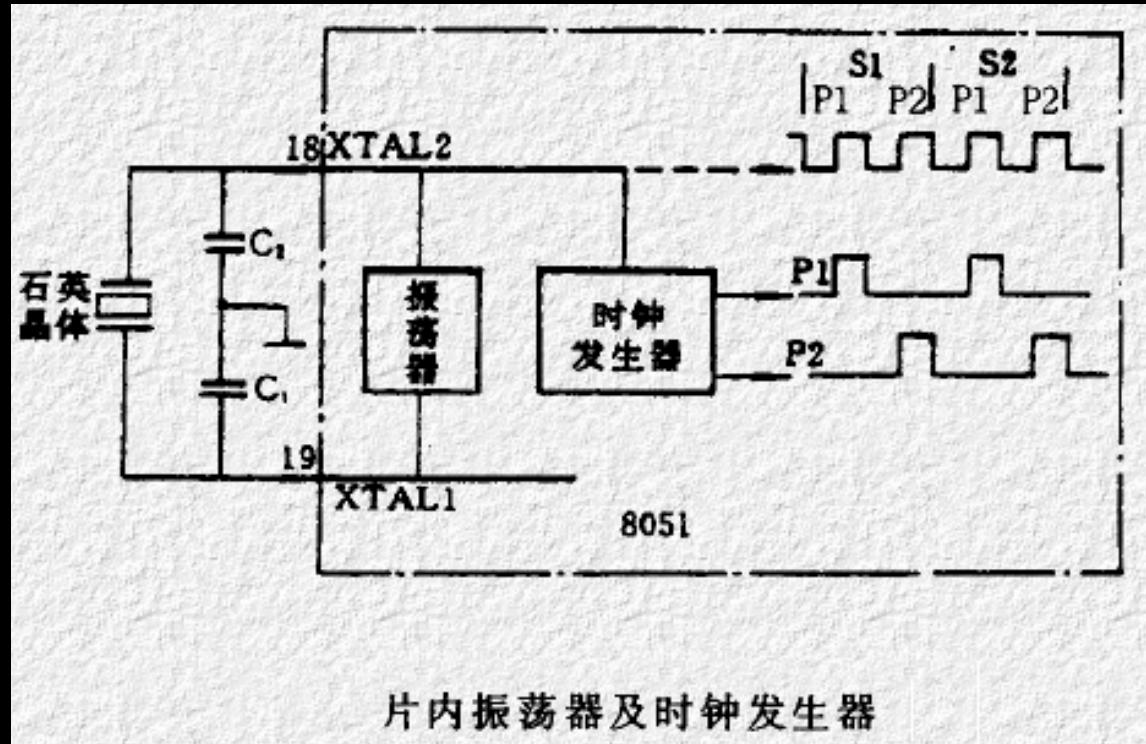
复位不影响内部RAM

# 最少外部电路条件下，可以独立工作的单片机系统 —— 最小系统



## 2.4.3 CPU时序

CPU的时序，就是CPU执行一条指令的各个微操作所对应的脉冲信号遵循的时间顺序。



时钟电路：振荡信号二分频，向芯片内提供2节拍信号。

时钟前半周期节拍P1有效，后半周期节拍P2有效。

按指令的执行过程定义了四种周期：

## 1. 振荡周期：

振荡周期指为单片机提供定时信号的振荡源的周期或外部输入时钟的周期

## 2. 时钟周期：又称作状态周期或状态时间S

- 振荡脉冲经二分频后形成时钟脉冲信号，是振荡周期的两倍
- 分为**P1**节拍和**P2**节拍
- **P1**节拍与**P2**节拍分工有所不同，通常**P1**完成算术逻辑运算，**P2**完成寄存器传送操作。

### 3. 机器周期：完成一个基本操作所需的时间

- 6个状态周期组成，12个振荡周期组成
- **S1P1, S1P2, S2P1, S2P2, …… S6P1, S6P2**

每个节拍持续一个振荡周期，每个状态持续两个振荡周期。

### 4. 指令周期：CPU执行一条指令所需的时间

一个指令周期通常含有1~4个机器周期

- 乘法和除法 —— 4个机器周期
- 其它指令：
  - 单字节和双字节：单周期或双周期
  - 三字节：双周期

若**MCS-51**单片机外接晶振为**12MHz**时，  
则单片机的四个周期的具体值为：

振荡周期=**1/12MHz = 1/12 μs**

时钟周期=**1/6 μs = 2倍振荡周期**

机器周期=**1 μs = 6个时钟周期**

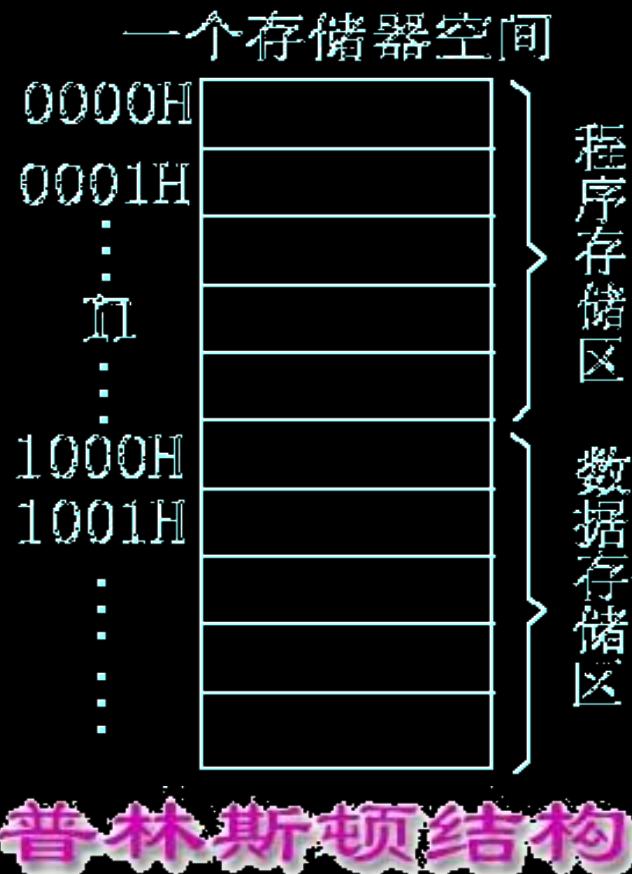
指令周期=**1~4 μs = 1~4个机器周期**

指令的执行时间分别为**1μs、 2μs、 4μs**

## 2.5 MCS-51存储器结构

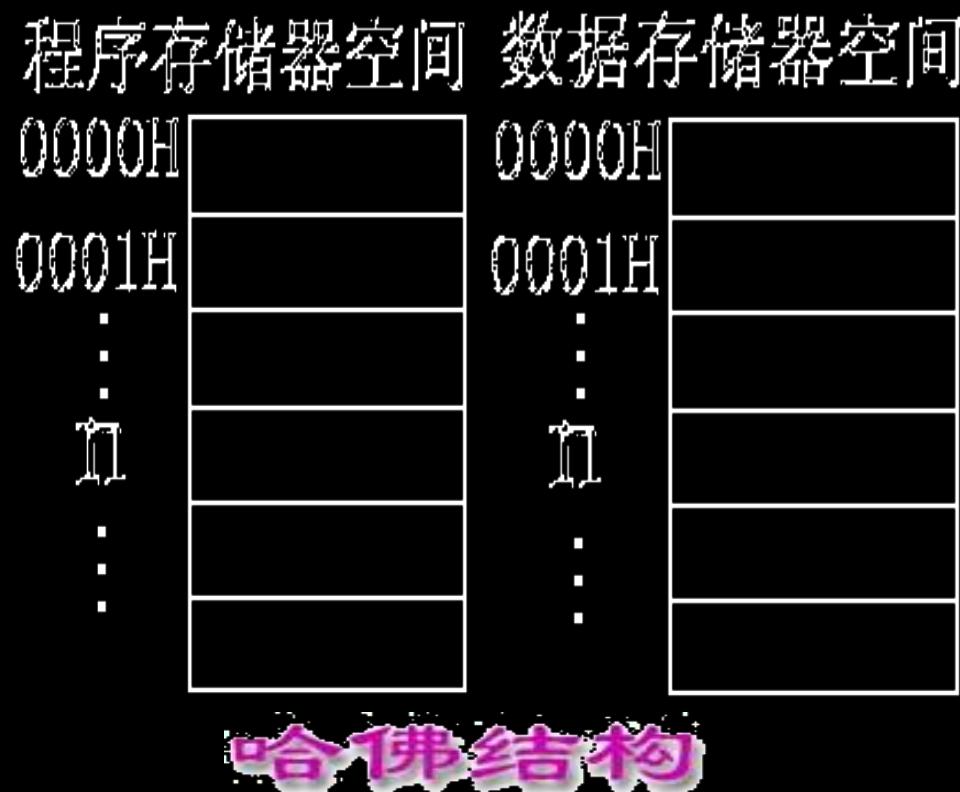
- ◆ 程序存储器**ROM**
- ◆ 数据存储器**RAM**

微型机的存储器在外部，在物理结构上只有一个地址空间，ROM和RAM可以随意安排在这一地址范围内不同的空间。CPU访问存储器时，一个地址对应唯一的存储器单元，可以是ROM也可以是RAM，并用同类访问指令。



单片机则采用程序存储器和数据存储器分开的结构形式

8051的存储器在物理结构上分开——  
分程序存储器空间和数据存储器空间



## 2.5.1 MCS-51存储器的结构

物理结构上有4个存储器地址空间：

片内程序存储器空间-片内4K ROM

片外程序存储器空间-片外可扩充64K ROM

片内数据存储器空间-片内128B RAM + SFR

片外数据存储器空间-片外可扩充64K RAM

从用户使用的角度，存储器地址空间分为三类

①程序存储器地址：

片内、片外统一编址0000H-FFFFH

**64K**字节（用**16**位地址）；

②数据存储器：

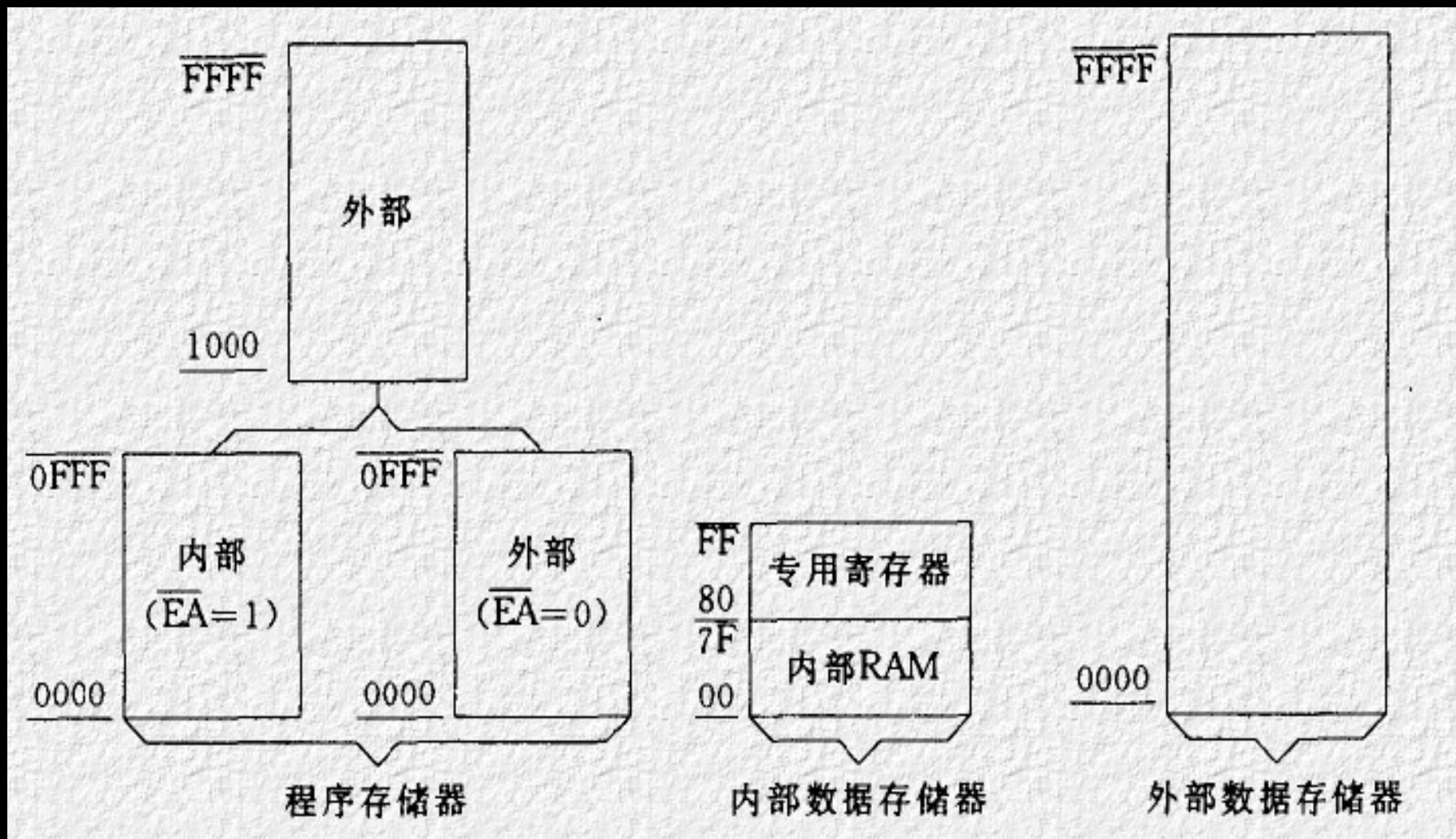
**64K**字节片外地址空间，

地址从0000H-FFFFH（用**16**位地址）；

③**256**字节片内数据存储器地址空间

（用**8**位地址）

# 8051存储器空间配置图



# 8051CPU区分存储器地址空间的方法

- 上述三个存储空间地址是重迭的，如何区别这三个不同的逻辑空间呢？

8051的指令系统设计了不同的数据传送指令符号：

- 访问片内、片外**ROM**指令用**MOVC**
- 访问片外**RAM**指令用**MOVX**
- 访问片内**RAM**指令用**MOV**

## 2.5.2 程序存储器地址空间

- 程序存储器用于存放编好的程序和表格常数
- 程序存储器通过16位程序计数器(PC)寻址，寻址空间为64K字节
- 没有指令使程序能从程序存储器空间转移到数据存储器空间

# 8051 / 8751的64K程序存储器空间

- 片内ROM/EPROM为4K字节，地址为0000H-0FFFH；
- 片外最多可扩至64K字节ROM / EPROM，地址0000H-FFFFH，片内外是统一编址的；
- 当外部允许访问引脚EA接高电平时，8051的程序计数器PC在0000H-0FFFH范围内(即前4K字节地址)执行片内ROM中的程序；当指令地址超过0FFFH后，就自动地转向片外ROM取指令；
- 当引脚EA接低电平(接地)时，8051片内ROM不起作用，CPU只能从片外ROM / EPROM中取指令，地址从0000H开始编址。这种接法特别适用于采用8031单片机的场合，由于8031片内不带ROM，所以使用时必须使EA=0，以便能够从外部扩展EPROM(如2764，2732)中取指令；
- 8051从片内程序存储器和片外程序存储器取指时执行速度相同。

# 程序存储器的某些单元留给系统使用

存储单元	保留目的
0000H-0002H	复位后初始化引导程序
0003H-000AH	外部中断0
000BH-0012H	定时器0溢出中断
0013H-001AH	外部中断1
001BH-002AH	定时器1溢出中断
0023H-002AH	串行端口中断

存储单元0000H-0002H用作8051上电复位后引导程序存放单元。

因为8051 / 8031 / 8751上电复位后程序计数器PC的内容为0000H，所以CPU总是从0000H开始执行程序。在这三个单元中存有无条件转移指令，那么程序就被引导到转移指令指定的ROM / EEPROM空间去执行。

在**8051**的程序存储器的开头都安排的是一条转移指令

ORG 0000H

AJMP #add11 ;

或 LJMP #add16 ;

# 中断矢量区

- 0003H-002AH单元均匀地分为五段，用作五个中断服务程序的入口地址。

例如，外部中断引脚INT0(P3.2)有效时，即引起中断申请，CPU响应中断后自动将地址0003H装入PC，程序就自动转向0003H单元开始执行。

- 如果事先在0003H-000AH存有无条件转移指令，程序就转移到指定的中断服务程序空间去执行。
- 0003H等也称中断矢量地址。

# 中断矢量地址表

中断源	中断服务程序入口地址
外部中断0	0003H
定时 / 计数器0溢出	000BH
外部中断1	0013H
定时 / 计数器1溢出	001BH
串行口中断	0023H

## 2.5.3 数据存储器地址空间

- 数据存储器**RAM**用于存放运算的中间结果、数据暂存和缓冲、标志位等
- 数据存储器空间也分成片内和片外两大部分，即片内**RAM**和片外**RAM**
- 8051片外数据存储器空间为64K，从0000H-0FFFFH
- 片内存储器空间为256字节，地址从00H-0FFH

# 片外RAM

- 片外数据存储器与片内数据存储器空间低地址0000H-00FFH是重迭的
- 8051有MOV和MOVX两种指令，用以区分片内、片外RAM空间
- 片内RAM使用MOV指令
- 片外64K RAM空间使用MOVX指令

# 片内**RAM**

- 片内数据存储器为**8**位地址， 最大可寻址**256**个单元
- 分为两个部分：  
低**128**字节 (00H-7FH) 是  
真正的**RAM**区  
高**128**字节 (80H-FFH) 为  
特殊功能寄存器(**SFR**)区

# 1. 低128字节RAM (00H-7FH)

分为三个区：

- 工作寄存器区 (00H-1FH)
- 位寻址区 (20H-2FH)
- 数据缓冲区 (30H-7FH)

7FH	127
2FH	47
2EH	46
2DH	45
2CH	44
2BH	43
2AH	42
29H	41
28H	40
27H	39
26H	38
25H	37
24H	36
23H	35
22H	34
21H	33
20H	32
1FH	31
18H	寄存器区3
17H	24
10H	寄存器区2
0FH	23
08H	寄存器区1
07H	16
00H	15
	8
	7
	0

# 工作寄存器区 (00H-1FH)

- 00H-1FH地址安排为四组工作寄存器区
- 每组有8个工作寄存器 (R0-R7) , 共占32个单元
- 通过对程序状态字PSW中RS1、RS0设置

RS1	RS0	工作寄存器组	R0	R1	R2	R3	R4	R5	R6	R7
0	0	工作寄存器组 0	00H	01H	02H	03H	04H	05H	06H	07H
0	1	工作寄存器组 1	08H	09H	0AH	0BH	0CH	0DH	0EH	0FH
1	0	工作寄存器组 2	10H	11H	12H	13H	14H	15H	16H	17H
1	1	工作寄存器组 3	18H	19H	1AH	1BH	1CH	1DH	1EH	1FH

# 位寻址区 (20H—2FH)

字节地址	D7 ~ D0							
	7F	7E	7D	7C	7B	7A	79	78
2FH								
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

每一位都分配一个8位的位地址，从00H~7FH。

程序设计时，一般把各种程序状态标志和一些位控制变量放在位寻址区中，由程序进行位处理。

位寻址区的RAM单元也可以作为一般的的数据缓冲区使用。

# 数据缓冲区

- 地址范围: 30H-7FH
- 真正用作存放数据的RAM: 非常灵活
  - 应合理安排, 如: 堆栈区, 显示区

## 2. 高128字节RAM——特殊功能寄存器(SFR)区

- 8051片内高128字节RAM中，有21个专用寄存器(SFR)。特殊功能寄存器是用于对片内各功能模块进行管理、控制、监视的控制寄存器和状态寄存器。
- 它们离散地分布在80H-FFH的RAM空间中。
- 在21个特殊功能寄存器SFR中，有11个特殊功能寄存器具有位寻址能力，它们的字节地址正好能被8整除。

# 本章小结

- **MCS-51的内部结构**
- **MCS-51的CPU** 
- 引脚和片外总线
- **CPU时序和辅助电路**
- 程序存储器与数据存储器 

# 第三章 MCS-51的指令系统 Instruction Set

指令是**CPU**执行某种操作的命令  
指令系统是全部指令的集合

重点掌握： **寻址方式和各种指令**

## § 3.1 概述

### MCS-51指令的格式

[标号:] 操作码 [目的操作数] [, 源操作数] [;注释]

- ◆ 方括符[ ]表示可选项
- ◆ 标号代表指令所在地址，1-8个字母/数字，以冒号结尾。标号是可选的，不是所有指令语句都需要标号，一般只在程序调用或者转移指令需要的地方才设置标号，主要是为了便于查找和修改程序。
- ◆ 操作码表示该指令所能执行的操作功能，不可缺省

# MCS-51指令的格式

[标号:] **操作码** [目的操作数] [, 源操作数] [;注释]

操作数表示参加操作的数本身或操作数所在的地址，包括目的操作数和源操作数。

- ◆ **目的操作数**是指数据源传送到的目的地。可以是累加器A、工作寄存器或者片内RAM存储单元。
- ◆ **源操作数**是指待传送的数据源。可以是立即数、累加器A、工作寄存器或者片内RAM存储单元。
- ◆ **注释，以分号开头，程序编译时不被翻译。**如果注释内容超过一行，换行后前面仍需加上一个分号。

如：

LOOP: **MOV P1, #80H ;将80H送P1**

[标号]	操作码	第一操作数	第二操作数	[注释]
		(目的操作数)	(源操作数)	

注：白色的内容不是必须的

# 指令表示方法

两种表示方法：

- **机器码表示方法**

用计算机能够直接识别、执行的二进制代码表示指令，常用二进制代码的速记形式——16进制代码表示

- **助记符表示方法**

用表征指令功能的字符形式表示指令

$A = A + 07$	
二进制	00100100 00000111
16进制	24 07
助记符	ADD A, #07H

# 指令格式——字节数



操作码

单字节指令

操作码

操作数

双字节指令

操作码

第一操作数

第二操作数

三字节  
指令

# 指令分类：

111条指令

字节数 {

- 单字节指令：49条
- 双字节指令：45条
- 三字节指令：17条

运算速度 {

- 单周期指令：64条
- 双周期指令：45条
- 四周期指令： 2条

功能

数据传送类: 29条

算术运算类: 24条

逻辑运算类: 24条

控制转移类: 17条

位操作类: 17条

## § 3.2 寻址方式

寻找操作数地址或指令地址的方式

七种寻址方式：

立即寻址

直接寻址

寄存器寻址

寄存器间接寻址

变址寻址

相对寻址

位寻址

寻址空间：

存储器的存储单元地址和位地址

程序存储器、片外数据存储器：以**字节**为单位寻址

片内数据存储器：以**字节**为单位寻址，某些存储单元以**位**为单位寻址（位地址空间包括字节地址能被8整除的11个SFR，以及低128字节RAM区中的位寻址区）

# 常用符号

**R<sub>n</sub>**: 当前选中的工作寄存器组R0-R7

**R<sub>i</sub>**: 当前选中的工作寄存器组中的R0或R1

**#data**: 8位立即数, # 为立即寻址符

**#data16**: 16位立即数

**direct**: 片内RAM或SFR的地址 (8位)

**@**: 寄存器间接寻址符

**bit**: 片内RAM或SFR的位地址

**addr11**: 11位目的地址, 用于AJMP和ACALL指令

**addr16**: 16位目的地址, 用于LJMP和LCALL指令

**rel:** 相对地址

范围为-126～+129（双字节指令）

或-125～+130（三字节指令）

**/**: 位操作指令中，该位求反后参与操作，不影响该位原来的值

**×**: 片内RAM的直接地址或寄存器

**(X)**: 相应地址单元中的**内容**。在直接寻址方式中，表示直接地址X中的内容。在间接寻址方式中，表示由间接寄存器X指出的地址单元中的内容。

**→**: 箭头左边的内容送入箭头右边的单元内

### 3.2.1 立即寻址 Immediate Addressing

操作数存在程序  
存储器中

操作数就包含在指令代码中，在操作码之后，称为立即数，用“#”表示。

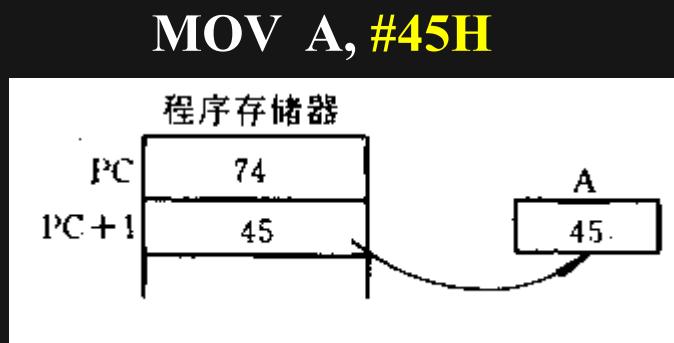
如：

MOV A, #45H ; #data

MOV DPTR, #1245H ; #data16

长度：2字节或3字节

作用：一般用于寄存器赋值



注意：立即数只能作源操作数，不能作目的操作数。

### 3. 2. 2 直接寻址

### Direct Addressing

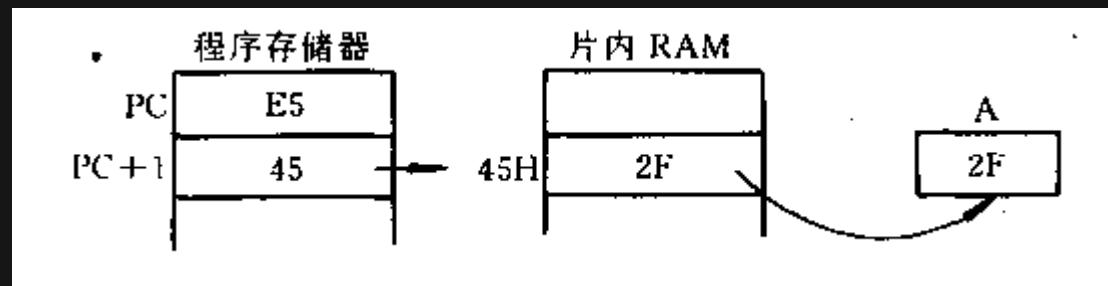
直接使用操作数所在单元的地址找到操作数，称这种方法为直接寻址 (direct)。

操作数地址只能以8位数指定，直接寻址方式只能在 SFR、片内RAM的低128个字节、位地址空间内寻址。

**MOV A, 45H**  
**MOV C, 60H**  
**MOV A, 0F0H**

SFR和位地址空间只能通过直接寻址访问

**MOV A, 45H**



## 直接寻址注意事项：

- 在**8位直接地址**前不加任何符号
- **SFR**寻址时既可使用物理地址，也可使用助记符  
**MOV A, SP** (E5 81H)  
**MOV A, 81H** (E5 81H)
- 堆栈操作用直接寻址方式访问累加器**A**，必须使用**ACC/0E0H**

<b>PUSH A</b>	<b>POP A</b>	×
<b>PUSH ACC</b>	<b>POP ACC</b>	√
<b>PUSH 0E0H</b>	<b>POP 0E0H</b>	√

- 字节地址与位地址的区分：

**MOV A, 20H; A← (20H)**

**MOV C, 20H; CY ← (20H) ((24H).0)**

### 3.2.3 寄存器寻址 Register Addressing

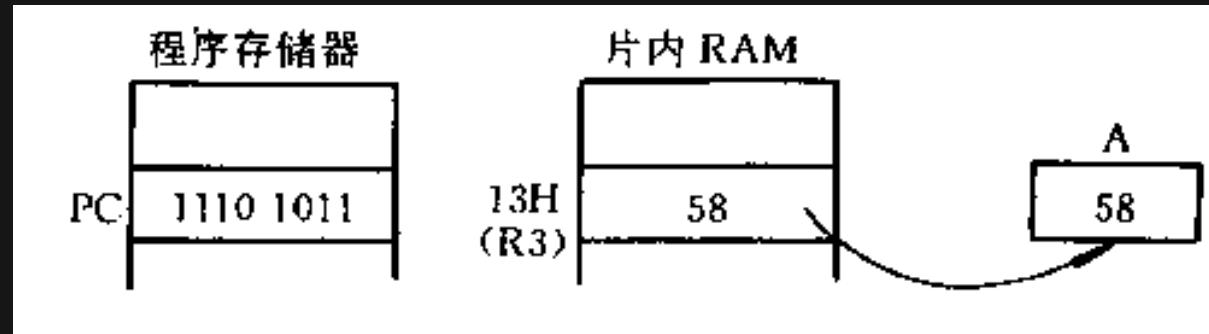
是由指令指出某个寄存器的内容作为操作数，也就是对选定的工作寄存器R0～R7、累加器A（C）、通用寄存器B、数据指针DPTR中的数进行操作。

例：MOV A, R0；将R0工作寄存器中的数据送到累加器A中去。

A、B、DPTR和C隐含在操作码中

Rn由指令操作码低三位选定

**MOV A, R3** (机器码: EBH)



**MOV A, 00H 和 MOV A, R0 有什么区别？**

这两条指令执行的结果是完全相同的，都是将 00H 单元中的内容送到 A 中去，执行两条指令都需要一个机器周期。但是执行的过程不同，第一条指令变成最终的目标机器码要两个字节（E5H 00H），而第二条则只要一个字节（E8H）就可以了。

### 3.2.4 寄存器间接寻址

### Register Indirect Addressing

把地址放在一个寄存器中，根据这个寄存器中的数值决定该到哪个单元中取数据。

- R0, R1—8位地址，片内低128字节或片外低256B RAM
- DPTR—16位，片外64KB RAM

如：  
MOV A, @R0  
MOVX A, @R0  
MOVX A, @DPTR

操作数在  
片内RAM中

操作数在  
片外RAM中

操作数在  
片外RAM中

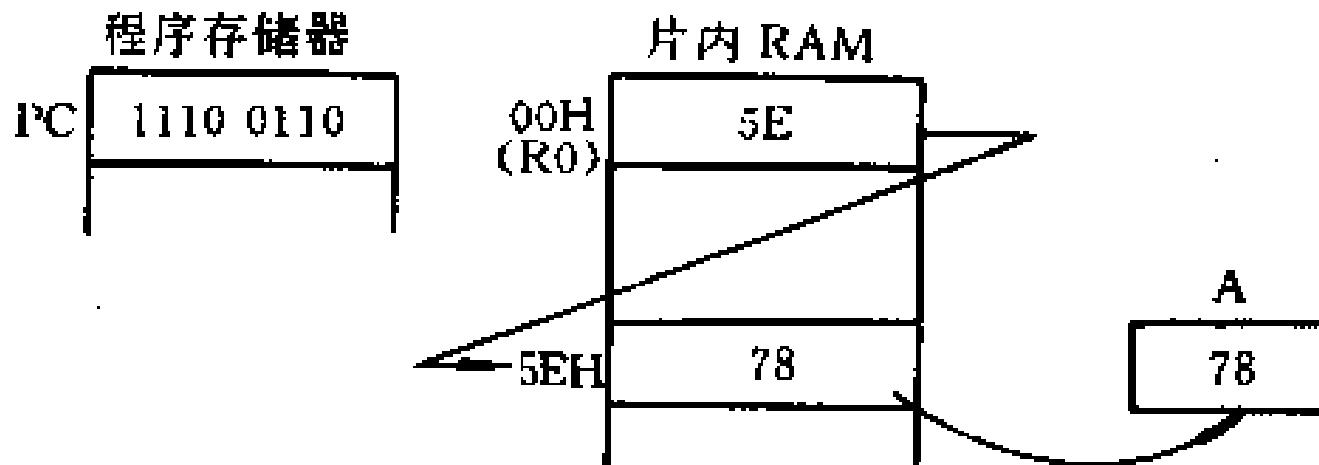
不用来访问SFR中的SP，  
MOV R0,#81H   
MOV A,@R0

而是PUSH direct  
POP direct

# 寄存器间接寻址示意图

**MOV A, @R0** (机器码: E6H)

指令机器码的最低一位指出R0或者R1  
在一个字节的机器码中既包含操作码又包含操作数



### 3.2.5 变址寻址(基址+变址)

#### Base-Register-plus-Index-Register-Indirect Addressing

以**DPTR**或**PC**为基址寄存器，存放操作数的基地址；累加器**A**为变址寄存器，存放相对于基地址的偏移量。把两者内容相加，结果作为操作数的地址。

操作数的有效地址=基址寄存器DPTR (或PC)+A

常用于查表操作，用来访问程序存储器中的数据表。

**MOVC A, @A+DPTR** ; (A+DPTR) → A

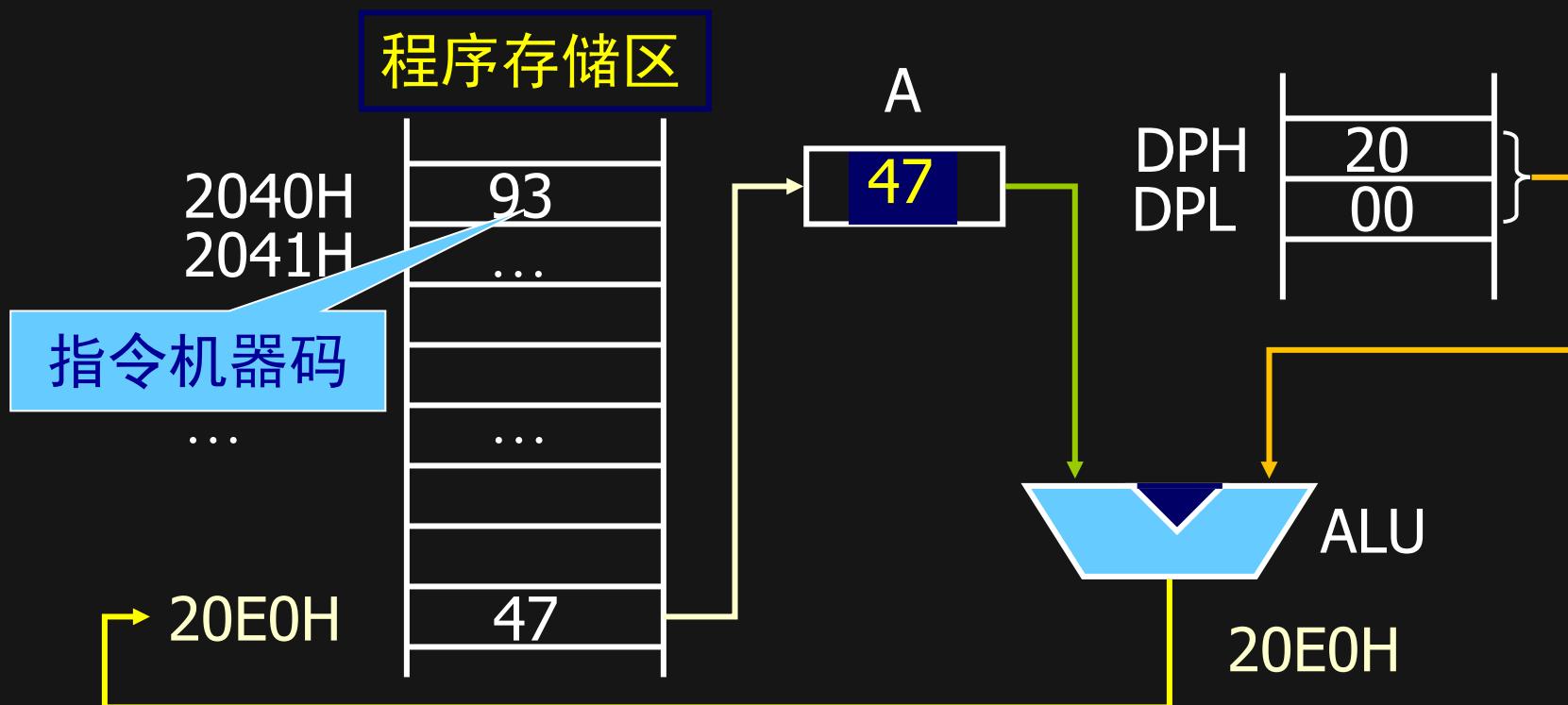
**MOVC A, @A+PC** ; PC+1 → PC, (A+PC) → A

特点：

1. 寻址空间：程序存储器**ROM**
2. 指令码内隐含有基址寄存器**DPTR**或**PC**
3. 隐含累加器**A**

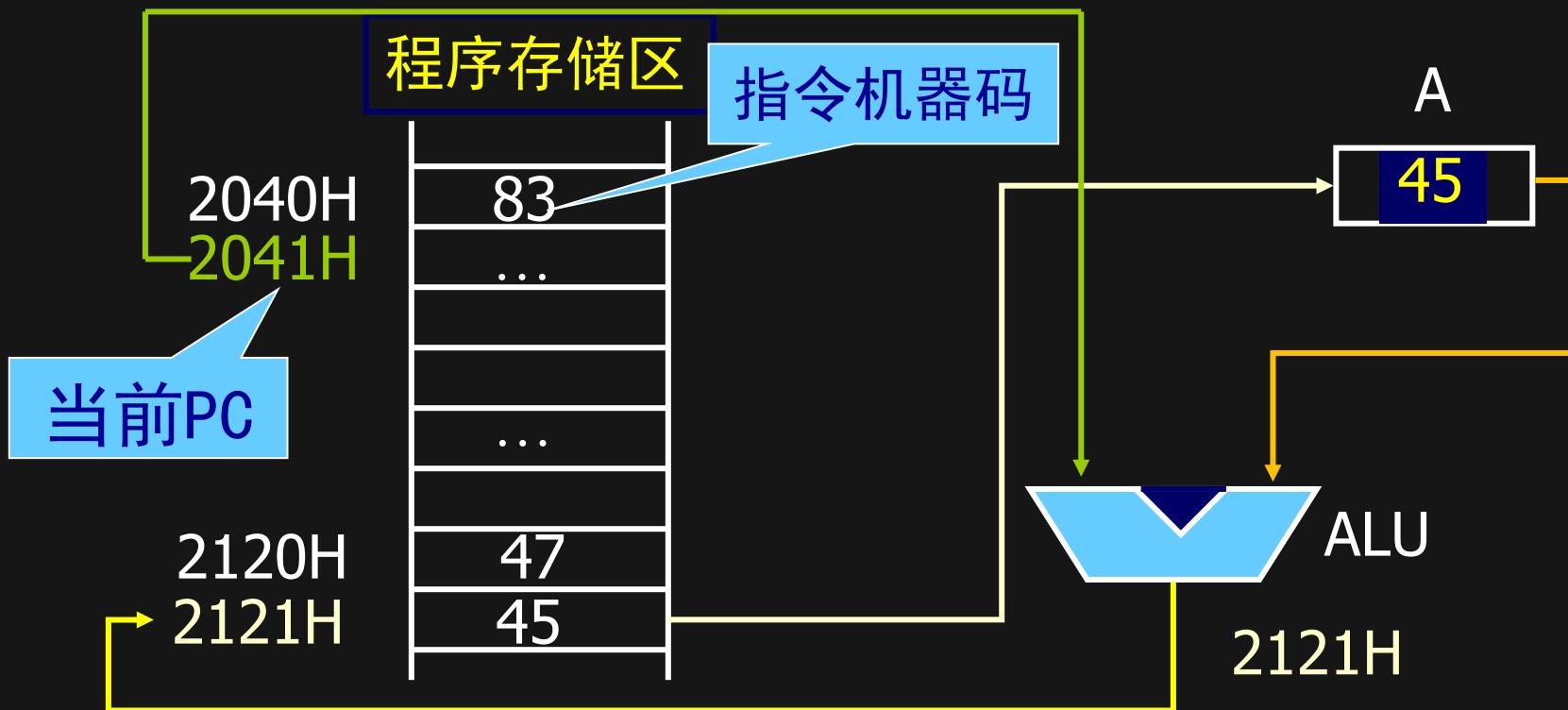
如: MOVC A, @A+DPTR (机器码:93)

设DPTR=2000H, A=47H



如: MOVC A, @A+PC (机器码:83)

设PC=2040H A=E0H



## 3.2.6 相对寻址

### Relative Addressing

用于相对转移指令的寻址方式，寻找下一条要执行指令的地址。将PC中的当前内容与指令第二字节给出的数据相加，结果作为跳转指令的转移地址（转移目的地址）。

改变PC值

相对寻址的有效地址为： $D = PC + rel$

其中，PC中的当前内容称为基地址，指令第二字节给出的数据称为偏移量，它是一字节带符号数。

常用于跳转指令。

如：JC 23H；地址用rel表示

若C=0，不跳转；C=1，跳转。

偏移量 取值范围 -128 - +127

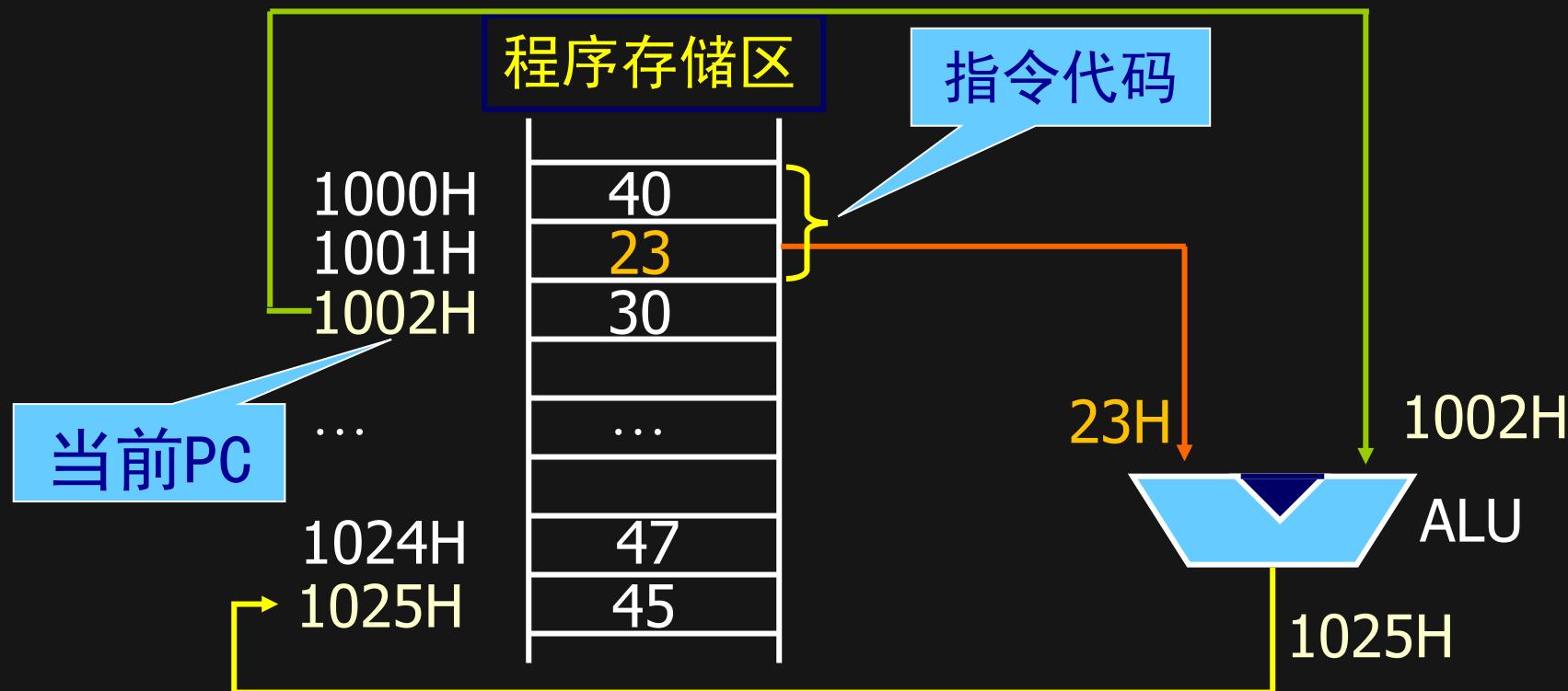
相对地址取值范围 -126 - +129 (2字节转移指令)

改变PC值，寻址程序存储器ROM空间

如: JC 23H (机器码:40 23)

假设PC=1000H CY=1

JC rel ; PC+2→PC, 若CY=1, 则PC+rel→PC



### 3. 2. 7 位寻址

### Bit Addressing

对片内RAM的位寻址区和某些可位寻址的特殊功能寄存器进行位操作时的寻址方式。

如： SETB 3DH; 将(27H).5位置1  
CLR C ; CY位清0

操作数在片内  
RAM的位地址区  
或SFR某些位中

8086/8088 CPU 无此寻址方式

## 小结：寻址方式以及相应的存储器空间

寻址方式	寻址空间（操作数存放空间）
立即寻址	程序存储器（在指令操作码之后）
直接寻址	片内RAM低128字节、SFR，位地址
寄存器寻址	工作寄存器R0～R7, A, B, C, DPTR
寄存器间接寻址	片内RAM:@R0, @R1 片外RAM:@R0, @R1, @DPTR
变址寻址	程序存储器:@A+PC, @A+DPTR
相对寻址	程序存储器256字节范围内:PC+偏移量
位寻址	片内RAM的位寻址区(20H～2FH字节地址) 某些可位寻址的SFR

## § 3.3 指令系统

### ■ 按功能分——

- 数据传送类指令
- 算术运算类指令
- 逻辑运算类指令
- 程序控制类指令
- 位操作类指令

### 3. 3. 1 数据传送类指令 (29条)

#### Data Transfer Instruction

数据传送类指令是最常用、最基本的一类指令。

- 可实现寄存器与寄存器之间、寄存器与片内RAM之间，以及片内RAM单元之间的数据传送。
- 可实现立即数送累加器和寄存器的传送。
- 可实现累加器与片外数据存储器之间的传送，及并行I/O口内容直接与片内RAM之间的数据传送。

助记符：

MOV、MOVX、MOVC

XCH、XCHD、SWAP

PUSH、POP

❖ 目的操作数寻址方式（3种）：

直接寻址、寄存器寻址、寄存器间接寻址

❖ 源操作数寻址方式（5种）：

立即寻址、直接寻址、寄存器寻址、  
寄存器间接寻址、变址寻址。

除了目的操作数为ACC的指令影响奇偶标志P外，  
一般不影响标志位。

# 一、8位数通用传送指令

## 1 以累加器为目的操作数的指令 (4条)

**MOV A, Rn** ; Rn→A 寄存器寻址

**MOV A, direct** ; (direct)→A 直接寻址

**MOV A, @Ri** ; (R<sub>i</sub>)→A 寄存器间接寻址

**MOV A, #data** ; data→A 立即数寻址

将源操作数指定内容送到A中。

## 2 以寄存器Rn为目的操作数的指令 (3条)

**MOV Rn, A**

**MOV Rn, direct**

**MOV Rn, #data**

将源操作数指定的内容送入当前工作寄存器，源操作数不变。

### 3 以直接地址为目的操作数的指令 (5条)

**MOV direct, A**

**MOV direct, Rn**

**MOV direct1, direct2** (机器码中源地址在前)

**MOV direct, @Ri**

**MOV direct, #data** (机器码中目的地址在前)

功能：将源操作数指定的内容送入由直接地址指出的片内存储单元中。

### 4 以间接地址为目的操作数的指令 (3条)

**MOV @Ri, A** ; A → (R<sub>i</sub>)

**MOV @Ri, direct** ; (direct) → (R<sub>i</sub>)

**MOV @Ri, #data** ; data → (R<sub>i</sub>)

功能：把源操作数指定的内容送入以R0或R1为地址指针的片内存储单元中。

## 二、十六位数的传送指令（1条）

**MOV DPTR, #data16**

8051是一种8位机，这是唯一的一条16位立即数传送指令。功能：将一个16位的立即数送入DPTR中去。其中高8位送入DPH，低8位送入DPL。

## 三、堆栈操作（2条）

**PUSH direct** ;  $SP \leftarrow SP + 1$ ,  $(SP) \leftarrow (direct)$

**POP direct** ;  $(direct) \leftarrow (SP)$ ,  $SP \leftarrow SP - 1$

第一条为压入指令，就是将**direct**中的内容送入堆栈中，  
第二条为弹出指令，就是将堆栈中的内容送回到**direct**中。

## 四、读程序存储器指令(查表指令)(2条)

**MOVC A, @A+DPTR ; A $\leftarrow$ (A+DPTR)**

**MOVC A, @A+PC ; PC $\leftarrow$ PC+1, A $\leftarrow$ (A+PC)**

变址寻址，将程序存储器ROM中的数送入A中。也称为查表指令，常用此指令来查一个已经在ROM中做好的表格数据。

## 五、累加器A与片外RAM之间的数据传送类指令(4条)

**MOVX A, @Ri ; A $\leftarrow$ (Ri)**

**MOVX @Ri, A; (Ri) $\leftarrow$  A**

**MOVX A, @DPTR; A $\leftarrow$ (DPTR)**

**MOVX @DPTR, A; (DPTR)  $\leftarrow$  A**

## 六、交换指令（5条）

XCH	A, Rn	; A $\leftrightarrow$ Rn
XCH	A, direct	; A $\leftrightarrow$ (direct)
XCH	A, @Ri	; A $\leftrightarrow$ (Ri)
XCHD	A, @Ri	; A. 3 $\sim$ A. 0 $\leftrightarrow$ (Ri). 3 $\sim$ (Ri). 0
SWAP	A	; A. 3 $\sim$ A. 0 $\leftrightarrow$ A. 7 $\sim$ A. 4

XCH是字节交换指令，将累加器A的内容与源操作数所指的数据相互交换。

XCHD是半字节交换指令，将累加器A中低4位与Ri间接寻址单元内容的低4位相互交换，而各自的高4位内容不变。

SWAP指令是将A的高低两半字节相互交换。

### 3. 3. 2 算术运算类指令 (24条)

#### Arithmetic Operations

主要对8位无符号数进行运算；也可用于带符号数运算。包括：加、减、乘、除、加1、减1运算指令

算术运算指令会影响程序状态字寄存器PSW有关位，包括进位位CY、溢出位OV、半进位位AC和奇偶标志位P。

# 一、加法类指令

## 1. 不带进位位的加法指令（4条）

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

用途：将A中的值与源操作数所指内容相加，最终结果保存在A中。

## 2. 带进位位的加法指令 (4条)

**ADDC A, Rn** ;  $A + Rn + CY \rightarrow A$

**ADDC A, direct** ;  $A + (direct) + CY \rightarrow A$

**ADDC A, @Ri** ;  $A + (Ri) + CY \rightarrow A$

**ADDC A, #data** ;  $A + data + CY \rightarrow A$

**用途：**将A中的值和其后面的值以及进位位CY中的值相加，最终结果存在A，常用于多字节数运算中。

**说明：**51单片机是一种8位机，只能做8位的算术运算（运算范围0~255），实际工作时需要进行扩展，将2个8位(即两字节)算术运算合起来，成为一个16位运算，使得可表达数的范围达到0~65535。

### 3. 加1指令 (5条)

INC A ; A+1→A, 影响P标志

INC Rn ; Rn+1→Rn

INC direct ; (direct)+1→(direct)

INC @Ri ; (Ri)+1→(Ri)

INC DPTR ; DPTR+1→DPTR

功能：将操作数所指定单元的内容加1。

### 4. 二-十进制调整指令 (1条)

DA A

在进行BCD码加法运算时，跟在ADD和ADDC指令之后，用来对BCD码加法运算结果进行自动修正。

## 二、减法指令 (8条)

### 1. 带借位的减法指令 (4条)

**SUBB A, Rn** ; A-Rn-CY→A

**SUBB A, direct** ; A-(direct)-CY→A

**SUBB A, @Ri** ; A-(Ri)-CY→A

**SUBB A, #data** ; A-data-CY→A

将A中的值减去源操作数所指内容以及借位位CY中的值，最终结果存在A中。

### 2. 减1指令 (4条)

**DEC A** ; A-1→A, 影响P标志

**DEC Rn** ; Rn-1→Rn

**DEC direct** ; (direct)-1→(direct)

**DEC @Ri** ; (Ri)-1→(Ri)

说明：没有不带借位的减法指令，如果需要做不带借位的减法指令（在做第一次相减时），只要将CY清零即可。

对带符号数，要注意OV标志。OV=1，出错。

## 2. 减1指令（4条）

**DEC A** ; A-1→A, 影响P标志

**DEC Rn** ; Rn-1→Rn

**DEC direct** ; (direct)-1→(direct)

**DEC @Ri** ; (Ri)-1→(Ri)

与加1指令类似，少DPTR。

### 三、乘法指令（1条）

**MUL AB** ;  $A \times B \rightarrow BA$

此指令的功能是将A和B中的两个8位无符号数相乘，两数相乘结果一般比较大，因此最终结果用1个16位数来表达，其中高8位放在B中，低8位放在A中。在乘积大于FFH时，OV置1，否则OV为0；而CY总是0。

例： A=4EH, B=5DH, 执行指令MUL AB后，乘积是1C56H，所以在B中放的是1CH，而A中放的则是56H。

此时， OV=1

CY=0

P=0 (A中1的个数为偶数)

## 四、除法指令（1条）

**DIV AB** ; A÷B的商→A, 余数→B

此指令的功能是将A中的8位无符号数除B中的8位无符号数 (A/B)。除了之后，商放在A中，余数放在B中。

**CY**和**OV**都是0。如果在做除法前B中的值是00H，即除数为0，则OV=1。

如：A=11H, B=04H, 执行指令DIV AB

结果：A=04H, B=01H。

此时， CY=0

OV=0

P=1 (A中1的个数为奇数)

### 3. 3. 3 逻辑运算类指令 (24条)

## Logic Operations

主要用于对2个操作数按位进行逻辑操作，结果送到A或直接寻址单元。

- ❖ **主要操作**

与、或、异或、移位、取反、清零等。

- ❖ **对标志位的影响**

除了目的操作数为累加器A的指令会影响奇偶标志P外，一般不影响标志位。

# 一、逻辑与指令 (6条)

**ANL A, Rn**

;  $A \wedge Rn \rightarrow A$

**ANL A, direct**

;  $A \wedge (\text{direct}) \rightarrow A$

**ANL A, @Ri**

;  $A \wedge (Ri) \rightarrow A$

**ANL A, #data**

;  $A \wedge \text{data} \rightarrow A$

**ANL direct, A**

;  $(\text{direct}) \wedge A \rightarrow (\text{direct})$

**ANL direct, #data**

;  $(\text{direct}) \wedge \text{data} \rightarrow (\text{direct})$

影响P标志

例：71H 和 56H 相与：

0111	0001	(71H)
Λ )	0101	0110
<hr/>		(56H)
0101	0000	即50H

有0则0，全1才1  
常用于屏蔽某些位

## 二、逻辑或指令 (6条)

<b>ORL A, Rn</b>	<b>； A <math>\vee</math> Rn <math>\rightarrow</math> A</b>	影响P标志
<b>ORL A, direct</b>	<b>； A <math>\vee</math> (direct) <math>\rightarrow</math> A</b>	
<b>ORL A, @Ri</b>	<b>； A <math>\vee</math> (Ri) <math>\rightarrow</math> A</b>	
<b>ORL A, #data</b>	<b>； A <math>\vee</math> data <math>\rightarrow</math> A</b>	
<b>ORL direct, A</b>	<b>； (direct) <math>\vee</math> A <math>\rightarrow</math> (direct)</b>	
<b>ORL direct, #data</b>	<b>； (direct) <math>\vee</math> data <math>\rightarrow</math> (direct)</b>	

例：71H 和 56H 相或：

$$\begin{array}{r} 0111\ 0001 \quad (71H) \\ \vee) \quad 0101\ 0110 \quad (56H) \\ \hline 0111\ 0111 \text{ 即} 77H \end{array}$$

有1则1，全0才0  
常用于某些位的置位

### 三、逻辑异或指令 (6条)

#### eXclusive-oR Logic Instruction

**XRL A, Rn**

;  $A \oplus Rn \rightarrow A$

**XRL A, direct**

;  $A \oplus (direct) \rightarrow A$

**XRL A, @Ri**

;  $A \oplus (Ri) \rightarrow A$

**XRL A, #data**

;  $A \oplus data \rightarrow A$

**XRL direct, A**

;  $(direct) \oplus A \rightarrow (direct)$

**XRL direct, #data**

;  $(direct) \oplus data \rightarrow (direct)$

例：A5H 和 F0H 相异或：

$$\begin{array}{r} 1010 \ 0101 \ (\text{A5H}) \\ \oplus) \ 1111 \ 0000 \ (\text{F0H}) \\ \hline 0101 \ 0101 \ \text{即}55\text{H} \end{array}$$

相同为0，不同为1  
实现某些位的取反

## 四、累加器A清0指令 (1条)

CLR A ; 0→A

## 五、累加器A取反指令 (1条)

Complement Logic Operation

CPL A ;  $\bar{A} \rightarrow A$  将累加器A的内容按位取反

例：若A=50H，执行CPL A

0101 1100

结果：A=A3H

1010 0011

# 六、累加器A循环移位指令 (4条)

## Rotate Logic instruction

**RL A**



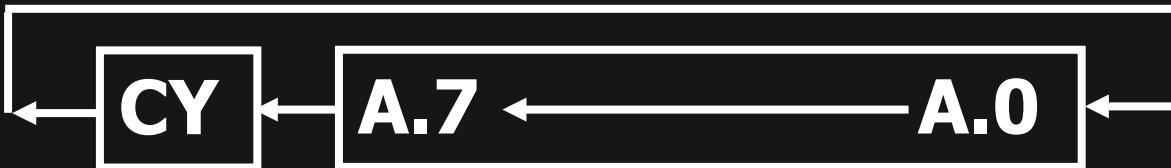
循环左移一位

**RR A**



循环右移一位

**RLC A**



**RRC A**



后两条指令，影响奇偶标志P和进位CY。

例：

若A=5CH, CY=1, 执行RLC A后,  
结果：A=B9H, CY=0, P=1



- ❖ 对RLC、RRC指令，在CY=0时
  - RLC相当于乘以2
  - RRC相当于除以2

## 综合举例：

把累加器A中的低4位状态，通过P1口的高4位输出，P1口的低4位状态不变。

```
ANL A, #0FH          ; 屏蔽A的高4位
SWAP A
ANL P1, #0FH          ; 清P1口高4位
ORL P1, A             ; P1口高4位输出A的低4位；
```

### 3. 3. 4 控制转移类指令（17条）

## Branching Instruction

程序控制指令(不包括位操作类的转移指令)。  
此类指令一般不会影响程序状态字PSW。

包括以下类型：

- 无条件转移和条件转移
- 相对转移和绝对转移
- 长转移和短转移
- 调用与返回指令

# 一、转移类指令

通过改变程序计数器PC的内容，来改变正在执行的指令顺序。

## 1. 无条件转移类指令（4条）

- 长转移类指令： L**JMP** **addr16**
- 短转移类指令： A**JMP** **addr11**
- 相对转移指令： S**JMP** **rel**
- 间接转移指令： **JMP** **@A+DPTR**

（1）上面的前三条指令，统统理解成： PC值改变，即跳转到一个标号处。

- ❖ 跳转的范围不同。

	转移范围：
■ 短转移类指令： AJMP   addr11	2KB
■ 长转移类指令： LJMP   addr16	64KB
■ 相对转移指令： SJMP   rel	-128~+127

- ❖ 指令构成不同。

AJMP、LJMP后跟的是绝对地址，  
而SJMP后跟的是相对地址。

- ❖ 指令长度不同。
- ❖ 原则上，所有用SJMP或AJMP的地方都可以用LJMP来替代。

```
org 0010H
AJMP START
LJMP START
SJMP START
ORG 0020h
```

```
Start:    inc    A
END
```

短转移指令，双字节  
长转移指令，三字节  
相对转移指令，双字节

0010H	0120	AJMP	0020H	; AJMP START
0012H	020020	LJMP	0020H	; LJMP START
0015H	8009	SJMP	0020H	; SJMP START
0017H	FF	MOV	R7, A	
0018H	FF	MOV	R7, A	
0019H	FF	MOV	R7, A	
001AH	FF	MOV	R7, A	
001BH	FF	MOV	R7, A	
001CH	FF	MOV	R7, A	
001DH	FF	MOV	R7, A	
001EH	FF	MOV	R7, A	
001FH	FF	MOV	R7, A	
0020H	04	INC	A	; inc A

## (2) 第四条指令与前三条指令相比有所不同

■ 间接转移指令：JMP @A+DPT<sub>R</sub> 变址寻址转移指令

这条指令跳转到什么地方去呢？

不由标号简单地决定。

转移地址由A+DPT<sub>R</sub>形成，并直接送入PC。

指令对A、DPT<sub>R</sub>和标志位均无影响。

本指令可代替判别跳转指令，又称为散转指令，  
多用于多分支程序结构中。

例： MOV DPTR, #TAB ; 将TAB代表的地址送入DPTR  
          JMP @A+DPTR ; 跳转  
TAB: AJMP ROUT0 ; 跳转ROUT0开始的程序段  
TAB+2: AJMP ROUT1 ; 跳转ROUT1开始的程序段  
TAB+4: AJMP ROUT2 ; 跳转ROUT2开始的程序段  
TAB+6: AJMP ROUT3 ; 跳转ROUT3开始的程序段

...

ROUT0:

...

ROUT1:

...

ROUT2:

...

ROUT3:

执行该段程序后，程序将根据A中的内容转移到不同的程序段去执行——散转。

A=0, 转ROUT0

A=2, 转ROUT1

A=4, 转ROUT2

A=6, 转ROUT3

## 2. 条件转移指令 (10条)

条件转移指令是指在满足一定条件时进行相对转移，否则程序继续执行本指令的下一条指令。

### 1) 判0转移指令 (2条)

**JZ rel** ; 如果A=0, 则转移, 否则顺序执行。

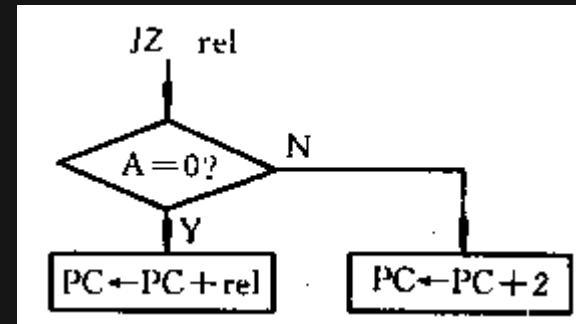
**JNZ rel** ; 如果A $\neq$ 0, 就转移。

转移到相对于当前PC值的8位偏移量的地址去。即：

**新的PC值=当前PC+偏移量rel**

在编写汇编语言源程序时, 可以直接写成：

**JZ 标号** ; 即转移到标号处。



例:

```
MOV A, R0
JZ  L1
MOV R1, #00H
AJMP L2
L1: MOV R1, #0FFH
L2: END
```

执行上面这段程序: 如果R0=0, 结果R1=0FFH。而如果R0≠0, 则结果是R1=00H。

把JZ改成JNZ, 看看程序执行的结果是什么?

如果R0=0, 结果R1=00H。如果R0 ≠0, 结果是R1中的值为0FFH。

## 2) 判CY转移指令 (2条)

**JC rel** ; 如果CY=1, 则转移, 否则顺序执行。  
**JNC rel** ; 如果CY≠1, 就转移。

### 3. 比较不等转移指令 (4条)

CJNE A, #data, rel

CJNE A, direct, rel

CJNE Rn, #data, rel

CJNE @Ri, #data, rel

此类指令的功能是将两个操作数比较，如果两者相等，就顺序执行，如果不相等，就转移。

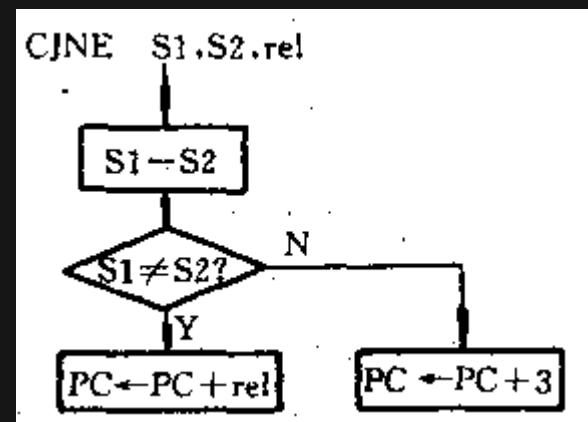
使用时，可以将rel理解成标号，即：

CJNE A, #data, 标号

CJNE A, direct, 标号

CJNE Rn, #data, 标号

CJNE @Ri, #data, 标号



利用这些指令，可以判断两数是否相等。  
但有时还想得知两数比较之后哪个大，哪个小，  
也可以这样进行判断：

**CJNE A, direct, rel ; PC+3 → PC**

；若 **A ≠ (direct)**，则 **PC+rel → PC**

；若 **A < (direct)**，则 **1 → CY**

如果两数不相等，则CPU还会用**CY**（借位位）  
来反映哪个数大，哪个数小。

如果前面的数大，则 **CY=0**，否则 **CY=1**。

因此在程序转移后再次利用CY就可判断出哪个  
数大，哪个数小了。

## 举例：

```
MOV A, R0
CJNE A, #10H, L1 ; 若R0≠10H, 则转移到L1
MOV R1, #0 ; 若R0=10H, 则不转移, R1=00H;
AJMP L3
L1:JC L2 ; 若CY=1, 即R0<10H, 则转移到L2
MOV R1, #0AAH ; 否则CY=0, 即R0>10H
AJMP L3
L2:MOV R1, #0FFH
L3:SJMP L3
```

因此最终结果是：

如果R0=10H, 则R1=00H;

如果R0>10H, 则R1=0AAH;

如果R0<10H, 则R1=0FFH。

## 4. 循环转移指令(减1不为0转移指令) (2条)

DJNZ Rn, rel

DJNZ direct, rel

DJNZ指令的执行过程是这样的：

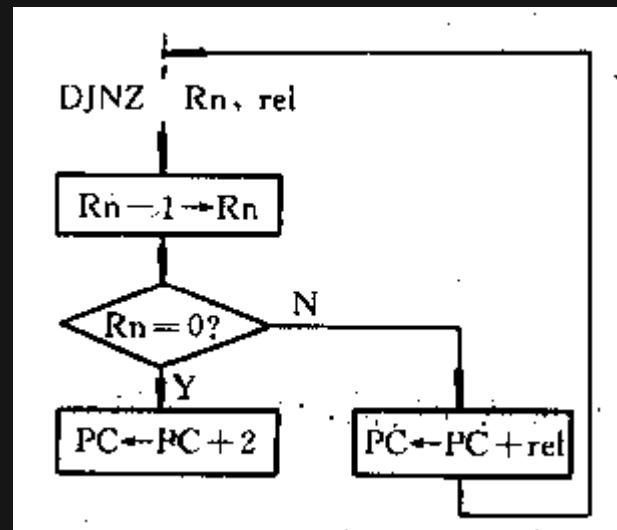
它将第一个参数中的值减1，然后看这个值是否等于0，如果等于0，就往下执行，如果不等于0，就转移到第二个参数所指定的地方去。

例：

LOOP: ...

...

DJNZ 10H, LOOP





在使用DJNZ指令前，  
一定要将循环次数  
赋值给计数器！

例：

MOV 23H, #0AH ; 设循环次数

CLR A ; A清0

LOOP: ADD A, 23H

DJNZ 23H, LOOP

SJMP \$

上述程序段的执行过程是：

将23H单元中的数连续相加，存至A中，每加一次，  
23H单元中的数值减1，直至减到0。

## 二、调用与返回指令 (4条)

### 1. 调用子程序指令 (2条)

**LCALL**   **addr16**      ; 长调用指令(3字节)  
**ACALL**   **addr11**      ; 短调用指令(2字节)

这两条指令都是在主程序中调用子程序，两者的区别：  
对短调用指令，被调用子程序入口地址必须与调用指令的下一条指令的第一字节在相同的2KB存储区之内；对长调用指令，可调用64KB范围内的子程序。

使用时可以用：

**LCALL**   **标号**      ; 标号表示子程序首地址  
**ACALL**   **标号**

来调用子程序。

指令的执行过程是：当前PC压栈，子程序首地址送 PC，实现转移。

## 2. 返回指令 (2条)

在子程序或者中断服务程序的末尾执行

**RET** ; 子程序返回指令

**RETI** ; 中断程序返回指令

两者不能互换使用。

**RET** 指令的执行过程是：堆栈栈顶内容（2字节，调用时保存的当前PC值）弹出给PC，实现返回。

**RETI** 指令除了具有**RET**指令的功能实现程序返回外，还有对中断优先级状态触发器的清零作用。

## 三、空操作指令 (1条)

**NOP**

空操作，就是暂停一个周期，一般用作短时间的延时。

### 3. 3. 5 位操作指令（17条）

MCS-51单片机的硬件结构中，有一个位处理器（又称布尔处理器），它有一套位变量处理的指令集，包括位数据传送、位逻辑运算、位控制程序转移等。

在MCS-51中，有一部分RAM和一部分SFR具有位寻址功能。

**位操作区**：内部RAM的20H-2FH这16个字节单元，即128个位单元（位地址空间为00~7FH）；

**可以位寻址的特殊功能寄存器**：这些SFR的特点是其字节地址均可被8整除，有A累加器，B寄存器、PSW、IP（中断优先级控制寄存器）、IE（中断允许控制寄存器）、SCON（串行口控制寄存器）、TCON（定时器/计数器控制寄存器）、P0-P3（I/O端口锁存器）。

在进行位处理时，CY用作“位累加器”。

# 位地址表达方式

以PSW中位4（RS1）为例

- 直接(位)地址方式：如 D4H；
- 点操作符号方式：如 PSW. 4, (D0H). 4；
- 位名称方式：如 RS1；
- 用户定义名方式：如用伪指令 BIT

SUB. REG BIT RS1

定义后，可用SUB. REG代替RS1

## 一、位传送指令 (2条)

**MOV C, bit** ; bit → C  
**MOV bit, C** ; C → bit

这组指令的功能是实现位累加器 (CY) 和其它位地址之间的数据传送。

例： **MOV C, P1. 0**； 将P1. 0的状态送给C

**MOV P1. 0, C**； 将C中的状态送到P1. 0引脚上去

## 二、状态修改指令 (6条)

### 位清0指令

**CLR C** ; 使CY=0

**CLR bit** ; 使指定位的值等于0

**例：** CLR P1. 0 ; 使P1. 0变为0

### 位置1指令

**SETB C** ; 使CY=1

**SETB bit** ; 使指定位的值等于1

**例：** SETB P1. 0 ; 使P1. 0变为1

均不影响其他标志位！

### 位取反指令

**CPL C** ; 使CY值取反

**CPL bit** ; 使指定的位的值取反

**例：** CPL P1. 0

### 三、位逻辑运算指令 (4条)

#### 位与指令

**ANL C, bit** ; CY与指定位的值相与，结果送CY

**ANL C, /bit** ; 先将指定的位地址中的值取出后取反，再和CY相与，结果送回CY。但注意：指定的位地址中的值本身并不发生变化。

例： **ANL C, /P1. 0**

#### 位或指令

**ORL C, bit**

**ORL C, /bit**

## 四、位条件转移指令 (3条)

也称判位变量转移指令，都是三字节相对转移指令

JB bit, rel

JNB bit, rel

JBC bit, rel

第一条指令：如果指定的 (bit) =1，则转移，否则顺序执行，第二条指令功能相反。

同样理解为： JB bit, 标号

第三条指令是如果指定的 (bit) =1，则转移，并把该位的内容清0，否则顺序执行，并把该位清0。

## 举例：

P3. 2和P3. 3上各接有一只按键，  
要求它们分别按下时 (P3. 2=0或  
P3. 3=0)，分别使P1口为0或FFH。

START: MOV P1, #0FFH

MOV P3, #0FFH

L1: JNB P3. 2, L2

JNB P3. 3, L3

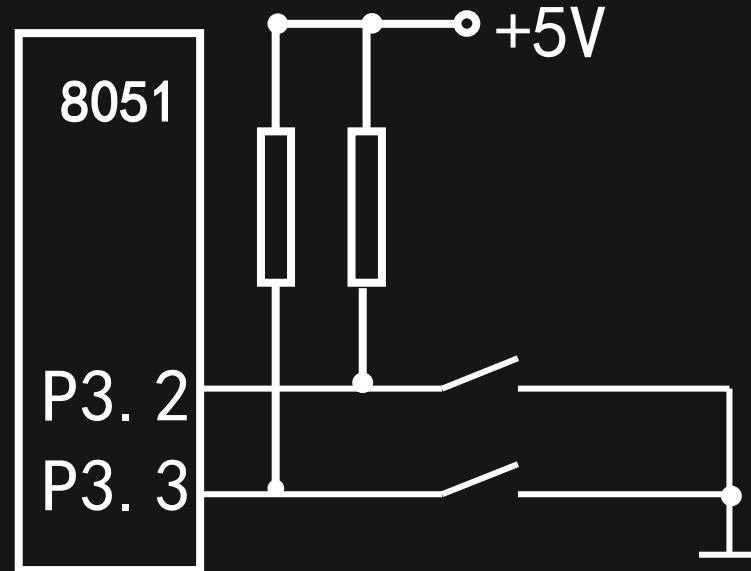
LJMP L1

L2: MOV P1, #00H

LJMP L1

L3: MOV P1, #0FFH

LJMP L1



; P3. 2=0, 转L2

; P3. 3=0, 转L3

; P3. 2=1, P3. 3=1, 等待

; P3. 2=0, 使P1口全为 “0”

; P3. 3=0, 使P1口全为 “1”

# 本章小结

字节数

{ 单字节指令：49条  
双字节指令：45条  
三字节指令：17条

寻址方式7种：立即、直接、寄存器、寄存器间接、  
变址、相对、位寻址

111条指令：

数据传送类（29条）

算术运算类（24条）

逻辑运算与循环类（24条）

控制转移类（17条）

位操作类（17条）

# 第四章 汇编语言程序设计

1. 汇编语言的基本概念
2. 汇编语言的程序设计举例

# § 4.1 汇编语言的基本概念

## 4.1.1 程序设计语言

用于编制计算机程序的语言称为程序设计语言。

按照结构和功能可以分为三种：

1. 机器语言
2. 汇编语言
3. 高级语言

机器语言是用二进制代码0和1来表示指令和数据的最原始的程序设计语言，计算机能**直接识别**和**执行**机器语言程序。

汇编语言是一种**面向机器**的符号语言，指令用助记符表示。其指令是直接访问CPU的寄存器、存储单元和I/O端口，可满足实时控制的要求。

高级语言则是一种**面向过程**且独立于机器的通用语言。简单易学，但生成的目标程序占用存储单元多，执行时间长，不适于实时控制。

在单片机应用中，一般使用**汇编语言**编写程序。

## 4.1.2 编制程序的步骤

1. 任务分析（硬件、软件系统分析）
2. 确定算法（解决问题的方法）
3. 程序总体设计和流程图绘制

关于流程图符号：

开始、结束----圆角矩形



工作任务----矩形



判断分支----菱形



程序流向---- ↓ →

程序连接---- ●



4. 分配内存，确定程序与数据区的存放地址；
5. 根据流程图和指令系统编写源程序；
6. 调试、修改，最终确定程序。

编程技巧——多看多写！

### 4. 1. 3 方法技巧

1. 模块化设计（按功能分：显示、打印、输入、发送等）
2. 尽量采用循环及子程序结构（节省内存）

## 4.1.4 汇编语言的规范

汇编语言源程序由以下两种指令构成：

- 基本指令（指令语句）
- 伪指令（指示性语句）

基本指令：指令系统中的指令，可转换成机器码执行。

伪指令：控制汇编用的特殊指令，这些指令不属于指令系统，不产生机器代码。

汇编语句的格式：

标号： 操作码      操作数      ; 注释

START:    MOV      A, #0F0H      ; A←0F0H

数据表示形式：

二进制 (B) 、十六进制 (H) 、十进制 (D或省略) 、  
ASCII 码 (以单引号标识)

注意：为了区分数字和字符串，规定凡是数字必须以0~9开头，对于十六进制数非0~9开头的数字之前要加上数字0。

## 常用的伪指令

### ❖ **ORG** (Origin)

定位程序或数据存储区的起始地址。

格式：ORG 表达式

如：ORG 1000H

注：表达式必须为16位地址值, 同一源程序中可多次使用，自小至大，不能重叠。

### ❖ **END**

汇编语言程序结束伪指令。

注：一个源程序只有一个END指令！一定放在程序末尾！汇编程序对END后语句不予处理。

# 常用的伪指令

## ❖ EQU (EQUate)

赋值伪指令。把操作数中的地址或数据赋值给字符名称。经过赋值后的字符名称，在整个程序中数值是不变的，可多次使用。

格式：   字符名称   EQU    数值或汇编符号

例：   AA    EQU   30H

          K1    EQU   40H

          MOV A, AA       ; (30H) → A

          MOV A, K1       ; (40H) → A

注意： 使用EQU指令时，必须先赋值。

## 常用的伪指令

### ❖ DB (Define Byte)

定义字节数据。从指定单元开始定义（或存储）若干个字节的数据或ASCII码字符，常用于定义数据常数表。

格式： DB 字节常数或ASCII字符

例： ORG 1000H

DB 34H, 0DEH, 'A', 'B'

DB 0AH, 0BH, 20

汇编结果：

(1000H) =34H (1001H) =DEH (1002H) =41H

(1003H) =42H (1004H) =0AH (1005H) =0BH

(1006H) =14H

# 常用的伪指令

## ❖ DW (Define Word)

定义字数据。从指定单元开始定义（存储）若干个字的数据或ASCII码字符，常用于定义地址表。

格式： DW      字常数或ASCII字符

例：      ORG    2000H  
            DW      1234H, 'B'           ;一个字占两个存储单元  
            DW      0AH, 20                ;不足则自动补一个字节

汇编结果：

(2000H) =12H	(2001H) =34H
(2002H) =00H	(2003H) =42H
(2004H) =00H	(2005H) =0AH
(2006H) =00H	(2007H) =14H

# 常用的伪指令

## 定义空间伪指令 DS

从指定地址开始，保留由表达式指定的若干字节空间作为备用空间。

格式： [标号：] DS 表达式

例 ORG 1000H

DS 0AH

DB 71H, 11H, 11H ; 从100AH开始存放

； 71H、11H、11H

注：表达式一般是数值，即要保留的内存单元个数。

# 常用的伪指令

## 数据地址赋值伪指令 DATA

将表达式指定的数据或代码地址赋予规定的标号

格式: 标号 DATA 表达式

注: 该指令与EQU指令相似, 可先使用后定义,  
放于程序开头、结尾均可。

用DATA定义的标识符在汇编时作为标号记录在符号表中, 可以先使用后定义; 而EQU定义的标识符在汇编时未记录在符号表中, 必须先定义后使用。

## 常用的伪指令

### ❖ BIT

位地址符号指令。

把位地址赋与规定的字符名称。

格式：字符名称 BIT 位地址

例： ABC BIT P1. 1

QQ BIT P3. 2

## § 4.2 汇编语言程序编辑和汇编

1. 编辑（借助TextPad或UltraEdit等编辑软件对源程序进行编辑，生成一个由汇编基本指令和伪指令组成的ASCII码文件，以. ASM扩展名存盘）
2. 汇编（用MASM或TASM编译器进行编译，将指令助记符翻译成相应的机器代码）

如：MOV A, #88H； 机器码74 88

又如： 地址 机器码 源程序

		ORG 1000H
1000H	747F	MOV A, #7FH
1002H	7944	MOV R1, #44H
		END

# 汇编程序的汇编过程

汇编有两种方法：手工汇编、机器汇编。

## 1、手工汇编：

第一次汇编：确定地址，翻译各条机器码，字符标号原样写出；

第二次汇编：标号代真，将字符标号用所计算出的具体地址值或偏移量代换。

手工汇编需要逐条查找指令表，同时要计算相对转移地址的偏移量，汇编过程比较麻烦。

## 2、机器汇编

两次扫描过程。

第一次扫描：检查语法错误，确定符号名称；

建立使用的全部符号名称表；

每一符号名称后跟一个对应值（地址或数）。

第二次扫描：是在第一次扫描基础上，将符号地址转换成地址（代真）；

利用操作码表将助记符转换成相应的目标机器码。

机器汇编使用汇编程序对编辑好的源程序进行汇编，并生成目标文件，较手工汇编简单。

## § 4. 3 程序设计基础与举例

在程序设计中，通常按照程序执行的方式分为四种基本结构的程序：

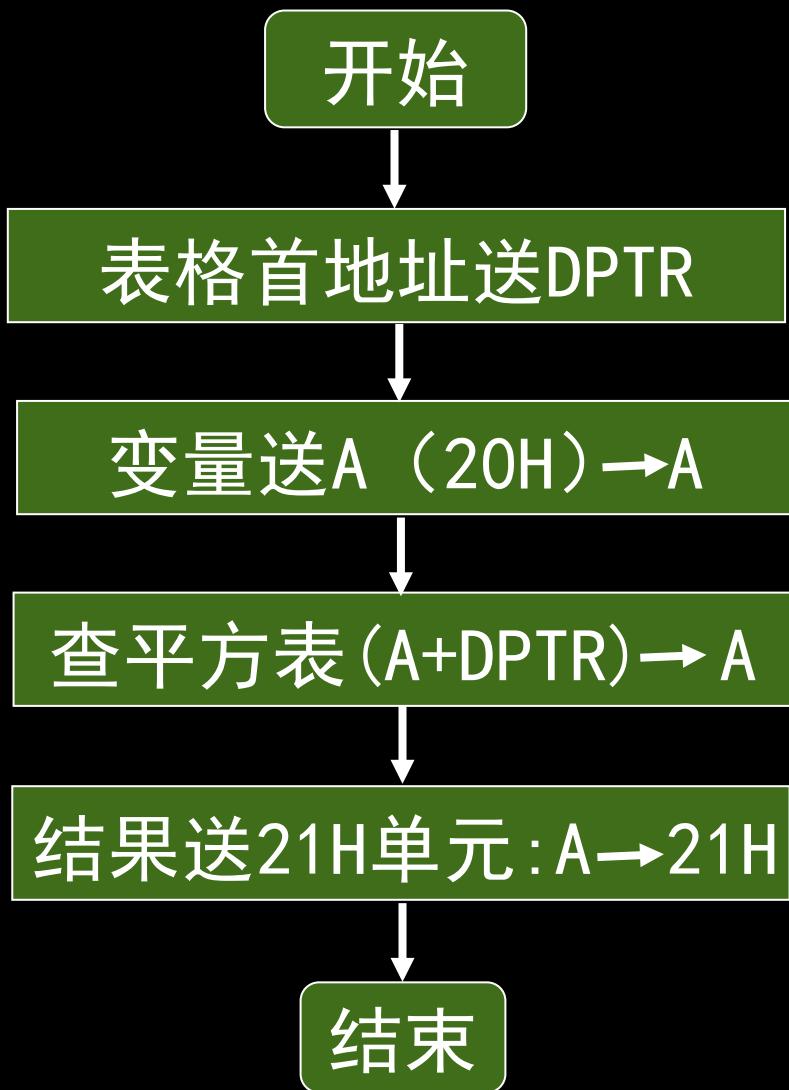
- 顺序程序
- 分支程序
- 循环程序
- 子程序

### 4. 3. 1 顺序结构程序

**例1：**变量存在内部RAM的20H单元中，其取值范围：0~5，用查表法编程求其平方值，结果送到21H单元。

思路：用查表指令即可完成。

查表指令有两条，一条以PC为基址寄存器，  
MOV A, @A+PC；这条指令受PC当前值和累加器  
A的限制，数据表只能放在以PC当前值为起点  
的256个字节范围内。另一条以DPTR为基址寄  
存器，MOV A, @A+DPTR；数据指针为16位寄存  
器，可在64K程序存储器范围内任意设定初值。



```
ORG 1000H
START:MOV DPTR, #TABLE
       MOV A, 20H
       MOVC A, @A+DPTR
       MOV 21H, A
       SJMP $
ORG 2000H
TABLE:DB 0, 1, 4, 9, 16, 25
```

**例2：**将20H单元的压缩BCD码低四位和高四位，分别拆成两个ASCII码存入21H、22H单元。

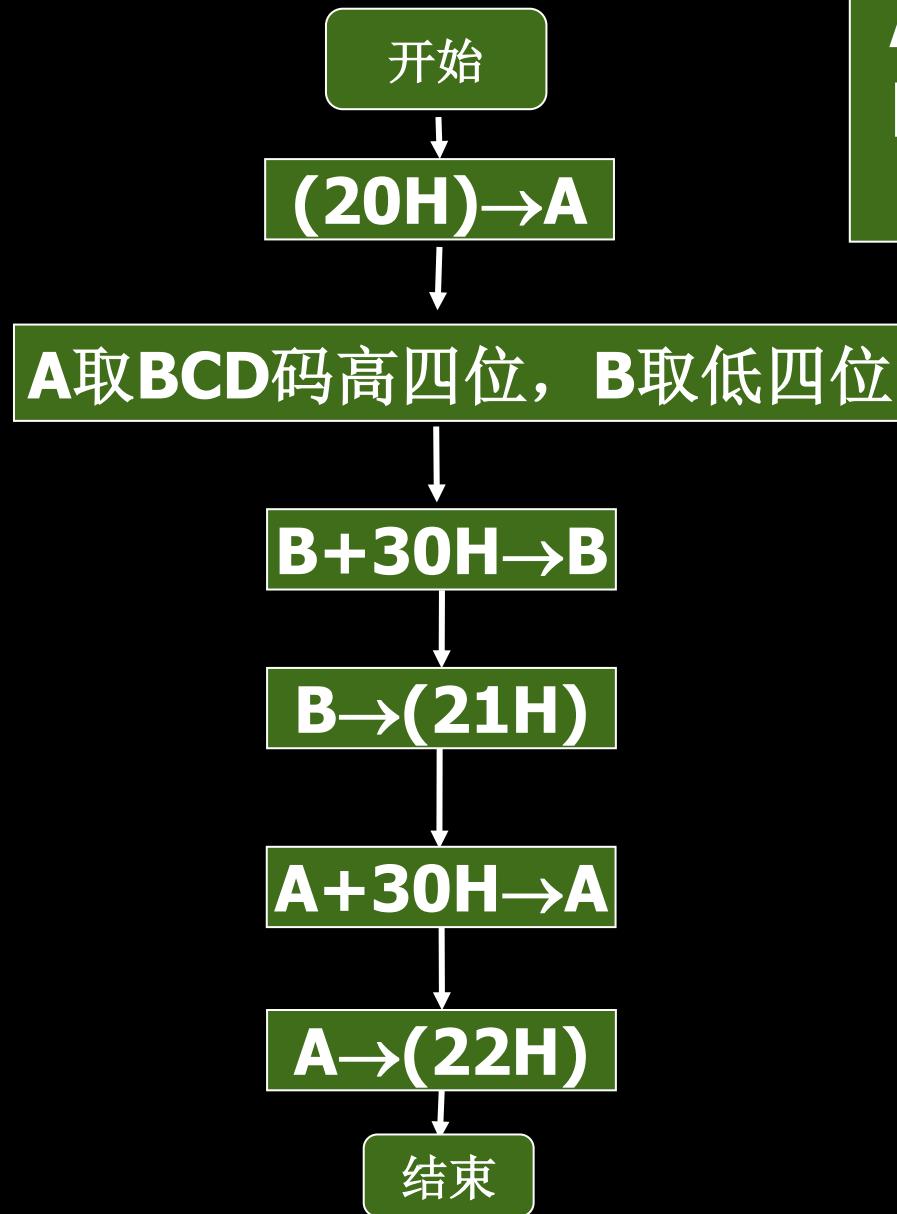
(BCD ASCII 码对照表)

BCD	ASCII
0	30H
1	31H
2	32H
3	33H
...	...
9	39H

内部RAM

	高四位	低四位
20H	2	9
21H	3	9
22H	3	2

## 方法 1



A: 高4位BCD码  
B: 低4位BCD码

周期数

源程序

ORG 2000H

1 MOV A, 20H

2 MOV B, #10

4 DIV AB ;A除以10, 商->A, 余数->B

2 ORL B, #30H ;低4位BCD码转换为ASCII码

2 MOV 21H, B

1 ORL A, #30H ;高4位BCD码转换为ASCII码

1 MOV 22H, A

END

## 方法 2



21H单元内容清0

BCD码送累加器A

半字节交换

低四位转换后送  
21H单元

高四位转换后送  
22H单元

周期数

源程序

半字节交换指令只能用  
工作寄存器R<sub>i</sub>间接寻址

**ORG 2000H**

1	<b>MOV R0, #21H</b>	
1	<b>MOV @R0, #00H</b>	; 清21H单元
1	<b>MOV A, 20H</b>	
1	<b>XCHD A, @R0</b>	; 低4位BCD码送21H单元
2	<b>ORL 21H, #30H</b>	; 低4位BCD码转换为ASCII码
1	<b>SWAP A</b>	; A的高4位、低4位交换
1	<b>ORL A, #30H</b>	; 高4位BCD码转换为ASCII码
1	<b>MOV 22H, A</b>	
9	<b>END</b>	

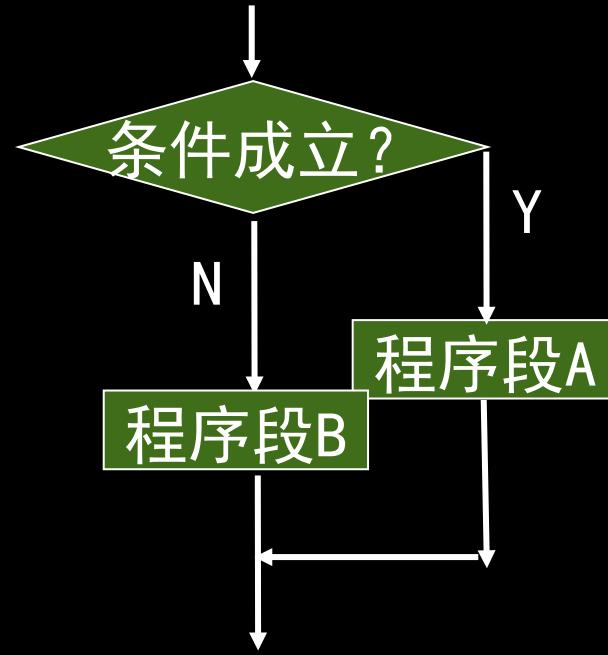
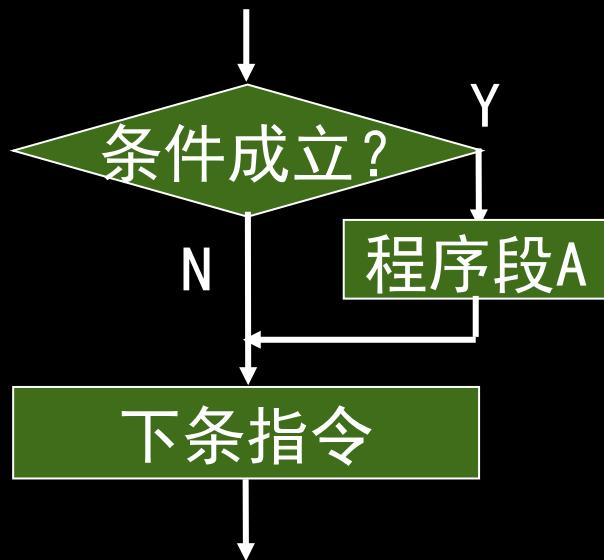
# 方法 3 (思考.....)

