目　　录

前言　面向应用的嵌入式系统在我国方兴未艾

微控制器，也就是单片机（MCU），在 80 年代进入中国。由于微控制器容易学、容易用，倍受青睐。这种把中央处理器、存储器、外设器件及 I/O 做在同一块芯片上的器件总是作为应用系统中的控制部件使用。现在，做在微控制器芯片上的外设部件越来越多，功能不断增强。针对具体的应用，利用微控制器可以设计出十分复杂的系统，这种系统称作嵌入式系统。在杭州召开的'99 全国单片微机学术交流会暨多国单片微机产品展览会上，许多专家呼吁要提高对嵌入式系统的认识。

目前，在全世界，嵌入式系统带来的工业年产值已超过 1 万亿美元。预计在美国，单是使用嵌入式电脑的全数字电视产品每年将产生 1500 亿美元的新市场。美国未来学家尼葛洛庞帝曾预言，四、五年后，嵌入式智能工具将是继 PC 和因特网后最伟大的发明。就目前国内微控制器的应用状况，全国微机单片机学会理事长陈章龙教授说，从整体来讲，在中国，微控制器的应用水平还不高，主要是用 8 位微控制器，用量也不大，绝大多数是用于 IC 卡设备等仪器仪表和控制领域中。嵌入式系统的核心部件是各种类型的嵌入式处理器，据不完全统计，全世界嵌入式处理器的品种已经过千，流行的结构有 30 多种，其中以我们熟悉的 PIC 系列结构的产品为最多。据中国单片机公共实验室高级工程师吕京建介绍，嵌入式处理器分为两大类，一类是以通用计算机中的 CPU 为基础的嵌入式微控制器，另一类是微控制器。与微处理器相比，微控制器具有单片化、体积小、功耗低、可靠性高、芯片上的个设资源丰富等特点，目前已成为嵌入式系统的主流器件。嵌入式微处理器的软件是实现嵌入式系统功能的关键，为了提高执行速度和系统的可靠性，嵌入式系统中的软件一般都固化在存储器芯片或微控制器中。

嵌入式系统是面向应用的，因此它可以应用在现代化工业的各个领域，如：航天、航空、军事、家用消费商品、仪器仪表、各种控制系统及 3C 系统。尤其在国内主要应用于家电消费类产品、通信和计算机外设等。

而福州高奇电子科技有限公司正在对 MCU 的广泛应用起着强大的推动作用。高奇电子科技有限公司创办于一九九三年十月，是一家专业的半导体集成电路授权代理商和专业集成电路应用、设计公司。目前代理、销售多家著名半导体厂商的产品，如单片机、E^2PROM、保安器件、电压检测器、LCD/VFD 驱动器、电话来电识别（Caller ID）以及交换机用的 Switch、Codec 芯片等等；同时，设有专业的研发部门，已经研制了一系列电子产品整机方案，这些方案包括完整的软硬件设计资料及样机，可提供给整机厂商直接采用生产。公司的工程部门还可以为客户量身定做，增加和删改功能以体现客户的产品特色。高奇公司坚持“以专业的态度和水准，供优质产品、创名牌服务”的经营理念，将全部资源专注于半导体 IC 的应用设计、行业市场的专用 IC（ASIC）设计以及 IC 市场营销，并将不断开拓出电子产品新领域，并缩短研发时间，使产品与时代同步。

下面就介绍一种简单的 PIC 单片机系列。

PIC16C5X 是美国 **Microchip** 推出的世界上第一种 8 脚的超小型单片机系列，体积虽小却集成了很多功能特点，节省了很多别的单片机应用中必须外接的元器件，所以它是目前最便宜的 8 位 OTP 单片机。加上它小巧，所以它可以嵌入几乎任何一种电子产品中，特别是对于那些便携式电子产品，如各种 IC 卡、电子身份牌、微型录音机、照像机、充电器、计时器、智能传感器、软件狗、灯光调节器、电子开关、儿童玩具等等，都已得到广泛应用。请详阅以下各章的内容（注：有关 PIC 芯片更详细的数据资料，请到以下网站查询或与我们联系： http://www.microchip.com）。

第一章　PIC16C5X 硬件结构

PIC16C5X 系列单片机是 8 位单片机，CMOS 工艺制造。本章将详细介绍其内部结构、寄存器组、I/O、时序、振荡形式等等。

§1.1　PIC16C5X 主要功能特点

• 采用精简指令集（RISC），仅 33 条指令。指令字长 12 位，全部指令都是单字节指令。除涉及 PC 值改变的指令外（如跳转指令等），其余指令都是单周期指令。

• 工作频率为 DC～20MHZ。

• 系统为哈佛结构。数据总线和指令总线各自独立分开，数据总线宽度为 8 位，指令总线宽度为 12 位。

• 内部程序存储器（ROM）从 384～2K 字节不等。内部寄存器组（RAM）有 25～72 个。

• 7 个特殊功能寄存器。

• 2 级子程序堆栈。

• 工作电源

- 商用级：2.5V～6.25V
- 工业级：2.5V～6.25V
- 军工级：2.5V～6.0V

• 内部自振式看门狗（WDT）

• 低功耗模式（Sleep），耗电小于 10uA。

• 内部复位电路

• 内带一个 8 位定时器/计数器（RTCC）

• 具备保密位。保密熔丝可在程序烧写时选择将其熔断，则程序不能被读出拷贝。

• 提供四种可选振荡方式

- 低成本的阻容（RC）振荡—RC
- 标准晶体/ 陶瓷振荡—XT
- 高速晶体/ 陶瓷振荡—HS
- 低功耗，低频晶体振荡—LP

• 12～20 根双向可独立编程 I/O 口。每根 I/O 口都可由程序来编程决定其输入/输出方向。

• 低功耗

<2mA　　@5V，4MHz
<15uA　　@3V，32KHz
<3uA　　低功耗模式　@3V，0°C～70°C

§1.2　PIC16C5X 型号介绍及引脚介绍

PIC16C5X 有五种型号，见下表：

型号	管脚	I/O	RAM	EPROM	振荡	最短指令周期	看门狗
16C54	18	12	32×8	512×12	DC～20M	200 nS	有
16C55	28	20	32×8	512×12	DC～20M	200 nS	有
16C56	18	12	32×8	1024×12	DC～20M	200 nS	有
16C57	28	20	80×8	2048×12	DC～20M	200 nS	有
16C58	18	12	80×8	2048×12	DC～20M	200 nS	有

表 1.1　PIC16C5X 型号表

PIC16C5X 管脚图如下：

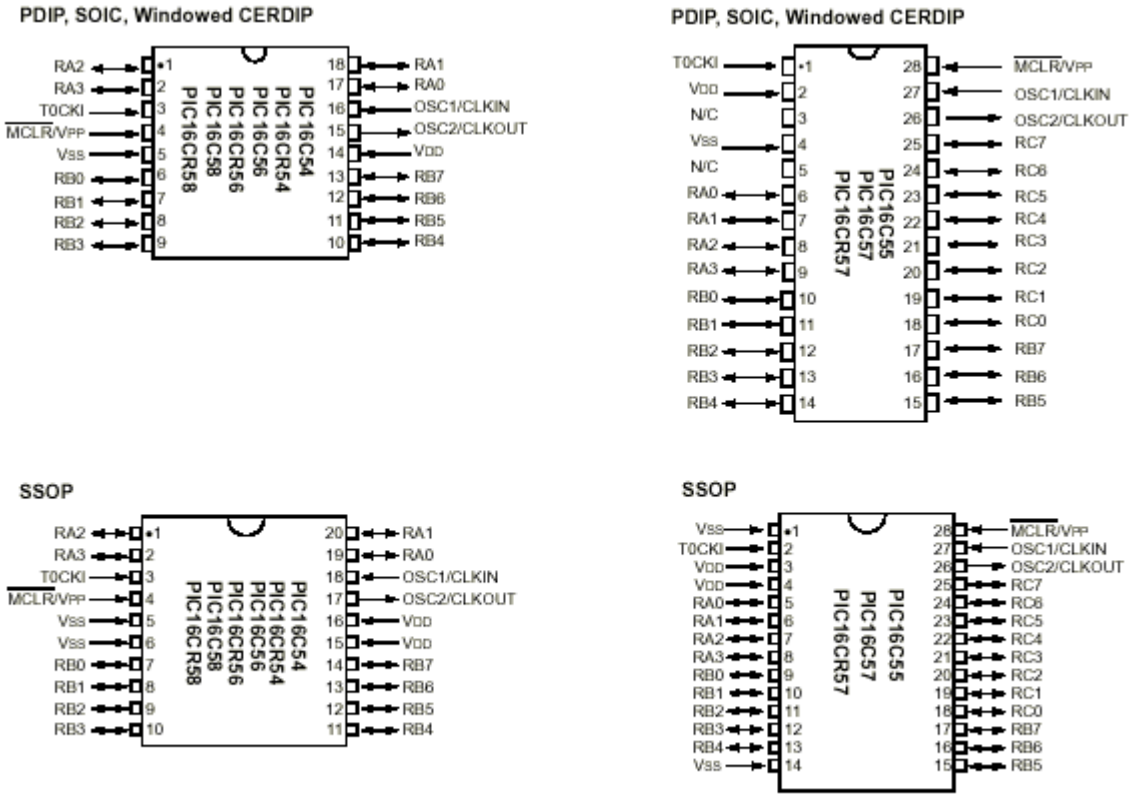


图 1.1　PIC16C5X 管脚图

表 1.2 描述了各引脚的功能：

引　脚　名	功　　能
RA0～RA3	I/O 口，双向可编程
RB0～RB7	I/O 口，双向可编程
RC0～RC7	I/O 口，双向可编程
RTCC	实时定时器/计数器
MCLR	复位脚，低电平有效
OSC1	振荡（输入）
OSC2	振荡（输出）
VDD	电源
VSS	地
N/C	未用

表　1.2　PIC16C5X　引脚功能描述

注：RTCC 设置成内部定时器时（由程序设定），这时应将 RTCC 端接 VSS 或 VDD，以避免干扰。采用 RC 振荡时，OSC2 端输出一 OSC1 的 4 分频信号。

§1.3　PIC16C5X　内部结构

PIC16C5X 在一个芯片上集成了一个 8 位算术逻辑单元 ALU 和工作寄存器（W）；384～2K 的 12 位程序存储器（ROM）；32～80 个 8 位数据寄存器（RAM）；12～20 个 I/ O 口端；8 位计数器及预分频器；时钟、复位、及看门狗计数器等。内部结构如图 1.2 所示：