### 例3：说明程序功能（书P106 4.1）

```
ADD1:  MOV      A, R2
        ADD      A, #1
        MOV      R2, A
        MOV      A, R3
        ADDC    A, #0
        MOV      R3, A
        RET
```

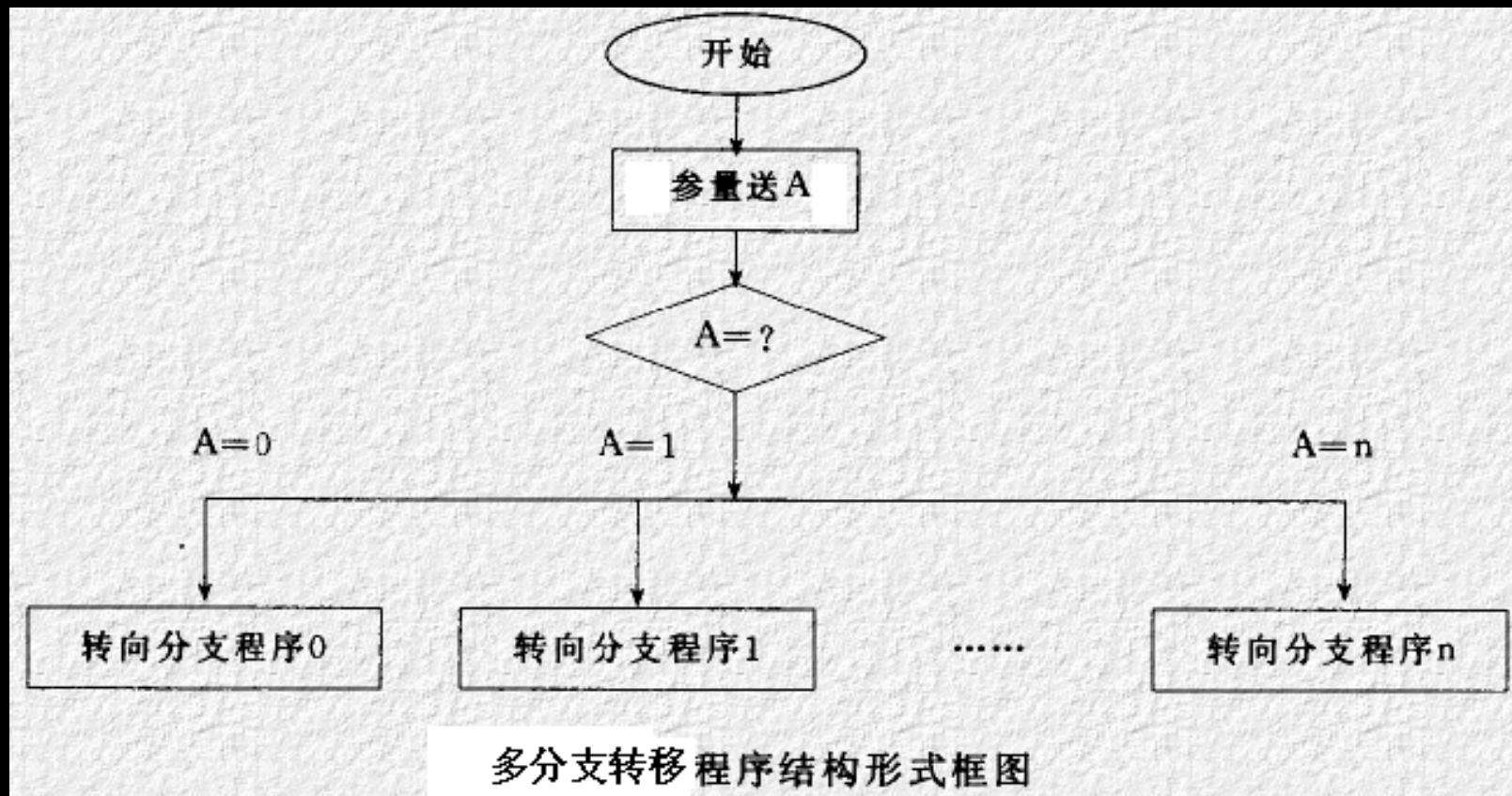
## 4.3.2 分支程序

分支程序可根据要求无条件或有条件地改变程序执行流向。编写分支程序主要在于正确使用转移指令。分支程序有：双分支结构、多分支结构（散转）



## 4.3.2 分支程序

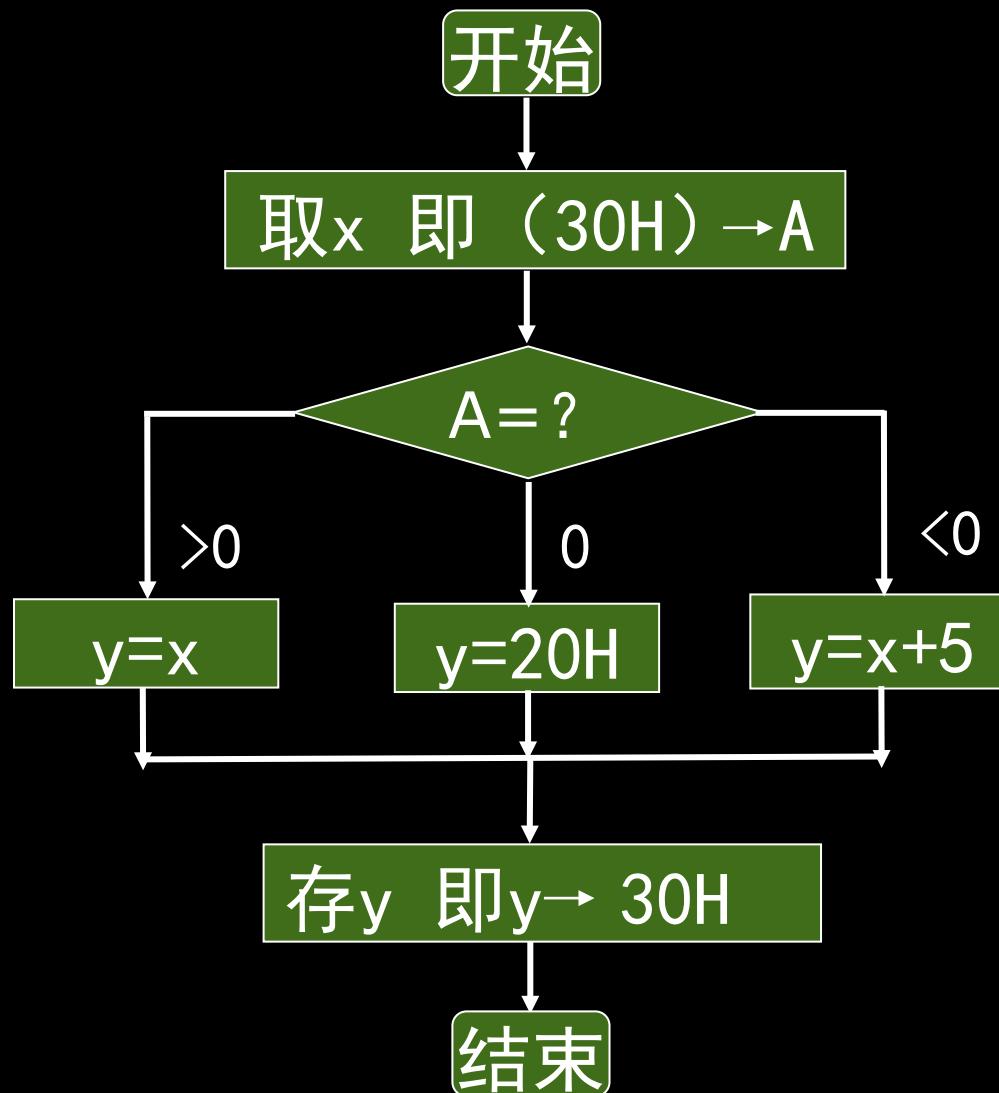
### 多分支结构



**例1：**设变量x以补码形式存放在片内RAM 30H单元中，变量y与x的关系是：

$$y = \begin{cases} x, & x > 0 \\ 20H, & x = 0 \\ x + 5, & x < 0 \end{cases}$$

编程根据x的值，求y值并放回原30H单元。



```
ORG 1000H
START:MOV A, 30H
        JZ NEXT ; x=0, 转移
        ANL A, #80H ; 保留符号位
        JZ ED ; x>0, 转移
        MOV A, #05H ; x<0
        ADD A, 30H
        MOV 30H, A
        SJMP ED
NEXT:MOV 30H, #20H
ED:END
```

注：正数的符号位为**0**，负数的符号位为**1**

# 多向分支程序设计—分支表法

- 分支地址表
  - 各分支程序的首地址组成一个表，每个首地址占连续的两个字节。
- 转移指令表
  - 转移指令组成一个表，各转移指令的目标地址即为分支程序的首地址
- 地址偏移量表
  - 各分支程序的首地址与地址偏移量表的标号之差称为地址偏移量，由地址偏移量组成的一个表

例1：8路分支程序，要求程序根据其运行中所产生的寄存器R7的值，来决定如何进行分支。

为实现8路分支，可以多次使用比较不等转移指令：  
**CJNE R7, #data, rel**，但这样比较次数太多，当分支较大时，执行速度就较慢。

可以用变址寻址的间接转移指令**JMP @A+DPTR**使问题得到解决。首先安排存放一个转移分支入口的地址表，把入口地址表的首地址送到DPTR，再把R7的内容送累加器A。

例2：根据R7的内容，转至对应的分支程序。设R7的内容为0~7，对应的处理程序地址分别为PM0~PM7

——分支地址表法

START: MOV DPTR, #TAB ; 取表首地址

MOV A, R7

PM0: ...

ADD A, R7

; R7 × 2 → A

MOV R3, A

; 暂存A于R3

MOVC A, @A+DPTR

; 取高位地址

PM1: ...

XCH A, R3

; A ←→ R3

:

INC A

; 取低位地址

PM7: ...

MOV DPL, A

MOV DPH, R3

; 转移地址送入DPTR

CLR A

JMP @A+DPTR

TAB: DW PM0 ; 分支地址表

TAB

PM0高位

DW PM1

PM0低位

...

TAB+2

PM1高位

DW PM7

PM1低位

例3：根据R0的值转向7个分支程序。

$R0 < 10$ , 转向SUB0;

$R0 < 20$ , 转向SUB1;

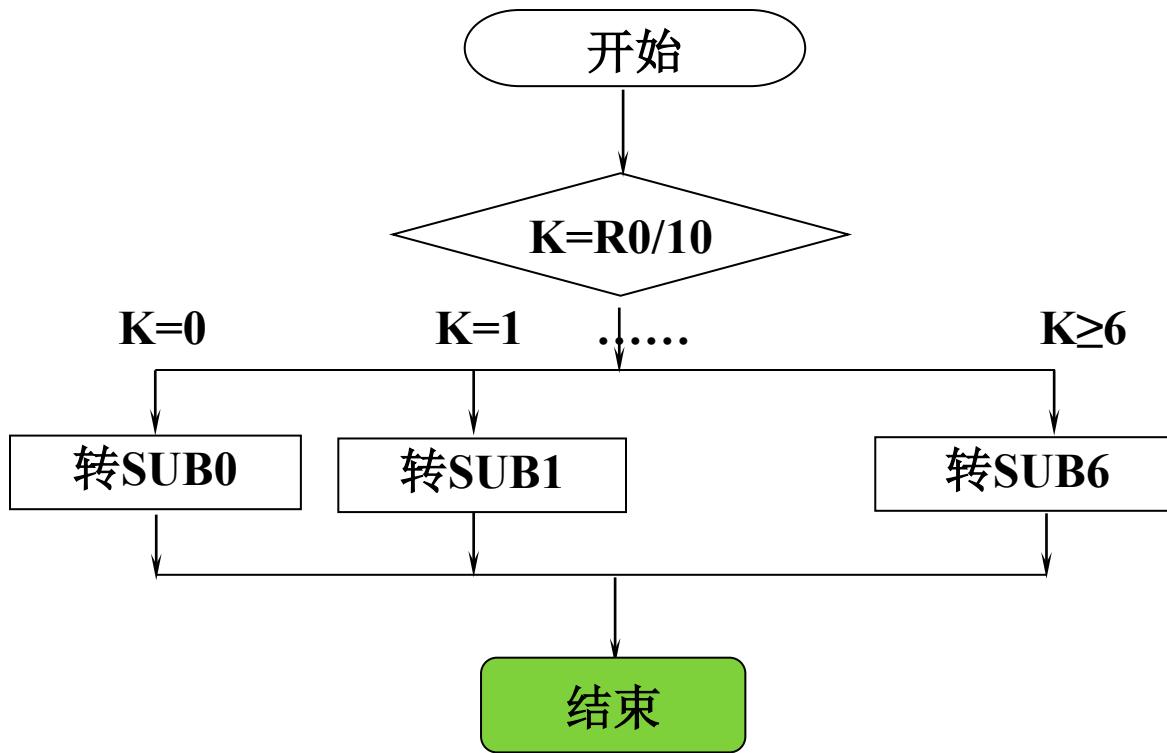
.....

$R0 < 60$ , 转向SUB5;

$R0 \geq 60$ , 转向SUB6;

**转移指令表法：**

利用JMP @A+DPTR 指令直接给PC赋值，使程序实现转移。



多向分支程序流程图

## 参考程序：

```
ORG 2000H
MOV DPTR, #TAB ; 转移指令表首地址
MOV A, R0 ; 取数
MOV B, #10
DIV AB ; A/10, 商在A中
CLR C
RLC A ; A←2×A
JMP @A+DPTR ; PC ← A+DPTR
TAB: AJMP SUB0 ; 转移指令表
AJMP SUB1
.....
AJMP SUB6
```

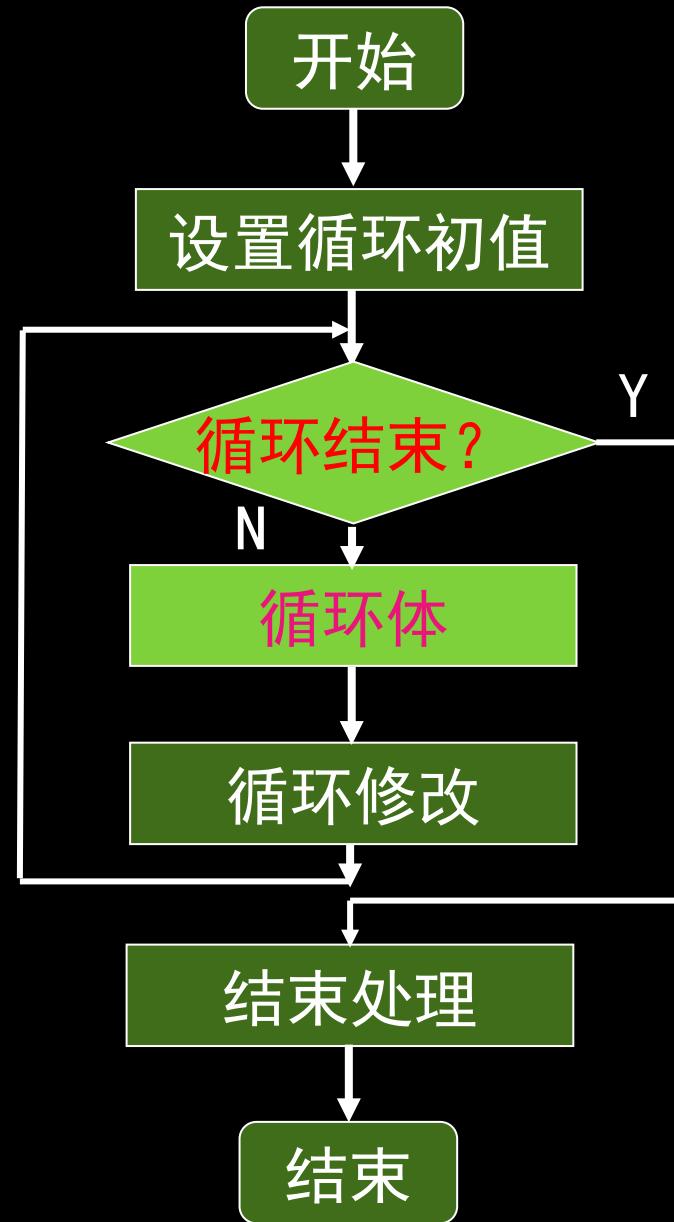
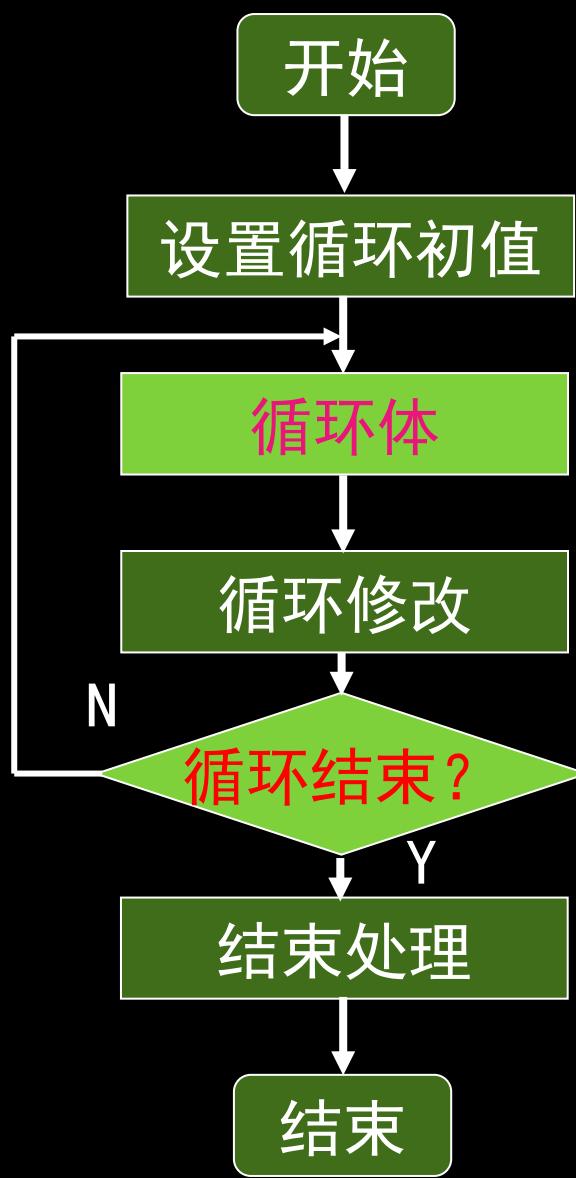
### 4.3.3 循环程序

循环程序一般由四部分构成：

- 初始化部分； 设置循环开始时的状态
- 循环体部分； 要重复执行的程序段
- 循环控制部分； 判断循环是否继续及修改指针等
- 结束部分； 存放结果或做其他处理

一般有两种结构：

- ◆ 先进入处理部分，再控制循环
  - 至少执行一次循环体
- ◆ 先控制循环，再进入处理部分
  - 循环体是否执行，取决于判断结果。



**例1** 将内部RAM中起始地址为data的数据串送到外部RAM中起始地址为buffer的存储区域中，直到发现‘\$’字符，传送停止——循环次数事先不知道，应先判断，后执行。

```
MOV  R0, #data
MOV  DPTR, #buffer
LOOP1:MOV  A, @R0
       CJNE A, #24H, LOOP2      ;判断是否为 $ 字符
       SJMP LOOP3                ;是， 转结束
LOOP2:MOVX @DPTR, A          ;不是， 传送数据
       INC  R0
       INC  DPTR                ;修改地址指针
       SJMP LOOP1                ;传送下一数据
LOOP3:END
```

思考：程序执行后，A=?  
(书P106 4.2)

```
        MOV      R2, #0AH
        CLR      A
LOOP:   ADD      A, R2
        DJNZ    R2, LOOP
        RET
```

## 例2、设8051使用12MHz晶振，试设计延迟100ms的延时程序。

延时程序的延迟时间就是该程序的执行时间，通常采用 MOV 和 DJNZ 两条指令。

$$\text{机器周期 } T = 12 / f_{osc} = 12 / (12 \times 10^6) = 1 \mu s$$

```
ORG 1000H

DELAY: MOV R2, #199 ; CTS = 199 1T = 1 μ s
LOOPS: MOV R0, #250 ; CTR = 250 1T = 1 μ s
LOOPR: DJNZ R0, LOOPR ; 2T = 2 μ s
        DJNZ R2, LOOPS ; 2T = 2 μ s
END
```

影响延时程序的延时时间有两个主要因素：晶振频率和循环次数。晶振频率确定之后，主要就是计算给定的延时循环次数。

```
ORG 1000H
    DELAY: MOV R2, #199 ; 1T = 1 μ s
    LOOPS: MOV R0, #250 ; 1T = 1 μ s
    LOOPR: DJNZ R0, LOOPR ; 2T = 2 μ s
    DJNZ R2, LOOPS ; 2T = 2 μ s
END
```

■ 内循环延时：

$$(2 \times \text{CTR}) T = 500 \mu \text{s} \text{ (假设)}$$

$$\text{则 } \text{CTR} = 250$$

■ 总延时： $T + (1T + 500T + 2T) \times \text{CTS} = 100\text{ms} = 100\ 000 \mu \text{s}$

所以， $\text{CTS} = 198.8$  取 199

实际延时： $[1 + (501 + 2) \times 199] = 100.098\text{ms}$

### 例3：排序程序设计。

若以30H为首地址的内部RAM中有7个连续存放的数据，按由小到大的次序排好后存入这7个单元。

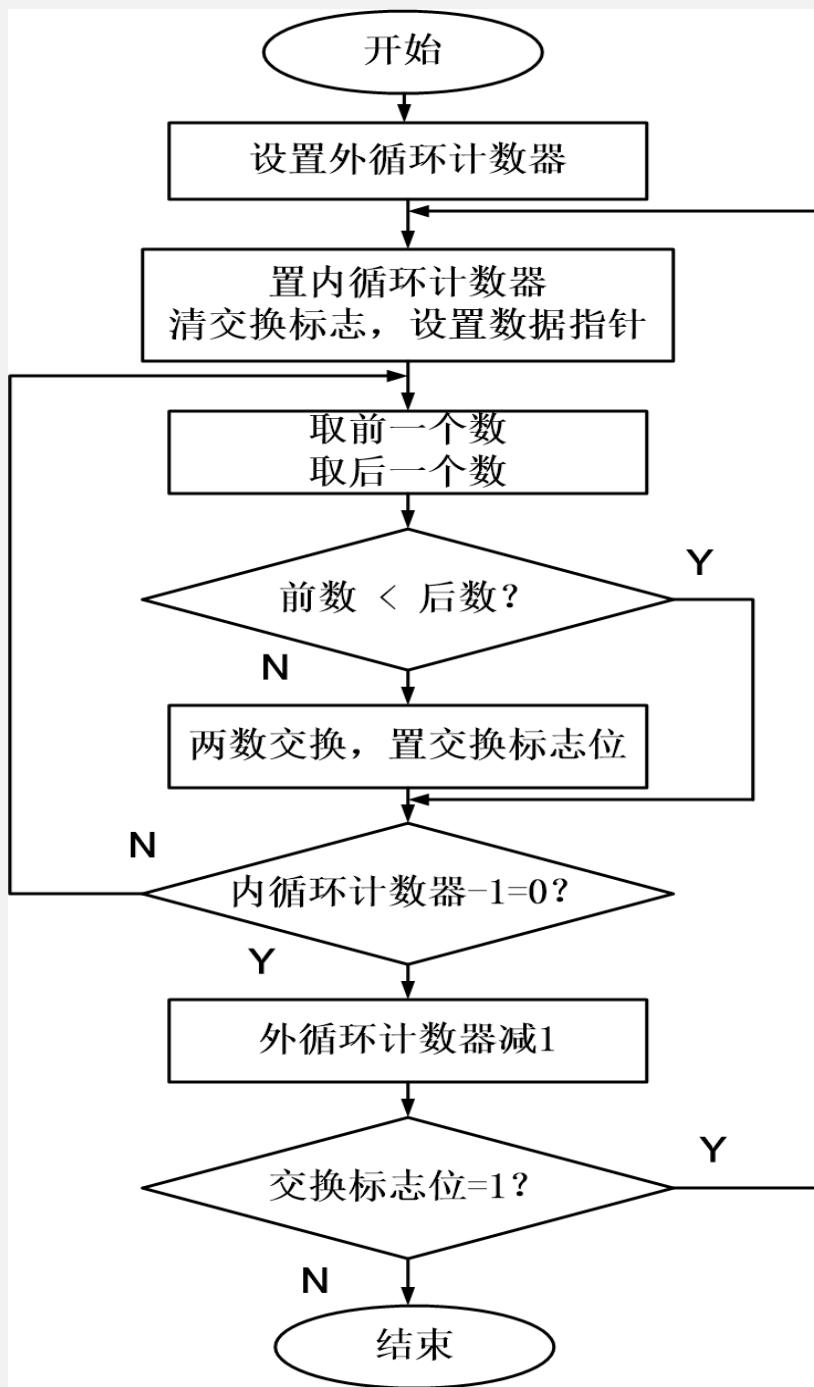
冒泡法排序基本思想：

是一种相邻数互换的排序方法。执行时从前向后依次将相邻两单元的数进行比较，即第一个数与第二个数、第二个数与第三个数……进行比较，如果符合从小到大的顺序则不变动，否则将它们交换位置。如此反复比较和交换，使小数向前移、大数向后移，直到数列排好为止。

表 4-1 数列分选过程

原 数 列	第一次分选结果	第二次分选结果	第三次分选结果	第四次分选结果
0	0	0	0	0
16	3	3	3	3
3	16	16	16	13
94	25	25	13	16
25	36	13	25	25
36	13	36	36	36
13	94	94	94	94

为了加快程序运行速度，编程时要考虑以下两点：一是要比较交换的数每次少一个，即每次冒泡所需进行的比较次数都递减1；二是只要不发生交换就表示已按大小顺序排列好了，可以结束比较。



**R0:数据首地址**  
**F0:交换标志位**  
**R7:外循环初值**  
**R6:内循环初值**

程序的关键在于每两个数进行比较，如果有交换发生，交换标志位要置1。

	ORG	2000H	
	MOV	R7, #06H	; 外循环计数器赋值
LP1:	MOV	R0, #30H	; R0 指向数据首址
	CLR	F0	; 交换标志清 0
	MOV	A, R7	
	MOV	R6, A	; 内循环计数器赋值
LP2:	MOV	A, @R0	; 取前一个数
	MOV	20H, A	; 存前数
	INC	R0	
	MOV	21H, @R0	; 取后一个数
	CLR	C	; CY 清 0
	SUBB	A, @R0	; 前数减后数
	JC	NEXT	; 前数小于后数, 不交换
	MOV	@R0, 20H	
	DEC	R0	
	MOV	@R0, 21H	; 前、后两数交换位置
	INC	R0	; 恢复数据指针
	SETB	F0	; 置互换标志
NEXT:	DJNZ	R6, LP2	; 未查完一遍, 进行下一次比较
	DEC	R7	; 修改内循环次数
	JB	F0, LP1	; 返回, 进行下一轮冒泡
	SJMP	\$	; 结束
	END		

#### 4.3.4 子程序

- 子程序内容：把多次进行的一些相同的计算和操作独立出来
- 指令：LCALL ACALL RET
- 注意事项
  - 子程序应取名，第一条指令应加标号
  - 正确传递参数
  - 注意保护现场和恢复现场
  - 子程序的末尾为RET

## 子程序调用方式:

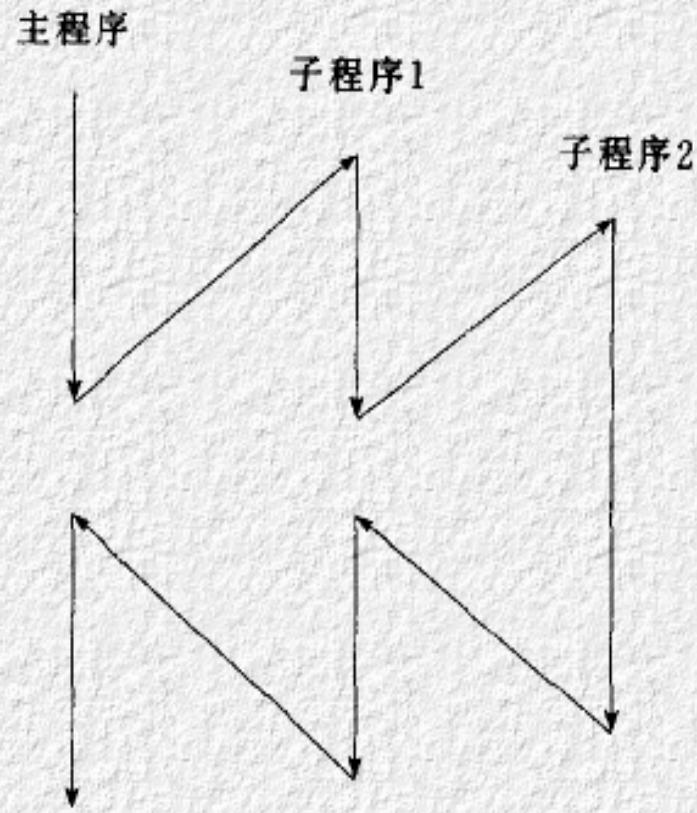
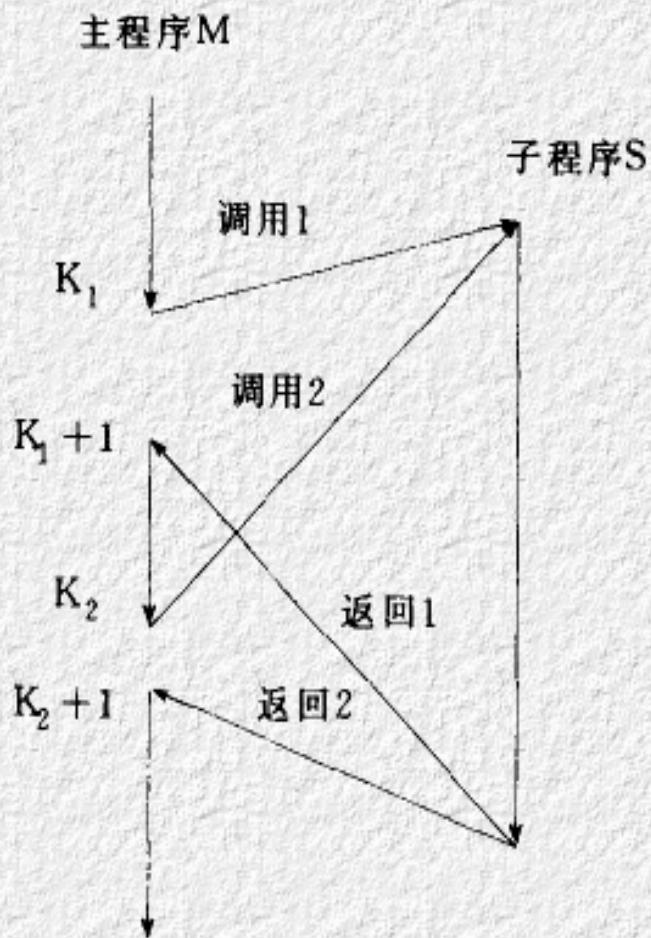


图4-12 子程序嵌套

图4-11 主程序和子程序之间的调用关系

在汇编语言源程序中使用子程序时，一般要注意两个问题——参数传递和现场保护。

参数传递一般可采用以下方法：

- **传递数据。** 将数据通过工作寄存器R0～R7或累加器来传送。即主程序和子程序在交接处，上述寄存器和累加器存储的是同一参数。
- **传送地址。** 数据存放在数据存储器中，参数传递时只通过R0、R1、DPTR传递数据所存放的地址。
- **通过堆栈传递参数。** 在调用之前，先把要传送的参数压入堆栈，进入子程序执行到结尾处RET之后，再将压入堆栈的参数弹出到工作寄存器或者其他内存单元。

## 例 1 单字节二进制数据转换为**BCD**码子程序**SBTOD**

功能：将单字节二进制数转换为三位**BCD**码。

入口：R2中存放要转换的二进制数。

出口：(R0)给出百位**BCD**码的存放地址。(R0)+1给出十位和个位**BCD**码的存放地址，高半字节放十位，低半字节放个位。

占用寄存器：A, B, R0, R2。

SBTOD:	MOV	SP, #60H	
	PUSH	ACC	
	MOV	A, R2	; 取数
	MOV	B, #64H	
	DIV	AB	; 除以100, (A)为百位数
	MOV	@R0, A	; 存入(R0)单元
	MOV	A, #0AH	
	XCH	A, B	; 余数(B)送A
	DIV	A, B	; 除以10, 得十位和个位
	SWAP	A	; 十位数放于高半字节
	ADD	A, B	; 个位数放于低半字节
	INC	R0	
	MOV	@R0, A	; 个位存入(R0)+1单元
	POP	ACC	
	RET		

例2 用程序实现 $c=a^2+b^2$ , (a, b, c内容分别放入MDA、MDB和MDC中, 设a、b小于10)

```
        ORG 1000H
MDA      DATA 20H
MDB      DATA 21H
MDC      DATA 22H
        MOV A,MDA
        ACALL SQR
        MOV R1,A
        MOV A,MDB
        ACALL SQR
        ADD A,R1
        MOV MDC,A
        SJMP S
SQR:    ADD A,#01H      ;地址调整
        MOVC A,@A+PC    ;查平方表
        RET               ;返回
SQRTAB: DB 0,1,4,9,16
        DB 25,36,49,64,81
        END
```

进行查表时, PC的当前值不会恰好是表首地址, 因此需要在查表指令前安排一条加法指令, 对累加器A 中的地址偏移量进行修正, 修正量:

data=数据表首地址-PC当前值

**例3 求两个无符号数据块中的最大值。数据块的首地址分别为60H和70H，每个数据块的第一个字节都存放数据块的长度，结果存入5FH单元。**

本例可采用分别求出两个数据块的最大值，然后比较其大小的方法，求最大值的过程可采用子程序。

**子程序名称： QMAX**

**子程序入口条件： R1中存有数据块首地址**

**出口条件： 最大值在A中**

**下面分别编写主程序和子程序。**

<b>ORG</b>	<b>2000H</b>	
<b>MOV</b>	<b>SP, #2FH</b>	; 设堆栈指针
<b>MOV</b>	<b>R1, #60H</b>	; 取第一数据块首地址送R1中
<b>ACALL</b>	<b>QMAX</b>	; 第一次调用求最大值子程序
<b>MOV</b>	<b>40H, A</b>	; 第一个数据块的最大值暂存40H
<b>MOV</b>	<b>R1, #70H</b>	; 取第二数据块首地址送R1中
<b>ACALL</b>	<b>QMAX</b>	; 第二次调用求最大值子程序
<b>CJNE</b>	<b>A, 40H, NEXT</b>	; 两个最大值进行比较
<b>NEXT:</b>	<b>JNC LP</b>	; A大, 则转LP
	<b>MOV A, 40H</b>	; A小, 则把40H中内容送入A
<b>LP:</b>	<b>MOV 5FH, A</b>	
	<b>SJMP \$</b>	

# 子程序：

	<b>ORG</b>	<b>2200H</b>	
<b>QMAX:</b>	<b>MOV</b>	<b>A, @R1</b>	; 取数据块长度
	<b>MOV</b>	<b>R2, A</b>	; R2做计数器
	<b>CLR</b>	<b>A</b>	; A清零，准备做比较
<b>LP1:</b>	<b>INC</b>	<b>R1</b>	; 指向下一个数据地址
	<b>CLR</b>	<b>C</b>	; 准备做减法
	<b>SUBB</b>	<b>A, @R1</b>	; 用减法做比较
	<b>JNC</b>	<b>LP3</b>	; 若A大，则转LP3
	<b>MOV</b>	<b>A, @R1</b>	; A小，则将大数送A中
	<b>SJMP</b>	<b>LP4</b>	; 无条件转LP4
<b>LP3:</b>	<b>ADD</b>	<b>A, @R1</b>	; 恢复A中值
<b>LP4:</b>	<b>DJNZ</b>	<b>R2, LP1</b>	; 计数器减1，不为零，转继续比较
	<b>RET</b>		; 比较完，子程序返回

# 本章小结

- 了解三种程序设计语言和程序设计的步骤
- 掌握常用伪指令
- 掌握四种基本程序结构
  - 顺序程序
  - 分支程序
  - 循环程序
  - 子程序

# 第五章 存储器

- ◆ 半导体存储器的分类
- ◆ CPU与存储器的连接
- ◆ MCS-51存储器的扩展

# 5.1 存储器系统基本知识

## 5.1.1 半导体存储器的分类

半导体存储器按照使用功能分为两大类

- **只读存储器ROM (Read Only Memory)**  
用于存放固定程序和常数，里面存储的信息只能读取不能写入，掉电后信息不会丢失。
- **随机存取存储器RAM (Random Access Memory)**  
用来存放运行程序、数据和中间结果，在使用过程中可以通过程序随机地对任意存储单元进行写入或者读出的操作。

## 1. 只读存储器（ROM）按照信息写入方式不同分类

- (1) **掩膜ROM** 生产厂家在芯片制造的时候用掩膜工艺写入。
- (2) **可一次性编程ROM（PROM）** 用户根据自己的需要采用一定的设备把程序和数据写入，一旦写入不能再更改。
- (3) **紫外线擦除可改写ROM（EPROM）** 出厂时未编程，用户可写入，如需修改则用紫外光照射芯片上的石英窗口20分钟左右，即可擦除原有信息再重新写入。对已编程的EPROM，一般以遮光纸遮挡石英窗口，以防光线照射。EPROM可多次擦除和写入，适用于进行研制和开发工作。
- (4) **电擦除可改写ROM（EEPROM或E<sup>2</sup>PROM）** 如需改写某个存储单元的信息时，只要将电流通入该存储单元，就能以字节为单位进行改写。
- (5) **快擦写ROM（flash ROM）** 属于E<sup>2</sup>PROM类型，但其存取速度很快，容量也很大。

## 2. 随机存取存储器 (RAM) 按照制造工艺分类

- (1) 双极型RAM 存取速度高, 但功耗大, 集成度较低。
- (2) 金属氧化物 (MOS) 型RAM ---微型机主要采用
  - ①动态RAM (DRAM) 存储单元以电容为基础, 电路比较简单, 集成度高。但存储信息易丢失。需定期刷新, 适合于大存储容量的微型机。
  - ②静态RAM (SRAM) 存储电路以双稳态触发器为基础, 状态稳定, 不需定时刷新, 但集成度低, 适合于存储容量小的微型机。
  - ③非易失性RAM (NVRAM) 是一种掉电自保护RAM, 由静态RAM (SRAM) 和E<sup>2</sup>PROM构成的存储器。掉电瞬间, 可将SRAM中的信息自动地保存在E<sup>2</sup>PROM中。这种存储器容量较小, 多用于存储重要信息和掉电保护。

## 5.1.2 存储器的主要性能指标

### 1. 存储容量

存储器芯片容量=存储单元数×数据线位数= $2^n \times m$

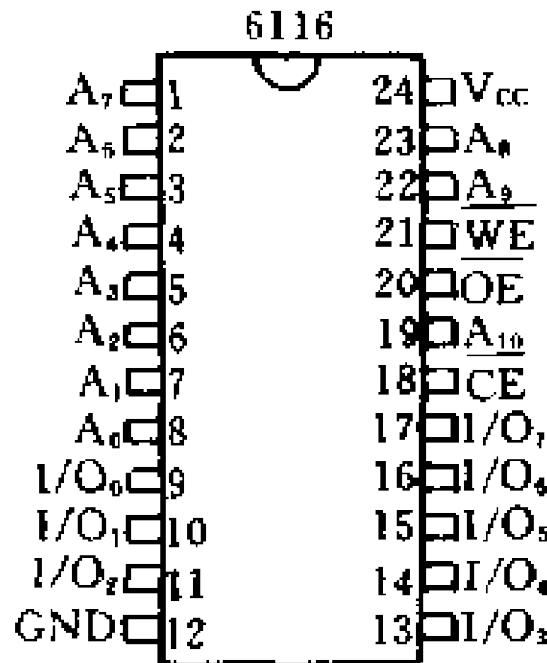
其中n-地址线根数， m-数据线位数

### 2. 存取速度

### 3. 可靠性

### 4. 功耗

## 典型静态RAM芯片



(a)

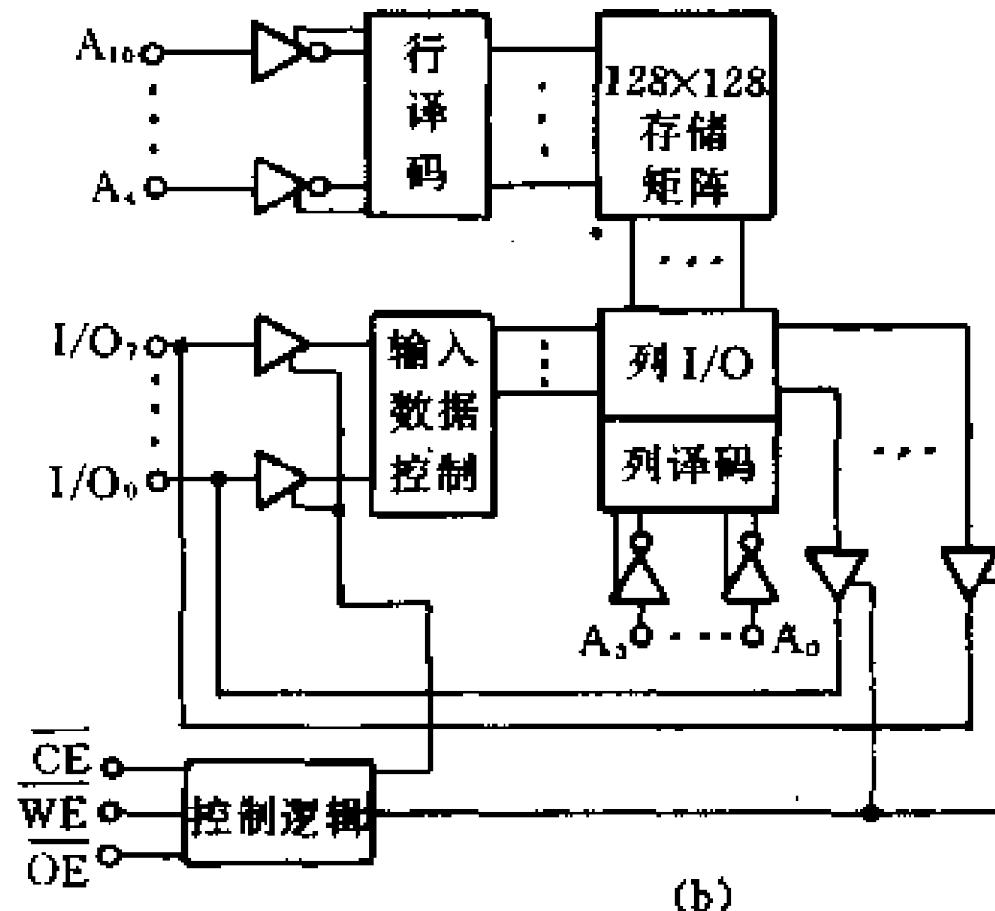
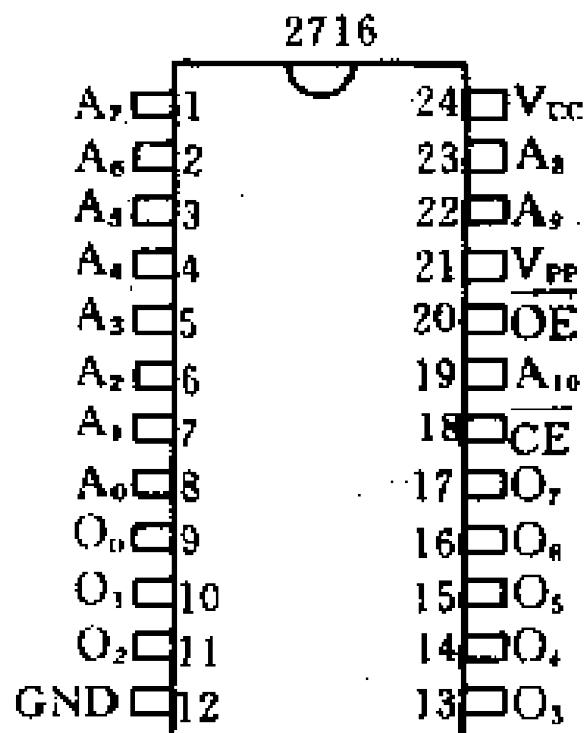
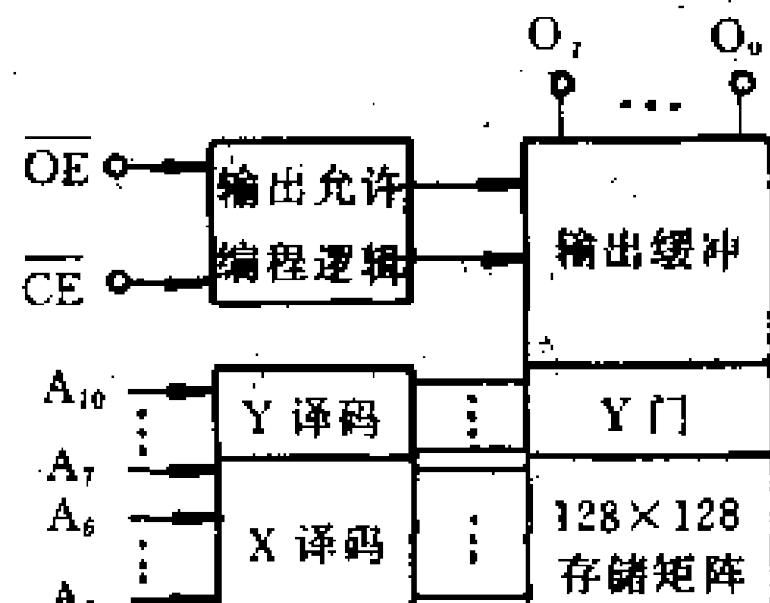


图 5.2.2 6116 芯片的引脚图和功能框图  
(a) 引脚图; (b) 功能框图

# 典型EPROM芯片



(a)



(b)

图 5.3.2 2716 芯片的引脚图和内部结构框图

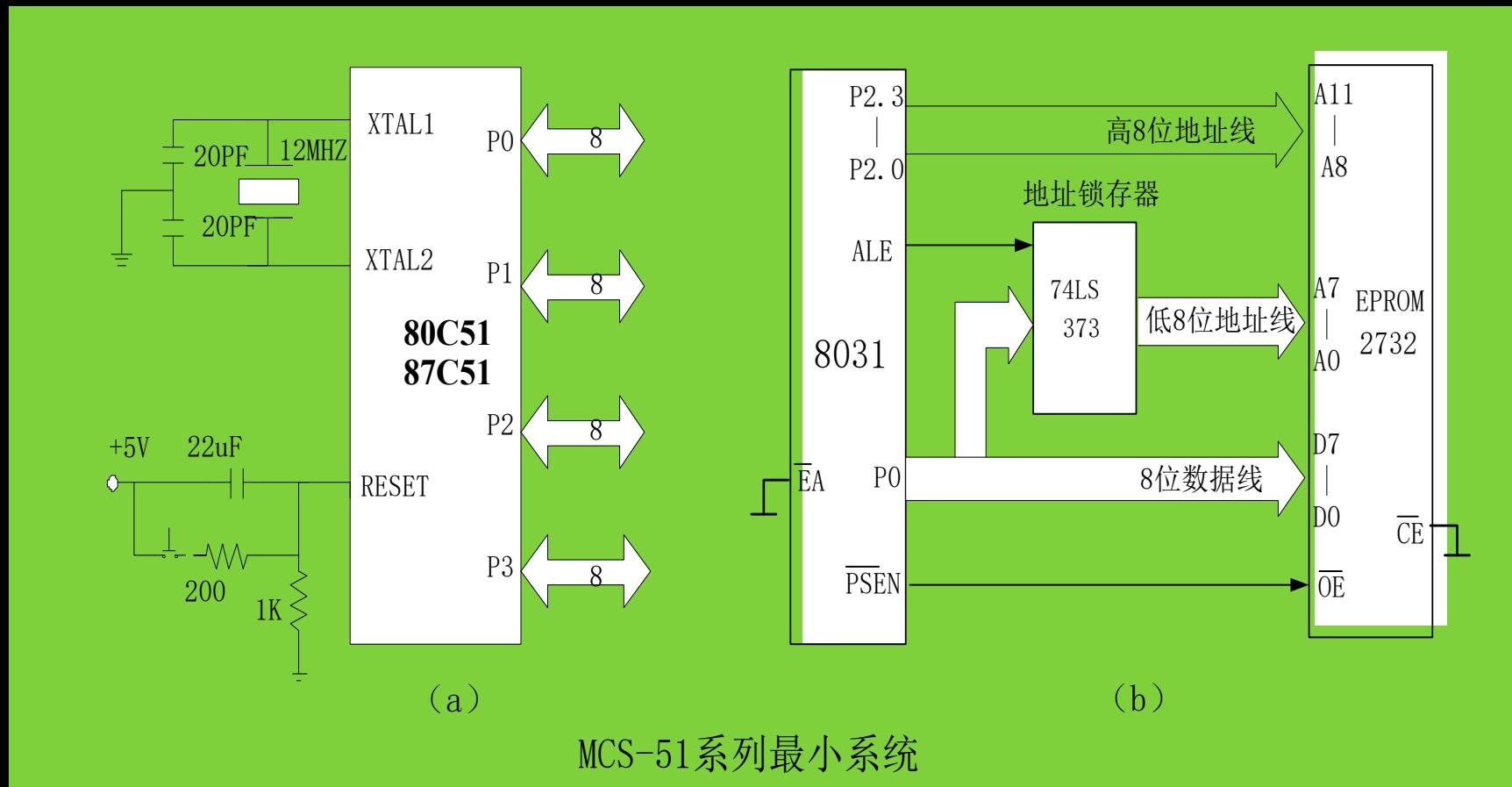
(a) 引脚图； (b) 内部结构框图。

## 5.2 CPU与存储器的连接

用单片机组成应用系统时，首先要考虑单片机所具有的各种功能能否满足应用系统的要求。如能满足，则称这样的系统为最小应用系统。

图 (a)为MCS-51系列中8051和8751单片机的最小系统。

图 (b)为由8031、8032单片机组成的最小系统。



微型机三总线结构---地址总线、数据总线和控制总线。

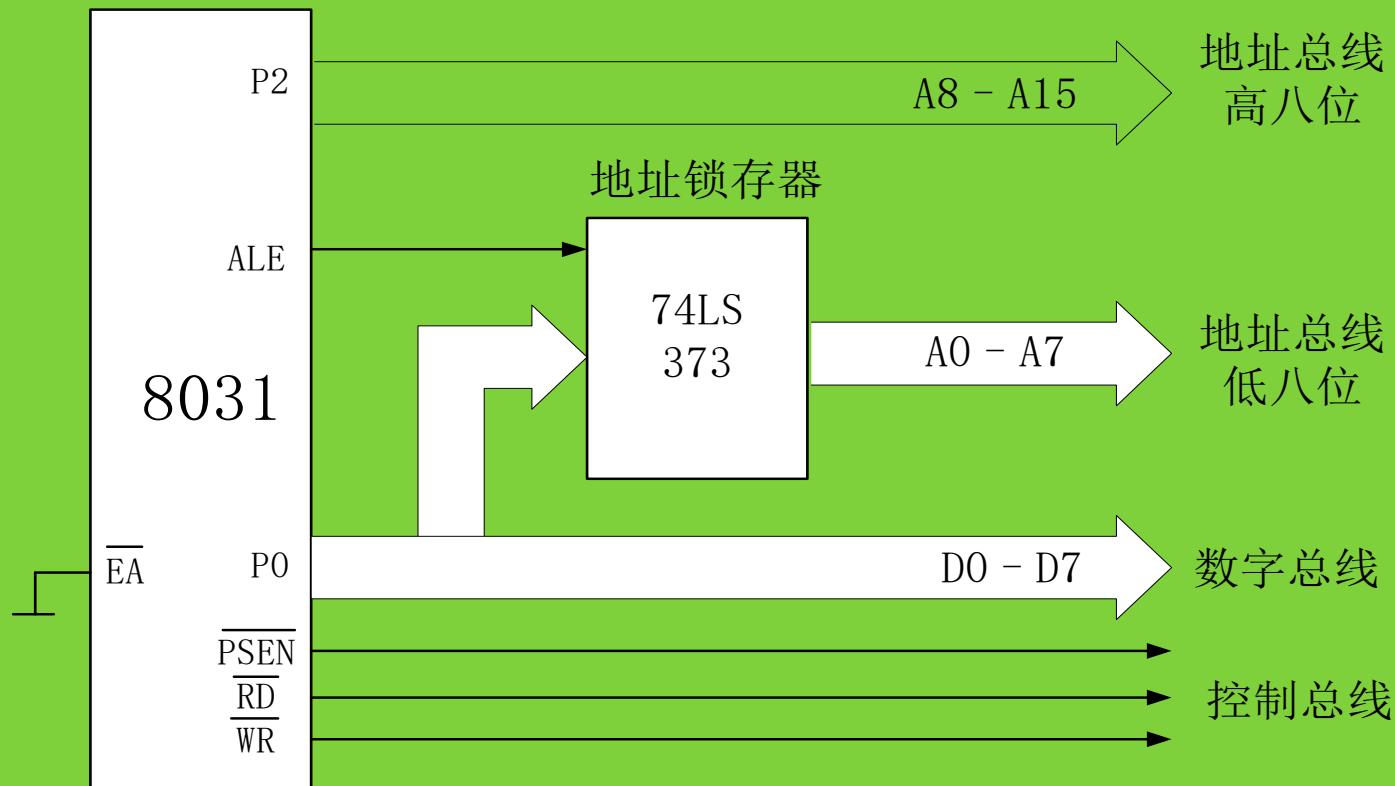
对MCS-51系列单片机，其三总线由I/O口及控制引线组成。

**地址总线**：由P<sub>2</sub>口提供高8位地址线（A8-A15），此口具有输出锁存的功能，能保留地址信息。由P<sub>0</sub>口提供低8位地址线。由于P<sub>0</sub>口是地址、数据分时使用的I/O口，为保存地址信息，需外加**地址锁存器**锁存低8位的地址信息。一般都用ALE正脉冲信号的下降沿控制锁存时刻。

**数据总线**：由P<sub>0</sub>口提供。此口是双向、三态控制的I/O口。

**控制总线**：扩展系统时常用的控制信号为ALE（地址锁存信号），PSEN（外部程序存储器读选通信号）以及数据存储器RAM和外设接口共用的读、写控制信号等。

扩展芯片与主机相连的方法同一般三总线结构的微处理器完全一样。



单片机的三总线结构

# 地址锁存器

- 常用的地址锁存器，有带三态缓冲输出的8D锁存器74LS373、8282和8D锁存器74LS273。
- OE是锁存器输出使能端。G、STB、CLK是选通脉冲输入端。CLR为清0端。

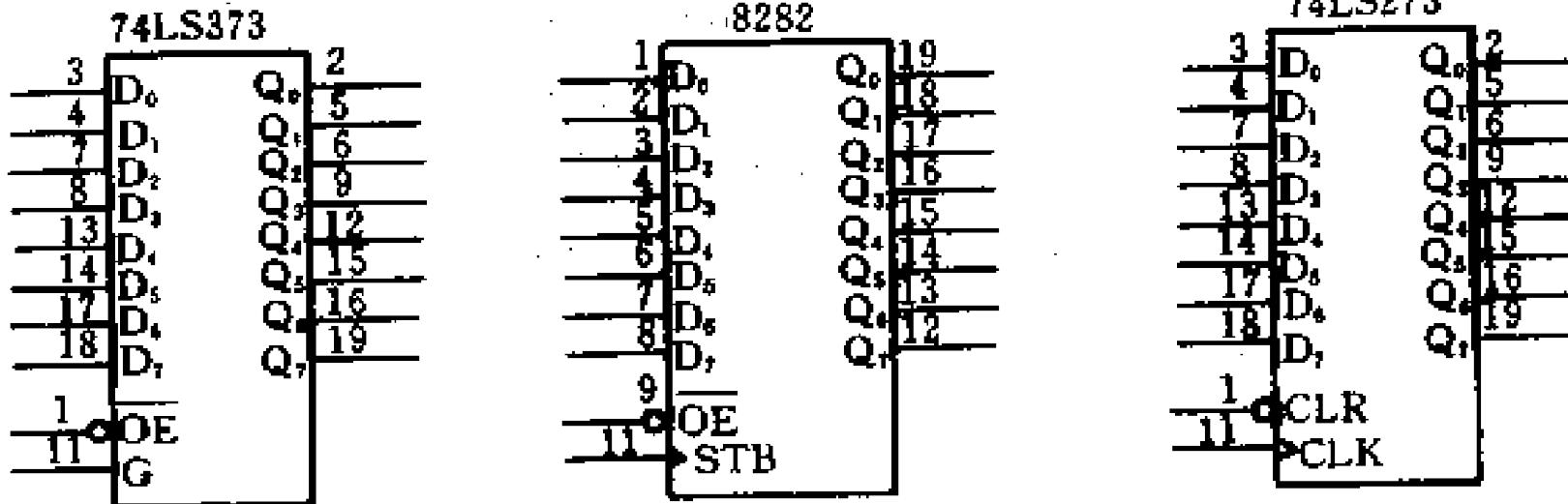
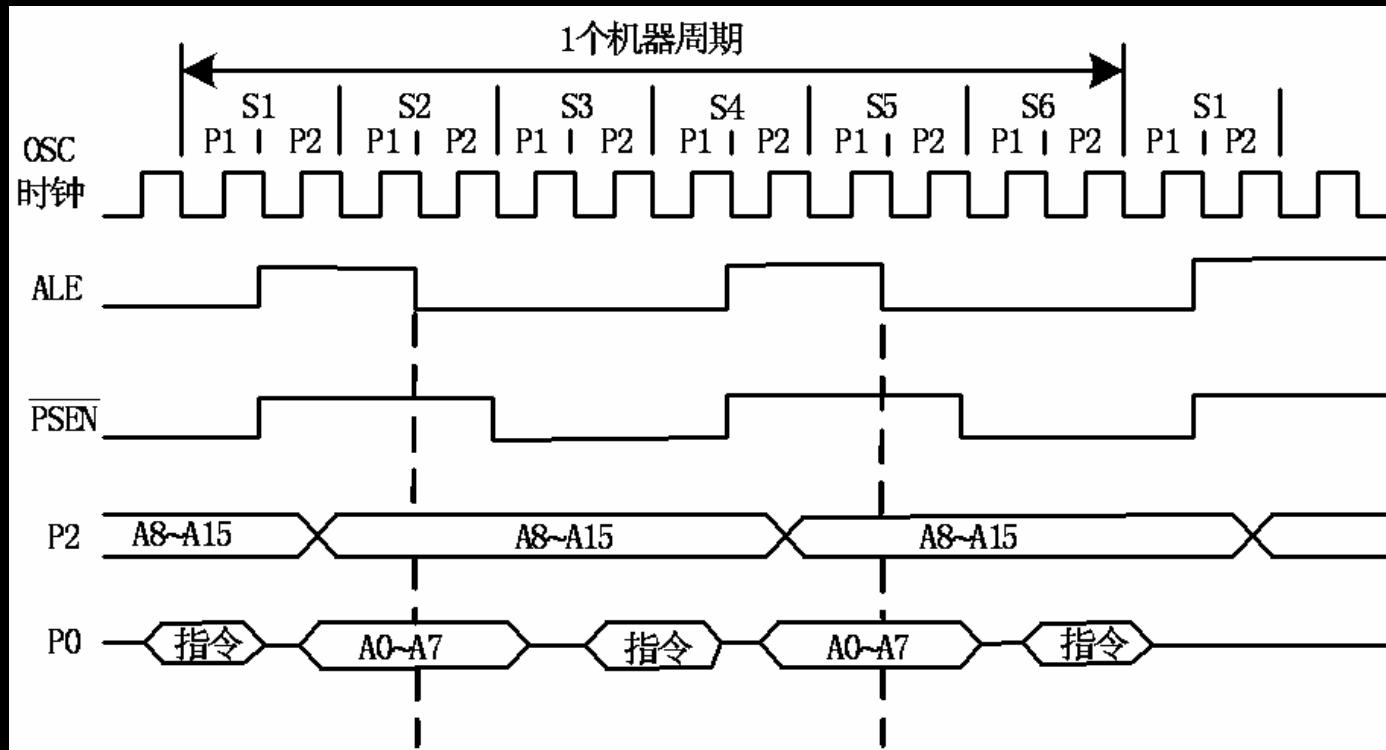


图 5.4.4 地址锁存器

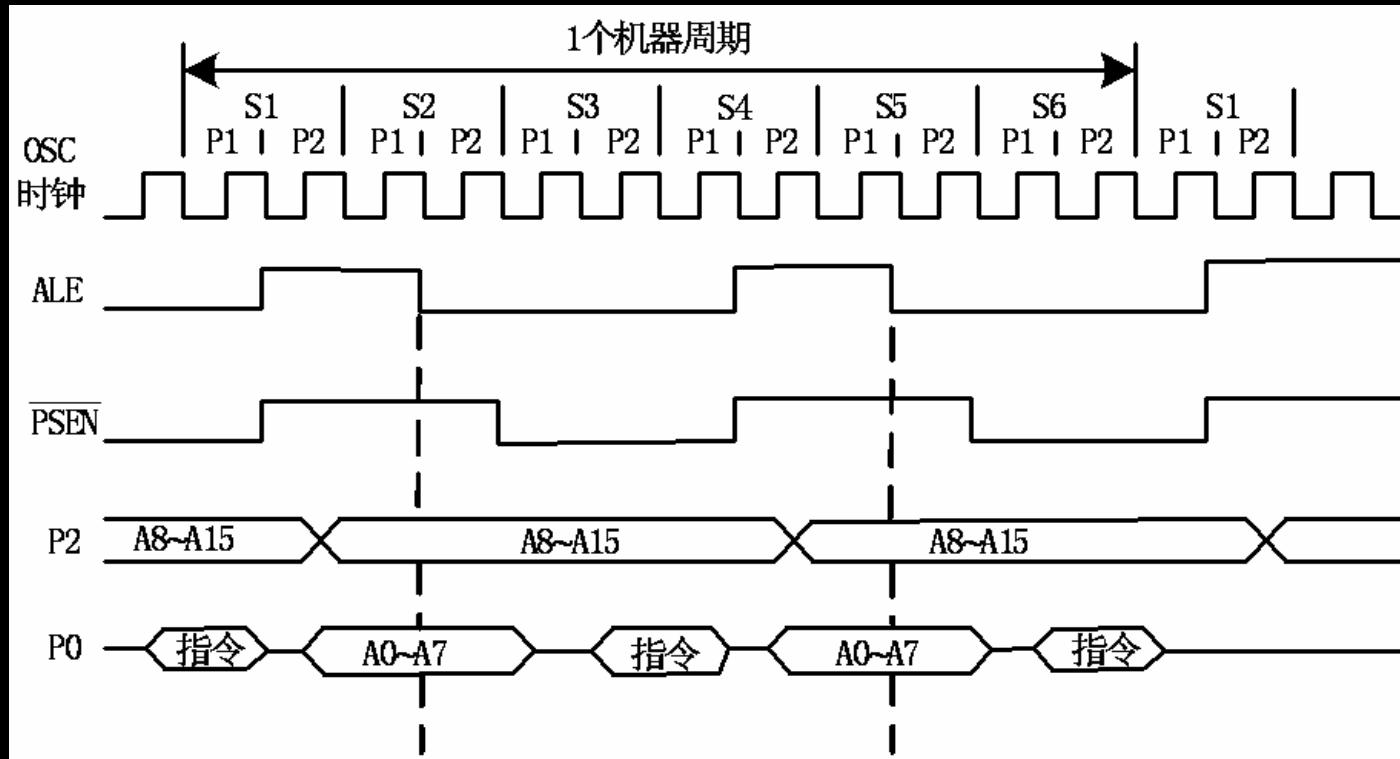
# 5.3 访问外部程序、数据存储器的时序

## 5.3.1 访问外部程序存储器时序



(1) 在S1P2时刻产生ALE信号。

(2) 由P0、P2口送出16位地址， P0口送出的低8位地址要利用ALE的下降沿信号将其锁存到地址锁存器中。从S2P2起，ALE信号失效。

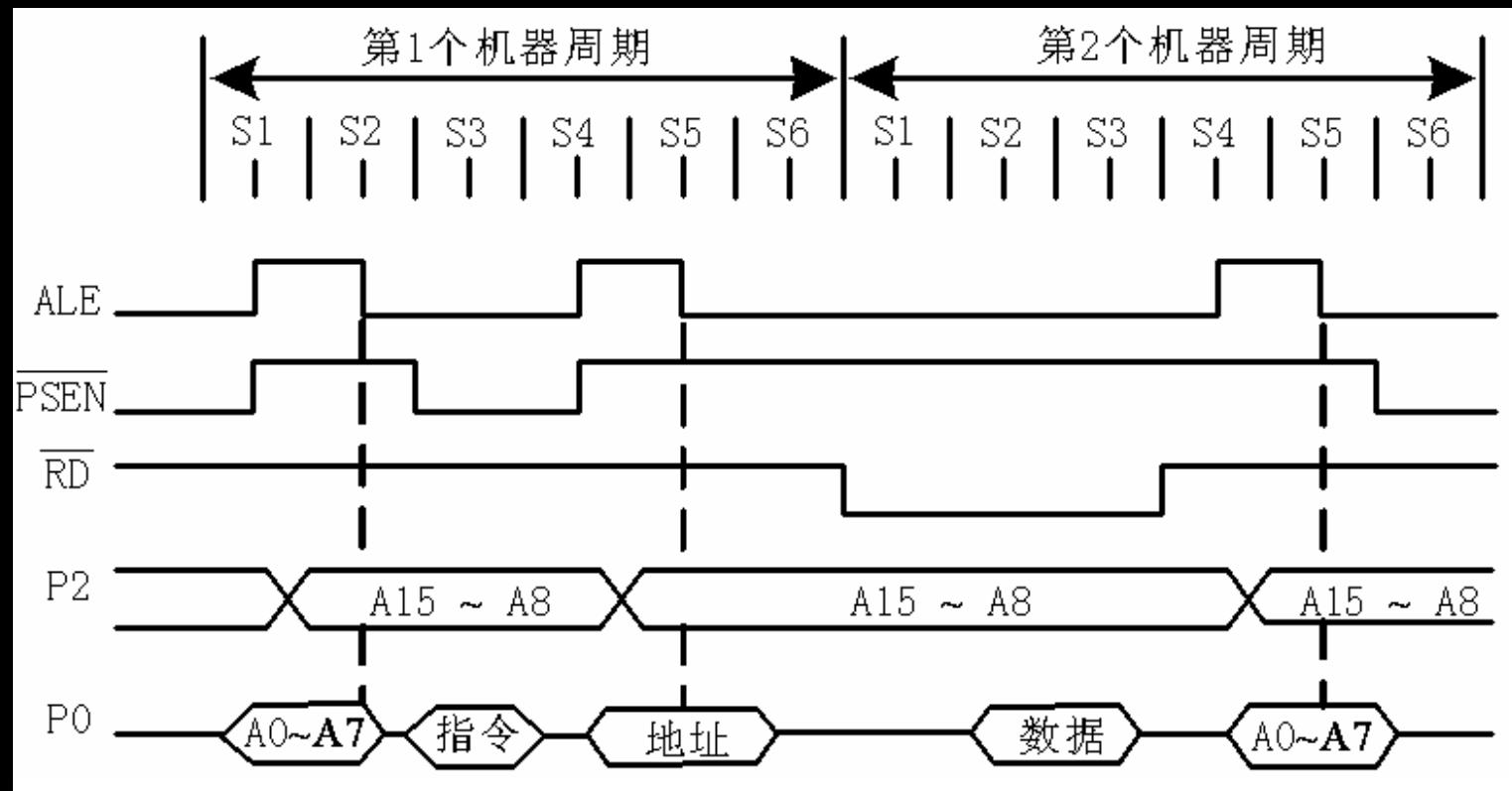


(3) 从S3P1开始，PSEN开始有效，对外部程序存储器进行读操作，将选中的单元中的指令代码从P0口读入，S4P2时刻，PSEN失效。

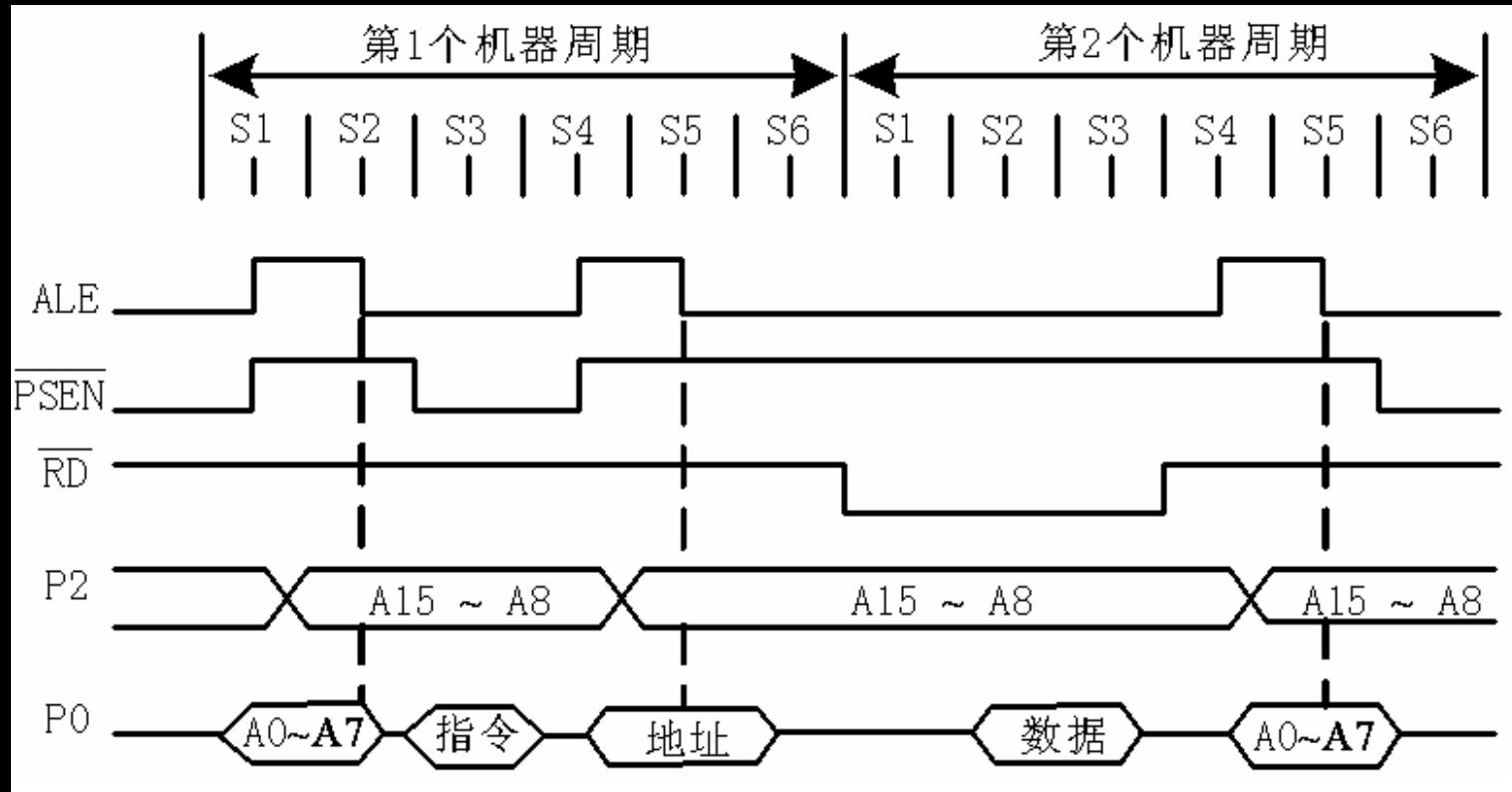
(4) 从S4P2后开始第二次读入，过程与第一次相似。

### 5.3.2 访问外部数据存储器时序

例：读时序



(1) 从第1次ALE有效到第2次ALE开始有效期间，P0口送出外部ROM单元的低8位地址，P2口送出外部ROM单元的高8位地址，并在PSEN有效期间，读入外部ROM单元中的指令代码。



(2) 在第2次ALE有效后，P0口送出外部RAM单元的低8位地址，P2口送出外部RAM单元高8位地址。

(3) 在第2个机器周期，第1次ALE信号不再出现，此时PSEN也失效，并在第2个机器周期的S1P1时，RD信号开始有效，从P0口读入选中RAM单元中的内容。

## 5.4 存储器扩展的编址技术

CPU 和片外存储器扩展连接，需要解决的重要问题就是存储器地址的分配和片选信号的产生，即要能够正确进行地址译码。

地址译码有两种方法。一种是线选法，一种是译码法。

## 5.4.1 线选法

所谓线选法，就是直接以系统的地址作为存储芯片的片选信号，只需把高位地址线与存储芯片的片选信号直接连接即可。特点是简单明了，不需增加另外电路。缺点是存储空间不连续，存储空间没有得到充分的利用。适用于小规模单片机系统的存储器扩展。

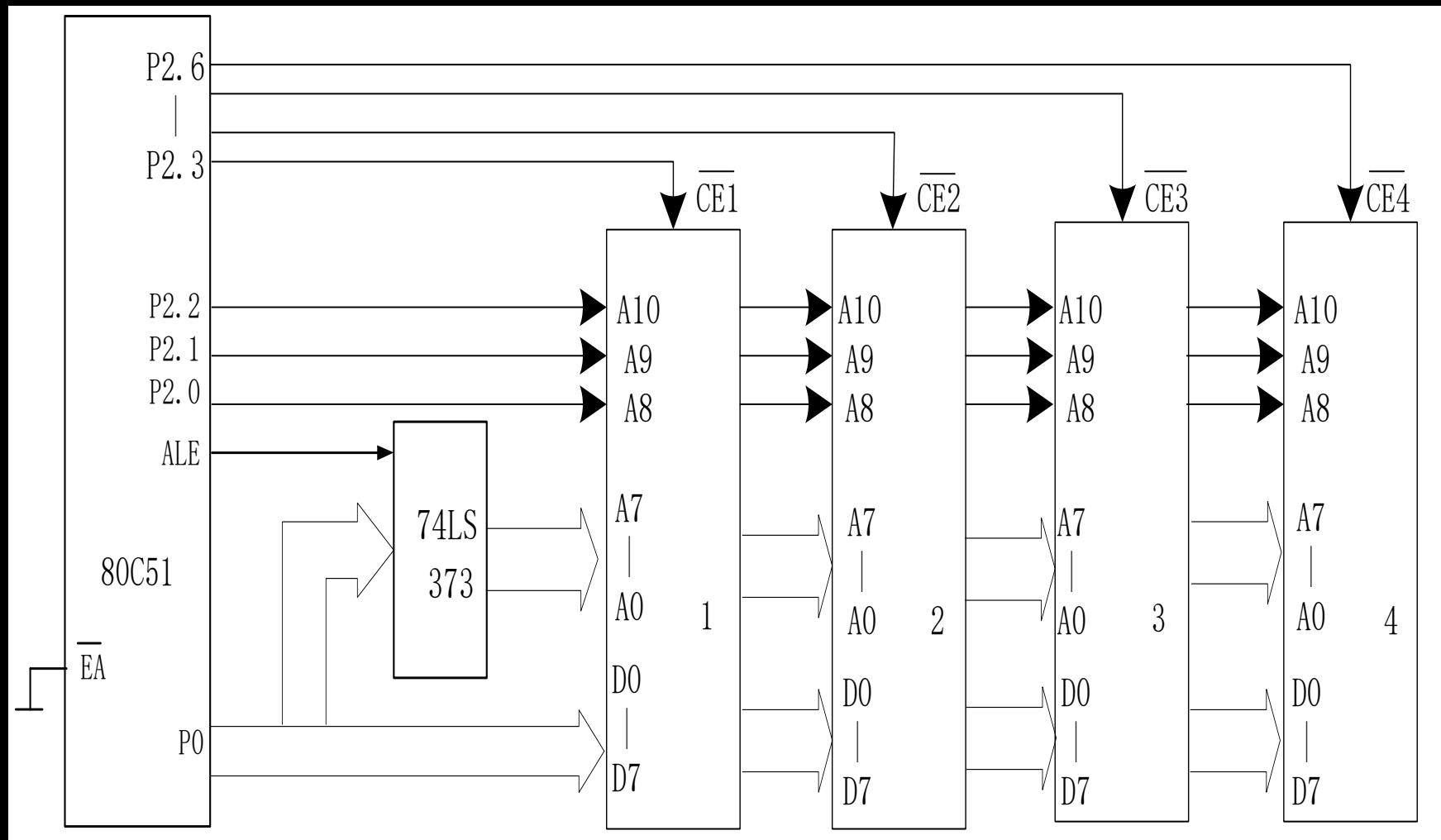
**【例5-1】**现有 $2K \times 8$ 位存储器芯片，需扩展 $8K \times 8$ 位存储结构，采用线选法进行扩展。

扩展 $8KB$ 的存储器结构需 $2KB$ 的存储器芯片4块。 $2K$ 的存储器所用的地址线为 $A_0 \sim A_{10}$ 共11根地址线，如何与CPU连接？

# 80C51与存储器的线路连接

80C51	存储器
P0口经锁存器锁存形成A0~A7	与A0~A7相连
P2.0、P2.1、P2.2	与A8~A10相连
P0口	与D0~D7相连
P2.3	与存储器1的片选信号相连
P2.4	与存储器2的片选信号相连
P2.5	与存储器3的片选信号相连
P2.6	与存储器4的片选信号相连

# 扩展存储器的硬件连接



线选法连线图

得到四个芯片的地址分配：

地址不连续!

线选方式地址分配表

	A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub> .... A <sub>0</sub>	地址范围
芯片1	0	1	1	1	0	0 .... 0	7000H---
	0	1	1	1	0	1 .... 1	77FFH
芯片2	0	1	1	0	1	0 .... 0	6800H---
	0	1	1	0	1	1 .... 1	6FFFH
芯片3	0	1	0	1	1	0 .... 0	5800H---
	0	1	0	1	1	1 .... 1	5FFFH
芯片4	0	0	1	1	1	0 .... 0	3800H—
	0	0	1	1	1	1 .... 1	3FFFH

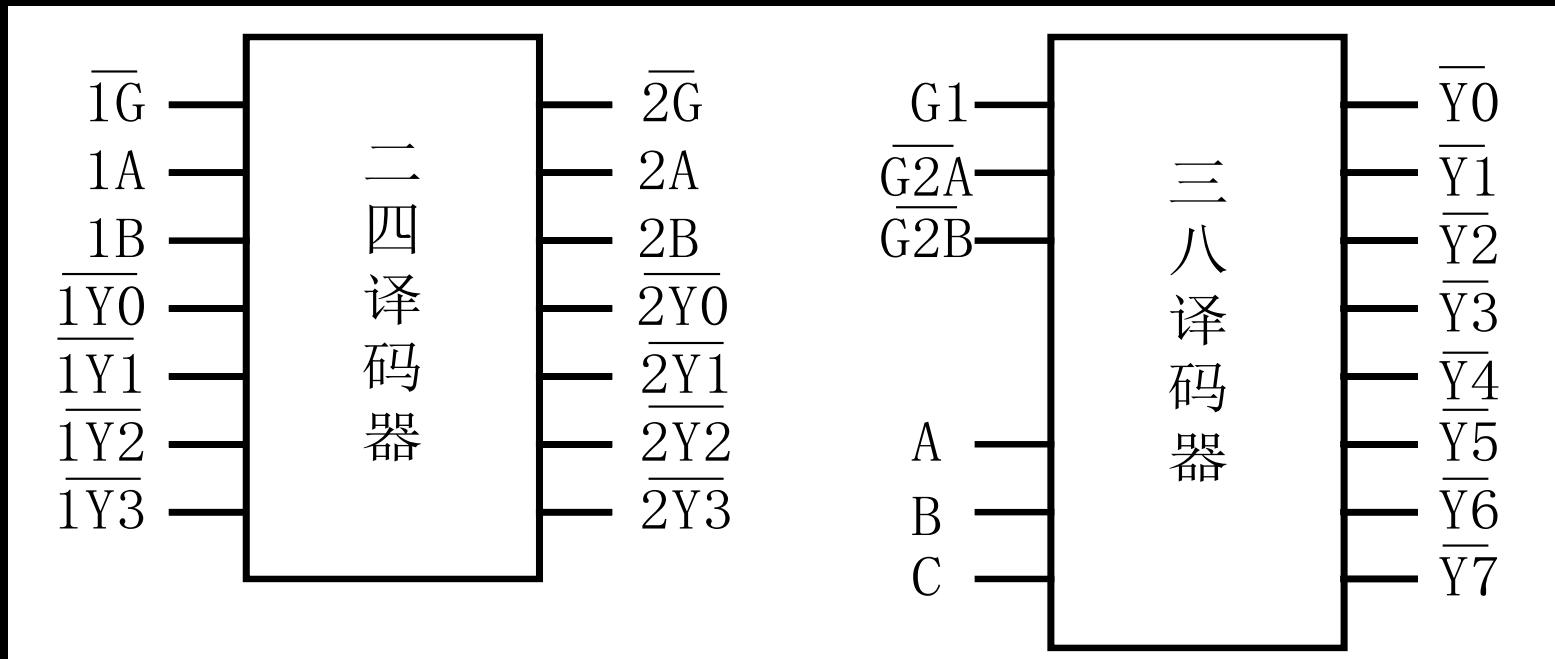
## 5.4.2 译码法

所谓译码法就是使用译码器对系统的高位地址进行译码，以其译码输出作为存储芯片的片选信号。这是一种最常用的存储器编址方法，能有效地利用空间，特点是存储空间连续，适用于大容量多芯片存储器扩展。

常用的译码芯片有：74LS139（双2线—4线译码器）和74LS138（3线—8线译码器）等，它们的CMOS型芯片分别是74HC139和74HC138。

74LS139译码器

74LS138译码器



译码器管脚图

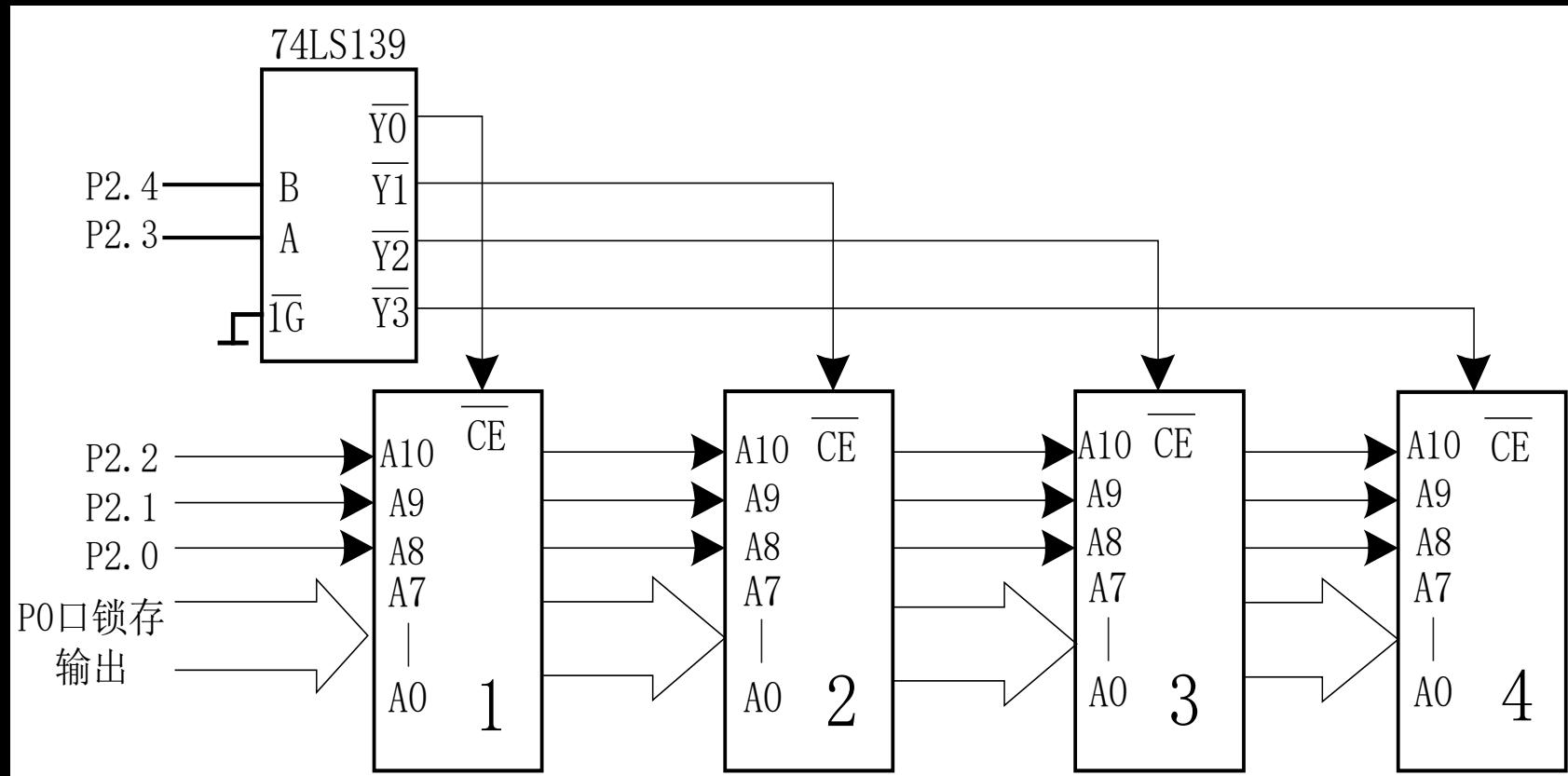
【例5-2】现有 $2K \times 8$ 位存储器芯片，需扩展 $8K \times 8$ 位存储结构，采用译码法进行扩展。

扩展8KB的存储器结构需2KB的存储器芯片4块。2K的存储器所用的地址线为 $A_0 \sim A_{10}$ 共11根地址线，采用译码法应如何连接呢？

## 80C51与存储器的线路连接

80C51	存储器		
$P_0$ 口经锁存器锁存形成 $A_0 \sim A_7$	与 $A_0 \sim A_7$ 相连		
$P_{2.0}$ 、 $P_{2.1}$ 、 $P_{2.2}$	与 $A_8 \sim A_{10}$ 相连		
$P0$ 口	与 $D_0 \sim D_7$ 相连		
$P_{2.4}$	$P_{2.3}$	译码输出与存储器的片选信号连接	
0	0	/Y0	与存储器1的片选信号相连
0	1	/Y1	与存储器2的片选信号相连
1	0	/Y2	与存储器3的片选信号相连
1	1	/Y3	与存储器4的片选信号相连

$P_{2.3}$ 、 $P_{2.4}$ 作为二-四译码器的译码地址，译码输出作为扩展4个存储器芯片的片选信号， $P_{2.5}$ 、 $P_{2.6}$ 、 $P_{2.7}$ 空置（部分译码）。



## 采用译码器扩展8KB存储器连线图

地址连续!

得到四个芯片的地址分配如下：

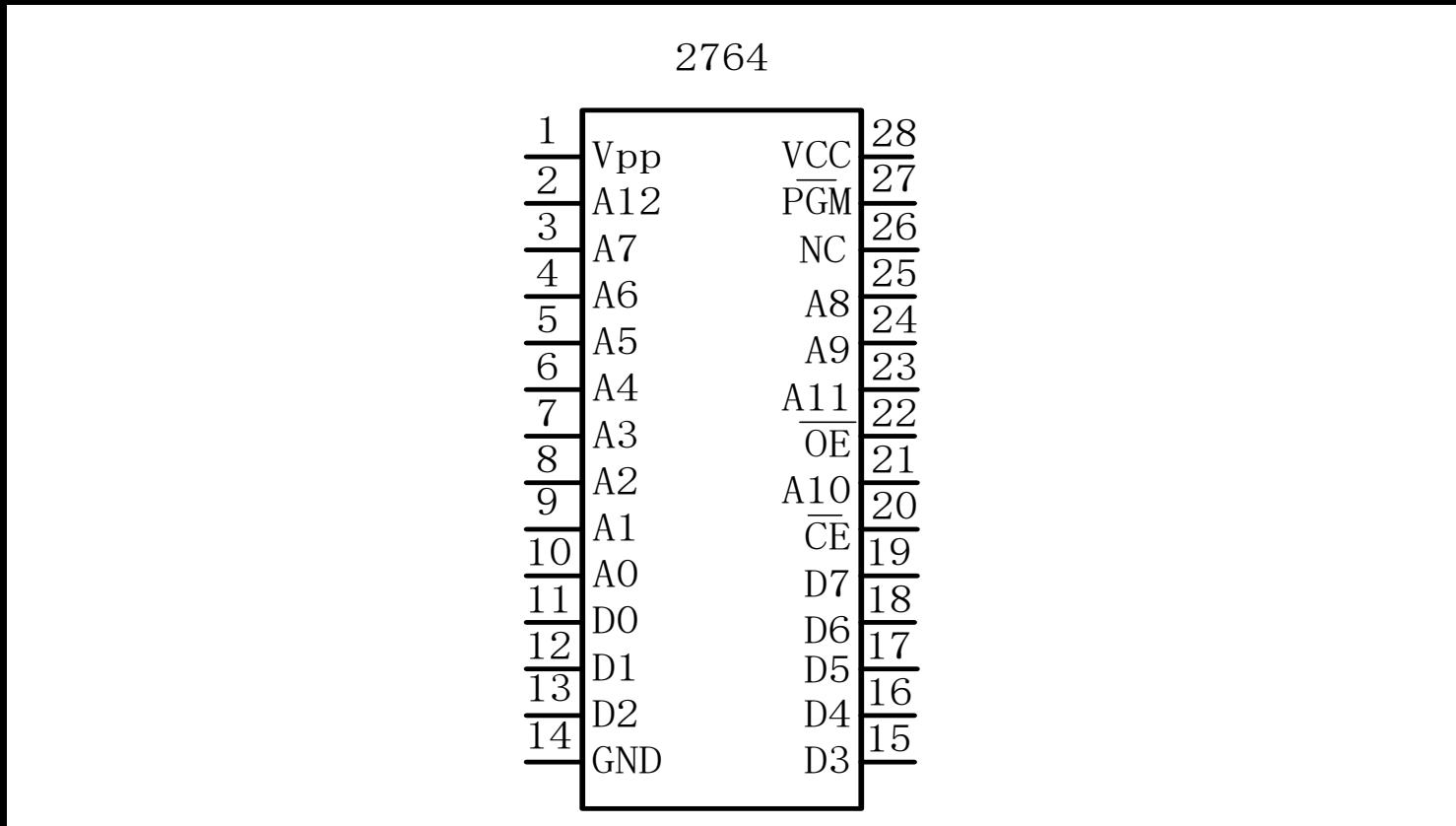
译码方式地址分配表

	$P_{2.7}$	$P_{2.6}$	$P_{2.5}$	$P_{2.4}$	$P_{2.3}$	$P_{2.2} \dots P_0$	地址范围
芯片1	0	0	0	0	0	0 .... 0	0000H---
	0	0	0	0	0	1 .... 1	07FFH
芯片2	0	0	0	0	1	0 .... 0	0800H---
	0	0	0	0	1	1 .... 1	0FFFH
芯片3	0	0	0	1	0	0 .... 0	1000H---
	0	0	0	1	0	1 .... 1	17FFH
芯片4	0	0	0	1	1	0 .... 0	1800H---
	0	0	0	1	1	1 .... 1	1FFFH

# 5.5 程序存储器 (EPROM) 的扩展

## 5.5.1 程序存储器扩展使用的典型芯片

程序存储器扩展使用的典型芯片有2716、2732、2764、27128、27256等。以2764作为单片机程序存储器扩展的典型芯片为例进行说明。



EPROM 2764 管脚

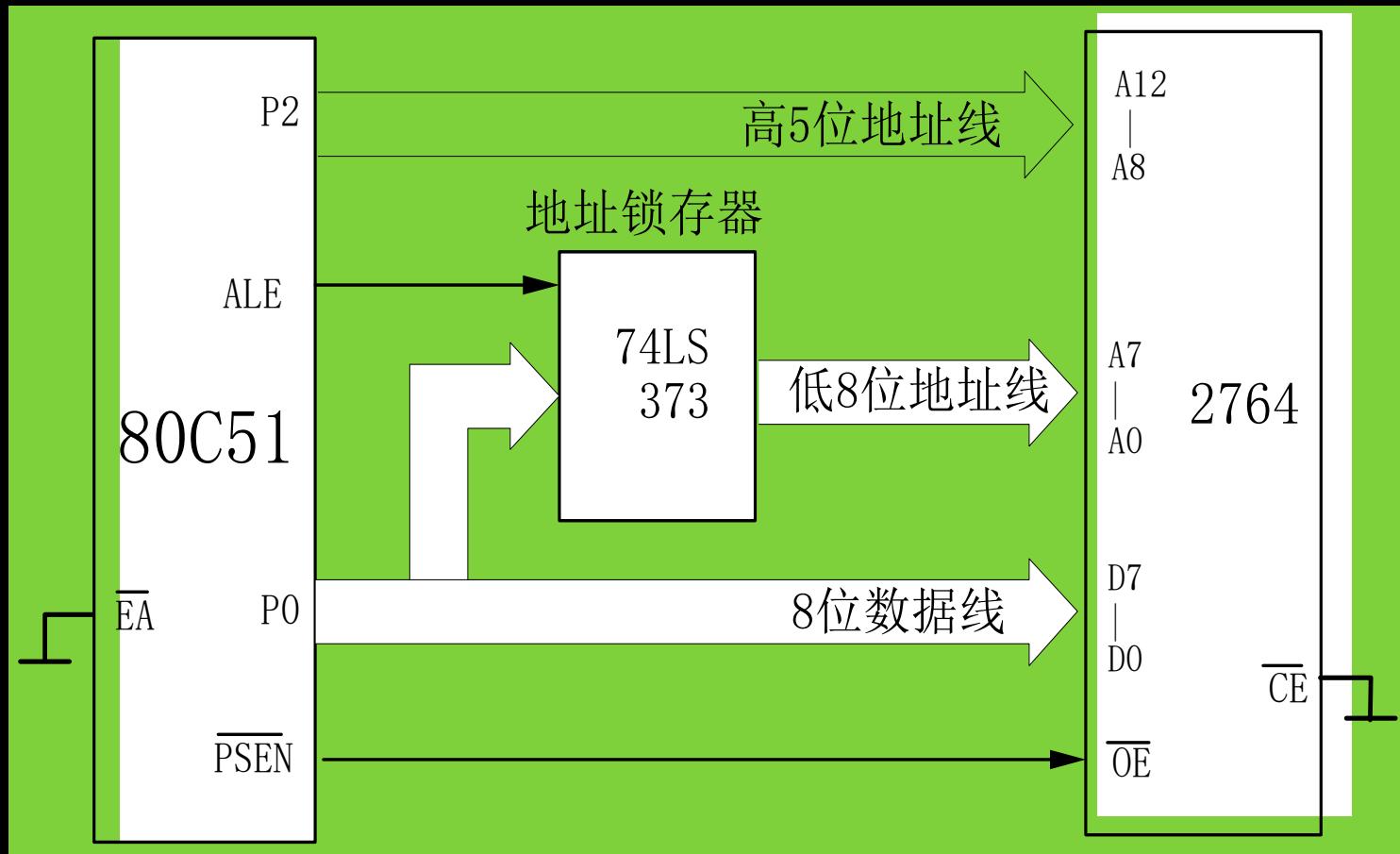
## 2764的引线

2764是一块8K×8bit（8KB）的EPROM芯片

- A12~A0——13位地址信号输入线，说明芯片的容量为8K
- D7~D0——8位数据，表明芯片的每个存储单元存放一个字节（8位二进制数）
- $\overline{CE}$ 为输入（片选）信号。当低电平有效时，选中该芯片
- $\overline{OE}$ 为输出允许信号。当OE为低电平时，芯片中的数据可由D7~D0输出
- $\overline{PGM}$ 为编程脉冲输入端。当对EPROM编程时，由此加入编程脉冲，读时 $\overline{PGM}$ 为高电平

## 2. 2764的连接使用

扩展一片EPROM的最小系统如下：



## 存储器地址范围

分析存储器在存储空间中占据的地址范围，实际上就是根据连接情况确定其最低地址和最高地址。由于 $P_{2.7}$ 、 $P_{2.6}$ 、 $P_{2.5}$ 的状态与2764芯片的寻址无关，所以 $P_{2.7}$ 、 $P_{2.6}$ 、 $P_{2.5}$ 可为任意。从000到111共有8种组合，其2764芯片的地址范围是：

最低地址: 0000H ( $A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3$   
 $A_2A_1A_0 = 0000\ 0000\ 0000\ 0000$ )

最高地址: FFFFH ( $A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3$   
 $A_2A_1A_0 = \times \times \times 1\ 1111\ 1111\ 1111$ )

共占用了64KB的存储空间，造成地址空间的重叠和浪费

# 5.6 数据存储器的扩展

## 5.6.1 数据存储器的扩展概述

单片机与数据存储器的连接方法和程序存储器连接方法大致相同。

1. 地址线的连接，与程序存储器连法相同。
2. 数据线的连接，与程序存储器连法相同。
3. 控制线的连接，主要有下列控制信号：

存储器输出信号和单片机读信号相连，即和P<sub>3.7</sub>相连。

存储器写信号 和单片机写信号相连，即和P<sub>3.6</sub>相连。

ALE：其连接方法与程序存储器相同。

使用时应注意，访问内部和外部数据存储器时，应分别使用MOV及MOVX指令。

外部数据存储器通常设置二个数据区：

(1) 低8位地址线寻址的外部数据区。此区域寻址空间为256个字节。

读存储器数据指令: **MOVX** A, @R i

写存储器数据指令: **MOVX** @R i , A

(2) 16位地址线寻址的外部数据区。当外部RAM容量较大,访问RAM地址空间大于256个字节时, 采用16位寻址指令。

读存储器数据指令: **MOVX** A, @DPTR

写存储器数据指令: **MOVX** @DPTR, A

DPTR为16位的地址指针, 可寻址64KRAM字节单元。

## 5.6.2 数据存储器扩展使用的典型芯片

### 1. 数据存储器SRAM芯片

数据存储器扩展常使用静态RAM芯片，典型产品是Intel公司的6116、6264和62128。  
以6264芯片为例进行说明。

6264

1	NC	VCC	28
2	A12	$\overline{WE}$	27
3	A7	CE2	26
4	A6	A8	25
5	A5	A9	24
6	A4	A11	23
7	A3	$\overline{OE}$	22
8	A2	A10	21
9	A1	$\overline{CE1}$	20
10	A0	D7	19
11	D0	D6	18
12	D1	D5	17
13	D2	D4	16
14	GND	D3	15

SRAM 6264 管脚图

6264芯片的主要引脚为：

- $A_{12} \sim A_0$ ——13根地址线，说明芯片的容量为8K。
- $D_7 \sim D_0$ ——8根数据线， $\overline{CE1}$ 、 $CE2$ 为片选信号。当 $\overline{CE1}$ 为低电平， $CE2$ 为高电平时，选中该芯片。
- $\overline{OE}$ 为输出允许信号。当 $\overline{OE}$ 为低电平时，芯片中的数据可由 $D_7 \sim D_0$ 输出。
- $\overline{WE}$ 为数据写信号。

## 2. 数据存储器扩展方法

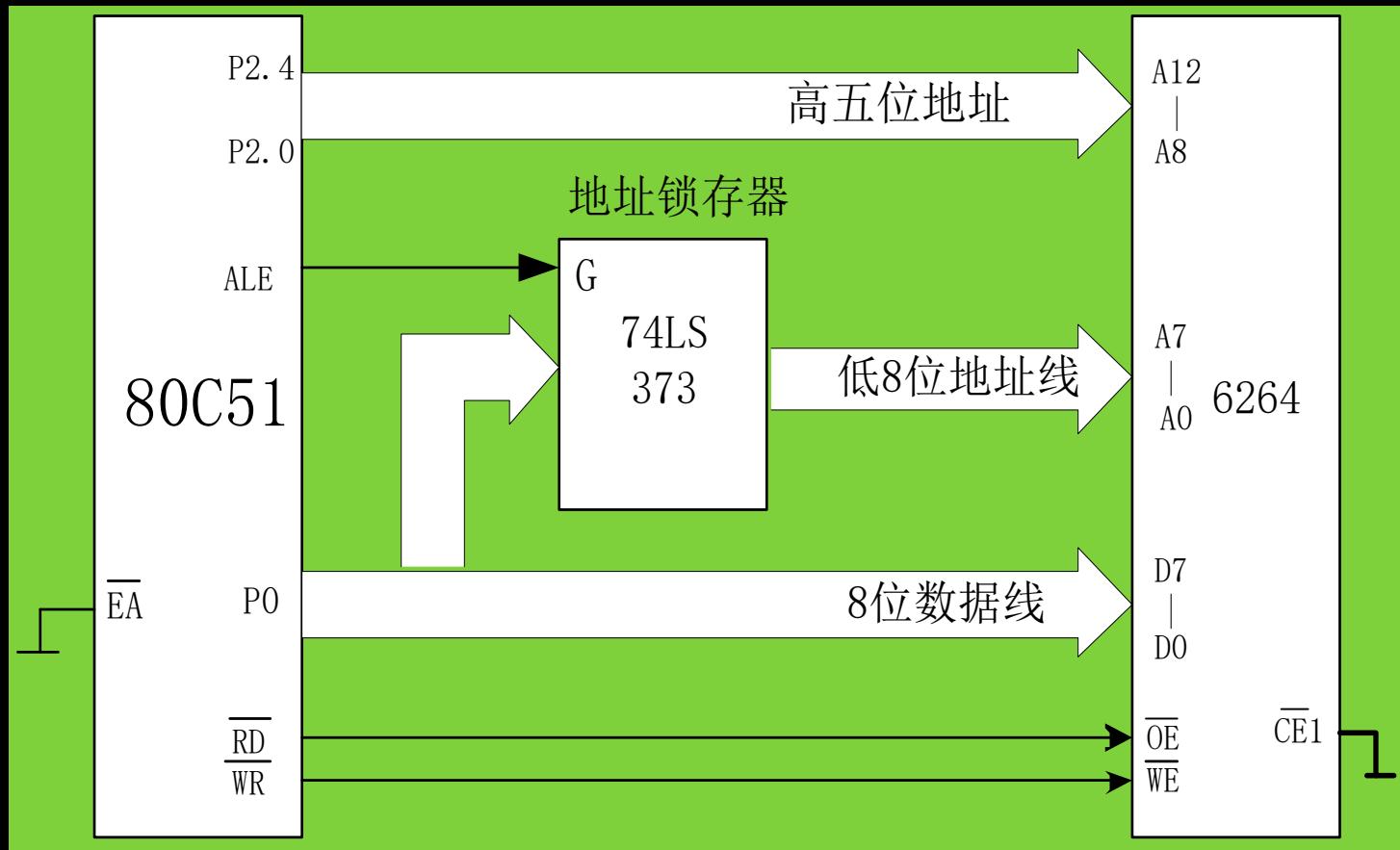
### (1) 单片数据存储器扩展

80C51与6264的连接如下表：

80C51与6264的线路连接

80C51	6264
$P_0$ 经锁存器锁存形成 $A_0 \sim A_7$	$A_0 \sim A_7$
$P_{2.0} \sim P_{2.4}$	$A_8 \sim A_{12}$
$D_0 \sim D_7$	$D_0 \sim D_7$

# 数据存储器扩展的硬件连接



单片RAM扩展连线图

## (2) 多片数据存储器扩展

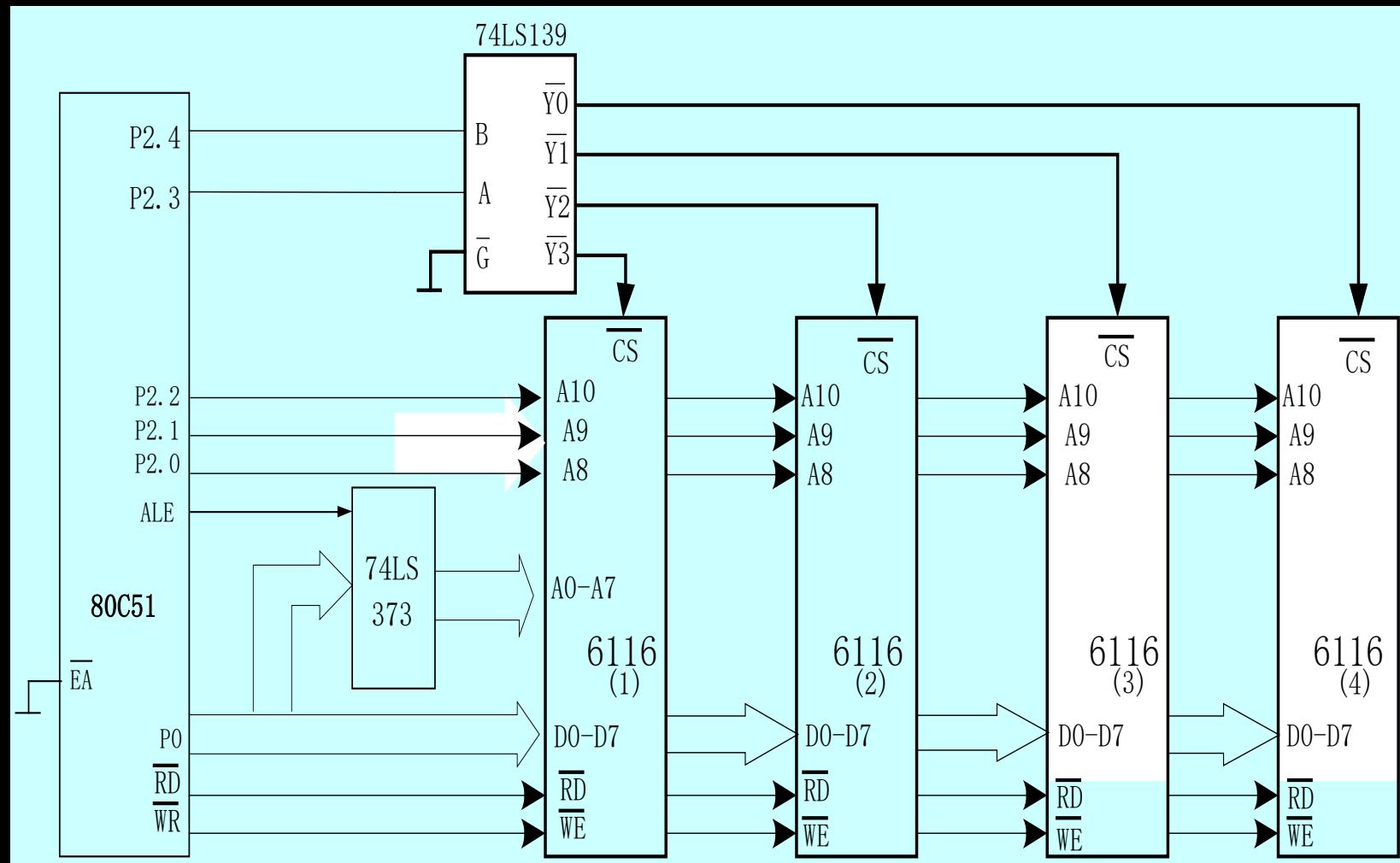
例如：用4片6116进行8KB数据存储器扩展，用译码法实现。

80C51与6116的线路连接如下表：

80C51与6116的线路连接

80C51	6116	二四译码器译码形成	二四译码器译码形成		
P <sub>0</sub> 口经锁存器锁存 A <sub>0</sub> ~A <sub>7</sub>	A <sub>0</sub> ~A <sub>7</sub>		A <sub>11</sub>	A <sub>12</sub>	译码控制片选
P <sub>10</sub> 、P <sub>11</sub> 、P <sub>12</sub>	A <sub>8</sub> ~A <sub>10</sub>		0	0	$\overline{Y_0} \leftrightarrow CS_4$
D <sub>0</sub> ~D <sub>7</sub>	D <sub>0</sub> ~D <sub>7</sub>		0	1	$\overline{Y_1} \leftrightarrow CS_3$
$\overline{RD}$	$\overline{OE}$		1	0	$\overline{Y_2} \leftrightarrow CS_2$
$\overline{WR}$	$\overline{WE}$			1	$\overline{Y_3} \leftrightarrow CS_1$

# 存储器扩展电路连接



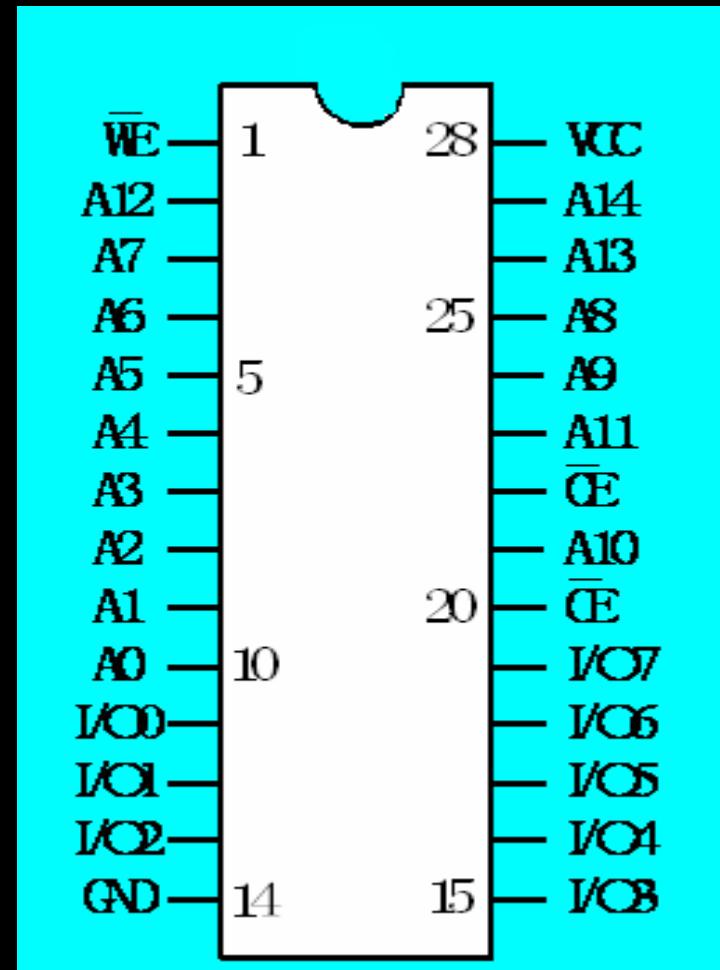
多片RAM扩展连线图

### 5.6.3 闪速存储器及其扩展

闪存的数据不易丢失，速度高、容量大，擦写非常灵活，被广泛用于单片机的程序存储器和数据存储器。

#### 1. 引脚功能和读写操作

以AT29C256芯片为例说明。其容量为32KB，引脚数量为28条。



AT29C256芯片引脚排列图

AT29C256芯片主要引脚功能：

$A_0 \sim A_{14}$ : 15根地址线

$I/O_0 - I/O_7$ : 三态双向数据线

$\overline{CE}$  : 片选信号线, 低电平输入有效

$\overline{OE}$  : 输出允许 (读允许) 信号线, 低电平

输入有效。

$\overline{WE}$  : 写允许信号线, 低电平输入有效。

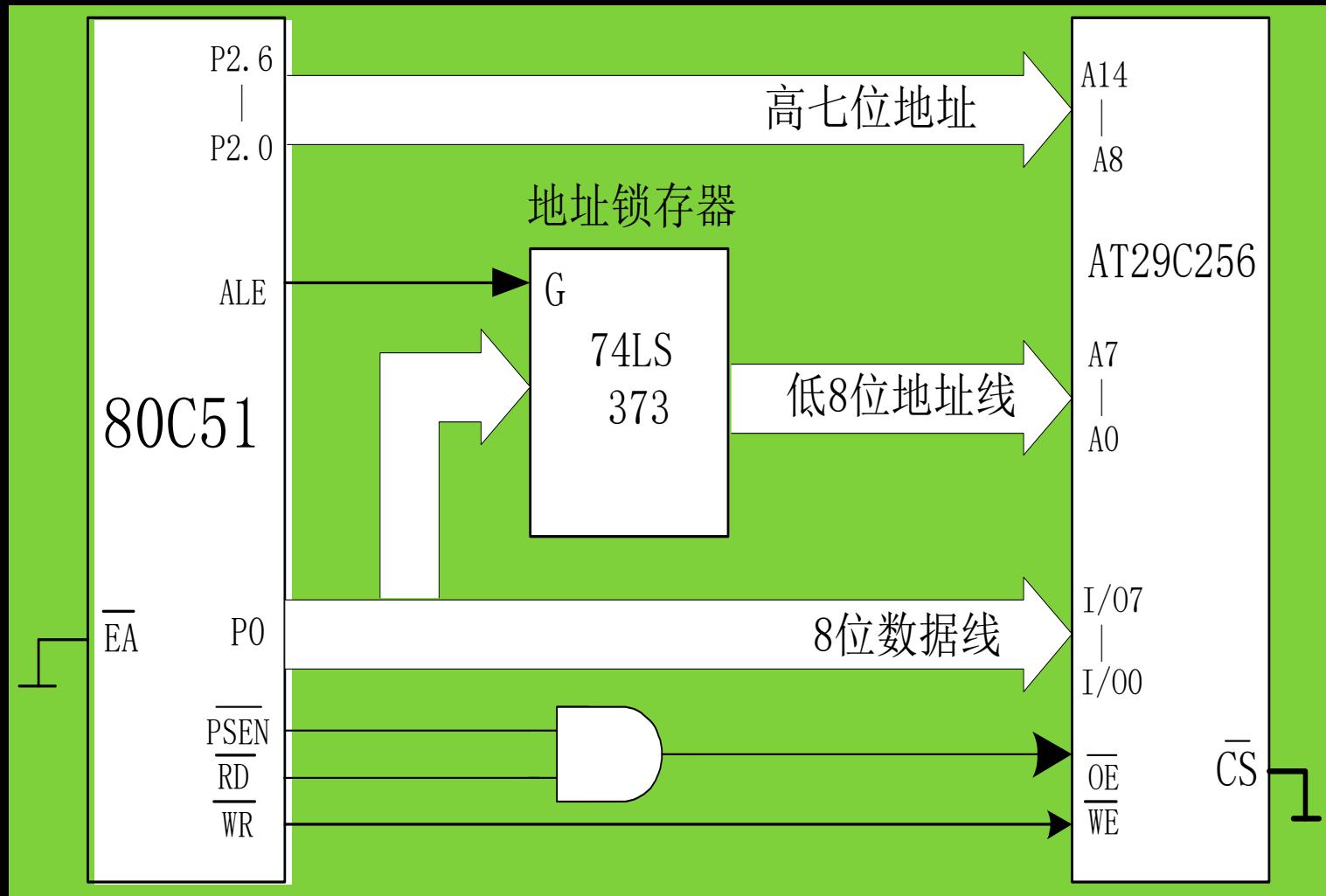
## (1) 读操作

当 $\overline{CE}=0$ ,  $\overline{OE}=0$ ,  $\overline{WE}=1$ 时, 被选中单元的内容读出到双向数据线 $I/O_0 \sim I/O_7$ 上。当 $\overline{OE}$ 为高电平, 输出线处于高阻状态。

## (2) 写操作

外部数据写入29C256芯片时, 数据要先装入内部锁存器, 装入时 $\overline{CE}=0$ ,  $\overline{OE}=1$ ,  $\overline{WE}=0$ 。数据写入以页为单位进行, 即要改写某一单元的内容整页都要重写, 没有被装入的字节内容被写成 $0FFH$ 。

## 2. MCS-51单片机与AT29C256的接口



80C51单片机与AT29C256芯片的接口电路

连线说明：

80C51单片机的 $\overline{\text{PSEN}}$ 和 $\overline{\text{RD}}$ 相“与”后与AT29C256芯片的 $\overline{\text{OE}}$ 端相连，80C51的 $\overline{\text{WR}}$ 与AT29C256芯片的 $\overline{\text{WE}}$ 相连，可实现对AT29C256芯片的读写信号的选通。

以上扩展的方法与数据存储器的扩展方法相同，单片机访问它时，也使用**MOVX**指令。

# 本章小结

了解半导体存储器的分类

掌握 CPU 与存储器的连接

**重点掌握 MCS-51 与片外存储器的扩展**

# 第6章 中断系统

## 6.1 概述

## 6.2 MCS - 51 中断系统

## 6.3 中断系统的应用

# 6.1 概述

## 1. 中断

中断是指计算机在执行某一程序的过程中，由于计算机系统内、外的某种原因，而必须中止原程序的执行，转去执行相应的处理程序，待处理结束之后，再回来继续执行被中止的原程序的过程。

◆ 中断处理与调用子程序的区别？

中断处理是随机的，而调用子程序是在程序中事先安排好的。

- 采用了中断技术后的计算机，可以解决CPU与外设之间速度匹配的问题，使计算机能够及时处理系统中许多随机的参数和信息，同时它也提高了计算机处理故障与应变的能力。

“中断”与“查询”相比：

执行效率 ↑

实时性 ↑

## 2. 中断源——提出中断请求的来源

### 1) 设备中断

由组成计算机应用系统的外部设备发出的中断申请，称为设备中断。

### 2) 定时时钟

定时提出中断申请。例如，在定时控制或定时数据采集系统中，由内部或外部时钟电路定时，一旦到达规定的时间，时钟电路就向CPU发出中断申请。

### 3) 故障源

例如：目前，微型机的内存RAM是采用半导体存储器，所以在电源掉电时，需要接入备用电源，以便保护存储器RAM中的信息。一般是在直流电源上并联电容，当电容电压因电源掉电下降到一定值时就发出中断申请，CPU响应中断执行保护现场信息的操作。

#### 4) 程序性中断源

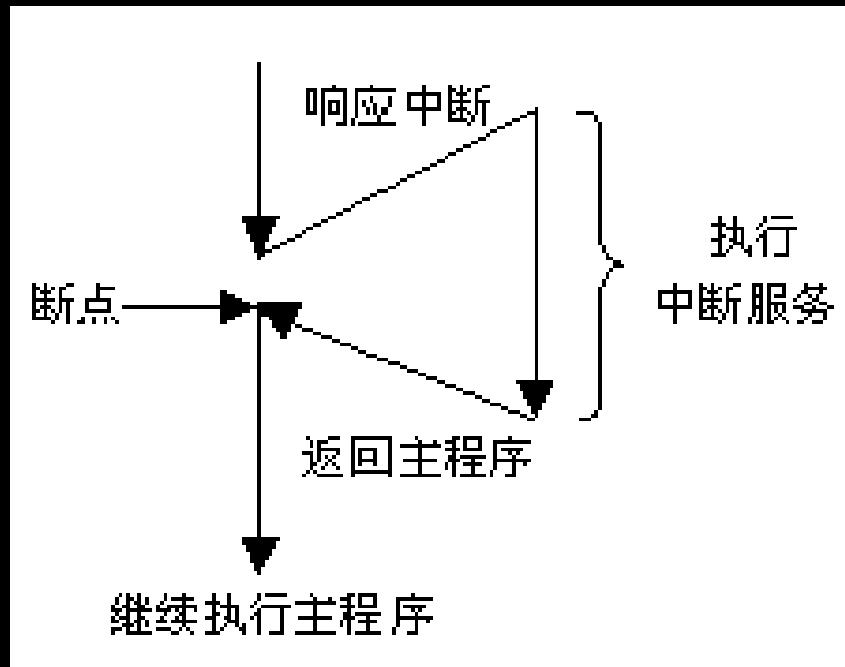
例如，为调试程序而设置断点、单步工作等。

对于每个中断源，不仅要求能发出中断请求信号，而且这个信号还要能保持一定的时间，直至CPU响应这个中断请求后才能撤消中断请求信号。既不因CPU未及时响应而丢失中断申请信号，也不会出现多次重复中断的情况。因此要求每个中断源的接口电路中有一个中断请求触发器。

另外，在实际系统中往往有多个中断源，为增加控制的灵活性，在每个中断源的接口电路中还设置一个中断屏蔽触发器，由它控制该中断源的中断申请信号能否送到CPU。

### 3. 中断系统的功能

## 1) 实现中断及返回



当某一中断源发出中断申请时，若CPU允许响应这个中断请求，则在现行指令执行完后，把断点处的PC值（下一条要执行指令的地址）、有关寄存器的内容和标志位的状态压入堆栈保存（保护断点和现场），再转到相应的中断服务程序的入口，同时清除中断请求触发器。

当中断服务程序执行完以后，再恢复被保留的寄存器的内容和标志位的状态（恢复现场），并将断点地址从堆栈中弹出到PC（恢复断点），使CPU返回断点处，继续执行主程序。

### 3. 中断系统的功能

#### 2) 实现中断优先权排队

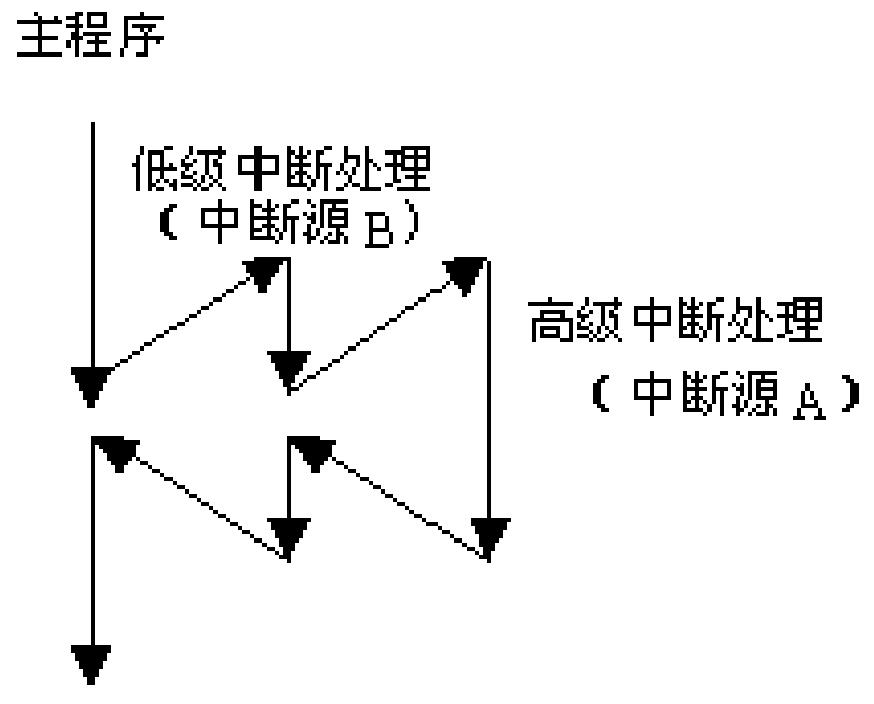
通常在系统中有多个中断源，若出现两个或多个中断源同时提出中断请求的情况，就需要CPU既能区分各个中断源的请求，又能确定首先为哪一个中断源服务。

为了解决这一问题，用户事先根据事件处理的紧迫性和实时性给各中断源规定优先级别，即中断优先权。CPU按中断优先权的高低逐次响应中断的过程称为中断优先权排队。

当有两个或多个中断源同时提出中断请求时，CPU识别出优先权高的中断源，并响应它的中断请求，待处理完后，再响应优先权低的中断源。

### 3. 中断系统的功能

#### 3) 实现中断嵌套



当CPU响应某一中断源请求而进行中断处理时，若有优先级别更高的中断源发出中断申请，CPU应能中断正在执行的中断服务程序，保留这个程序的断点，响应优先权级别高的中断，在高级中断处理完后，再返回被中断的中断服务程序，继续原先的处理。这个过程就是中断嵌套。

优先权低的中断不能中断优先权高的中断处理。

## 4. 中断响应的一般过程

- (1) 在每条指令结束后, 系统都自动检测中断请求信号, 如果有中断请求, 且CPU处于开中断状态下, 则响应中断。
- (2) 保护现场。在保护现场前CPU要关中断, 以防止现场被破坏。保护现场一般是用堆栈指令将原程序中用到的寄存器等压入堆栈。
- (3) 中断服务, 即为相应的中断源服务。

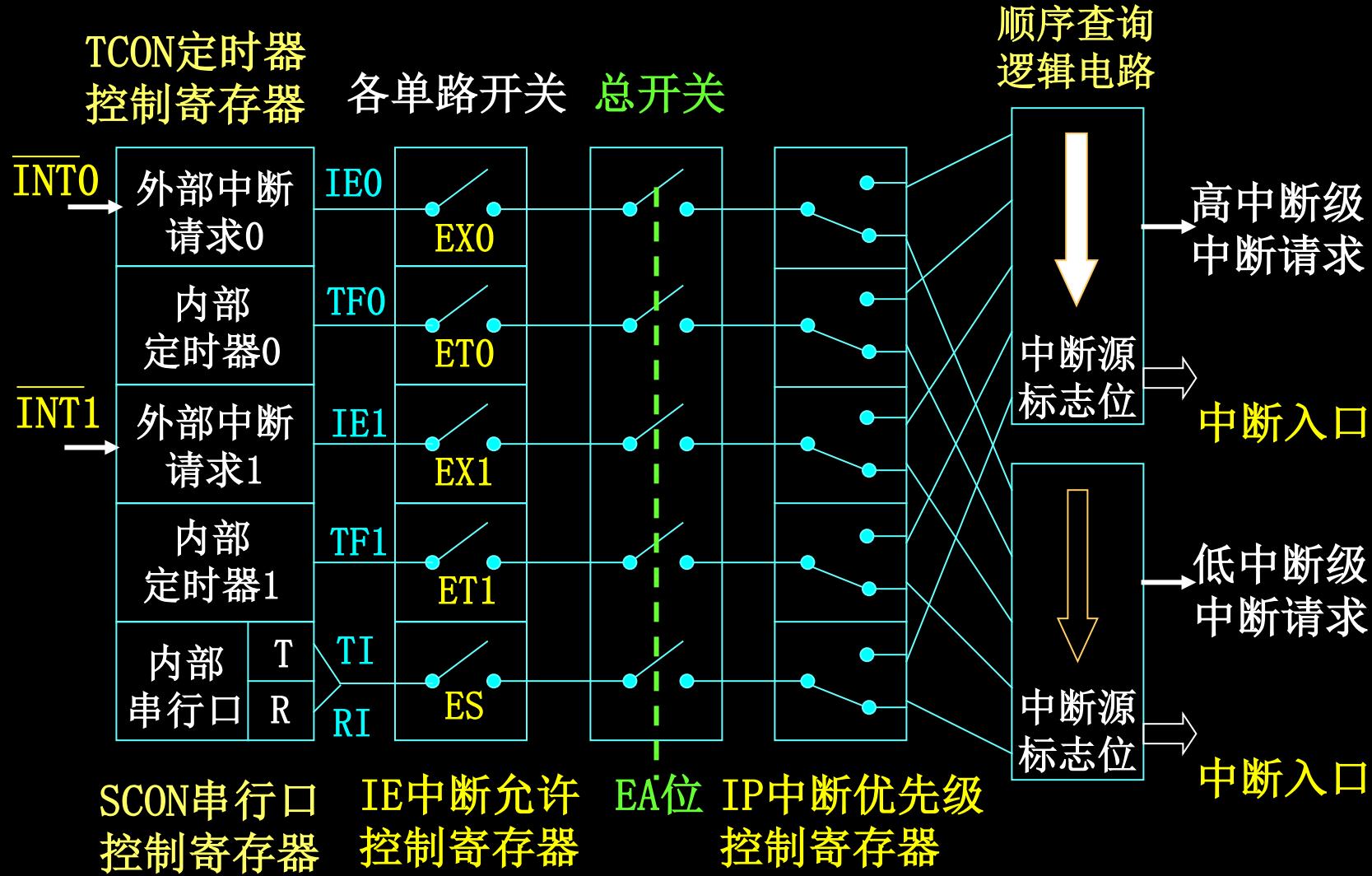
(4) 恢复现场。用堆栈指令将保存在堆栈中的数据弹出来，在恢复现场前要关中断，以防止现场被破坏。在恢复现场后应及时开中断。

(5) 返回。此时CPU将压入到堆栈的断点地址弹回到程序计数器，从而使CPU继续执行刚才被中断的程序。

## 6.2 MCS - 51 中断系统

- 总体结构
- 中断源
- 中断控制
- 中断处理

## 6.2.1 中断系统总体结构



注：各中断允许控制位=0, 开关断开； =1, 开关接通

## 6.2.2 中断源

### 8051 中断源

中 断 源	说 明
$\overline{\text{INT0}}$	P3. 2 引脚输入, 低电平/负跳变有效, 在每个机器周期的 S5P2 采样并建立 IE0 标志
定时器 0	当定时器 T0 产生溢出时, 置位内部中断请求标志 TF0, 发中断申请
$\overline{\text{INT1}}$	P3. 3 引脚输入, 低电平/负跳变有效, 在每个机器周期的 S5P2 采样并建立 IE1 标志
定时器 1	当定时器 T1 产生溢出时, 置位内部中断请求标志 TF1, 发中断申请
串行口	当一个串行帧接收/发送完时, 使中断请求标志 RI/TI 置位, 发中断请求

# MCS-51中断分为三类：

## 1.外部中断(IE0, IE1)

- 外部原因引起。
- 两个中断源，即外部中断0 和 外部中断1。其中断请求信号分别由引脚 INT0(P3.2) 和 INT1(P3.3) 引入。
- 外部中断请求有两种信号方式：电平方式和脉冲方式。可通过 **TCON** 的有关控制位进行规定。电平方式是低电平有效；而脉冲方式则是脉冲的下跳沿有效。

## 2. 定时中断(TF0, TF1)

- 定时中断是为满足定时或计数的需要而设置的。定时时间到或计数值满则产生中断请求。
- 中断请求是在单片机芯片内部发生的，无需在芯片上设置引入端。

## 3. 串行中断(TI, RI)

- 串行中断是为串行数据传送的需要而设置的。每当串行口接收或发送完一帧串行数据时，就产生一个中断请求。
- 串行中断请求也是在单片机芯片内部自动发生的，不需在芯片上设置引入端。

# 中断请求标志 (Flag)

IE0: 外部中断0 中断标志

IE1: 外部中断1 中断标志

TF0: 定时器/计数器0 中断标志

TF1: 定时器/计数器1 中断标志

TI: 串行口发送 中断标志

RI: 串行口接收 中断标志

## 1. 定时器控制寄存器TCON的格式 (88H )

Diagram of a 9-bit register with MSB at the top and LSB at the bottom. Bits are labeled from left to right: TF1, TR1, TFO, TR0, IE1, IT1, IEO, IT0.

## IT0: 外部中断0的触发方式控制位

0:低电平触发 1:边沿（负跳变）触发

IE0：外部中断0的中断请求标志

IT1、IE1 类似 IT0、IE0

TR0: 定时/计数器0的运行控制位 0:停止 1:运行

**TF0：** 定时/计数器0的溢出中断标志位

## TR1、TF1 类似 TR0、TF0

## 2. 串行口控制寄存器SCON的格式 (98H)

MSB	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	LSB

**SM0**和**SM1**: 串行口工作方式选择位

**SM2**: 多机通信使能位

**REN**: 接收允许位

**TB8**: 发送数据位8

**RB8**: 接收数据位8

**TI**: 串行口发送中断请求标志

**RI**: 串行口接收中断请求标志

MCS-51复位后，  
TCON、SCON中  
各位均清0！

## 6.2.3 中断控制

### 1. 中断允许控制

MCS - 51单片机有 5个（8052有 6个）中断源，为了使每个中断源都能独立地被允许或禁止，以便用户能灵活使用，CPU内部在每个中断信号的通道中设置了一个中断允许触发器，它控制CPU能否响应中断。只有对应的中断允许触发器被使能（置“1”），相应的中断才能得到响应。

# 中断允许控制寄存器IE (0A8H)

(MSB)				(LSB)			
EA	-	-	ES	ET1	EX1	ET0	EX0

EA : 中断总允许位

总控制位

EX0: 外部中断0 允许位

ET0: 定时器/计数器0 中断允许位

EX1: 外部中断1 允许位

ET1: 定时器/计数器1 中断允许位

ES : 串行口 中断允许位

分控制位

置1, 允许中断; 置0, 禁止中断。

当系统复位后, IE全部清0。

--两级控制

## 2. 中断优先级控制寄存器IP (0B8H)

(MSB)

(LSB)

—	—	—	PS	PT1	PX1	PT0	PX0
---	---	---	----	-----	-----	-----	-----

PX0: 外部中断0 优先级控制位

PT0: 定时器/计数器中断0 优先级控制位

PX1: 外部中断1 优先级控制位

PT1: 定时器/计数器1 中断优先级控制位

PS : 串行口 中断优先级控制位

置1, 为高优先级中断; 置0, 为低优先级中断。

当系统复位后, IP全部清0, 将所有中断源设置为低优先级中断。

当两个以上的中断源同时提出申请时，  
CPU到底响应哪个中断呢？

中断响应遵循两条规则：

- (1) 低优先级中断可以被高优先级中断所中断，反之不能；
- (2) 一种中断（不论哪个优先级）一旦得到响应，与它同级的中断不能再中断它。

当同时收到处于同一优先级的多个中断请求时，哪一个中断能得到响应呢？

取决于“内部查询次序”，即在每个优先级中，还有一个“内部优先级”。

中断源	中断标志位	同级内优先级
外部中断 0	IE0	最高
定时器 0 溢出中断	TF0	
外部中断 1	IE1	
定时器 1 溢出中断	IF1	
串行口中断	RI 或 TI	最低

同级内部优先级次序

例如，某程序中对寄存器IE、IP设置如下：

**MOV IE, # 8FH**

EA	-	-	ES	ET1	EX1	ET0	EX0
1			0	1	1	1	1

**MOV IP, # 06H**

-	-	-	PS	PT1	PX1	PT0	PX0
			0	0	1	1	0

则此时该系统中：

- CPU中断允许；
- 允许外部中断 0、外部中断 1、定时器/计数器 0、定时器 /计数器 1 提出的中断申请；

· 允许中断源的中断优先次序为：

定时器/计数器 0 > 外部中断 1 > 外部中断 0 > 定时器/计数器 1。

## 6.2.4 中断处理过程

一个完整的中断过程包括中断申请、中断响应、中断处理、中断返回。

### 一、中断响应

中断响应是指在CPU响应中断的条件得到满足后，CPU对中断源中断请求的回答。

#### 1、CPU响应中断的条件有：

- ① 首先要有中断源发出中断申请；
- ② CPU中断是开放的，即中断总允许位EA=1，CPU允许所有中断源申请中断；
- ③ 申请中断的中断源的中断允许位为1，即此中断源可以向CPU申请中断。

若有下列任何一种情况存在，则中断响应会被阻止：

- ① CPU正处在为一个同级或高级的中断服务中。
- ② 现行机器周期不是所执行的指令的最后一个机器周期。  
做此限制的目的在于使当前指令执行完毕后，才能进行中断响应，以确保当前指令的完整执行。
- ③ 当前指令是返回指令（RETI）或访问IE、IP的指令。  
按MCS-51中断系统的特性规定，在执行完这些指令之后，还应再继续执行一条指令，然后才能响应中断。

若存在上述任何一种情况，CPU将丢弃中断查询结果；否则，将在紧接着的下一个机器周期内执行中断查询结果，响应中断。若阻断条件结束时，中断标志已经消失，则这个被拖延了的中断请求就不会再得到响应。

## 2、中断响应过程

### (1) 外部中断请求采样

所谓中断请求采样，就是识别外部中断请求信号，并将其锁定在TCON的相应标志位中。每个机器周期的S5P2对中断请求引脚 INT0 (P3.2) 和 INT1 (P3.3) 进行采样。

- ✓ 对于 **电平触发** 方式的外中断请求，若采样为高电平，表明没有中断请求，TCON寄存器的外中断请求标志位IE0或IE1继续为0；若为低电平，则中断请求有效，把IE0或IE1置1。
- ✓ 对于 **脉冲触发** 方式的外中断请求，若在两个相邻机器周期采样到的是先高电平后低电平，则中断请求有效，把IE0或IE1置1；否则IE0或IE1继续为0。

**内部中断**直接置位相应的中断请求标志位，不存在中断请求采样问题。但同样存在从中断请求信号发生到中断请求标志位置位的过程。

采样解决的是外中断请求的锁定问题，即把有效的外中断请求信号锁定在各中断请求标志位中。CPU如何知道中断请求的发生？

对中断请求标志位进行查询。

## (2) 中断查询与响应

- 每一个机器周期的最后一个状态(S6)，按照优先级顺序对中断请求标志位进行查询。
- 如果查询到有标志位为1，则表明有中断请求发生，下一个机器周期的S1状态开始进行中断响应。首先将当前程序计数器PC的内容压栈，再将中断入口地址装入PC，使程序转向相应的中断区入口地址去执行。

# 中断服务程序入口地址表

中 断 源	中 断 矢 量
外部中断0	0003H
定时器 T0中断	000BH
外部中断1	0013H
定时器 T1中断	001BH
串行口中断	0023H

中断响应过程，相当于执行了一条隐含的调用指令（或称隐指令）LCALL。

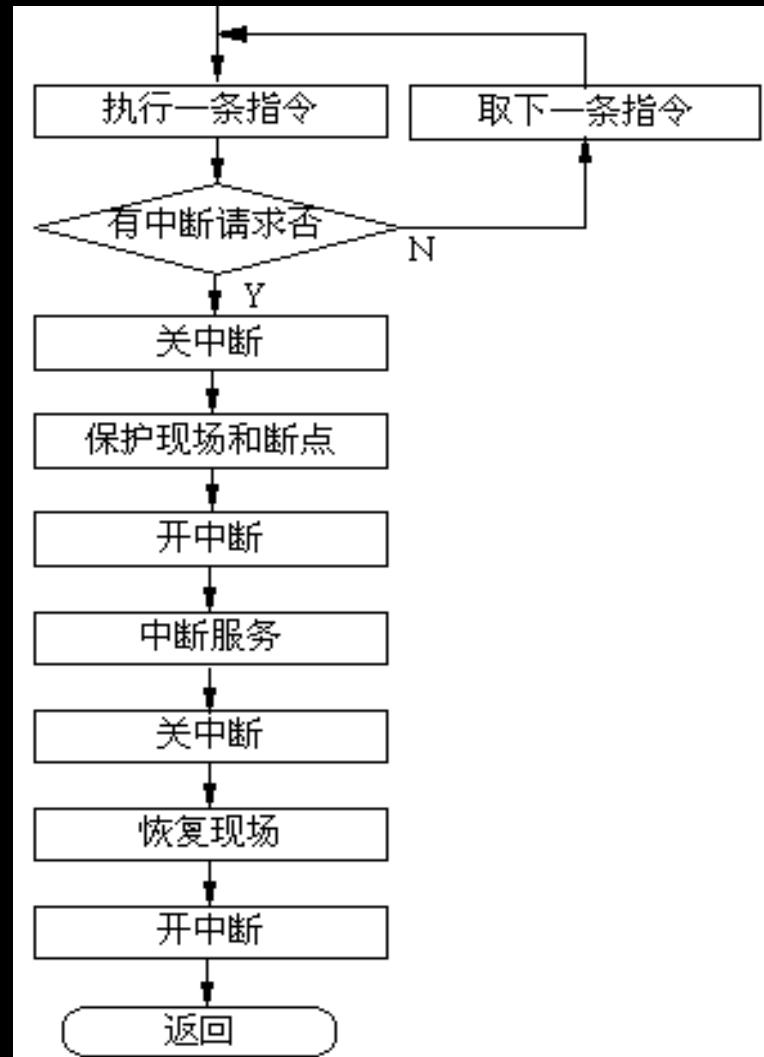
例如当TF0被置1且得到中断响应时，CPU就自动执行一条隐指令“LCALL 000BH”。

应当注意，在中断服务子程序的调用过程中，仅仅保存了PC的信息，其余的现场信息（如寄存器的值等）都要由编程人员通过软件来进行保护。

## 二、中断处理和中断返回

中断服务程序从入口地址开始执行，直到返回指令RETI为止。这个过程称为中断处理或中断服务。

通常为了使现场信息不受到破坏或者造成混乱，一般在保护和恢复现场时，CPU不能响应新的中断请求。



ORG 中断入口地址  
ZHDN: LJMP ITA

ORG XXX  
ITA: CLR EA ; 关中断  
PUSH ACC ; 保护现场  
PUSH R1  
|  
SETB EA ; 开中断  
| ; 中断服务  
CLR EA ; 关中断  
|  
POP R1 ; 恢复现场  
POP ACC  
SETB EA ; 开中断  
RETI ; 中断返回

中断服务程序的一般编写格式

# 三、中断请求的撤除

中断响应后，TCON或SCON中的中断请求标志应及时清除。否则就意味着中断请求仍然存在，会造成中断的混乱。

## 1. 定时中断硬件自动撤除

定时中断响应后，硬件自动把标志位（TF0或TF1）清0，因此定时中断的中断请求是自动撤除的。

## 2. 外部中断自动与强制撤除

对于边沿触发方式的外部中断，在中断响应后同样通过硬件自动地把标志位（IE0或IE1）清0，即中断请求的撤除也是自动撤除。

对于电平触发方式，情况则不同。光靠清除中断标志，并不能彻底解决中断请求的撤除问题，还需要解决外部电平问题。

## 3. 串行中断软件撤除

串行中断的标志位是TI和RI，对这两个中断标志内部硬件不进行自动清0。必须在中断服务程序中用软件来清除中断请求标志位TI或RI。

## 四、中断响应时间

中断响应时间是指从查询中断请求标志位到转向中断区入口地址所需的机器周期数。

- 最短： 3个机器周期
  - 中断请求标志位查询： 1T（恰好是指令的最后一个机器周期）
  - LCALL指令： 2T
- 最长： 8个机器周期
  - 刚好执行RETI或访问IE, IP时： 2T
  - 后续最长指令MUL、DIV： 4T
  - LCALL指令： 2T

## 6. 3 中断初始化

- 管理四个与中断有关的SFR(TCON, SCON, IE, IP)

中断初始化步骤：

- (1) 开中断
- (2) 对外部中断，需确定触发方式
- (3) 对多个中断源，需设定中断优先级

例：选用外部中断/INT0，电平触发，高优先级

SETB EA ; 置IE 中断总允许

SETB EX0 ; 置IE 外部中断0 允许中断

CLR IT0 ; 置TCON 外部中断0 为电平触发方式

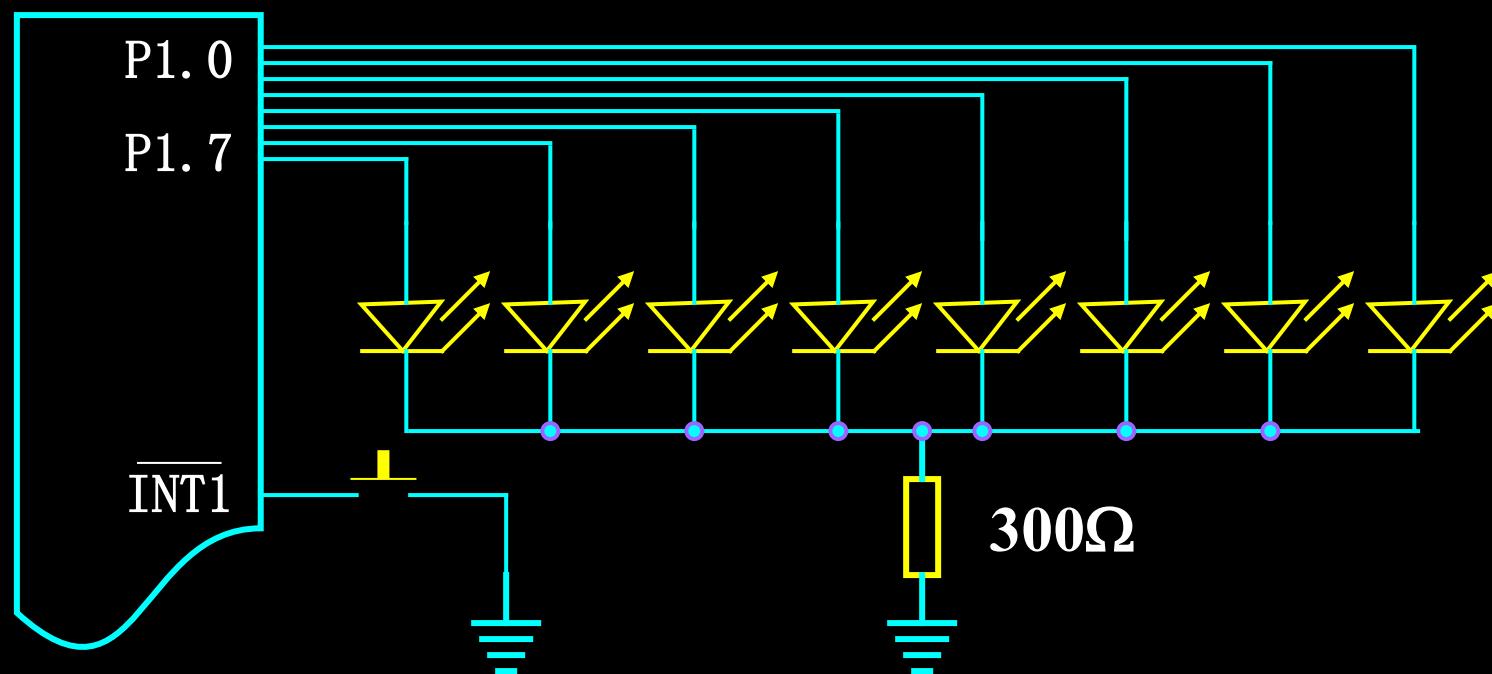
SETB PX0 ; 置IP 外部中断0 为高优先级中断

# 中断应用程序举例：

通过外部中断1，在中断服务中将R0寄存器里的内容左循环移动一位。

已知:  $(R0) = 01H$ , 要求采用边沿触发, 低优先级。

实际意义: 在INT1引脚接一个按钮开关到地, 每按一下按钮就申请一次中断, 中断服务则是: 依次点亮八盏灯中的一盏。



**ORG 0000H**

**LJMP MAIN**

**ORG 0013H ;外部中断1的中断矢量**

**LJMP INT**

**ORG 0050H**

**MAIN:** SETB EA ;置IE **开总中断允许“开关”**  
SETB EX1 ;置IE 外部中断1允许“开关”  
**CLR PX1** ;置IP外部中断1为**低**优先级中断  
SETB IT1 ;置TCON 外部中断1为边沿触发方式  
MOV R0, #01H ;给 R0 **寄存器赋初值**

**HERE:** SJMP HERE ;原地等待中断申请

**INT:** MOV A, R0 ;自R0**寄存器中取数**

RL A ;**左循环移动一次**

MOV R0, A ;**存回R0,备下次取用**

MOV P1, A ;**输出到P1口**

RETI ;**中断返回**

**中断服务程序**

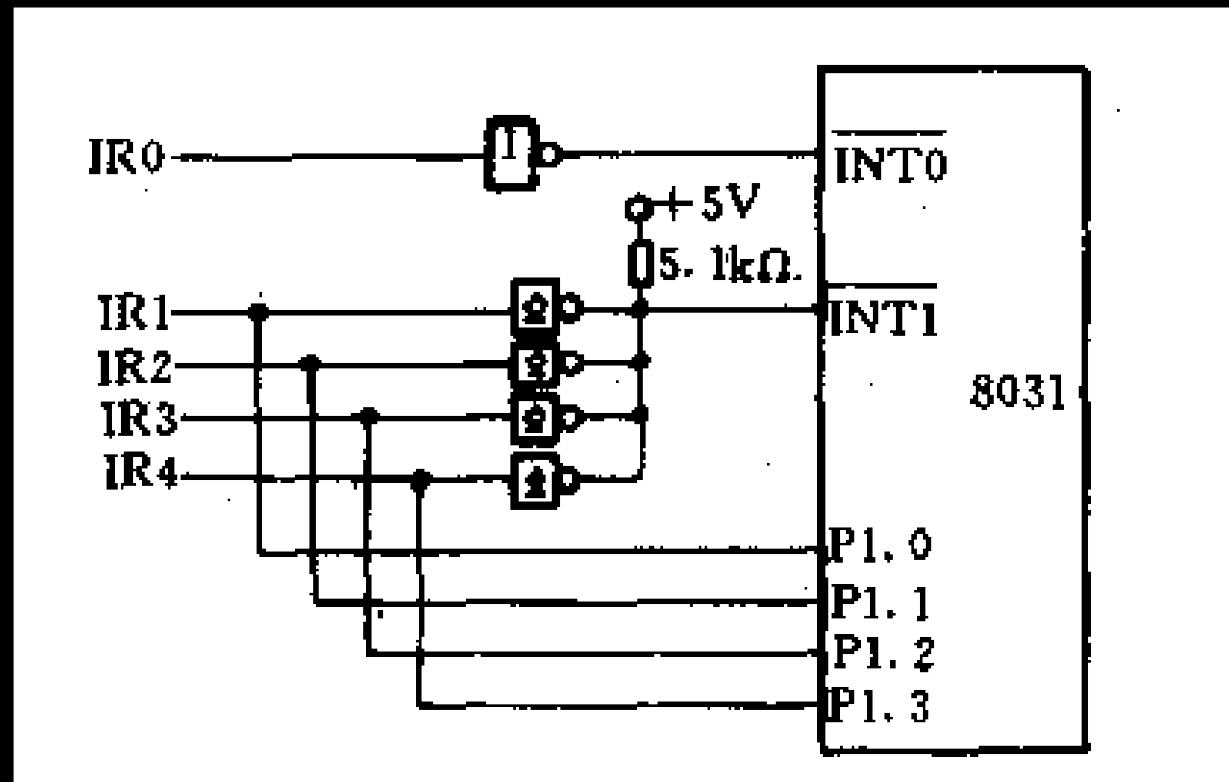
## 6.4 中断源扩展

### 1. 定时器作外部中断源

MCS-51有两个定时器/计数器，如果设定为计数方式，计数初值设为满量程，当T0或者T1计数器加1时就会产生溢出中断。因此，可以把**T0 (P3.4)** 和 **T1 (P3.5)** 作为外部中断请求输入线，而定时器的溢出中断标志**TF0**和**TF1**作为外部中断请求标志，**T0**和**T1**的中断入口地址作为扩展的外部中断源的中断入口地址。

## 2. 中断与查询结合的方式

如果系统中有多个外部中断请求源，可以按轻重缓急来进行排队，把优先级最高的中断源直接连外部中断0（INT0），其余的中断源输入经过集电极开路门‘线与’之后再和外部中断1（INT1）相连，同时接到I/O口，以备查询。



### 3. 采用优先编码器扩展外部中断源

在当外部中断源较多，而对响应速度又要求很高时，采用软件查询会无法满足实时的要求，就可以用**硬件**来实现对外部中断源的优先级排队。

# 本章小结

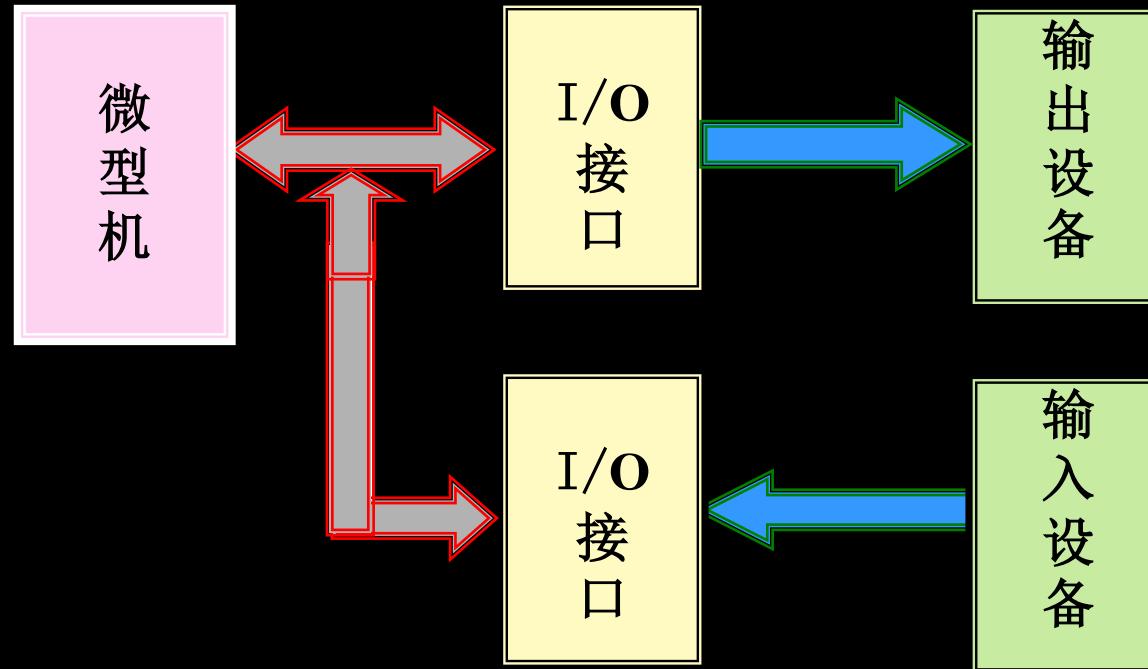
- 了解中断及其相关概念
- 了解中断的功能
- 掌握MCS-51中断系统的组成 ★
- 理解中断过程（中断申请、中断响应、中断处理及中断返回）
- 了解外部中断源的扩展方法

# 第7章 输入/输出

- ◆基本概念
- ◆MCS-51的并行I/O口
- ◆扩展并行接口常用芯片8255A



## § 7-1 概述 输入/输出的基本概念



I/O设备一般需要通过I/O接口与计算机连接

# 一、I/O接口

## 1 为什么需要I/O接口？

微机的外部设备多种多样，工作原理、驱动方式、信息格式、及工作速度方面彼此差别很大，所以它们不能与CPU直接相连，必须经过中间电路再与系统相连，这部分电路被称为I/O接口电路。

## 2 什么是I/O接口？

- I/O接口是位于系统与外设间、用来协助完成数据传送和控制任务的逻辑电路。

PC机系统板的可编程接口芯片、I/O总线槽的电路板（适配器）等都是接口电路。

I/O接口是CPU与外设之间传递信息和控制信号的部件。  
每个外设都需要与之对应的接口，而I/O接口的结构，因外设的不同而异。

常用的接口芯片：

**8255A-并行I/O**

**8251A-串行I/O**

**8253 -定时/计数器**

**8259A-中断控制器**

**8237A-DMA控制器**

**0809 -A/D转换器**

**0832 -D/A转换器**

## 二、I/O接口的基本功能

### (1) 对输入输出数据进行缓冲和锁存 (速度匹配)

#### 输出接口：锁存环节

微型机传送信息的速度要远远高于外设，当计算机向外设输出信息时，在输出接口电路中应设置数据锁存器，以便及时地把**CPU**输出的数据锁存起来，再由外设处理。

#### 输入接口：缓冲环节

**CPU**通过总线和多个外设相连，各个外设的数据线不能直接连到**CPU**的数据总线上，必须经过输入缓冲器。缓冲电路可以实现在同一时刻**CPU**只与被选中的一个外设交换信息。

## 二、I/O接口的基本功能

(2) 对信号的形式和数据的格式进行变换

串行-并行转换、A/D、D/A转换

电平转换：电平幅值或正/负逻辑转换

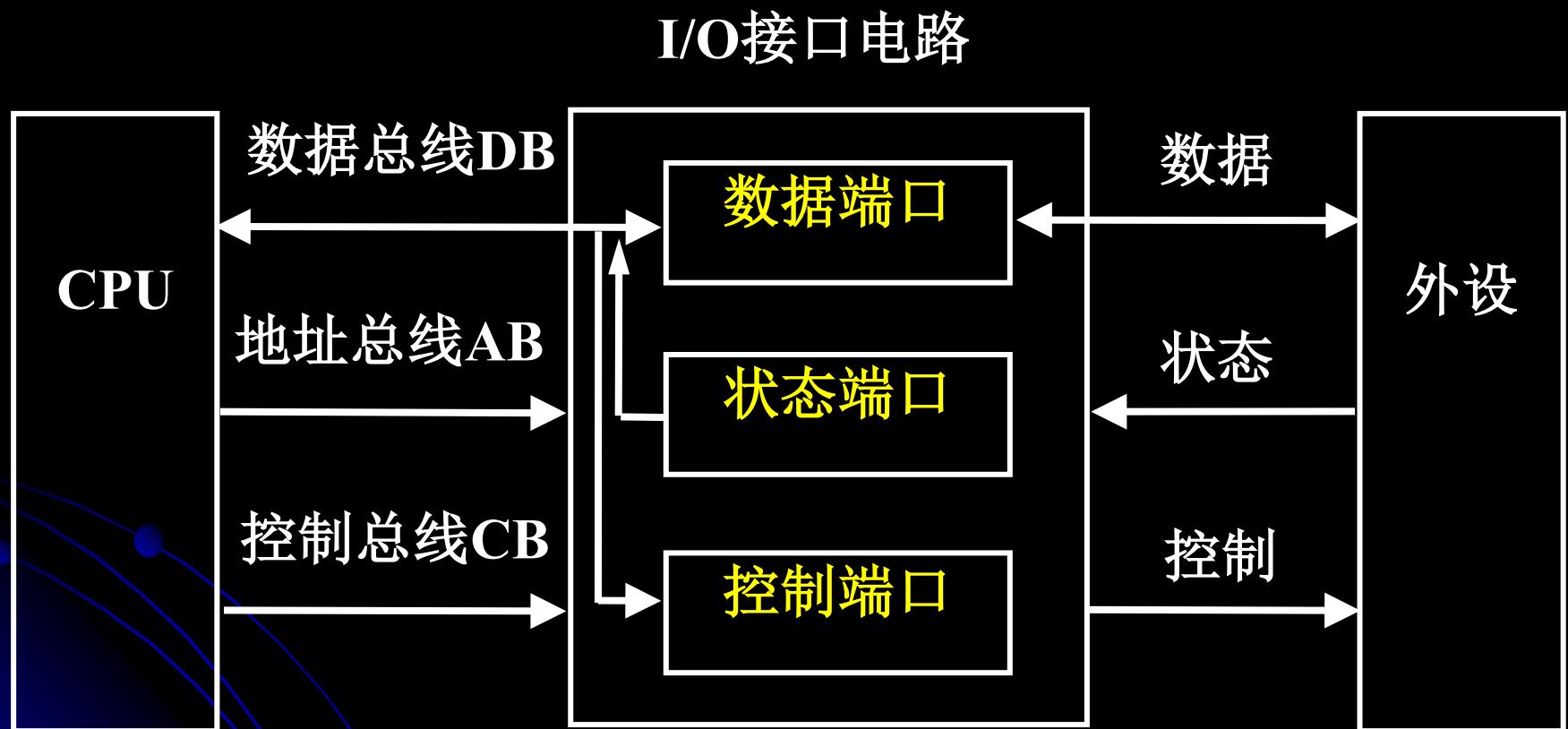
(3) 对I/O 端口进行寻址

指定外设端口，便于CPU对所寻址的外设进行读写。

(4) 与CPU和外设进行联络

为进行通信联络，I/O 接口电路需提供外设状态信息，以及CPU对外设的启停控制信号等。

### 三、 I/O接口的典型结构



## 四、I/O端口的编址方式

### 1、统一编址：

存储器与I/O端口统一编址，I/O端口共用存储器的地址空间，每个I/O端口视为一个存储单元。无专用I/O指令，用访问存储器的指令对I/O端口进行操作。

### 2、独立编址：

存储器与I/O端口分开编址，有专用I/O控制信号和I/O指令。I/O接口独立编址，不占用存储器的地址空间。端口寻址是通过地址总线，经过地址译码器确定地址的方式。

**8051统一编址、8086采用I/O端口独立编址**

## 五、微型机与外设的数据传送方式

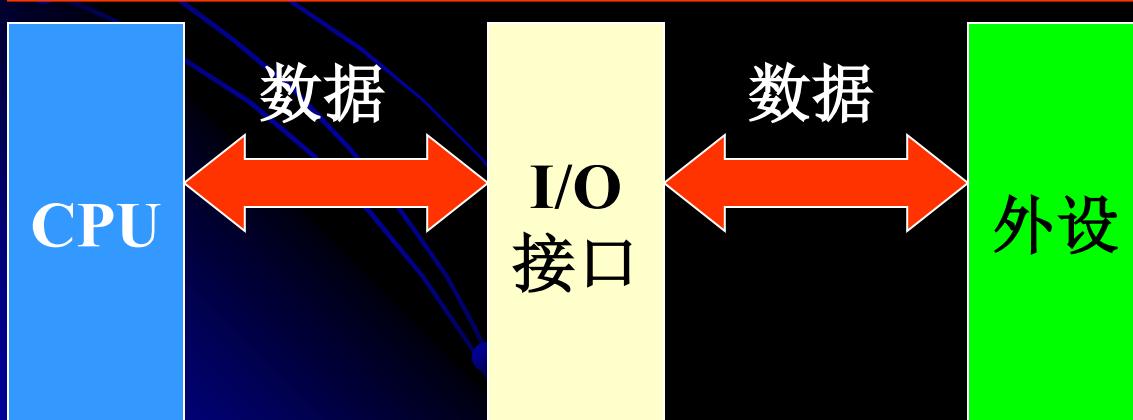
微型机和外设之间的数据传送方式有三种：

- 1、程序传送
- 2、中断传送
- 3、直接存储器存取传送

# 1. 程序传送: CPU与外设间的数据传送在程序控制下进行

## (1) 无条件传送方式

- ¶ 数据能够传送只取决于程序的执行, 而与外设的状态无关。即在需要传送时认为外设是处于“就绪”状态, 随时可进行数据传送。又称立即传送、同步传送。
- ¶ 适用于简单设备, 如LED数码管、按键或开关等。
- ¶ 无条件传送的接口和操作均十分简单。



```
MOV DPTR, #0100H  
MOVX A, @DPTR; 输入  
.....  
MOVX @DPTR, A; 输出
```

## (2) 条件 (查询式) 传送方式

◆CPU需要先了解 (查询) 外设的工作状态, 然后在外设可以交换信息的情况下 (准备就绪) 实现数据输入或输出。

**准备就绪——**

输入设备, 输入数据寄存器已满, 可供CPU读取;

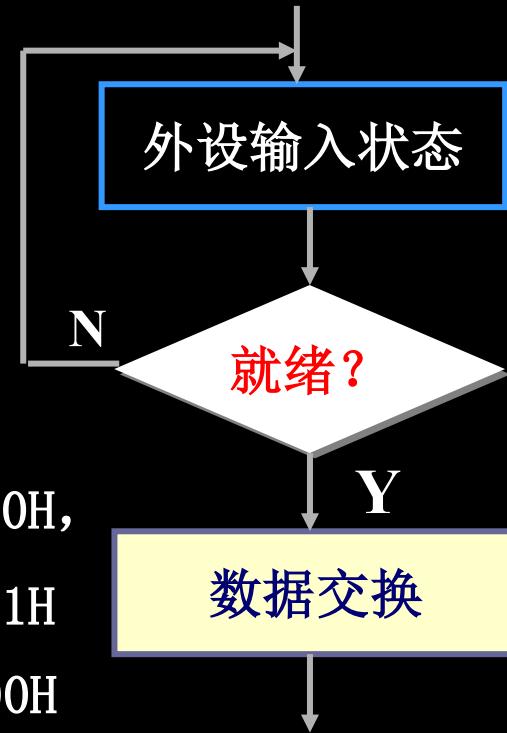
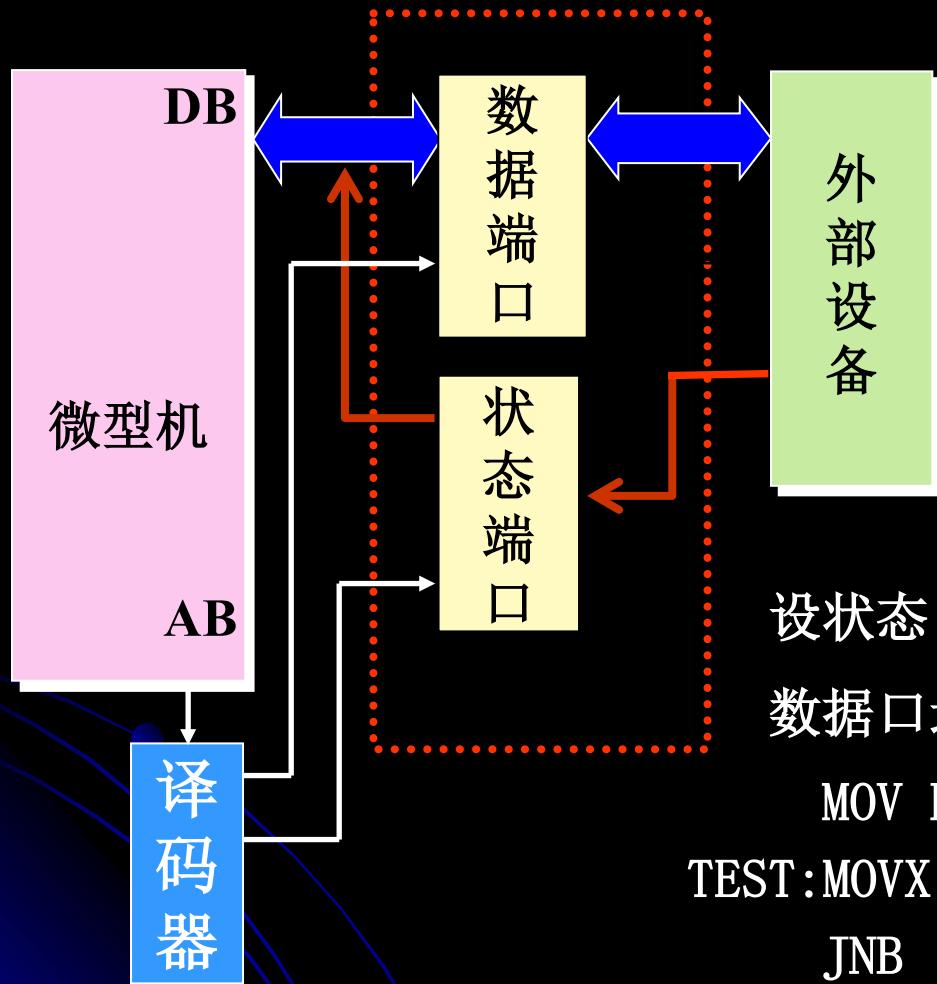
输出设备, 数据寄存器已空, 可接收CPU的新数据。

◆对多个外设的情况, 则CPU按一定顺序依次查询, 先查询

的外设将优先进行数据交换。

◆查询传送的特点是: 工作可靠, 适用面宽, 但传送效率低。

# 查询式程序框图



设状态口地址0100H,  
数据口地址为0101H

MOV DPTR, #0100H

TEST:MOVX A, @DPTR

; 读状态信息

JNB ACC. 7 TEST ; 测试状态

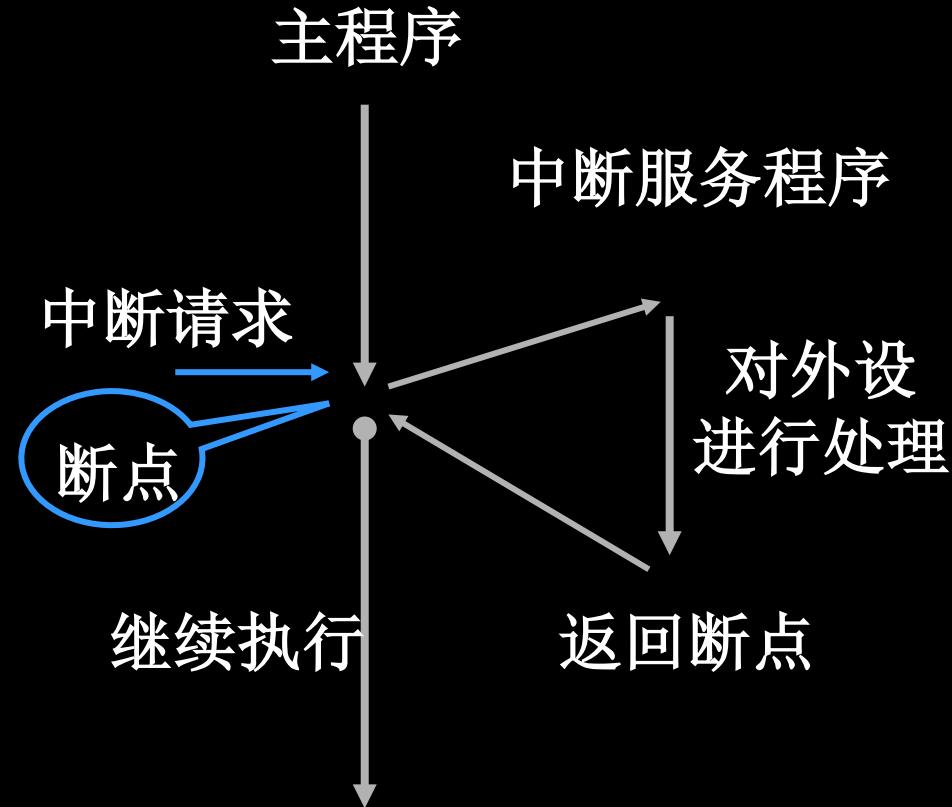
INC DPTR

MOVX A, @DPTR

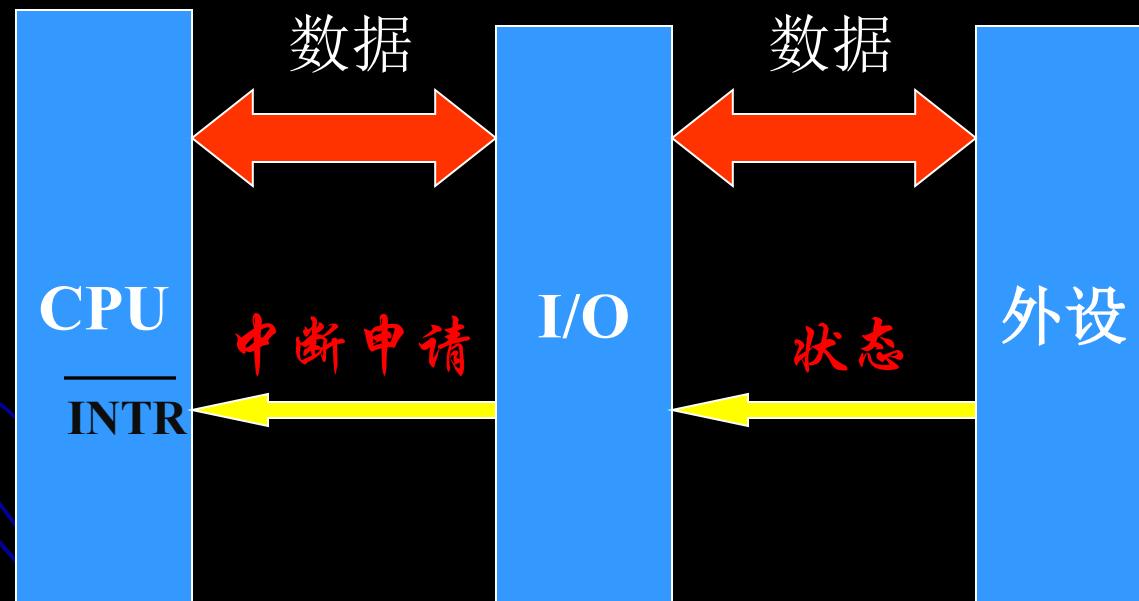
; 读输入数据

## 2. 中断传送

- ◆ 中断传送是一种CPU利用效率更高的传送方式
- ◆ 进行传送的中断服务程序是预先设计好的
- ◆ 中断请求是外设随机向CPU提出的
- ◆ CPU对中断请求的检测是有规律的：每个机器周期的S5P2进行检测



当外设准备就绪，向CPU发出中断请求信号。CPU暂停当前程序，转去为外设服务。当I/O操作结束，CPU仍继续被中断的工作。



### 3、DMA控制方式（直接存储器存取传送）

- 希望克服程序控制传送的不足：

输入时，外设→CPU→存储器

输出时，外设←CPU←存储器

对于高速输入/输出设备，速度达不到要求

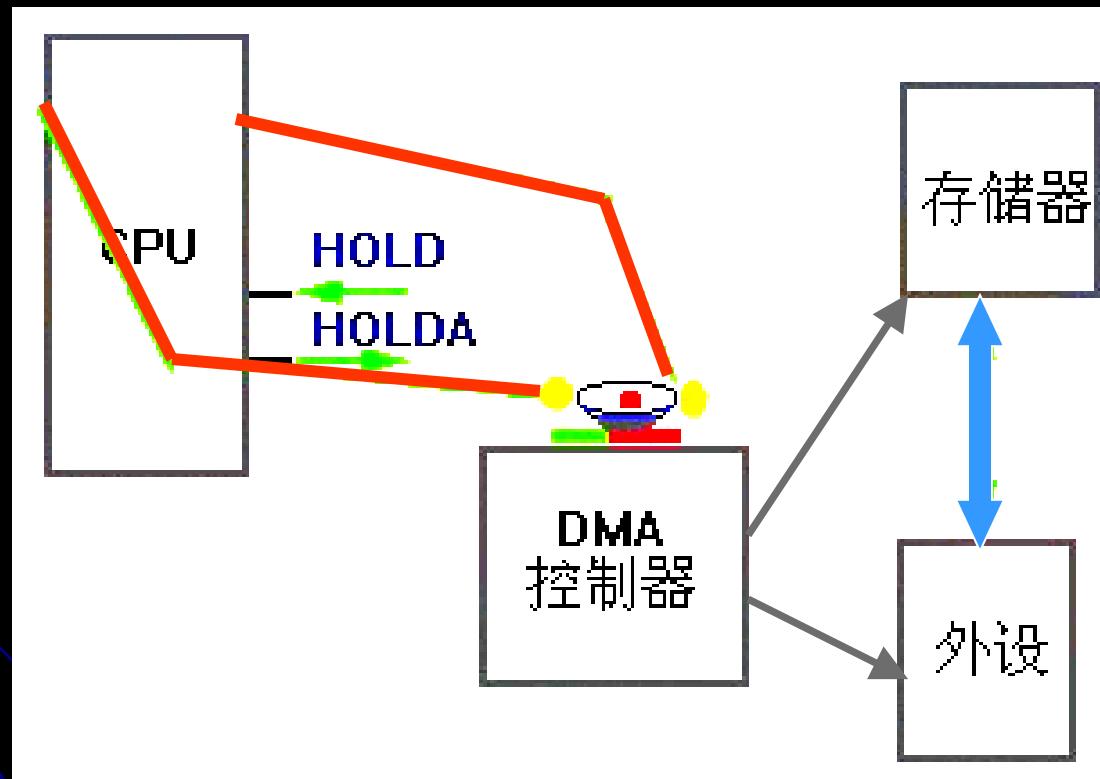
- 直接存储器存取DMA：

外设→存储器

外设←存储器

CPU释放总线，由DMA控制器管理

直接存储器存取主要用于微型机与高速外设进行大批量数据交换，由DMA控制器接管总线控制权，存储器与外设之间直接数据传输，不需CPU的介入。



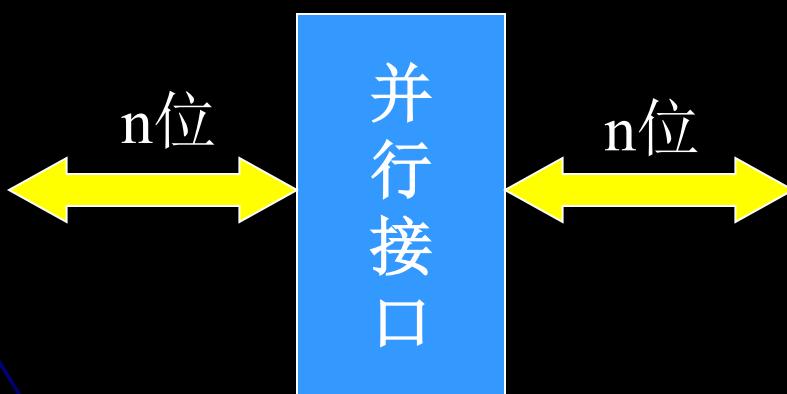
## MCS-51不支持DMA方式的数据传送

## § 7-2 MCS-51 的并行I/O口

### 并行通信

**n** 位数据同时传送，是最基本的信息交换方法，完成这种传送的接口称为并行接口。

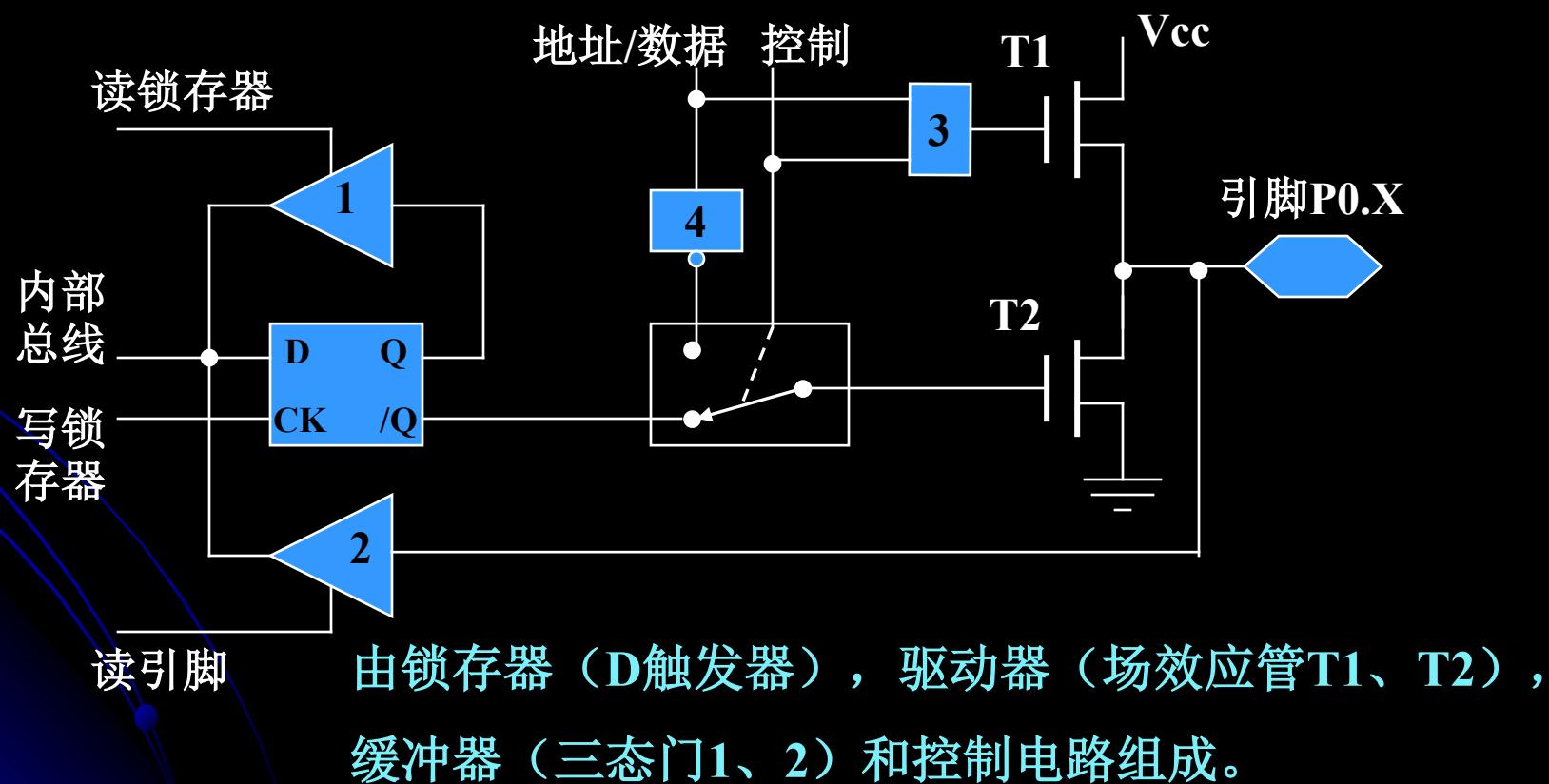
适合于外部设备与微机之间进行近距离、大量和快速的信息交换。例如：微机与并行接口打印机、磁盘驱动器



# MCS-51单片微机的P0口

- P0.0—P0.7: 双向I/O (内置场效应管上拉)

寻址外部存储器时分时作为双向8位数据口和输出低8位地址复用口；不接外部存储器时可作为8位双向I/O口使用。



# MCS-51单片微机的P0口

- 当控制信号为**0**时，封锁与门**(3)**，使**T1**截止，同时使多路开关接通输出锁存器的**Q**非端和场效应管**T2**的栅极。此时，**P0**切换到内部总线，即**I/O**状态。
- 当**P0**口作**输出口**时，在**CPU**发来的“写锁存器”（**D**触发器的时钟**CLK**输入）作用下，**CPU**通过内部总线把数据写入锁存器，内部总线和**P0**口的引脚线是同相的，故内部总线上的数据会立即输出到**P0**口的引脚上。

当**P0**口作**输入口**，有“读引脚”和“读锁存器”两种读操作。

- 读**引脚**，即读取**P0**口引脚上的外部输入信息。该口**D**锁存器必须先为**1**，使**T2**截止。当**CPU**执行一条由口输入的指令时，“读引脚”脉冲把三态门**2**打开，引脚上的外部数据通过三态输入缓冲器**2**传送到内部总线。
- 读**锁存器**，即读取口锁存器的状态。由“读锁存器”信号把三态门**1**打开，锁存器**D**的输出**Q**经三态输入缓冲器**1**传送到内部总线。

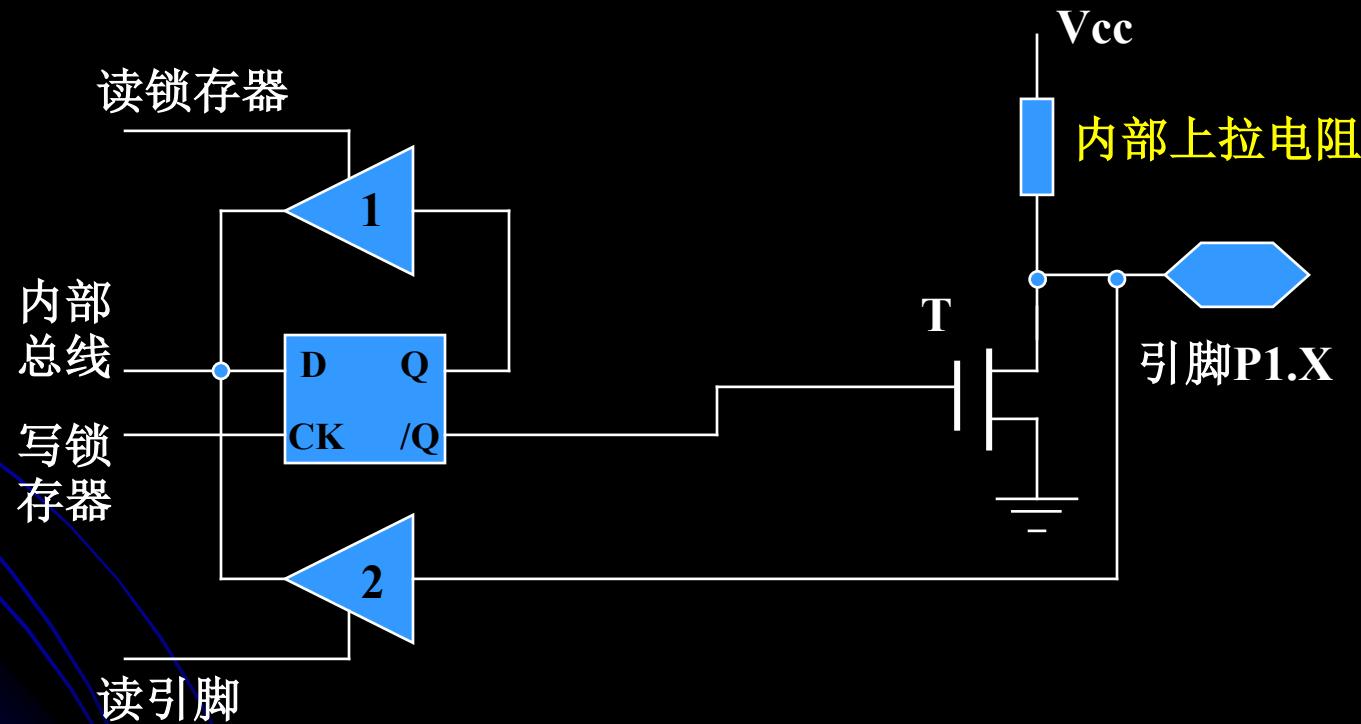
# MCS-51单片微机的P0口

- 当控制信号为**1**时，使多路开关断开输出锁存器的**Q**非端和**T2**栅极的连接，地址/数据线经反相器（**4**）与**T2**的栅极相连。此时，**P0**口工作在**地址/数据总线**状态。
- 当输出地址/数据线为**0**时，封锁与门（**3**），使**T1**截止，通过反相器（**4**）使**T2**导通，**P0**的这一位引脚上面就是**0**。
- 当输出地址/数据线为**1**时，打开与门（**3**），使**T1**导通，同时通过反相器（**4**）使**T2**截止，**P0**的这一位引脚上面就是**1**。
- 当从**P0**口引脚输入数据信息时，输入指令将使引脚与内部总线直通，即输入信号会从引脚通过输入缓冲器（三态门**2**）进入内部总线。

# MCS-51单片微机的P1口

- P1.0—P1.7: 准双向I/O口 (内置了上拉电阻)

P1口由锁存器 (D触发器)、驱动器 (场效应管T) 和缓冲器 (1、2两个三态门) 构成。

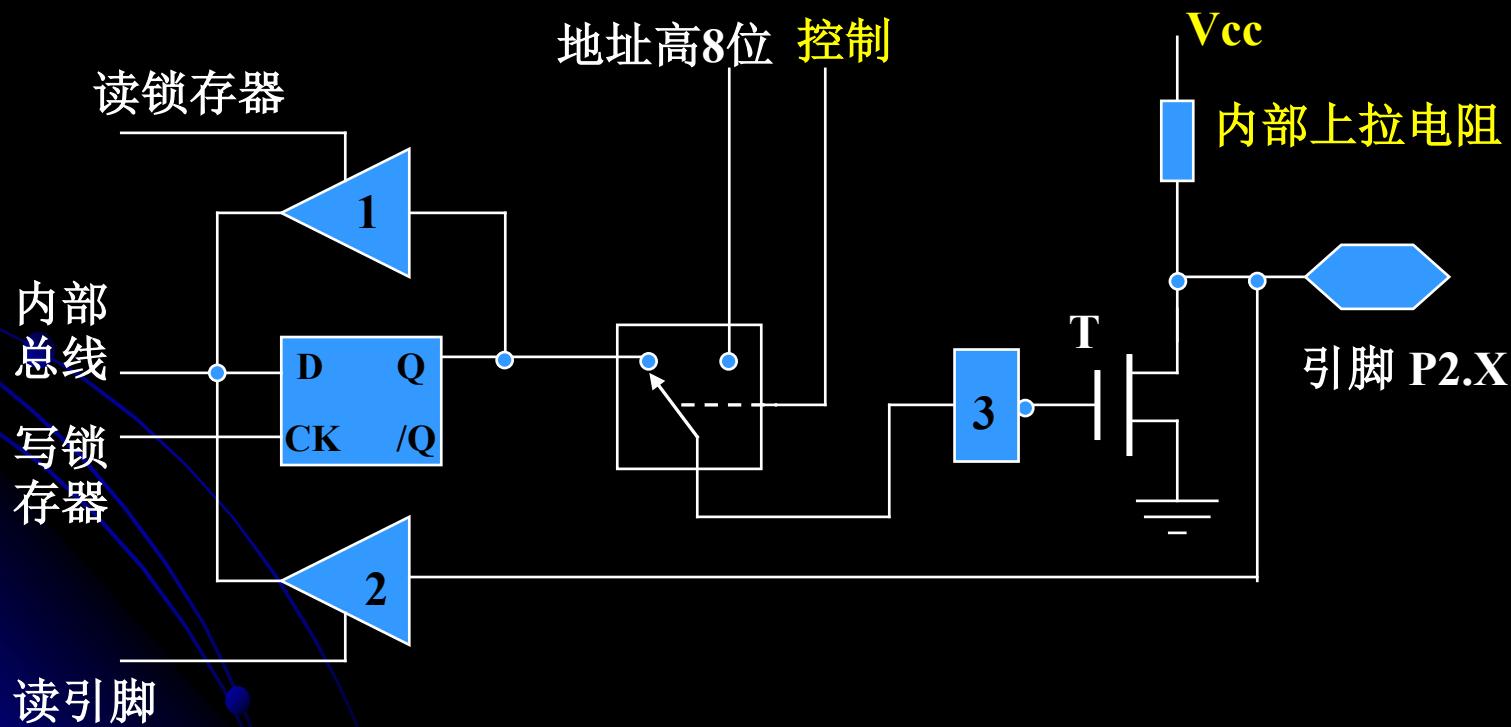


## MCS-51单片微机的P1口

- P1口的每一位都可以分别定义为输入线或者输出线。
- 作输出线时，如果把“1”从内部总线写入某一位的锁存器，则Q非端为0，使驱动器（场效应管T）截止，这一位的输出引脚由内部上拉电阻拉成高电平，引脚就输出“1”。
- 作输出线时，如果把“0”从内部总线写入某一位的锁存器，则Q非端为1，使驱动器（场效应管T）导通，这一位的输出引脚就输出低电平“0”。
- P1口的某一位作输入线时，必须先向对应的锁存器写入1，Q非端为0，关断驱动器（场效应管T截止），这时这一位的引脚可以由内部上拉电阻拉成高电平，也可以由外部电路拉成低电平。