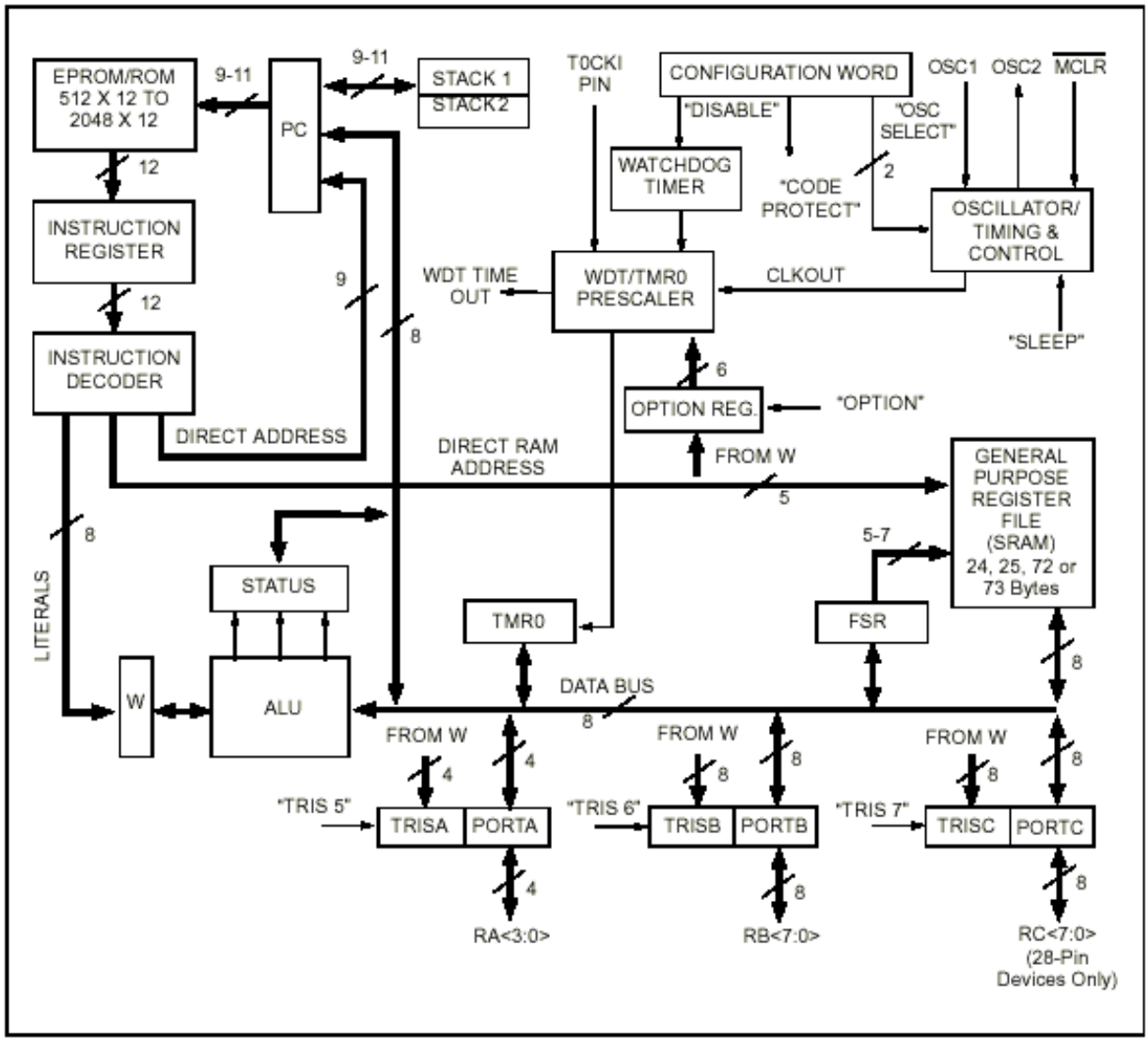


图　1.2　PIC16C5X　内部结构

从图中可以看到，PIC16C5X 有个特点，就是把数据存储器 RAM 当作寄存器来寻址使用以方便编程。寄存器组按功能分成二部分，即特殊寄存器组和通用寄存器组。特殊寄存器组包括实时时钟计数器 RTCC，程序计数器 PC，状态寄存器 Status，I/O 口寄存器以及存储体选择寄存器 FSR。这些寄存器稍后我们还要详细论述。

PIC 总线结构采取数据线（8 位）和指令线（12 位）独立分离的哈佛（Harvard）结构。这样可使单片机的指令速度得到提高。当一条指令在 ALU 中执行时，下一条指令已经被取出放到指令寄存器等待执行了。算术逻辑单元 ALU 和工作寄存器（W）承担算术逻辑操作任务。

PIC16C5X 提供二级堆栈（Stack），所以子程序调用只有二层。使用时一定要注意这点，否则程序运行将失去控制。

§1.4　程序存储器及堆栈

PIC16C5X 内部有 384～2K 的只读程序存贮器，下面论述其结构和堆栈。

§1.4.1　程序存储器结构

PIC16C5X 程序存储器结构如图 1.3 所示：

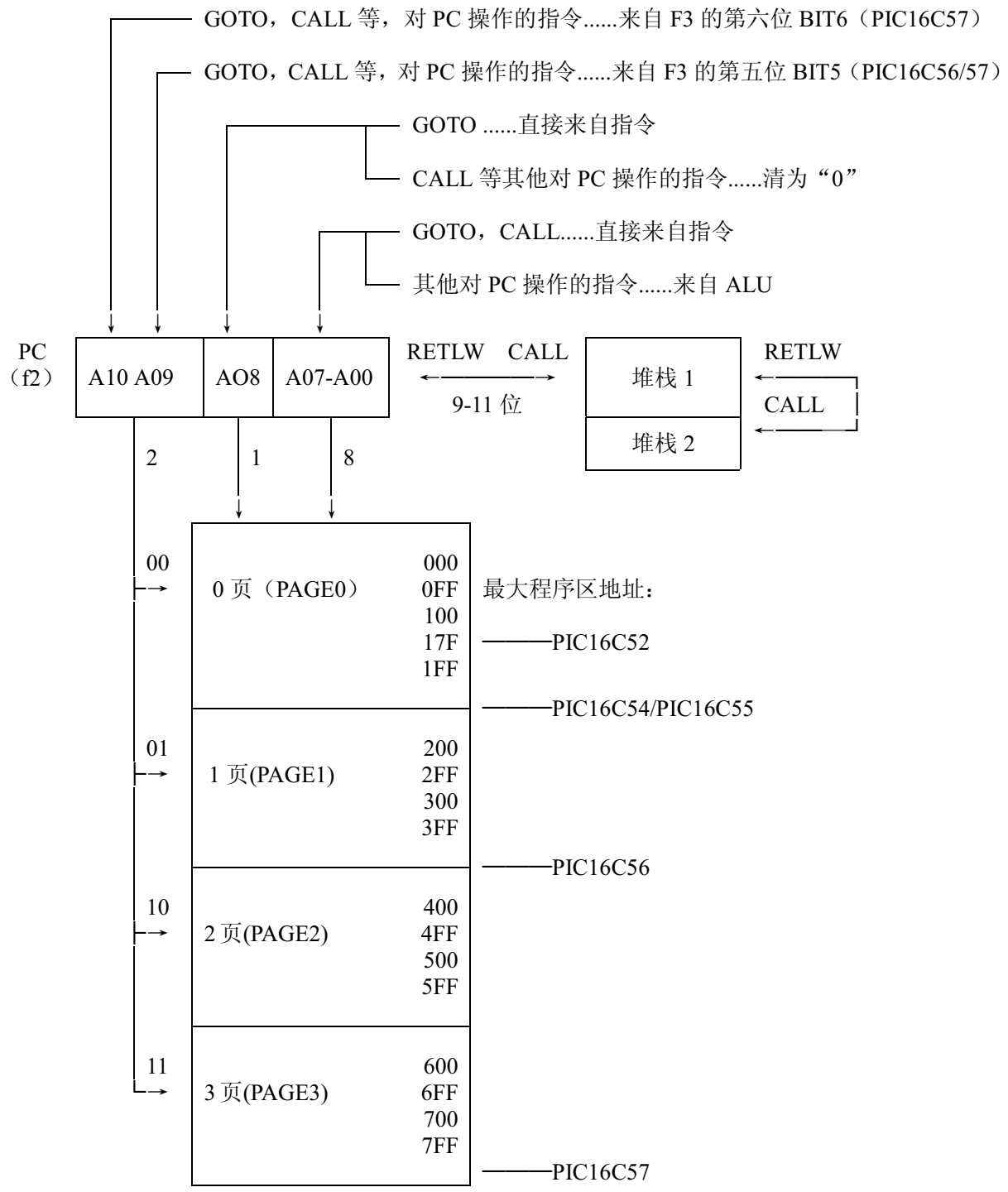


图 1.3 PIC16C5X 程序存贮器结构

从上图可看出，PIC 程序存储器采用分页结构，每页长 0.5K。因此对于 PIC16C52 程序存储器在 1 页之内，而对于 PIC16C54 和 PIC15C55 程序存储器容量为 1 页，PIC16C56 和 PIC16C57 的容量则分别为 2 页和 4 页。页面地址由状态寄存器 f3 的第 5 位和第 6 位（PA0、PA1）确定。程序转移时，在本页内可直接进行；在需跨页跳转时（GOTO、CALL 指令），则必须根据将要跳转去的页面，把 f3 中的 PA0、PA1 位置成相应的值。具体请参考 f3 寄存器描述及§2.7.2 程序设计基础的程序技巧例子。

§1.4.2 堆栈

PIC16C5X 设有二层堆栈，堆栈 1 和堆栈 2，供子程序调用。涉及堆栈操作的指令有二条。

1、CALL——在主程序中第一次执行 CALL 指令时，将 PC 值加 1 后推入堆栈 1，堆栈 1 原有的内容则被推入堆栈 2 中。这时子程序中还可再做一次子程序嵌套，即再执行一次 CALL 指令。如果子程序调用多于二层时，堆栈中只存放最近的二个返回地址。

当执行一条 CALL 指令时，状态寄存器 f3 中的将页面寻址位 PA1、PA0 将被置入到 PC 的最高二位（第 11 位和第 10 位），而 PC 的第 9 位总是被置为 0，如图 1.3 所示。所以这时 PC 值将是

11	10	9	8	7	6	5	4	3	2	1	
PA1	PA0	0	*	*	*	*	*	**	*	*	PC

这意味着在 PIC16C5X 中，子程序起始地址只能放在每个程序存储页面的上半页，即低地址的那一半（000－0FF、200－2FF、400－4FF、600－6FF）。注意，这里指的是子程序的起始地址，即子程序头。而子程序体是可以延伸到下半页去的。

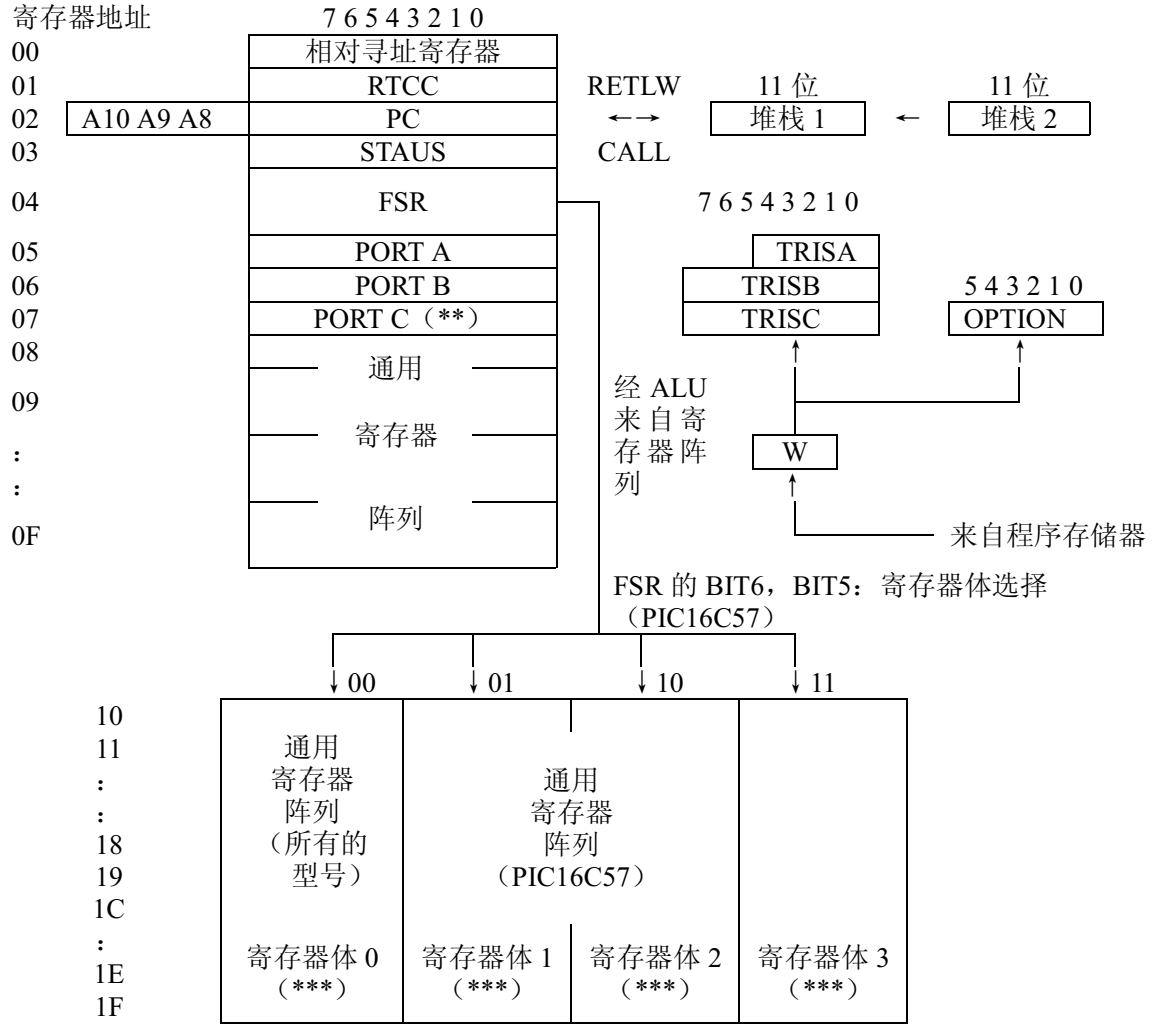
对于 PIC16C56 和 PIC16C57，由于程序空间分别为 1K 和 2K，可能存在跨页面子程序调用。所以调用子程序前必须先把 f3 中的 PA0、PA1 设置成该子程序所在的页面地址。返回后再将其恢复成当前的页面值。当然如果这时子程序是在同一页面，则可省去这一过程。

2、RETLW——该指令把栈顶（堆栈 1）的值写入 PC，同时还把堆栈 2 的值拷贝到堆栈 1 去。子程序总是返回到调用它时所在的地方，不管它是处在什么页面，也不管 f3 寄存器中的 PA0、PA1 这时是指在什么页面。但是执行 RETLW（子程序返回）指令并不会改变 f3 中 PA0、PA1 的值，所以当你从一次跨页面的子程序调用返回时，不要忘了恢复 f3 中的原先的 PA0 和 PA1 值。请参考上面关于 CALL 指令的叙述。

由于堆栈和 PC 的宽度是相同的，所以你可以在程序的任何地方执行一条 CALL 指令来调用子程序。但是对于跨页面的调用，你要小心处理 f3 中的页面地址位 PA0 和 PA1，请参考第二章指令详解中的 CALL 实例。

§1.5 数据存储器

PIC16C5X 把数据存储器 RAM 都当作寄存器来使用以使寻址简单明洁，它们功能上可分为操作寄存器、I/O 寄存器、通用寄存器和特殊功用寄存器。它们的组织结构如图 1.4 所示：这些寄存器用代号 F0～F79 来表示。F0～F4 是操作寄存器，F5－F7 是 I /O 寄存器，其余为通用寄存器。特殊功用寄存器地址对用户不透明。



（*）　非实际存在的执行寄存器。

（**）　F7 对于 PIC16C52/54/56 是通用寄存器。

（***）　存储器体 0 各型号皆有，存储器体 1-3 仅 PIC16C57 有。

图 1.4　PIC16C5X 寄存器结构

§1.5.1　操作寄存器

1、F0　间址寄存器

寻址 F0 实际上意味着间址寻址。实际地址为寄存器选择寄存器 F4 的内容。

例：
```
MOVLW   10
MOVWF   f4          ；10→f4
MOVLW   55
MOVWF   f0          ；55→f10
```

2、F1　实时时钟/计数寄存器（RTCC）

此寄存器是一个 8 位计数器。和其他寄存器一样可由程序进行读写操作。它用于对外加在 RTCC 引脚上的脉冲计数，或对内部时钟计数（起定时器作用）。

RTCC 及其相关电路如图 1.5 所示。

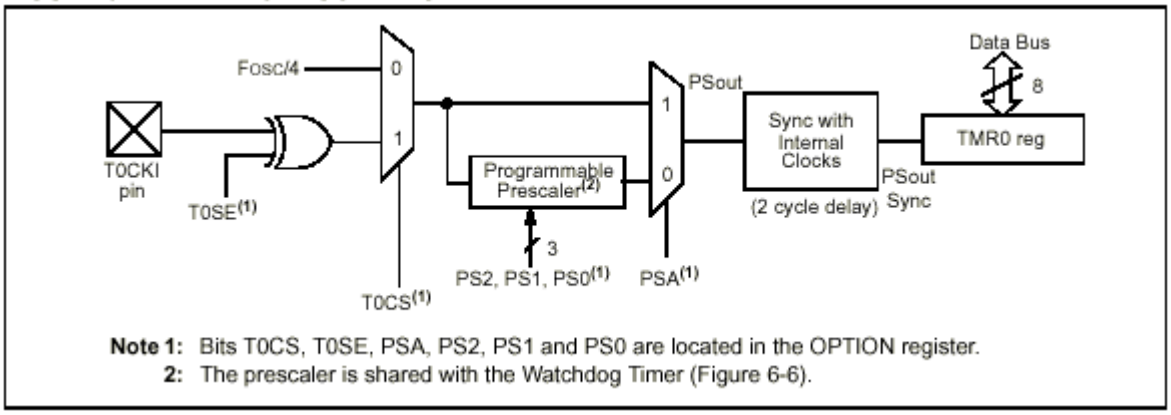


图 1.5　RTCC　方块图

上图中可看出 RTCC 工作状态由 OPTION 寄存器控制（参见 § 1.5.4），其中 OPTION 寄存器的 RTS 位用来选择 RTCC 的计数信号源，当 RTS 为“1”时，信号源为内部时钟，RTS 为“0”时，信号源为来自 RTCC 引脚的外部信号。OPTION 寄存器的 PSA 位控制预分频器（Prescaler）分配对象，当 PSA 位为“1”，8 位可编程预分配给 RTCC，即外部或内部信号经过预分频器分频后再输出给 RTCC。预分频器的分频比率由 OPTION 内的 PS0～PS2 决定。这时涉及写 f1（RTCC）寄存器的指令均同时将预分频器清零。但要注意 OPTION 寄存器内容仍保持不变，即分配对象、分频比率等均不变。OPTION 的 RTE 位用于选择外部计数脉冲触发沿。当 RTE 为“1”时为下降沿触发，为“0”时为上升沿触发。

RTCC 计数器采用递增方式计数，当计数至 FFH 时，在下一个计数发生后，将自动复零，重新开始计数，以此一直循环下去。RTCC 对其输入脉冲信号的响应延迟时间为 2 个机器周期，不论输入脉冲是内部时钟、外部信号或是预分频器的输出。响应时序见图 1.6。

RTCC 对外部信号的采样周期为 2 个振荡周期。因此当不用预分频器时，外加在 RTCC 引脚上的脉冲宽度不得小于 2 个振荡周期，即 1/2 指令周期。同理，当使用预分频器时，预分频器的输出脉冲周期不得小于指令周期，因此预分频器最大输入频率可达 N.fosc/4，N 为预分频器的分频比，但不得大于 50MHZ。

当 RTCC 使用内部时钟信号时，如果没有预分频器，则 RTCC 值随指令节拍增 1。当一个值写入 RTCC

时，接下来的二个指令节拍 RTCC 的值不会改变，从第三个指令节拍才开始递增，见下图。

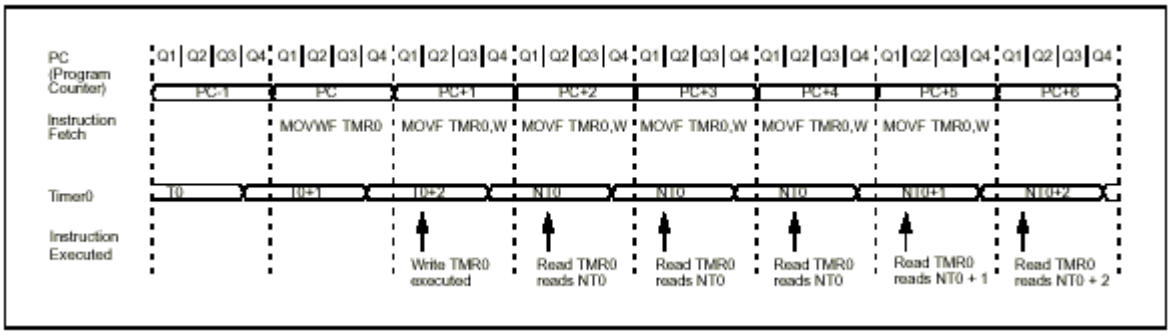


图 1.6A　RTCC 时序图:内部时钟/无预分频器

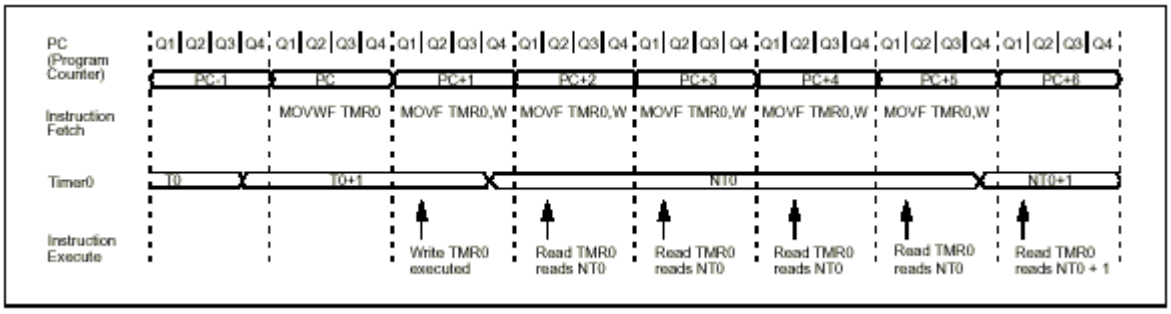


图 1.6B　RTCC 时序图:内部时钟/预分频比 1:2

应注意的是尽管 PIC 对外部加于 RTCC 信号端上的信号宽度没有很严格的要求，但是如果高电平或低电平的维持时间太短，也有可能使 RTCC 检测不到这个信号。一般要求信号宽度要大于是 10nS。

3、F2　程序计数器（PC）

程序计数器 PC 可寻址最多 2K 的程序存储器。表 1.3 列出了 PIC16C5X 各种型号的 PC 长度和堆栈的长度。

型号	PC 宽度（位）	堆栈宽度（位）
PIC16C54/55	9	9
PIC16C56	10	10
PIC16C57/58	11	11

表 1.3　　PC 及堆栈的宽度

单片机一复位（RESET），F2 的值全置为“1”。除非执行地址跳转指令，否则当执行一条指令后，F2（PC）值会动加 1 指向下一条指令。

下面这些指令可能改变 PC 的值：

a、“GOTO”指令。它可以直接写（改变）PC 的低 9 位。对于 PIC16C56/57/58，状态寄存器 F3 的 PA1、PAO 两位将置入 PC 的最高二位。所示“GOTO”指令可以跳转到程序存储器的任何地方。

b、“CALL”指令。它可以直接写 PC 低 8 位，同时将 PC 的第 9 位清零。对于 PIC16C56/57/58，状态寄存器 F3 的 PA1、PAO 两位将置入 PC 的最高二位（第 10、11 位）。

c、“RETLW”指令。它把栈项（堆栈 1）的值写入 PC。

d、“MOVWF　F2”指令。它把 W 寄存器的内容置入 PC。

e、“ADDWF　F2”指令。它把 PC 值加 1 后再和 W 寄存器的值相加，结果写入 PC。

在以上 b、d 和 e 中，PC 的第 9 位总是被清为零。所以用这三条指令来产生程序跳转时，要把子程序或分支程序放在每页的上部地址（分别为 000-0FF、200-2FF、400-4FF、600-6FF）。

4、F3　状态寄存器（STATUS）

如图 1.7 所示 F3 包含了 ALU 的算术状态、RESET 状态、程序存储器页面地址等。F3 中除 PD 和 TO 两位外，其他的位都可由指令来设置或清零。注意，当你执行一条欲改变 F3 寄存器的指令后，F3 中的情况可能出乎你的意料。

例：CLRF　F3　　；清 F3 为零

你得到的结果是 F3=000UU100（U 为未变）而不是想像中的全零。UU 两位是 PD 和 TO，它们维持

不变，而 2 位由于清零操作被置成“1”。所以如果你要想改变 F3 的内容，建议你使用 BCF、BSF 和 MOVWF 这三条指令，因为它们的执行不影响其他状态位。

例：MOVLW 0 ；0→W
MOVWF F3 ；把 F3 除 PD 和 TO 以外的位全部清零，则你可得到 F3=000UU000。

有关各条指令对状态位的影响请看第二章介绍。

在加法运算（ADDWF）时，C 是进位位。在减法运算（SUBWF）时，C 是借位的反（Borrow）。

例：CLRF F10 ；F10=0
MOVLW 1 ；1→W
SUBWF F10 ；F10-W=0-1=FFH→F10
C=0:运算结果为负

例：MOVLW 1 ；1→W
MOVWF F10 ；F10=1
CLRW ；W=0
SUBWF F10 ；F10-W=1-0=1→F10
C=1:运算结果为正

复位后的状态位：
* PA2，PA1，PA0 清为“0”；
* TO，PD 变化见表 1.5；
* Z，DC，C 在上电复位后处未知状态，在别的复位后则保持不变。