# MCS-51单片微机的P2口

- P2.0—P2.7: 准双向I/O (内置了上拉电阻)  
寻址外部存储器时输出高8位地址; 不接外部存储器时可作为8位准双向I/O口使用。



## MCS-51单片微机的P2口

- 当控制信号为**0**时，多路开关连接锁存器的**Q**输出端，**P2**口引脚作通用**输出口**。如果此时内部总线上有数据为“**1**”，经过反相器（**3**），使场效应管**T**截止，这一位的输出引脚由内部上拉电阻拉成高电平，引脚就输出“**1**”。
- 如果内部总线上有数据为“**0**”，经过反相器（**3**），使场效应管**T**导通接地，这一位的输出引脚就输出“**0**”。
- **P2**口的某一位作**输入线**时，必须先向对应的锁存器写入**1**，关断驱动器（场效应管**T**截止），这时这一位的引脚可以由内部上拉电阻拉成高电平，也可以由外部电路拉成低电平。

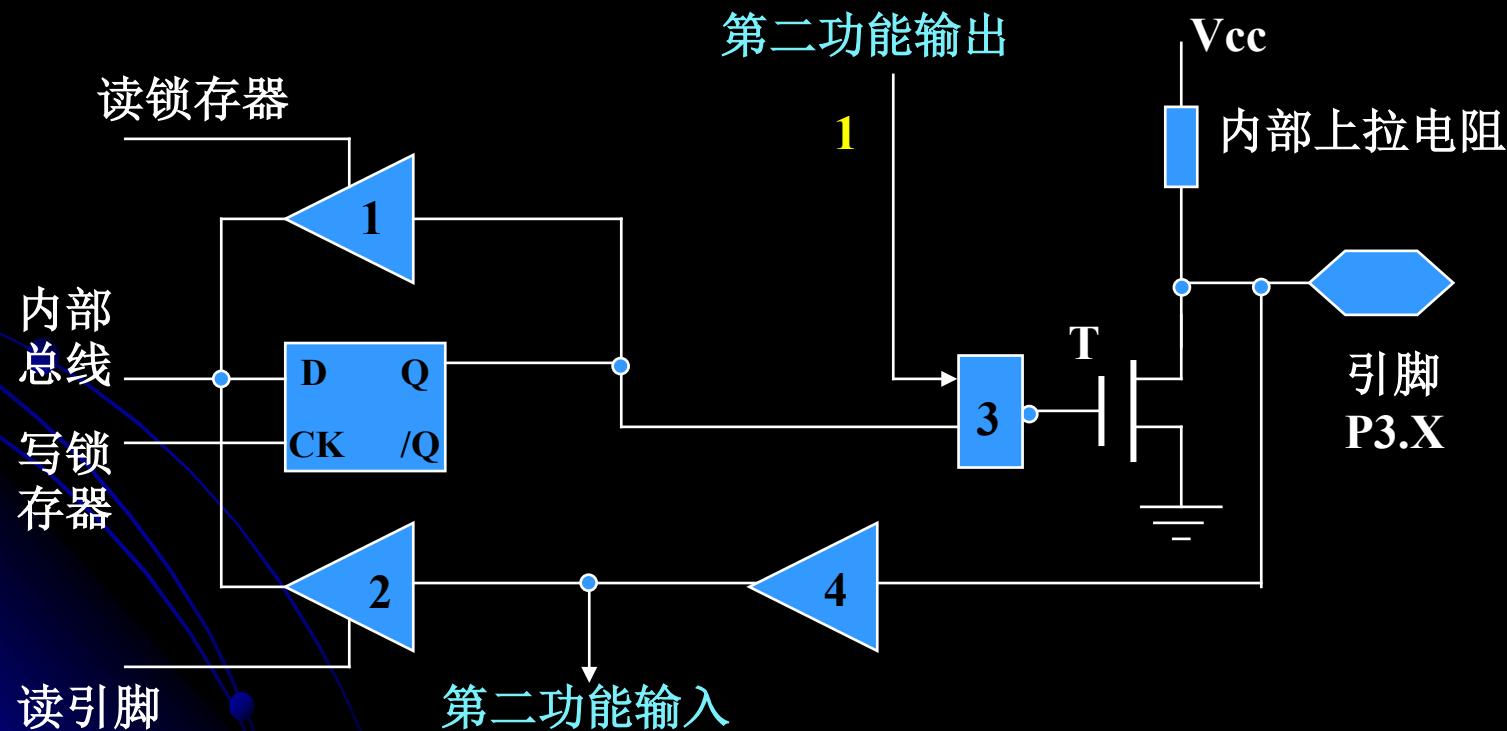
## MCS-51单片微机的P2口

- 当控制信号为**1**时，多路开关连接**地址线控制端**，**P2口**引脚作高**8位地址****A8-A15**输出口。当输出地址为**1**时，经过反相器（3），使场效应管T截止，这一位的输出引脚由内部上拉电阻拉成高电平，引脚就输出“**1**”。
- 当输出地址为**0**时，经过反相器（3），使场效应管T导通接地，这一位的输出引脚就输出“**0**”。

# MCS-51单片微机的P3口

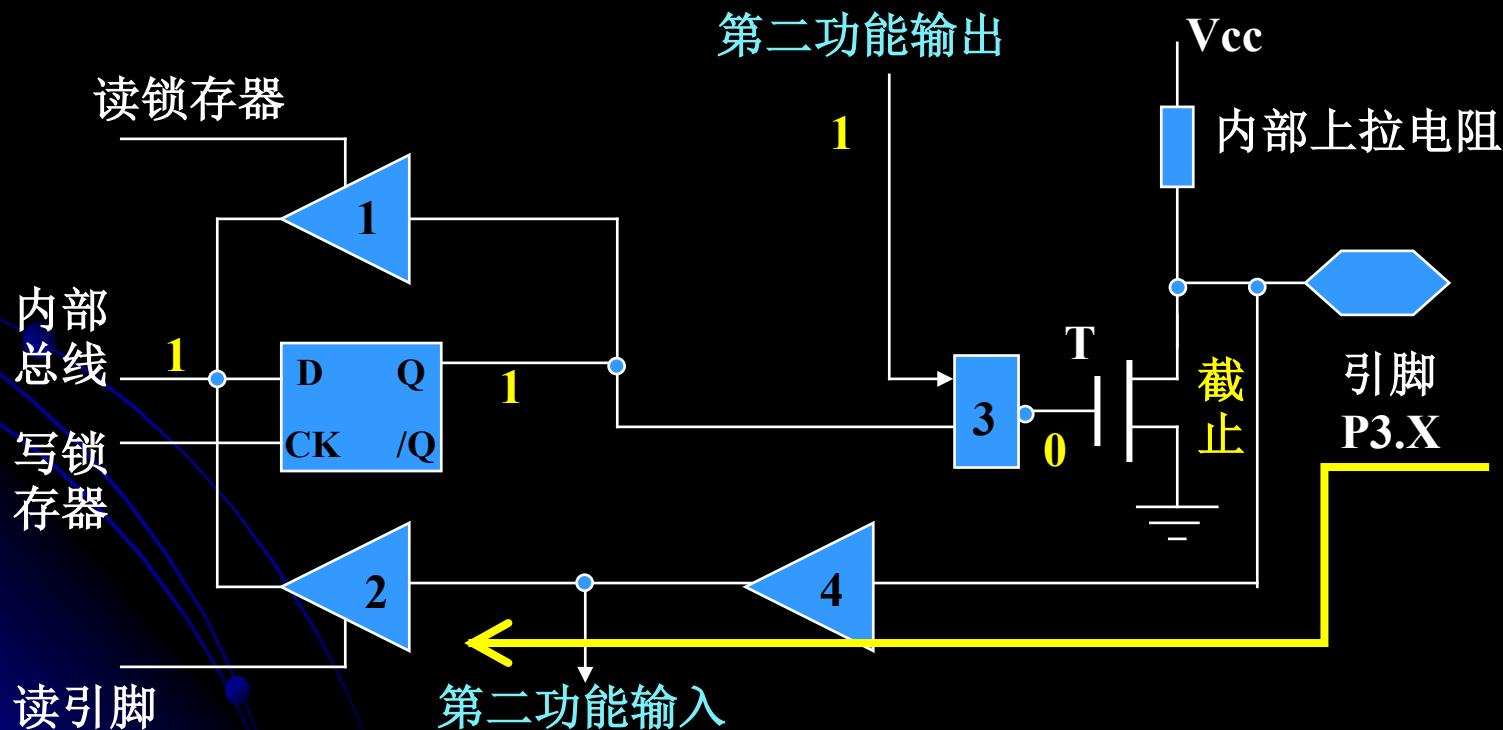
- P3.0—P3.7: 双功能口 (内置了上拉电阻)

**P3口具有特定的第二功能。**第一功能作通用的I/O口，此时第二功能输出保持高电平，打开与非门（3），锁存器的输出可以通过与非门送到驱动器输出引脚。



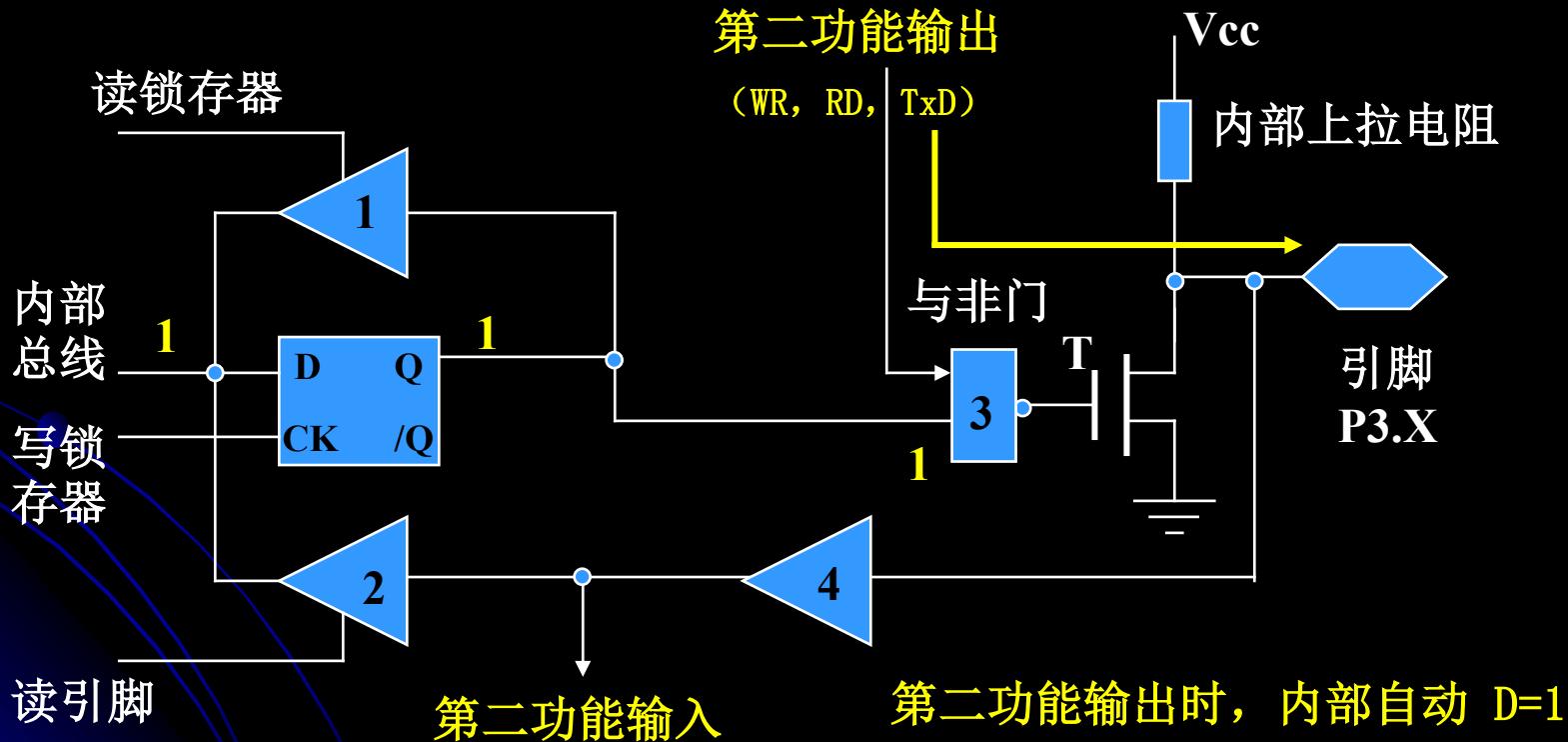
# MCS-51单片微机的P3口

- 输入时，必须先对锁存器置1，与非门（3）为0，关断驱动器（场效应管T截止），这时这一位的引脚可以由内部上拉电阻拉成高电平，也可以由外部电路拉成低电平。CPU读P3脚状态，实际上就是读取外部电路的输入信息。引脚上的外部信号通过三态输入缓冲器送入内部总线。



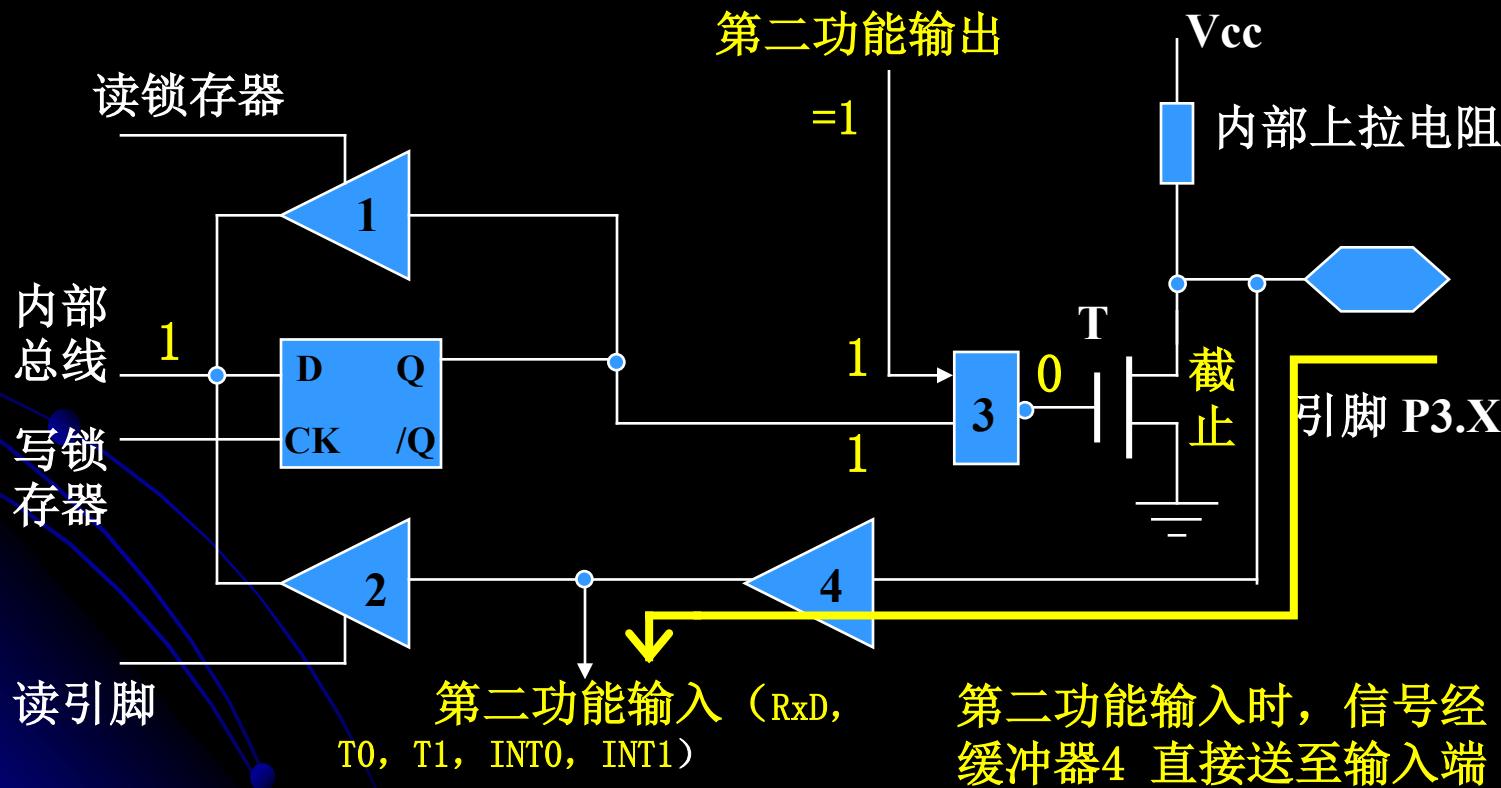
# MCS-51单片微机的P3口

- P3口用于第二功能输出时，内部总线自动置为D=1，锁存器输出Q端为1，打开与非门（3），第二功能输出内容通过与非门和驱动器送至引脚。



# MCS-51单片微机的P3口

- P3口用于第二功能输入时，第二功能输出保持高电平，内部总线也保持高电平，与非门（3）为0，关断驱动器（场效应管T截止），外部输入信号经缓冲器4直接送到第二功能输入端。



## P3口 第二功能表

引脚	第二功能
P3.0	RxD: 串行口接收数据输入端
P3.1	TxD: 串行口发送数据输出端
P3.2	INT0: 外部中断申请输入端 0
P3.3	INT1: 外部中断申请输入端 1
P3.4	T0: 外部计数脉冲输入端 0
P3.5	T1: 外部计数脉冲输入端 1
P3.6	WR: 写外设控制信号输出端
P3.7	RD: 读外设控制信号输出端

## MCS-51单片机的4个8位的I/O口

- ★ P0.0–P0.7 8位数据口和输出低8位地址复用口 (双向口)
- ★ P1.0–P1.7 通用I/O口 (准双向口)
- ★ P2.0–P2.7 输出高8位地址  
(用于寻址时是输出口；不寻址时是准双向口)
- ★ P3.0–P3.7 具有特定的第二功能 (准双向口)

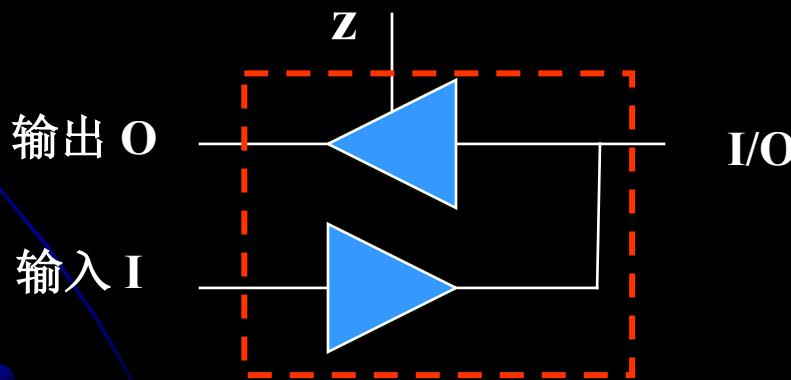
**注意：**在不外部扩展ROM/RAM时，P0～P3均可作通用I/O口使用

# 为何称P0口是双向口，而其他I/O口为准双向口？

P0无上拉电阻，其输出驱动器是由两个场效应管T1、T2组成的三态门，P0口作为系统数据总线使用时，可以保证在数据传送时芯片内外接通；不进行数据传送时，芯片内外处于隔离状态。因此P0口是真正的双向I/O口。

而P1、P2、P2口内部有上拉电阻，代替P0口的场效应管T1，输出驱动器不是三态的，称为准双向I/O口。

三态门



Z=1时，只能输入  
Z=0时，可输出

# I/O口数据输入、输出指令

数据输出方式  
(CPU总线上的数  
据输出到I/O端口，  
即往I/O口送数据)

MOV P <sub>0</sub> , A	;累加器 A 中内容送 P <sub>0</sub> 口
ORL P <sub>0</sub> , #data	;P <sub>0</sub> Vdata 送 P <sub>0</sub> 口
ANL P <sub>0</sub> , A	;P <sub>0</sub> AA 送 P <sub>0</sub> 口
XOR P <sub>0</sub> , #data	;P <sub>0</sub> $\oplus$ data 送 P <sub>0</sub> 口

读锁存器方式  
(读锁存器状态)

MOV A, P <sub>1</sub>	;P <sub>1</sub> 锁存器中数据送 A
MOV R <sub>1</sub> , P <sub>1</sub>	;P <sub>1</sub> 锁存器中数据送 R <sub>1</sub>
MOV 20H, P <sub>1</sub>	;P <sub>1</sub> 锁存器中数据送 20H
MOV @R <sub>0</sub> , P <sub>1</sub>	;P <sub>1</sub> 锁存器中数据送 (R <sub>0</sub> )

读端口引脚方式  
(读引脚上的外部  
输入信息)

MOV P <sub>1</sub> , #0FH	;使 P <sub>1</sub> 口低 4 位锁存器置位
MOV A, P <sub>1</sub>	;读 P <sub>1</sub> 口低 4 位引脚线信号

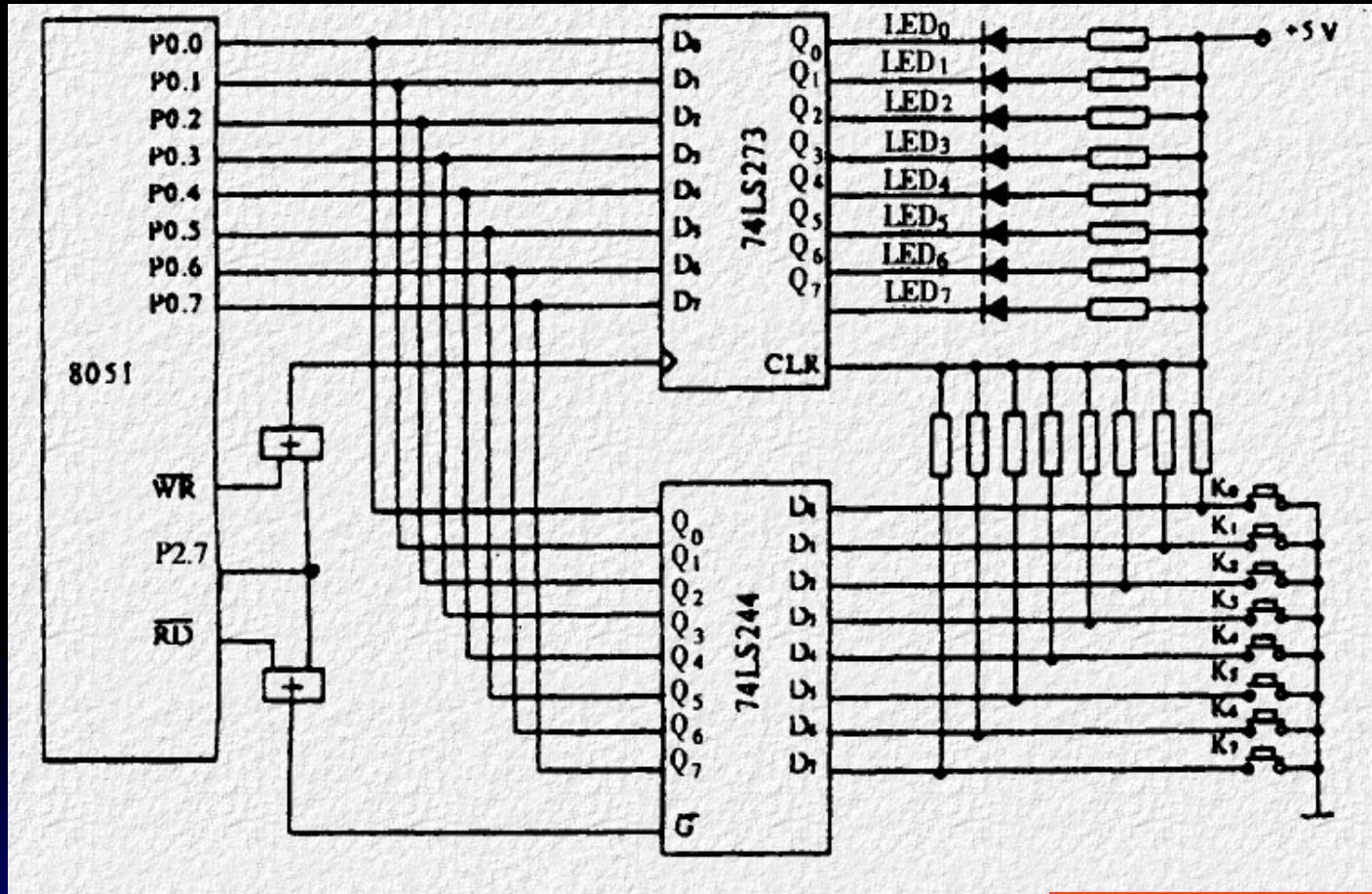
# I/O口的扩展

MCS-51系统中，经常需要外部扩展存储器，可能出现I/O口不够用的情况，就要进行I/O口的扩展。

扩展I/O口所用的芯片主要有不可编程接口芯片和通用可编程接口芯片两大类。

MCS-51系统中，I/O口与外部数据存储器统一编址。用访问片外RAM的指令（MOVX）来访问片外的I/O口，对其进行读/写操作。

# 例：用TTL芯片扩展并行I/O口（不可编程）



简单 I/O 接口扩展电路

电路实现的功能：当按下任意一个按键，对应的LED就点亮。采用74LS244作为扩展输入，74LS273作为扩展输出。

P2.7 (7FFFH)  
作为输入、输出  
的控制信号

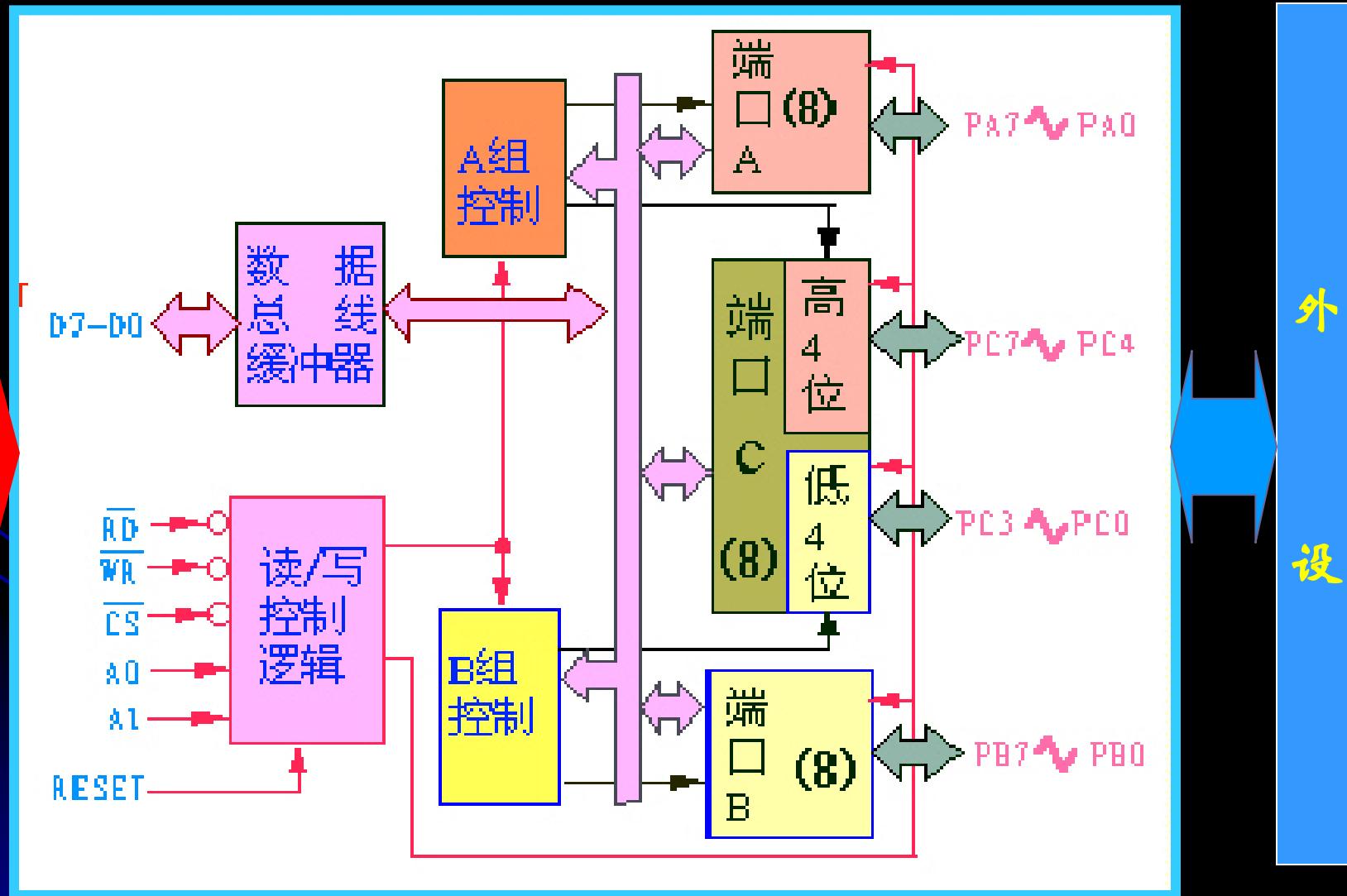
当P2.7和RD同时有效时，通过244输入按键的数据；当P2.7和WR同时有效时，P0口通过273输出数据显示。

LOOP:	MOV	DPTR, #7FFFH
	MOVX	A, @DPTR
	MOVX	@DPTR, A
	SJMP	LOOP

# § 7-3 可编程并行通信接口芯片8255A

## 一、8255A的结构和外引脚

### 1、结构（主要由三个部分组成）



(1) 外部接口部分：8255A具有3个8位并行I/O端口A、B、C

① 三个端口均可做I/O使用

② 又可分为2组控制

A组：A口和C<sub>4~7</sub>

B组：B口和C<sub>0~3</sub>

- 端口A：PA0~PA7 具有一个8位数据输出锁存器/缓冲器和一个8位数据输入锁存器，用作数据端口时，输入输出均可锁存
- 端口B：PB0~PB7 具有一个8位数据输出锁存器/缓冲器和一个8位数据输入缓冲器，用作数据端口时，仅能锁存输出数据
- 端口C：PC0~PC7
  - 可作数据、状态和控制端口
  - 分两个4位，每位可独立操作
  - 既可作独立的I/O口，又可配合A、B口工作

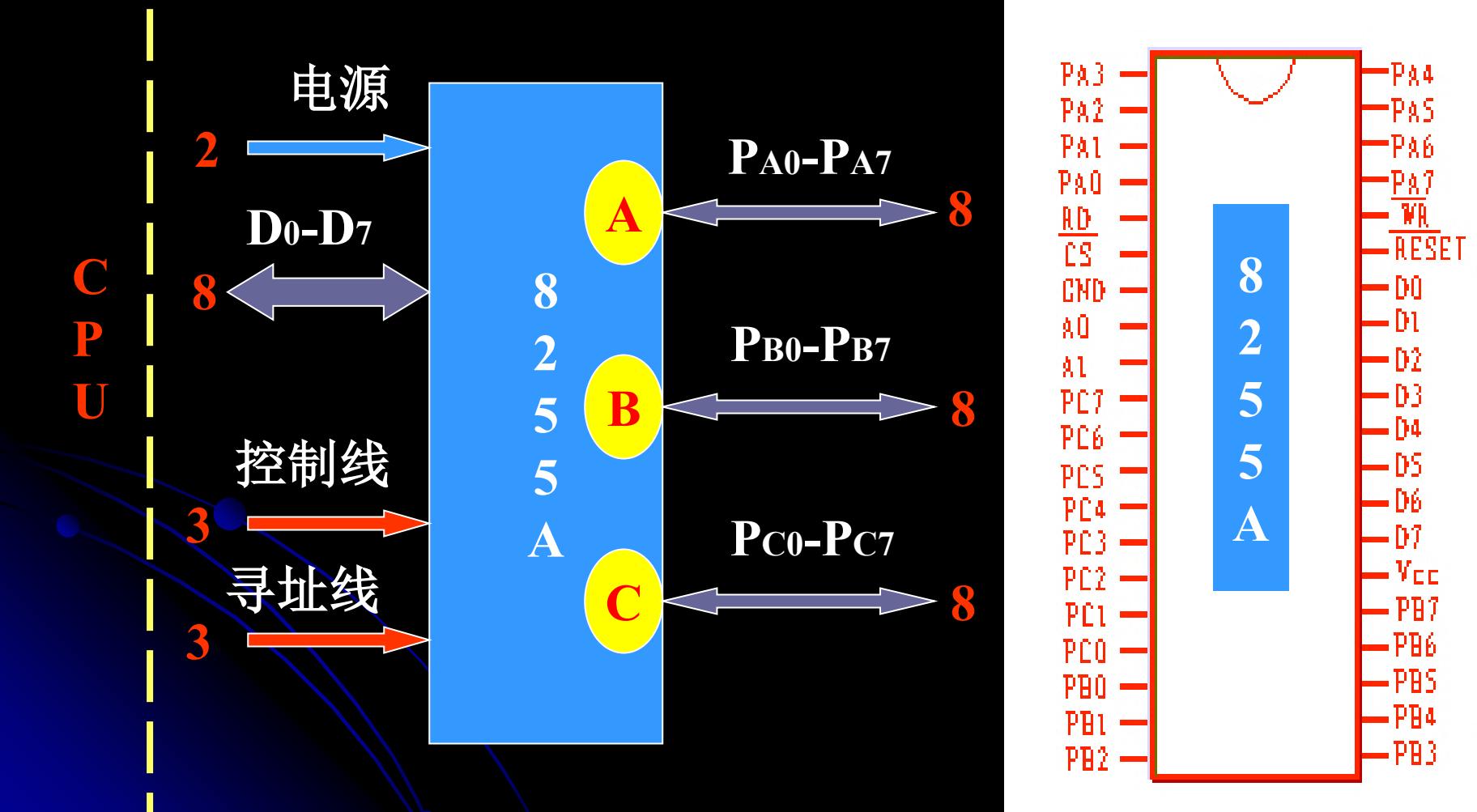
## (2) 总线接口部分（数据总线缓冲器）：

是一个双向三态的8位数据缓冲器，完成与数据总线（DB）的连接，实现MCS-51系统和8255A之间的数据传送。

## (3) 读写控制逻辑部分：

- ① 接收地址总线AB、译码电路命令，完成8255A、B、C三个端口及控制口的选择。
- ②接收控制总线CB命令，完成规定的操作。

## 2、外引脚 (40脚)



## 说明：

### (1) 控制线（与微处理器接口）

**RD**: 读信号 (CPU  $\leftarrow$  8255A的数据)

**WR**: 写信号 (CPU  $\rightarrow$  8255A的数据或控制字)

**RESET**: 复位信号 (清内部寄存器, A、B、C口为输入方式)。

### (2) 寻址线（用于选择8255A的三个端口和控制寄存器）

**CS**: 片选信号 (访问8255A时, 该线有效)

**A<sub>1</sub>A<sub>0</sub>**: 端口选择信号

CS	A <sub>1</sub>	A <sub>0</sub>	选择
0	0	0	A口
0	0	1	B口
0	1	0	C口
0	1	1	控制口

# 8255A的并行接口的基本操作

A1	A0	RD	WR	CS	基本操作
0	0	0	1	0	端口 A 数据送 DB
0	1	0	1	0	端口 B 数据送 DB
1	0	0	1	0	端口 C 数据送 DB
1	1	0	1	0	非法
0	0	1	0	0	DB 数据送 端口 A
0	1	1	0	0	DB 数据送 端口 B
1	0	1	0	0	DB 数据送 端口 C
1	1	1	0	0	DB 数据送 控制口
X	X	X	X	1	DB 高阻态
X	X	1	1	0	DB 高阻态

## 二、8255A的控制字和工作方式

### 1、工作方式：

#### ◆方式0：基本输入输出方式

- ◆适用于无条件传送和查询方式的接口电路

#### ◆方式1：选通输入输出方式

- ◆适用于查询和中断方式的接口电路

#### ◆方式2：双向选通传送方式 (A口)

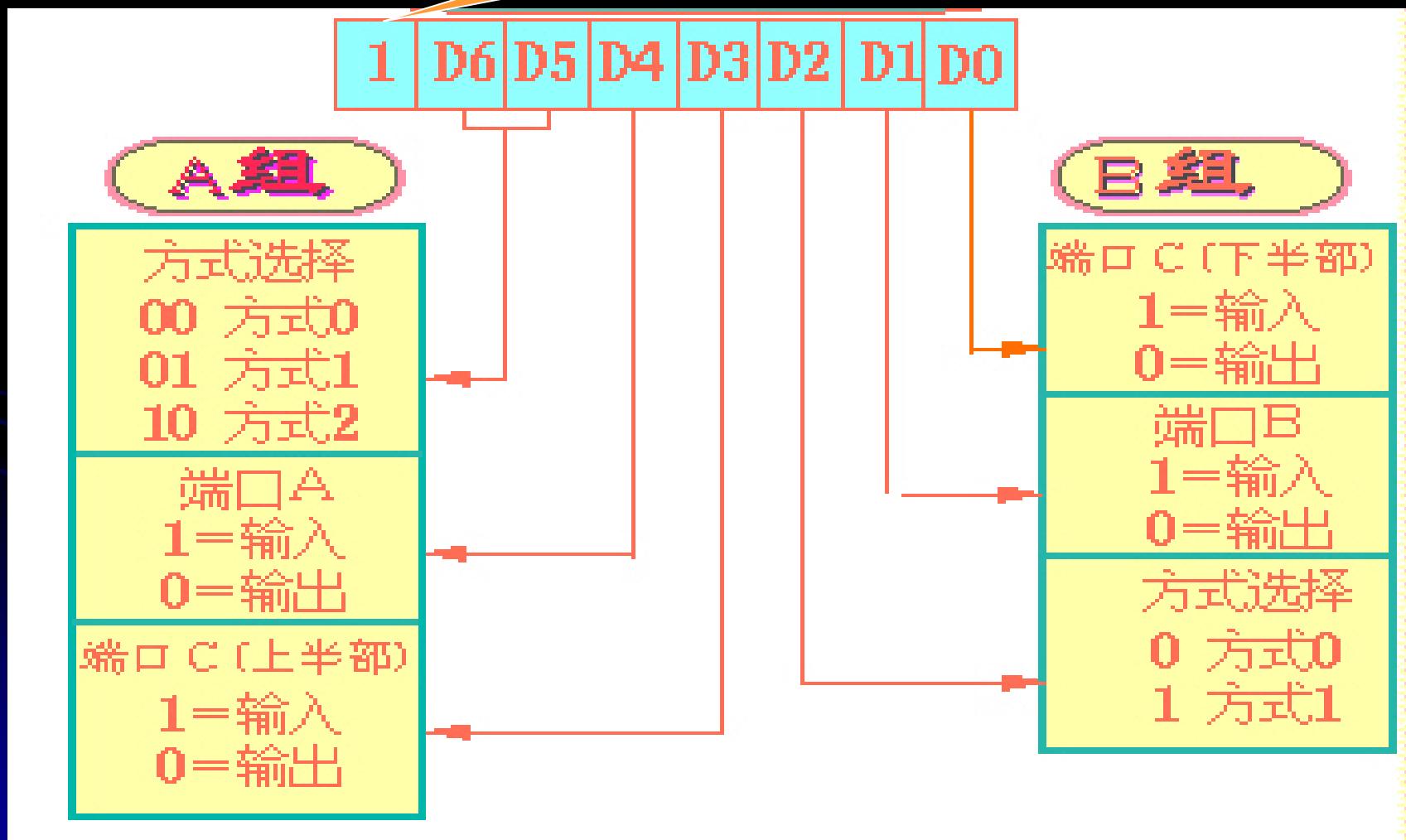
- ◆适用于双向传送数据的外设

- ◆适用于查询和中断方式的接口电路

## 2、控制字

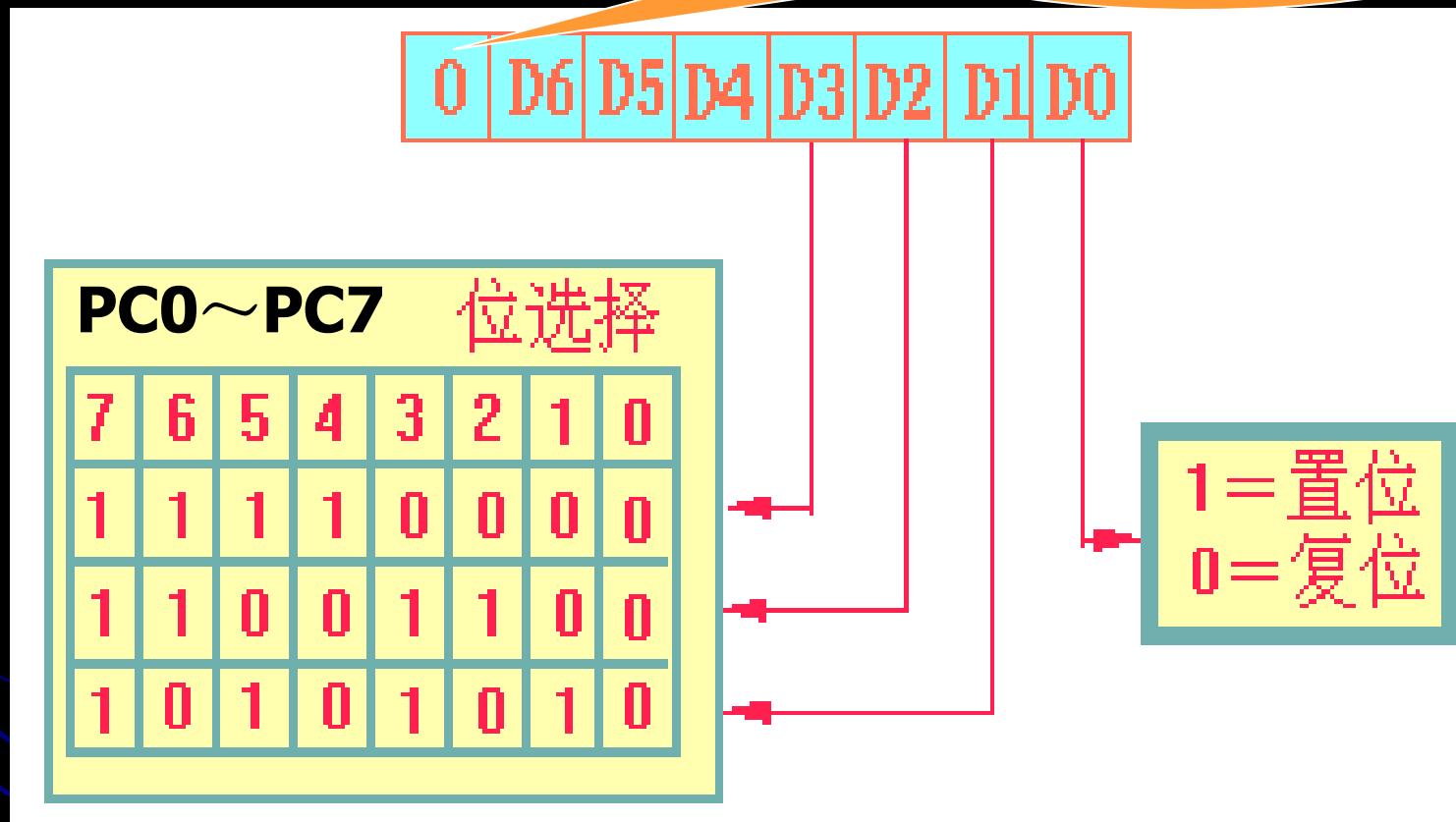
### (1) 工作方式控制字

特征位  
 $D7=1$ , 设置  
工作方式



## (2) C口置位/复位控制字

特征位  
D7=0, 置位/复位

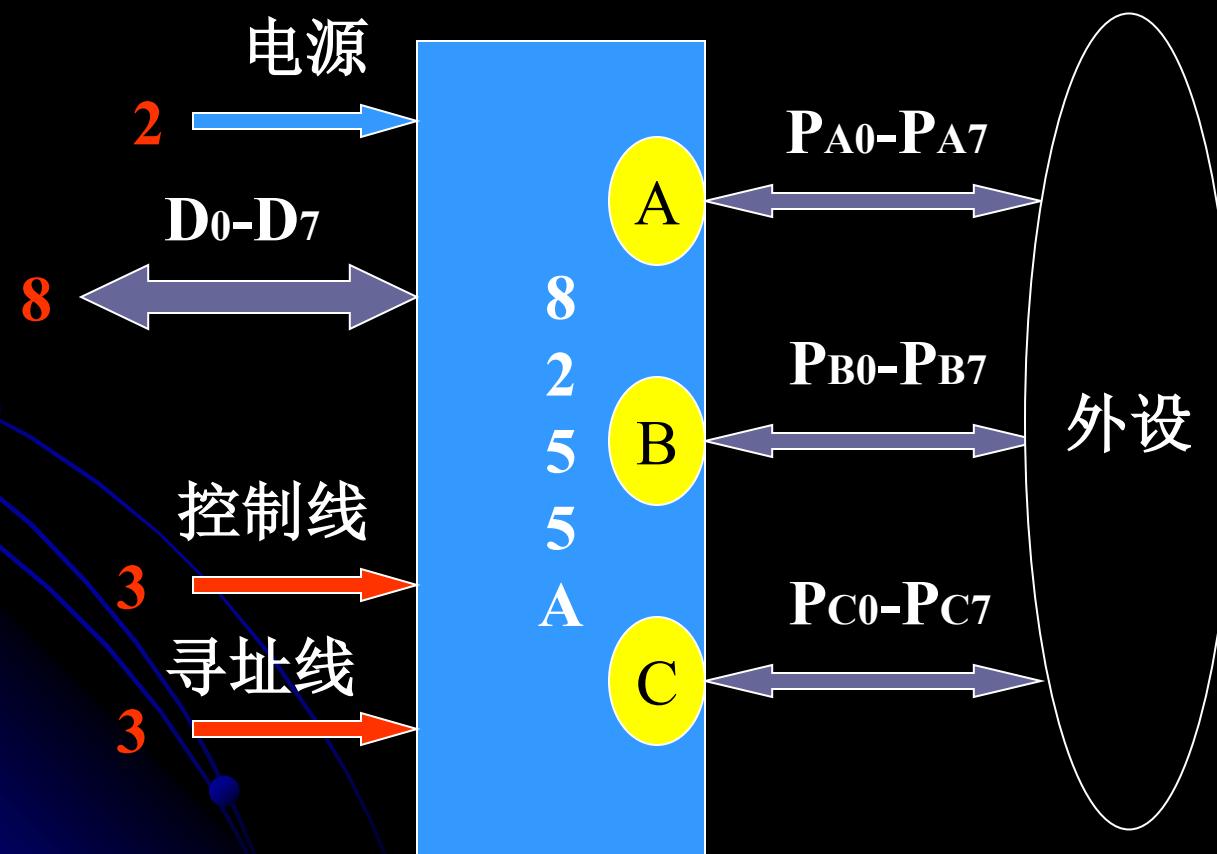


- 位控制字写入控制端口
- 特别便于置位/复位 内部中断允许触发器INTE

### 3. 工作方式的选择与确定

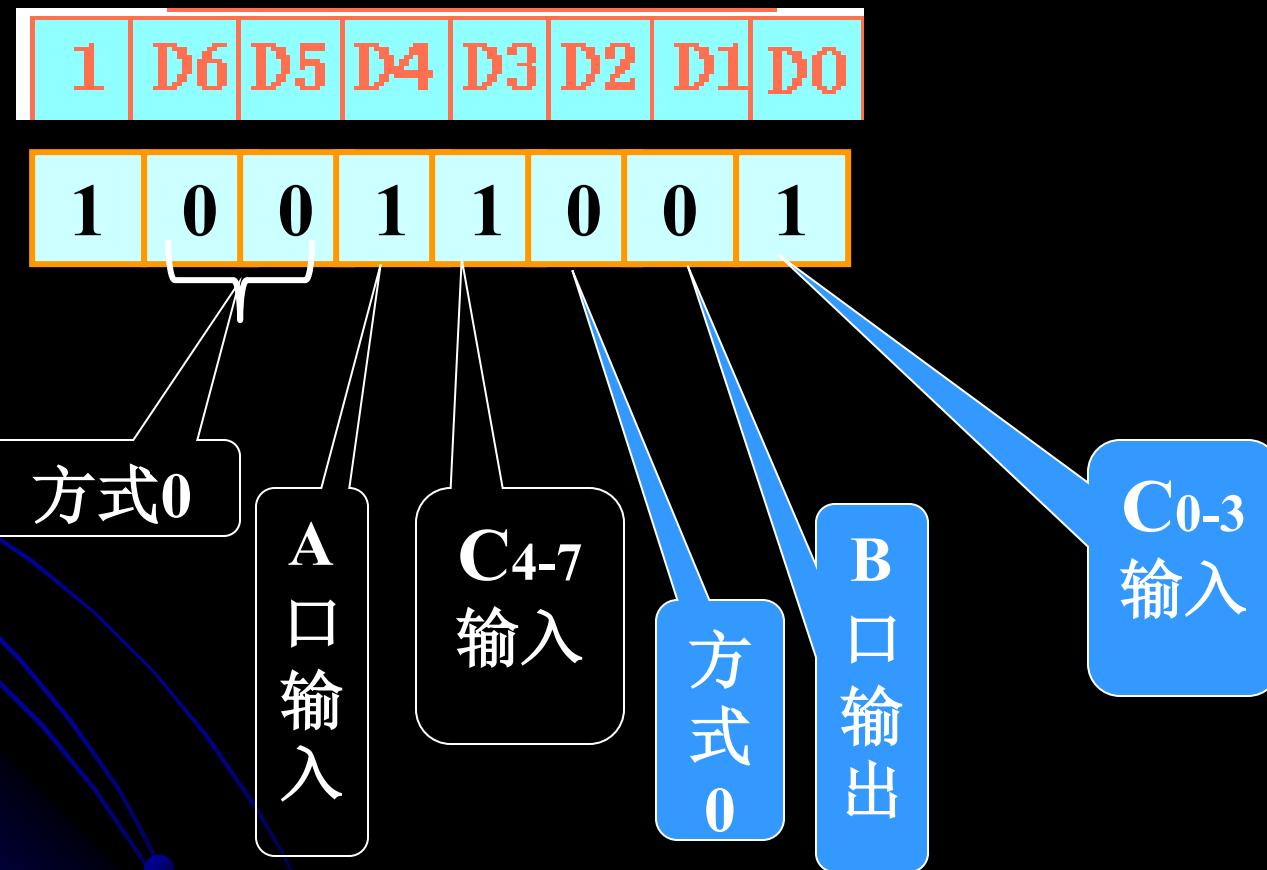
#### (1) 方式0 (基本输入/输出方式)

A、B、C 三个端口都可以设定为输入口或输出口，无握手联络线。提供24根I/O线。



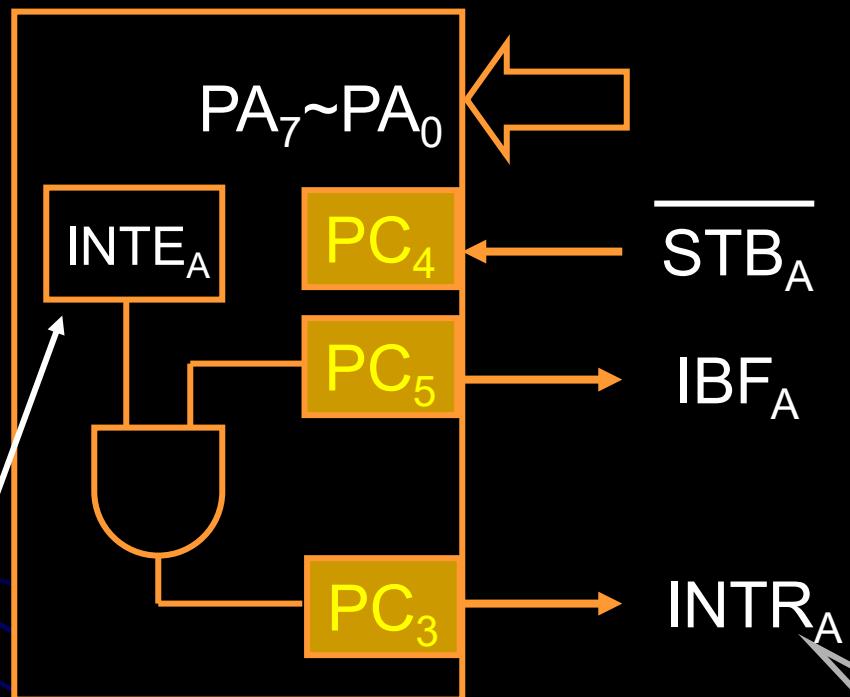
如：A口为输入； B口为输出； C口为输入。

则：选择8255的工作方式为方式0，对应控制字为：



## (2) 方式1（选通输入输出方式）

① 方式1输入引脚: (A口)

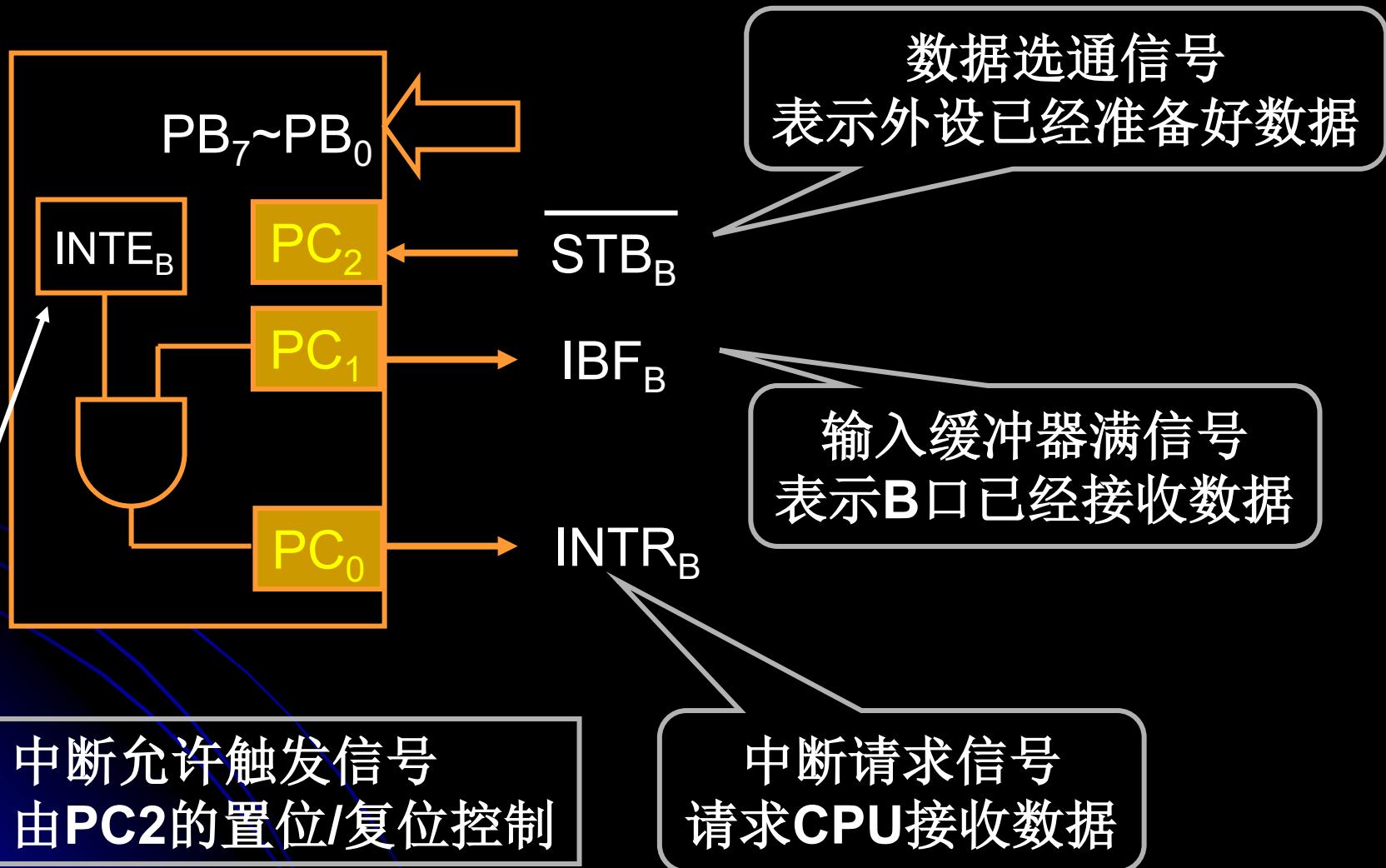


数据选通输入信号  
表示外设已经准备好数据

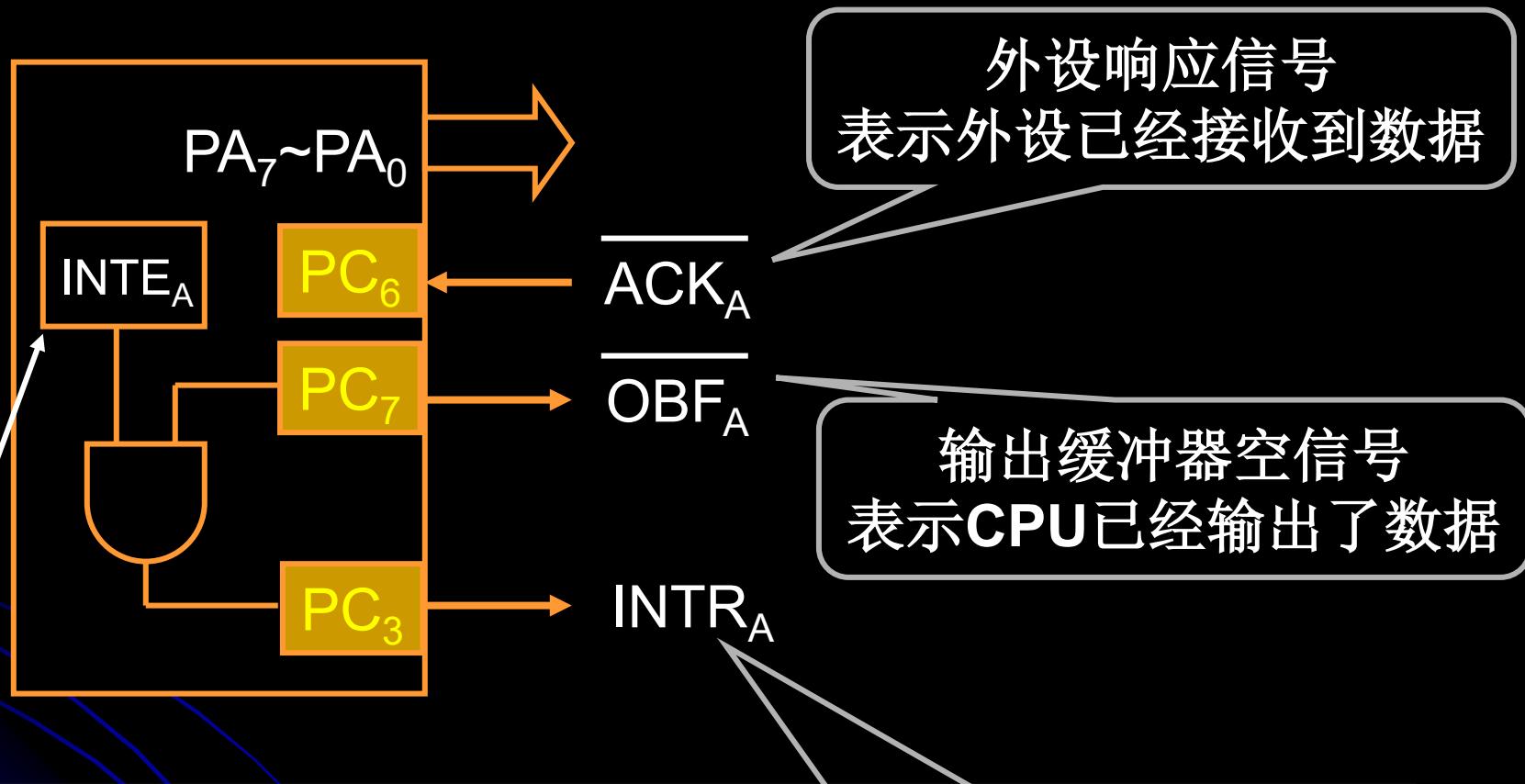
输入缓冲器满信号  
表示A口已经接收数据

STB和IBF是外设和8255A间的一对对应答联络信号，  
为的是可靠地输入数据

(B口)

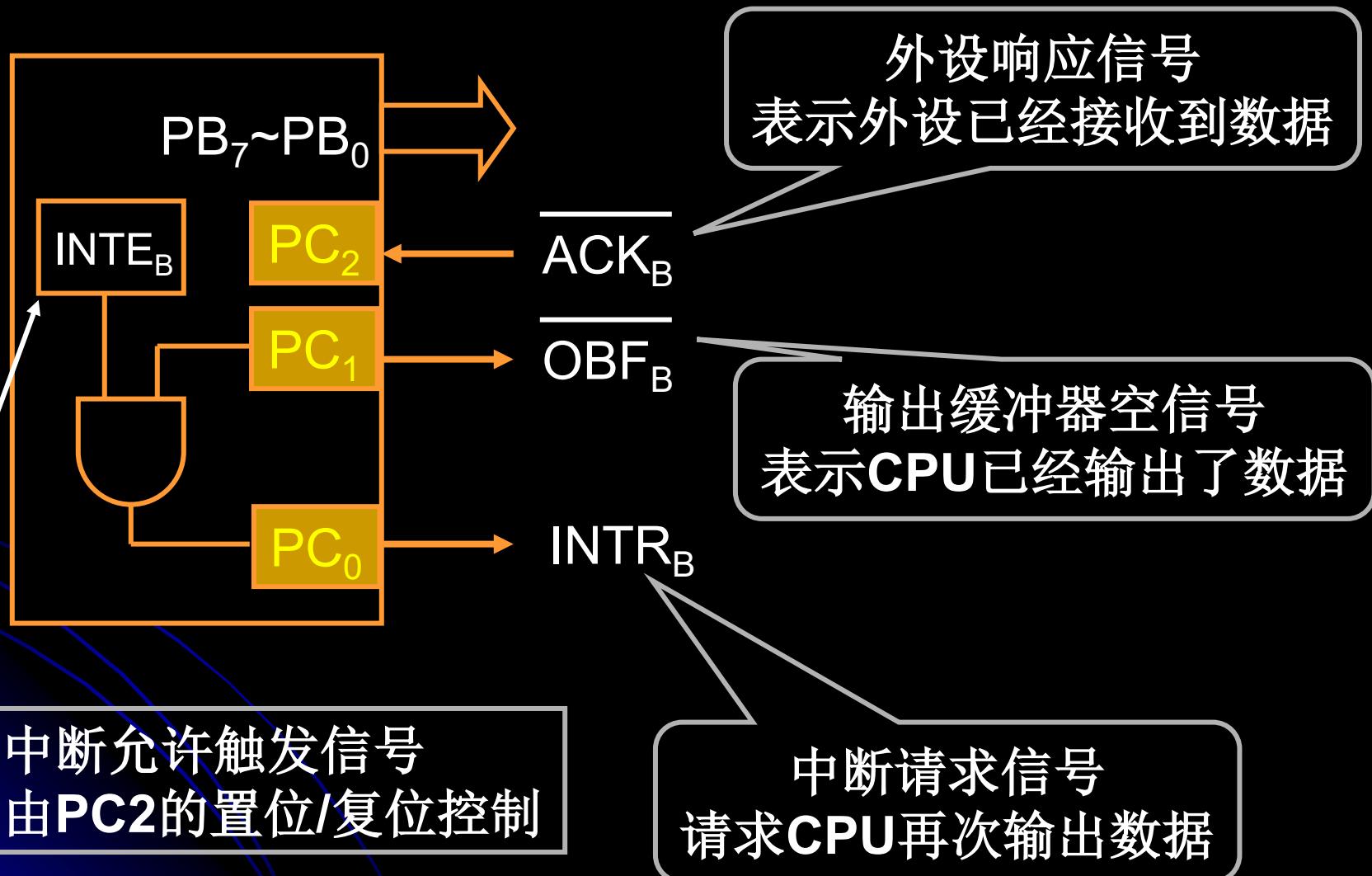


## ②方式1输出引脚: (A口)



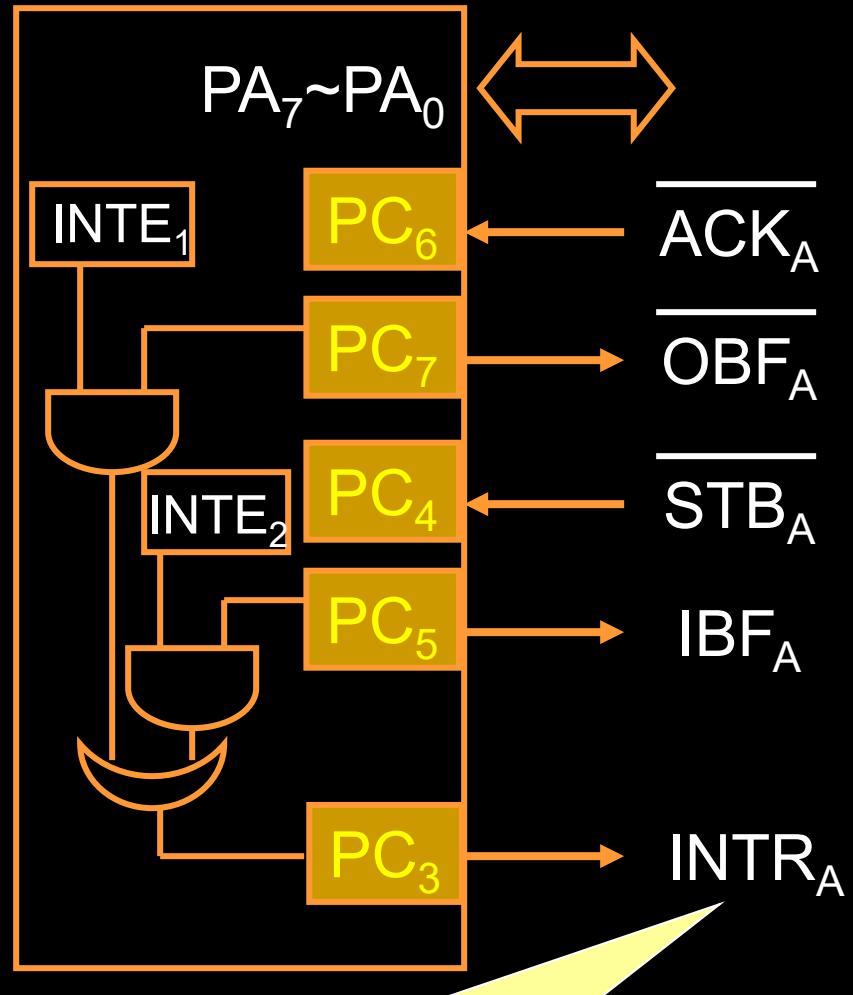
$OBF_A$ 和 $ACK_A$ 是外设和8255A间的一对应答联络信号，  
为的是可靠地输出数据

## ②方式1输出引脚: (B口)



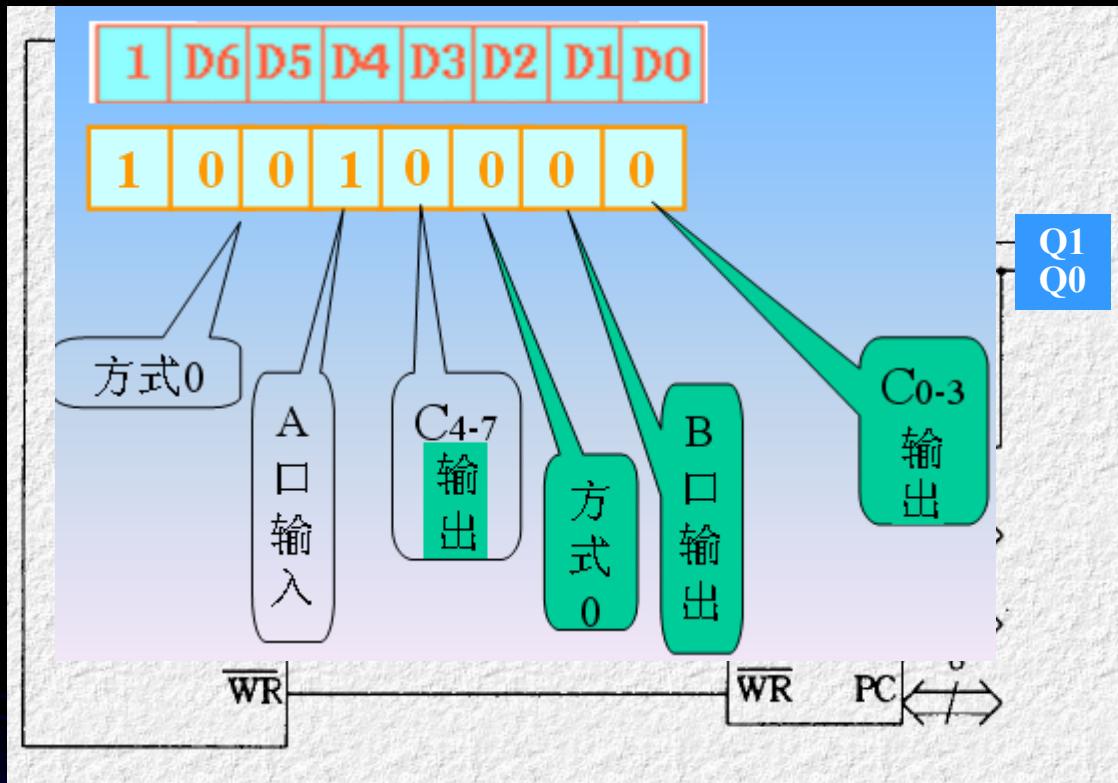
## (2) 方式2双向方式

- ◆ 方式2将方式1的选通输入输出功能组合成一个双向数据端口，可以发送数据和接收数据。
- ◆ 只有端口A可以工作于方式2，需要利用端口C的5个信号线，其作用与方式1相同。



当输入缓冲器已满或输出缓冲器已空时，发出中断申请。

### 三、8031 与 8255A 的接口



无状态或中断请求信号线相连，只能工作在方式0

A 口: 7FFCH

B 口: 7FFDH

C 口: 7FFEH

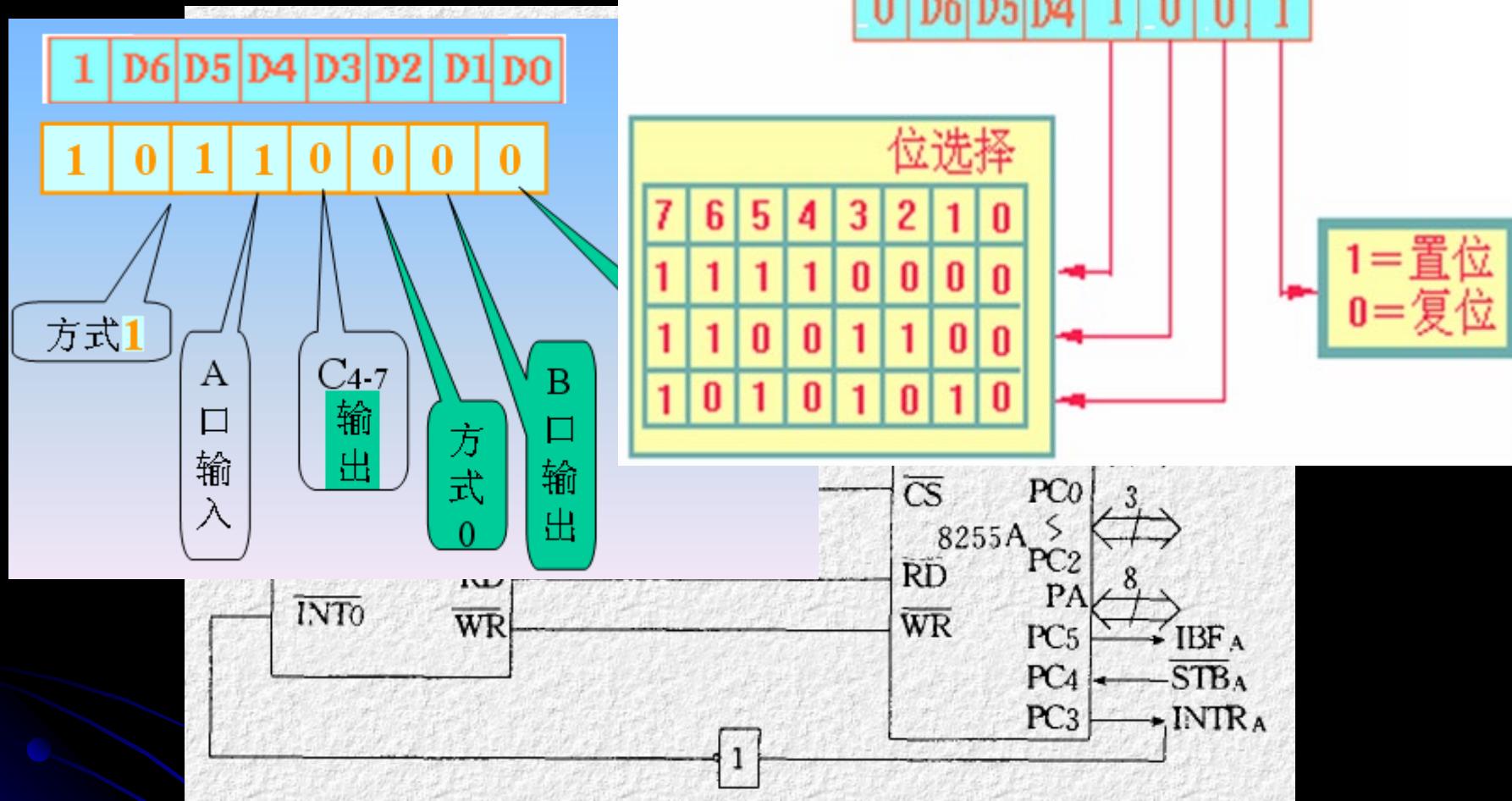
控制字寄存器: 7FFFH

设A口、B口和C口为基本输入输出工作方式0，A口为输入，B口和C口为输出，初始化程序为：

MOV DPTR, #7FFFH ;置控制字的地址

MOV A, #90H ;置控制字的内容

MOVX @DPTR, A



设8255A的A口为工作方式1输入，B口和C口下半部分为工作方式0输出，数据存入以R0为工作寄存器的地址单元中。

地址:

A口: FEFCH

B口: FEFDH

C口: FFEFH

控制字寄存器: FFFFH

工作方式控制字: B0H  
若A口允许中断，则PC4置位，位控制字: 09H

初始化程序：

```
MOV DPTR, #0FEFFH ;置控制字地址
MOV A, #0B0H        ;置控制字内容: 1011 0000B
MOVX @DPTR, A       ;A 口方式 1 输入, B 口、C 口下半部方式 0 输出
MOV A, #09H          ;置 C 口位操作控制内容: 0000 1001 B
MOVX @DPTR, A       ;允许 A 口中断请求
SETB EA
SETB EX0             ;开外部中断0
SETB IT0             ;边沿触发
⋮
```

+

中断服务程序：

```
AI: PUSH ACC
    MOV DPTR, #0FEFCH ;指向 A 口
    MOVX A, @DPTR
    MOV @R0, A
    INC R0
    POP ACC
    RETI
```

设8255A的A口为工作方式1输入，B口和C口下半部分为工作方式0输出，数据存入以R0为地址寄存器的单元中。

# 本章小结

- ✓ 微型机和外设之间的数据传送方式有三种：程序传送、中断传送和直接存储器存取传送。
- ✓ I/O接口是CPU与外设之间传递信息和控制信号的部件。
- ✓ MCS-51有4个并行I/O口，分别是P0、P1、P2和P3口，每个口有8位。
- ✓ 8255A是Intel系列的可编程并行接口芯片，具有A、B、C三个8位并行口。

# 第8章 定时器/计数器

8.1 定时器/计数器的结构及工作原理

8.2 方式和控制寄存器

8.3 工作方式

8.4 定时器/计数器应用举例

8.5 定时器/计数器的功能扩展

在工业检测、控制中，很多场合都要用到计数或定时功能。例如对外部脉冲进行计数、产生精确的定时时间等。

定时方法：软件定时、硬件定时、可编程定时器定时

- **软件定时**是执行一个循环程序来进行时间延迟，不需硬件电路，但是占用CPU时间。
- **硬件定时**不占用CPU，但若要调整定时的时间，就需要改变电路元件的参数。
- **可编程定时器定时**，是通过对系统时钟脉冲计数来实现的。改变计数值即可改变定时时间。同时还可以对外部事件脉冲进行计数。

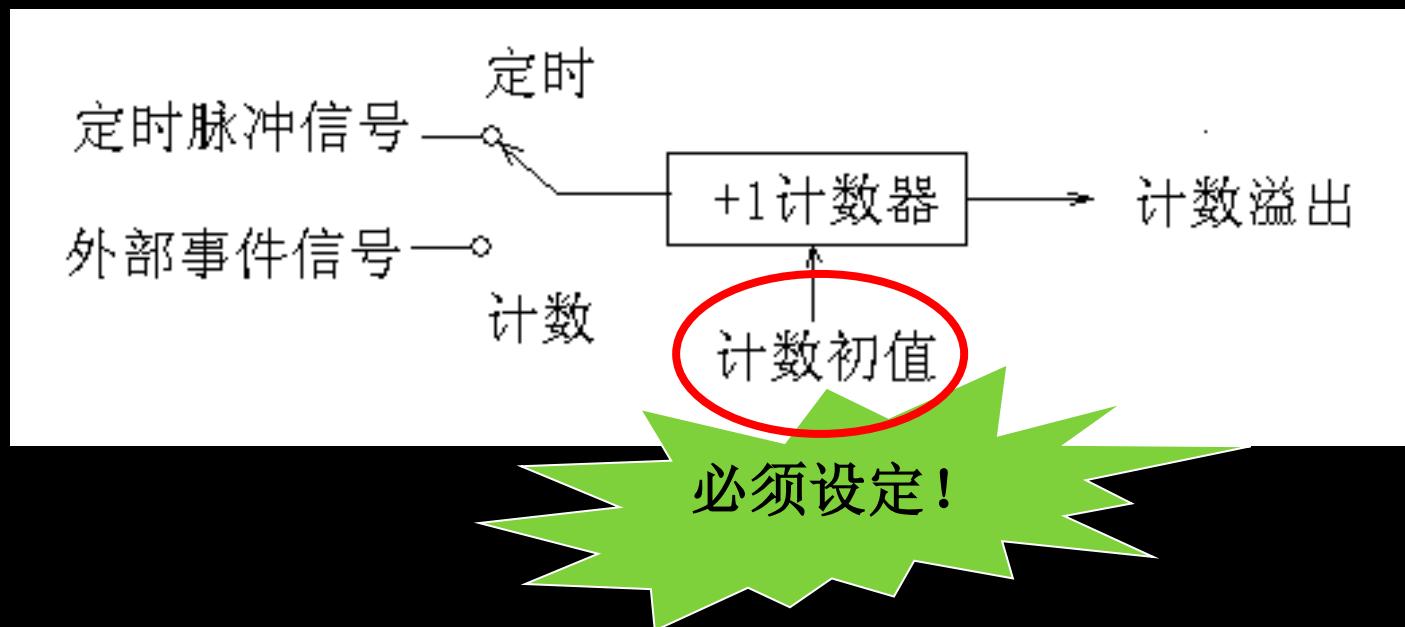
MCS-51单片机内部有两个16位可编程的定时器/计数器T0和T1。

具有两种工作模式（计数器模式、定时器模式）  
四种工作方式（方式0、方式1、方式2、方式3），其控制字均在相应的特殊功能寄存器（SFR）中，通过对SFR编程，可以方便地选择工作模式和工作方式。

## 8.1 定时器/计数器的结构及工作原理

定时器/计数器： Timer/Counter

本质上都是加1计数器，当对固定周期的脉冲信号计数时是定时器，对长度不确定的外部脉冲信号计数时是计数器。



## 1、计数原理

对外部事件进行计数。每来一个外部脉冲，计数器加1。

假设计数初值为x，计数器在初值x的基础上加1计数，若CPU读取计数器的当前值为Nc，则实际计数值为：

$$N = Nc - x$$

## 2、定时原理

计数脉冲接入周期变化的定时脉冲信号，计数器从某个初值x开始加1计数，到计满回零的瞬间产生溢出中断，表示定时时间到。

$$\text{定时时间 } t = (M - x) * T$$

M：计数器从0开始计数到溢出的最大值

T：计数脉冲的周期

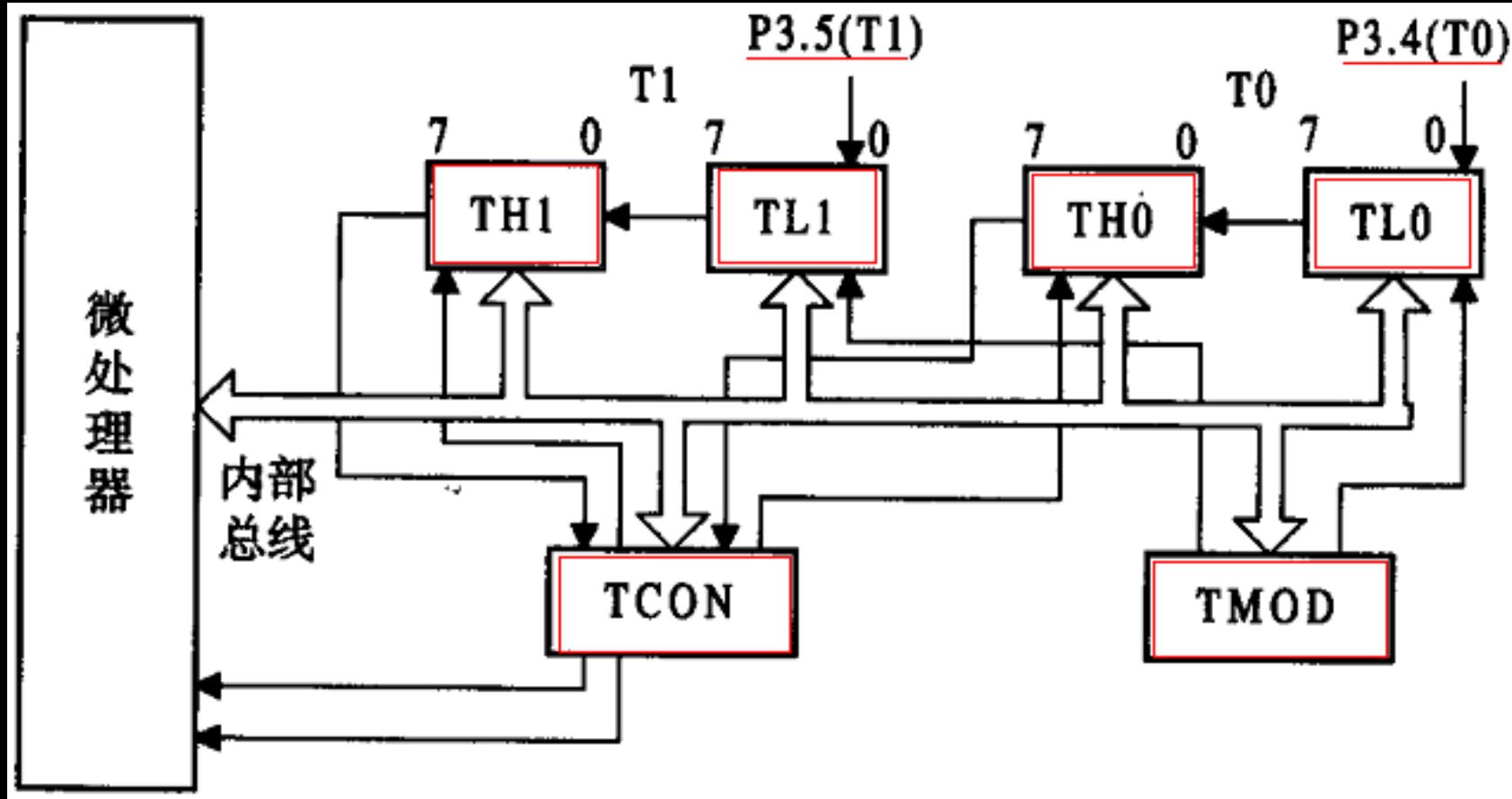
定时器初值的确定：

加1计数器是计满溢出时才申请中断，所以在给定时器/计数器赋初值时，不能直接输入所需的计数值，而应输入的是计数器计数的最大值与这一计数值的差值。设最大值为M，计数值为N，初值为X，则X的计算方法如下：

计数状态：  $X=M-N$

定时状态：  $X=M-\text{定时时间}/T$

而机器周期  $T=12 \div \text{晶振频率}$

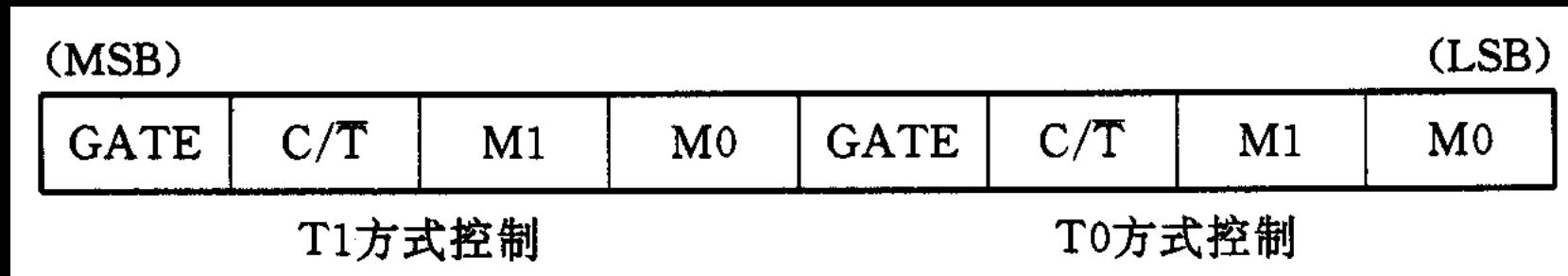


MCS-51定时器/计数器结构框图

定时器T0分别由两个8位寄存器TH0和TL0组成，T1由TH1和TL1组成。MCS-51两个引脚（P3.4和P3.5）分别是两个定时器的计数输入端。寄存器TCON和TMOD用来对定时器/计数器进行控制。

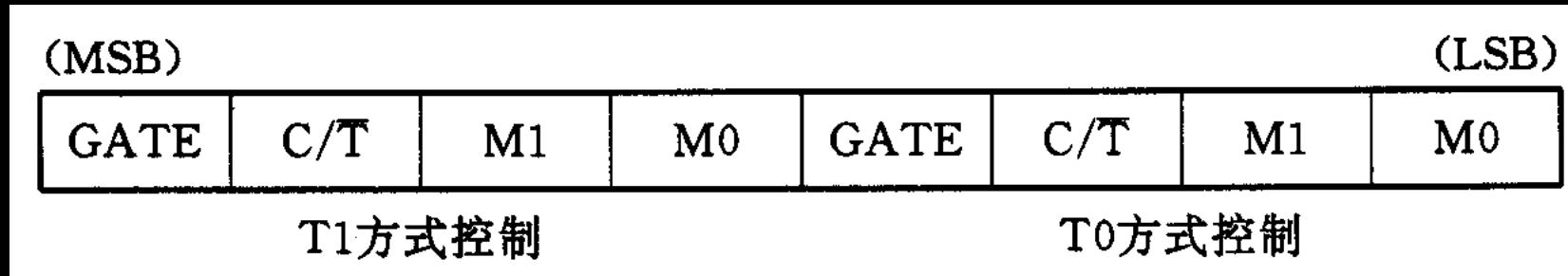
## 8.2 方式和控制寄存器

### 一、 定时器/计数器的方式寄存器TMOD (89H)



用来设定定时器/计数器的工作方式。

8位分为两组，高四位控制T1，低4位控制T0

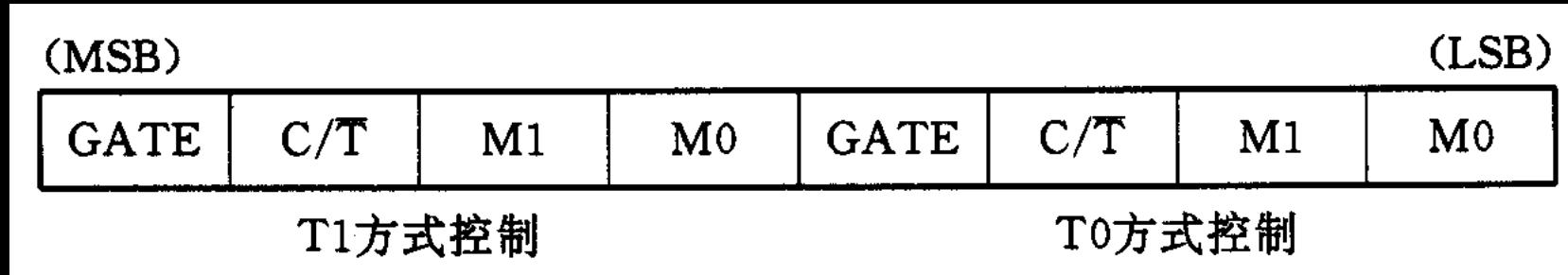


## 对TMOD的各个位的说明:

## GATE位：门控制位

GATE=1时，T0、T1是否计数要受到外部引脚输入电平的控制， $\overline{\text{INT0}}$ 引脚控制T0， $\overline{\text{INT1}}$ 引脚控制T1。可用于测量在 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ 引脚出现的正脉冲的宽度。（外部控制）

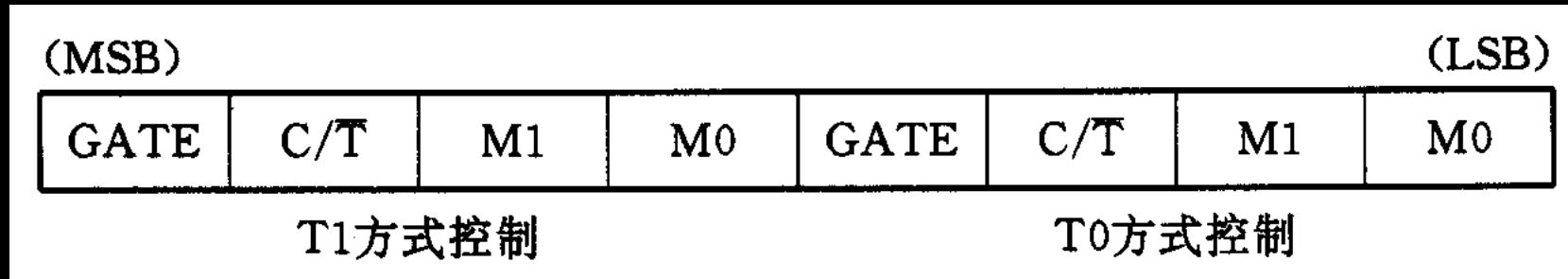
GATE=0, 不使能门控功能, 定时计数器的运行不受外部输入引脚INT0、INT1的控制。 (内部控制)



$C/T$ 位：计数方式或定时方式的选择位

$C/T=0$ ，为定时方式，内部计数器对晶振脉冲12分频后的脉冲进行计数，即对机器周期进行计数。从计数值可以求得计数的时间。

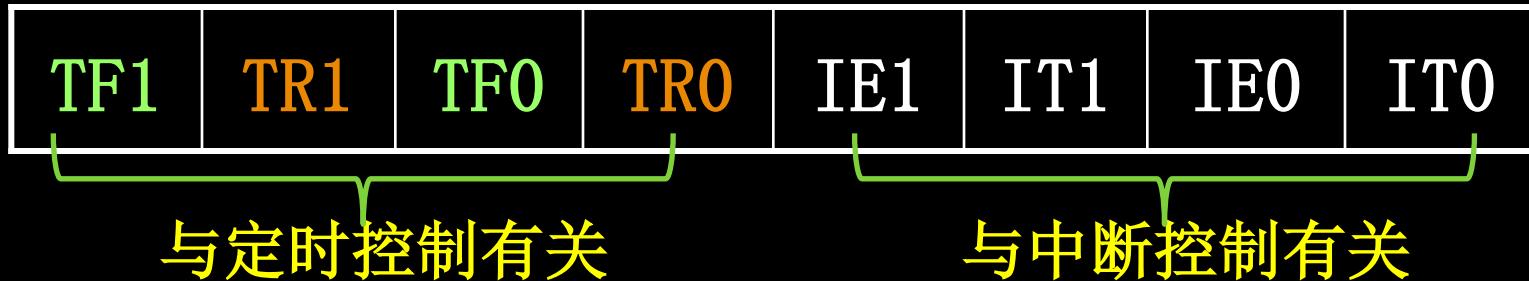
$C/T=1$ ，为计数方式，计数器对外部输入引脚T0（P3.4）或T1（P3.5）的外部脉冲（负跳变）进行计数，允许的最高计数频率为晶振频率的1/24。



## M1 M0: 四种工作方式的选择位

M1 M0	方式	说 明
0 0	0	13 位定时器/计数器 (TH的 8 位和TL的低 5 位)
0 1	1	16 位定时器/计数器
1 0	2	自动重新装入计数初值的 8 位计数器
1 1	3	T0 分成两个独立的 8 位计数器, T1 在方式 3 时停止工作

## 二、定时器/计数器控制寄存器TCON (88H)

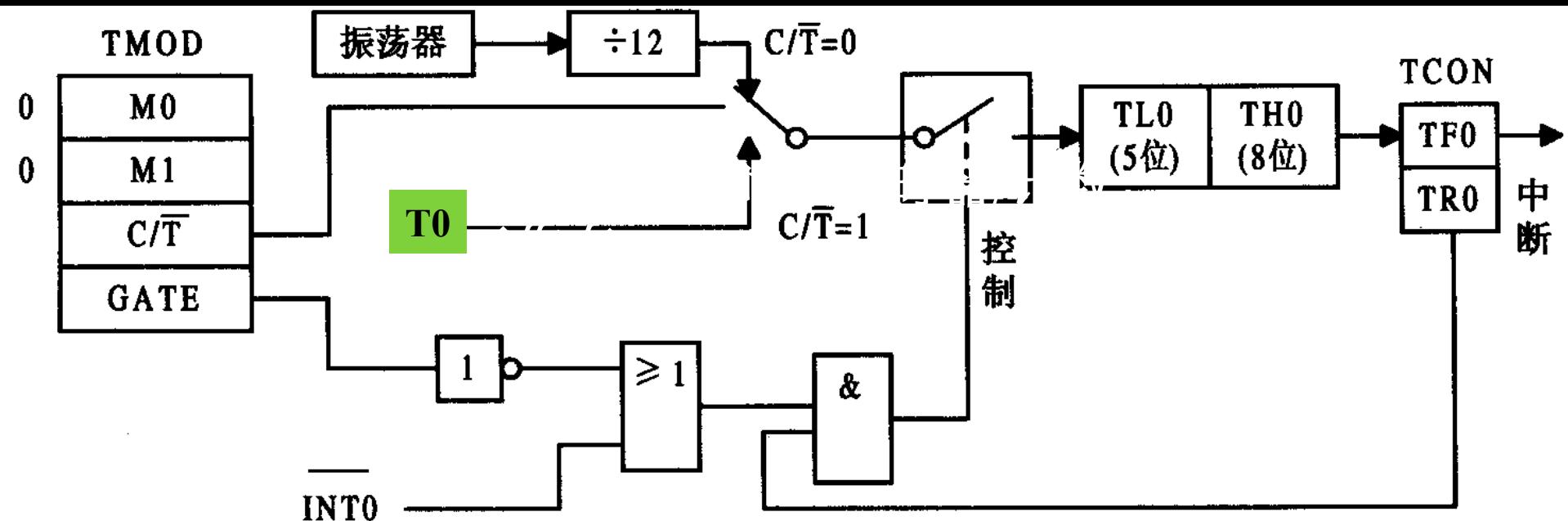


TF0、TF1分别是定时器/计数器T0、T1的溢出标志位，加法计数器计满溢出时置1，申请中断，在中断响应后自动清0。TF产生的中断申请是否被接受，需要由中断是否开放来决定。

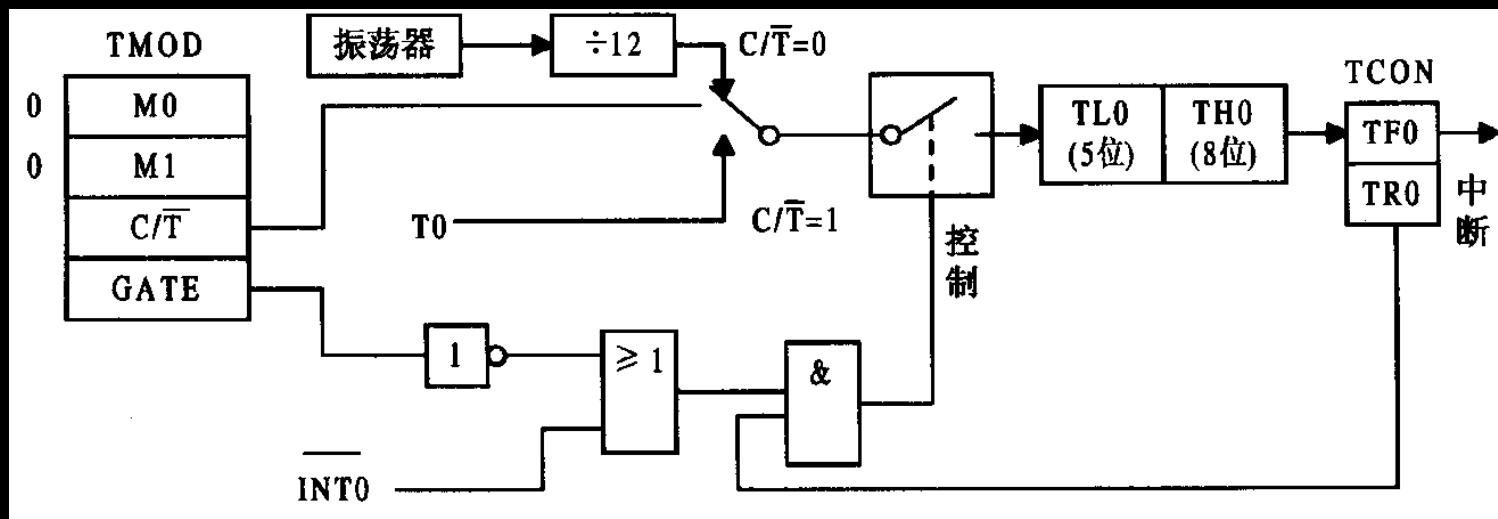
TR1、TR0 分别是定时器 / 计数器 T1、T0 的运行控制位。通过软件置 1 后，定时器 / 计数器启动工作，清 0 则停止工作。

## 8.3 工作方式

### 一、方式 0

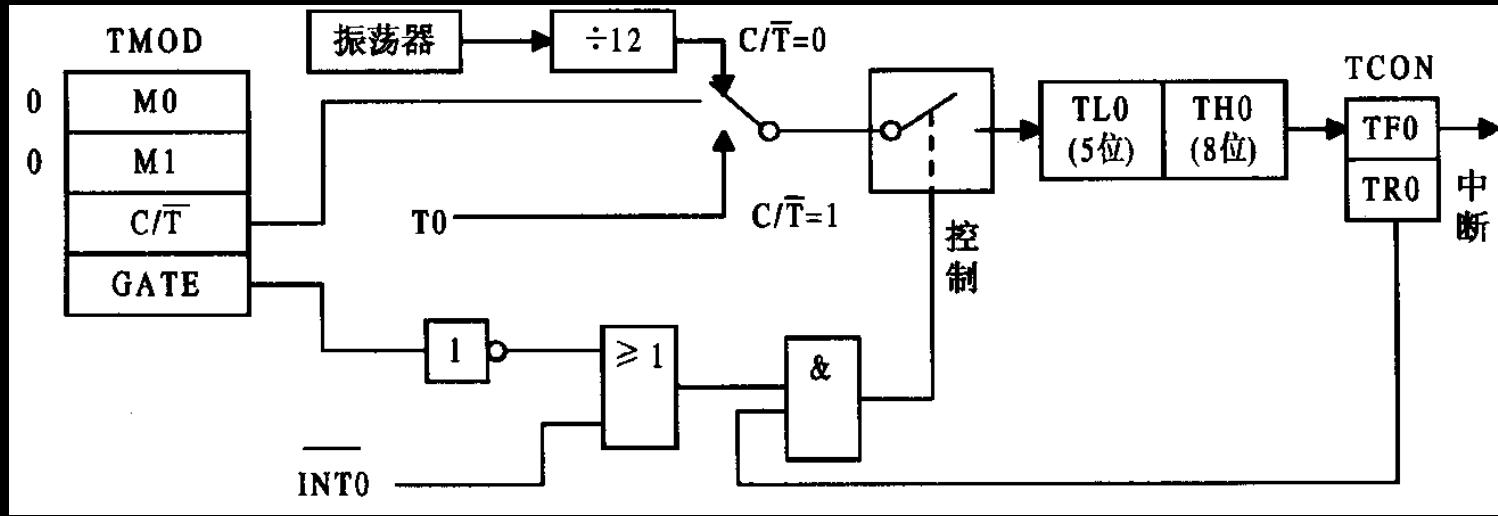


当M1 M0为00时，定时器/计数器工作在方式 0 (13位计数器)



当 $C/T=0$ 时， T0工作在定时方式， 多路开关接通振荡脉冲的12分频输出， 计数器对振荡脉冲进行计数；

当 $C/T=1$ 时， T0工作在外部事件计数方式， 多路开关接通外部计数脉冲的输入端T0（P3.4）， 当外部计数脉冲发生由1到0的负跳变时， 计数器加1。



在方式0下，T0和T1工作在13位的定时/计数器方式，由TH的高8位和TL的低5位组成。

当TL的低5位计满溢出时，向TH进位；当T0的13位计数器全部为1以后，再加1就产生溢出，此时置TCON的溢出中断标志位TF0为1，同时把计数器全部清0，从0开始继续计数。

方式0的计数长度  $M$  为2的13次方。

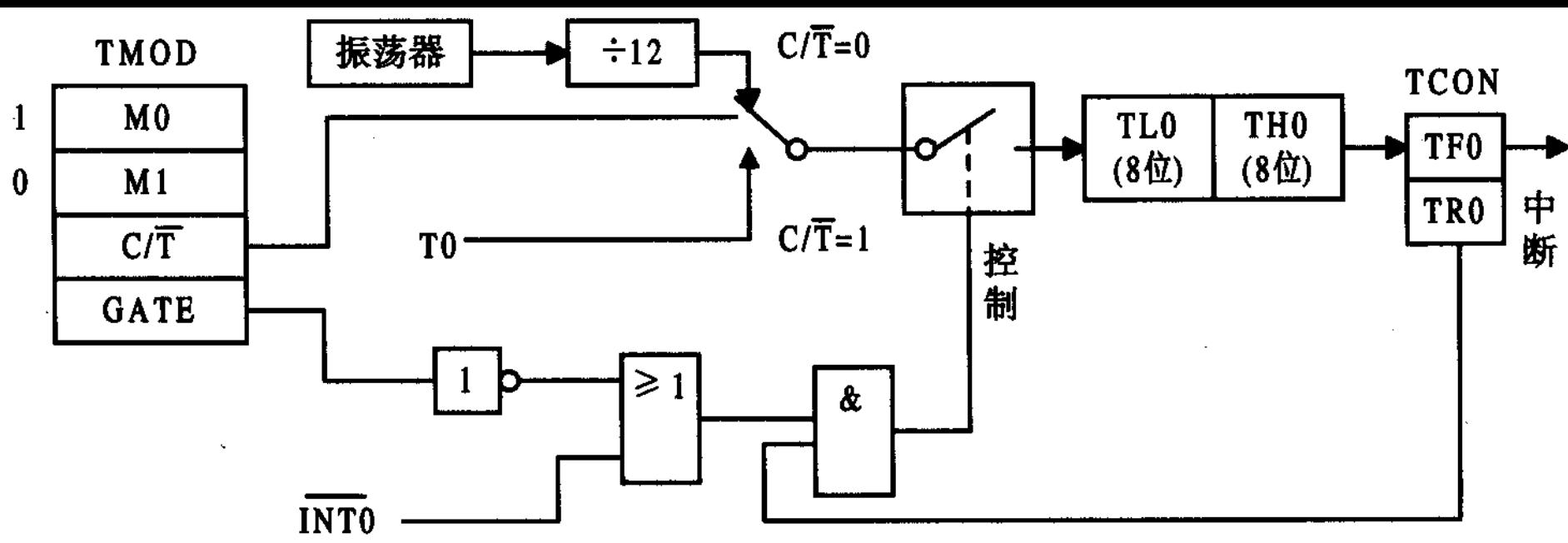
初值是13位二进制数，注意：高8位赋值给 TH0，低5位前面补足3个0凑成8位赋值给 TL0。

例如，若要求计数值为1000，则初值为

$$\begin{aligned} x = M - 1000 &= 8192 - 1000 = 7192 = 1C18H \\ &= \underline{1\ 1100\ 0001\ 1000} \text{ B} \end{aligned}$$

则赋初值时， $TH0 = 0E0H$ ， $TL0 = 18H$

## 二、 方式1



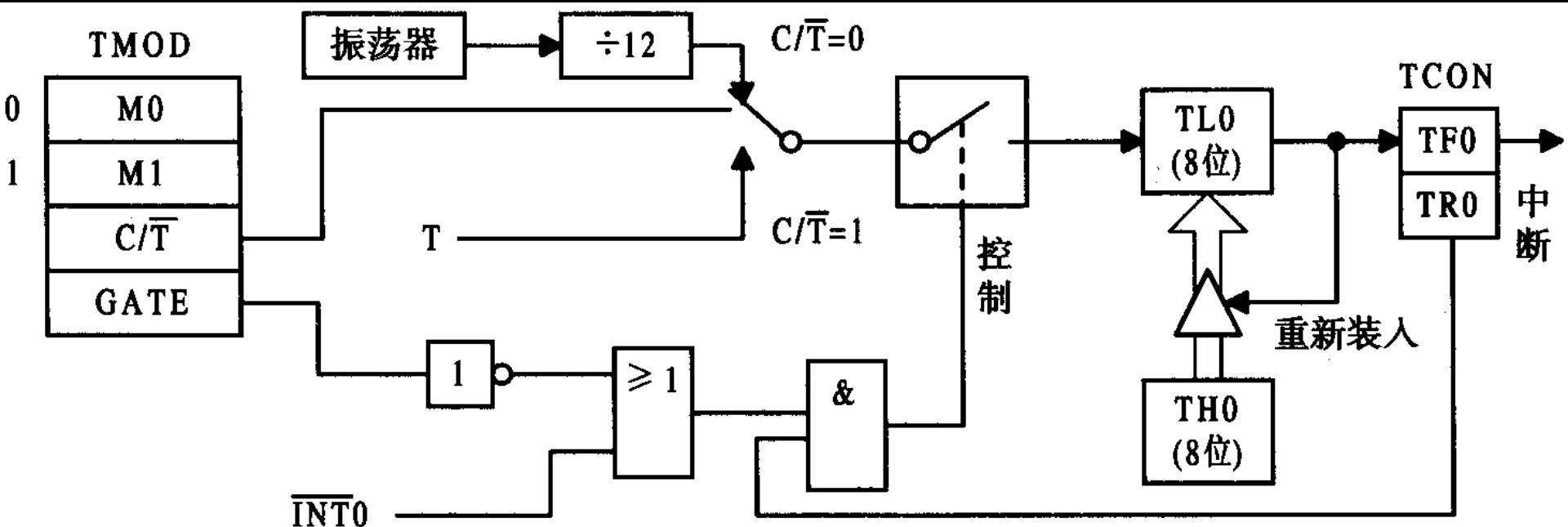
当M1 M0为01时，定时器/计数器工作在方式 1 (16位计数器)

方式1和方式0的工作原理基本相同，唯一不同是T0和T1工作在方式1时是16位的定时/计数器。

方式1的计数长度M是2的16次方。16位初值直接拆成高低字节，分别送入TH和TL。

TL计满溢出时，向TH进位；16位计数器计满溢出时，TCON的溢出中断标志位TF0/TF1置1，计数器清0。

### 三、方式 2



当M1 M0为10时，定时器/计数器工作在方式 2  
(计数初值自动重装载)

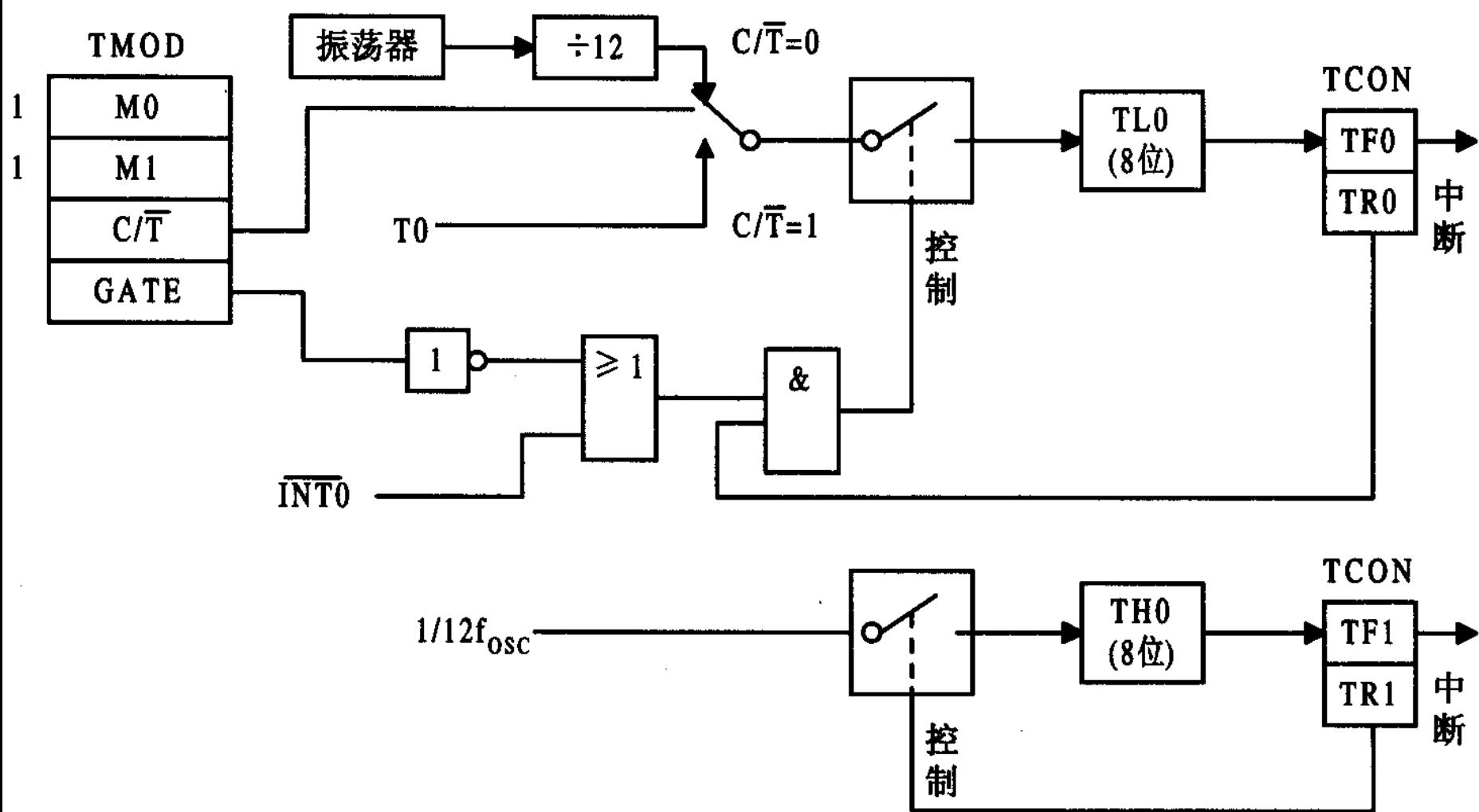
# 为什么要有自动重新装入计数初值的功能？

工作方式0和工作方式1 在计数溢出后，计数器为全0，在循环定时或循环计数应用时就存在反复设置初值的问题，给程序设计带来许多不便，同时也会影响计时精度。

针对此问题设置工作方式2，它具有自动重装载功能，即自动加载计数初值。

工作方式2中，16位计数器分为两部分：以TL0为计数器，以**TH0**作为预置寄存器，初始化时把计数初值分别加载至TL0和TH0中。当计数溢出时，不需要“人工干预”（由软件重新赋值），由预置寄存器TH0以硬件方法自动给计数器TL0重新加载，使计数器从初值重新开始计数。

## 四、 方式 3



当M1 M0为11时，定时器/计数器工作在方式 3  
(两个独立8位计数器)

在工作方式3下，定时/计数器T0被拆成两个独立的8位计数器TL0和TH0。

TL0既可以作计数器使用，也可以作为定时器使用。占用定时/计数器T0的控制位TR0和TF0，其功能和操作与方式0 / 方式1完全相同。

TH0只能作为定时器使用。需要借用定时/计数器T1的控制位TR1和TF1（以计数溢出置位TF1，TR1负责控制TH0定时的启动和停止）。

TL0既能作定时器也能作计数器使用， TH0只能作定时器使用，在方式3下，定时/计数器T0可以构成二个定时器或者一个定时器和一个计数器。

定时/计数器T1的控制位被定时/计数器T0借用，在方式3时停止工作。

## 五、定时器/计数器应用中的问题

### 1、门控位GATE的功能和使用方法

- 一般情况下，把GATE置为 0，定时/计数器的运行只受TR1或TR0的控制。
- 当GATE为 1 时，T0、T1除了受TR1或TR0的控制之外，还受到外部中断引脚的控制，INT0控制T0，INT1控制T1。

例如：只有当INT0和TR0同时为1时，启动计数；

当INT0为0时停止计数。

可用于测量在INT0和INT1引脚出现的正脉冲宽度。

# 测试INTx引脚上正脉冲 的宽度

```
ORG 2000H
MOV TMOD, #09H ; 置T0的GATE=1, 定时方式1, 16位
MOV TL0, #00H ; 置初值
MOV TH0, #00H
CLR EA ; 关中断
MOV R0, #30H

HERE1: JB P3.2, HERE1 ; INT0=1则循环, 等待INT0为低电平
        SETB TR0 ; INT0变低, 置TR0=1, 准备启动T0
HERE2: JNB P3.2, HERE2 ; 等待INT0为高电平, 启动T0
HERE3: JB INT0,HERE3 ; 等待INT0再次变低
        CLR TR0 ; INT0变低,置TR0=0, 停止计数
        MOV @R0, TH0 ;存入计数值
        INC R0
        MOV @R0, TL0
SJMP $
```

## 2、定时器/计数器运行中读取计数值

- 方法：先读THx，后读TLx，再读THx，两次THx内容一致时读数正确；否则重来。

```
RDTIME:  MOV A, TH0          ; 读TH0
          MOV R0, TL0          ; (TL0) ->R0
          CJNE A, TH0, RDTIME ; A≠TH0, 则循环
          MOV R1, A            ; (TH0) ->R1
          RET
```

### 3、定时器/计数器对输入信号的要求

- 定时方式时：输入信号来源于内部时钟，定时的精度取决于内部时钟脉冲的精度。
- 计数方式时：计数脉冲来源于外部引脚T0和T1
  - 每个机器周期S5P2对外部信号采样
  - 输入脉冲有一个负跳变时为1次有效计数
  - 新的计数值在检测到跳变后的下一个机器周期的S3P1出现
  - 最高计数频率为系统振荡频率的1/24：  $f_{osc}/24$

## 8.4 定时器/计数器应用举例

### 一、方式 0 的应用

例 1 设单片机晶振频率为 6 MHz, 利用定时器输出周期为2ms的方波。

选用定时器/计数器T0 作定时器, 输出为P1.0引脚, 2ms的方波可由间隔1ms的高低电平相间而成, 只需每隔1ms对P1.0 取反一次。

定时1ms的初值:

机器周期=12÷6 MHz= 2  $\mu$ s

1 ms内T0 需要计数的值:

$$N = 1 \text{ ms} \div 2 \mu\text{s} = 500$$

使用方式 0 的 13 位计数器，T0 的初值X：

$$X = M - N = 8192 - 500 = 7692 = 1E0CH = \text{1111000001100B}$$

13 位计数器中，低 8 位 TL0 只使用 5 位，其余均计入高 8 位 TH0 的初值，则 T0 的初值调整为：

$$TH0=0F0H, TL0=0CH$$

TMOD初始化: TMOD=00000000B=00H

(GATE=0, C/T=0, M1=0, M0=0)

TCON初始化: 启动TR0=1

IE初始化: 开放中断EA=1, 定时器T0 中断允许ET0=1

程序清单如下：

**ORG 0000H**

**AJMP START; 初始化程序入口**

**ORG 000BH ; T0中断矢量**

**AJMP TOINT ; T0中断入口**

**ORG 0300H**

**START: MOV SP, #60H; 堆栈**

**MOV TH0, #0F0H ; T0赋初值**

**MOV TL0, #0CH**

**MOV TMOD, #00H**

**SETB TR0 ; 启动T0**

SETB ET0 ; 开T0中断

SETB EA ; 开总允许中断

MAIN: AJMP MAIN ; 主程序

T0INT: CPL P1.0 ; 输出值取反

MOV TL0, #0CH ; 重新置计数初值

MOV TH0, #0F0H

RETI

## 说明：

上例是采用中断方式进行工作，即当定时器从初值开始递增计数，一直到计满溢出时，由单片机内部硬件对TF0置1，然后向CPU请求中断，在CPU响应中断转向中断服务程序后，TF0位由硬件自动清零。

书上P180的程序是采用查询方式进行工作，查询溢出中断标志位TF1是否为1，为1则转中断服务程序去执行，查询有效后需及时将TF1位清零。

## 二、 方式 1应用

方式 1 与方式 0 基本相同，只是方式 1 改用了 **16** 位计数器。要求定时周期较长时，13 位计数器不够用，就可用 16 位计数器。

### 三、 方式 2 应用

方式 2 是定时器自动重装载计数初值的操作方式。在计数值满溢出的同时，将 8 位二进制初值自动重装载。定时器 T1 工作在方式 2 时，可直接用作串行口波特率发生器。

#### 四、方式3的应用

定时器 T0工作在方式3 时是 2 个 8 位定时器，且TH0借用了定时器T1的溢出中断标志位 TF1 和运行控制位 TR1。

例2 假设有一个用户系统中已使用了两个外部中断源，并置定时器 T1于方式2，作串行口波特率发生器用，现要求再增加一个外部中断源，并由P1.0口输出一个5KHz的方波（假设晶振频率为 6 MHz）。

在不增加其它硬件开销时，可把定时器/计数器T0置于工作方式3，利用外部引脚T0端作附加的外部中断输入端，把TL0预置为0FFH，当在T0 端出现由1至0的负跳变时，TL0 立即溢出，申请中断，相当于边沿触发的外部中断源。

在方式3下，TH0 总是作 8 位定时器用，可以靠它来控制由 P1.0 输出的 5 kHz方波。5 kHz方波，即每隔 100  $\mu$ s使 P1.0 的电平发生一次变化。则TH0中的初始值  $X=M-N=256-100/2=206$ 。

MOV TL0, #0FFH

MOV TH0, #206 ; 或0CEH

MOV TL1, #BAUD ; BAUD根据波特率要求设置常数

MOV TH1, #BAUD

MOV TMOD, #27H ; 置T1工作方式2, T0工作方式3

; TL0工作于计数器方式

MOV TCON, #55H ; 启动定时器 T0、 T1, 置外部中断 0 和 1

; 为边沿触发方式

MOV IE, #9FH ; 开放全部中断

TL0 溢出中断服务程序（由000BH转来,定时器T0中断入口地址）

TL0INT: MOV TL0, #0FFH

... ; 外部引脚 T0 引起中断处理程序

RETI

TH0 溢出中断服务程序（由001BH转来,定时器T1中断入口地址）

TH0INT: MOV TH0, #206

CPL P1.0

RETI

此处串行口中断服务程序、外部中断0和外部中断1的中断服务程序没有列出。

## 8.5 定时器/计数器的功能扩展

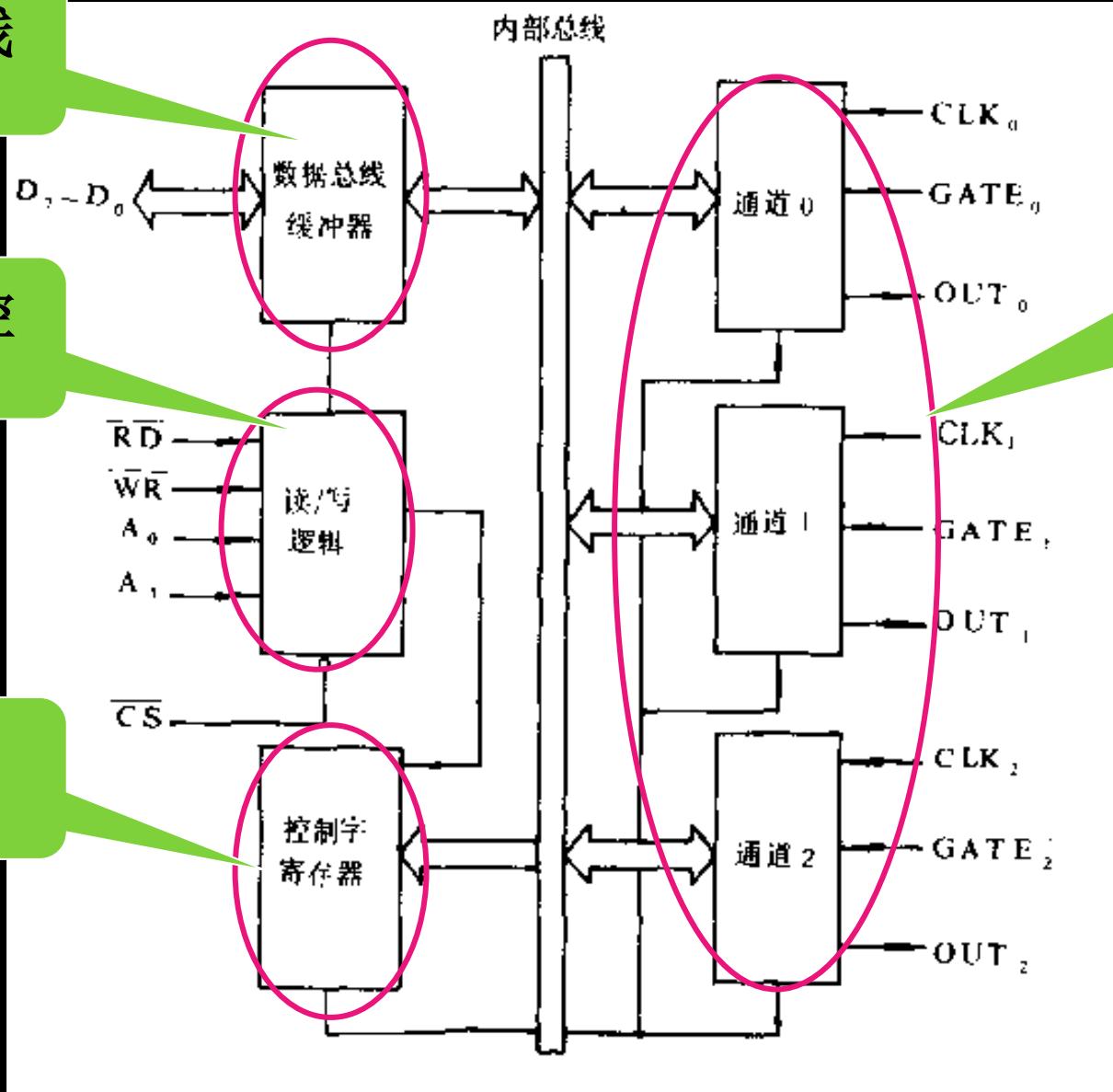
可编程定时器/计数器8253内部有3个独立的16位定时/计数器通道。每个计数器都是一个可预置数的减1计数器，可按照二进制或十进制计数，计数和定时范围可在1~65535之间改变，每个通道有6种工作方式，计数频率可高达2MHz以上。

# 8253的内部结构

数据总线  
缓冲器

读 / 写控  
制电路

控制字  
寄存器



脉冲输入端  
CLK, 计数器  
对引脚的输入  
脉冲进行计数

3个16位  
定时/计  
数通道

门控信号  
GATE用来控  
制计数/定时的  
启动、禁止计  
数等操作

计数器输出端  
OUT, 每当计  
数器中计数值  
减到0时, 输  
出一个信号

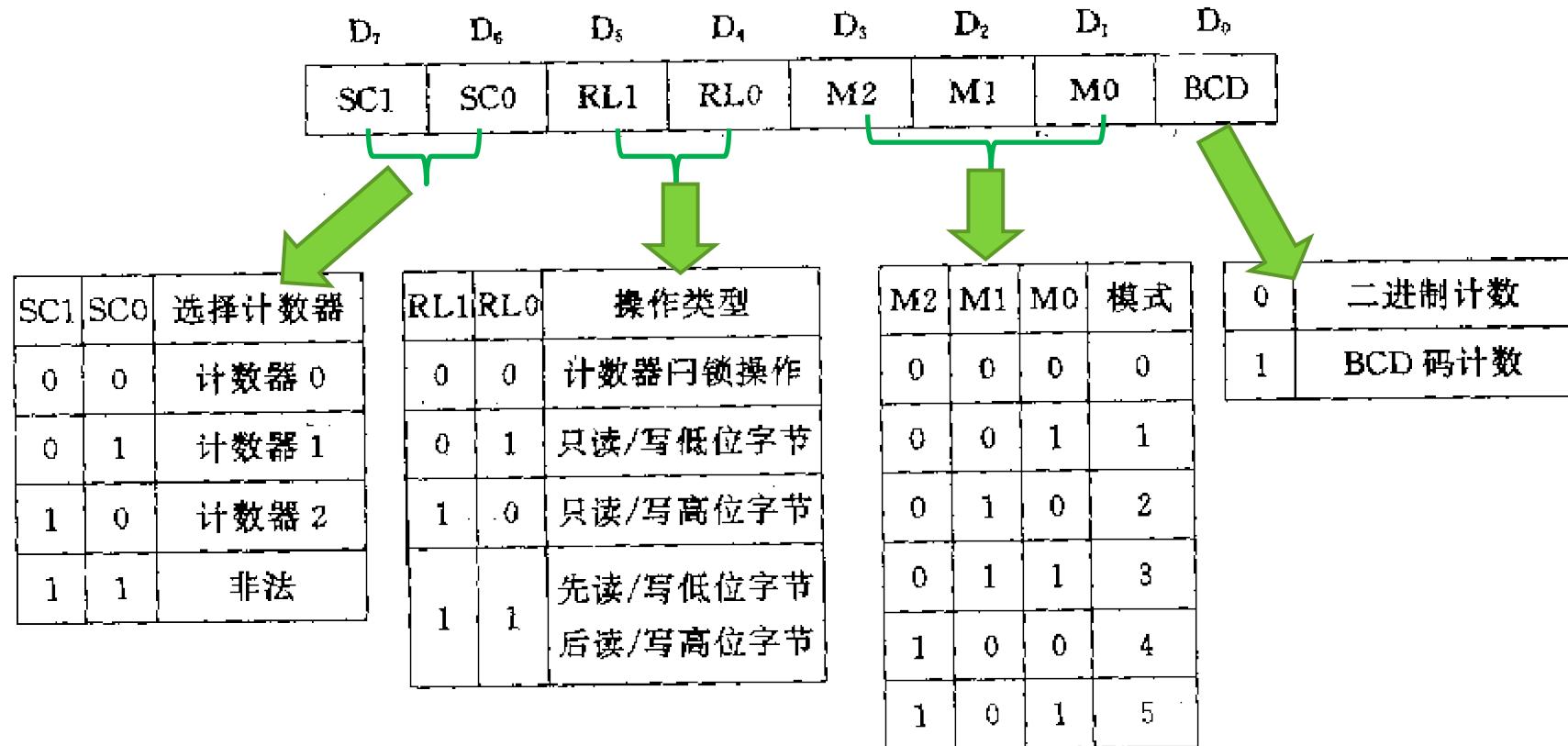
# CPU对8253的基本操作

/CS	/RD	/WR	A <sub>1</sub>	A <sub>0</sub>	操作
0	1	0	0	0	计数值写入计数器0
0	1	0	0	1	计数值写入计数器1
0	1	0	1	0	计数值写入计数器2
0	1	0	1	1	写控制字寄存器
0	0	1	0	0	读计数器0
0	0	1	0	1	读计数器1
0	0	1	1	0	读计数器2
0	0	1	1	1	不操作
1	×	×	×	×	禁止

# 8253的工作方式

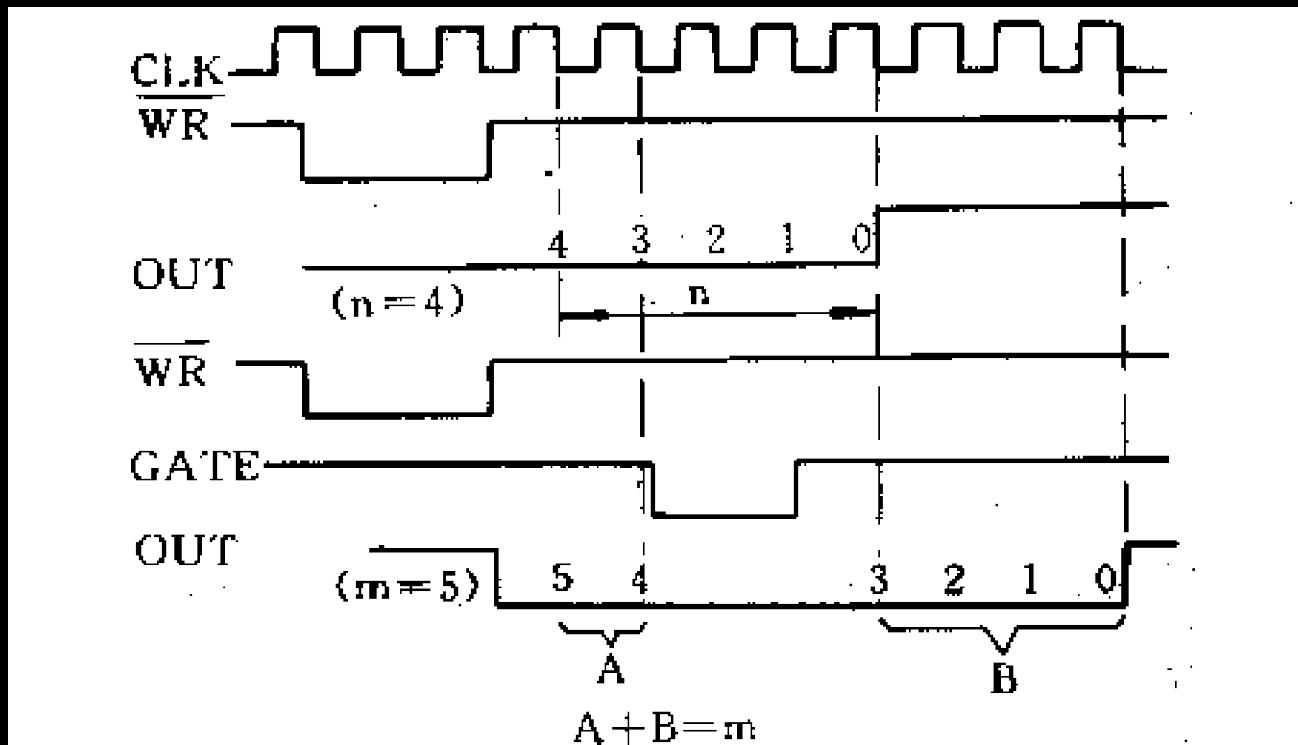
8253只有一个控制字，决定一个计数通道的工作方式。共分为4部分，通道选择、计数器读/写方式、工作方式和计数码的选择。

控制字的格式如下：



# 一、方式0：计数结束中断方式

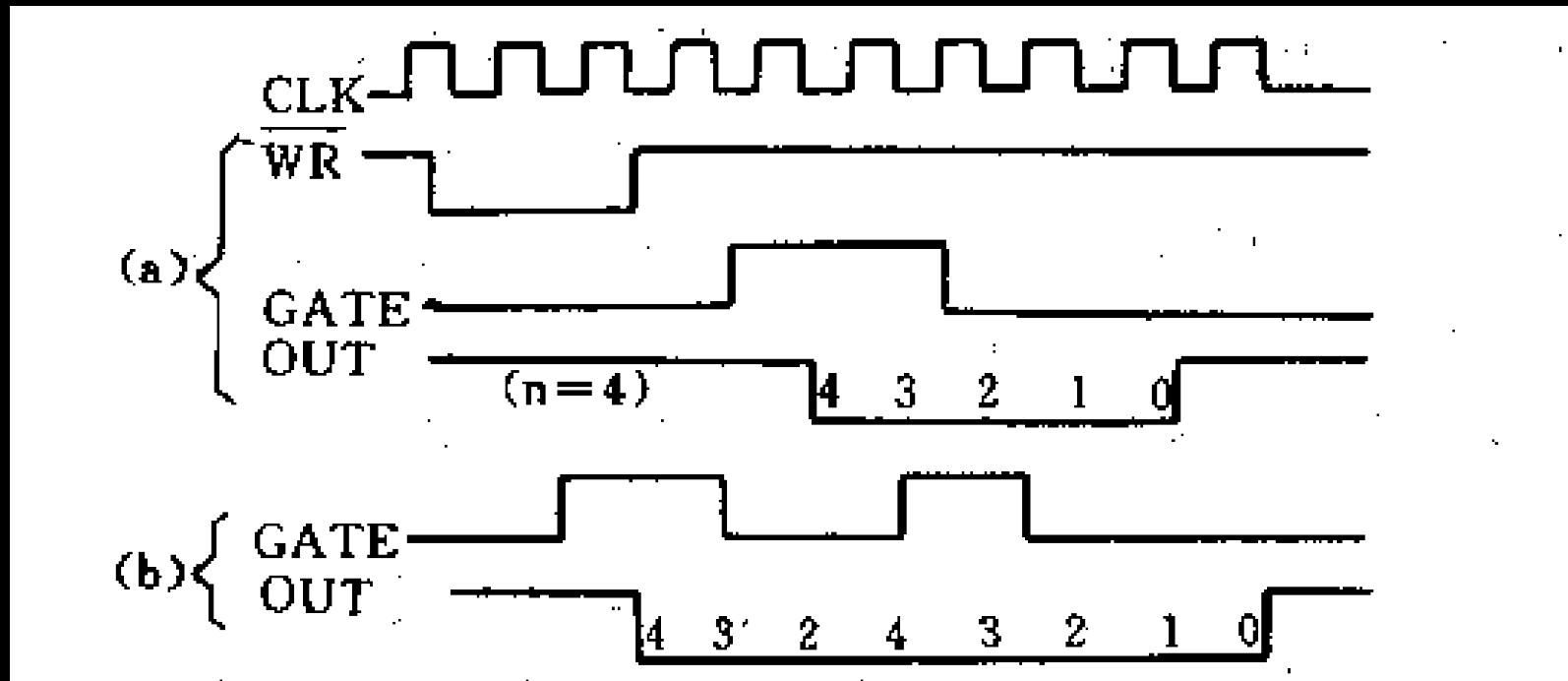
- 门控信号**GATE**必须为1，计数器才能计数；
- 计数时通道输出端**OUT**一直为0；
- 通道计数器计数到0后，**OUT**由0到1，同时计数器停止工作。



门控信号**GATE=0**，暂停计数；等到**GATE=1**后，从暂停时的计数值继续计数

## 二、方式1：可编程单稳态输出方式

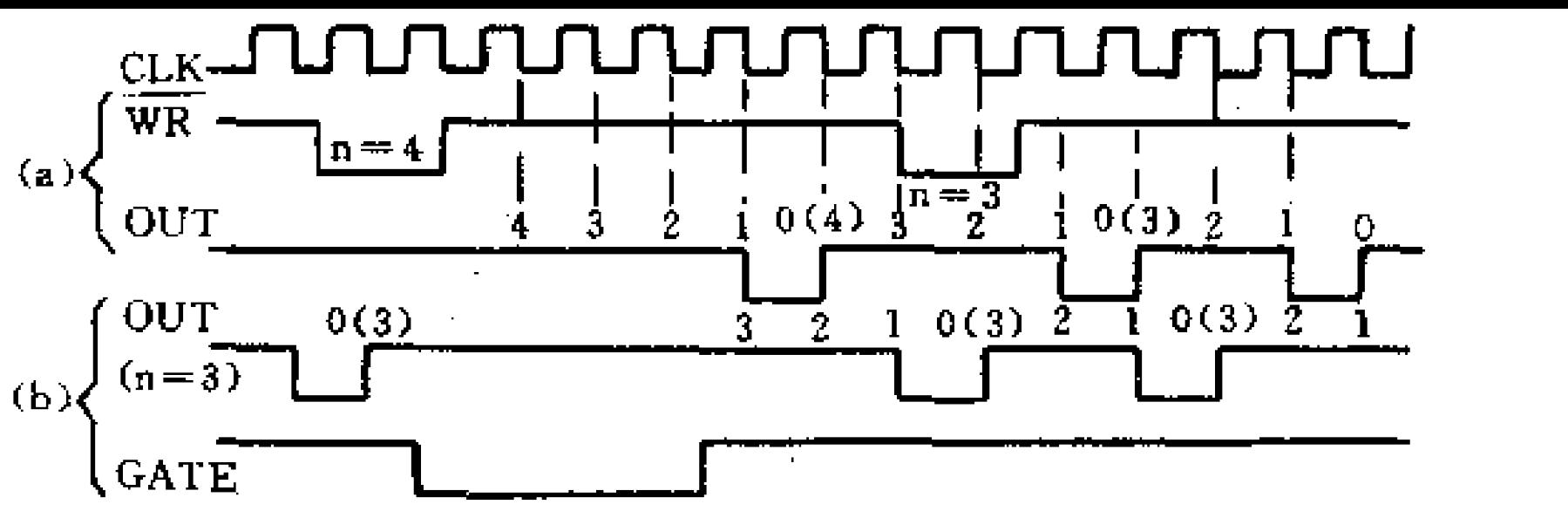
- 门控信号GATE是触发信号，上升沿有效。即开始计数是由GATE的上升沿触发的；
- 触发后，通道计数器开始计数，输出端OUT由高变低；
- 计数器计数到0，OUT再由低变高。



在计数过程中，GATE若又出现上升沿，计数器会重新装入原来设定的计数值，并重新开始计数。

### 三、方式2：速率发生器工作方式 可产生连续的负脉冲信号

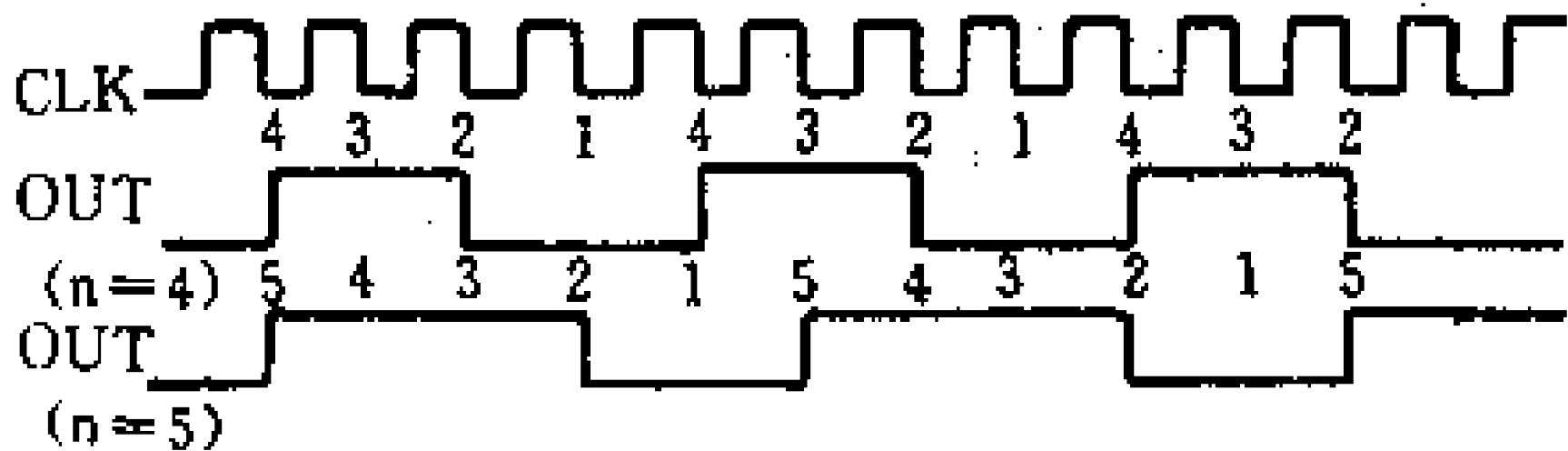
- GATE门为1，计数器才能工作，对CLK端上的脉冲进行计数；
- 当计数器“减”计数到1时，输出端由高变低，再经过一个CLK周期，即计数器计数到0时，输出端OUT又跳变为高。所以方式2输出周期性负脉冲信号，其宽度固定为一个CLK周期；
- 当计数器的值减为0时，自动重新装入计数初值，实现循环计数。



如果重新写入新的计数值，则在写信号WR有效且计数器减为0之后，再按照新的计数值开始计数。

#### 四、方式3：方波发生器工作方式

方式2输出是窄脉冲，如果是需要输出方波，可以选择方式3

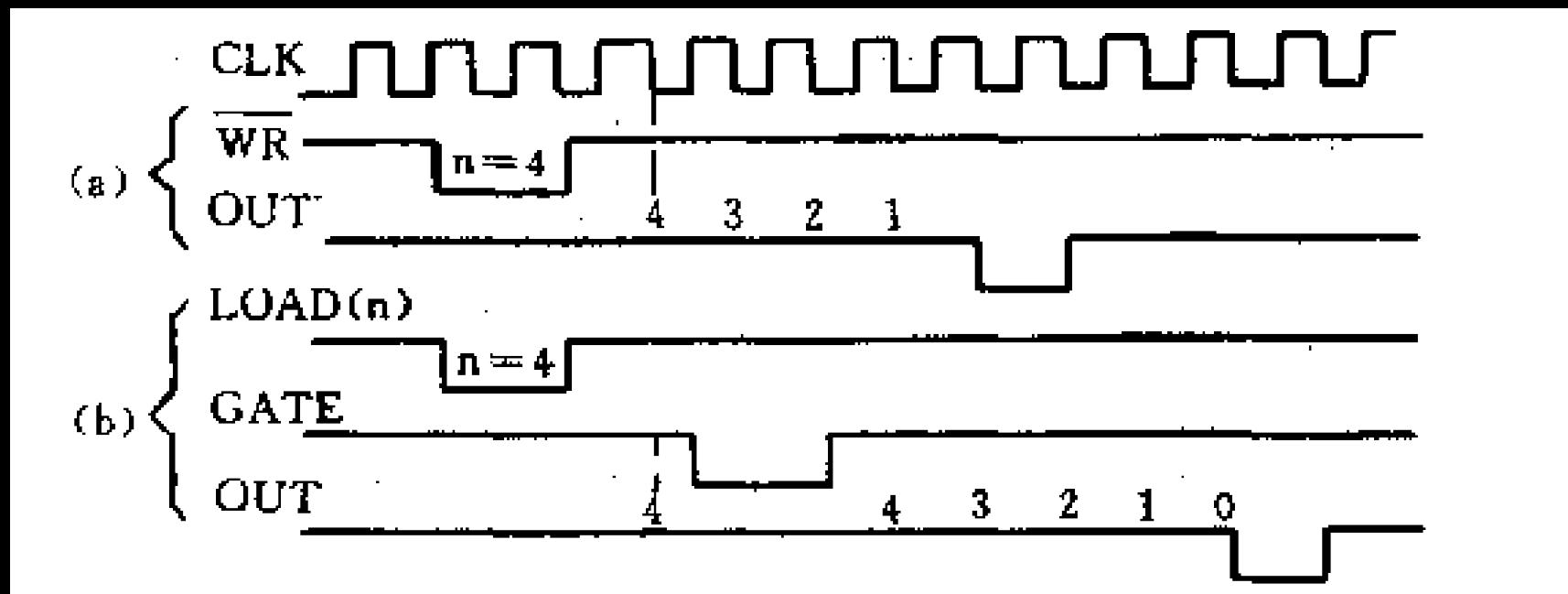


计数值为偶数，输出的是对称方波，即前 $n/2$ 计数值时输出高电平，后 $n/2$ 计数值时输出低电平。

计数值为奇数，输出的是不对称方波，即前 $(n+1)/2$ 计数值时输出高电平，后 $(n-1)/2$ 计数值时输出低电平。

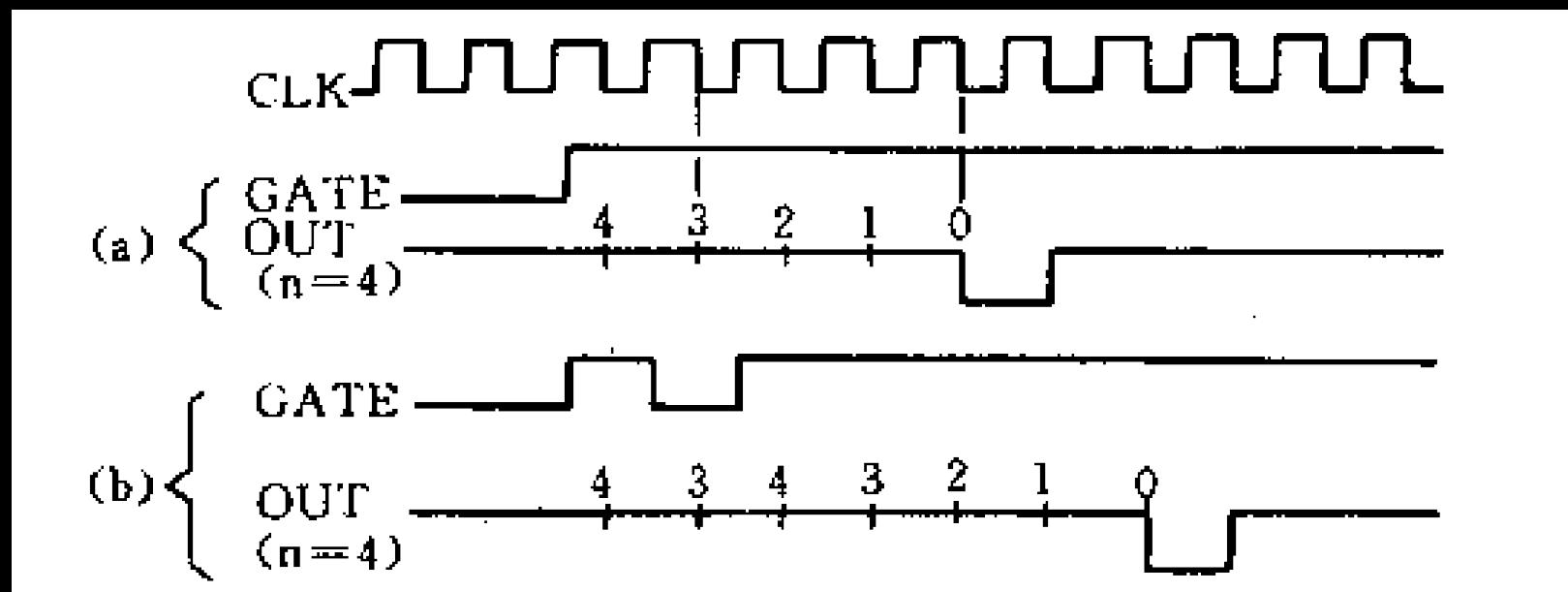
## 五、方式 4：软件触发选通工作方式

- 装入控制字后，OUT端为高电平，在WR写入计数值后开始计数，当递减计数到0时，输出一个时钟周期的负脉冲，然后再变为高电平；
- 在计数期间，门控信号GATE出现低电平时，计数器就停止工作，当GATE恢复为高电平后，又从原来设定的计数值开始重新计数。



## 六、方式5：硬件触发选通工作方式

- 与方式4相似，当控制字写入控制寄存器后，输出端OUT变高。
- 与方式4不同的一点是当计数值写入通道计数器后，通道并未被触发，即计数器并不立即开始计数。只有当GATE信号的上升沿触发通道后，通道计数器才开始计数。



在计数过程中，**GATE**若再次出现上升沿，计数器会重新按原来设定的计数值开始计数。

## 8253的工作方式小结

8253的六种工作方式可归为两类：

- 充当频率发生器
- 作计数器

与频率发生器有关的工作方式：方式2和方式3

对OUT端，方式2 提供给用户的是负脉冲，方式3 提供给用户的是方波。在这两种方式下，GATE信号要始终保持为高电平。

# 8253的工作方式小结

与计数器有关的工作方式: 方式0、1和方式4、5。

启动计数器的方式有两种:

软件启动计数器, 方式0和方式4: CPU把时间常数写入相应通道后, 计数器就开始工作, 在这种启动方式下, GATE要始终保持为高电平。

硬件启动计数器, 方式1和方式5: CPU把时间常数写入计数器后, 即使GATE为高电平, 计数器并不工作, 只有GATE发生跳变, 用其上升沿启动计数器。

OUT有两种输出形式, 要么是电平, 要么是负脉冲。前者有方式0和方式1, 后者有方式4和方式5。

# 8253的读/写操作

## ➤ 读出操作

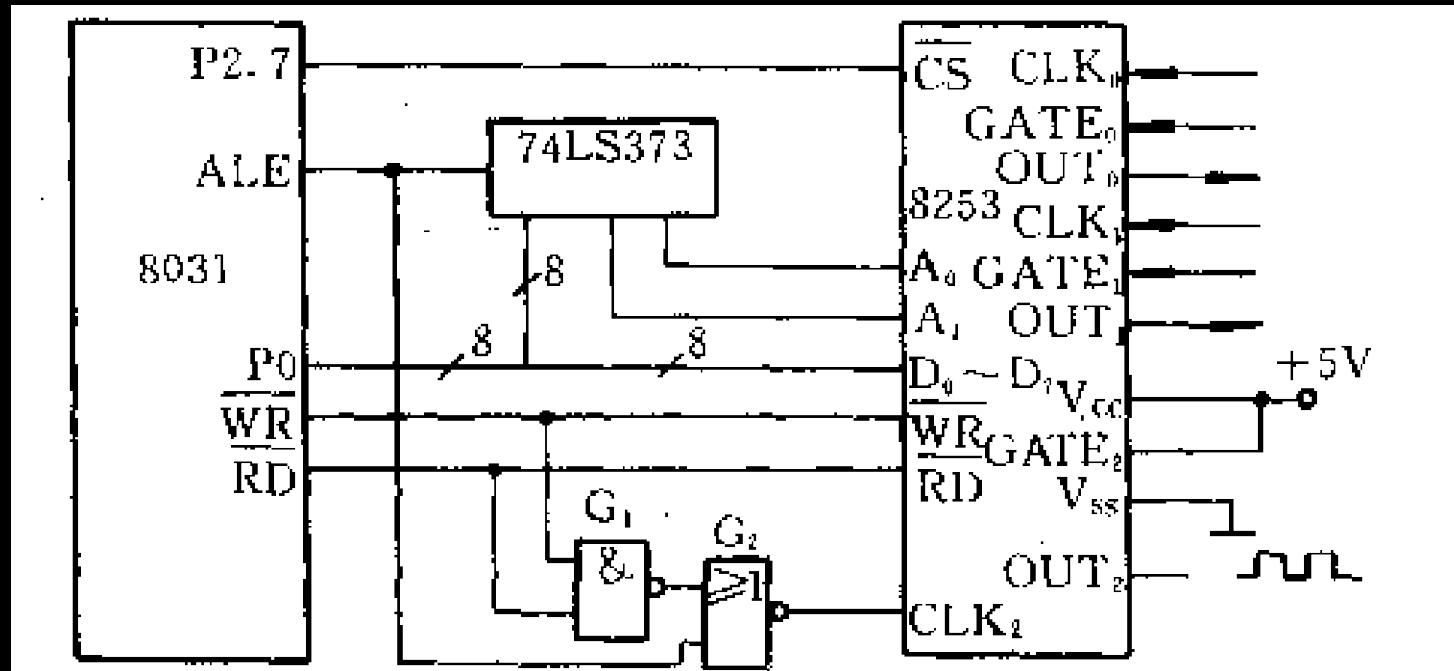
- (1) 简单的I/O读操作，被读计数器可用GATE终止计数过程，以便读出稳定的计数值；
- (2) 在不影响计数过程的情况下，RL1、RL0=00置计数器为闭锁操作，再执行读操作，8253内部逻辑将当前计数值锁定到锁存器中，以保证读到准确而稳定的计数值。

## ➤ 写入操作

- 8253每个计数器的编程应写入控制字和计数值。写入各个计数器的控制字时，在顺序上没有限制；但写入计数值时，16位计数值应分两次写入，并且写入操作必须按照控制字中RL1、RL0所规定的顺序进行。

## 应用：8031单片机控制8253输出方波信号

如果晶振频率 $f=12MHz$ , 8253计数器2设置成工作方式3, 要求在OUT2输出40kHz方波信号。



数据线D0~D7直接和8031 P0口相连；地址线A1、A0是P0口分时使用的8位地址经过74LS373锁存之后得到。片选端CS连P2.7，读写信号直接与单片机的读写线相连。

# 分析：

- ALE是地址锁存使能输出端，当CPU不访问外部存储器时，ALE端以1/6的时钟振荡频率固定地输出正脉冲，可以用它来作为计数器2的时钟输入。晶振频率 $f=12MHz$ ，则ALE输出脉冲的频率为 $f_{osc}/6=2MHz$
- 计数器2设置成工作方式3，要求输出40kHz的方波，计数初值为 $2MHz \div 40kHz = 50$  (32H)
- 根据题意，可以写出8253控制字为 B6H  
10 (选择计数器2)  
11 (先读/写低位字节，后读/写高位字节)  
011 (选择工作方式3) 0 (二进制计数)

实现方波输出的程序如下：

```
MOV  DPTR, #7FFFH ; 指向控制字寄存器
MOV  A, #0B6H      ; 计数器2输出方波控制字 10 11 011 0
MOVX @DPTR, A      ; 控制字送入控制字寄存器
MOV  DPTR, #7FFEH ; 指向计数器2
MOV  A, #32H       ; 50分频计数值为0032H
MOVX @DPTR, A      ; 先写入低8位值
CLR  A             ; 高8位值为00H
MOVX @DPTR, A      ; 再写入高8位值后,
                    ; OUT 2输出40kHz 方波信号
```

# 本章小结

- 了解MCS-51定时器/计数器的结构及其工作原理
- 掌握定时器/计数器的四种工作方式，并能熟练应用
- 了解定时器/计数器的扩展芯片8253及其应用

# 第九章 串行通信及其接口

串行通信的基本概念  
MCS-51的串行口



## § 9.1 串行通信基础

数据通信方式：

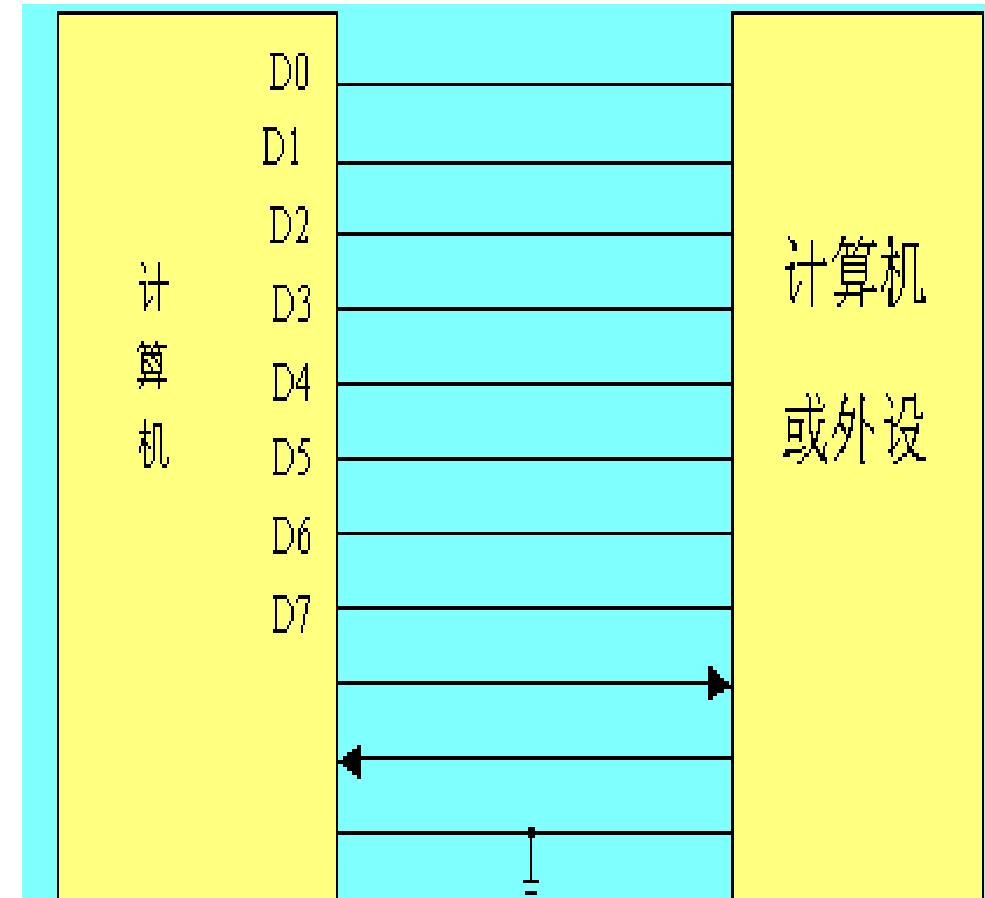
并行通信与串行通信

✓ 并行通信：一次传输8  
(16、32、64) 位

-8根数据线，1根控制线，

1根状态线，地线，11根

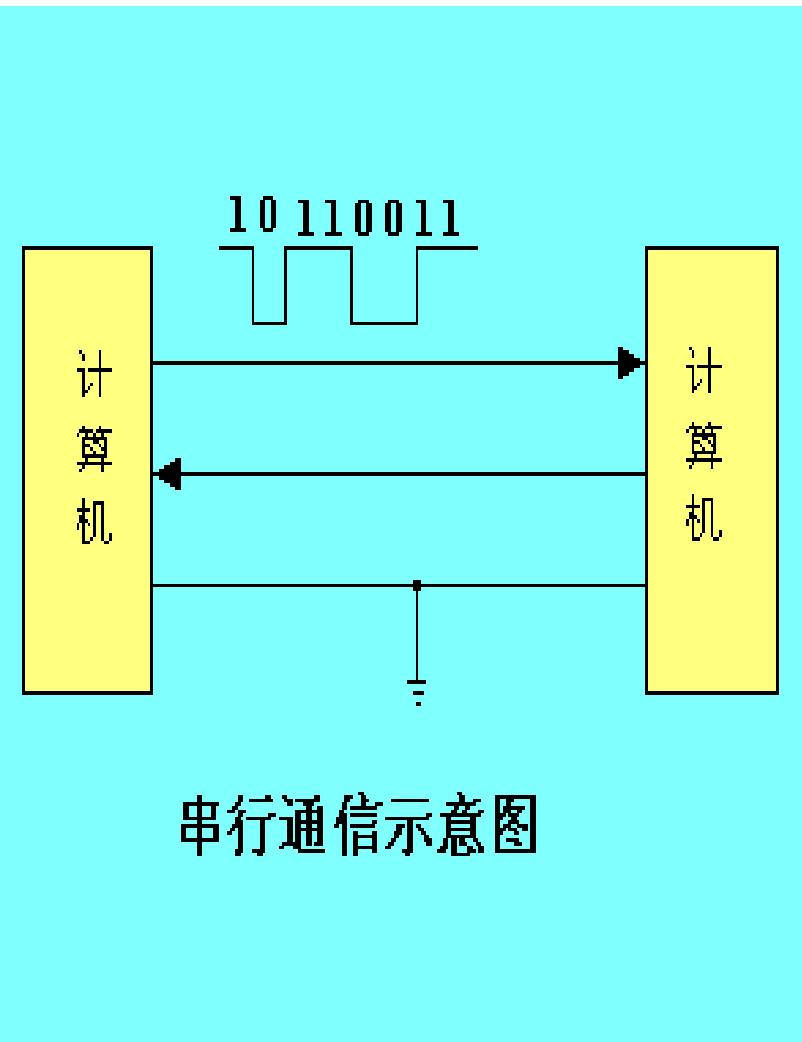
-特点：速度快，成本高，  
适合近距离传输



并行通信示意图

✓ 串行通信：  
数据一位一位地发送

- 3根线：一根发送线，  
一根接收线，地线
- 特点：硬件简单，适合  
远距离，对速度要求不高  
的场合



串行通信是将并行的数据一位一位地发送出去，接收方一位一位地接收数据，需要通信双方有一个协议，何时开始发送，何时发送完毕，接收方收到的信息是否正确等，这些信息只能以电平的高低来表示。

串行通信对信息的传送格式有固定的要求，分为异步通信和同步通信两种信息格式。



# 一、异步通信

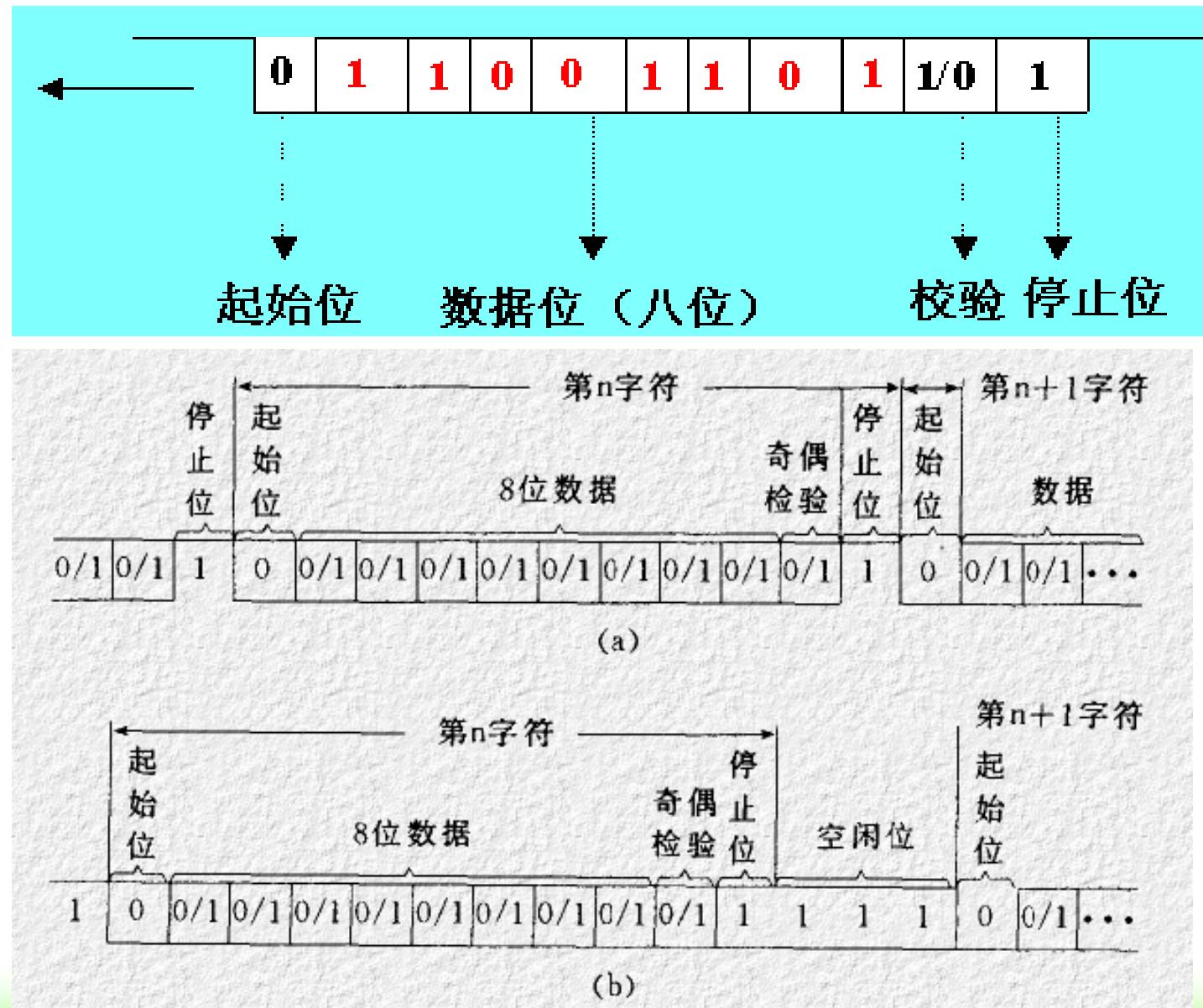
- 特点：以字符为单位，一个字符一个字符地传送，每个字符均有起始位、停止位作为开始和结束的标志。
- 异步串行通信规定了传输数据的结构即帧格式：



- **起始位:**
  - 在数据发送线上规定无数据时电平为1，当要发送数据时，首先发送一个低电平0，表示数据传送的开始，这就是起始位。
- **数据位:**
  - 真正要传送的数据，可以是5、6、7或8位，数据位是低位在前，高位在后。
- **奇偶校验位:**
  - 数据发送完后，发送奇偶校验位，以检验数据传送的正确性，这种校验方法是有限的，但是容易实现。
- **停止位:**
  - 表示数据传送的结束，可以是一位或两位。



## 异步通信 字符传送 帧格式：



## 二、同步通信

- 以数据块为单位的数据传送
- 同步通信先发送同步字符，之后连续发送数据，数据之间不能有间隔，直到数据发送完毕
- 一次性发送整块数据，速度要比异步通信快

面向位的同步协议数据桢格式：

8位	8位	8位	$\geq 0$ 位	16位	8位
01111110	A	C	I	FC	01111110
开始标志	地址场	控制场	信息场	校验场	结束标志

8位	8位	8位	≥0位	16位	8位
01111110	A	C	I	FC	01111110
开始标志	地址场	控制场	信息场	校验场	结束标志

- 开始标志字符和结束标志字符相同，构成每一帧数据的边界
- 地址场是规定接收方的地址
- 控制场规定通信时的命令
- 信息场是真正要传送的数据块

注：若无信息场，此帧为控制命令



8位	8位	8位	$\geq 0$ 位	16位	8位
01111110	A	C	I	FC	01111110
开始标志	地址场	控制场	信息场	校验场	结束标志

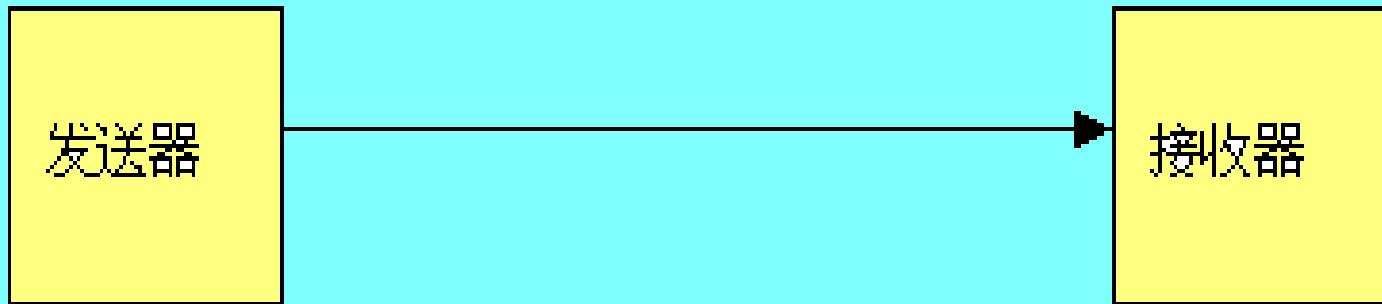
- **校验场**对数据帧进行校验。为区分传送的数据块中与开始标志或结束标志相同的字符，采用“0”插入和删除技术。即发送端在发送数据时，当遇到连续5个“1”，自动插入一个“0”。接收端接收数据时，当遇到连续的5个“1”，自动删除其后的“0”。这是为了保证数据的正确传送所规定的协议。



串行传输按数据传送方向分有三种——  
    单向、双向和半双向。

### 三、单工、半双工、全双工通信方式

1. 单工方式：一端是发送端，另一端是接收端，  
    数据只向一个方向传送。

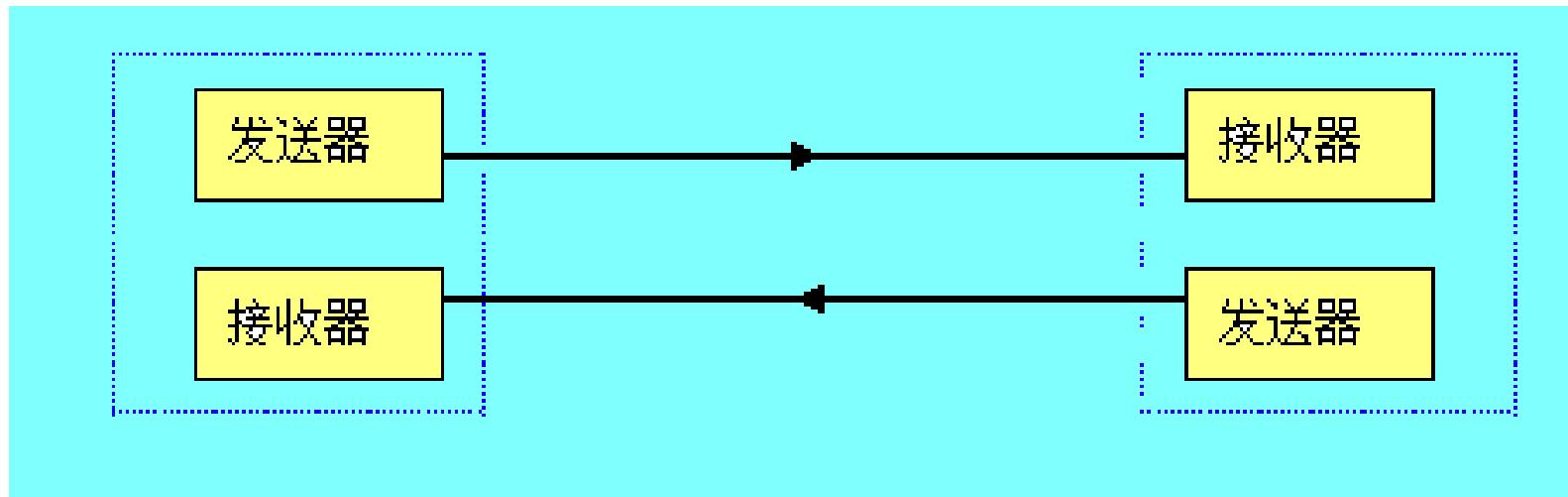


## 2、半双工方式

每端口有一个发送器和接收器，通过开关连接在线路上，数据可以在两个方向传送，但不能同时发送和接收。



### 3. 全双工方式



- 通信双方用两个独立的收发器单独连接，可以同时发送和接收数据。



## 四、 波特率 (Baud rate)

- 单位时间内传送的信息量，可以反映串行数据传送的速率。单位：位/秒， bps (bit per second)

假如数据传送的速率是120个字符 / 秒，每一个字符规定包含10个位(一个起始位、8个数据位和1个停止位)，则传送的波特率为：

$$10 \times 120 = 1200 \text{位 / 秒} = 1200 \text{bps}$$

- 传送过程中平均每位占用时间，即波特率的倒数

$$T_d = 1/1200 = 0.833 \text{ms}$$



## § 9.2 串行通信总线标准及接口

测控系统中，计算机通信主要采用异步串行通信方式，常用的异步串行通信接口标准有三种：

- RS-232 (RS-232A RS-232B RS-232C)
- RS-449 (RS-422 RS-423 RS-485)
- 20mA电流环 (非标准的串行接口，具有对电气噪声不敏感等优点)



# 一、通信接口的选取

## 1.1 通信速率和通信距离

降低通信速率，可以提高通信距离。

**RS-232C:** 最大速率20Kbit/s; 最大通信距离30m

**RS-485:** 10Mbit/s ; 300m  
90Kbit/s ; 1200m

## 1.2 抗干扰能力

采用标准的通信接口，本身具有一定的干扰能力，但工业现场的情况往往很恶劣，应根据具体情况进行选择。



## 二、RS-232C简介

是美国电子工业协会（EIA）公布的一种异步串行通信接口标准：

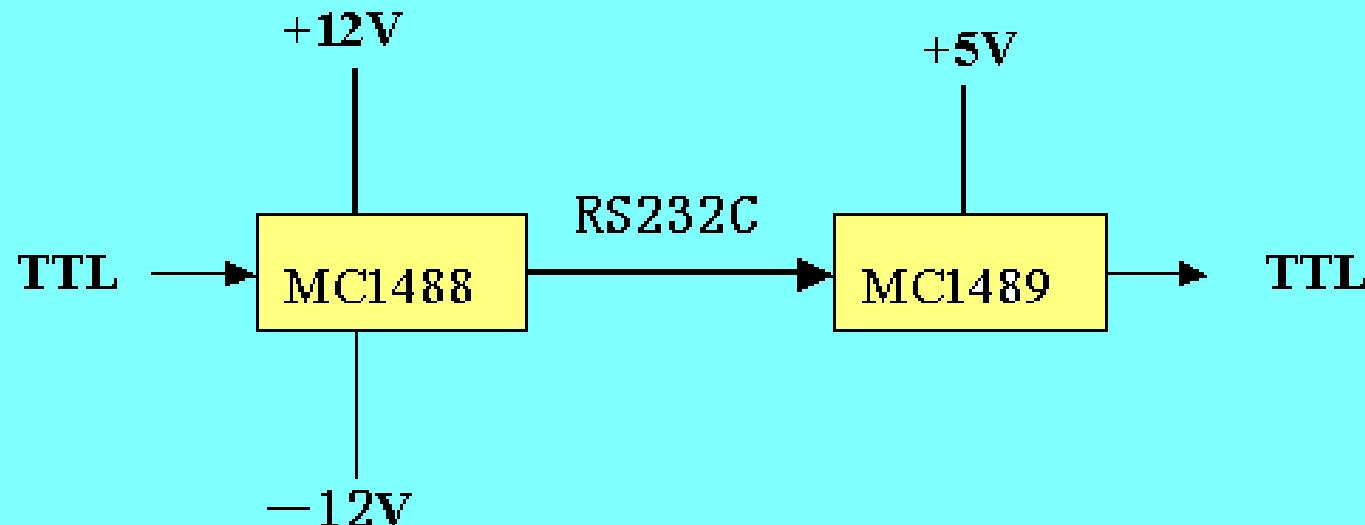
- 设备之间通信的距离小于30米
- 最大传输速率20Kbit/s
- 采用负逻辑：“1”—— $-5V \sim -15V$   
“0”—— $+5V \sim +15V$
- 不带负载时输出电平： $-25V \sim +25V$
- 输出短路电流： $< 0.5A$
- 最大负载电容： $2500pF$



为实现RS-232C电平与TTL电平接口，必须进行电平转换。

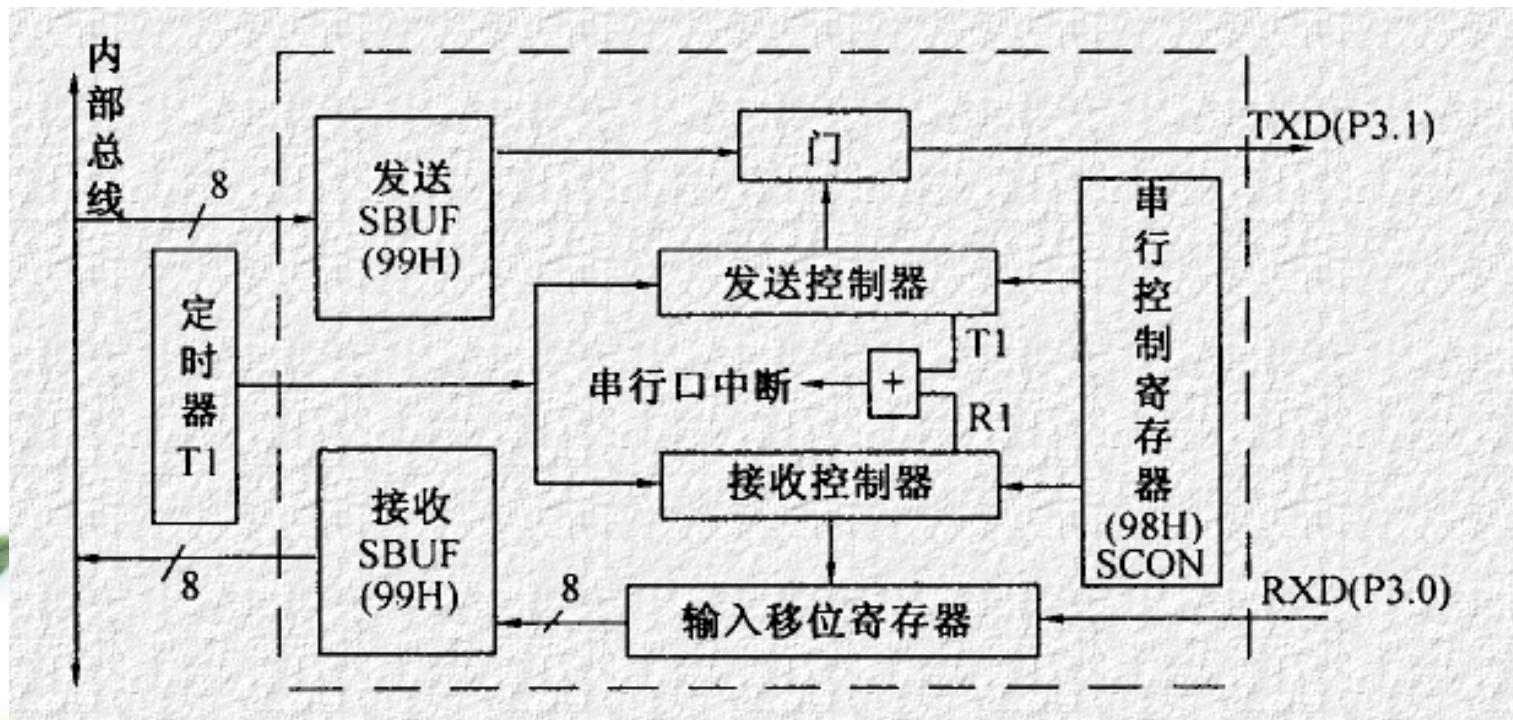
TTL电平可以由专用集成电路转换成RS232C标准；  
如: MC1488 或 75188                   TTL → RS232C

MC1489 或 75189                   RS232C → TTL



## § 9.3 MCS-51的串行口

8051内部有个可编程的全双工串行通信接口，可作UART（Universal Asynchronous Receiver/Transmitter，通用异步接收/发送器）用，也可作同步移位寄存器，有4种工作方式，并能设置各种波特率。

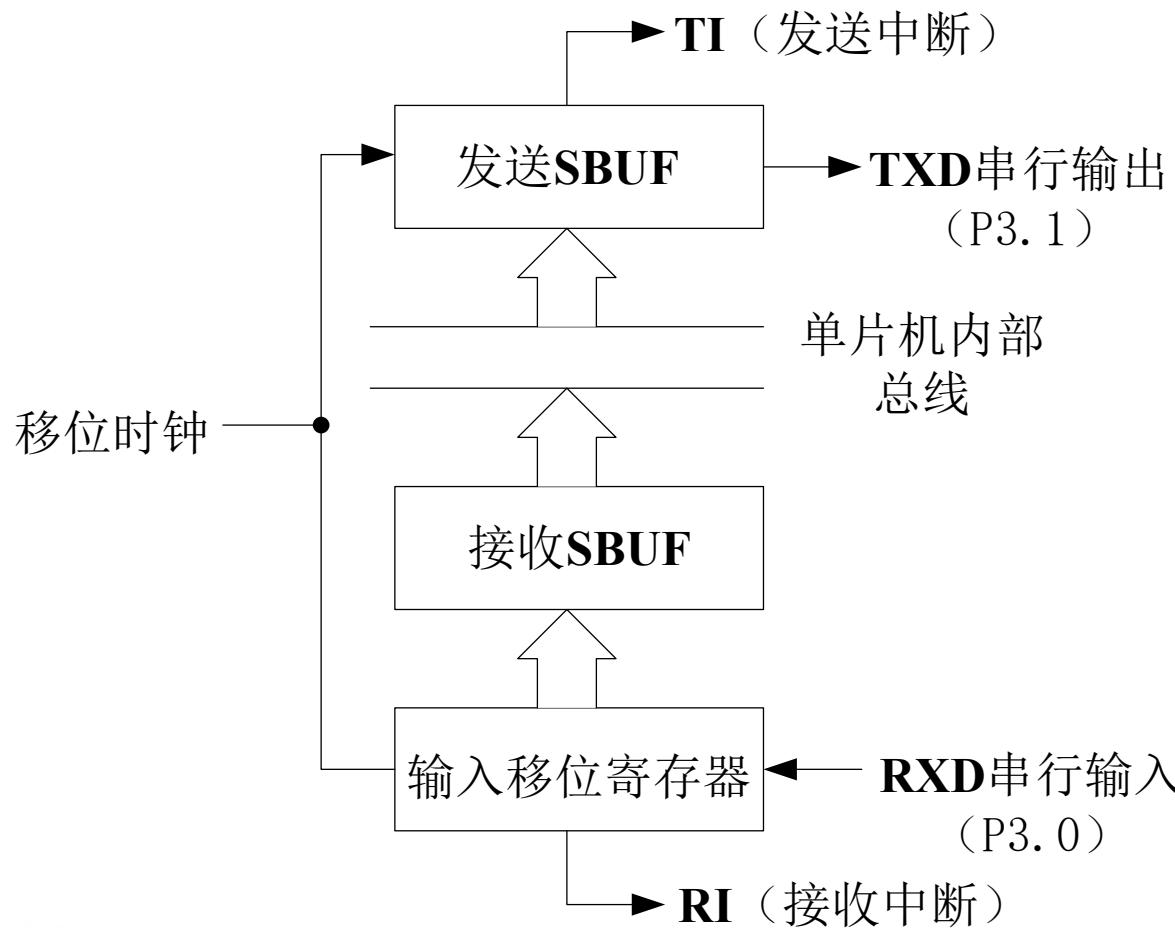


### § 9.3.1 与串行口有关的特殊功能寄存器

#### 1. 数据缓冲器SBUF

- 串行口缓冲器SBUF是可直接寻址的特殊功能寄存器，其内部RAM字节地址是**99H**。
- 在物理上，它对应着两个独立的寄存器，一个发送寄存器，一个接收寄存器，可**同时发送和接收数据**。
- 发送是CPU写入 SBUF的过程（51 系列单片机没有专门的启动发送状态的指令）。
- 接收是CPU读取 SBUF的过程。接收寄存器是双缓冲的，以避免在接收下一帧数据之前，CPU未能及时响应接收器的中断，没有把上一帧数据读走，而产生两帧数据重叠的问题。





当 RXD 串行输入到移位寄存器，即接收数据结束时，向 CPU 发出接收中断申请；当 TXD 串行输出，即发送数据结束时，向 CPU 发出发送中断申请。

MCS-51串行口寄存器结构

## 2. 串行口控制寄存器 (SCON)

SCON是MCS-51单片机的一个可位寻址的专用寄存器，用于串行数据通信的控制。字节地址98H。

位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
位符号	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

各位的说明如下：

1) SM0、SM1——串行口其状态组合和对应工作

SM0 SM1	工作方 方式0	方式0	移位寄存器方式
		方式1	波特率可变的8位异步串行通信接口
		方式2	9位异步串行通信接口
		方式3	波特率可变的9位异步串行通信接口

SM0 SM1

0 0

0 1

工作方  
方式0

方式1

方式1

1

1

方式3



位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
位符号	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

## 2) SM2——允许方式2、3的多机通信控制位

方式2和3中，若SM2=1且接收到的第九位数据（RB8）为1，才将接收到的前8位数据送入接收SBUF中，并置位RI产生中断请求；否则丢弃前8位数据。若 SM2=0，则不论第九位数据（RB8）为1还是为0，都将前8位送入接收SBUF中，并产生中断请求。

方式1时，若SM2=1，当接收到有效停止位时RI置1，以便接收下一帧数据。

方式0时，SM2应置0。

## 3) REN——允许接收位

REN=0 禁止接收数据

REN=1 允许接收数据

## 4) TB8——发送数据位

在方式2、3时，TB8的内容是要发送的第9位数据，其值由用户通过软件来设置（可以是奇偶校验位、数据/地址识别位）。

位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
位符号	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

### 5) RB8——接收数据位

在方式2、3时，RB8是接收的第9位数据。

在方式1时，若SM2=0，RB8是接收到的停止位。

在方式0时，不使用RB8。

复位时，SCON的所有位清0！

### 6) TI——发送中断标志位

在方式0时，发送完第8位数据后，该位由硬件置位。

在其它方式下，在发送停止位的开始时，由硬件置位。

TI=1表示帧发送结束，其状态既可供软件查询使用，也可请求中断。

TI必须由软件清“0”。

### 7) RI——接收中断标志位

在方式0时，接收完第8位数据后，该位由硬件置位。

在其它方式下，在接收到停止位的中间时刻，该位由硬件置位。

RI=1表示帧接收结束，其状态既可供软件查询使用，也可请求中断。

RI必须由软件清“0”。



## 3. 电源控制寄存器 (PCON)

PCON字节地址为87H，不可位寻址。是为CHMOS型单片机80C51的电源控制而设置的专用寄存器。

位序	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
位符号	SMOD	/	/	/	GF <sub>1</sub>	GF <sub>0</sub>	PD	IDL

与串行通信有关的只有D<sub>7</sub>位 (SMOD)，其他位是80C51的掉电方式控制位。D<sub>7</sub>位为波特率倍增位，当SMOD=1时，串行口波特率增加一倍；当SMOD=0时，串行口波特率为设定值，不加倍。系统复位时，SMOD=0。



## 9.3.2 MCS-51单片机串行通信工作方式

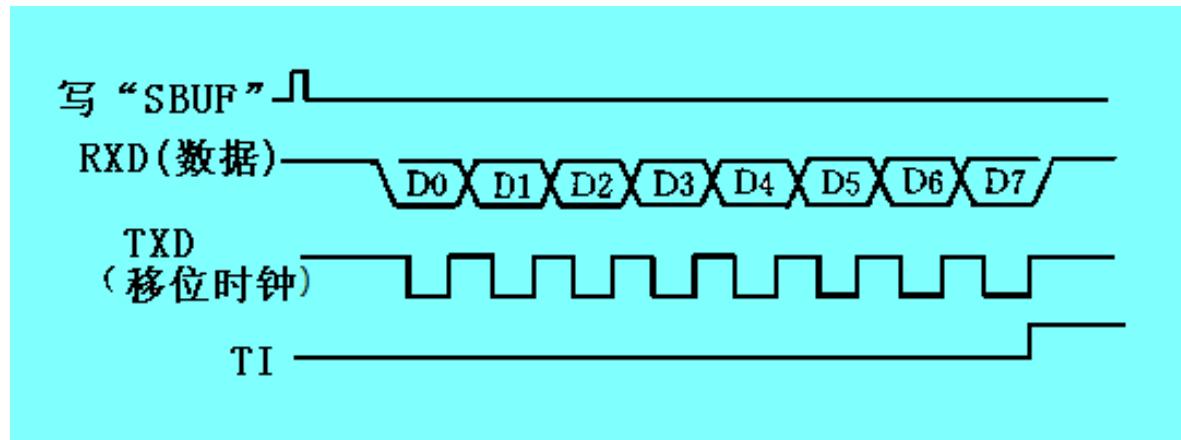
串行口的工作方式由SM0和SM1确定

SM0	SM1	方式	功能说明	波特率
0	0	方式0	移位寄存器方式	$f_{osc}/12$
0	1	方式1	8位异步串行通信	可变
1	0	方式2	9位异步串行通信	$f_{osc}/64$ 或者 $f_{osc}/32$
1	1	方式3	9位异步串行通信	可变