PA2	PA1	PA0	TO	PD	Z	DC	C

进位标志：

ADDWF 及 SUBWF 指令执行后，若 MSB 有进位则置 C 为 1。RRF/RLF 指令可与 C 一起移位操作。

辅助进位标志：

当进行加法或减法操作而产生由低 4 位数向高 4 位数进位时，被置 1。

零标志：

算术或逻辑运算结果为零时，置 1。

低功耗位：

系统上电或执行 CLRWDT 指令后，PD=1。执行 SLEEP 指令后，PD=0。

超时位：

系统上电和执行 CLRWDT 与 SLEEP 时，TO=1，WDT 溢出时，TO=0。

PIC16C52/54/C55：二位通用读写位

PIC16C56： PA0 为页选择位
0=PAGE0（000-1FF）
1=PAGE1（200-3FF）
PA1 为通用读写位

PIC16C57/58：二位页面选择位
00—PAGE0（000～1FF）
01—PAGE1（200～3FF）
10—PAGE2（400～5FF）
11—PAGE3（600～7FF）

保留位

图 1.7 状态寄存器结构

PD 和 TO 两位可用来判断 RESET 的原因。例如判断 RESET 是由芯片上电引起的，或是由看门狗 WDT 计时溢出引起的，或是复位端加低电平引起的，或是由 WDT 唤醒 SLEEP 引起的。

表 1.4 列出了影响 TO、PD 位的事件。表 1.5 列出了在各种 RESET 后的 TO、PD 位状态。

事　　件	TO	PD	注
电源上电	1	1	
WDT 溢出	O	X	不影响 PD
SLEEP 指令	1	O	
CLRWDT 指令	1	1	
SLEEP	1	O	

表 1.4 影响 PD/TO 的事件

TO	PD	RESET 产生的原因
0	0	WDT 溢时唤醒 SLEEP
0	1	WDT 溢时（非在 SLEEP 状态）
1	0	MCLR 端加低电平唤醒 SLEEP
1	1	电源上电
X	X	MCLR 端加低电平（非在 SLEEP 状态时）

表 1.5 RESET 后的 PD，TO 状态

判断 RESET 从何处引起有时是很必要的。例如在对系统初始化时，经常需判断这次复位是否是上电引起的。如果不是上电复位，则不再进行初始化。

例：（设本例中 WDT disabled）

```
                :
                :
                :
          ┌──  BTFSC      F3，PD        ；测 F3 的 PD 位是否为 1
PD=0      │    CALL       INIT          ；PD=1，复位由上电引起
跳过初    │    :                        ；PD=0，非上电复位，不做初始化工作
始化      └──→ :
                :
```

页面选择位 PA1、PA0 的作用前面已描述过，RESET 时清 PA0-PA2 位为零，所以复位后程序区页面自动选择在 0 页。

5、F4 寄存器选择寄存器（FSR）

a、 PIC16C52/54/55/56

F4 的 0－4 位在间接寻址中用来选择 32 个数据寄存器。5－7 位为只读位，并恒为 1。请参考 F0 寄存器描述。

b、PIC16C57/58

FSR<6:5>位用来选择当前数据寄存器体（Bank）。PIC16C57 有 80 个数据寄存器，如图 1.4 所示。80 个寄存器分为 4 个体（Bank0～Bank3），每个体的低 16 个寄存器的物理位置是相同的（参考§1.5.3 通用寄存器的描述）。当 FSR 的第 4 位为“1”时，则要根据 FSR<6:5>位来选择某个寄存器体中的某一个高 16 的寄存器。

FSR<6:5>位		寄存器体	物理地址
0	0	Banko	10H-1FH
0	1	Bank1	30H-3FH
1	0	Bank2	50H-5FH
1	1	Bank3	70H-7FH

表 1.6 寄存器体高 16 寄存器地址

注意：当芯片上电复位时，FSR<6:5>是不定的，所以它可能指向任何一个 Bank。而其他复位则保持原来的值不变。

具体程序技巧请参阅§2.7.2 程序设计基础的描述。

§1.5.2　I/O 寄存器

PIC16C52/54/56/58 有二个 I/O 口 RA、RB（F5、F6），PIC16C55/57 有三个 I/O 口 RA、RB、RC（F5、F6、F7）。与其它寄存器一样，它们皆可由指令来读写。它们是可编程双向 I/O 口，可由程序来编程确定每一根 I/O 端的输入/输出状态。控制方法见§1.8 节。

RESET 后所有的 I/O 口都置成输入态（等于高阻态），即 I/O 控制寄存器（TRISA、TRISB、TRISC）都被置成“1”。

1、F5（A 口）

4 位 I/O 口寄存器。只能使用其低 4 位。高 4 位永远定义为“0”。

2、F6（B 口）

8 位 I/O 口寄存器。

3、F7（C 口）

对于 PIC16C55/PIC16C57，它是一个 8 位 I/O 口寄存器。

对于 PIC16C54/56/58，它是一个通用寄存器。

§1.5.3　通用寄存器

PIC16C54/56：

07H～1FH

PIC16C55：

08H～1FH

PIC16C57/58：

08H-0FH：共有通用寄存器（无须体选择即可寻址）。

10H-1FH：Bank0 的通用寄存器

20H-2FH：物理上等同于 00H-0FH。

30H-3FH：Bank1 的通用寄存器

40H-4FH：物理上等同于 00H-0FH。

50H-5FH：Bank2 的通用寄存器

60H-6FH：物理上等同于 00H-0FH。

70H-7FH：Bank3 的通用寄存器。

请参考图 1.4。对寄存器体 Bank 的寻址请参阅 F4 寄存器描述和第四章的实例。

§1.5.4　特殊功能寄存器

1、工作寄存器（W）

W 用来存放两操作数指令中的第二个操作数，或用以进行内部数据传送。算术逻辑单元 ALU 把 W 和寄存器连接起来，ALU 的运算结果通过总据总线可以送到 W 保存。

2、I/O 控制寄存器（TRISA、TRISB、TRISC）

TRISA、TRISB、TRISC 分别对应 I/O 口 A、B、C。其中 TRISA 只有 4 位，和 A 口对应。执行“TRIS f”指令可把 W 的值置入 I/O 控制寄存器，以此来定义各 I/O 端的输入/输出态。当写入“1”时，将相应的 I/O 端置成输入态（高阻态），当写入“0”，则将相应的 I/O 端置成输出态。I/O 控制寄存器都是只写寄存器，在 RESET 后自动置为全“1”，即所有 I/O 口都为输入态。

3、预设倍数/RTCC 选择寄存器（OPTION）

OPTION 可用于：

a、定义预分频器的预分频参数。

b、分配预分频器（Prescaler）给 RTCC 或 WDT。注意预分频器只能分配给 RTCC 或 WDT 其中之一使用，不能同时分配。

c、定义 RTCC 的信号源。

d、定义 RTCC 信号源的触发沿（上升沿触发或下降沿触发）。

图 1.8 显示了 OPTION 各位的意义。

当预分频器分配给 RTCC 后，所有写 RTCC 寄存器的指令如 CLRF　1、MOVWF　1 等都会清除预分频器。同理，分配给 WDT 时，诸如 CLRWDT 和 SLEEP 指令将清除预分频器里已有的值使其归零。

通过执行“OPTION”指令可将 W 值置入 OPTIOW 寄存器，RESET 后 OPTION 被置成全“1”。

```
例：MOVLW     07H        ；W=7
    OPTION                ；7→OPTION ；预分频器(1:256)分配给 RTCC。RTCC 信号源为内
                          部指令时钟周期。
```

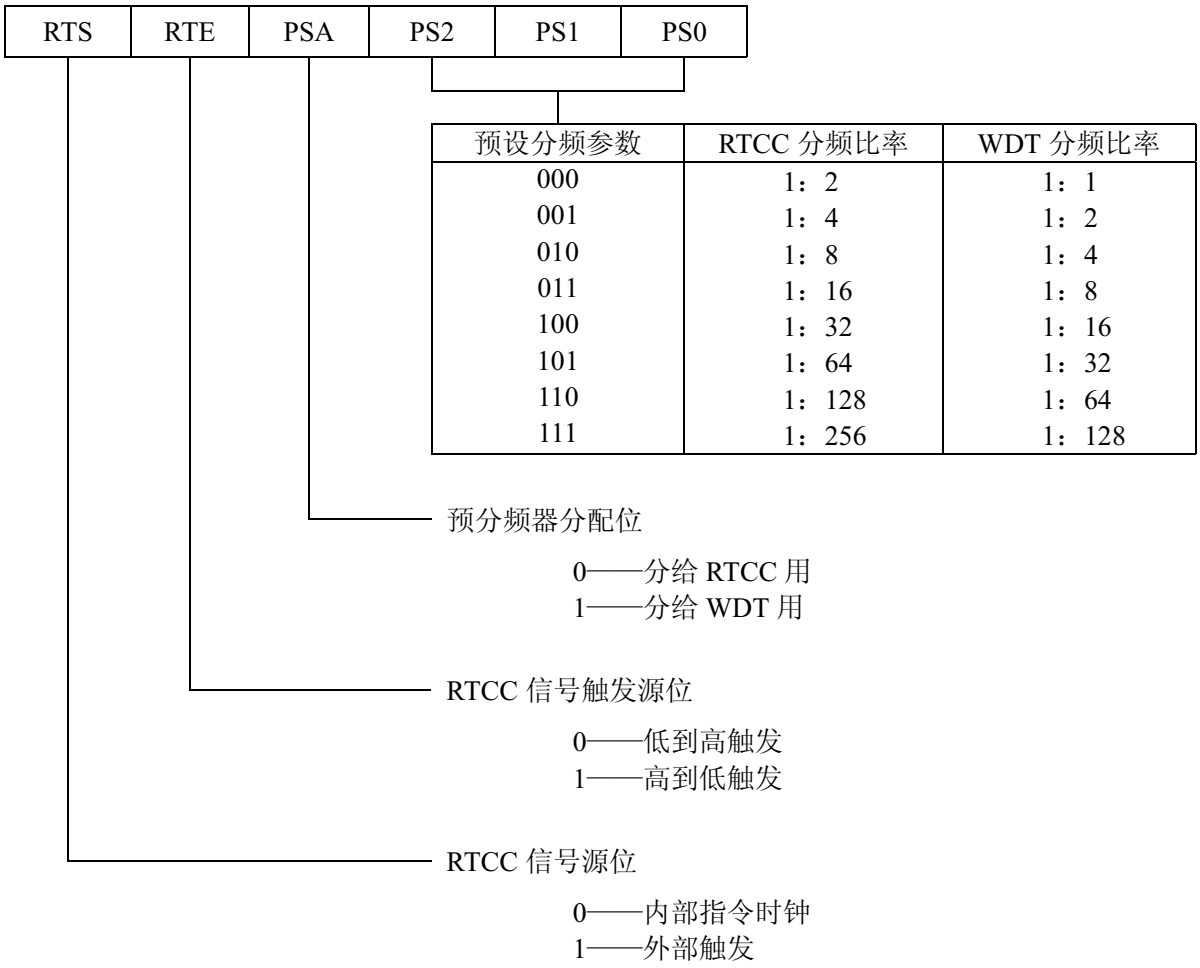


图 1.8　OPTION 寄存器

§1.6　预分频器 Prescaler

预分频器是一个分频倍数可编程的 8 位计数器。其结构如图 1.9 所示上节对预分频参数已有描述，这里不赘述。

预分频器的分配对象完全由程序控制。可以在程序中改变预分频器分配对象。

1、从 RTCC 到 WDT 的改变

```
MOVLW     B'XX0X0XXX'      ；选择内部时钟和新的预分频值
    OPTION                 ；如果新的预分频值=“000”或者
    CLRF      RTCC         ；=“001”，则暂时先选一个另外的值
    MOVLW     B'XXXX1XXX'  ；清零 RTCC 和预分频器
    OPTION                 ；选择 WDT 为对象，但不要改变预分频值
    CLRWDT                 ；清 WDT 和预分频器
    MOVLW     B'XXXX1XXX'  ；选择新的预分频值
```