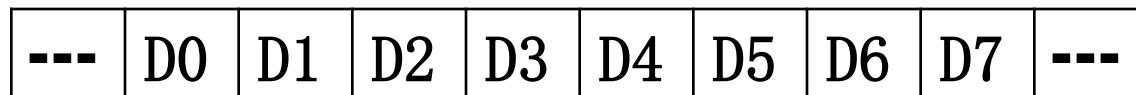
 方式0和方式2的波特率是固定的，而方式1和方式3的波特率是可变的，由T1的溢出率决定。

## 一、串行工作方式0

方式0为同步移位寄存器输入 / 输出方式，常用于扩展I / O口。串行数据通过RXD输入或输出，而TXD用于输出移位时钟，作为外接部件的同步信号。

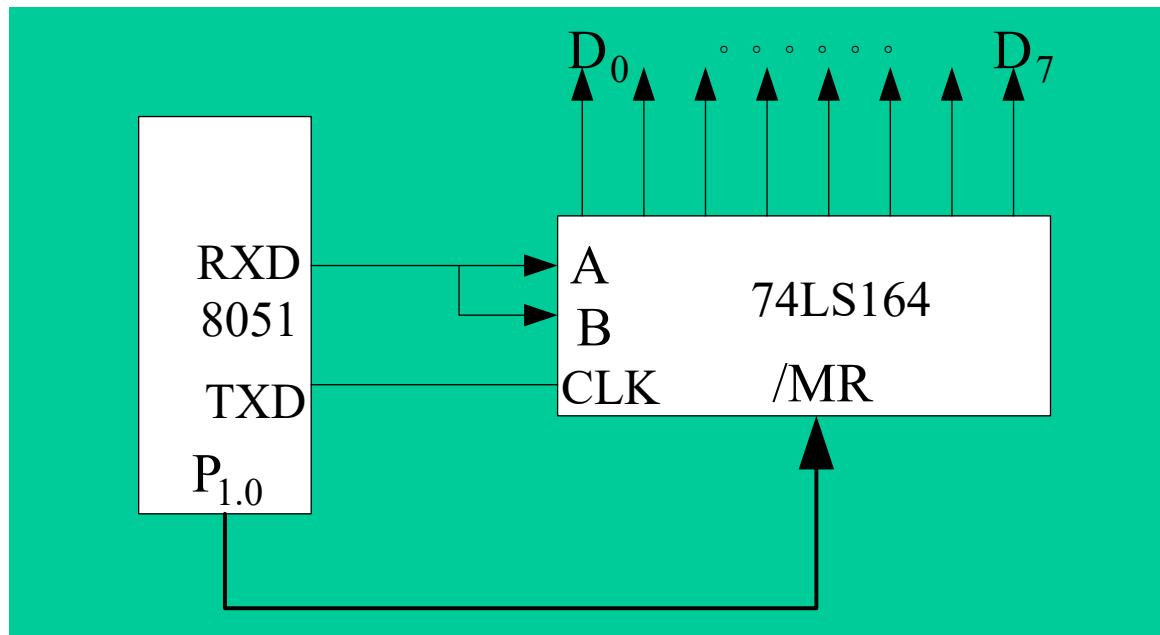


以8位数据为一帧、不设起始位和停止位：



## 1. 数据输出（发送）

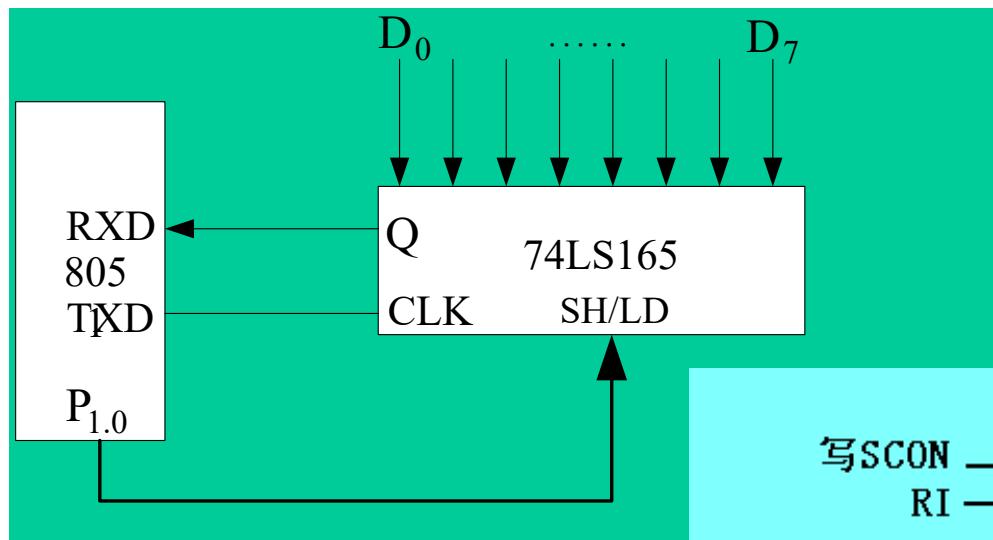
当数据写入SBUF后，数据从RXD端在移位脉冲（TXD）的控制下，逐位移入74LS164，74LS164能完成数据的串并转换。当8位数据全部移出后，TI由硬件置位，发出中断请求。若CPU响应中断，则从0023H单元开始执行串行口中断服务程序，数据由74LS164并行输出。



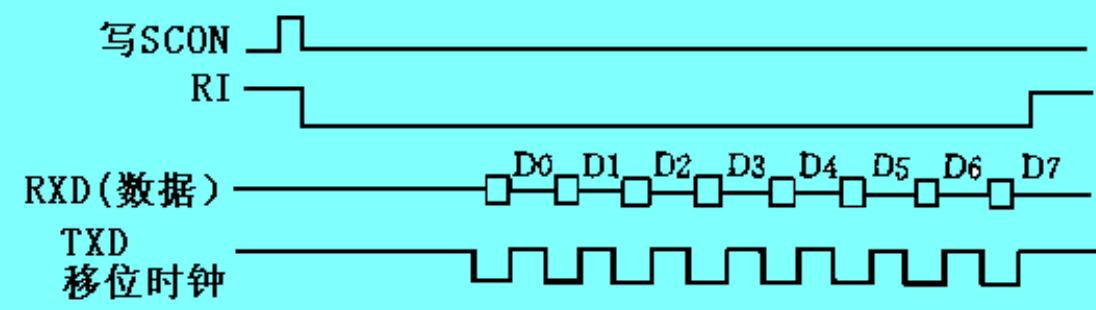
外接移位寄存器输出

## 2. 数据输入（接收）

要实现接收数据，必须首先把SCON中的允许接收位REN设置为1。当REN设置为1时，数据就在移位脉冲（TXD）的控制下，从RXD端输入。当接收到8位数据时，置位接收中断标志位RI，发生中断请求。通过外接74LS165，串行口能够实现数据的并行输入。

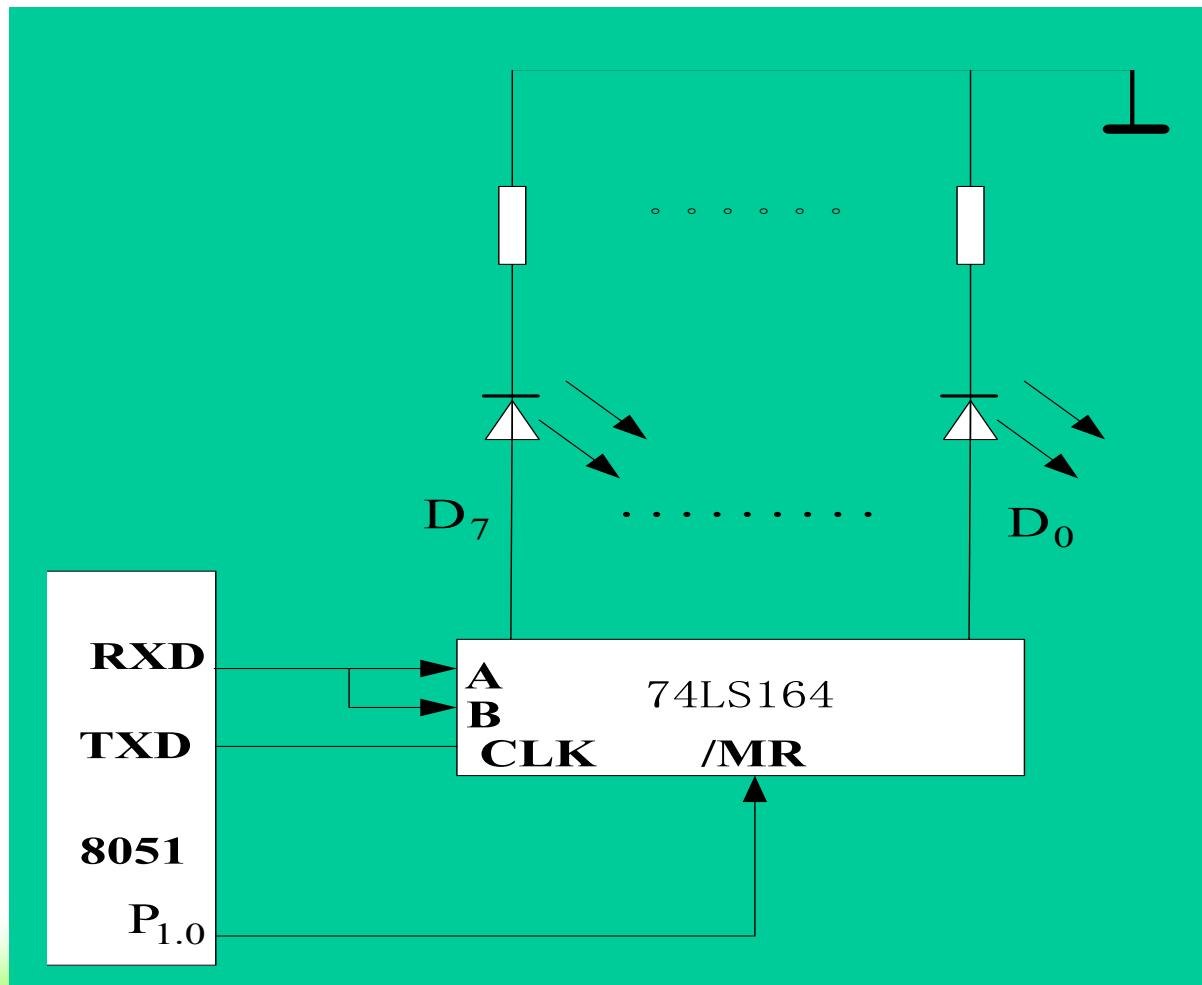


外接移位寄存器输入



【例】使用74LS164的并行输出端接8支发光二极管，利用它的串入并出功能，把发光二极管从左到右依次点亮，并反复循环。假定发光二极管为共阴极接法。

电路设计

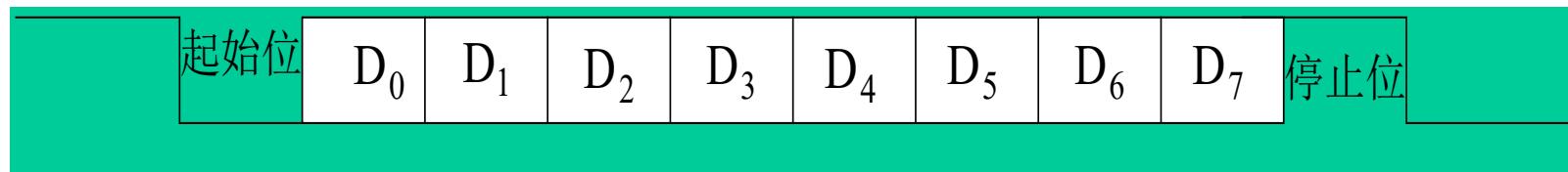


程序如下：

	ORG	0000H	
	LJMP	MAIN	
	ORG	1000H	
MAIN:	MOV	SCON, #00H	; 串行口工作在方式0
	CLR	ES	; 禁止串行中断
	MOV	A, #80H	; 发光二极管从左边亮起
DELR:	CLR	P1.0	; 关闭74LS164并行输出
	MOV	SBUF, A	; 串行输出
WAIT:	JNB	TI, WAIT	; 状态查询
	SETB	P1.0	; 开启74LS164并行输出
	ACALL	DELAY	; 调用延时子程序
	CLR	TI	; 清发送中断标志
	RR	A	; 发光右移
	AJMP	DELR	; 继续

## 二、串行工作方式1

方式1是波特率可变的8位异步串行通信方式。一帧为10位，其帧格式为1个起始位、8个数据位和1个停止位。



方式1的帧格式

### 1. 数据输出（发送）

数据写入SBUF后，开始发送，此时由硬件加入起始位和停止位，构成一帧数据，由TXD串行输出。输出一帧数据后，TXD保持在高电平状态，并将TI置位，通知CPU可以进行下一个字符的发送。



## 2. 数据输入（接收）

当REN=1且接收到起始位后，在移位脉冲的控制下，把接收到的数据移入接收缓冲寄存器（SBUF）中，停止位到来后，把停止位送入RB8中，并置位RI，通知CPU接收到一个字符。

以波特率的16倍速率采样RXD引脚，1至0负跳变时，启动接收器  
接收值是3次对起始位采样中的至少两次相同的值

满足RI=0，SM2=0（或停止位为1）两个条件时，接收过程正常进行（实现装载SBUF、RB8及置位RI）

## 3. 波特率的设定

$$\text{波特率} = \frac{2^{SMOD}}{32} \times (\text{定时器 } T1 \text{ 的溢出率})$$

其中，SMOD为PCON寄存器最高位的值，其值为1或0。



当定时器T1作波特率发生器使用时，选用工作方式2（即自动加载定时初值方式）。选择方式2可以避免通过程序反复装入定时初值所引起的定时误差，使波特率更加稳定。假定计数初值为X，则计数溢出周期为：

$$\frac{12}{f_{osc}} \times (256 - X)$$

溢出率为溢出周期的倒数。则波特率的计算公式为：

$$\text{波特率} = \frac{2^{SMOD}}{32} \times \frac{f_{osc}}{12 \times (256 - X)}$$

实际使用中，波特率是已知的。因此需要根据波特率的计算公式求定时初值X。用户只需要把定时初值设置到定时器**T1**，就能得到所要求的波特率。

### 4. 应用举例（用方式1实现双机串行通信）

#### （1）通信双方的硬件连接

两片8051的串行口直接相连，一片8051的TXD与另一片的RXD相连，RXD与另一片的TXD相连，地与地连通。8051串行口的输出是TTL电平，两片直接相连时所允许的距离极短。

#### （2）通信双方的软件约定

为实现双机通信，规定如下：

假定A机为发送机，B机为接收机。

当A机发送时，先送一个“AA”信号，B机收到后回答一个“BB”信号，表示同意接收。

当A机接收到“BB”后，开始发送数据，每发送一次求一次“校验和”，假定数据块长16个字节，起始地址为30H，一个数据块发送完后再发出“校验和”。



B机接收到数据并转存到数据区，起始地址也为30H，同时每接收一次也计算一次“校验和”，当一个数据块收齐后，再接收A机发来的“校验和”，并将它与B机的“校验和”进行比较。若两者相等，说明接收正确，B机回答一个00；若两者不相等，说明接收不正确，B机回答一个FF，请求重发。

A机收到00的答复后，结束发送。若收到的答复非0，则重新将数据发送一次。

双方均以1200bps的速率传送。假设晶振频率为6MHz，计算定时器T1的计数初值（为使波特率不倍增，设定PCON寄存器的SMOD=0，则 PCON=00H）：

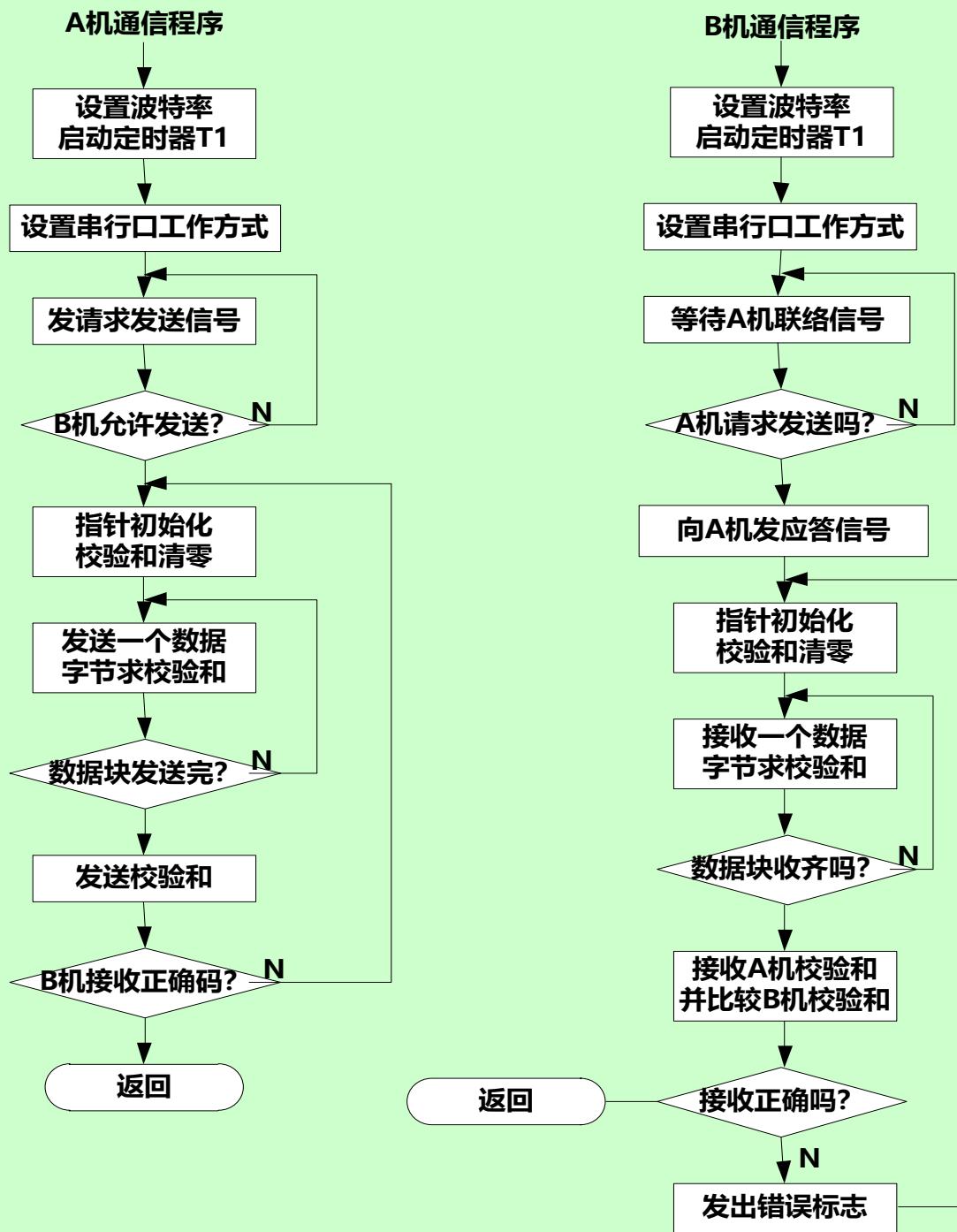
$$x = 256 - \frac{6 \times 10^6 \times 1}{12 \times 32 \times 1200} = 256 - 13 = 243 = 0F3H$$

### （3）基本的通信程序

设计程序框图如图所示。



# 双机通信程序流程图



## A机通信程序：

```
ASTART:    MOV    TMOD, #20H      ; 设定定时器1工作方式2
            MOV    TL1, #0F3H      ; 设定计数初值
            MOV    TH1, #0F3H      ; 计数重装值
            SETB   TR1          ; 启动定时器T1工作
            MOV    PCON, #00H      ; SMOD=0, 波特率不倍增
            MOV    SCON, #50H      ; 设置方式1, 允许接收
ATT1:      MOV    SBUF, #0AAH      ; 发送请求信号“AA”
AWAIT1:    JBC    TI, ARR1      ; 发送完成TI=1, 则置0
            SJMP   AWAIT1      ; 数据未发送完, 则等待
ARR1:      JBC    RI, ARR2      ; 等待应答信号, RI=1
            SJMP   ARR1       ; 无应答信号, 则等待
```

ARR2: **MOV** A, SBUF ; 检查接收数据缓冲器内容  
**XRL** A, #0BBH ; 与0BBH做异或  
**JNZ** ATT1 ; 判断是否是应答信号“BB”

ATT2: **MOV** R0, #30H ; 数据块起始地址  
**MOV** R7, #10H ; 设置数据块长度, 16个字节  
**MOV** R6, #00H

ATT3: **MOV** SBUF, @R0 ; 数据块内容送接收缓冲器  
**MOV** A, R6  
**ADD** A, @R0 ; 做校验和  
**MOV** R6, A  
**INC** R0

AWAIT2: **JBC** TI, ATT4 ; 等待一帧数据块发送完  
**SJMP** AWAIT2

ATT4: **DJNZ** R7, ATT3 ; 判断是否传送完毕  
**MOV** SBUF, R6 ; 发送校验和

**AWAIT3:** JBC TI, ARR3 ; 等待校验和传送结束  
SJMP AWAIT3 ;  
**ARR3:** JBC RI, ARR4 ; 等待B机的应答信号  
SJMP ARR3 ;  
**ARR4:** MOV A, SBUF ; B机应答信号送到A  
JNZ ATT2 ; A不为0, 则需重传  
**AEND:** RET

B机通信程序:

**BSTART:** MOV TMOD, #20H ; 设定定时器1工作方式2  
MOV TL1, #0F3H ; 设定计数初值  
MOV TH1, #0F3H ; 计数重装值  
SETB TR1 ; 启动定时器T1工作  
MOV PCON, #00H ; SMOD=0, 波特率不倍增  
MOV SCON, #50H ; 设置方式1, 允许接收



<b>BRR1:</b>	<b>JBC</b>	<b>RI,BRR2</b>	； 等待A机请求信号 接收， RI=1
	<b>SJMP</b>	<b>BRR1</b>	； 等待
<b>BRR2:</b>	<b>MOV</b>	<b>A,SBUF</b>	； 把接收到的数据送入A
	<b>XRL</b>	<b>A,#0AAH</b>	； 判断接收到数据是否是 “AA”
	<b>JNZ</b>	<b>BRR1</b>	； 如果不是继续等待
<b>BTT11:</b>	<b>MOV</b>	<b>SBUF,0BBH</b>	； 发送应答信号
<b>BWAIT1:</b>	<b>JBC</b>	<b>TI, BRR3</b>	； 发送完毕， TI=1， 置0
	<b>SJMP</b>	<b>BWAIT1</b>	
<b>BRR3:</b>	<b>MOV</b>	<b>R0,#30H</b>	； 接收有效数据
	<b>MOV</b>	<b>R7,#10H</b>	； 设置数据块长度， 16个字节
	<b>MOV</b>	<b>R6, #00H</b>	
<b>BRR4:</b>	<b>JBC</b>	<b>RI, BRR5</b>	； 等待接收完毕
	<b>SJMP</b>	<b>BRR4</b>	

**BRR5:**      **MOV    @R0, SBUF** ; 数据块内容送接收缓冲器  
                **MOV    A, R6**  
                **ADD    A, @R0**                ; 做校验和  
                **MOV    R6, A**  
                **INC    R0**  
                **DJNZ    R7, BRR5**

**BWAIT2:**      **JBC    RI, BRR6**        ; 等待接收“校验和”  
                **SJMP    BWAIT2**

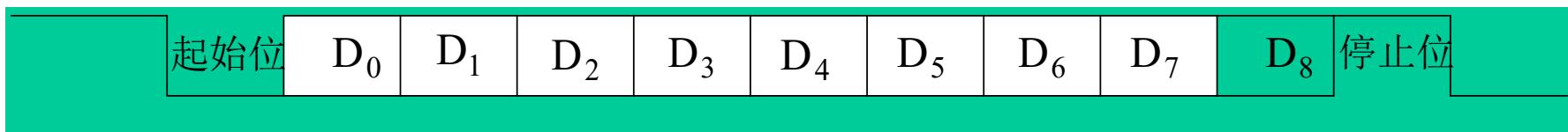
**BRR6:**      **MOV    A, SBUF**  
                **XRL    A, R6**                ; 判断接收到校验和是否相等  
                **JZ    BEND**  
                **MOV    SBUF, #0FFH**

**BWAIT3:**      **JBC    TI, BRR3**        ; 等待发送完毕  
                **SJMP    BWAIT3**

**BEND:**      **MOV    SBUF, #00H**  
                **RET**

### 三、串行工作方式2

方式2是9位异步串行通信接口。其帧格式为1个起始位、9个数据位和1个停止位。



方式2的帧格式

在方式2下，字符是8个数据位，增加了一个第9个数据位( $D_8$ )，其功能由用户确定，是一个可编程位。

在发送数据时，应先在SCON的TB8位中把第9个数据位的内容准备好。可使用如下指令完成：

SETB TB8 ; TB8位置“1”

CLR TB8 ; TB8位置“0”



发送数据（D<sub>0</sub>~D<sub>7</sub>）由MOV指令向SBUF写入，D<sub>8</sub>位的内容由硬件电路从TB8中直接送到发送移位寄存器的第九位，并以此来启动串行发送。一个字符帧发送完毕后，将TI位置“1”，其它过程与方式1相同。

方式2的接收过程也与方式1基本类似，不同之处在于第9数据位，串行口把接收到的前8个数据位送入SBUF，而把第九数据位送入RB8。

方式2的波特率是固定的，有两种：一种是晶振频率的1/32；另一种是晶振频率的1/64。即fosc/32和fosc/64。

$$\text{波特率} = \frac{2^{SMOD}}{64} \times fosc$$



当**SMOD**为0时，波特率为fosc/64

当**SMOD**为1时，波特率为fosc/32

## 四、串行工作方式3

工作方式3是波特率可变的9位异步串行通信方式。方式3同方式2类似，但方式3的波特率可变，由用户来确定。其波特率的确定同方式1。

$$\text{波特率} = \frac{2^{SMOD}}{32} \times (\text{定时器1的溢出率})$$

$$\text{即: 波特率} = \frac{2^{SMOD}}{32} \times \frac{f_{osc}}{12 \times (256 - X)}$$



### 9.3.3 不同工作方式下波特率设定方法

串行口每秒钟发送或接收的数据位数称为波特率。假设发送一位数据所需时间为T，则波特率为  $1/T$ 。

(1) 方式 0 的波特率等于单片机晶振频率的  $1/12$ , 即每个机器周期接收或发送一位数据。

(2) 方式 2 的波特率与电源控制器PCON的最高位SMOD的写入值有关:

$$\text{方式2的波特率} = \text{晶振频率} \times \frac{2^{SMOD}}{64}$$

即  $SMOD=0$ , 波特率为  $(1/64) f_{OSC}$

  $SMOD=1$ , 波特率为  $(1/32) f_{OSC}$

(3) 方式1 和方式3 的波特率除了与SMOD位有关之外，还与定时器 T1 的溢出率有关。定时器 T1 作为波特率发生器，常选用定时方式2（8位重装载初值方式），并且禁止 T1 中断。此时 TL1 从初值计数到产生溢出，它每秒钟溢出的次数称为溢出率。

$$\begin{aligned}\text{方式1或3的波特率} &= T1\text{的溢出率} \times \frac{2^{SMOD}}{32} \\ &= \frac{2^{SMOD}}{32} \times \frac{f_{OSC}}{12 \times (256 - TH1)}\end{aligned}$$



表 9.3 定时器T1产生的常用波特率

串行口 模 式	波特率 /MHz	晶振频率 $f_{osc}/\text{MHz}$	SMOD	定时器 T1		
				C/T	定时器方式	重装载值
模式 0	最大 1 M	12	×	×	×	×
模式 2	最大375 k	12	1	×	×	×
模式1 或 模式3	62.5 k	12	1	0	2	FFH
	19.2 k	11.059	1	0	2	FDH
	9.6 k	11.059	0	0	2	FDH
	4.8 k	11.059	0	0	2	FAH
	2.4 k	11.059	0	0	2	F4H
	1.2 k	11.059	0	0	2	E8H
	137.5	11.986	0	0	2	1DH
	110	6	0	0	2	72H
	110	12	0	0	1	FEEBH

例：假设某MCS-51单片机系统，串行口工作于方式3，要求传送波特率为1200Hz，作为波特率发生器的定时器T1工作在方式2时，请求出计数初值为多少？设单片机的振荡频率为6MHz。

因为串行口工作于方式3时的波特率为：

$$\text{方式3的波特率} = \frac{2^{SMOD}}{32} \times \frac{f_{osc}}{12 \times (256 - TH1)}$$

所以  $TH1 = 256 - \frac{f_{osc}}{\text{波特率} \times 12 \times (32 / 2^{SMOD})}$



当SMOD=0 时, 初值TH1=256-6×10<sup>6</sup>/(1200×12×32/1)

=243=0F3H

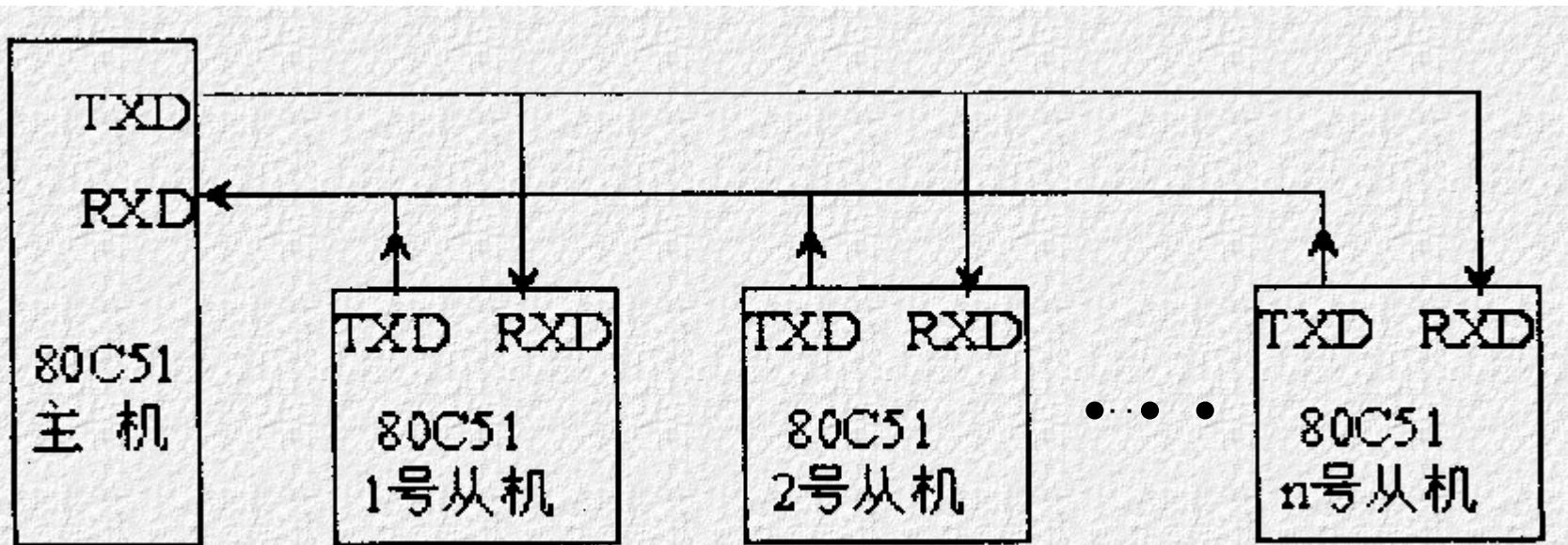
当SMOD=1 时, 初值TH1=256-6×10<sup>6</sup>/(1200×12×32/2)

=230=0E6H



### 9.3.4 多机通信

- 在实际应用中，如果不只使用一个单片机，就需要几个单片机协同工作。
- 分布式多机系统，采用一台主机和多台从机，主机控制各个从机之间的信息交换。主机的RXD端与所有从机的TXD端相连，主机的TXD端与所有从机的RXD端相连。主机发送的信息可以被各个从机接收，而各个从机发送的信息只能被主机接收。



- 可利用串行口控制寄存器SCON中的多机通信控制位SM2和TB8来实现多机通信。

当串行口以方式2或方式3工作时，发送和接收的每一帧信息都是11位，其中第9位是可编程数据位，即TB8。定义**TB8=1**，为**地址帧**；**TB8=0**，为**数据帧**。

- ✓ 当从机的控制位SM2=1时，如果接收的是地址帧，就把数据装入SBUF，并置接收中断标志位RI=1，向CPU发出中断请求；如果接收的是数据帧，则拒收，CPU不做任何处理。
- ✓ 若SM2=0，则无论是地址帧还是数据帧，都会把数据装入SBUF，然后置接收中断标志位RI=1。



## 一般通信过程：

1. 所有从机SM2置1，处于接收地址帧状态
2. 主机发送地址帧信息（TB8=1）
3. 从机接收地址帧后，与自身地址进行比较
4. 被寻址的从机清除其SM2(=0)，其它从机维持SM2=1
5. 主机发送数据信息（TB8=0）,被寻址的主机由于SM2=0，接收，其它从机（SM2=1）拒收
6. 主机与其它从机联系时，再发地址帧，先被寻址的从机恢复SM2=1

要保证通信的可靠性，必须有严格的通信协议

# 本章小结

- 了解串行通信的基本概念
- 掌握MCS-51单片机串行通信的四种工作方式
- 理解多机通信的通信过程



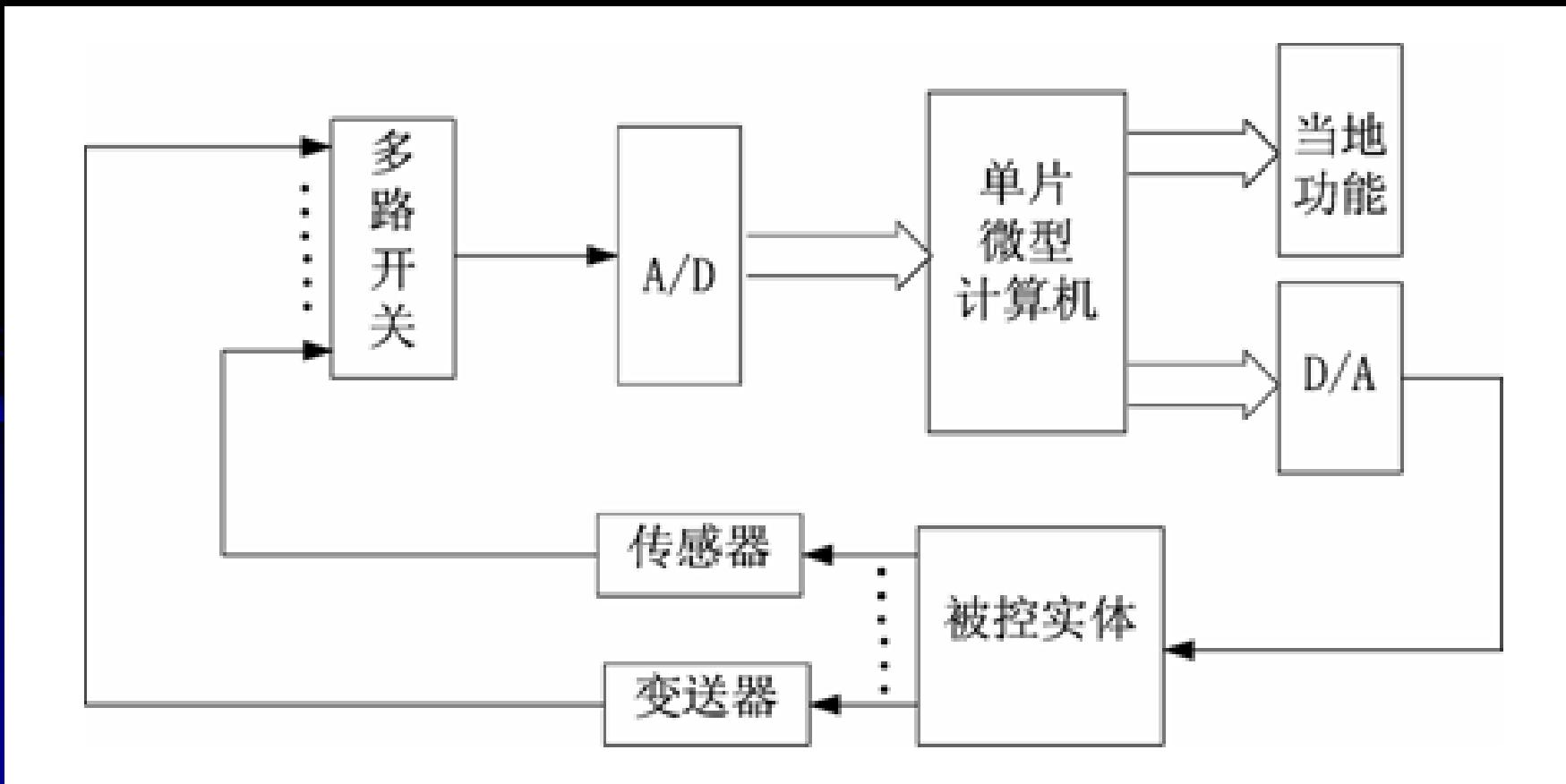
# 第10章 MCS-51单片机数/模 和模/数转换接口

- D/A转换器及其与单片机接口
- A/D转换器及其与单片机接口

# 概述

数/模转换器实现数字量到模拟量(简称D/A)的转换

模/数转换器实现模拟量到数字量(简称A/D)的转换



# D/A转换器和A/D转换器的主要性能指标

## 一、分辨率(Resolution)

- 对于D/A转换器，分辨率反映了输出模拟电压的最小变化量
- 对于A/D转换器，分辨率反映了输出数字量变化一个相邻数码所需输入的模拟电压的变化量

表 10.1.1 A/D 转换器或 D/A 转换器的分辨率与位数的关系

位数 n	分辨率	
	分数	%满刻度电压(近似)
8	1/255	0.4
9	1/511	0.2
10	1/1023	0.1
11	1/2047	0.05
12	1/4095	0.024
13	1/8191	0.012
14	1/16383	0.006
15	1/32767	0.003
16	1/65535	0.0015

$$\text{分辨率} = \frac{\text{满刻度电压}}{2^n - 1}$$

n-A/D转换器或D/A  
转换器中对应二进  
制代码的位数

## 二、量化误差(Quantizing Error)

一个分辨率有限的A/D转换器在进行A/D转换时，把采样电压化为某个规定的最小数量单位(即量化单位)的整数倍，就是量化。量化过程需要取整，存在量化误差。

## 三、线性度(Linearity)

线性度是实际的转换特性曲线与理想的转换特性曲线之间的最大偏移量。

## 四、绝对精度(Absolute Accuracy)

整个工作区间内实际的输出电压与理想的输出电压之间的最大偏差。

## 五、建立时间(Setting Time)

D/A转换器输入数码变化时，模拟输出电压也跟着变化，经过一定时间后新的模拟电压才能稳定下来。

## 六、转换时间(Conversion Time)

A/D转换器完成一次从模拟量的采样到数字量的编码所需的时间。

# D/A转换器

D/A转换器的工作是将离散的数字信号转换为连续变化的模拟信号。

## ● **D/A转换的基本工作原理：**

数字量是由一位一位的数位构成，每个数位代表一定的权值。为了把一个数字量转换成模拟量，必须把每一位的数码按照权来转换成对应的模拟分量，再把各模拟分量相加，得到与数字量成正比的总模拟量，从而实现数字到模拟的转换。

常用**电阻分压/分流**来实现D/A转换。

D/A转换器将数字信息转换成与数值成正比的电压/电流。有权电阻解码网络与 T型解码网络两种构建方法。

### 权电阻解码网络

简单。但随着D/A转换的位数增加，权电阻值跨度增大，在集成电路中难于实现。

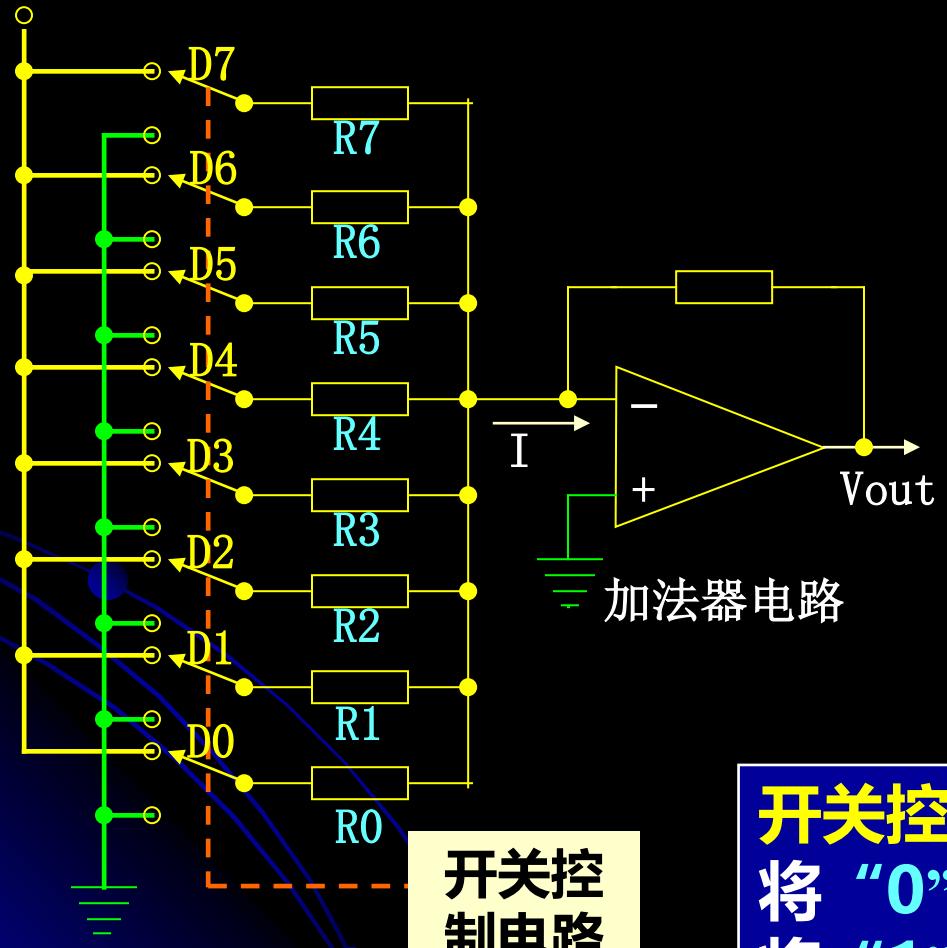
### T型解码网络

电阻数量要多用一倍。但电阻值归一化程度高,容易集成，精度高。应用最为普遍。

# 权电阻解码网络

参考电压

$V_{ref}$

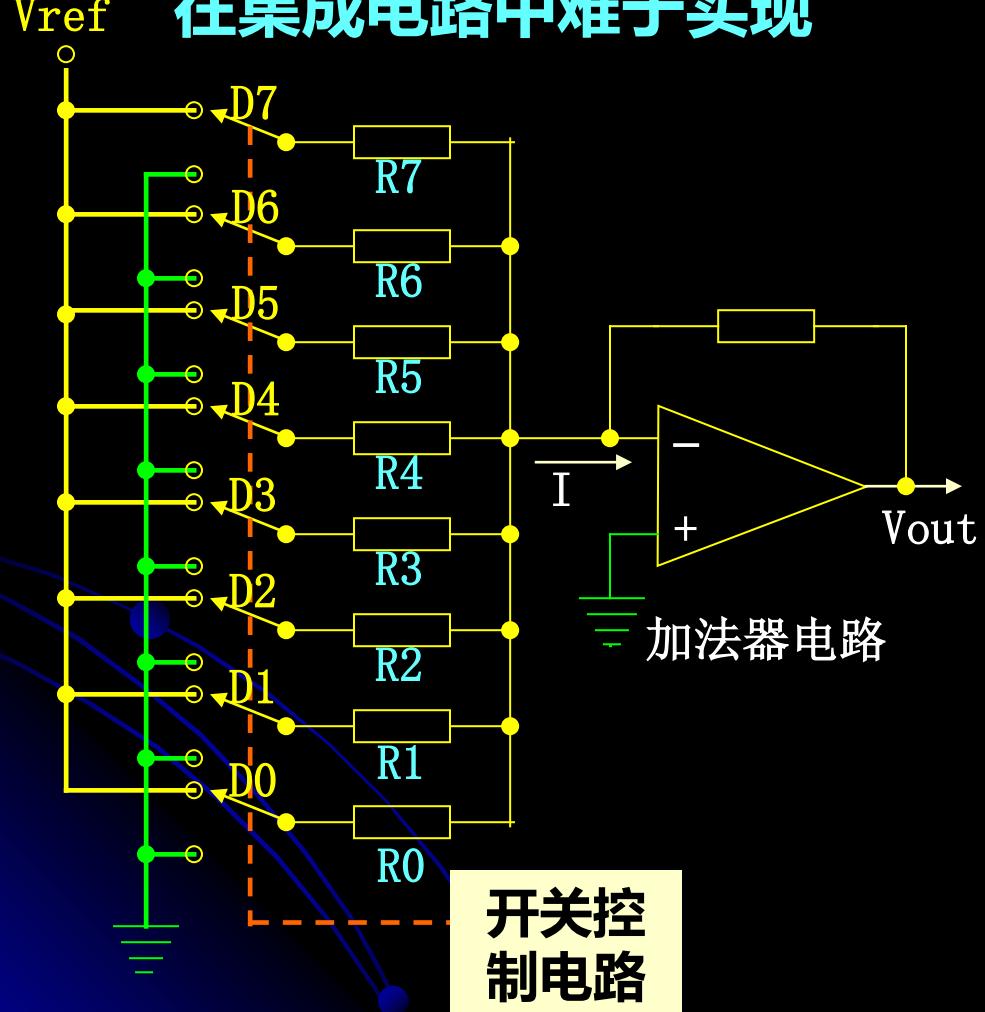


由输入的二进制数的各位来控制相应的电子开关，通过权电阻组成的解码网络，在运算放大器的输入端产生与二进制数各位的权成比例的电流，经过运算放大器相加求和，从而转换成与二进制数成正比的模拟电压。

**开关控制电路的作用：**  
**将“0”值对应的开关接地，**  
**将“1”值对应的开关接通 $V_{ref}$**

# 权电阻解码网络

简单, 权电阻值跨度大,  
在集成电路中难于实现



若:  $R_0 = R$

$$R_1 = R/2$$

$$R_2 = R/4$$

$$R_3 = R/8$$

.....

$$R_7 = R/128$$

从而:  $I_0 = V_{ref}/R$

$$I_1 = 2 V_{ref}/R$$

$$I_2 = 4 V_{ref}/R$$

$$I_3 = 8 V_{ref}/R$$

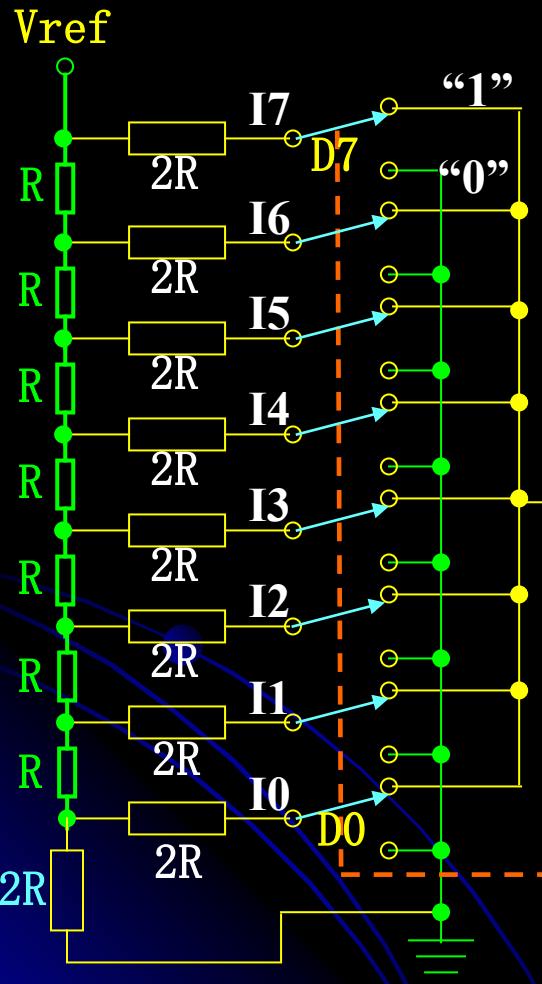
.....

$$I_7 = 128 V_{ref}/R$$



# T型解码网络

电阻数量增大一倍, 但阻值归一, 集成容易, 精度高



深度负反馈条件下,运放的“虚短特性”：  
 $V_d \approx 0$ , (因开环增益极大,输入端之间压差极小)  
 $I_i \approx 0$ , (因输入阻抗极大,输入电流极小)  
从而: 由节点向下看等效电阻均为R

每个2R支路的电流均为  
上一支路的1/2:

$$I_7 = (V_{ref}/2R)$$

$$I_6 = (V_{ref}/2R)/2$$

$$I_5 = (V_{ref}/2R)/4$$

.....

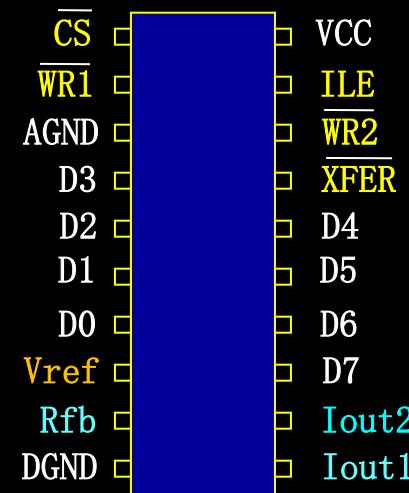
$$I_0 = (V_{ref}/2R)/128$$

开关控制电路:“0”值将开关接地; “1”值将开关接通Vref

## ● 8位CMOS数模转换芯片 DAC 0832：

- ✓ 8位D/A, 分辨率 =  $V_{ref}/255$
- ✓ CMOS低功耗器件,  $+5 \sim +15V$
- 单电源供电
- ✓ 电流输出型器件(需外接运放)
- ✓ 具有双缓冲控制输出
- ✓ 采用T型电阻解码网络结构
- ✓ 参考电压源,  $-10 \sim +10V$

DAC0832



20 PIN DIP封装

# ● DAC 0832 引脚定义

**D0—D7:** 8位数字量输入端

**/CS:** 输入选通, 低有效

**ILE:** 数据锁存允许, 高有效

**/WR1:** 写控制信号1, 低有效

**/WR2:** 写控制信号2, 低有效

**/XFER:** 数据传送控制信号

**Iout1:** 电流输出端1

**Iout2:** 电流输出端2

**Rfb:** 内置反馈电阻端

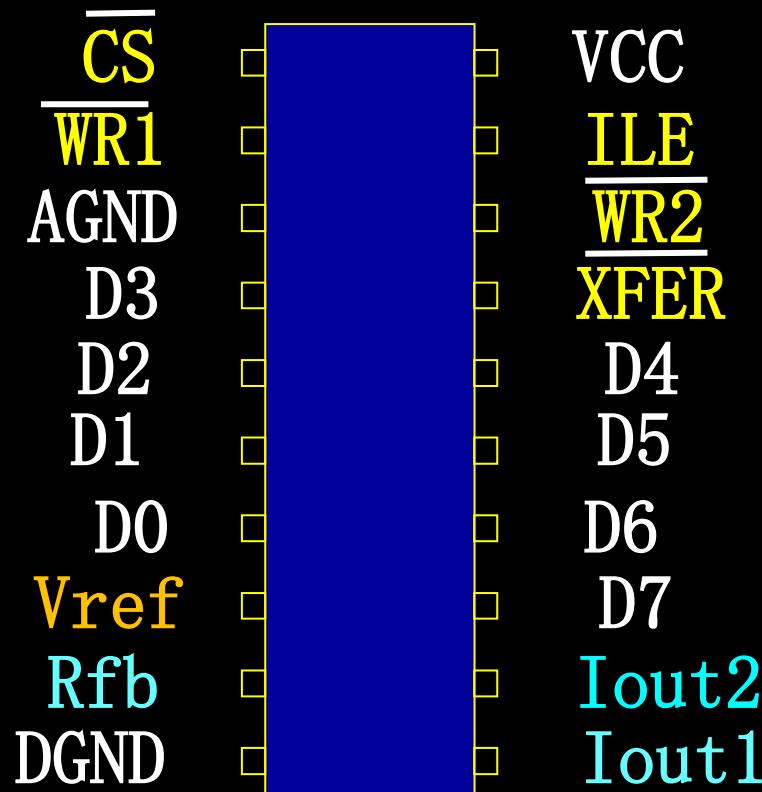
**Vref:** 参考电压源, -10~+10V

**DGND:** 数字量地

**AGND:** 模拟量地

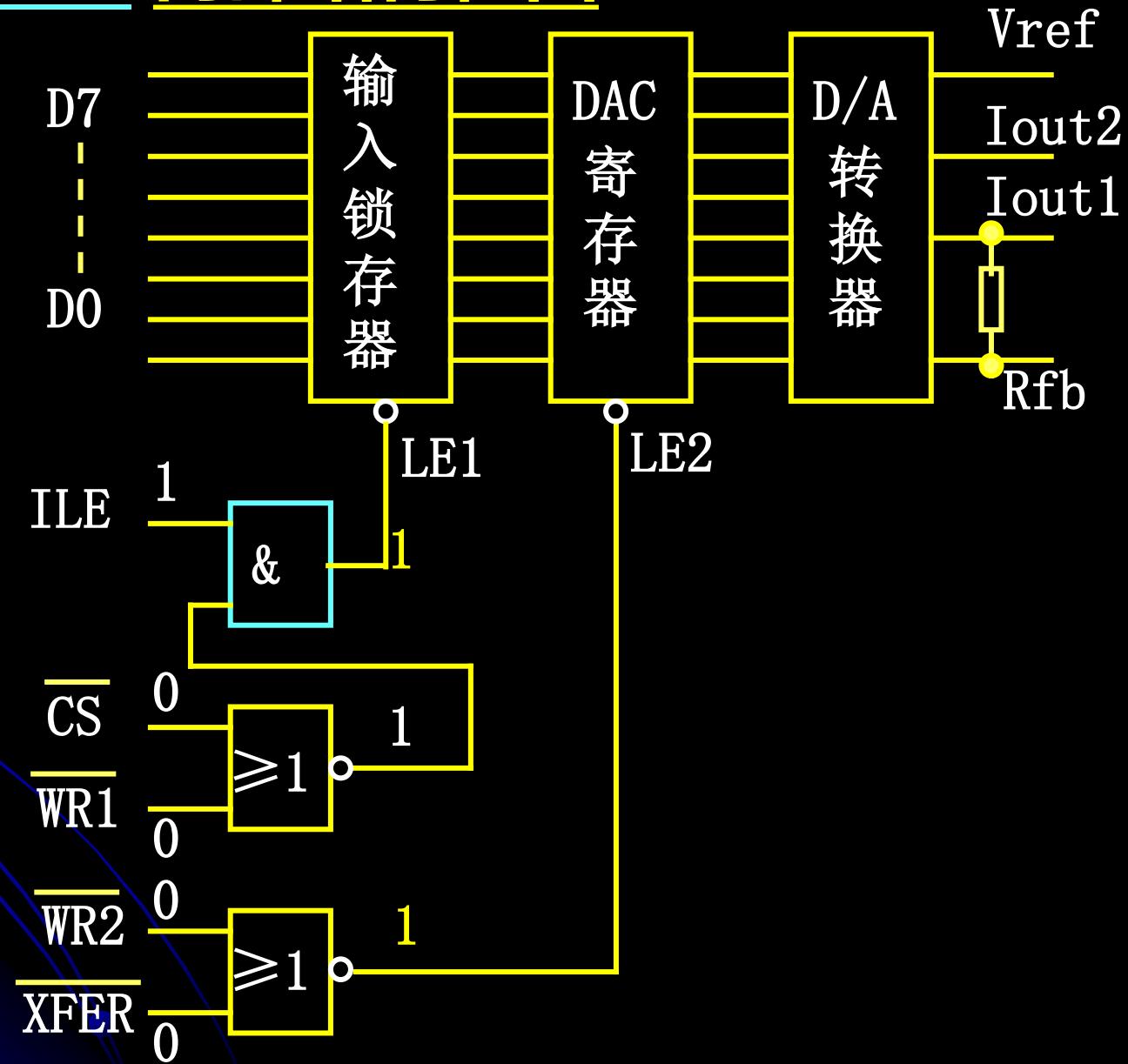
**Vcc:** +5~+15V单电源供电端

DAC0832



20 PIN DIP封装

# DAC0832 内部结构框图





# DAC0832 内部结构框图

**/CS:** 输入选通, 低有效

**ILE:** 数据锁存允许, 高有效

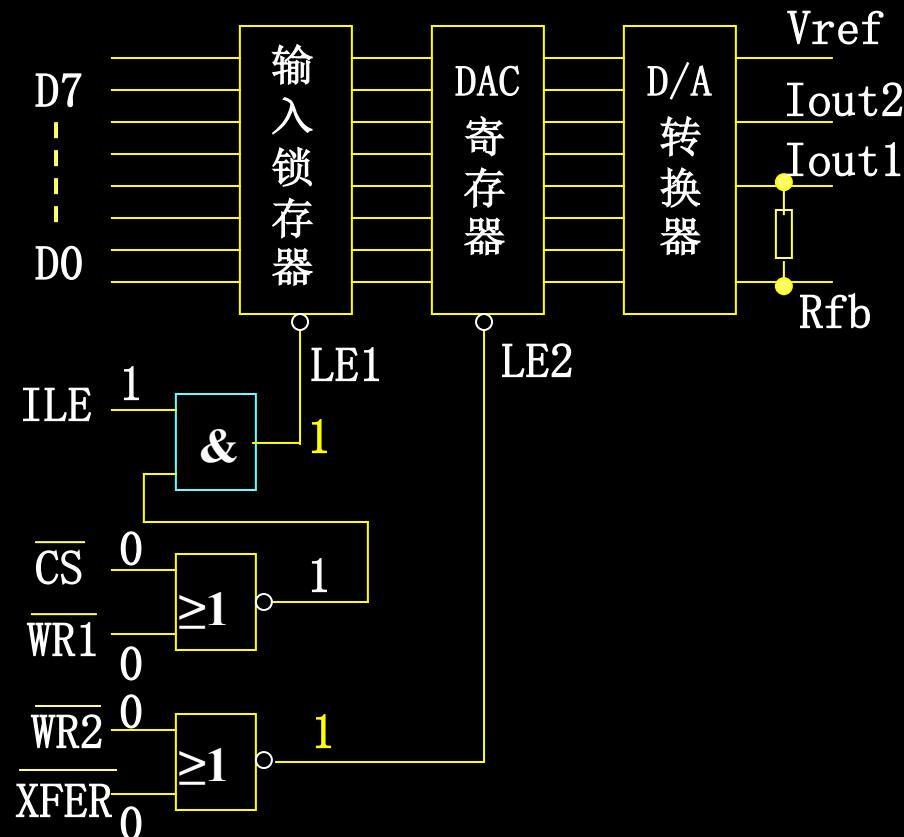
**/WR1:** 写控制信号1, 低有效

**/WR2:** 写控制信号2, 低有效

**/XFER:** 数据传送控制信号

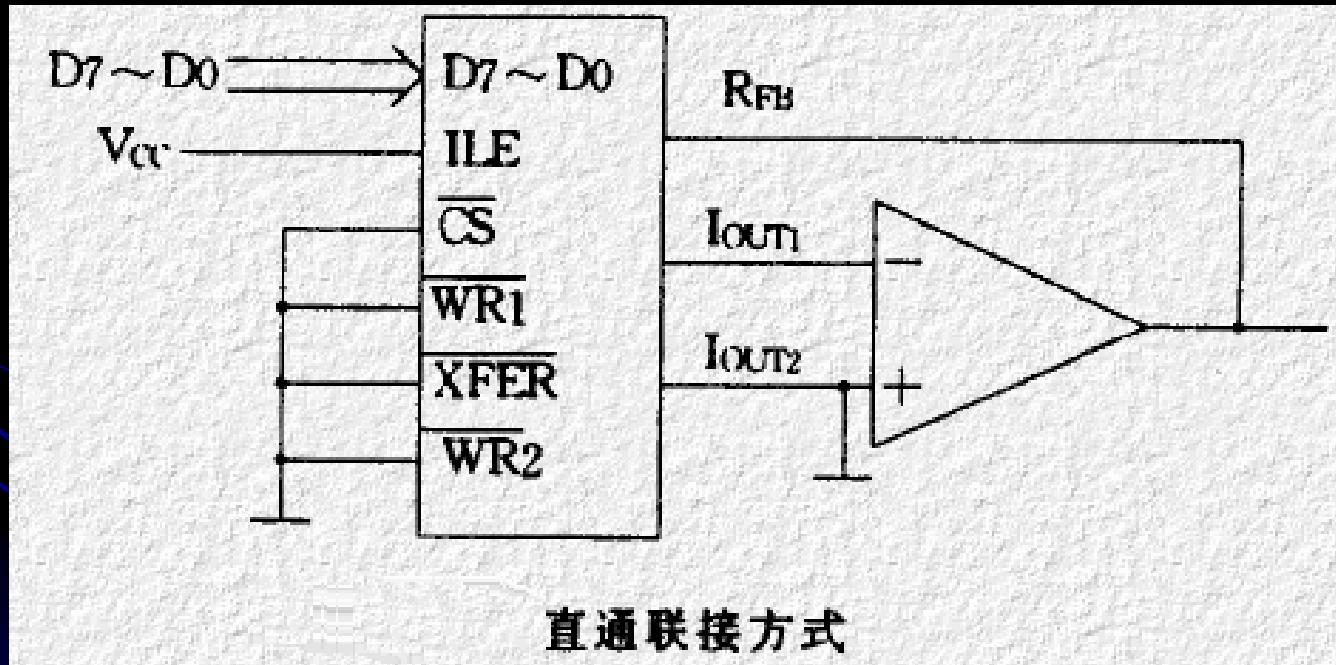
当/CS和/WR1同时是低电平，而ILE是高电平时，输入锁存器接收待转换的输入数据。

/WR2 与 /XFER配合使用，使输入到锁存器的8位数据传送到DAC寄存器中。



# DAC0832常见的三种工作方式

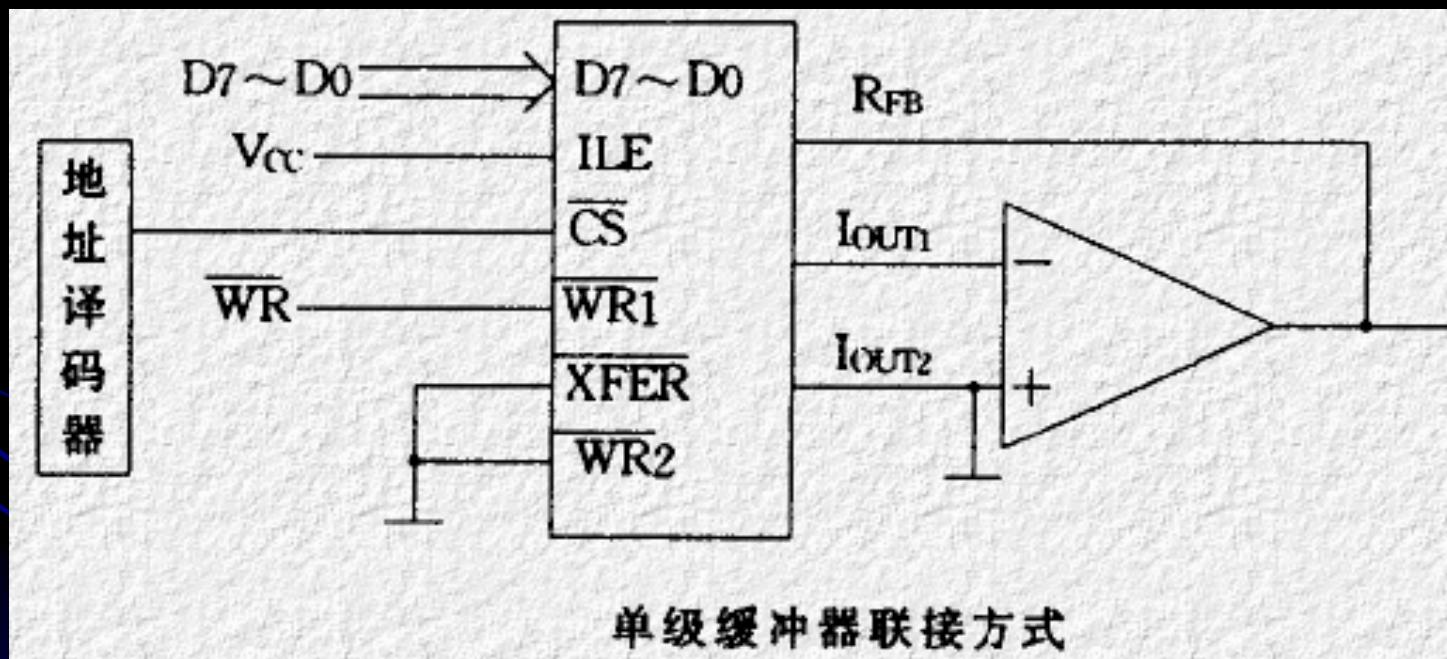
**直通方式**：使输入锁存器和DAC寄存器始终处于选通状态（无地址），数字量的变化会直接反映到模拟量上。



ILE为高电平，/CS、/WR1、/WR2和/XFER等信号均为低电平

# DAC0832常见的三种工作方式

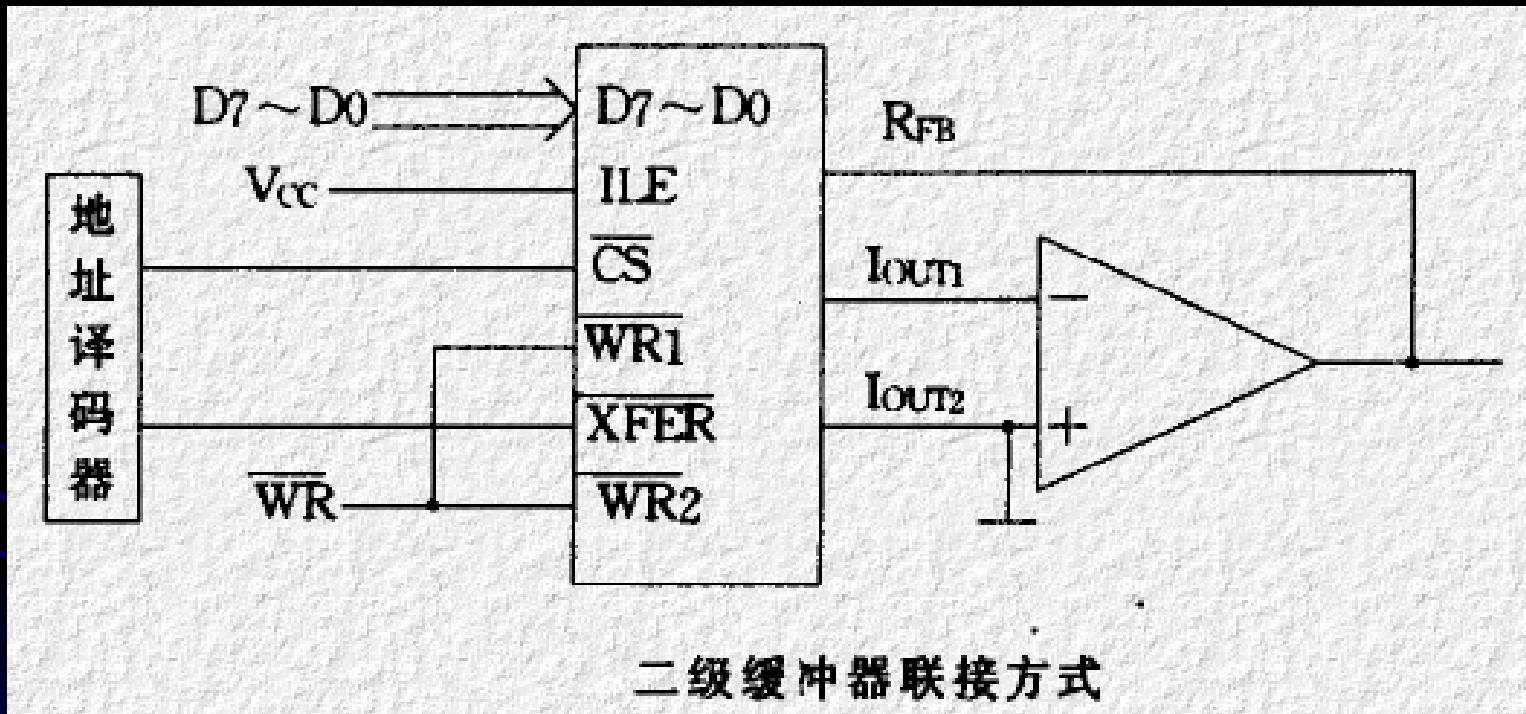
**单缓冲方式：**两个寄存器同时选通及锁存，或只用输入锁存器，把DAC寄存器接成直通方式(共用一个地址)



用/CS和/WR1这两个控制信号对输入锁存器进行选通，/WR2和/XFER接地，使DAC寄存器处于直通状态。

# DAC0832常见的三种工作方式

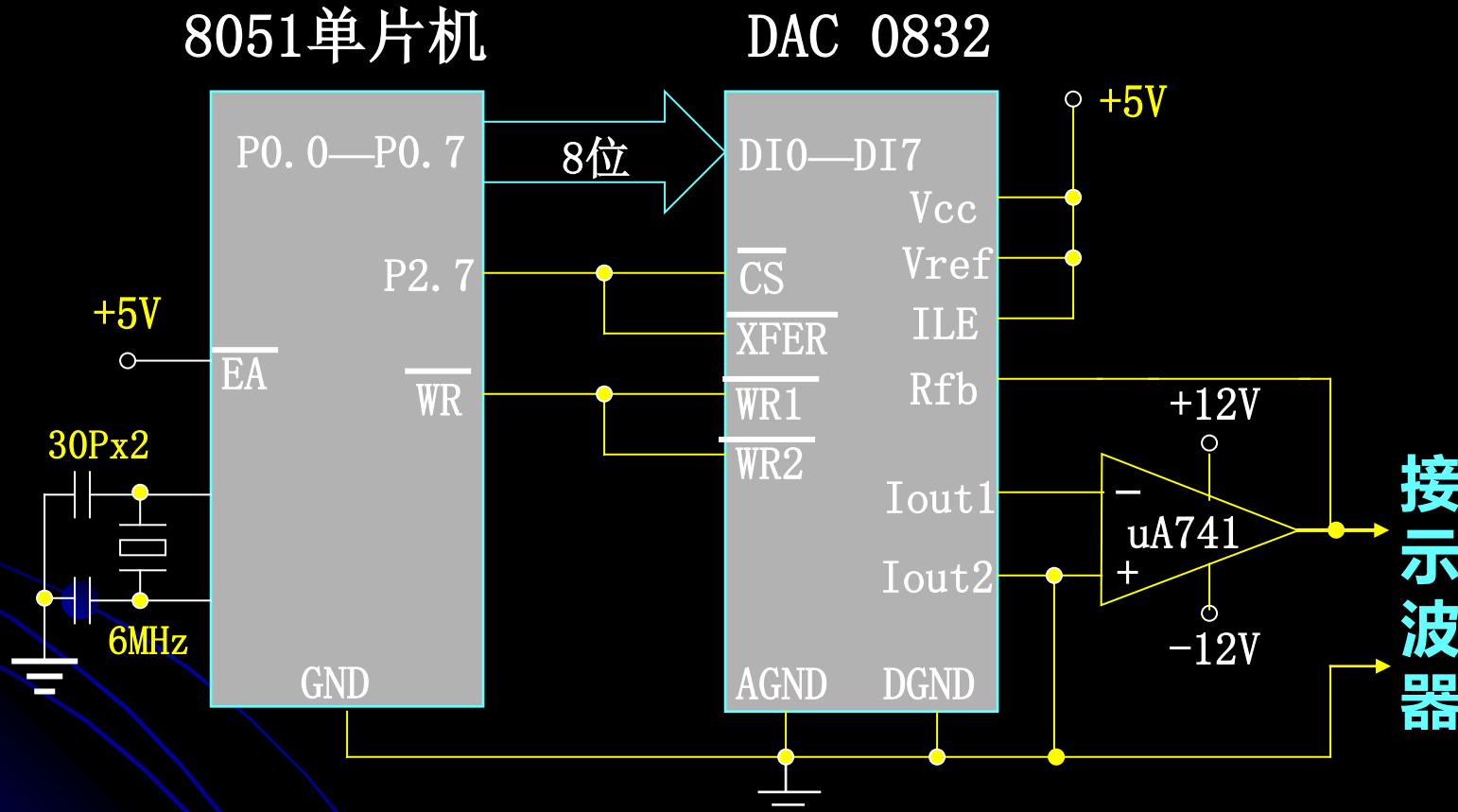
**双缓冲方式：输入锁存器和DAC寄存器先后选通  
(各有一个地址)**



地址译码器一个时刻只能输出一个地址，可以先使/CS有效，选通输入锁存器，再使/XFER有效，选通DAC寄存器。



# DAC0832 与单片机的连接-单缓冲举例



对应 单缓冲方式 是两个寄存器同时选通及锁存的方式



# DAC0832 的编程应用举例

## 例1 产生矩形波

```
LL:MOV A, #00H ;低电平
    MOV DPTR, #7FFFH
    MOVX @DPTR, A ;送转换
    LCALL DELAYL ;低宽度
    MOV A, #0FFH ;高电平
    MOVX @DPTR, A ;送转换
    LCALL DELAYH ;高宽度
    SJMP LL
```



(1) 程序产生矩形波，其低电平的宽度由延时子程序DELAYL所延时的时间决定，高电平的宽度则由DELAYH所延时的时间决定。

(2) 改变延时子程序DELAYL和DELAYH延时时间，就可改变矩形波上下沿的宽度。若 $DELAYL=DELAYH$ ，则输出的是方波。

(3) 改变上限值或下限值便可改变矩形波的幅值。



# DAC0832 的编程应用举例

## 例2 产生锯齿波

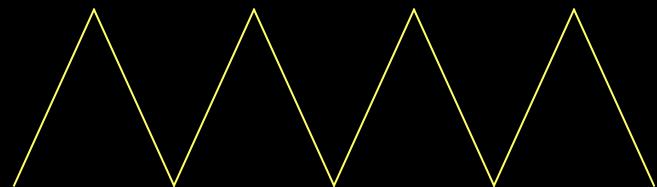
```
MOV A, #00H ;起始值
    MOV DPTR, #7FFFH
MM: MOVX @DPTR, A ;送转换
    INC A
    NOP
    NOP
    NOP ;决定坡度
    SJMP MM
```



- (1) 程序每循环一次，A加1，锯齿波的上升边是由256个小阶梯构成，由于阶梯很小，可以看作是线性增长锯齿波。
- (2) 若要改变锯齿波的周期，可在SJMP MM指令前加入延迟子程序即可。延时较短时可用NOP指令实现。延时时间不同，波形周期不同，锯齿波的斜率就不同。
- (3) 通过A加1，得到正向的锯齿波，反之A减1可得到负向的锯齿波。

## ● DAC0832编程应用举例：例3 产生三角波

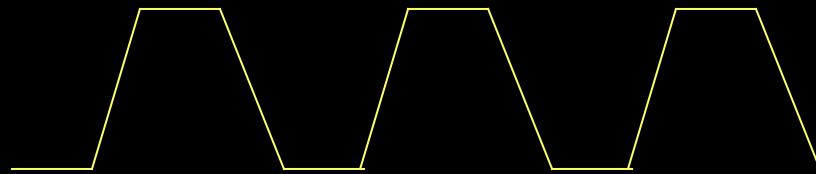
```
MOV A, #00H
MOV DPTR, #7FFFH
SS1: MOVX @DPTR, A ;送转换
NOP
NOP
NOP
SS2: INC A      ;等速上升
CJNE A, #0FFH, SS1
SS3: DEC A
MOVX @DPTR, A
NOP
NOP
NOP      ;等速下降
CJNE A, #00H, SS3
SJMP SS1
```



三角波

- (1) 程序产生三角波，谷值为0，峰值为+5V (或-5V)。若改变下限值或上限值，三角波的谷值和峰值也随之改变。
- (2) 改变延时时间可改变三角波的斜率。

上述三角波发生程序若在谷值和峰值处延时时间较长时，则输出梯形波。延时时间的长短决定梯形波上下边的宽度。

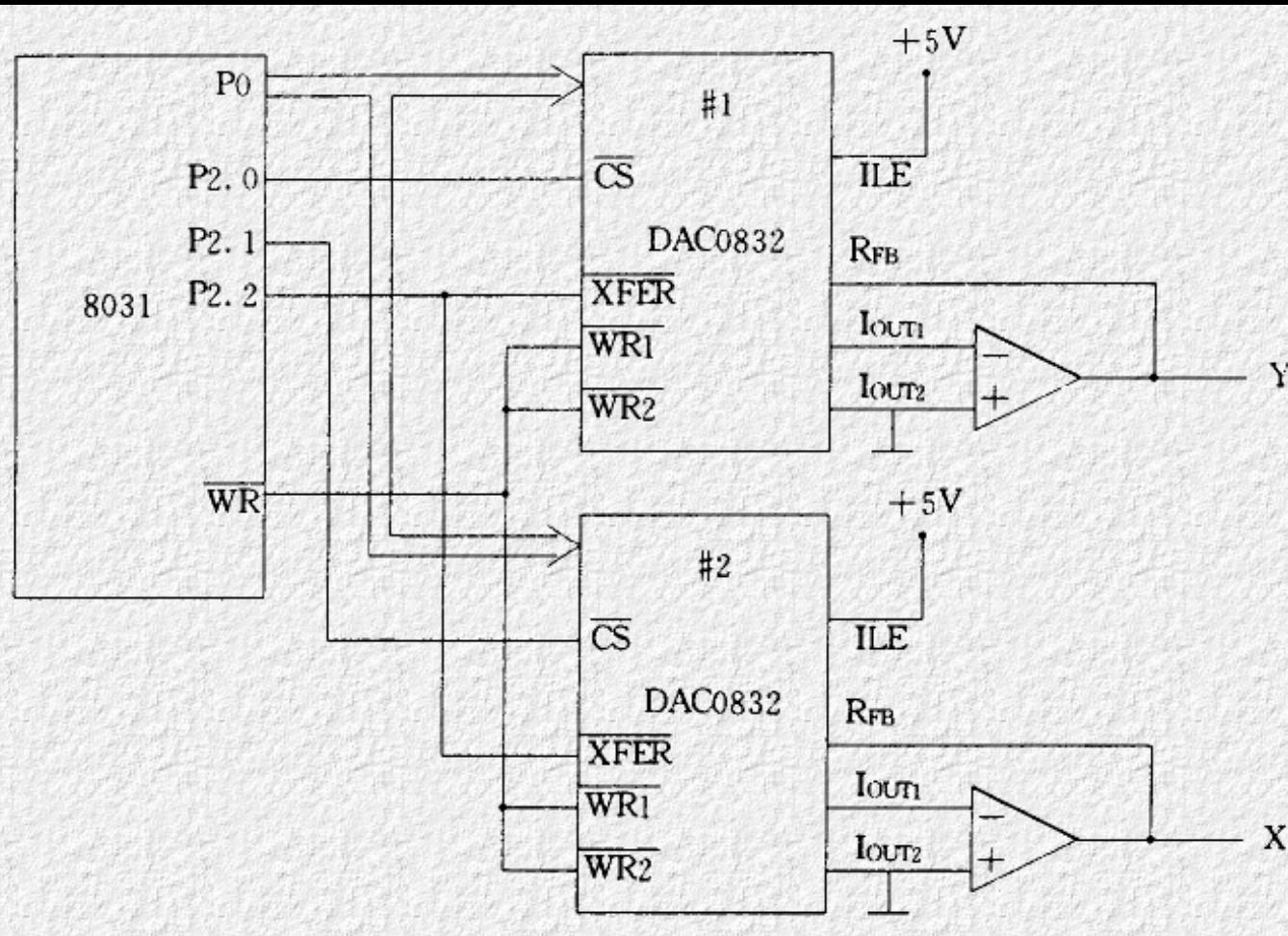


梯形波

如何编程，请自己思考。



# DAC0832 与单片机的连接-双缓冲举例

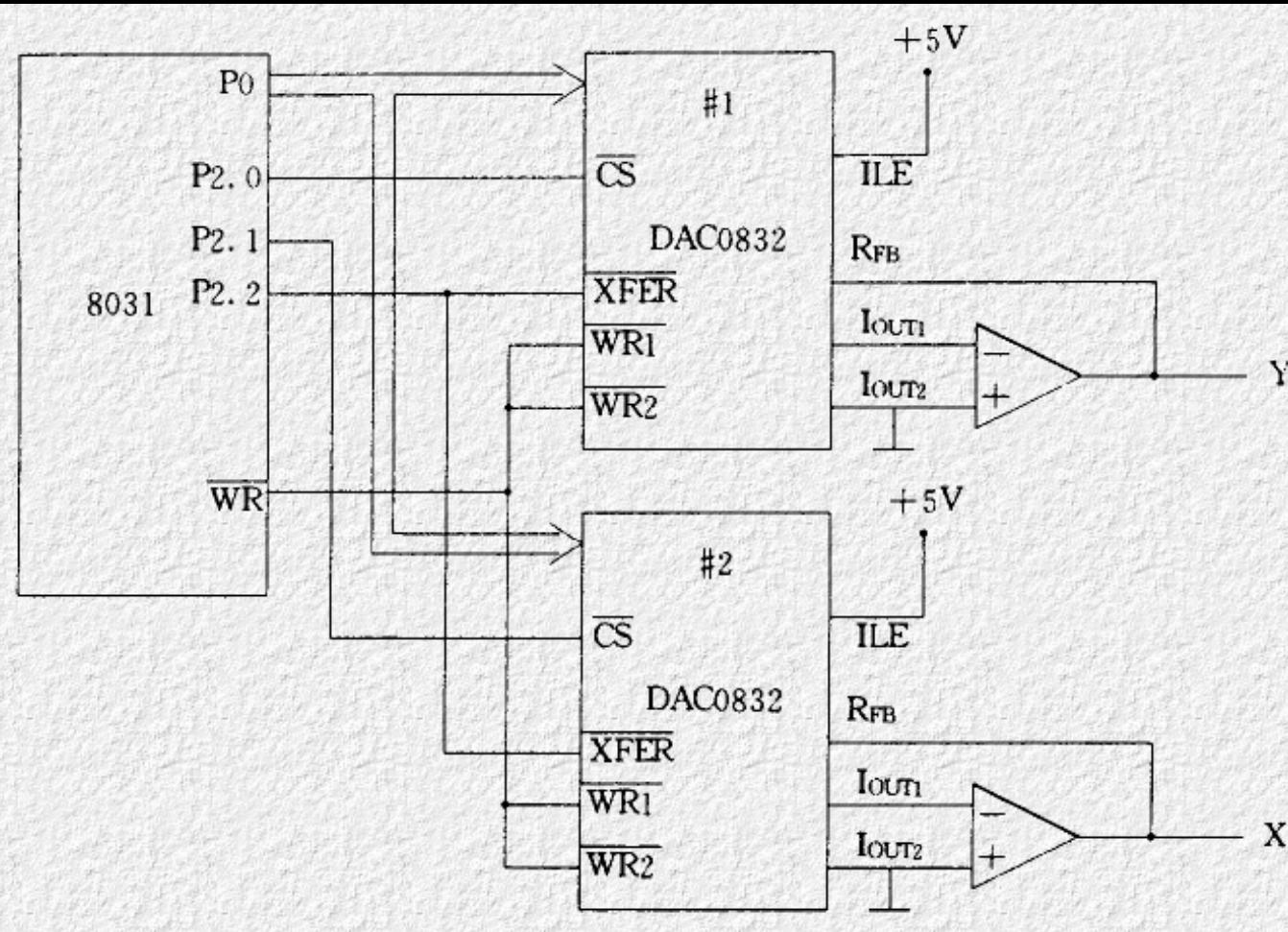


DAC0832采用双缓冲方式时，数字量的输入锁存和D/A转换分两步进行。

两片DAC0832与8031双缓冲连接，可实现两路同步输出。

/CS分别接地址P2.0和P2.1，两片0832的输入寄存器具有不同的地址；/XFER均接P2.2，两片0832的DAC寄存器具有相同的地址。

# ● DAC0832 与单片机的连接-双缓冲举例



CPU分时向各路D/A转换器输入要转换的数字量，并锁存在各自的输入寄存器中。

CPU对所有D/A转换器发出控制信号，使各路输入寄存器中的数据进入DAC寄存器，实现同步转换输出。

1#DAC0832输入寄存器: FEFFH, DAC寄存器: FBFFH  
2#DAC0832输入寄存器: FDFFH, DAC寄存器: FBFFH

# 在两个DAC通道上同时输出不同电压的程序

```
MOV DPTR, #0FEFFH ;置 #1 输入寄存器地址
MOV A, #80H ;A←数字量
MOV @DPTR, A ;数字量送 #1 输入寄存器
MOV DPTR, #0FDFFH ;置 #2 输入寄存器地址
MOV A, #00H
MOV @DPTR, A
MOV DPTR, #0FBFFH ;置 #1、#2DAC 寄存器地址
MOV @DPTR, A ;数据送 DAC 寄存器
```

两路数据同步转换输出

# A/D转换器

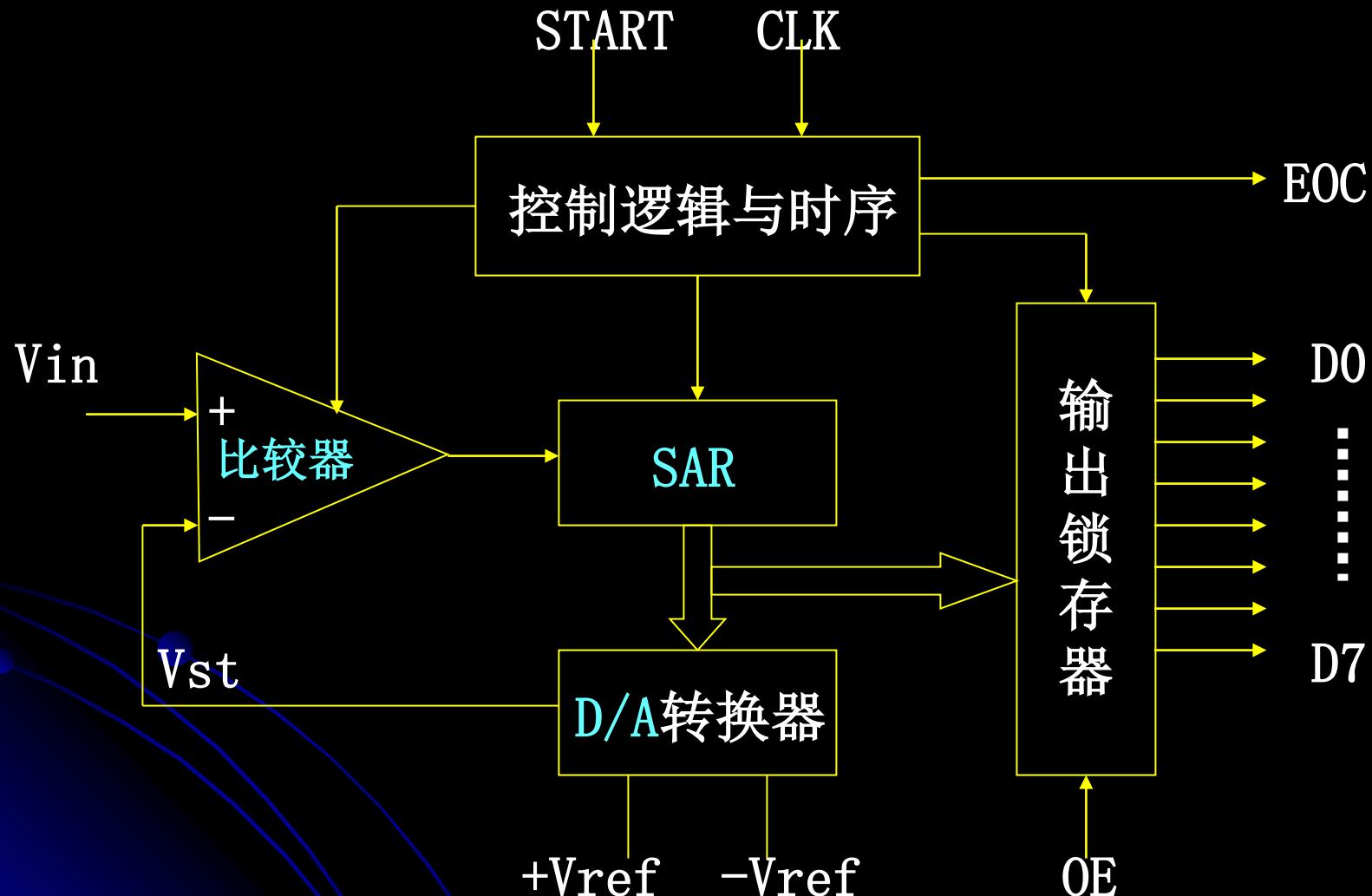
**采样保持电路**:把一个时间连续的信号变换为时间离散的信号，并将采样信号保持一段时间。

**量化编码电路**是A/D转换器的核心组成部分，根据转换形式不同可分为——

**直接A/D转换器**: 将输入的模拟电压直接转换为输出的数字代码(无中间变量)。如**逐次逼近型**

**间接A/D转换器**: 首先将输入的模拟信号转换成与之成正比的时间或频率，然后通过计数的方式转换成数字量输出。如**双积分型**（电压/时间变换型）和**计数型**（电压/频率变换型）

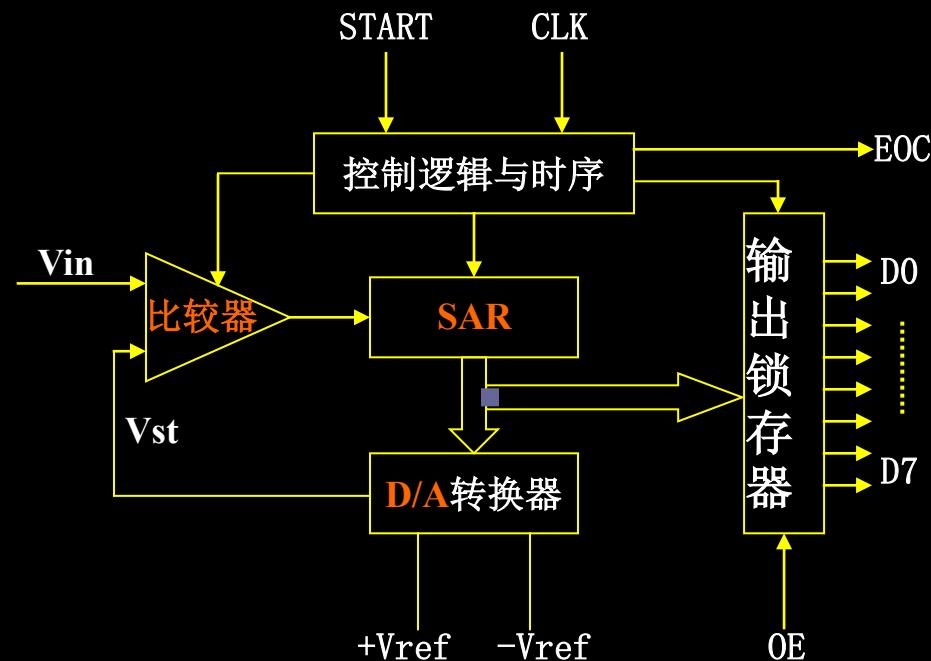
# 逐次逼近式A/D转换器结构



由逐次逼近寄存器SAR、D/A转换器、电压比较器、输出锁存器、控制逻辑与时序电路构成。

# 逐次逼近式A/D转换器工作原理

D/A转换器的输出，从二进制数据的最高位起，依次逐位置1，与待转换的模拟量比较，若前者小于后者，该位置1并保留下，若前者大于后者，该位清0；然后再照此比较下一位,.....直至比完最低位。最后得到的结果即A/D转换的值。



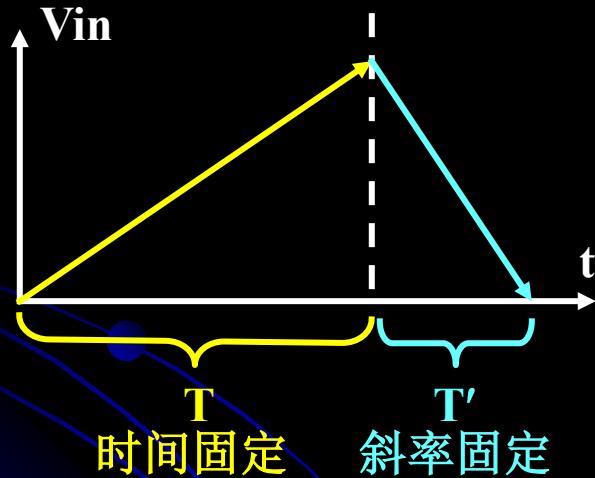
## ● 逐次逼近式A/D转换器特点

- ✓ 转换速度较快 (比较次数等于A/D的位数) 通常在几 $\mu$ S至几百 $\mu$ S数量级。
- ✓ 被转换的模拟量若频率很高 (变化较快) 则要加采样/保持 (S/H) 电路。
- ✓ 被转换的模拟量若幅度过小 (信号微弱) 则需要加信号调理电路。



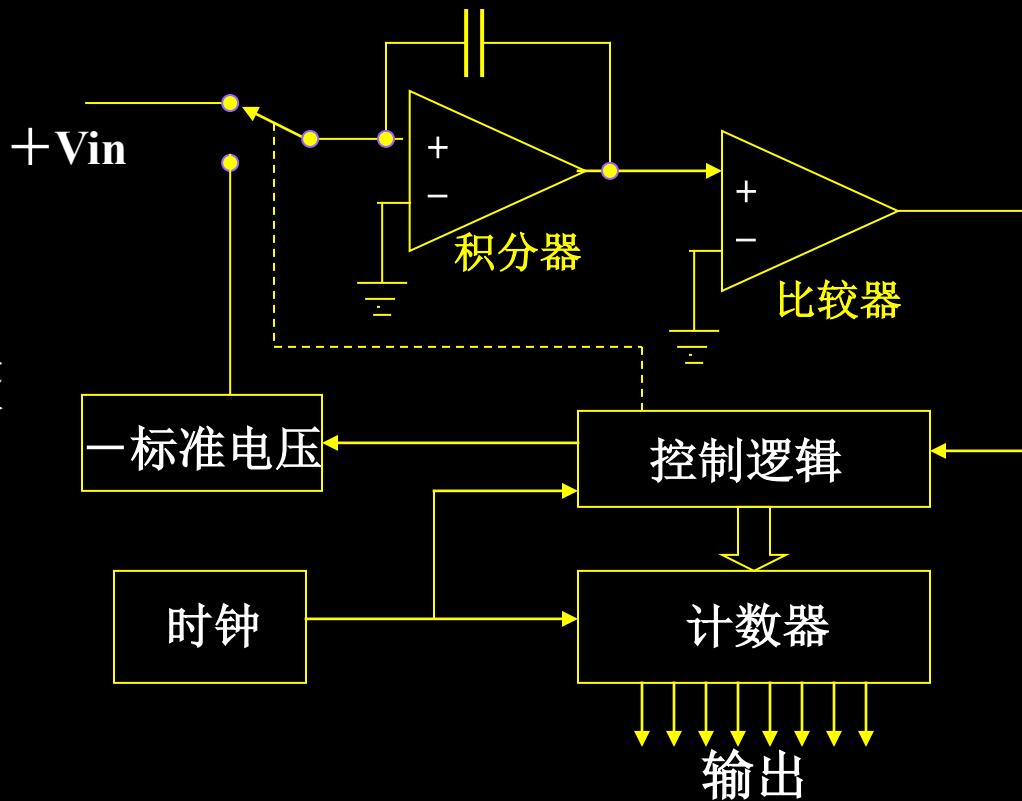
## 双积分式A/D转换器结构与工作原理

电路对未知输入电压先进行固定时间T的积分(充电),然后对已知标准电压进行反向积分(放电),直至放电为0, 放电所花时间T'与输入模拟电压成正比。



例如: ICL7107

反向积分时计数器开始计数, 计数值即为A/D转换后的数字量。





## 双积分式A/D转换器特点

- ✓ 转换速度较慢 (因为A/D转换的过程要两次积分)  
通常在几mS至几百mS数量级。
- ✓ 适用于转换变化较缓慢的模拟量 (频率较低) ,  
输入端一般不需要再加采样/保持电路。
- ✓ 抗干扰性强 (信号波动对精度影响不大) 。
- ✓ 性价比高, 与逐次逼近型相比, 同样价格条件下  
转换精度较高, 常用于数字多用表。

# ● 8位CMOS模/数转换芯片ADC 0809

✓ 8位A/D,量化间隔=Vin/(256-1)

✓ CMOS低功耗器件

✓ 8通道多路开关输入切换电路

✓ 单电源供电+5V, Vref=+5V

Vin范围:单极性0 ~ +5V

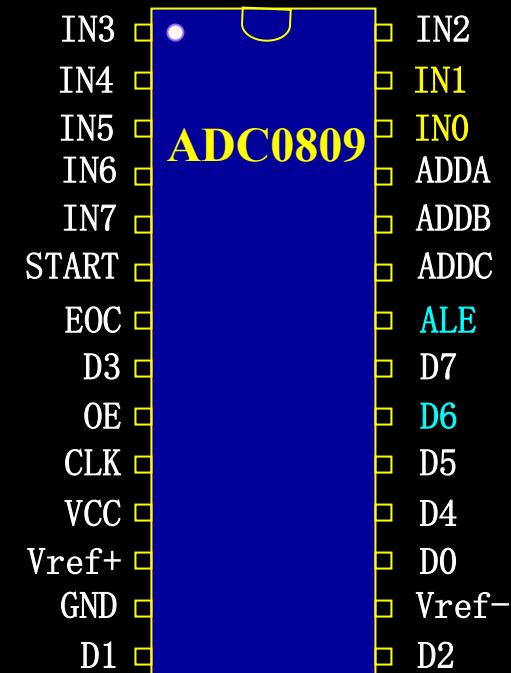
✓ 逐次逼近结构

✓ 每次转换时间:  $\approx 100\mu\text{s}$

60个时钟周期,  $f_{\text{max}}=640\text{KHz}$   
(推荐 $\text{CLK}=500\text{KHz}$ )

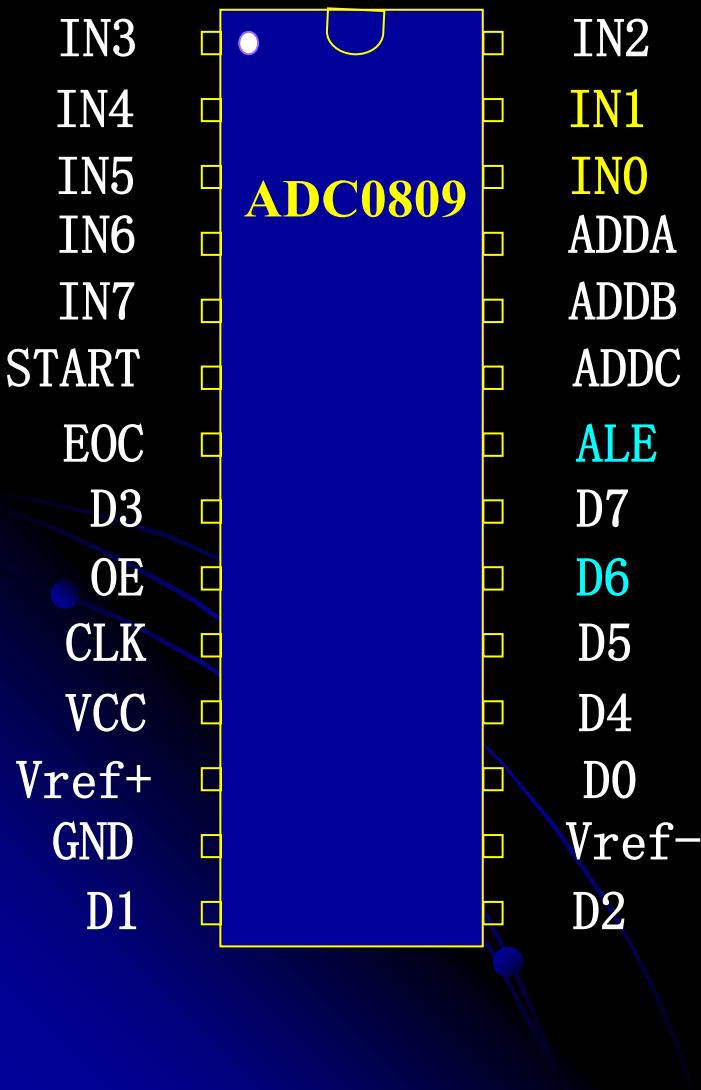
✓ 转换结果读取方式:

① 延时读数 ② 查询EOC=1 ③ EOC申请中断



# ADC 0809

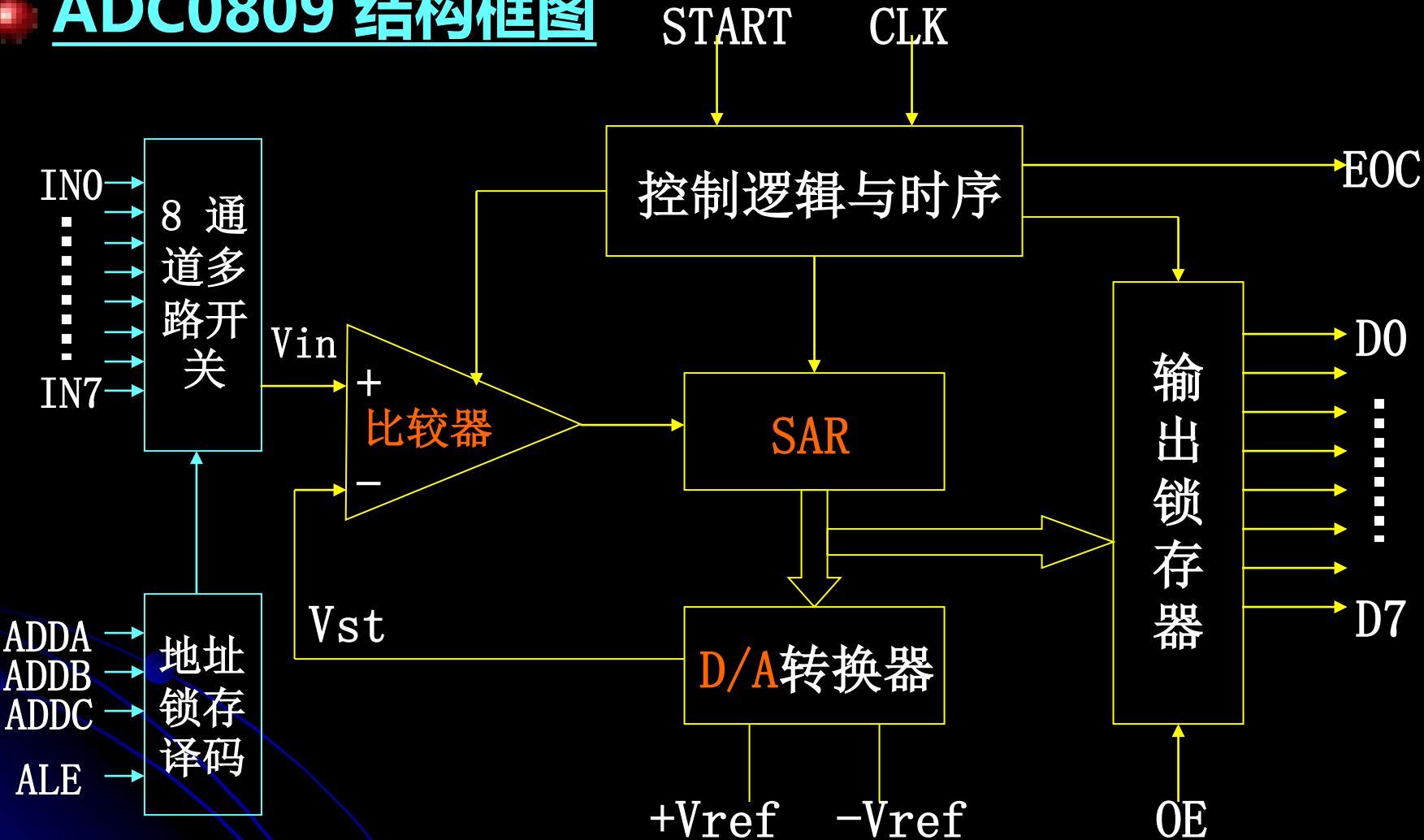
## 引脚定义：



**IN0-IN7:** 8通道模拟量输入端  
**D0-D7:** 8位数字量输出端  
**ADDA、ADDB、ADDC:** 8路地址选通输入端, 接地址锁存器的低三位地址  
**ALE:** 地址锁存允许控制信号  
**START:** 启动转换  
**OE:** 允许读A/D结果, 高有效  
**CLK:** 时钟输入端, 应 $\leq 640\text{KHz}$   
**EOC:** 转换结束时为高  
**Vcc:** +5V  
**Vref+:** 参考电压, +5V  
**Vref-:** 0V



# ADC0809 结构框图



在之前的逐次逼近型A/D转换器基础上添加了8通道多路开关和地址锁存译码器。

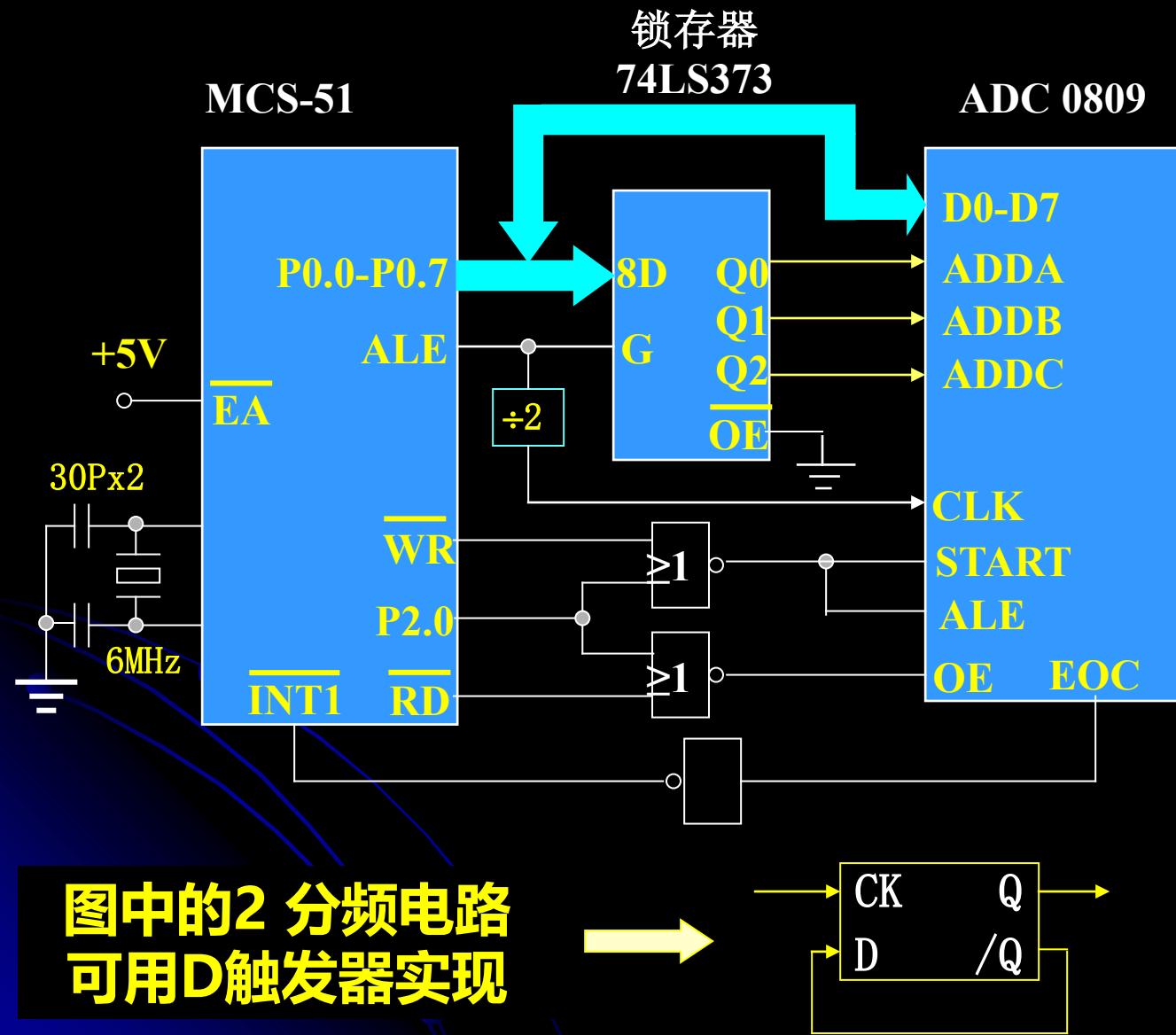
# ADC0809的转换过程

CLK	给SAR置位	SAR试探值	$U_O$ (V)	$U_I$ (V)	$U_I$ 与 $U_O$ 比较	SAR逼近值
1	$D_7=1$	10000000	2.56	4.34	$U_I > U_O$	10000000
2	$D_6=1$	11000000	3.84	4.34	$U_I > U_O$	11000000
3	$D_5=1$	11100000	4.48	4.34	$U_I < U_O$	11000000
4	$D_4=1$	11010000	4.16	4.34	$U_I > U_O$	11010000
5	$D_3=1$	11011000	4.32	4.34	$U_I > U_O$	11011000
6	$D_2=1$	11011100	4.40	4.34	$U_I < U_O$	11011000
7	$D_1=1$	11011010	4.36	4.34	$U_I < U_O$	11011000
8	$D_0=1$	11011001	4.34	4.34	$U_I = U_O$	11011001

$U_I > U_O$  保留该位;  $U_I < U_O$  丢弃该位; .....  
一直到  $U_I = U_O$  即为转换结果。



# ADC0809 与单片机的连接

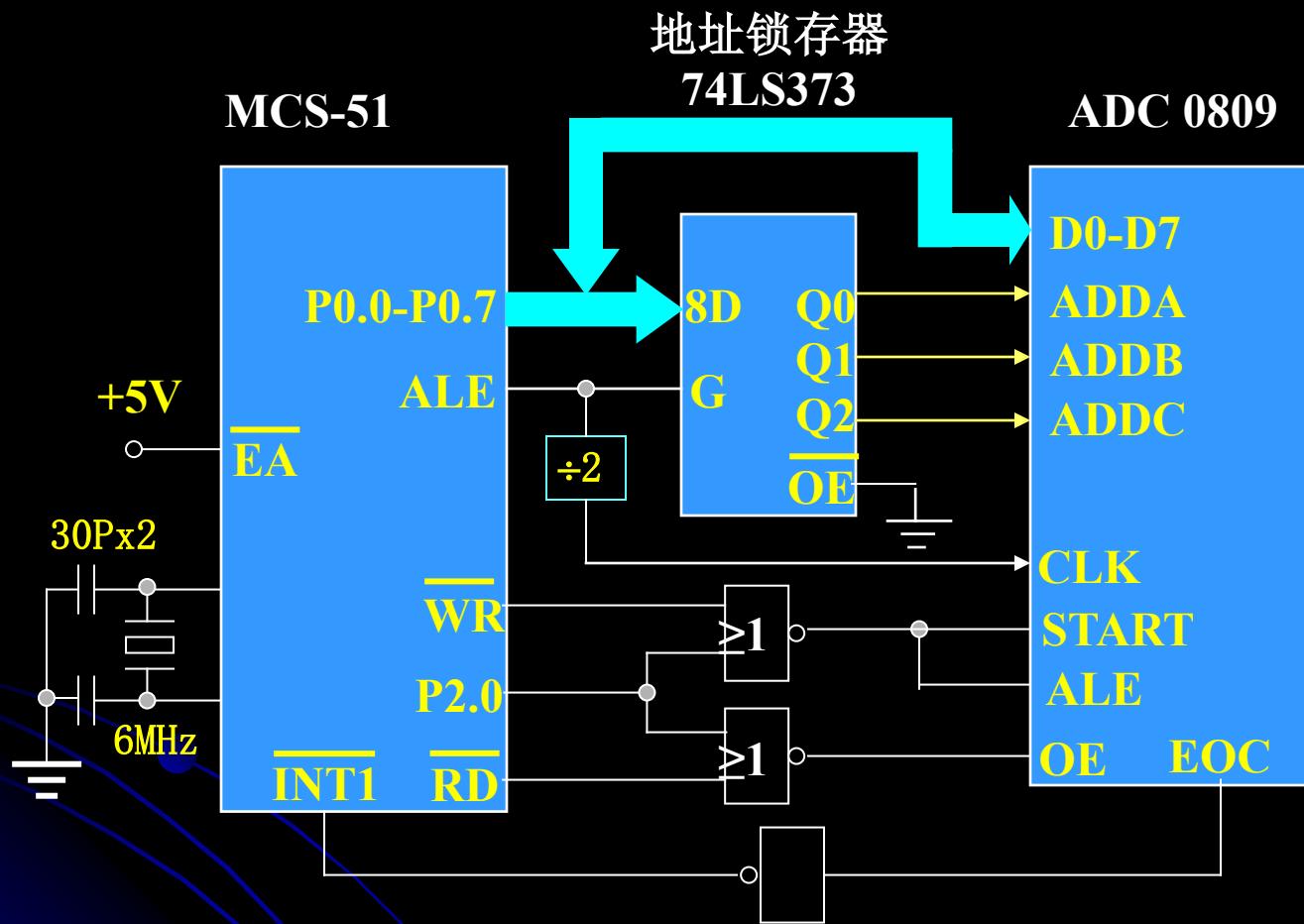


ADC0809与单片机的接口方式有查询方式、中断方式和等待延时方式。

ADC0809内部没有时钟脉冲源，可利用单片机提供的地址锁存控制输入信号ALE经D触发器2分频之后，作为ADC0809的时钟输入。



# ADC0809 与单片机的连接



口地址  
可以为：  
**0FE00**  
~  
**0FE07H**  
也可为：  
**0FEF8**  
~  
**0FEFFH**

P0口是数据/地址分时复用，要有地址锁存器74LS373。

ADC0809的ALE和START连在一起，可以在锁存通道地址的同时启动A/D转换。

P2.0分别和读写控制信号一起来控制对ADC0809的启动和读取A/D转换的结果。

EOC转换结束信号输出端经非门接INT1，可用中断方式进行工作。



## ADC0809延时方式进行数据采集

```
MOV  DPTR, #0FEF8H ; ADC 0口地址
MOVX @DPTR, A      ; 启动A/D转换
LCALL DELAY        ; 等待转换结束
MOVX A, @DPTR      ; 取转换结果
```

# ADC0809八路巡回中断式数据采集

```
ORG 0000H
AJMP MAIN
ORG 0013H ;外部中断1的中断矢量
AJMP INT
MAIN: MOV R0,#0A0H ;存结果的缓冲区·A0H-A7H
      MOV R2,#08H ;待采集的数据量
      SETB IT1 ;选择下降沿触发
      SETB EA ;开中断
      SETB EX1 ;开外部中断1
      MOV DPTR,#0FEF8H ;通道0的地址
      MOVX @DPTR, A ;启动转换。注意:A=??
HERE: SJMP HERE ;等待中断
```

由于要选通某一通道，  
A的内容必须和所选输入通道号一致，可以  
是00H~07H

# ● ADC0809八路巡回中断式数据采集

```
ORG 0000H
AJMP MAIN
ORG 0013H
AJMP INT
MAIN: MOV R0,#0A0H
      MOV R2,#08H
      SETB IT1
      SETB EA
      SETB EX1
      MOV
      DPTR,#0FEF8H
      MOVX @DPTR, A
HERE: SJMP HERE
```

INT:MOVX A, @DPTR ;读数据  
MOV @R0, A  
;数据放进缓存单元  
INC R0 ;指向下一缓存  
INC DPTR ;指向下一通道  
DJNZ R2, RTN  
;8 次未完就继续采集,  
;已完就关中断、停采集  
CLR EA  
CLR EX1  
RETI  
RTN:MOVX @DPTR,A;启动采集  
RETI

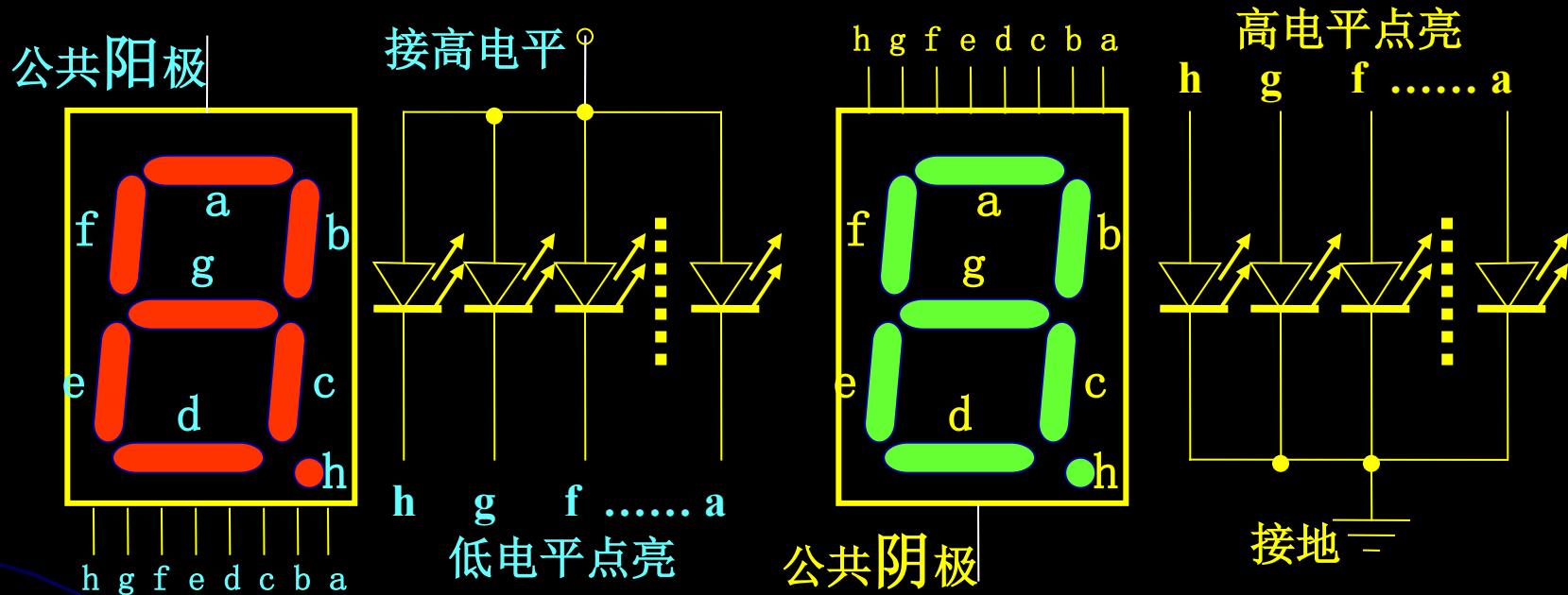
# 本章小结

- 理解D/A转换的基本工作原理
- 掌握DAC0832与MCS-51的接口、三种工作方式及编程应用
- 理解A/D转换的基本工作原理
- 掌握ADC0809与MCS-51的接口及数据采集方式

# 第11章 显示器、键盘、打印机接口

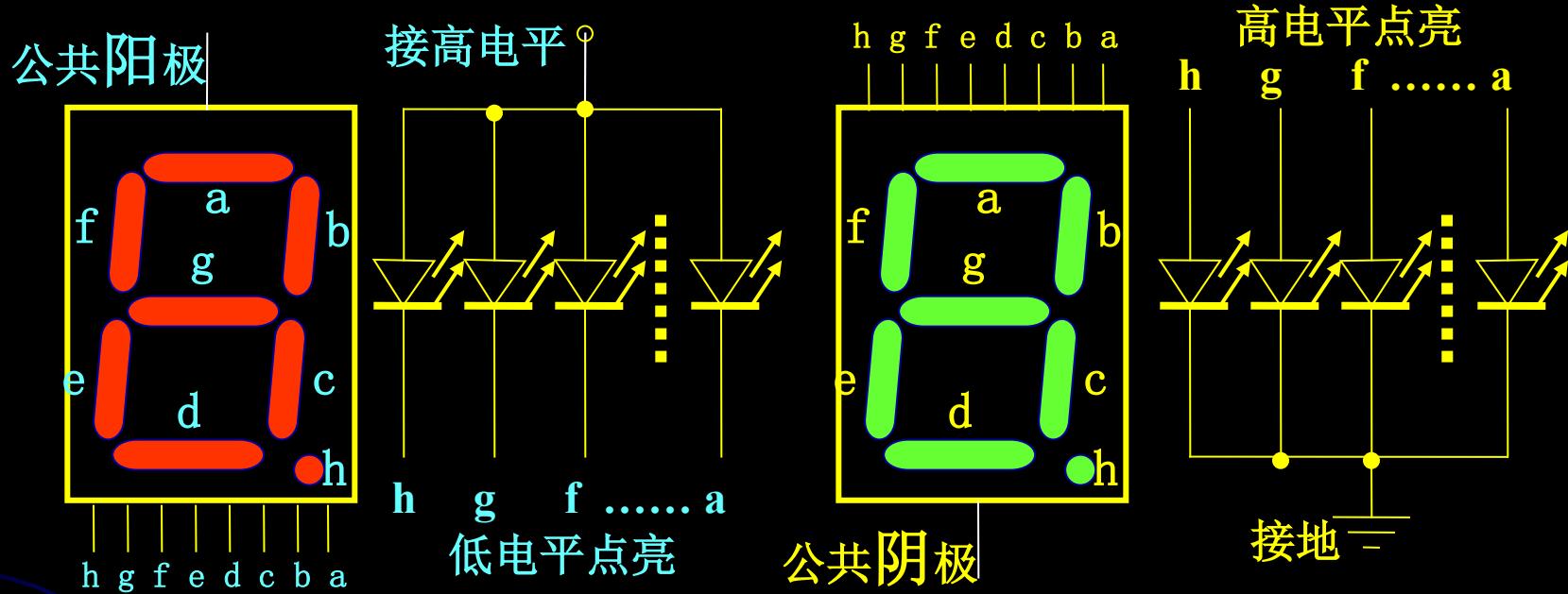
- ◆ LED显示器接口电路
- ◆ 键盘接口电路
- ◆ 打印机接口电路

# 一、LED显示器接口电路



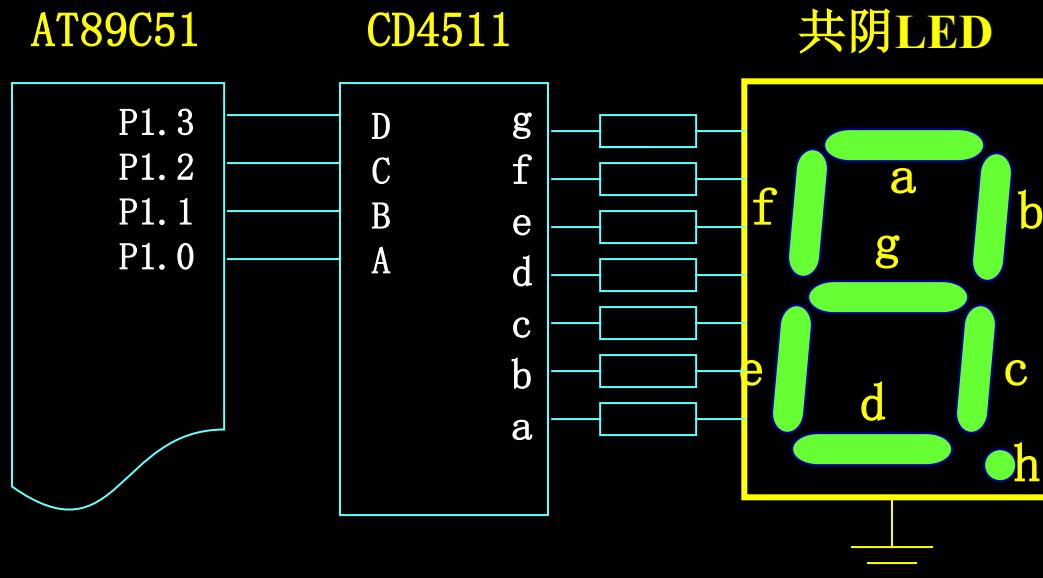
LED是发光二极管，它是一种能把电能转换为光能的发光器件。LED显示器（也称为LED数码管）是由七段条形发光二极管组成，各段依次记为a、b、c、d、e、f、g，加上小数点h，就构成了八段LED显示器。

# LED数码管的结构：共阳极与共阴极



**共阳数码管**的所有二极管的阳极都并接在一起接高电平，各段笔画用低电平(“0”)点亮,要求驱动功率很小；  
**共阴数码管**的所有二极管的阴极都并接在一起接低电平，各段笔画用高电平(“1”)点亮，要求驱动功率较大。  
通常每个段笔画要串一个数百欧姆的**限流电阻**。

# LED数码管的译码：硬件译码与软件译码



- **硬件译码特点:**采用专用的译码/驱动器件,驱动功率较大;增加了硬件的开销;软件编程简单;字型固定。

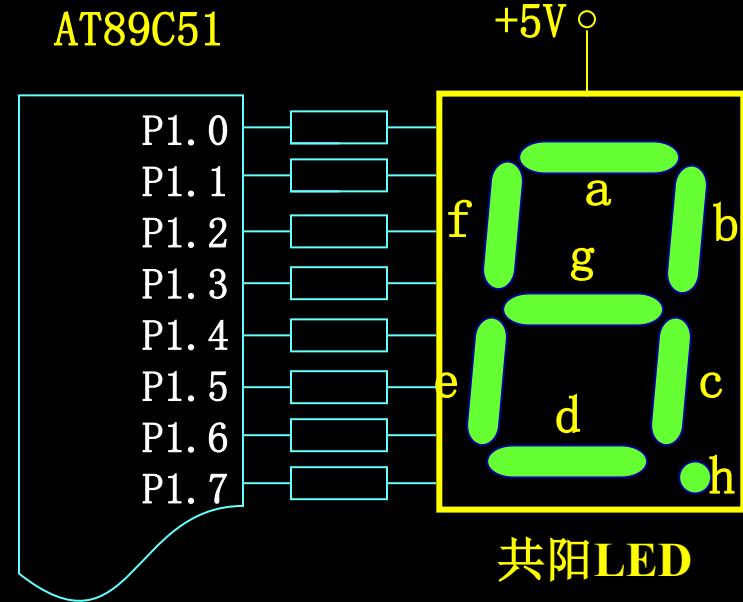
- **74LS48/CD4511**是“BCD码→七段**共阴**译码/驱动”IC; **74LS47**是“BCD码→七段**共阳**译码/驱动”IC。

## LED数码管的译码：软件译码

### 硬件译码特点：

采用专用译码/驱动器件，驱动功率较大；增加了硬件的开销；软件编程简单；字型固定。

AT89C51

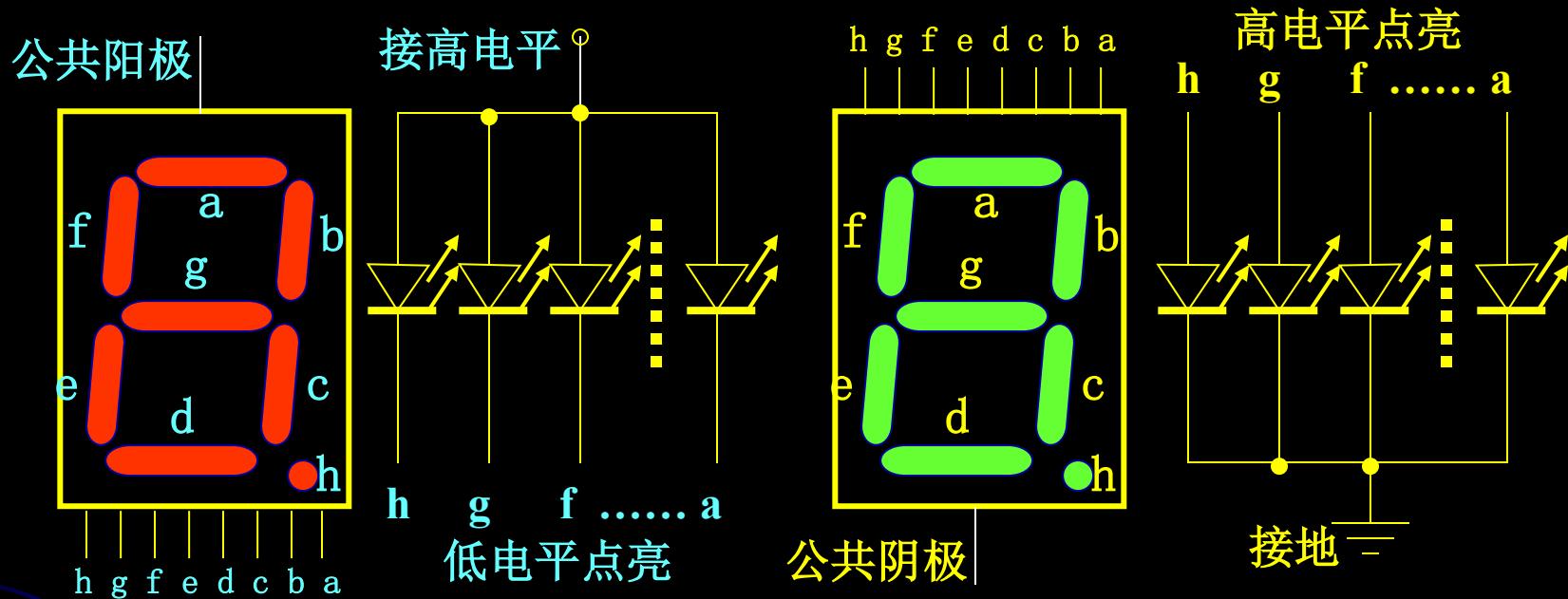


### 软件译码特点：

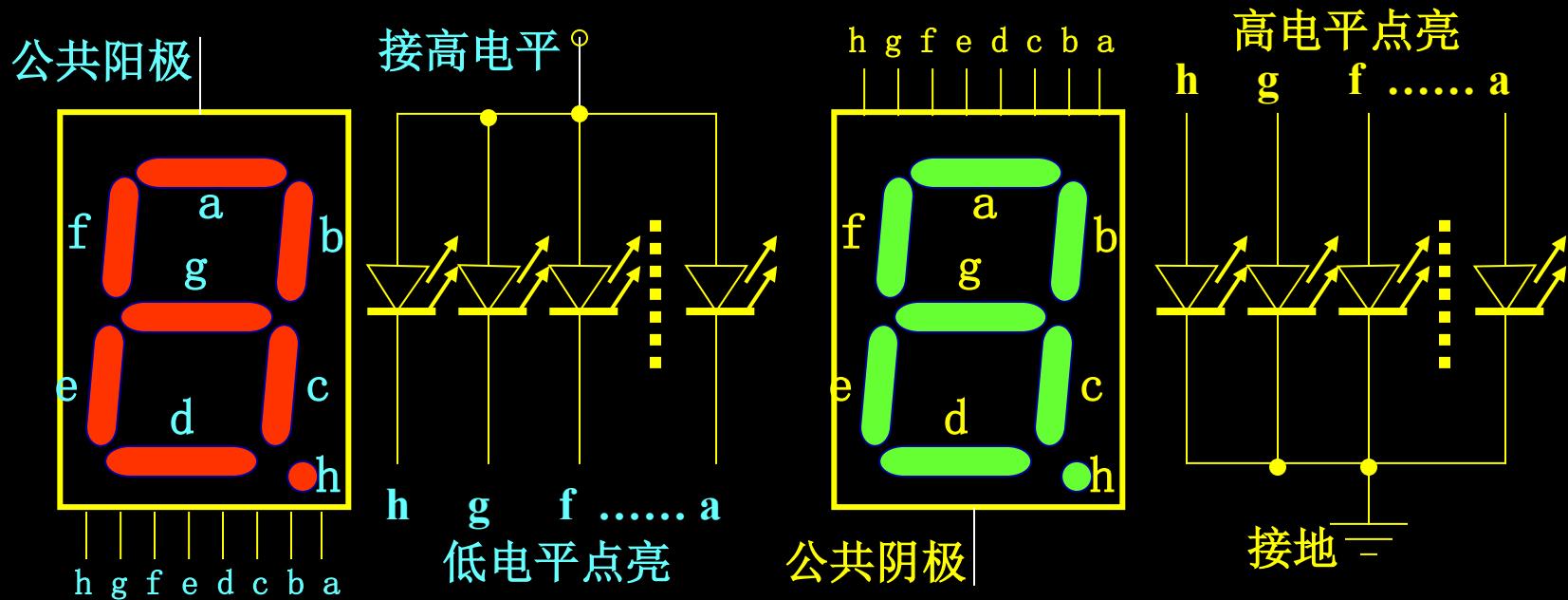
不用专用的译码/驱动器件，驱动功率较小；不增加硬件的开销；软件编程较复杂；字型灵活。



# LED数码管的软件译码



软件译码，就是通过对加到各段上的代码进行编码，从而控制显示器显示不同的数字和字符，这个代码也称为段码。



## 八段LED数码管段代码编码表(连线不同可有多种表):

字形	0	1	2	3	4	5	6	7	8	9	黑
共阳	0C0	0F9	0A4	0B0	99	92	82	0F8	80	90	0FF
共阴	3F	06	5B	4F	66	6D	7D	07	7F	6F	00

共阳极显示器的段码和共阴极显示器的段码是取反的关系



# LED数码管的显示方式：静态与动态



## 静态显示：

各数码管在显示过程中持续得到送显信号，与各数码管接口的I/O口线是专用的。

### 静态显示特点：

无闪烁，无须扫描，节省CPU时间，编程简单，但用元器件多，占I/O线多，线路复杂。



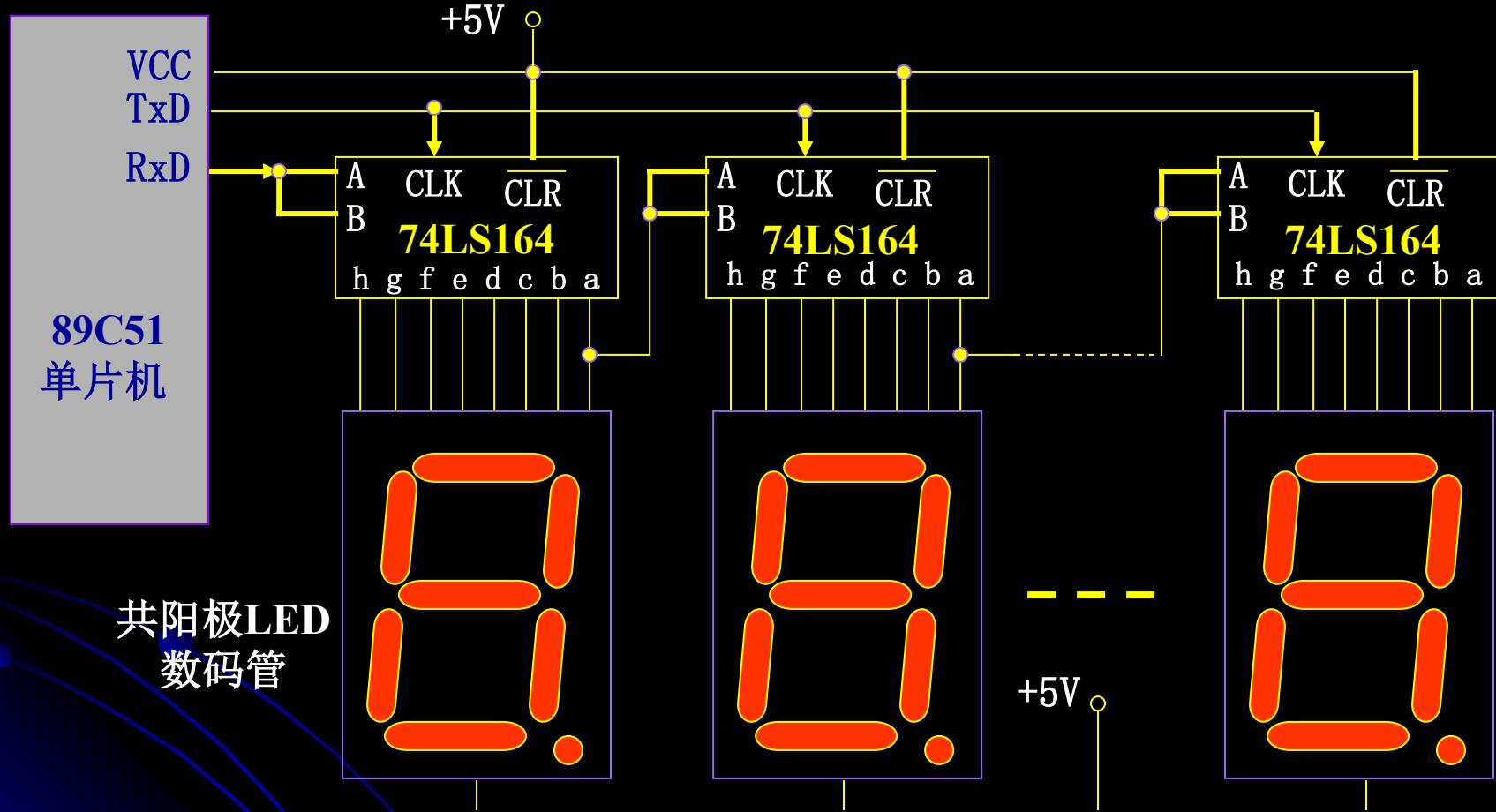
## 动态显示：

各数码管在显示过程中轮流得到送显信号，分时工作，每次只有一个数码管显示，与各数码管接口的I/O口线是共用的。

### 动态显示特点：

有闪烁，用元器件少，占I/O线少，但必须定时扫描，占用CPU时间，编程复杂。（有多个LED时尤为突出）

# LED数码管静态显示举例



74LS164是串行输入并行输出的8位移位寄存器。有几个LED就要几个74LS164，只要数据不变，送一次就可保持住，且不闪烁，编程简单。

要求：根据上图编写通过串行口和74LS164驱动共阳极LED数码管查表显示的子程序。

条件：系统有6个LED数码管，待显数据(00H-09H)已放在35H-30H单元中(分别对应十万位→个位)

DSPLY: MOV DPTR, #TABLE ;共阳极LED数码管译码表首址  
MOV R0, #30H ;待显数据缓冲区的个位地址

REDO: MOV A, @R0 ;通过R0实现寄存器间接寻址  
MOVC A, @A+DPTR ;查表

MOV SBUF, A ;经串行口发送到74LS164

JNB TI, \$ ;查询送完一个字节的第8位?

CLR TI ;为下一字节发送作准备

INC R0 ;R0指向下一个数据缓冲单元

CJNE R0, #36H, REDO ;判断是否发完6个数?

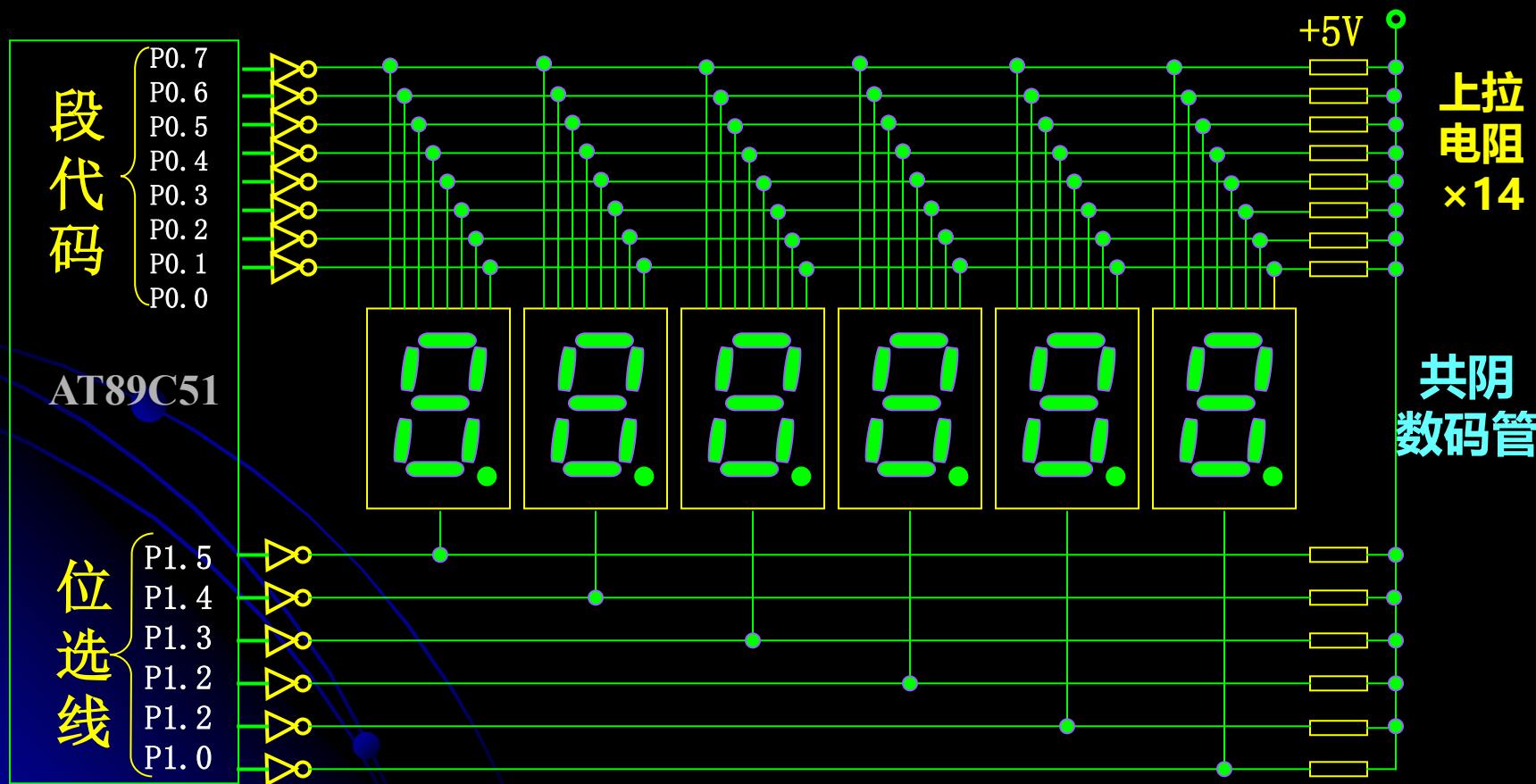
RET ;发完6个数就返回

TABLE: DB 0C0H,0F9H,0A4H,0B0H,99H ;共阳极LED译码表  
DB 92H, 82H, 0F8H, 80H,90H



# LED数码管动态显示举例

工作原理：从P0口送段代码, P1口送位选信号。段码虽同时到达 6个LED，但一次仅一个LED被选中。利用“视觉暂留”，每送一个字符并选中相应位线，延时一会儿，再送/选下一个……循环扫描即可。

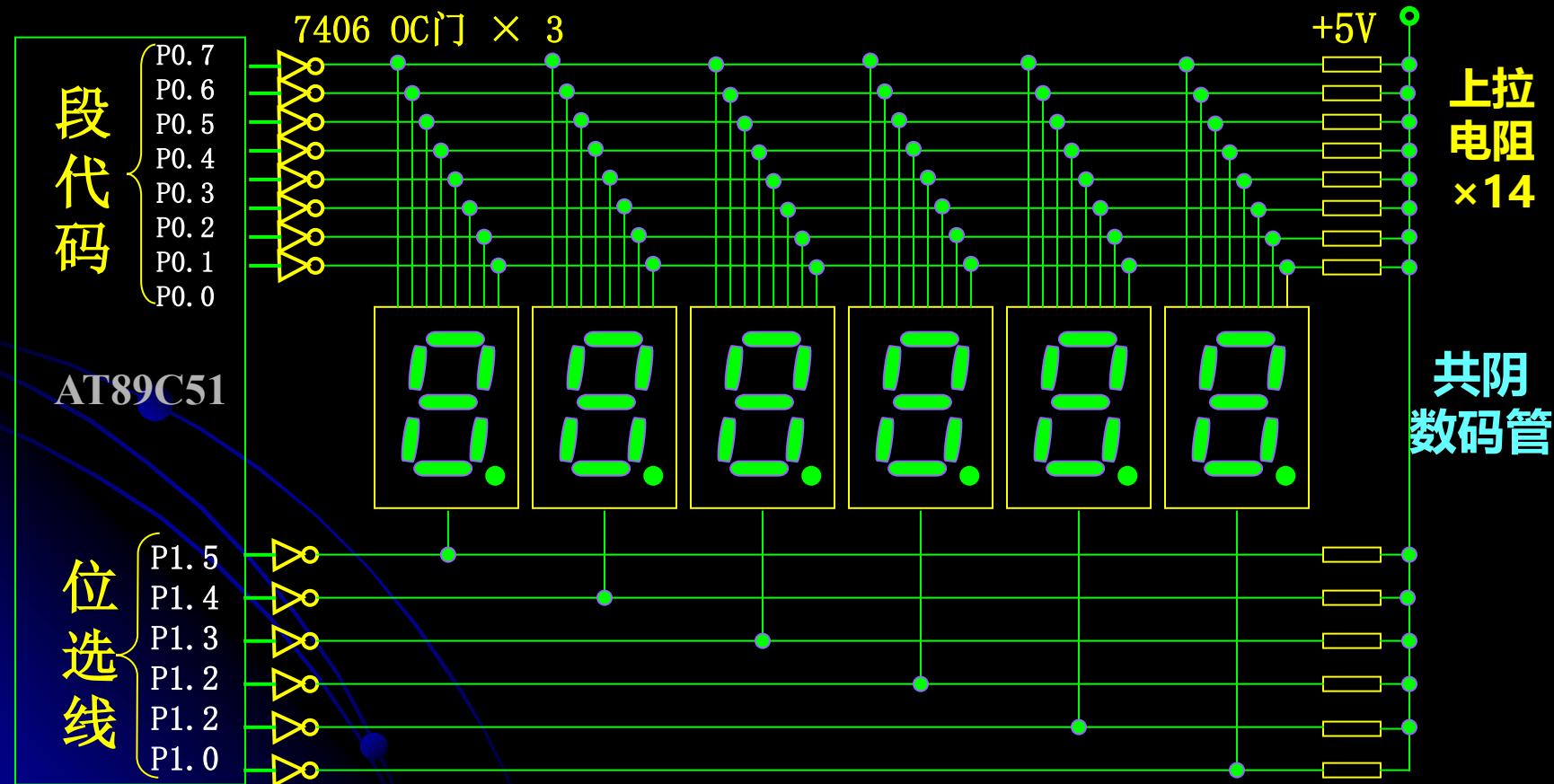


**要求:** 此处为共阴数码管, P0口送段代码, P1口送位选信号。

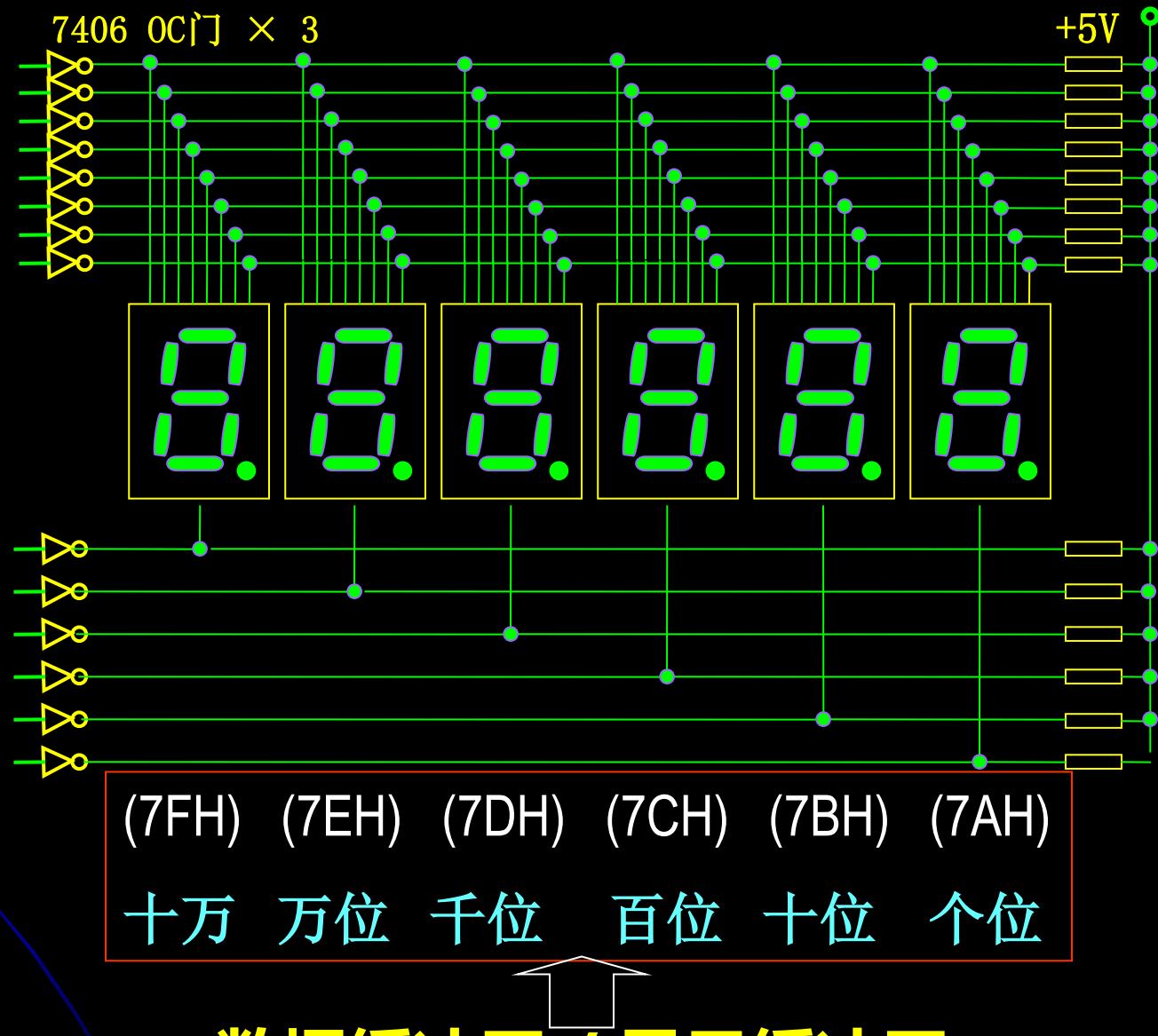
通过查表实现动态显示。

**条件:** 待显数据(00H-09H)已放在: 7FH-7AH单元中(分别对应十万位~个位)

**说明:** 由于用了反相驱动器7406, 要用共阳译码表。



P0口送段  
代码, P1  
口送位选  
信号。  
待显数据  
已经放在:  
**7FH-7AH**  
单元(分别  
对应十万  
位→个位)  
使用共阳  
译码表。



**P0口送段  
代码, P1  
口送位选  
信号。  
待显数据  
已经放在:  
7FH-7AH  
单元(分别  
对应十万  
位→个位)  
使用共阳  
译码表。**

```

DIR:    MOV    DPTR, #DSEG ;数码管译码表首址
        MOV    R0, #7AH ;待显缓冲区个位地址
        MOV    R3, #01H ;个位的位选信号=01H
LD1:    MOV    A, @R0 ;通过R0间接寻址
        MOVC  A, @A+DPTR ;查表
        MOV    P0, A ;字段码送到P0口
        MOV    P1, R3 ;位选信号送到P1口
        LCALL DELY ;调延时1ms子程序
        INC    R0 ;R0 指向下一字节
        MOV    A, R3
        JB    ACC. 5, LD2 ;判是否发完6个数?
        RL    A ;指向下一个位
        MOV    R3, A ;位选信号存回R3
        SJMP  LD1 ;跳去再显示下一个数
LD2:    RET ;发完6个数就返回
DSEG:   DB  0C0H, 0F9H, 0A4H, 0B0H, 99H;共阳译码表
        DB  92H, 82H, 0F8H, 80H, 90H

```

## 二、键盘接口

### ● 键盘

单片机系统中完成控制参数输入及修改的基本输入设备，操作人员可以通过键盘输入数据和指令，是人工干预系统的重要手段。

单片机与计算机在键盘规模/键符设置等方面差别很大。

### ● 键盘分类

① 按键值编码方式分

(硬件)编码键盘与非(硬件)编码键盘

② 按键组连接方式分

独立连接键盘与矩阵连接键盘

# 按键值编码方式: 编码键盘与非编码键盘

● **编码键盘:** 采用专用的编码/译码器件, 被按下的键由该器件译码输出相应的键码/键值。

**特点:** 增加了硬件开销, 编码因选用器件而异, 编码固定, 但编程简单。适用于规模大的键盘。

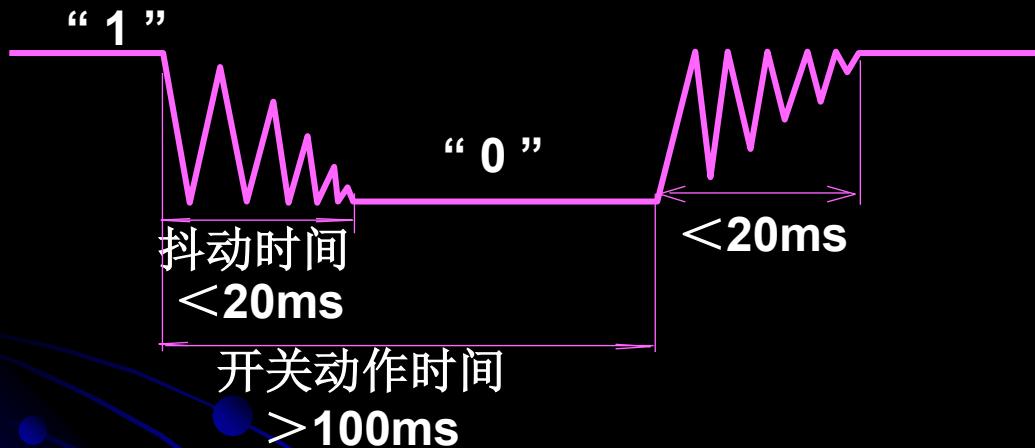
● **非编码键盘:** 单片机系统多采用此类键盘

采用软件编/译码的方式, 通过扫描, 对每个被按下的键判别输出相应的键码/键值。

**特点:** 不增加硬件开销, 编码灵活, 适用于小规模的键盘, 特别是单片机系统。但编程较复杂, 占CPU时间, 还必须进行软件“消抖”。

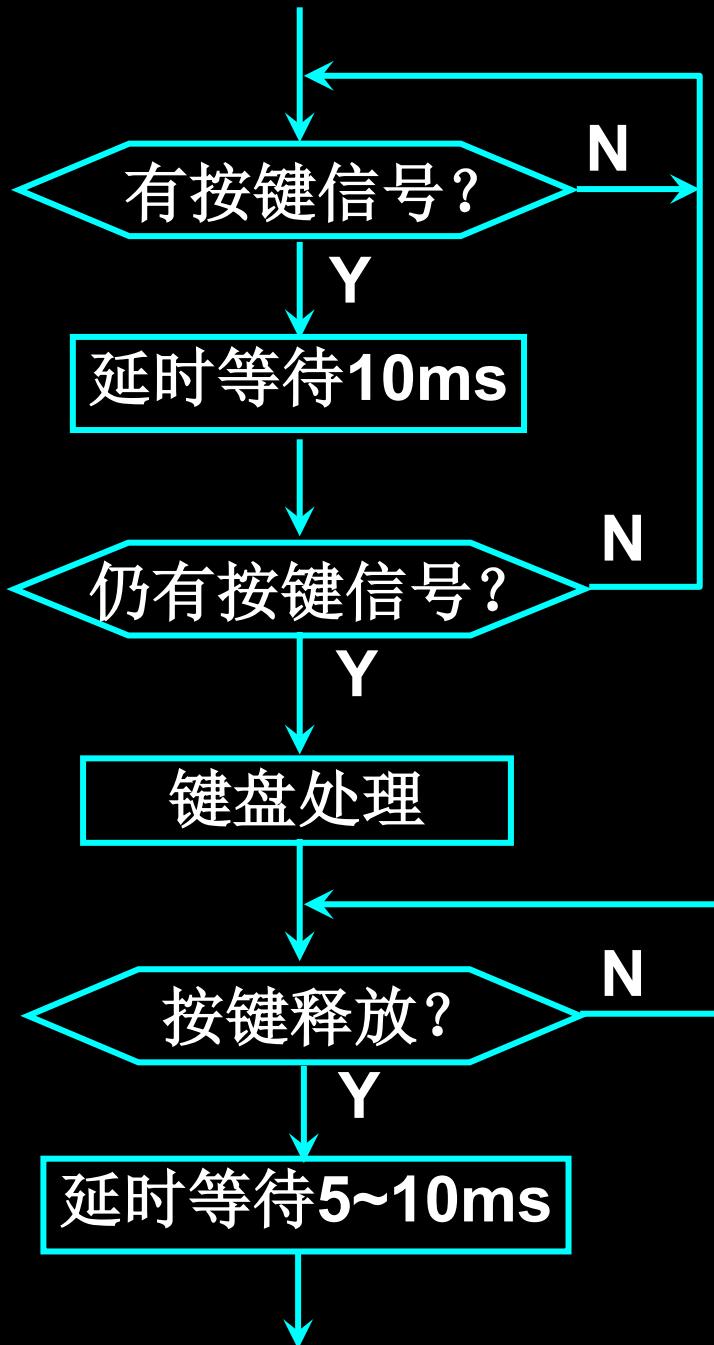
# 抖动及消除 - 软件消除抖动

抖动：键盘的按键从开启到闭合稳定，或者从闭合到完全打开，会有几毫秒的弹跳时间。

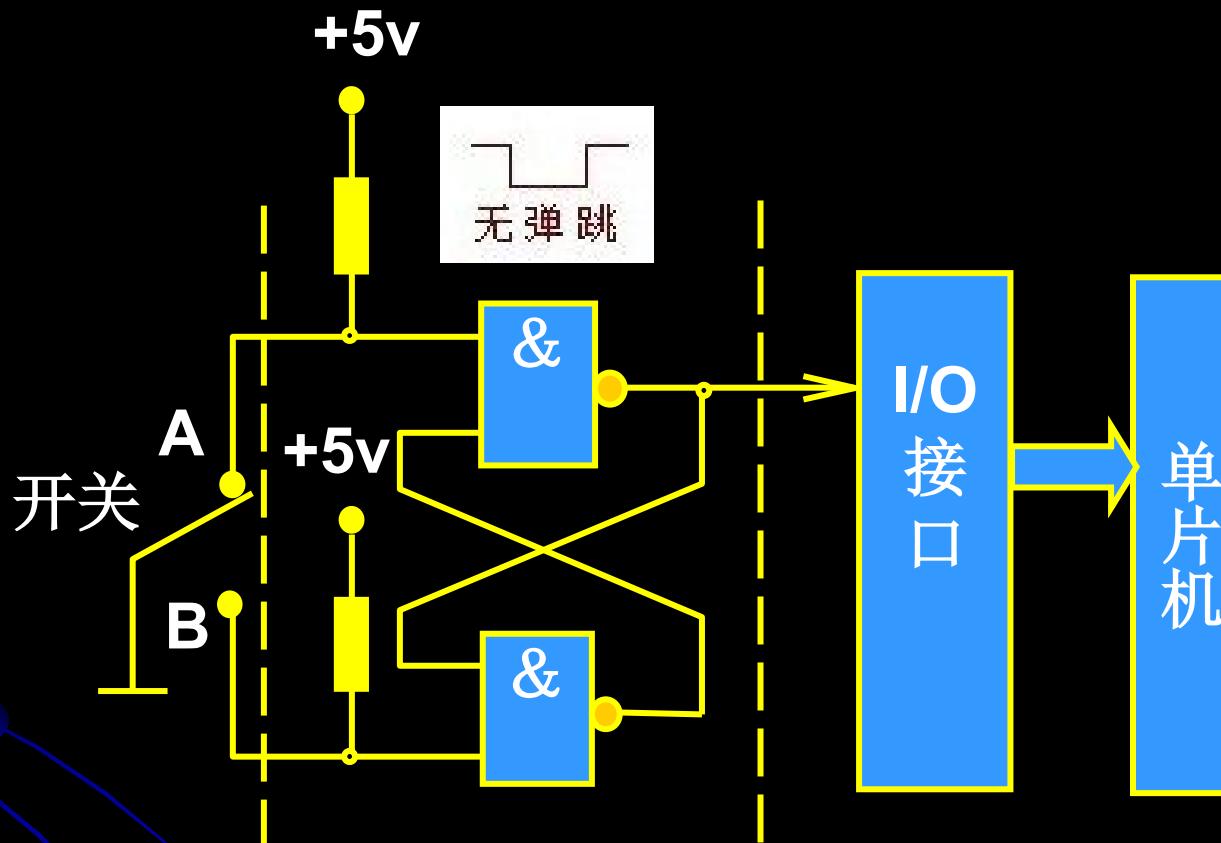


抖动会引起按一次键给多次输入的误动作。因此在键盘处理程序中，需要设法消除抖动。

软件消抖一般用延时的方法，即在程序中加入延时等待。



# 抖动及消除 - 硬件消除抖动



两个“与非”门构成一个RS触发器。当按键未按下时，输出为1；当键按下时，输出为0。此时即使有按键因为弹性抖动而产生瞬时断开（抖动跳开B），只要按键不返回原始状态A，双稳态电路的状态不改变，输出保持为0，就不会产生抖动的波形。

# 按键组连接方式: 独立连接键盘与矩阵连接键盘

● **独立连接键盘:** 每键相互独立, 各自与一条I/O线相连, CPU可直接读取该I/O线的高/低电平状态。

**特点:** 占I/O口线多, 但判键速度快, 多用于设置**控制键、功能键**。适用于键数少的场合。

● **矩阵连接键盘:** 键按矩阵排列, 各键处于矩阵行/列的结点处, CPU通过对连接行(列)的I/O线送已知电平的信号, 然后读取列(行)线的状态信息。逐线扫描, 得出键码。

**特点:** 键多时占用I/O口线少, 但判键速度慢, 多用于设置**数字键**。适用于键数多的场合。



## 独立连接式键盘例1：

KEY: JNB P1.0, FUNC1 ;逐键判别

JNB P1.1, FUNC2

JNB P1.2, FUNC3

JNB P1.3, FUNC4

RET ;无任何键按下由此返回

**FUNC1:** ..... ;做P1.0要求的“功能1”

RET

**FUNC2:** ..... ;做P1.1要求的“功能2”

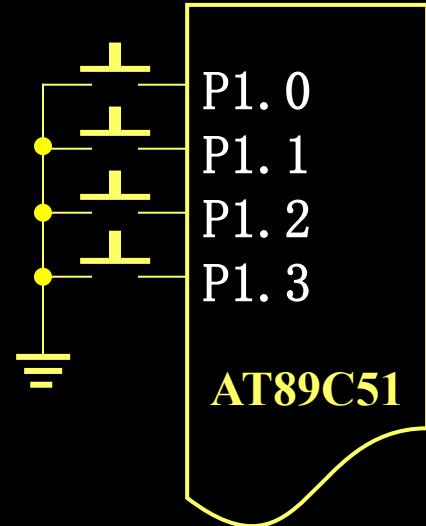
RET

**FUNC3:** ..... ;做P1.2要求的“功能3”

RET

**FUNC4:** ..... ;做P1.3要求的“功能4”

RET



**特点：**此子程序需不断(或定时)调用，否则可能发生漏判。每个键的优先级由指令顺序决定。

为解决漏判的问题，可以增加一个与门，  
连接到89C51的外部中断0 (INT0)。

ORG 0003H ;**外部中断0入口地址**  
LJMP KEY ;**转移到按键判别程序**

.....  
KEY: JNB P1.0, FUNC1 ;**逐键判别**

JNB P1.1, FUNC2

JNB P1.2, FUNC3

JNB P1.3, FUNC4

RETI ;**无任何键按下由此返回**

**FUNC1:** ..... ;**做P1.0要求的“功能1”**

RETI

**FUNC2:** ..... ;**做P1.1要求的“功能2”**

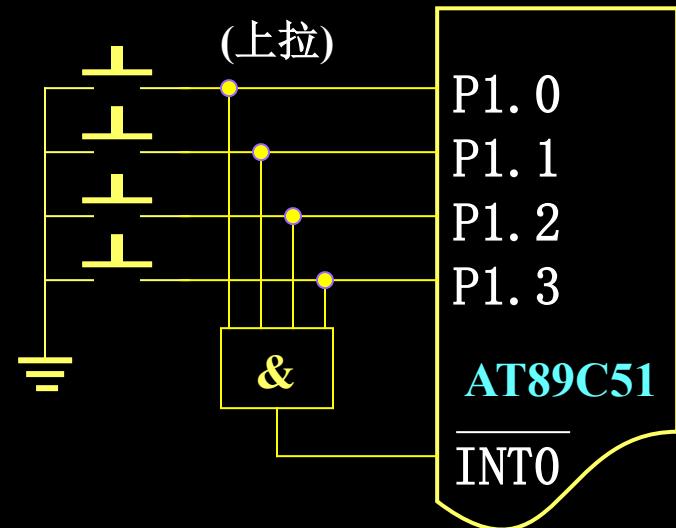
RETI

**FUNC3:** ..... ;**做P1.2要求的“功能3”**

RETI

**FUNC4:** ..... ;**做P1.3要求的“功能4”**

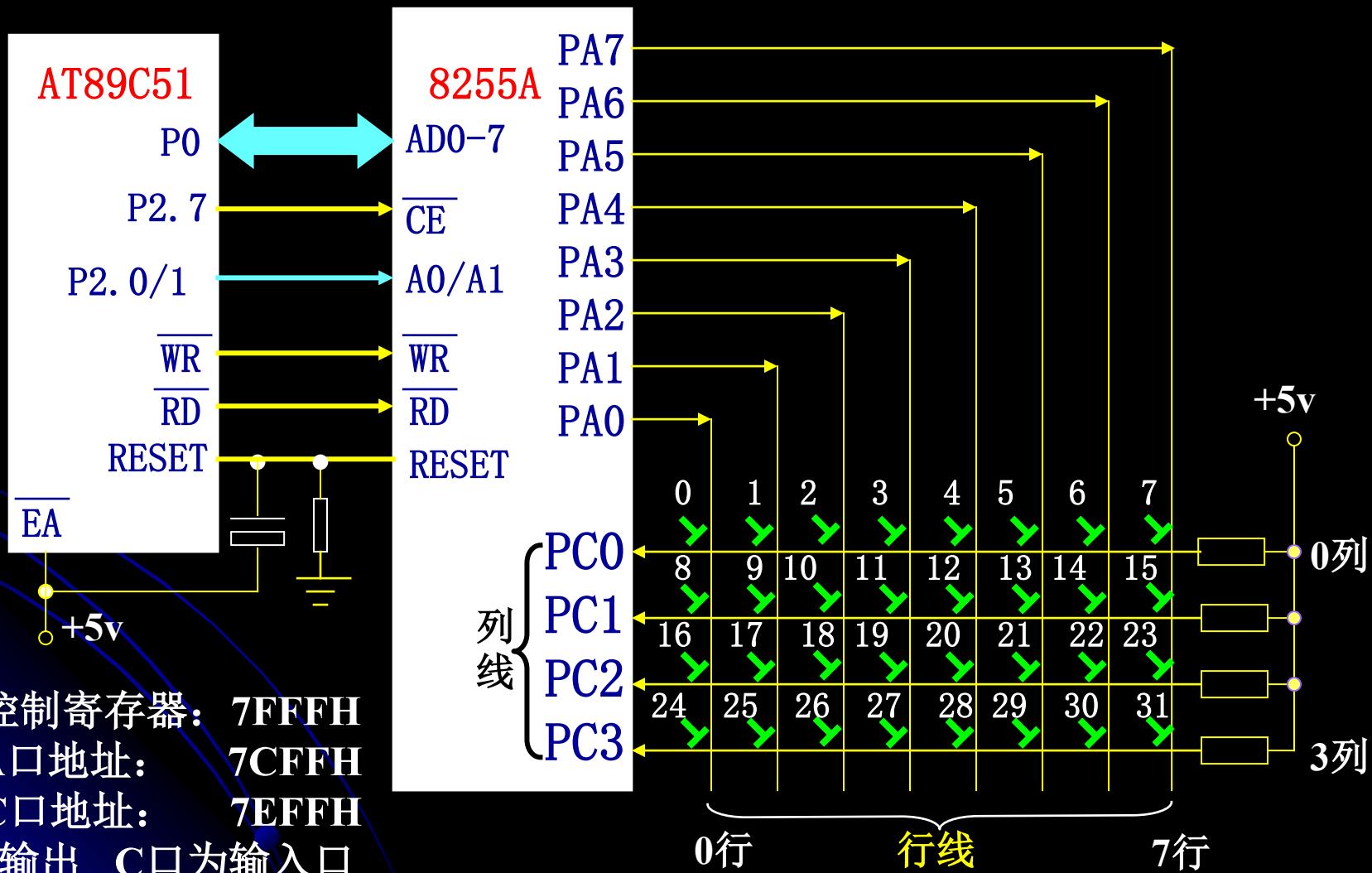
RETI



### 特点：

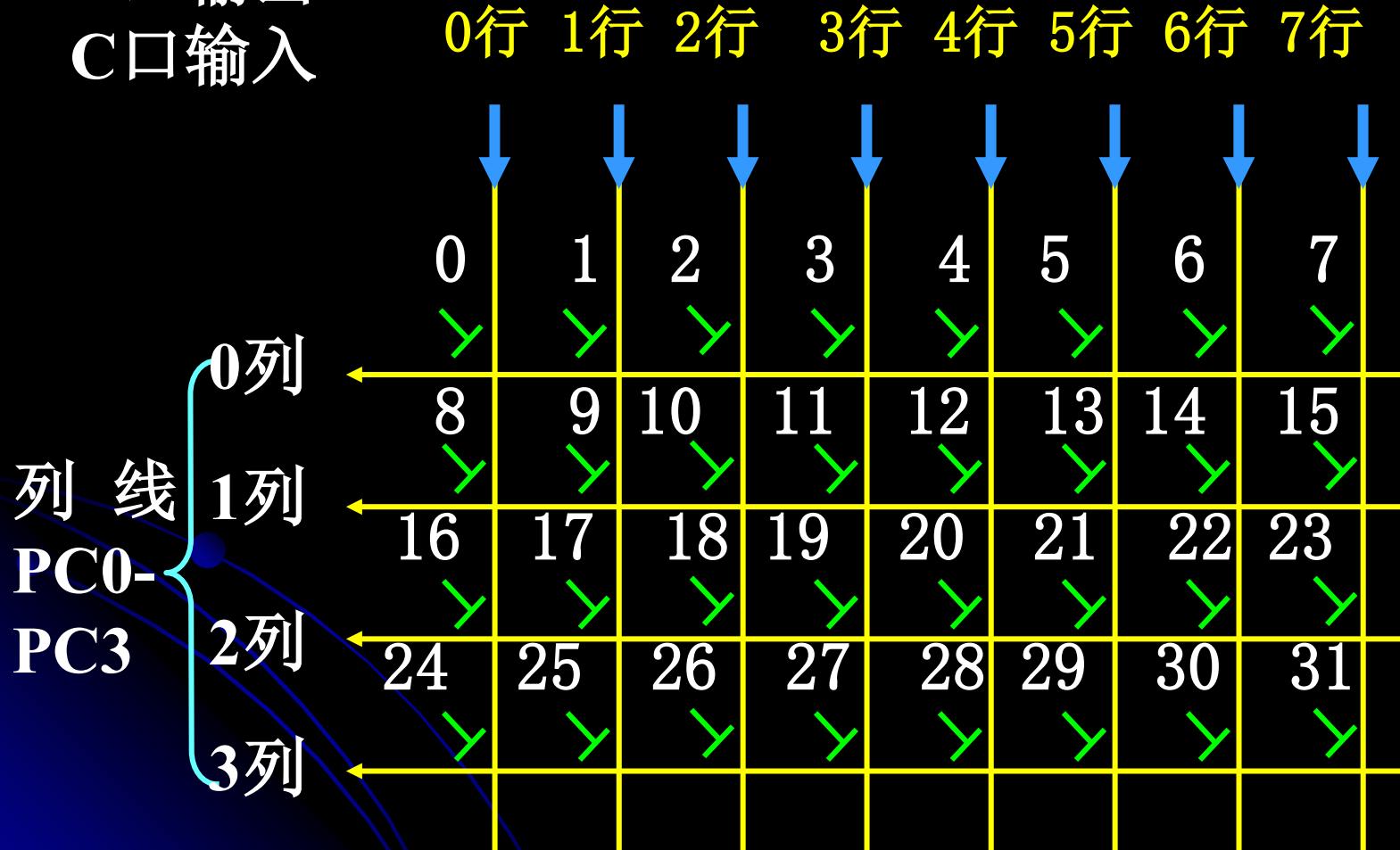
- ✓ 此子程序采用**中断查询**不会漏判,省时。
- ✓ 键的优先级由指令顺序决定。
- ✓ 为防止一次按键多次中断，在功能子程序里应安排“关/开中断指令”并“延时”。

**矩阵式键盘：**按矩阵排列，各键处于矩阵行/列的结点处，CPU逐线扫描，得出键码。



# 行线(PA0—PA7)

A口输出  
C口输入





**矩阵式键盘**扫描程序一般采用行扫描法，先由行线送出数据，送全“0”或每次只送一位“0”；然后读进列线，判断有无键按下或按键的位置并算出键值。顺序扫描。

可编程键盘/显示器接口  
芯片8279不做要求。

### 三、打印机接口电路

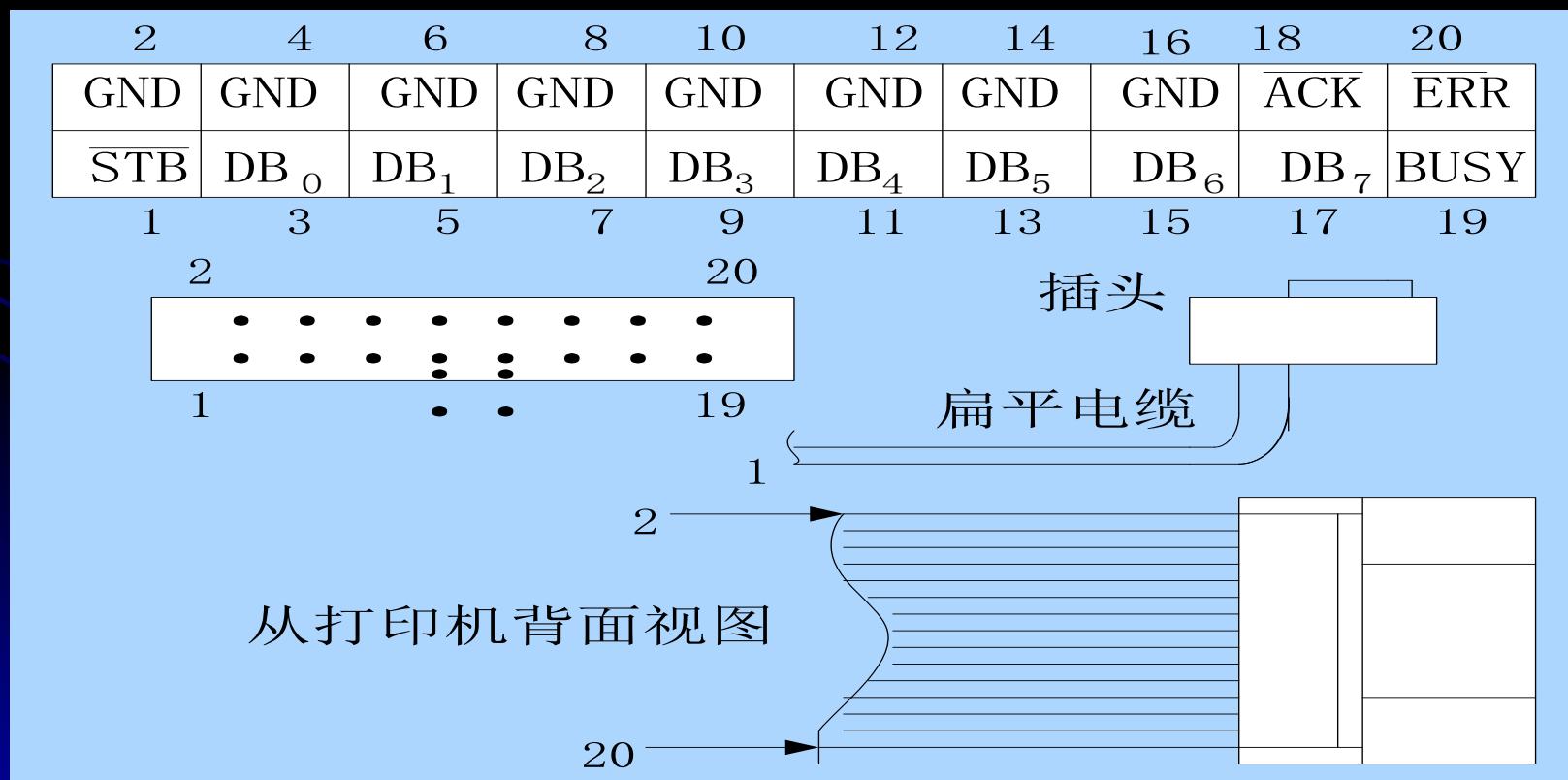
单片机系统一般使用微型打印机。如超小型点阵式打印机TP-**μP-16A**，其体积小、重量轻，多用于智能化仪器仪表等。TP-**μP-16A**的主要技术性能如下：

- 每行可打印5×7点阵字符16个，打印一行字符约1s；
- 配有240个字符的字库（其中96个标准**ASCII**代码字符，128个非标准字符和符号，16个由用户自定义的字符）；
- 带有标准8位并行接口，通过机后20芯扁平电缆及接插件与主计算机连接；
- 设置有复位/运行、自检和送纸3个开关；
- +5V电源供电。

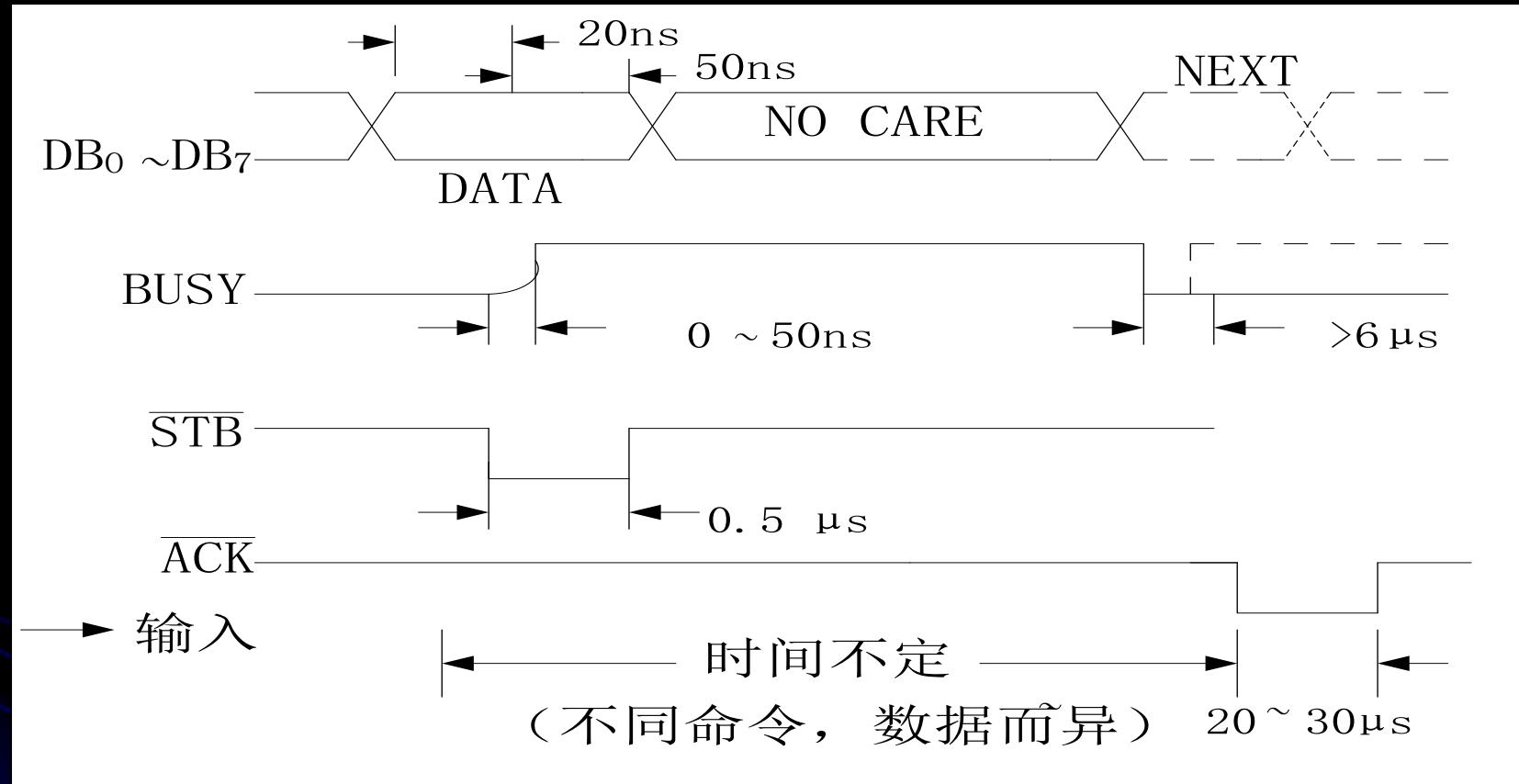
## 2. 接口要求

### TP- $\mu$ P-16A微型打印机引脚信号：

- $DB_0 \sim DB_7$ : 数据线, 由计算机单向输入打印机
- $/STB$ : 数据选通信号
- $BUSY$ : 打印机“忙”状态信号
- $/ACK$ : 打印机的应答信号
- $/ERROR$ : 出错信号

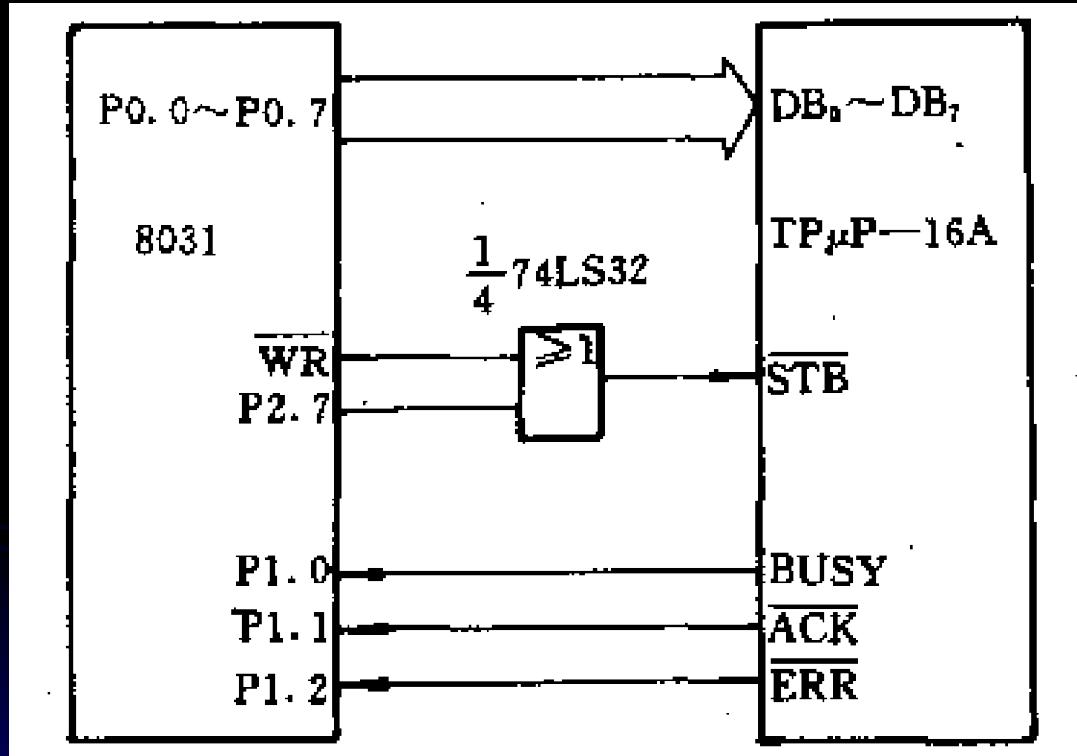


### 3. 接口信号时序



**STB**上升沿把数据线上的8位数据输入打印机内部锁存。**BUSY**为高电平，表示打印机正在处理数据。在**BUSY**有效期间，计算机不能用**STB**信号向打印机送入新的数据。当数据传送完毕时，**ACK**为低电平，表示打印机已经取走数据。

## 4 打印机与单片机接口



TP- $\mu$ P-16A微型打印机与8031单片机接口电路

8位数据线与8031的P0口直接相连。

微型打印机读入锁存选通信号STB由高8位地址线的P2.7位控制，其口地址为7FFFH。

3个联络输出信号BUSY、ACK和ERR分别接P1口的低三位P1.0、P1.1、P1.2。

8031采用**查询**方式进行管理，驱动打印机的具体程序如下：

```
PRINT: MOV R1, #60H      ; 待打印数据在RAM区的首地址
       MOV R7, #XXH      ; 待打印数据的字节数送到R7
       MOV DPTR, #7FFFH   ; 送打印机口地址
       JNB P1.2, HALM   ; 如果(ERR) P1.2=0, 出错, 则停机
LOOP:  MOV A, @R1        ; 取打印数据
       MOV @DPTR, A; 送打印机打印
       NOP
WAIT:  JB P1.0, WAIT   ; 如果(BUSY) P1.0=1, 正在打印, 等待
       INC R1
       DJNZ R7, LOOP    ; 打印完成? 未完则LOOP循环
HALM: SJMP HALM      ; 停机
```