2、从 WDT 到 RTCC 的改变

```
CLRWDT                          ；清 WDT 及预分频器
MOVLW     B'XXXX0XXX            ；选择 RTCC
OPTION
```

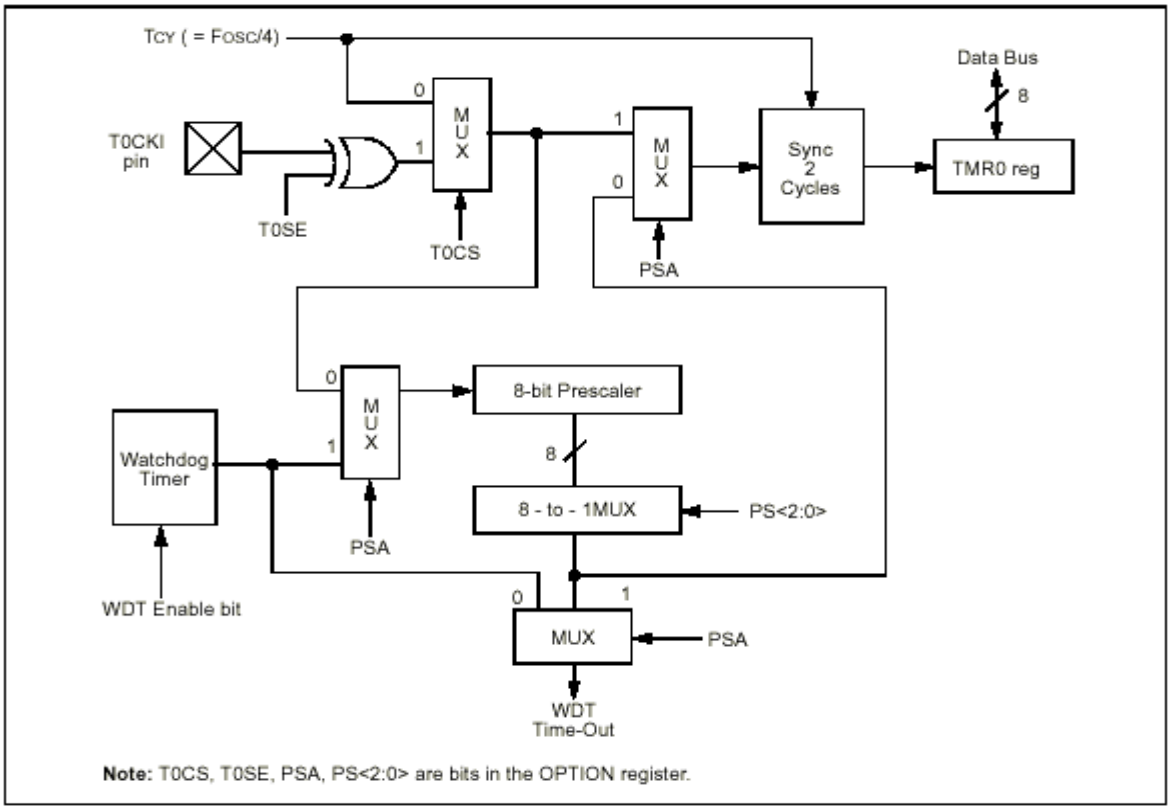


图 1.9　预分频器方块图

§1.7　看门狗 WDT

看门狗计时器（Watch Dog Timer）是一个片内自振式的 RC 振荡计时器，无需任何的外接元件。这意味着即使芯片 OSC1/OSC2 上振荡停止了（例如执行指令 SLEEP 后），WDT 照样保持计时。WDT 计时溢出将产生 RESET。在 PIC16C5X 芯片内有一个特殊的谓之“定义 EPROM”（Configuration EPROM）的单元，其中的一个位是用于定义 WDT 的。你可以将其置“0”来抑制 WDT 使之永远不起作用。这将在第七章的烧写器介绍部分详细说明，请参阅。

1、WDT 周期

WDT 有一个基本的溢出周期 18ms（无预分频器），如果你需要更长的 WDT 周期，可以把预分频器分配给 WDT，最大分频比可达 1:128，这时的 WDT 溢出周期约为 2.5S。WDT 溢出周期和环境温度、VDD 等参数有关系，请参阅附录的图表。

“CLRWDT”和“SLEEP”指令将清除 WDT 计时器以及预分频器里已有的计数值（当预分频器分配给 WDT 时）。WDT 一般用来防止系统失控或者说防止单片机程序运行“失控”。在正常情况下，WDT 应在计满溢出前被 CLRWDT 指令清零，以防止产生 RESET。如果程序由于某种干扰而失控，那么不能在 WDT 溢出前执行一条 CLRWDT 指令，就会使 WDT 溢出而产生 RESET，使系统重新启动运行而不至失去控制。若 WDT 溢出产生 RESET，则状态寄存器 F3 的“TO”位会被清零，用户程序可藉此判断复位是否由 WDT 溢时所造成。

2、WDT 编程注意事项

如果使用 WDT，一定要仔细在程序中的某些地方放一条“CLRWDT”指令，以保证在 WDT 在溢出前能被请零。否则会造成芯片不停地产生 RESET，使系统无法正常工作。

PEC16C5X 的所有 I/O 端的结构都是相同的，如图 1.10 所示：

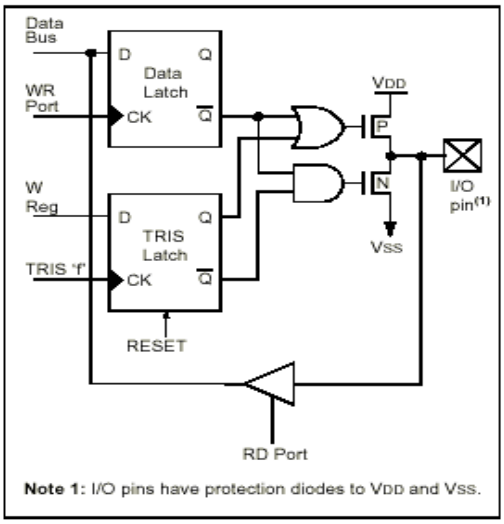
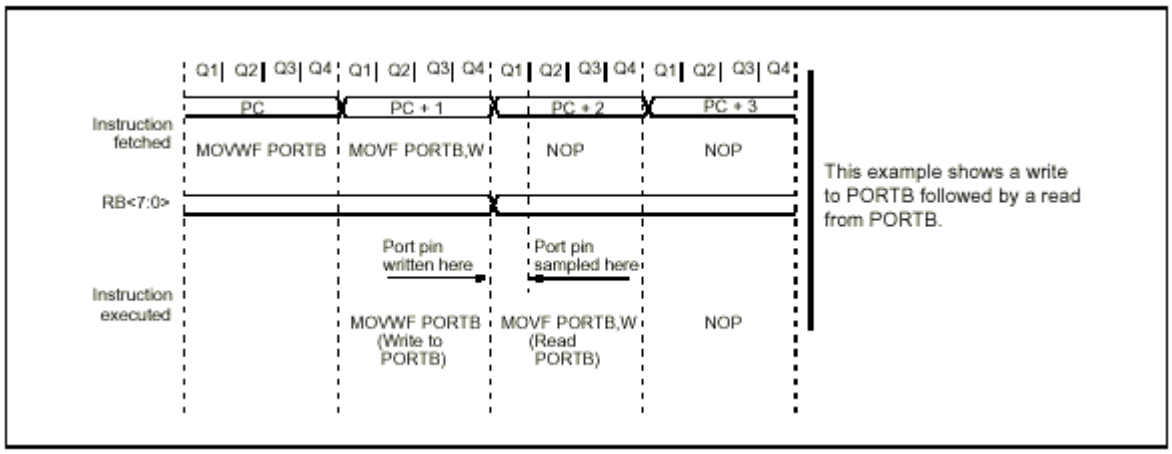


图 1.10　I/O 结构

所有 I/O 端皆可置成输入或输出态。输入无锁存，所以外部输入信号应保持到让 CPU 读入为止。输出锁存。

I/O 端的输入/输出状态由对应的 I/O 控制寄存器“TRIS　f”控制，当“TRIS　f”将“1”置入 I/O 控制器时 Q1 和 Q2 都处于截止态，所以 I/O 端即呈高阻态（输入态）。当执行 I/O 读指令（如 MOVF 6，W），把当前 I/O 端的状态读入数据总线。当“TRIS　f”将“0”置入 I/O 控制器时，Q1 和 Q2 的导通情况将要由数据锁存器 Q 端的状态来决定。当写入数据为“1”时，Q 端为低电平 0，则 Q1 导通，I/O 输出为高电平。反之，当写入数据为“0”时，Q 端为“1”，则 Q2 导通，I/O 端输出为低电平。I/O 读写时序如图 1.11 所示：



注：本图显示了 PORTB 口的一个写入→读出的连续动作。I/O 脚电平的建立时间=0.25TCY-TPD，其中 TCY 为指令周期，所以对于高速振荡来说，连续的写入→读出可能会有问题，两者中间应有延迟。

图 1.11　I/O 口读/写时序图

I/O 口使用注意事项：

a、I/O 方向转置的问题

某时候可能需要一个 I/O 口一会做输入，一会又做输出。这就是 I/O 方向的转置。

在编写这种 I/O 转置程序时必须注意，有些指令如位设置指令（BSF、BCF）写 I/O 口时是先从 I/O 读入其状态，执行位操作后再将结果写回去覆盖原来的内容（输出的结果放在 I/O 口的数据锁存器）。举

个例说：“BSF　6，5”这条指令的目的是要把 B 口的第 6 位置为高电平“1”。执行这条指令时，先把整个 B 口当前的状态内容读入到 CPU，把第 6 位置成“1”后再把结果（8 个位）重新输出到 B 口。如果 B 口中的有一个 I/O 端是需要方向转置的（比如说 bit1），而这时是处于输入态，那么 B 口的状态值重新写入后，B 口的数据锁存器 1（见图 1.9 相对于 B 口 bit1 的锁存器）的锁存值就是当前 B 口 Bit1 的状态。这可能和先前 Bit1 作为输出时所锁存的值不同，所以当 Bit1 再转置成输出态时，出现在 bit1 端的状态就可能和先前的输出态不同了。

b、I/O 的“线或”和“线与”

从图 1.10 看出:PIC I/O 端输出电路为 CMOS 互补推挽输出电路。 因此与其他这类电路一样，当某个 PIC I/O 端设置为输出状态时，不能与其他电路的输出端接成“线或”或“线与”的形式。否则可能引起输出电流过载，烧坏 PIC。

如需要与其他电路接成“线或”电路时，PIC I/O 端必须置于“0”状态或输入状态并外接上拉电阻。如需要接成“线与”电路时，则 PIC I/O 端必须置于“1”状态或输入状态，并外接下拉电阻。电阻的阻值根据实际电路和 PIC I/O 端最大电流来选定。

c、I/O 口的连续操作

一条写 I/O 的指令，对 I/O 真正写操作是发生在指令的后半周期（参照图 1.11）。而读 I/O 的指令却是在指令的周期开始就读取 I/O 端状态。所以当你连续对一个 I/O 端写入再读出时，必须要让 I/O 端上的写入电平有一个稳定的时间，否则读入的可能是前一个状态，而不是最新的状态值。一般推荐在两条连续的写，读 I/O 口指令间至少加一条 NOP 指令。

```
例：　　MOVWF　　6　　　　；　写 I/O
　　NOP　　　　；　稳定 I/O 电平
　　MOVF　　6，W　　；　读 I/O
```

d、噪声环境下的 I/O 操作

在噪声环境下（如静电火花），I/O 控制寄存器可能因受干扰而变化。比如 I/O 口可能会从输入态自己变成输出态，对于这种情形，WDT 也是无法检测出来的。因此如果你的应用环境是较恶劣的，建议你每隔一定的间隔，都重新定义一下 I/O 控制寄存器。

§1.9　振荡电路

PIC16C5X 系列可以使用 4 种类型振荡方式：标准晶体/陶瓷振荡 XT、高速晶体振荡 HS（4MHZ 以上）、低频晶体振荡 LP（32KHz）以及阻容振荡 RC。