也可采用**中断**方式对打印机进行管理，需将**ACK**联络信号连到外部中断请求INT0或者INT1。

# 本章小结

了解LED显示器、键盘、打印机等外设  
与单片机的接口电路及其相关应用。

# 微机原理复习

# 考试题型

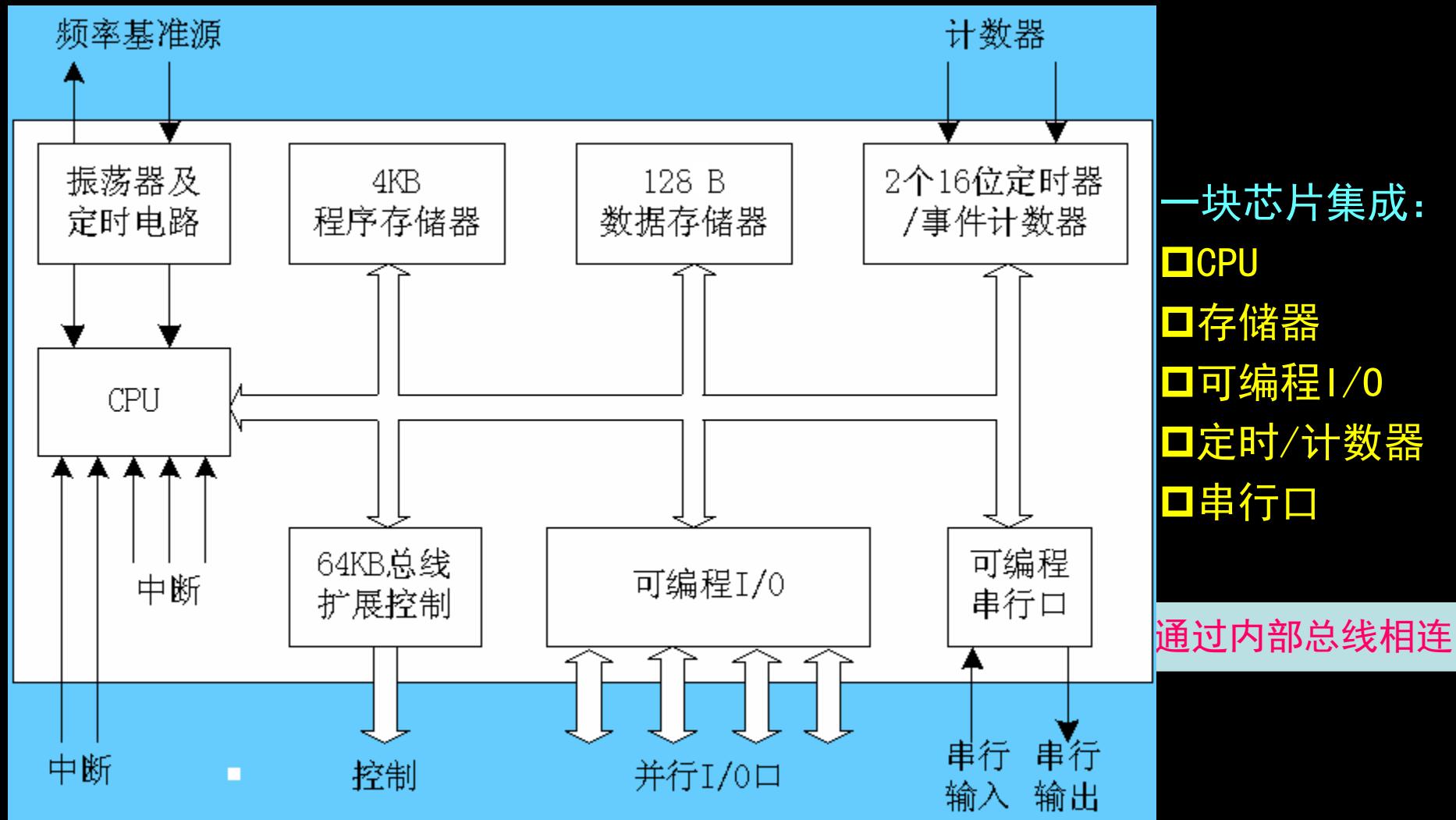
- 填空
- 简答
- 编程与应用

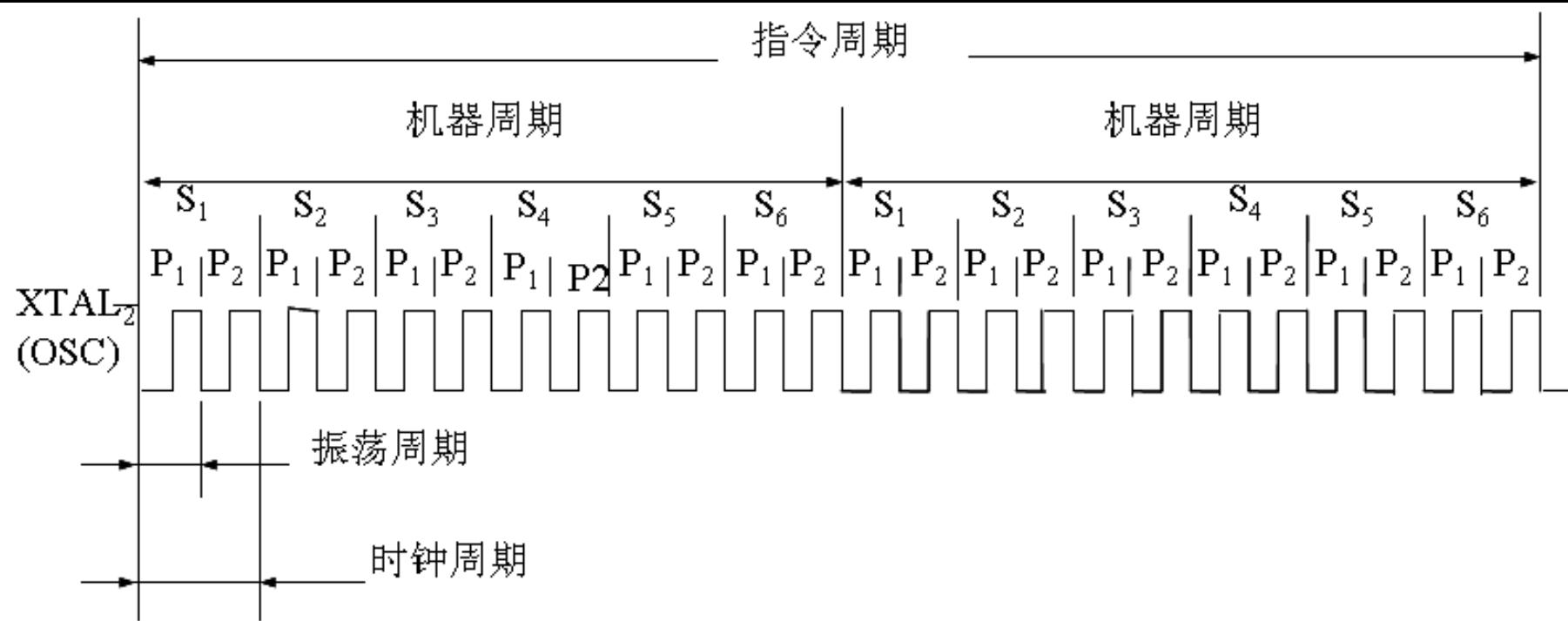
# MCS-51系列单片机分类

资源 配置 子 系 列	片内ROM形式				片 内 ROM	片 内 RAM	定时/ 计数器	中 断 源
	无	ROM	EPR OM	E <sup>2</sup> PR OM				
51子系列	8031	8051	8751	8951	4KB	128B	2×16	5
52子系列	8032	8052	8752	8952	8KB	256B	3×16	6

# MCS-51单片机的结构

以8051为例给出的单片机总体结构框图如下





MCS-51单片机各种周期的相互关系

若MCS-51单片机外接晶振为12MHz时，  
则单片机的四个周期的具体值为：

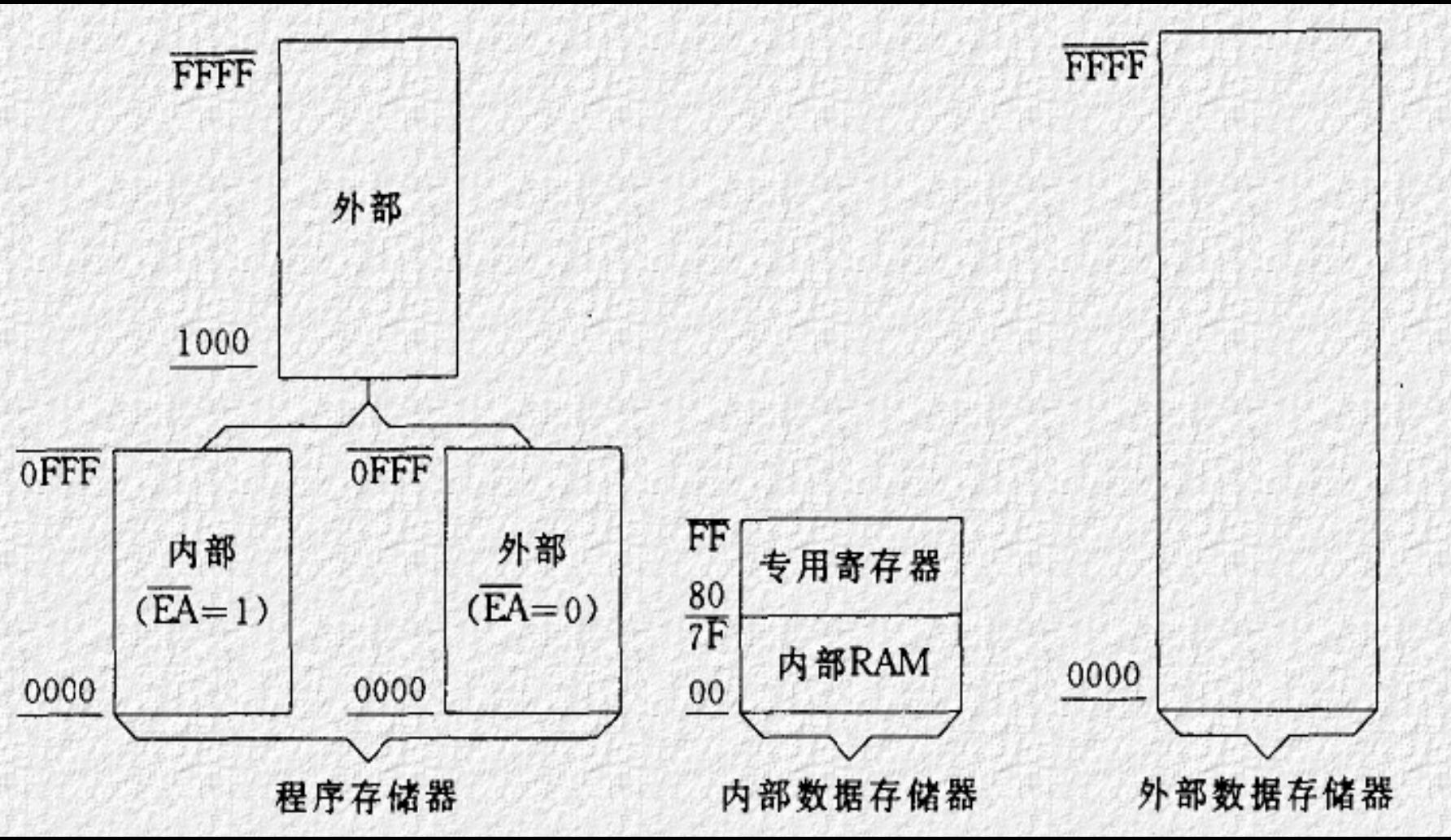
振荡周期=1/12MHz=1/12μs

时钟周期=1/6μs

机器周期=1μs

指令周期=1~4μs

# 8051存储器空间配置图



# 8051CPU区分存储器地址空间的方法

- 上述三个存储空间地址是重迭的，如何区别这三个不同的逻辑空间呢？

8051的指令系统设计了不同的数据传送指令符号：

- 访问片内、片外**ROM**指令用**MOVC**
- 访问片外**RAM**指令用**MOVX**
- 访问片内**RAM**指令用**MOV**

# 数据存储器地址空间

- 数据存储器**RAM**用于存放运算的中间结果、数据暂存和缓冲、标志位等
- 数据存储器空间也分成片内和片外两大部分，即片内**RAM**和片外**RAM**
- 8051片外数据存储器空间为64K，从0000H—0FFFFH
- 片内存储器空间为256字节，地址从00H——0FFH

# 片外RAM

- 片外数据存储器与片内数据存储器空间低地址0000H—00FFH是重迭的
- 8051有**MOV**和**MOVX**两种指令、用以区分片内、片外**RAM**空间
- 片内**RAM**使用**MOV**指令
- 片外64K **RAM**空间专门为**MOVX**指令所用

# 片内RAM

- 片内数据存储器最大可寻址256个单元
- 分为两个部分：  
    低128字节(00H—7FH)是  
        真正的RAM区  
    高128字节(80H—FFH)为  
        特殊功能寄存器(SFR)区

## 1. 低128字节RAM (00H—7FH)

- 分为三个区：
    - 工作寄存器区 (00H—1FH)
    - 位寻址区 (20H—2FH)
    - 真正的RAM区 (30H—7FH)

## 2. 高128字节RAM——特殊功能寄存器(SFR)区

- 8051片内高128字节**RAM**中，除程序计数器**PC**外，有21个专用寄存器(**SFR**)，也称特殊功能寄存器
- 它们离散地分布在**80H—FFH**的**RAM**空间中
- 访问**SFR**仅允许使用直接寻址方式(**A,B,DPTR,C**)
- 在21个特殊功能寄存器**SFR**中，有11个特殊功能寄存器具有位寻址能力，它们的字节地址正好能被**8**整除

# 程序存储器的某些单元留给系统使用

存储单元	保留目的
0000H-0002H	复位后初始化引导程序
0003H-000AH	外部中断0
000BH-0012H	定时器0溢出中断
0013H-001AH	外部中断1
001BH-002AH	定时器1溢出中断
0023H-002AH	串行端口中断

# MCS-51的指令系统

字节数 {

- 单字节指令
- 双字节指令
- 三字节指令

寻址方式7种：立即、直接、寄存器、寄存器间接、变址、相对、位寻址

111条指令：

数据传送类

算术运算类

逻辑运算与循环类

控制转移类

位操作类

# 寻址方式涉及的存储器空间

寻址方式	寻址空间（操作数存放空间）
立即寻址	程序存储器
直接寻址	片内RAM低128字节、SFR, 位地址
寄存器寻址	工作寄存器R0~R7, A, B, C, DPTR
寄存器间接寻址	片内RAM:@R0, @R1, SP 片外RAM:@R0, @R1, @DPTR
变址寻址	程序存储器:@A+PC, @A+DPTR
相对寻址	程序存储器256字节范围内:PC+偏移量
位寻址	片内RAM的位寻址区(20H~2FH字节地址) 某些可位寻址的SFR

**程序设计举例：求两个无符号数据块中的最大值。数据块的首地址分别为60H和70H，每个数据块的第一个字节都存放数据块的长度，结果存入5FH单元。**

**本例可采用分别求出两个数据块的最大值，然后比较其大小的方法，求最大值的过程可采用子程序。**

**子程序名称：QMAX**

**子程序入口条件：R1中存有数据块首地址**

**出口条件：最大值在A中**

<b>ORG</b>	<b>2000H</b>		
<b>MOV</b>	<b>SP, #2FH</b>	; 设堆栈指针	
<b>MOV</b>	<b>R1, #60H</b>	; 取第一数据块首地址送R1中	
<b>ACALL</b>	<b>QMAX</b>	; 第一次调用求最大值子程序	
<b>MOV</b>	<b>40H, A</b>	; 第一个数据块的最大值暂存40H	
<b>MOV</b>	<b>R1, #70H</b>	; 取第二数据块首地址送R1中	
<b>ACALL</b>	<b>QMAX</b>	; 第二次调用求最大值子程序	
<b>CJNE</b>	<b>A, 40H, NEXT</b>	; 两个最大值进行比较	
<b>NEXT:</b>		; A大, 则转LP	
	<b>MOV</b>	<b>A, 40H</b>	; A小, 则把40H中内容送入A
<b>LP:</b>	<b>MOV</b>	<b>5FH, A</b>	
	<b>SJMP</b>	<b>\$</b>	

# 子程序：

	<b>ORG</b>	<b>2200H</b>	
<b>QMAX:</b>	<b>MOV</b>	<b>A, @R1</b>	; 取数据块长度
	<b>MOV</b>	<b>R2, A</b>	; R2做计数器
	<b>CLR</b>	<b>A</b>	; A清零，准备做比较
<b>LP1:</b>	<b>INC</b>	<b>R1</b>	; 指向下一个数据地址
	<b>CLR</b>	<b>C</b>	; 准备做减法
	<b>SUBB</b>	<b>A, @R1</b>	; 用减法做比较
	<b>JNC</b>	<b>LP3</b>	; 若A大，则转LP3
	<b>MOV</b>	<b>A, @R1</b>	; A小，则将大数送A中
	<b>SJMP</b>	<b>LP4</b>	; 无条件转LP4
<b>LP3:</b>	<b>ADD</b>	<b>A, @R1</b>	; 恢复A中值
<b>LP4:</b>	<b>DJNZ</b>	<b>R2, LP1</b>	; 计数器减1，不为零，转继续比较
	<b>RET</b>		; 比较完，子程序返回

# 存储器扩展的编址技术

## 线选法

所谓线选法，就是直接以系统的地址作为存储芯片的片选信号，为此只需把高位地址线与存储芯片的片选信号直接连接即可。特点是简单明了，不需增加另外电路。缺点是存储空间不连续。适用于小规模单片机系统的存储器扩展。

## 译码法

所谓译码法就是使用译码器对系统的高位地址进行译码，以其译码输出作为存储芯片的片选信号。这是一种最常用的存储器编址方法，能有效地利用空间，特点是存储空间连续，适用于大容量多芯片存储器扩展。

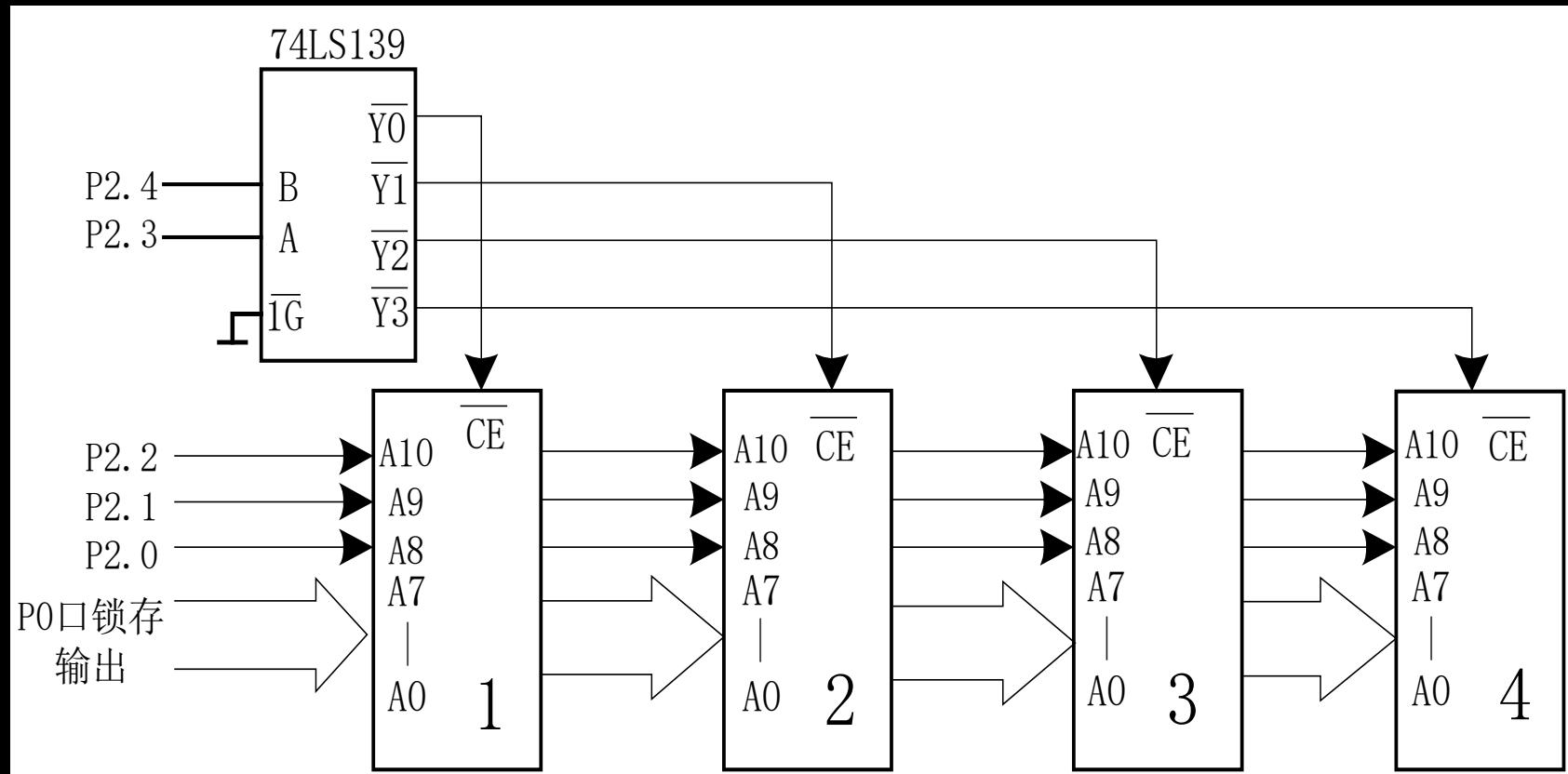
【例】现有 $2K \times 8$ 位存储器芯片，需扩展 $8K \times 8$ 位存储结构采用译码法进行扩展。

扩展8KB的存储器结构需2KB的存储器芯片4块。2K的存储器所用的地址线为 $A_0 \sim A_{10}$ 共11根地址线和片选信号与CPU的连接如下表所示。

## 80C51与存储器的线路连接表

80C51		存储器	
$P_0$ 口经锁存器锁存形成 $A_0 \sim A_7$		与 $A_0 \sim A_7$ 相连	
$P_{2.0}$ 、 $P_{2.1}$ 、 $P_{2.2}$		与 $A_8 \sim A_{10}$ 相连	
$P0$ 口		与 $D_0 \sim D_7$ 相连	
$P_{2.4}$	$P_{2.3}$	译码输出与存储器的片选信号连接	
0	0	/Y0	与存储器1的片选信号相连
0	1	/Y1	与存储器2的片选信号相连
1	0	/Y2	与存储器3的片选信号相连
1	1	/Y3	与存储器4的片选信号相连

$P_{2.3}$ 、 $P_{2.4}$ 作为二-四译码器的译码地址，译码输出作为扩展4个存储器芯片的片选信号， $P_{2.5}$ 、 $P_{2.6}$ 、 $P_{2.7}$ 悬空。扩展连线图如图所示。



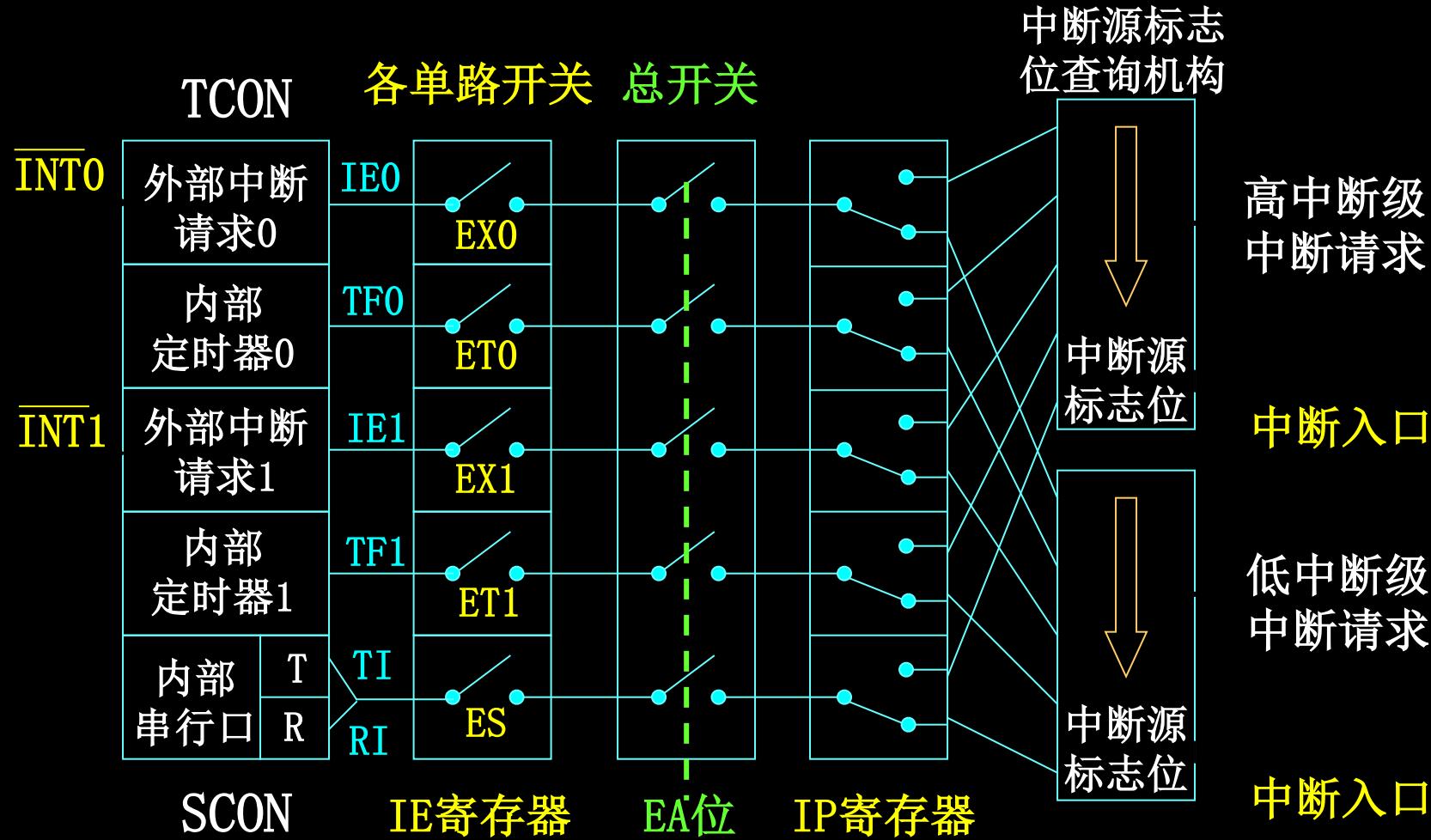
采用译码器扩展8KB存储器连线图

这样得到四个芯片的地址分配如下表所示。

译码方式地址分配表

	$P_{2.7}$	$P_{2.6}$	$P_{2.5}$	$P_{2.4}$	$P_{2.3}$	$P_{2.2} \dots P_0$	地址范围
芯片1	0	0	0	0	0	0 .... 0	0000H---
	0	0	0	0	0	1 .... 1	07FFH
芯片2	0	0	0	0	1	0 .... 0	0800H---
	0	0	0	0	1	1 .... 1	0FFFH
芯片3	0	0	0	1	0	0 .... 0	1000H---
	0	0	0	1	0	1 .... 1	17FFH
芯片4	0	0	0	1	1	0 .... 0	1800H—
	0	0	0	1	1	1 .... 1	1FFFFH

# 中断系统总体结构



注：各中断允许控制位=0, 开关断开； =1, 开关接通

# 中断处理过程

一个完整的中断过程包括中断申请、中断响应、中断处理、中断返回。

## 中断响应

中断响应是指在CPU响应中断的条件得到满足后，CPU对中断源中断请求的回答。

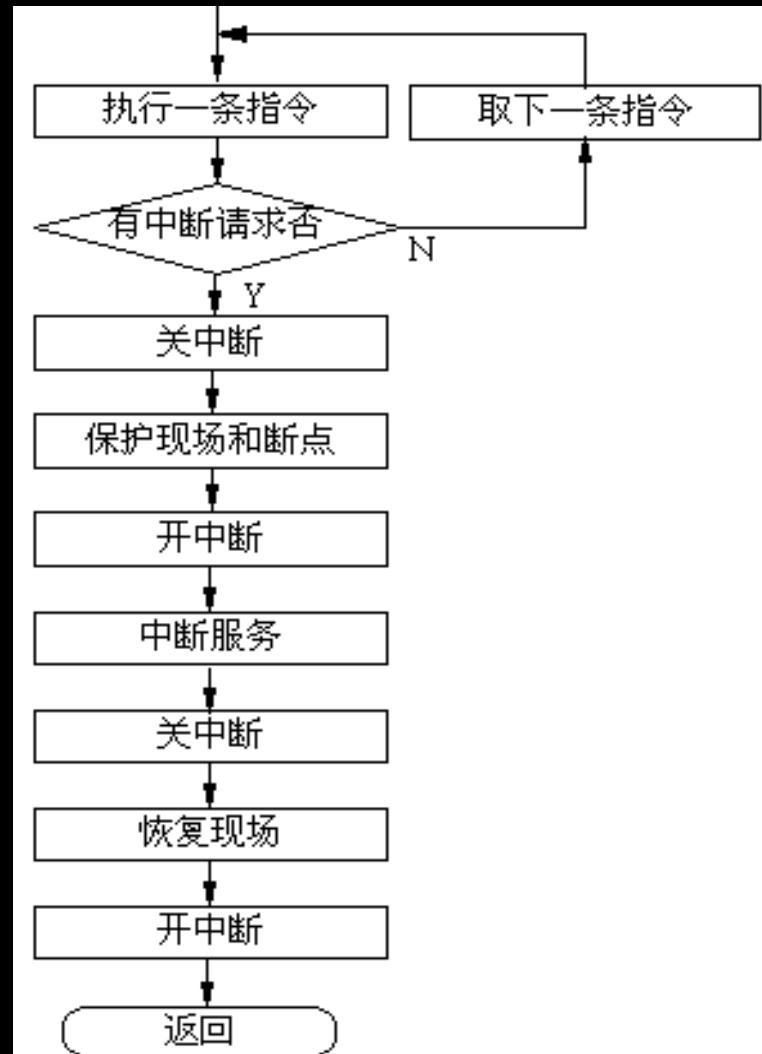
### CPU响应中断的条件有：

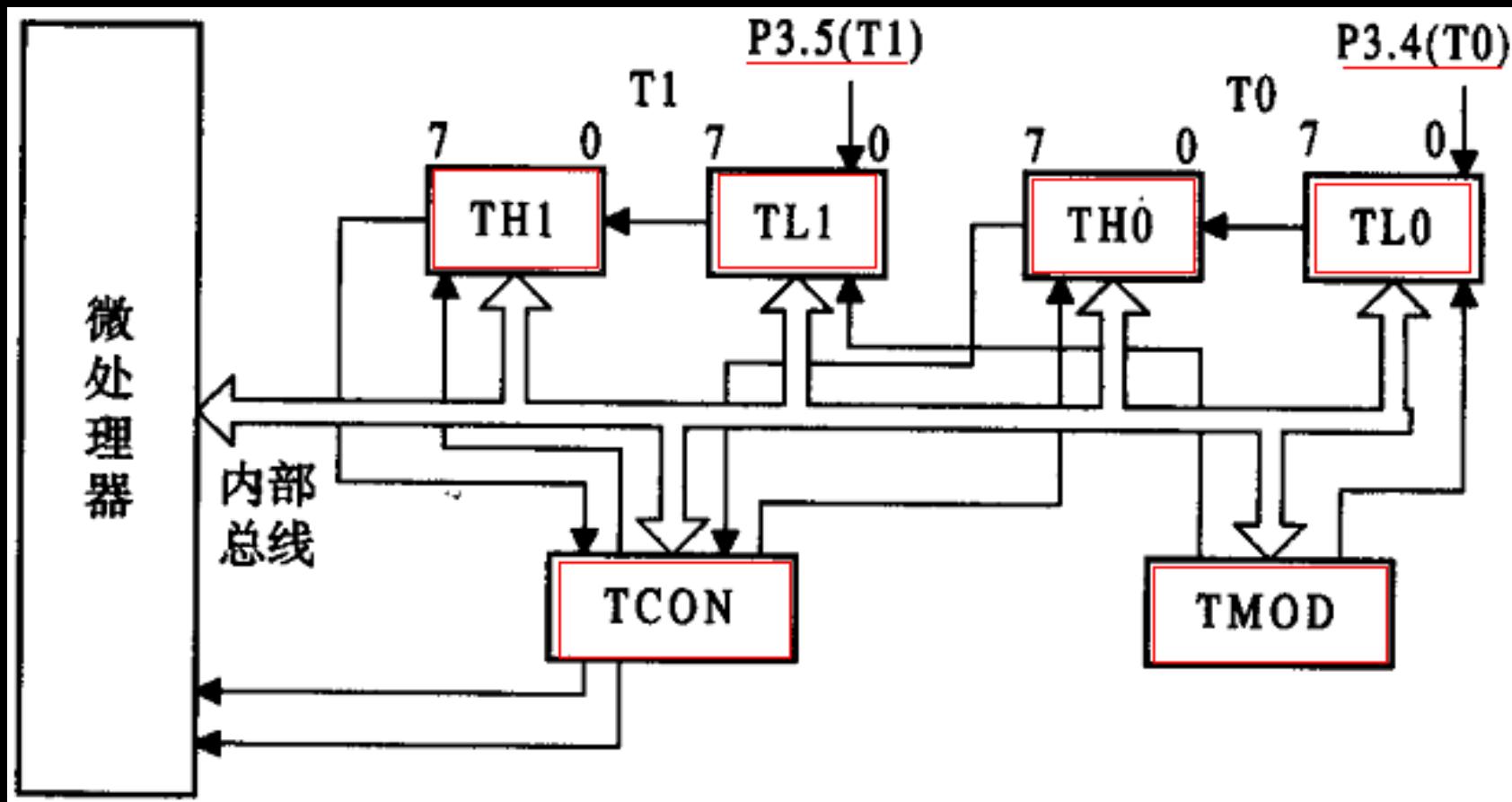
- ① 首先要有中断源发出中断申请；
- ② CPU中断是开放的，即中断总允许位EA=1，CPU允许所有中断源申请中断；
- ③ 申请中断的中断源的中断允许位为1，即此中断源可以向CPU申请中断。

# 中断处理和中断返回

中断服务程序从入口地址开始执行，直到返回指令RETI为止。这个过程称为中断处理或中断服务。

通常为了使现场信息不受到破坏或者造成混乱，一般在保护和恢复现场时，CPU不能响应新的中断请求。





## 定时器/计数器结构框图

M1	M0	方式	说 明
0	0	0	13 位定时器(TH的 8 位和TL的低 5 位)
0	1	1	16 位定时器/计数器
1	0	2	自动重装入初值的 8 位计数器
1	1	3	T0 分成两个独立的 8 位计数器, T1 在方式 3 时停止工作

## 一、方式 0 的应用

例 1 设单片机晶振频率为 6 MHz, 利用定时器输出周期为2ms的方波。

选用定时器/计数器T0 作定时器, 输出为P1.0引脚, 2ms的方波可由间隔1ms的高低电平相间而成, 只需每隔1ms对P1.0 取反一次。

定时1ms的初值:

机器周期=12 ÷ 6 MHz= 2  $\mu$ s

1 ms内T0 需要计数的值:

$$N = 1 \text{ ms} \div 2 \mu\text{s} = 500$$

使用方式 0 的 13 位计数器，T0 的初值X：

$$X = M - N = 8192 - 500 = 7692 = 1E0CH = 1111000001100B$$

13 位计数器中，低 8 位 TL0 只使用 5 位，其余均计入高 8 位 TH0 的初值，则 T0 的初值调整为：

$$TH0=0F0H, TL0=0CH$$

TMOD初始化: TMOD=00000000B=00H

(GATE=0, C/T=0, M1=0, M0=0)

TCON初始化: 启动TR0=1

IE初始化: 开放中断EA=1, 定时器T0 中断允许ET0=1

程序清单如下：

**ORG 0000H**

**AJMP START; 初始化程序入口**

**ORG 000BH ; T0中断矢量**

**AJMP TOINT ; T0中断入口**

**ORG 0300H**

**START: MOV SP, #60H; 堆栈**

**MOV TH0, #0F0H ; T0赋初值**

**MOV TL0, #0CH**

**MOV TMOD, #00H**

**SETB TR0 ; 启动T0**

**SETB ET0 ; 开T0中断**

**SETB EA ; 开总允许中断**

**MAIN: AJMP MAIN ; 主程序**

**T0INT: CPL P1.0 ; 输出值取反**

**MOV TL0, #0CH ; 重新置计数初值**

**MOV TH0, #0F0H**

**RETI**

上例是采用中断方式进行工作，即当定时器从初值开始递增计数，一直到计满溢出时，由单片机内部硬件对TF0置1，然后向CPU请求中断，在CPU响应中断转向中断服务程序后，TF0位由硬件自动清零。

书上P180的程序是采用查询方式进行工作，查询溢出中断标志位TF1是否为1，为1则转中断服务程序去执行，查询有效后需及时将TF1位清零。

# 一、异步通信

- 特点：以字符为单位，一个字符一个字符地传送，每个字符均有起始位、停止位作为开始和结束的标志。
- 异步串行通信规定了传输数据的结构即帧格式：

起始位	数据位	奇偶校验位	停止位
-----	-----	-------	-----

## 二、同步通信

- 以数据块为单位的数据传送
- 同步通信先发送同步字符，之后连续发送数据，数据之间不能有间隔，直到数据发送完毕
- 一次性发送整块数据，速度要比异步通信快

面向位的同步协议数据桢格式：

8位	8位	8位	≥0位	16位	8位
01111110	A	C	I	FC	01111110
开始标志	地址场	控制场	信息场	校验场	结束标志



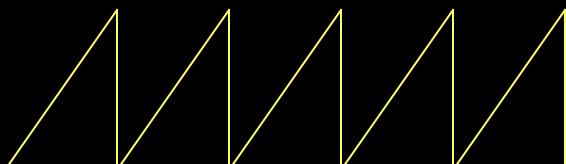
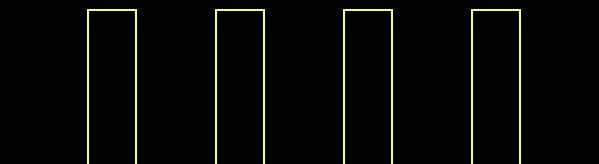
# DAC0832 的编程应用举例

## 例1 产生矩形波

```
LL:MOV A, #00H ;低电平
    MOV DPTR, #7FFFH
    MOVX @DPTR, A ;送转换
    LCALL DELAYL ;低宽度
    MOV A, #0FFH ;高电平
    MOVX @DPTR, A ;送转换
    LCALL DELAYH ;高宽度
    SJMP LL
```

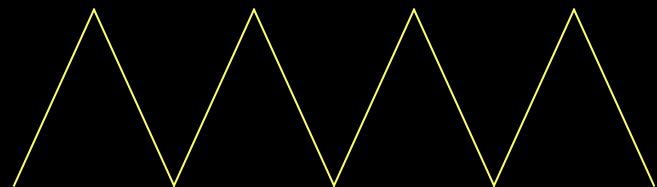
## 例2 产生锯齿波

```
MOV A, #00H ;起始值
    MOV DPTR, #7FFFH
MM: MOVX @DPTR, A ;送转换
    INC A
    NOP
    NOP
    NOP ;决定坡度
    SJMP MM
```



# ● DAC0832编程应用举例：例3产生三角波

```
MOV A, #00H
MOV DPTR, #7FFFH
SS1: MOVX @DPTR, A ;送转换
NOP
NOP
NOP
SS2: INC A      ;等速上升
CJNE A, #0FFH, SS1
SS3: DEC A
MOVX @DPTR, A
NOP
NOP
NOP      ;等速下降
CJNE A, #00H, SS3
SJMP SS1
```



三角波

同样的编程思路，若要产生  
如下的梯形波也很容易：



梯形波

## A/D转换器接口

**采样保持电路:**把一个时间连续的信号变换为时间离散的信号，并将采样信号保持一段时间。

**量化编码电路:**

**直接A/D转换器:**能将输入的模拟电压直接转换为输出的数字代码(无中间变量)。如逐次逼近型

**间接A/D转换器:**首先将输入的模拟信号转换成与之成正比的时间或频率，然后通过计数的方式转换成数字量输出。如双积分型

祝大家取得好成绩！

# 微机原理复习

# 考试题型

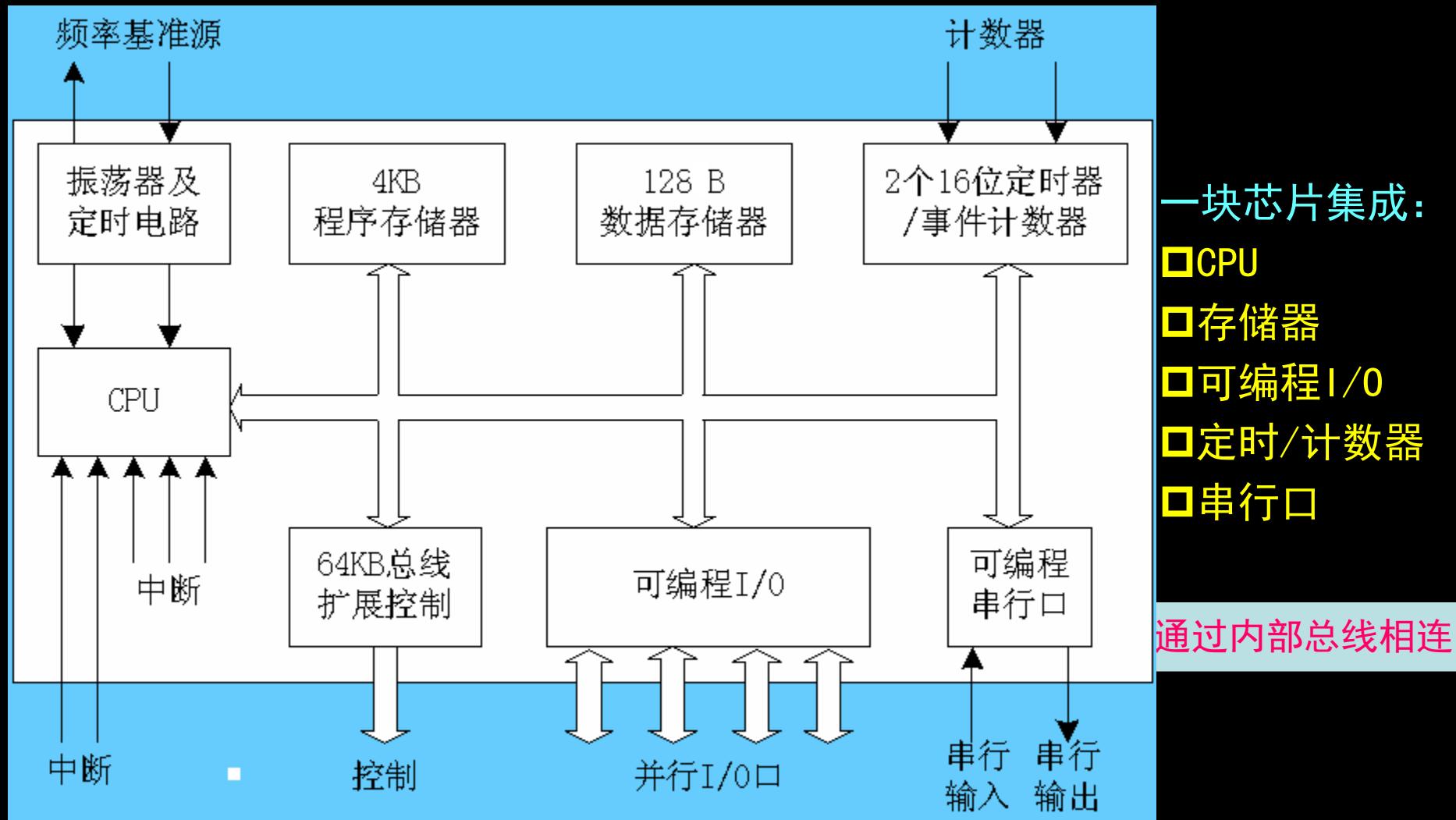
- 填空
- 简答
- 编程与设计

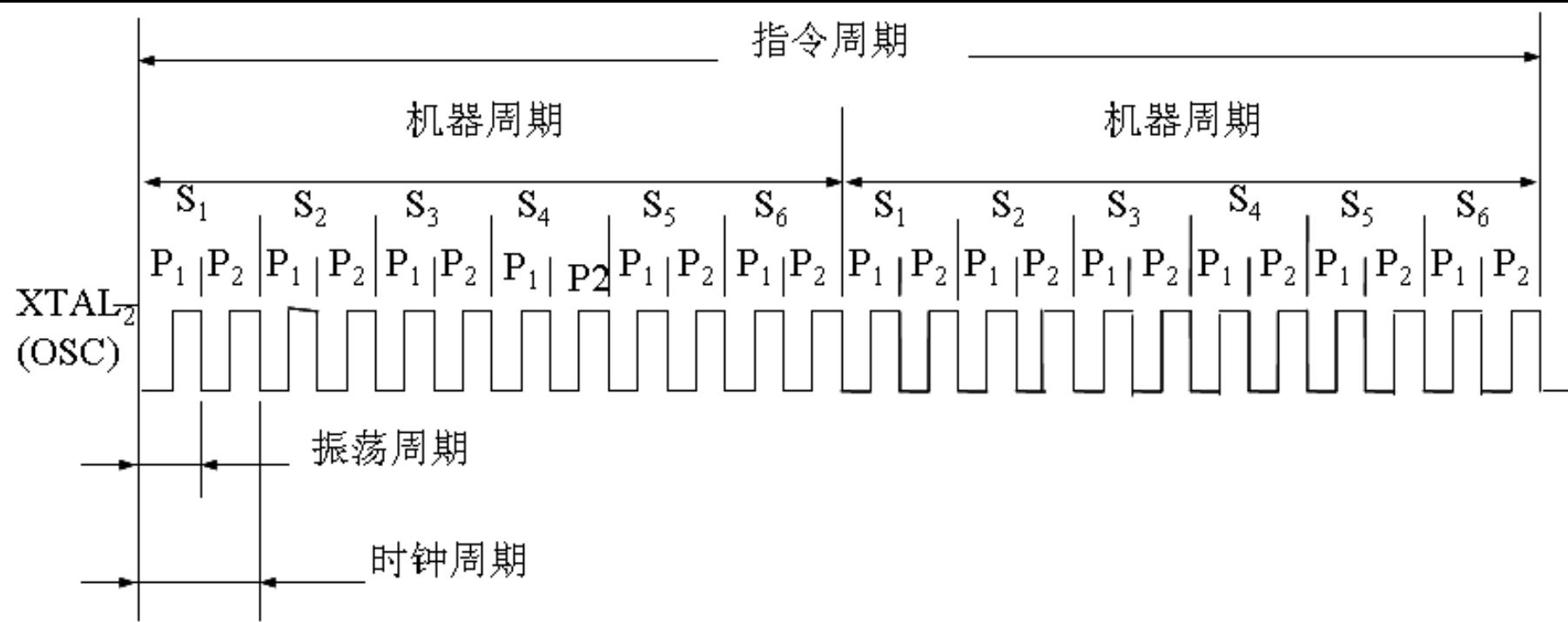
# MCS-51系列单片机分类

资源 配置 子 系 列	片内ROM形式				片 内 ROM	片 内 RAM	定时/ 计数器	中 断 源
	无	ROM	EPR OM	E <sup>2</sup> PR OM				
51子系列	8031	8051	8751	8951	4KB	128B	2×16	5
52子系列	8032	8052	8752	8952	8KB	256B	3×16	6

# MCS-51单片机的结构

以8051为例给出的单片机总体结构框图如下





MCS-51单片机各种周期的相互关系

若MCS-51单片机外接晶振为12MHz时，  
则单片机的四个周期的具体值为：

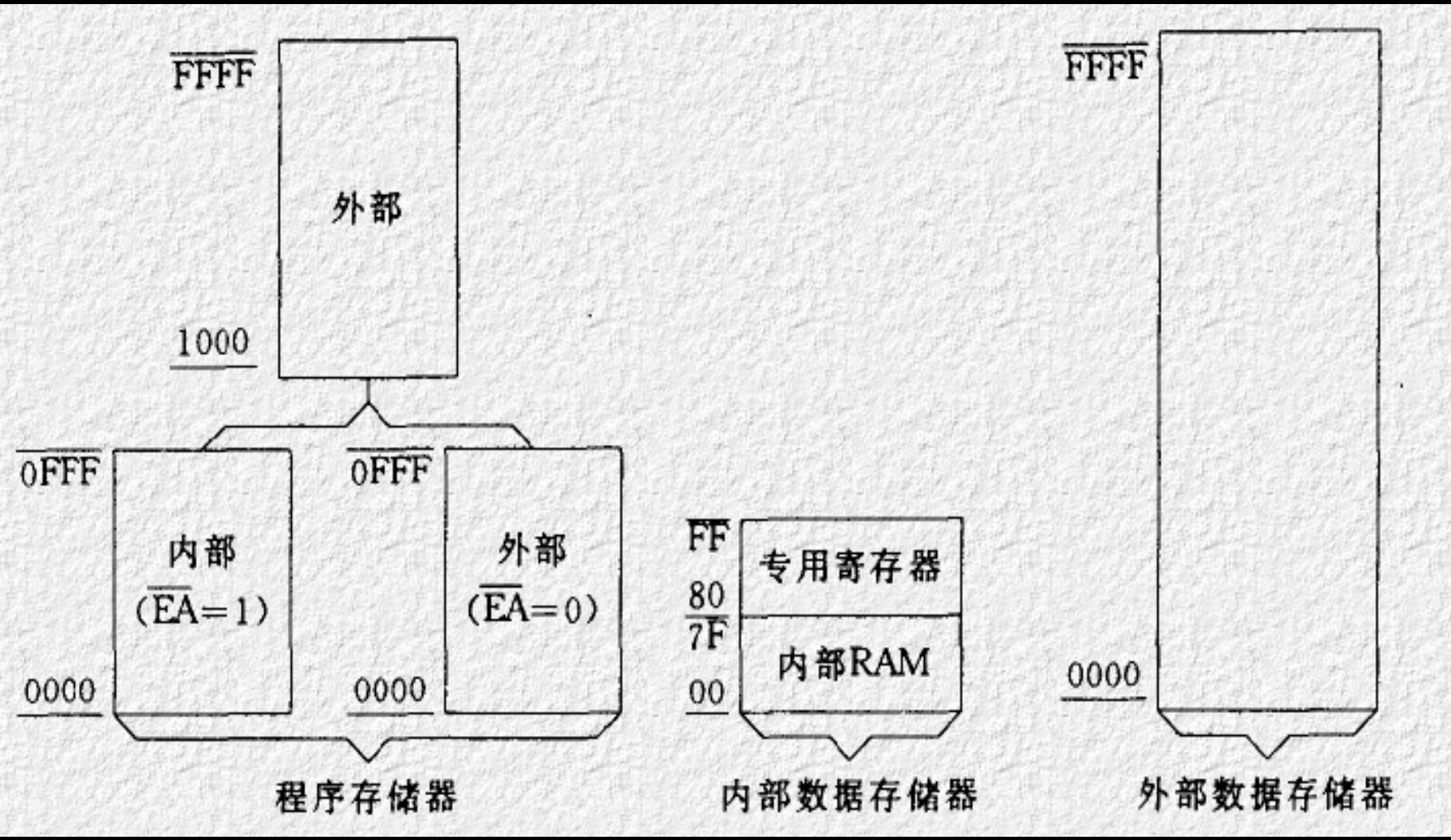
振荡周期=1/12MHz=1/12μs

时钟周期=1/6μs

机器周期=1μs

指令周期=1~4μs

# 8051存储器空间配置图



# 8051CPU区分存储器地址空间的方法

- 上述三个存储空间地址是重迭的，如何区别这三个不同的逻辑空间呢？

8051的指令系统设计了不同的数据传送指令符号：

- 访问片内、片外**ROM**指令用**MOVC**
- 访问片外**RAM**指令用**MOVX**
- 访问片内**RAM**指令用**MOV**

# 数据存储器地址空间

- 数据存储器**RAM**用于存放运算的中间结果、数据暂存和缓冲、标志位等
- 数据存储器空间也分成片内和片外两大部分，即片内**RAM**和片外**RAM**
- 8051片外数据存储器空间为64K，从0000H—0FFFFH
- 片内存储器空间为256字节，地址从00H——0FFH

# 片外RAM

- 片外数据存储器与片内数据存储器空间低地址0000H—00FFH是重迭的
- 8051有**MOV**和**MOVX**两种指令、用以区分片内、片外**RAM**空间
- 片内**RAM**使用**MOV**指令
- 片外64K **RAM**空间专门为**MOVX**指令所用

# 片内RAM

- 片内数据存储器最大可寻址256个单元
- 分为两个部分：  
    低128字节(00H—7FH)是  
        真正的RAM区  
    高128字节(80H—FFH)为  
        特殊功能寄存器(SFR)区

## 1. 低128字节RAM (00H—7FH)

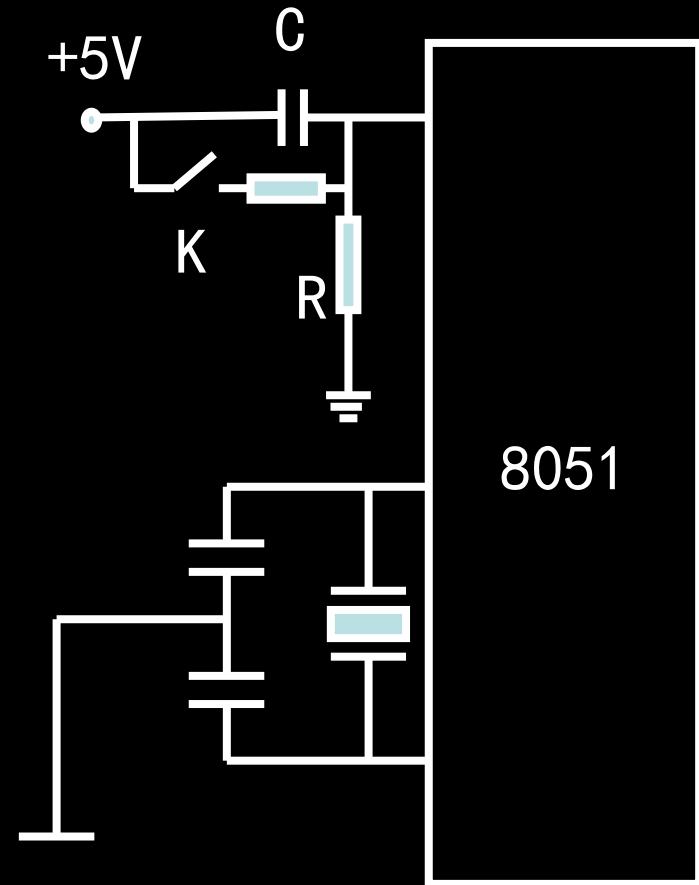
- 分为三个区：
    - 工作寄存器区 (00H—1FH)
    - 位寻址区 (20H—2FH)
    - 真正的RAM区 (30H—7FH)

## 2. 高128字节RAM——特殊功能寄存器(SFR)区

- 8051片内高128字节**RAM**中，除程序计数器**PC**外，有21个专用寄存器(**SFR**)，也称特殊功能寄存器
- 它们离散地分布在**80H—FFH**的**RAM**空间中
- 访问**SFR**仅允许使用直接寻址方式(**A,B,DPTR,C**)
- 在21个特殊功能寄存器**SFR**中，有11个特殊功能寄存器具有位寻址能力，它们的字节地址正好能被**8**整除

最少外部电路条件下，可以独立工作的单片机系统

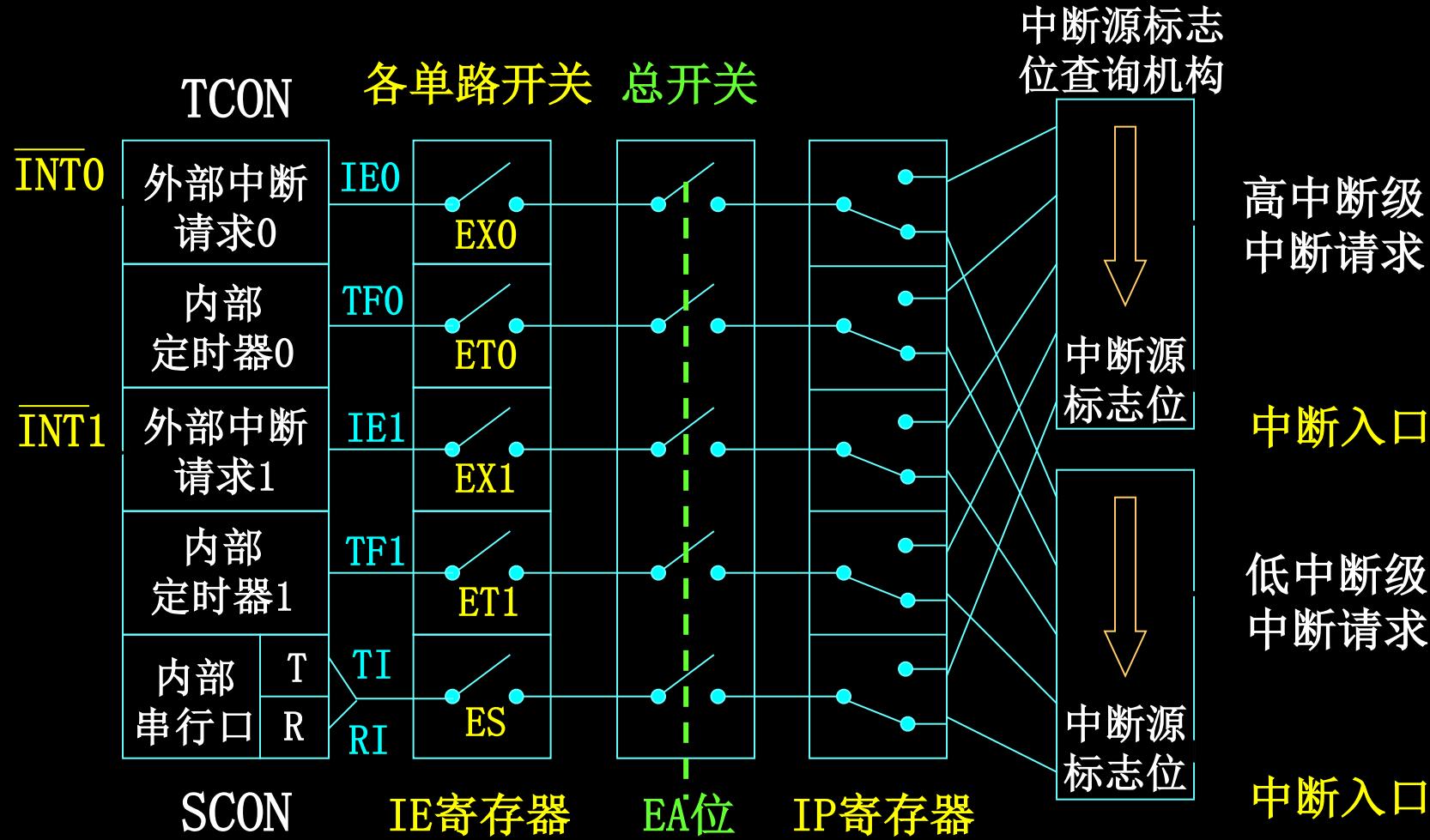
—— 最小系统



# 程序存储器的某些单元留给系统使用

存储单元	保留目的
0000H-0002H	复位后初始化引导程序
0003H-000AH	外部中断0
000BH-0012H	定时器0溢出中断
0013H-001AH	外部中断1
001BH-002AH	定时器1溢出中断
0023H-002AH	串行端口中断

# 中断系统总体结构



注: 各中断允许控制位=0, 开关断开; =1, 开关接通

# MCS-51的指令系统

字节数 {

- 单字节指令：49条
- 双字节指令：45条
- 三字节指令：17条

寻址方式7种：立即、直接、寄存器、寄存器间接、变址、相对、位寻址

111条指令：

数据传送类（29条）

算术运算类（24条）

逻辑运算与循环类（24条）

控制转移类（17条）

位操作类（17条）

# 寻址方式涉及的存储器空间

寻址方式	寻址空间（操作数存放空间）
立即寻址	程序存储器
直接寻址	片内RAM低128字节、SFR, 位地址
寄存器寻址	工作寄存器R0~R7, A, B, C, DPTR
寄存器间接寻址	片内RAM:@R0, @R1, SP 片外RAM:@R0, @R1, @DPTR
变址寻址	程序存储器:@A+PC, @A+DPTR
相对寻址	程序存储器256字节范围内:PC+偏移量
位寻址	片内RAM的位寻址区(20H~2FH字节地址) 某些可位寻址的SFR

# 存储器扩展的编址技术

## 线选法

所谓线选法，就是直接以系统的地址作为存储芯片的片选信号，为此只需把高位地址线与存储芯片的片选信号直接连接即可。特点是简单明了，不需增加另外电路。缺点是存储空间不连续。适用于小规模单片机系统的存储器扩展。

## 译码法

所谓译码法就是使用译码器对系统的高位地址进行译码，以其译码输出作为存储芯片的片选信号。这是一种最常用的存储器编址方法，能有效地利用空间，特点是存储空间连续，适用于大容量多芯片存储器扩展。

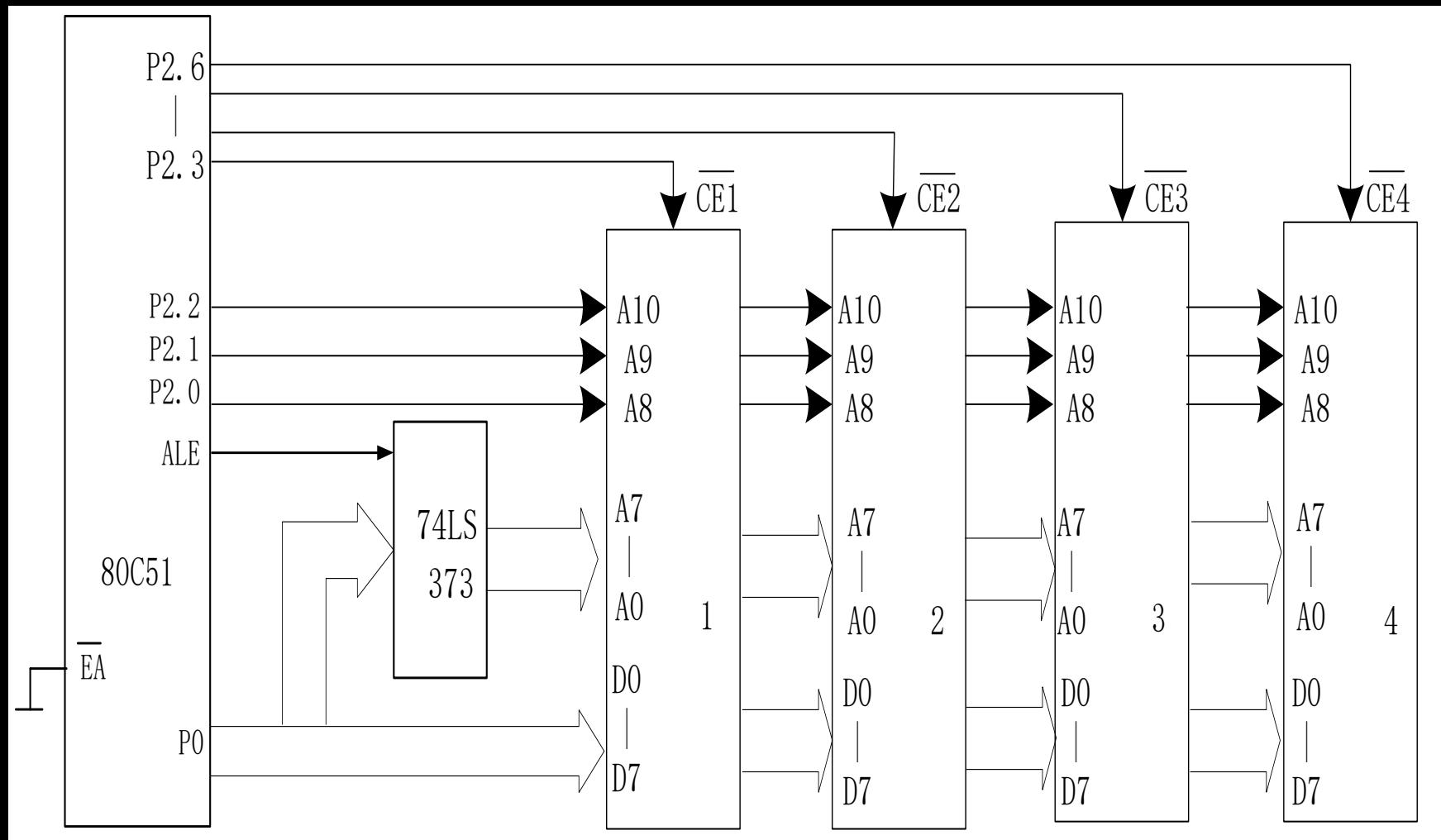
【例】现有 $2K \times 8$ 位存储器芯片，需扩展 $8K \times 8$ 位存储结构，采用线选法进行扩展。

扩展8KB的存储器结构需2KB的存储器芯片4块。2K的存储器所用的地址线为A0~A10共11根地址线，如何与CPU连接？

# 80C51与存储器的线路连接

80C51	存储器
P0口经锁存器锁存形成A0~A7	与A0~A7相连
P2.0、P2.1、P2.2	与A8~A10相连
P0口	与D0~D7相连
P2.3	与存储器1的片选信号相连
P2.4	与存储器2的片选信号相连
P2.5	与存储器3的片选信号相连
P2.6	与存储器4的片选信号相连

# 扩展存储器的硬件连接



线选法连线图

得到四个芯片的地址分配：

地址不连续！

线选方式地址分配表

	A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub> .... A <sub>0</sub>	地址范围
芯片1	0	1	1	1	0	0 .... 0	7000H---
	0	1	1	1	0	1 .... 1	77FFH
芯片2	0	1	1	0	1	0 .... 0	6800H---
	0	1	1	0	1	1 .... 1	6FFFH
芯片3	0	1	0	1	1	0 .... 0	5800H---
	0	1	0	1	1	1 .... 1	5FFFH
芯片4	0	0	1	1	1	0 .... 0	3800H—
	0	0	1	1	1	1 .... 1	3FFFH

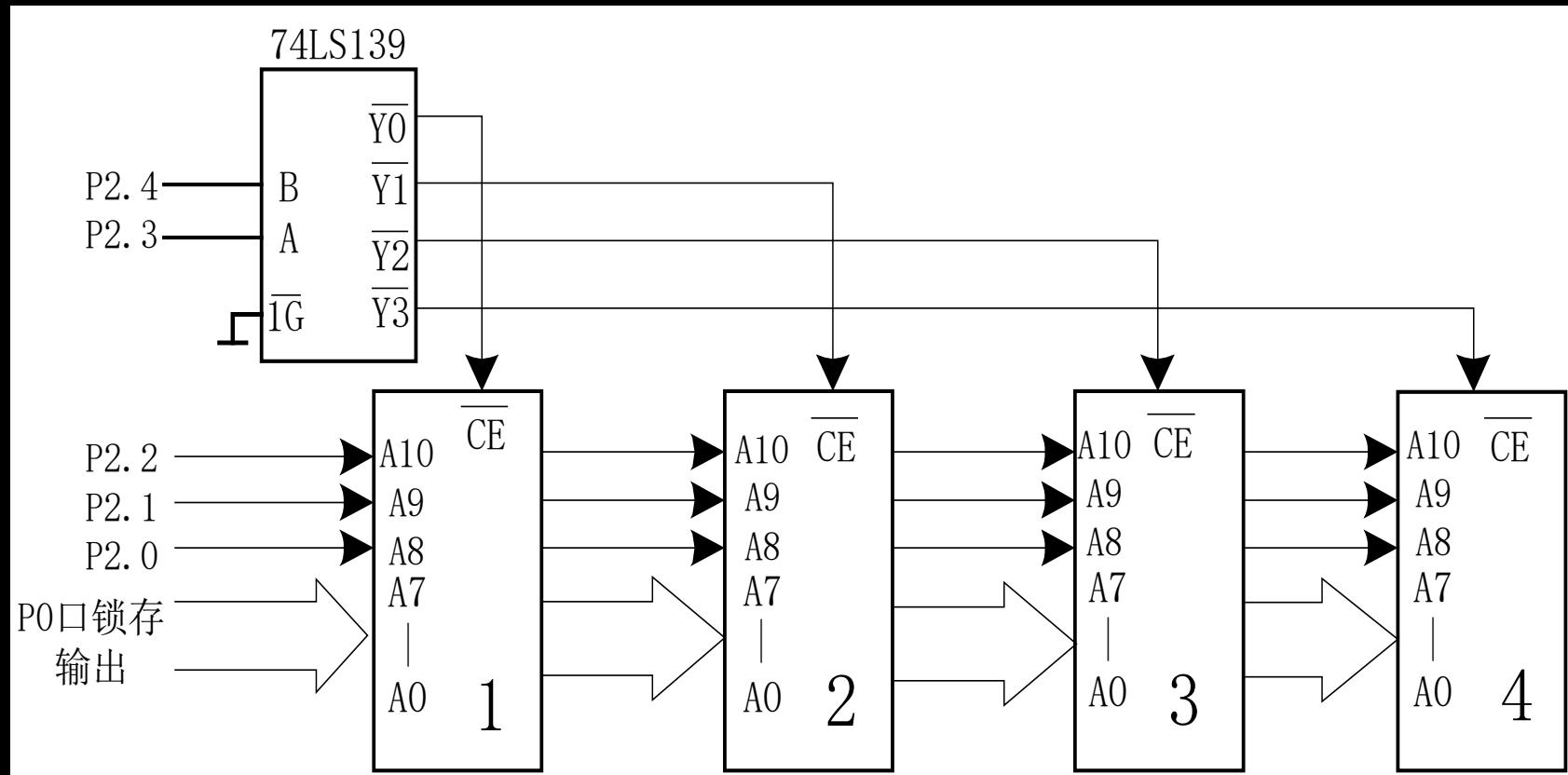
【例】现有 $2K \times 8$ 位存储器芯片，需扩展 $8K \times 8$ 位存储结构采用译码法进行扩展。

扩展8KB的存储器结构需2KB的存储器芯片4块。2K的存储器所用的地址线为 $A_0 \sim A_{10}$ 共11根地址线和片选信号与CPU的连接如下表所示。

## 80C51与存储器的线路连接表

80C51		存储器	
$P_0$ 口经锁存器锁存形成 $A_0 \sim A_7$		与 $A_0 \sim A_7$ 相连	
$P_{2.0}$ 、 $P_{2.1}$ 、 $P_{2.2}$		与 $A_8 \sim A_{10}$ 相连	
$P0$ 口		与 $D_0 \sim D_7$ 相连	
$P_{2.4}$	$P_{2.3}$	译码输出与存储器的片选信号连接	
0	0	/Y0	与存储器1的片选信号相连
0	1	/Y1	与存储器2的片选信号相连
1	0	/Y2	与存储器3的片选信号相连
1	1	/Y3	与存储器4的片选信号相连

$P_{2.3}$ 、 $P_{2.4}$ 作为二-四译码器的译码地址，译码输出作为扩展4个存储器芯片的片选信号， $P_{2.5}$ 、 $P_{2.6}$ 、 $P_{2.7}$ 悬空。扩展连线图如图所示。

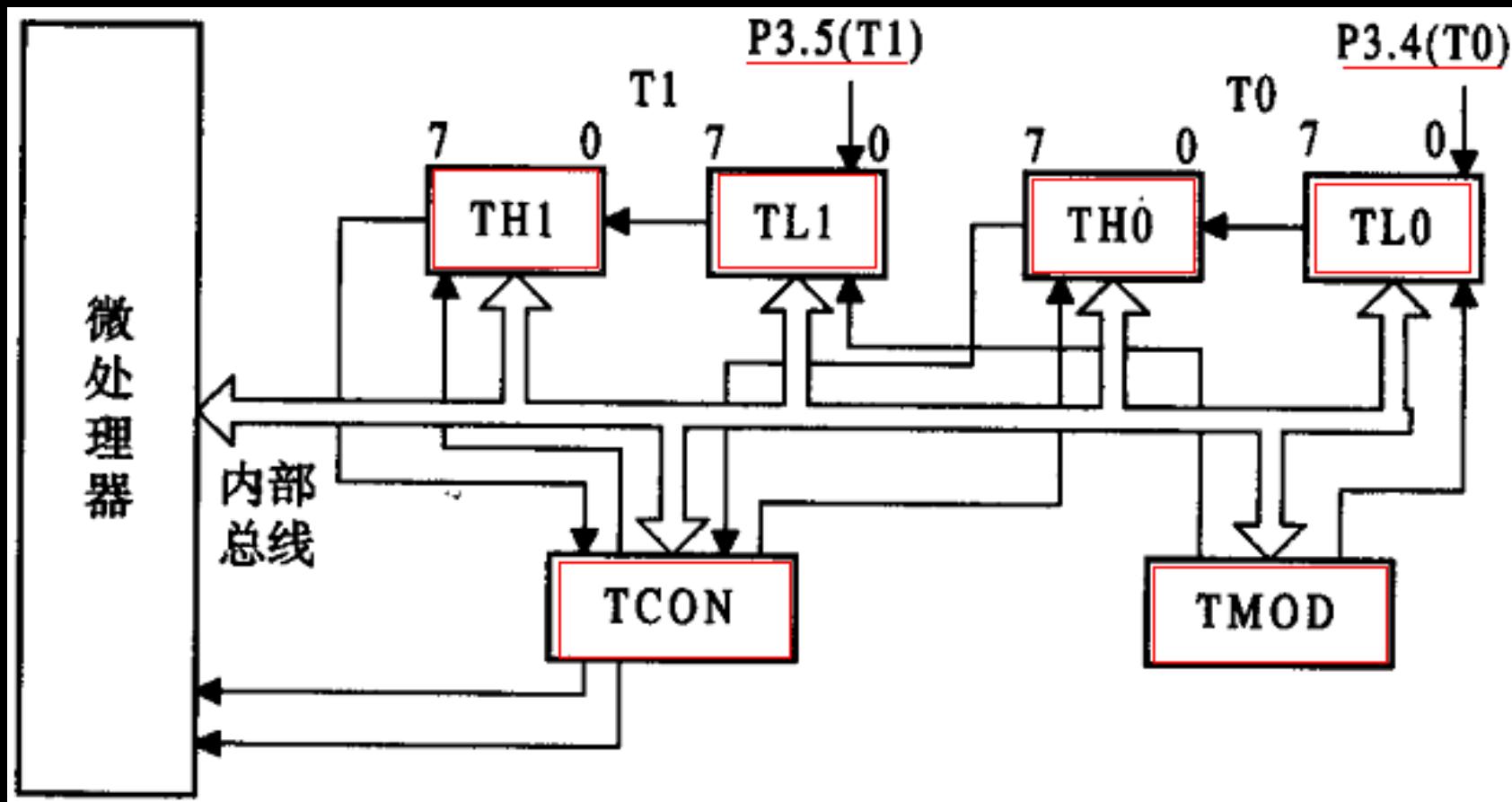


采用译码器扩展8KB存储器连线图

这样得到四个芯片的地址分配如下表所示。

译码方式地址分配表

	$P_{2.7}$	$P_{2.6}$	$P_{2.5}$	$P_{2.4}$	$P_{2.3}$	$P_{2.2} \dots P_0$	地址范围
芯片1	0	0	0	0	0	0 .... 0	0000H---
	0	0	0	0	0	1 .... 1	07FFH
芯片2	0	0	0	0	1	0 .... 0	0800H---
	0	0	0	0	1	1 .... 1	0FFFH
芯片3	0	0	0	1	0	0 .... 0	1000H---
	0	0	0	1	0	1 .... 1	17FFH
芯片4	0	0	0	1	1	0 .... 0	1800H—
	0	0	0	1	1	1 .... 1	1FFFH



定时器/计数器结构框图

M1	M0	方式	说 明
0	0	0	13 位定时器(TH的 8 位和TL的低 5 位)
0	1	1	16 位定时器/计数器
1	0	2	自动重装入初值的 8 位计数器
1	1	3	T0 分成两个独立的 8 位计数器, T1 在方式 3 时停止工作

# 串行通信

SM0	SM1	方式	功能说明	波特率
0	0	方式0	移位寄存器方式	$fosc/12$
0	1	方式1	8位异步串行通信	可变
1	0	方式2	9位异步串行通信	$fosc/64$ 或者 $fosc/32$
1	1	方式3	9位异步串行通信	可变

## D/A转换器接口

### ● D/A转换的一般工作原理:

常用电阻分压/分流来实现D/A转换。

D/A转换器将数字信息转换成与数值成正比的电压/电流。 $VO=D*VREF/2^N$ .有权电阻解码网络与 T型解码网络两种构建方法,又以T型解码网络最为常用。

### ● 权电阻解码网络

简单。但随着D/A转换的位数增加, 权电阻值跨度增大, 在集成电路中难于实现。

### ● T型解码网络

电阻数量大一倍。但电阻值归一化程度高,容易集成,精度高。应用最为普遍。



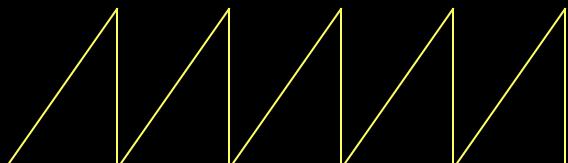
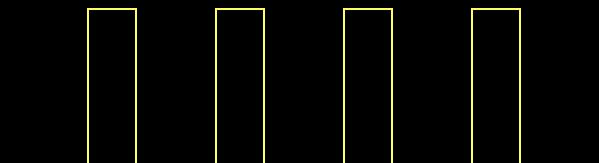
# DAC0832 的编程应用举例

## 例1 产生矩形波

```
LL:MOV A, #00H ;低电平
    MOV DPTR, #7FFFH
    MOVX @DPTR, A ;送转换
    LCALL DELAYL ;低宽度
    MOV A, #0FFH ;高电平
    MOVX @DPTR, A ;送转换
    LCALL DELAYH ;高宽度
    SJMP LL
```

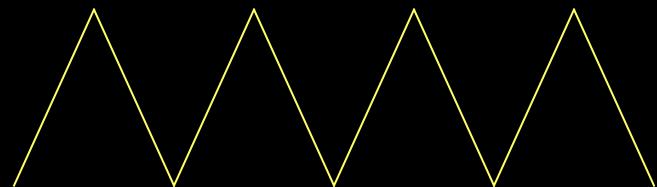
## 例2 产生锯齿波

```
MOV A, #00H ;起始值
    MOV DPTR, #7FFFH
MM: MOVX @DPTR, A ;送转换
    INC A
    NOP
    NOP
    NOP ;决定坡度
    SJMP MM
```



# ● DAC0832编程应用举例：例3产生三角波

```
MOV A, #00H
MOV DPTR, #7FFFH
SS1: MOVX @DPTR, A ;送转换
NOP
NOP
NOP
SS2: INC A      ;等速上升
CJNE A, #0FFH, SS1
SS3: DEC A
MOVX @DPTR, A
NOP
NOP
NOP      ;等速下降
CJNE A, #00H, SS3
SJMP SS1
```



三角波

同样的编程思路，若要产生  
如下的梯形波也很容易：



梯形波

## A/D转换器接口

**采样保持电路:**把一个时间连续的信号变换为时间离散的信号，并将采样信号保持一段时间。

**量化编码电路:**

**直接A/D转换器:**能将输入的模拟电压直接转换为输出的数字代码(无中间变量)。如逐次逼近型

**间接A/D转换器:**首先将输入的模拟信号转换成与之成正比的时间或频率，然后通过计数的方式转换成数字量输出。如双积分型

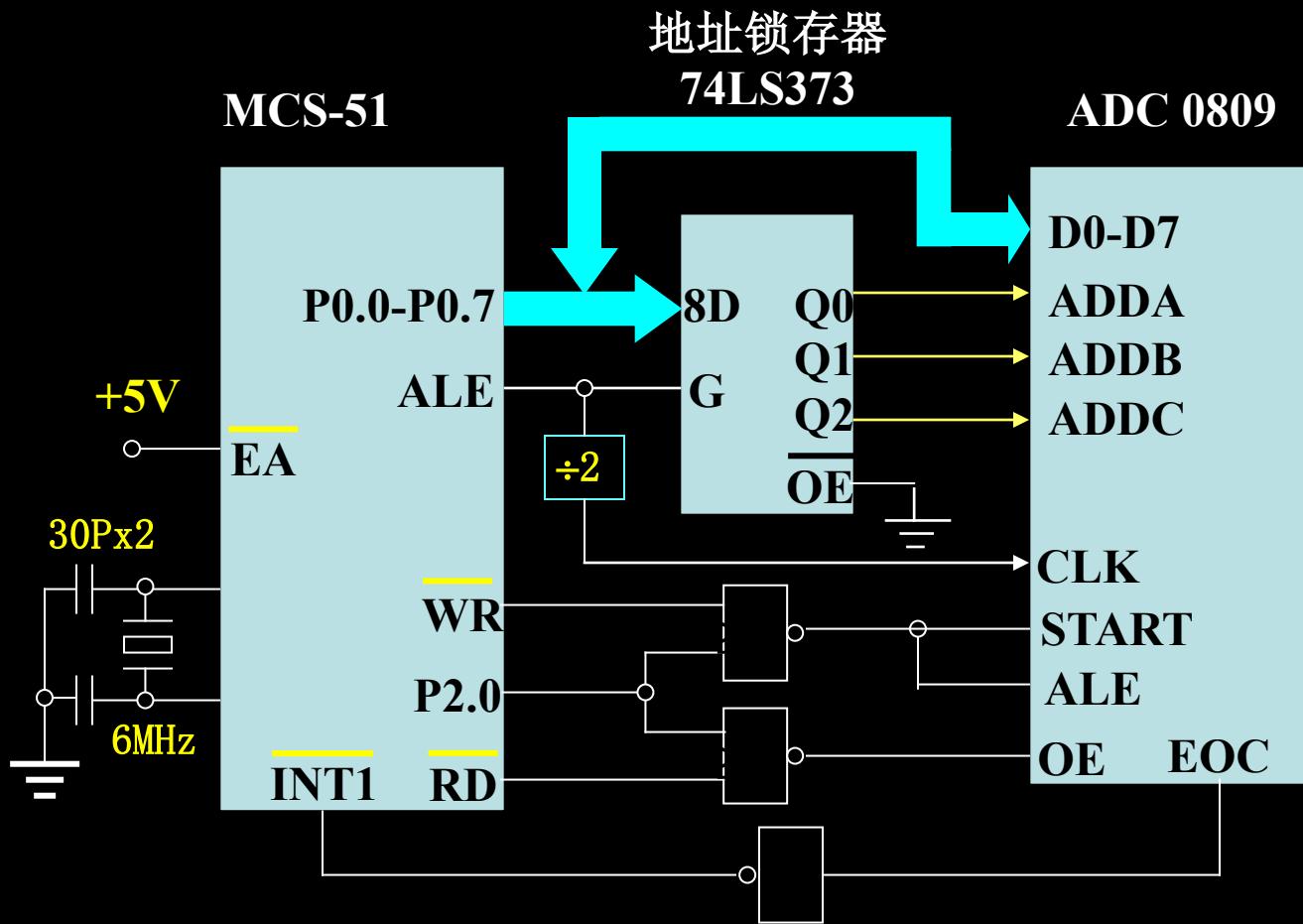
# ADC0809的转换过程

CLK	给SAR置位	SAR试探值	$U_O/(V)$	$U_I/(V)$	$U_I$ 与 $U_O$ 比较	SAR逼近值
1	$D_7=1$	10000000	2.56	4.34	$U_I > U_O$	10000000
2	$D_6=1$	11000000	3.84	4.34	$U_I > U_O$	11000000
3	$D_5=1$	11100000	4.48	4.34	$U_I < U_O$	11000000
4	$D_4=1$	11010000	4.16	4.34	$U_I > U_O$	11010000
5	$D_3=1$	11011000	4.32	4.34	$U_I > U_O$	11011000
6	$D_2=1$	11011100	4.40	4.34	$U_I < U_O$	11011000
7	$D_1=1$	11011010	4.36	4.34	$U_I < U_O$	11011000
8	$D_0=1$	11011001	4.34	4.34	$U_I = U_O$	11011001

$U_I > U_O$  保留该位;  $U_I < U_O$  丢弃该位; .....  
一直到  $U_I = U_O$  即为转换结果。



# ADC0809 与单片机的连接



口地址可以为：  
**0FE00**  
~  
**0FE07H**  
也可为：  
**0FEF8**  
~  
**0FEFFH**

P0口是数据/地址分时复用，要有地址锁存器74LS373。

ADC0809的ALE和START连在一起，可以在锁存通道地址的同时启动A/D转换。

P2.0分别和读写控制信号一起来控制对ADC0809的启动和读取A/D转换的结果。

EOC转换结束信号输出端经非门接INT1，可用中断方式进行工作。



## ADC0809延时方式进行数据采集

```
MOV  DPTR, #0FEF8H ; ADC 0口地址
MOVX @DPTR, A      ; 启动A/D转换
LCALL DELAY        ; 等待转换结束
MOVX A, @DPTR      ; 取转换结果
```

## ● ADC0809八路巡回中断式数据采集

ORG 0000H

AJMP MAIN

ORG 0013H ;外部中断1的中断矢量

AJMP INT

MAIN: MOV R0,#0A0H ;存结果的缓冲区:A0H-A7H

MOV R2,#08H ;待采集的通道数为 8

SETB IT1 ;选择下降沿触发中断

SETB EA ;开中断

SETB EX1 ;开外部中断1中断

MOV DPTR,# H ;通道0的地址

MOVX @DPTR, A ;启动转换。

HERE: SJMP HERE ;等待中断

# ● ADC0809八路巡回中断式数据采集

```
ORG 0000H
AJMP MAIN
ORG 0013H
AJMP INT
MAIN: MOV R0,#0A0H
      MOV R2,#08H
      SETB IT1
      SETB EA
      SETB EX1
      MOVDPTR,#0FEF8H
      MOVX @DPTR, A
HERE: SJMP HERE
```

```
INT:MOVX A, @DPTR ;读数据
      MOV @R0, A
      ;数据放进缓存单元
      INC R0 ;指向下一缓存
      INC DPTR ;指向下一通道
      DJNZ R2, RTN
      ;8 次未完就继续采集,
      ;已完就关中断、停采集
      CLR EA
      CLR EX1
      RETI
RTN:MOVX @DPTR,A;启动采集
HERE: SJMP HERE
```

# 循环程序

循环程序一般由：

- ◆ 初始化部分
- ◆ 循环体部分
- ◆ 循环控制部分
- ◆ 结束部分

一般有两种结构：

- ◆ 先进入处理部分，再控制循环
  - 至少执行一次循环体
- ◆ 先控制循环，再进入处理部分
  - 循环体是否执行，取决于判断结果。

**例：**将内部RAM中起始地址为data的数据串送到外部RAM中起始地址为buffer的存储区域中，直到发现‘\$’字符，传送停止——循环次数事先不知道先判断，后执行。

```
MOV  R0, #data
MOV  DPTR, #buffer
L00P1:MOV  A, @R0
        CJNE A, #24H, L00P2      ;判断是否为 $ 字符
        SJMP L00P3                ;是， 转结束
L00P2:MOVX @DPTR, A            ;不是， 传送数据
        INC  R0
        INC  DPTR
        SJMP L00P1                ; 传送下一数据
L00P3:END
```

## 例：排序程序设计

若以30H为首地址的内部RAM中有7个连续存放的数据，按由小到大的次序排好后存入这7个单元。

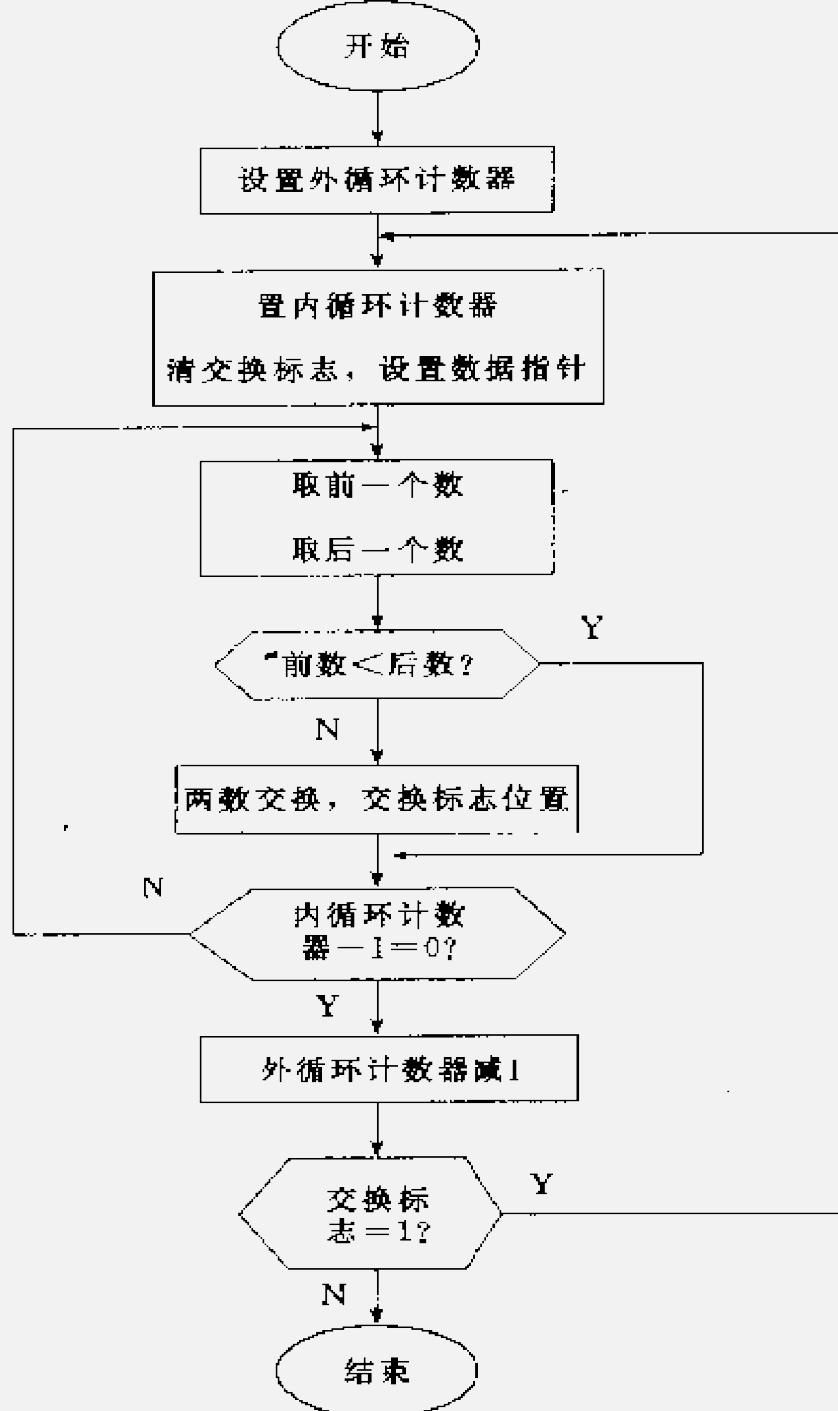
冒泡法排序基本思想：

是一种**相邻数互换**的排序方法。执行时从前向后依次将相邻两单元的数进行比较，即第一个数与第二个数、第二个数与第三个数……进行比较，如果符合从小到大的顺序则不变动，否则将它们交换位置。如此反复比较和交换，使小数向前移、大数向后移，直到数列排好为止。

表 4-1 数列分选过程

原数列	第一次分选结果	第二次分选结果	第三次分选结果	第四次分选结果
0	0	0	0	0
16	3	3	3	3
3	16	16	16	13
94	25	25	13	16
25	36	13	25	25
36	13	36	36	36
13	94	94	94	94

为了加快程序运行速度，编程时要考虑以下两点：一是要比较交换的数每次少一个，即每次冒泡所需进行的比较次数都递减1；二是只要不发生交换就表示已按大小顺序排列好了，可以结束比较。



**R0:数据首地址**  
**F0:交换标志位**  
**R7:外循环初值**  
**R6:内循环初值**

程序的关键在于每两个数进行比较，如果有交换发生，交换标志位要置1。

	ORG	2000H	
	MOV	R7, #06H	; 外循环计数器赋值
LP1:	MOV	R0, #30H	; R0 指向数据首址
	CLR	F0	; 交换标志清 0
	MOV	A, R7	
	MOV	R6, A	; 内循环计数器赋值
LP2:	MOV	A, @R0	; 取前一个数
	MOV	20H, A	; 存前数
	INC	R0	
	MOV	21H, @R0	; 取后一个数
	CLR	C	; CY 清 0
	SUBB	A, @R0	; 前数减后数
	JC	NEXT	; 前数小于后数, 不交换
	MOV	@R0, 20H	
	DEC	R0	
	MOV	@R0, 21H	; 前、后两数交换位置
	INC	R0	; 恢复数据指针
	SETB	F0	; 置互换标志
NEXT:	DJNZ	R6, LP2	; 未查完一遍, 进行下一次比较
	DEC	R7	; 修改内循环次数
	JB	F0, LP1	; 返回, 进行下一轮冒泡
	SJMP	\$	; 结束
	END		

祝大家取得好成绩！