对于窗口型可重擦除芯片可以通过对“定义 EPROM”（Coriguratiou EPROM）编程来选择任何一种振荡方式。对于 OTP 和掩腌片 QTP 则由厂家定义好振荡方式，并通过相应的检测。

仿真工具如 PICMATE 仿真器和 PICKIT 烧写器等可以让用户选择所需的振荡方式进行仿真和烧写，请参阅第六章和第七章介绍。

§1.9.1　晶体/陶瓷振荡

这种振荡包括 XT、HS 和 LP。其电路是在 OSC1 和 OSC2 两端加一晶体/陶瓷振荡，如图 1.12。只有“HS”晶体振荡才可能需要 Rs（100Ω<Rs<1KΩ）。

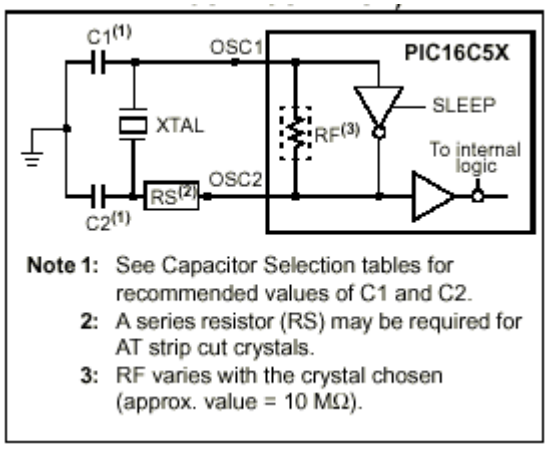


图 1.12　晶体/陶瓷振荡电路

表 1.7 列出了使用陶瓷振荡器时所需的电容值。表 1.8 列出了使用晶体振荡器时所需的电容值。

类型	频　率	电容值（C1=C2）
XT	455 KHZ	150－330P
	2.0 MHZ	20－330P
	4.0 MHZ	20－330P
HS	8.0 MHZ	20－200P

表 1.7 使用陶瓷振荡器时所需的电容值

类型	频率	C1	C2
LP	32KHZ	15P	15P
XT	100KHZ	15－30P	200－300P
	455KHZ	15－30P	100－200P
	1MHZ	15－30P	15－30P
	2MHZ	15P	15P
	4MHZ	15P	15P
HS	8MHZ	15P	15P
	20MHZ	15P	15P

表 1.8 使用晶体振荡时所需的电容值表

电容值取大有利于振荡的稳定，但却延长了起振时间。表中的电容值能满足一般的要求。

§1.9.2　RC 振荡

这种振荡类型成本最低，但频率的精确性较差，适用于时间精确度要求不高的应用场合。RC 振荡的频率是ＶDD、RC 值以及环境温度的函数。请参阅附录的 RC 频率函数图。RC 振荡的连接如图 1.13 所示。

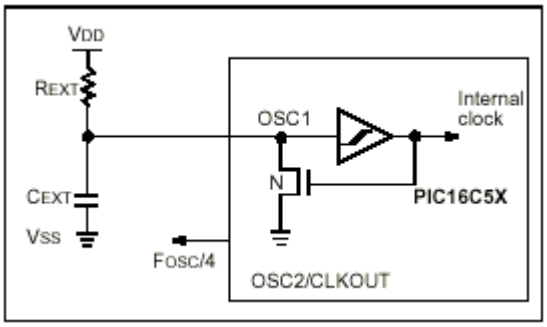


图 1.13　RC 振荡电路

RC 振荡是在 OSC1 端连接一个串联的电阻电容。这个电阻如果低于 2.2K，振荡不稳定，甚至不能振荡。但是电阻高于 1M 时，则振荡又易受干扰。所以电阻值最好取 5K－100K 之间。尽管电容 C 值为 0 时，电路也能振荡，但也易受干扰且不稳定，所以电容值应取 20P 以上。RC 值和频率关系如表 1.9 所示。RC 振荡时 OSC2 端输出－OSC1 的 4 分频脉冲（f=1/4 OSC1）。

Rext	Cext	Vdd	Fosc/25℃
5KΩ	0PF	5.0	4.0MHZ
5KΩ	20PF	6.0	2.2MHZ
5KΩ	20PF	3.5	2.5MHZ
10KΩ	130PF	5.0	480KHZ
10KΩ	290PF	5.0	245KHZ
100KΩ	300PF	3.5	30KHZ

表 1.9　　R/C 与频率的关系

§1.9.3　外部振荡

PTC16CSX 可以接受外部振荡源（仅适合于 HS、XT 和 LP 类型振荡）。连接时将外部振荡接入 OSC1，

OSC2 则开路。如图 1.14 所示。

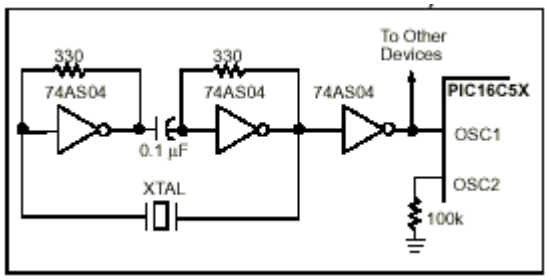


图 1.14 外部振荡电路

§1.9.4 时钟/指令时序

振荡器信号从 OSC1 端输入单片机后，经过 4 分频电路产生 4 个不重叠的内部时钟信号 Q1、Q2、Q3、Q4。时序图如 1.15 所示。

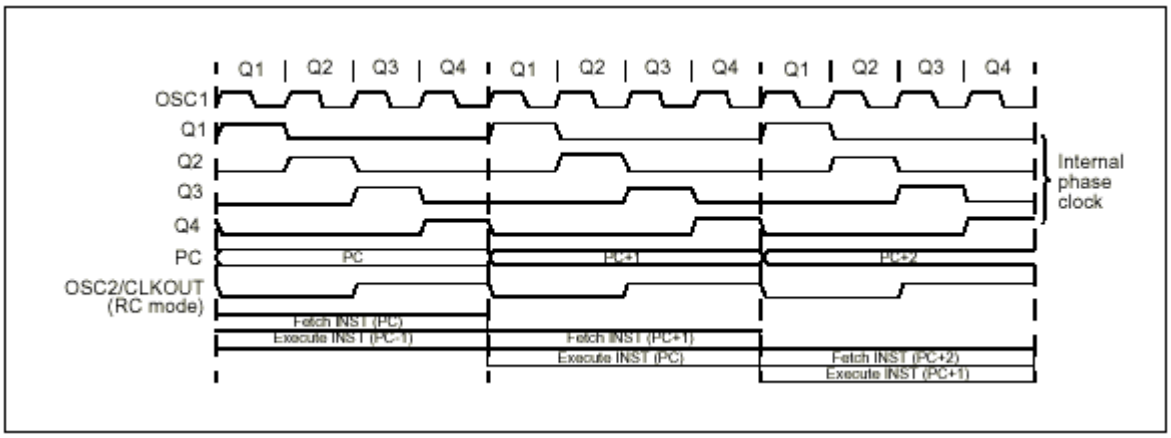


图 1.15 时钟/指令时序图

一条指令周期需经 Q1～Q4 四个节拍时间来完成。PIC16C5X 除了地址跳转指令是二周期指令，其余全是单周期指令。一条指令正在执行时（Q1～Q4 节拍间），PC 值又在 Q1 节拍间加 1，把下一条要执行的指令取到指令寄存器，准备让 CPU 执行下一条指令了，这是 RISC 结构单片机的特点，它使单片机的速度加快（同等振荡频率下比一般 CISC 结构的单片机如 Z86、68HC 等快 4 倍速）。

表 1.10 列出了振荡时钟频率和指令速度的关系。

频率	双周期指令	单周期指令
1MHz	8us	4us
4MHz	2us	1us
8MHz	1us	500ns
20MHz	400ns	200ns

表 1.10 频率和指令速度关系表

§1.10 复位（RESET）

PIC16C5X 内藏有上电复位电路(POR)。在芯片上有一复位端 MCLR，对于一般的应用，只要把 MCLR 端接在高电位（VDD）即可，因为内部复位电路会在芯片上电时自动复位，无需在 MCLR 端再加上电复位电路。对于某些特殊应用，则需在 MCLR 端加上外部上电复位电路，在§1.10.5 我们会谈及这个问题。

§1.10.1 复位的条件和原因

复位可由下面事件引发产生：

a、芯片上电；

b、把芯片 MCLR 端置低电平；

c、看门狗（WDT）超时溢出。

§1.10.2　复位时的 PIC 状态

在芯片复位期间，芯片状态为：

a、振荡器处于起振准备状态；

b、所有 I/O 口都被置成高阻态（即输入态）；

c、PC 值被置为全“1”；

d、OPTION 被置为全“1”；

e、WDT 和预分频器被清零；

f、状态寄存器（F3）的程序页面位（高三位）被清零。

§1.10.3　振荡起振计时器（OST）

对于晶体/陶瓷振荡电路，上电后它们还需要一定的时间来起振或产生稳定的振荡信号，有鉴于此，PIC 在其内部专门设置了一个“振荡起振计时器”OST（Oscillator Start-up Timer）。OST 在 MCLR 端达到高电平后才开始启动计时 18ms，使 RESET 状态保持 18ms 以便让振荡电路起振及稳定下来。在一般情况下，我们都将 MCLR 端直接在 VDD（+5V）上即可。这样上电后一旦 MCLR 端电平升高到一定程序后 OST 即开始计数 18ms，这段时间已足够让振荡起振，OST 计满 18ms 后，芯片结束 RESET 状态，开始进入程序运行。

当 WDT 计时溢出后，OST 也是马上启动计时 18ms，保持 18ms 的 RESET 状态，然后再进行程序运行。

§1.10.4　内部上电复位路（POR）

PIC16C5X 片内上电复位电路 POR（Power On Reset）能使 PIC 芯片上电后自动会产生复位，所以一般不需要再在 MCLR 端加外部复位电路，只要将其接在 VDD 上即可。图 1.16 是 POR 的简图。

从图中我们可以看到，当上电（Power On）、或 MCLR 端变低，都会置位（set）“复位锁存器”，使其输出复位电平让芯片处于 RESET 状态，这时 OST 也处于复位状态。当 OST 检测到 MCLR 变为高电平后即开始计时 18ms，计满 18ms 后会复位（Reset）“复位锁存器”使其 Q 端输出高电平，从而使芯片结束复位状态，进入运行。

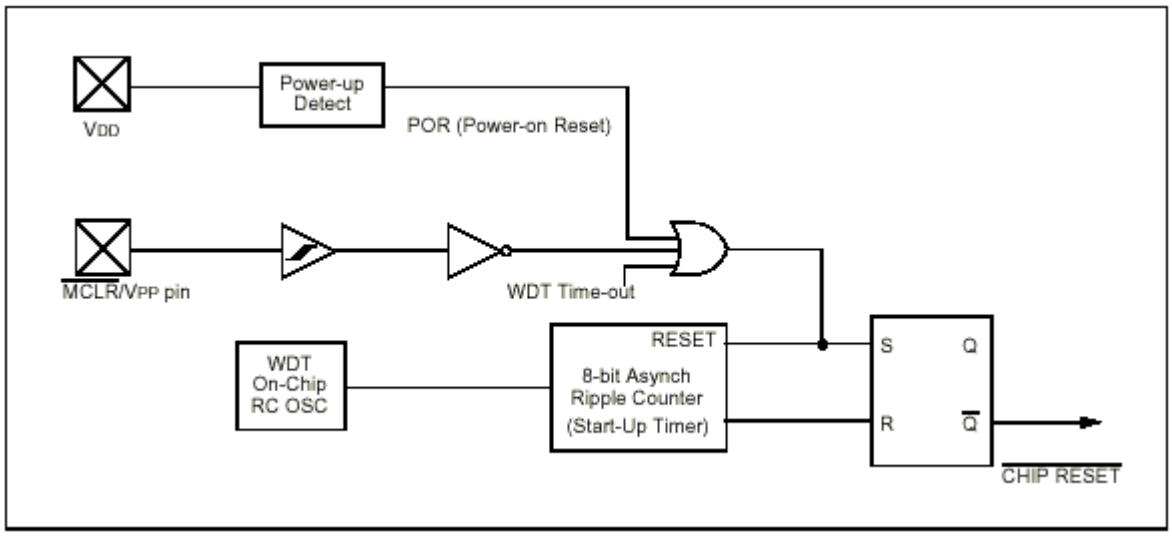


图 1.16　内部上电复位路简图

§1.10.5　外部复位电路

在某些应用情况下我们也可能需要外部复位电路。

一、手动复位开关

当你在应用中需要一个手动复位开关时，可以使用下面的电路。如图 1.17。

二、低频振荡电路

当使用低频振荡（LP）时，OST 的 18ms 不足以使其建立稳定的振荡，所以也许你需要更长的 RESET 时间，这时可以用外部上电复位电路来延长复位时间。如图 1.18。

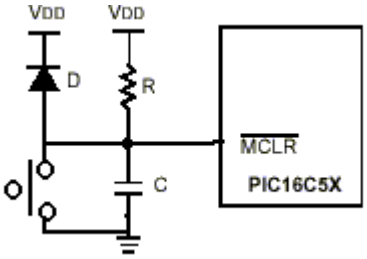


图 1.17 按健复位电路

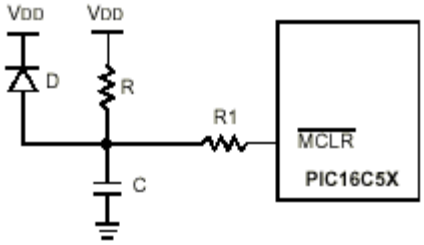


图 1.18 外部上电复位电路

注：1. 二极管 D 使电容 C 能在 VDD 掉电时快速放电。

2. R<40KΩ，以保证其两端的电压降不大于 0.2V（即 IMCLR*R<0.2V，IMCLR 为 MCLR 端最大输入电流）。

3. R1 取 100Ω～1KΩ，用来限制在静电环境，电容 C 充放电时的冲击电流。

使用了外部加电复位电路，其复位过程即如图 1.19 所示：

图 1.19　使用外部上电复位电路的复位过程

VDD 上升到稳定值一段时间后 MCLR 才上升到高电平。而 OST 只有检测到 MCLR 升为高电平后才开始计时 18ms，所以就可取得长于 18ms 的复位时间了。

§1.10.6　复位后对寄存器值的影响

对于通用寄存器来说，芯片上电复位后它们的值是随机不定的，而其他类型的复位则保持原值不变。

对于特殊功能寄存器，各种复位后它们会等于一个固定的复位值，见以下二表：

复 位 类 型	程序计数器 PC	状态寄存器 STATUS
上电复位	1111　1111	0001　1xxx
运行时 MCLR 端拉低	1111　1111	000u　uuuu
睡眠时 MCLR 端拉低	1111　1111	0001　0uuu
运行时 WDT 溢出	1111　1111	0000　1uuu
睡眠时 WDT 溢出	1111　1111	0000　0uuu

注：x=不定，u=不变

表 1.11　复位对 PC 和 STATUS 的影响

寄存器	地 址	程序计数器 PC	MCLR 和 WDT 复位
W	—	xxxx　xxxx	uuuu　uuuu
TRIS	—	1111　1111	1111　1111
OPTION	—	--11　1111	--11　1111
F0	00h	xxxx　xxxx	uuuu　uuuu
RTCC	01h	xxxx　xxxx	uuuu　uuuu
PC	02h	1111　1111	1111　1111
STATUS	03h	0001　1xxx	000q　quuu
FSR	04h	1xxx　xxxx	1uuu　uuuu
PORT A	05h	----　xxxx	----　uuuu
PORT B	06h	xxxx　xxxx	uuuu　uuuu
PORT C	07h	xxxx　xxxx	uuuu　uuuu

注：x=不定，u=不变，　-=未用，q=取决于某条件。

PORT C 仅 16C55/57 有，其余则为通用寄存器 F7。

表 1.12　特殊寄存器的复位值

§1.11　低功耗模式（SLEEP）

一、进入 SLEEP

执行一条“SLEEP”指令即可进入低功耗模式。当进入 SLEEP 后，WDT 被清零，然后重新开始计数。状态寄存器 F3K 中的 PD 位被置成“0”，TO 位置成“1”，同时振荡停止（指 OSC1 端的振荡电路）。所有的 I/O 口保持原来的状态。这种工作模式功耗最低。

为使耗电流最小，进入 SLEEP 前，应使所有的 I/O 口处于高电平 VDD 或低电平 VSS，而不应使其处于高阻态，以免产生开关电流损耗。你可以在 I/O 口加上拉或下拉电阻，或者把 I/O 口都置成输出态来避免其处于高阻态（浮态）。

RTCC 端亦应置为 VDD 或 VSS（通过上拉或下拉）。

MCLR 必须处于高电平状态。

二、唤醒 SLEEP

SLEEP 可被 WDT 溢出唤醒；或在 MCLR 端加低电平唤醒 SLEEP。后一种唤醒方法经常用在以下应用场合：在系统主电源掉电，并由后备电源（电池）供电后，执行“SLEEP”指令进入低功耗模式，这样电池就可长时间保持系统数据。当主电源恢复供电时，让其在 MCLR 产生一低电平唤醒 SLEEP，并重新复位。这样需在 MCLR 端加一外部复位电路，请参考 § 1.10.5。

系统上电时，F3 的 PD 被置为“1”，而执行“SLEEP”指令后，PD 位被置成“0”。所以通过 PD 位可以判断系统是从 SLEEP 模式唤醒而复位，还是上电后的复位。F3 中的 TO 位则可判断当处于 SLEEP 状态的系统是由 WDT 溢时唤醒或是由外界给 MCLR 端一个低电平而唤醒。这些区别有时是很重要的，特别是对系统的一些初始化工作来说。

§1.12　系统定义字（Configuration）

在 PIC 芯片内有一特殊的系统定义字含有 4 个 EPROM 熔丝。它不是程序存贮器 EPROM 的组成部分（不包括在 0.5K-2K 的程序空间内）。其中两个熔丝用以选择四种振荡方式（RC、XT、HS、LP），另两个熔丝一个用来选择使能（enable）看门狗 WDT，一个用来选择使能程序保密位。

用户可以在烧写 OTP 或窗口型芯片时，选择烧写这四个熔丝。详见第七章烧写工具介绍说明。对于掩膜芯片，则由生产厂根据客户需要在芯片生产过程中予以烧写。

§1.12.1　程序保密位（Protection Fuse）

当你选择将芯片的程序保密位熔丝熔断（写入 0）后，程序存贮区 ROM 中的程序代码（12 位宽）的高 8 位将被遮没。具体地说，就是当再去读 ROM 中的程序代码时，每一个代码都呈现 00XH 的形式。这样高 8 位被用 0 替代了，只留低 4 位，别人就无法恢复这些被加密的代码，也无法进行代码复制、拷贝了。但单片机的功能不受影响，加密后的程序码并不影响其在单片机内的运行，只是不能被还原读出来。

注意：当芯片被选择为保密方式后，程序存贮区 40H 以上的空间即不能再被编程，而 003FH 之间的空间还能编程。在程序存贮区中，“1”可被烧写成“0”，反之则不可。

§1.12.2 用户识别码（Customer ID Code）

在 PIC16C5X 内部还有一个 16 位的特殊 EPROM（不包括在程序存贮区内），可让用户烧入 4 个十六进制码，以作为芯片标识。这个识别码只起识别作用，对程序无影响，用户可在烧写器上将其烧入和读出验识。

第二章　PIC16C5X 指令集及程序设计技巧

§2.1　PIC165X 指令概述

PIC16C5X 每条指令长 12 位，指令由操作码和操作数组成。PIC16C5X 共有 33 条指令，按操作分成三大类：

1. 面向字节操作类
2. 面向位操作类
3. 常数操作和控制操作类

全部指令如表 2.1 所示。

§2.2　PIC16C5X 指令寻址方式

PIC16C5X 单片机寻址方式根据操作数的来源，可分为寄存器间接寻址、立即数寻址、直接寻址和位寻址四种。

一、寄存器间接寻址

这种寻址方式通过寄存器 F0、F4 来实现。实际的寄存器地址放在 F4 中，通过 F0 来进行间接寻址。

```
例：    MOVLW    05H          ; W=5
        MOVWF    4            ; W(=5)→F4
        MOVLW    55H          ; W=55H
        MOVWF    0            ; W(=55H)→F5
```

上面这段程序把 55H 送入 F5 寄存器。间址寻址方式主要用于编写查表、写表程序，非常方便。请参考 §2.7 程序设计技巧。

二、立即数寻址

这种方式就是操作数为立即数，可直接从指令中获取。

```
例：  MOVLW  16H          ; 16H →W
```

三、直接寻址

这种方式是对任何一寄存器直接寻址访问。对 16C52/54/55/56 来说，寄存器地址（5 位）直接包括在指令中。对 PIC16C57，寄存器地址中高 2 位由（选 Bank）由 FSR<6:5>二位决定。

```
例：    MOVWF    8            ; W→F8 寄存器
        MOVF     8，W         ; F8→W
```

四、位寻址

这种寻址方式是对寄存器中的任一位（bit）进行操作。

```
例：    BSF      11，0        ；把 F11 的第 0 位置为“1”。
```

§2.3　面向字节操作类指令

这类指令共有 18 条，包括有数据传送、算术和逻辑运算、数据移位和交换等操作。它们的操作都是在 W 数据寄存器 f 之间进行，其指令码结构为：

（11—6）	（5）	（4—0）
OPCODE	d	f（File#）

高 6 位是指令操作码。第 6 位 d 是方向位。d=1，则操作结果存入 f（数据寄存器），d=0，则操作结果存入 W。低 5 位是数据寄存器地址，可选中 32 个寄存器。对于 PIC16C57，则还要参考寄存器体选择器 F4 的 bit5 或 bit6 来选择存入哪一个寄存器体（bank0－bank3）。

面向字节操作类指令

（11-6）	（5）	（4-0）
OPCODE	d	f（FILE#）

二进制代码	HEX	名称	助记符，操作数	操作	状态影响	注
0000 0000 0000	000	空操作	NOP		无	
0000 001f ffff	02f	W 送到 f	MOVWF f	W→f	无	1，4
0000 0100 0000	040	W 清零	CLRW -	0→W	Z	
0000 011f ffff	06f	f 清零	CLRF f	0→f	Z	4
0000 10df ffff	08f	f 减去 W	SUBWF f，d	f-W→d	C，DC，Z	1，2，4
0000 11df ffff	0Cf	f 递减	DECF f，d	f-1→d	Z	2，4
0001 00df ffff	10f	W 和 f 做或运算	IORWF f，d	W∨f→d	Z	2，4
0001 01df ffff	14f	W 和 f 做与运算	ANDWF f，d	W∧f→d	Z	2，4
0001 10df ffff	18f	W 和 f 做异或运算	XORWF f，d	W○f→d	Z	2，4
0001 11df ffff	1Cf	W 加 f	ADDWF f，d	W +f→d	C，DC，Z	1，2，4
0010 00df ffff	20f	传送 f 到 d	MOVF f，d	f→d	Z	2，4
0010 01df ffff	24f	f 取补	COMF f，d	f→d	Z	2，4
0010 10df ffff	28f	f 递增	INCF f，d	f+1→d	Z	2，4
0010 11df ffff	2Cf	f 递减，为 0 则跳	DECFSZ f，d	f-1→d，skip if zero	Z	2，4
0011 00df ffff	30f	f 循环右移	RRF f，d	f(n)→d(n-1)，f(0)→C，C→d(7)	C	2，4
0011 01df ffff	34f	f 循环左移	RLF f，d	f(n)→d(n+1)，f(7)→C，C→d(0)	C	2，4
0011 10df ffff	38f	f 半字节交换	SWAPF f，d	f(0.3)←→f(4-7)→d	Z	2，4
0011 11df ffff	3Cf	f 递增，为 0 则跳	INCFSZ f，d	f+1→d，skip if zero	Z	2，4

面向位操作类指令

（11-8）	（7-5）	（4-0）
OPCODE	b（BIT#）	f（FILE#）

二进制代码	HEX	名称	助记符，操作数	操作	状态影响	注
0100 bbbf ffff	4bf	清除 f 的位 b	BCF f，b	0→f(b)	Z	2，4
0101 bbbf ffff	5bf	设置 f 的位 b	BSF f，b	1→f(b)	Z	2，4
0110 bbbf ffff	6bf	测试 f 的位 b，为 0 则跳	BTFSC f，b	Test bit(b) in file(f)：Skip if clear	Z	
0111 bbbf ffff	7bf	测试 f 的位 b，为 1 则跳	BTFSS f，b	Test bit(b) in file(f)：Skip if clear	Z	

常数操作和控制类指令

（11-8）	（7-0）
OPCODE	k（LITERAL）

二进制代码	HEX	名称	助记符，操作数	操作	状态影响	注
0000 0000 0010	002	写 OPTION 寄存器	OPTION -	W→OPTION register	无	
0000 0000 0011	003	进入睡眠状态	SLEEP -	0→WDT，stop oscillator	TO，PD	
0000 0000 0100	004	清除 WDT 计时器	CLRWDT -	0 → WDT(and prescaler ， if assigned)	TO，PD	
0000 0000 0fff	00f	设置 I/O 状态	TRIS f	W→I/O control register f	无	3
1000 kkkk kkkk	8kk	子程序带参数返回	RETLW k	k→W，Stack→PC	无	

1001 kkkk kkkk	9kk	调用子程序	CALL k	PC+1→Stack，K→PC	无	1
101K kkkk kkkk	Akk	跳转(K 为 9 位)	GOTO k	k→PC(9 bits)	无	
1100 kkkk kkkk	Ckk	常数置入 W	MOVLW k	k→W	Z	
1101 kkkk kkkk	Dkk	常数和 W 做或运算	IORLW k	k∨W→W	Z	
1110 kkkk kkkk	Ekk	常数和 W 做与运算	ANDLW k	k∧W→W	Z	
1111 kkkk kkkk	Fkk	常数和 W 做异或运算	XORLW k	k○W→W	Z	

表 2.1　PIC16C5X 指令集

注：（1）除 GOTO 指令外，任何有关写 PC（F2）的指令（例如 CALL、MOVWF　2）都将会把 PC 寄存器的第 9 位清零。

（2）若对 I/O 口寄存器进行操作，如“SUBWF 6，1”，则使用的 F6 的值是当前 B 口上的状态值，而非 B 口输出锁存器里的值。

（3）指令“TRIS　f”（f=5、6 或 7）将 W 寄存器中的内容写入 f 的 I/O 口控制寄存器中：“1” 关断对应端口的输出缓冲器，使其为高阻状态。

（4）当预分频器分配给 RTCC 后，任何对 RTCC 寄存器（F1）写操作的指令都将使预分频器（Prescaler）清零。

1、寄存器加法指令

格式：　ADDWF　f ，d

指令码：

000111	d	fffff

指令周期：　1

操作：　W+f→d

影响状态位：　C，DC，Z

说明：　将 f 寄存器和 w 相加，结果存入 f(d=1)或 W(d=0)。

例：　ADDWF　8，0　; F8+W→W

2、寄存器与指令

格式：　ANDWF　f，d

指令码：

000101	d	fffff

指令周期：　1

操作：　W∧f→d

影响状态位：　Z

说明：　将 f 寄存器和 w 做逻辑与运算，结果存入 f(d=1)或 W(d=0)

例：　ANDWF　10 0　;F10∧W→W

ANDWF　10，1　;F10∧W→F10

3、寄存器清零指令

格式：　CLRF　f

指令码：

0000011	fffff

指令周期：　1

操作：　0→f

影响状态位：　Z

说明：　将 f 寄存器清零。

例：　CLRF　8　; F8 清为零(0→F8)

4、W 清零指令

格式：　CLRW

指令码：

000001	0	00000

指令周期：　1

操作：　0→W

影响状态位：　Z

说明：　将 W 寄存器清零，状态位 Z 将被置为 1。

5、寄存器取反指令

格式：　COMF　f，d

指令码：

00	d	fffff

指令周期：　1

操作：　f→d

影响状态位：　Z

说明：　将 f 寄存器内容做逻辑求反运算，结果存入 f(d=1)或 W(d=0)。

例：　COMF　12 ，0　;F12 取反→F12

　　COMF　12，1　;F12 取反→W

6、寄存器减 1 指令

格式：　DECF　f，d

指令码：

000011	d	fffff

指令周期：　1

操作：　f－1→d

影响状态位：　Z

说明：　f 寄有器内容减 1 存入 f(d=0)或 W(d=0)。

例：　DECF　15，1　;F15－1→F15

　　DECF　15，0　;F15－1→W

7、寄存器减 1，结果为零则跳指令

格式：　DECFSZ　f，d

指令码：

0000	11df	ffff

指令周期：　1 或 2（产生跳转时为 2）

操作：　f－1→d; 结果为零则跳(PC+1→PC)

影响状态位：　无

说明：　将 f 寄存的内容减 1 存入 f(d=1)或 W(d=0)。如果结果为 0，则跳过下一条指令不执行。否则顺序执行下一条指令。

例：

```
        ┌──DECFSZ   10，1   ;F10－1→F10，如果 F10 为 0
F10=0   │   MOVLW   55H     ;则跳过 MOVLW 55H 指令
        └──→MOVF    12，0
```

8、寄存器加 1 指令

格式：　INCF　f，d

指令码：

001010	d	fffff

指令周期：　1

操作：　f+1→d

影响状态位：　Z

说明：　f 寄存器加 1，结果存入 f(d=1)或 W(d=0)。

例：　INCF　10，0　;F10+1→W

　　INCF　10，1　;F10+1→F10

9、寄存器加 1，结果为零则跳指令

格式：　INCFSZ　f，d

指令码：

001111	d	fffff

指令周期：　1 或 2（产生跳转时为 2）

操作：　f+1→d，结果为零则跳(PC+1→PC)

影响状态位：　无

说明：　将 f 寄存器内容加 1 存入 f(d=1)或 W(d=0)，如果结果为零则 PC 值加 1 跳过下一条指令。

例：
```
L00P    ┌──INCFSZ   8，1    ；将 F8 寄存器加 1，结果存入 F8
        │  GOTO     LOOP    ；加 1 后结果为零则跳到 MOVWF F9 指令
F8=0    └─→MOVWF    9
```

10、寄存器或指令

格式：　IORWF　f，d

指令码：

000100	d	fffff

指令周期：　1

操作：　W∨f→d

影响状态位：　Z

说明：　将 f 暂存器内容和 W 内容做逻辑或运算，结果存入 f(d=1)或 W(d=0)。

例：
```
IORWF   18，1       ;F18∨W→F18
IORWF   18，0       ;F18∨W→W
```

11、f 寄存器传送指令

格式：　MOVF f，d

指令码：

001000	d	fffff

指令周期：　1

操作：　f→d

影响状态位：　Z

说明：　将 f 寄存器内容传送至 W(d=0)或自己本身 f(d=1)。如果是传给自己，一般是用来影响状态位 Z，即可判断 f 是否为零。

例：
```
MOVF    10，1       ;F10→F10
BTFSS   3，2        ;判断 F3 的第二位，即 Z 状态位。如果 F10=0，则 Z=1。
```

12、W 寄存器传指令

格式：　MOVWF　f

指令码：

000000	1	fffff

指令周期：　1

操作：　W→f

影响状态位：　无

说明：　将 W 内容传给 f 寄存器。

例：
```
MOVWF   6                   ; W→F6(B 口)
```

13、空操作指令

格式：　NOP

指令码：

000000	000000

指令周期：　1

操作：　无任何操作

影响状态位：　无

说明：　不做任何操作，只有使 PC 加 1。

14、带进位位左移指令

格式：　　RLF　f，d

指令码：

001101	d	fffff

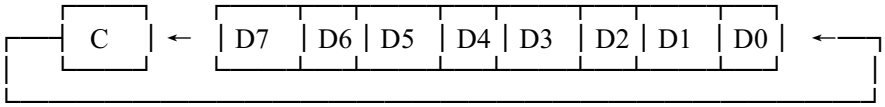
指令周期：　1

操作：　　f(n)→d(n+1)，f(7)→c，c→d(0)

影响状态位：　C

说明：　　将 f 寄存器左移，结果存入 f(d=1)或 W(d=0)。f 左移时，其最高位(bit7)移入状态位 C(进位位)，如下图：

进位位



例：　　RLF　8，1　　; F8 左移→F8

RLF　8，0　　; F8 左移→W

15、带进位位右移指令

格式：　　RRF　f，d

指令码：

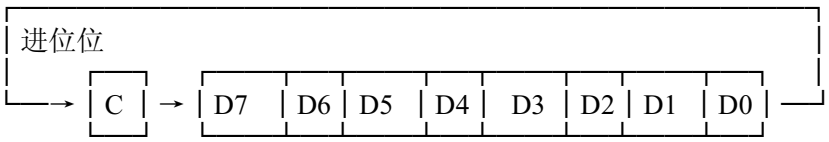
001100	d	fffff

指令周期：　1

操作：　　f(n)→d(n-1)，f(0)→c，c→d(7)

影响状态位：　C

说明：　　将 f 寄存器右移，结果存入 f(d=1)或 W(d=0)。f 右移时，其最低位(bito)移入状态位 C， 而原来的状态位 C 移入 f 最高位(bit7)，如下图：



例：　　RRF　8 ，1　　;F8 右移→F8

RRF　8，0　　;F8 右移→W

16、寄存器减法指令

格式：　　SUBWF　f，d

指令码：

000010	d	fffff

指令周期：　1

操作：　　f-w→d

影响状态位：　C，DC，Z

说明：　　将 f 寄存器内容减去 W 内容，结果存入 f(d=1)或 W(d=0)。

例：　　CLRF　20　　;F20=0

MOVLW　1　　;W=1

SUBWF　20，1　　;F20-W=0-1=-1→F20

;C=0，运算结果为负。

17、寄存器交换指令

格式：　　SWAPF　f，d

指令码：

001110	d	fffff

指令周期：　1

操作：　　f(0-3)→d(4-7)，f(4-7)→d(0-3)
影响状态位：　无
说明：　　将 f 寄存器内容的高 4 位(bit7-bit4)和低 4 位(bit3-bit0)交换
　　结果存入 f(d=1)或 W(d=0)。
例：　　MOVLW　56H
　　MOVWF　8　　;F8=56H
　　SWAPF　8，1　　;F8 交换，结果存入 F8，则 F8=65H。

18、寄存器异或运算指令
格式：　XORWF　f，d

指令码：			
	00010	d	fffff

指令周期：　1
操作：　　W○f→d
影响状态位：　Z
说明：　　将 f 寄存器和 W 进行异或运算，结果存入 f(d=1)或 W(f=0)。
例：　　XORWF　5，1　　;F5○W→F5(A 口)
　　XORWF　5，0　　;F5○W→W

§2.4　面向位操作指令

这类指令共有 4 条，指令码基本结构为：

（11—8）	（7—5）	（4—0）
OPCODE	bbb	file#

高 4 位是操作码。bit5－bit7 是位地址（可寻址 8 个位），bito－bit4 是寄存器地址。

19、位清零指令
格式：　BCF　f，b

指令码：			
	0100	bbb	fffff

指令周期：　1
操作：　　0→f(b)
影响状态位：　无
说明：　　将 f 寄存器的 b 位清为 0。
例：　　BCF　8，2　　;将 F8 的第 2 位(bif2)清为 0。

20、位置 1 指令
格式：　BSF　f，b

指令码：			
	0101	bbb	fffff

指令周期：　1
操作：　　1→f(b)
影响状态位：　无
说明：　　将 f 寄存器的 b 位置为 1。
例：　　BSF　8，2　　;将 F8 的第 2 位(bif2)置为 1。

21、位测试，为零则跳指令
格式：　BTFSC　f，b

指令码：			
	0110	bbb	fffff

指令周期：　1 或 2（产生跳转则为 2）
操作：　如果 f(b)=0 则跳(PC+1→PC)
影响状态位：　无
说明：　测试 f 寄位器的第 b 位，如果位 b 为零则跳过下一条指令，否则顺序执行下去 。
例：

```
        ┌──BTFSC   8，2      ;测试 F8 的 bit2，如果为 0
bit2=0  │   MOVF   5，0      ;则跳到 INCF9，1 执行。
        └──→INCF   9，1      ;
```

22、位测试，为 1 则跳指令
格式：　BTFSS　f，b
指令码：

0	11	bbb	fffff

指令周期：　1 或（产生跳转则为 2）
操作：　如果 f(b)=1 则跳(PC+1→PC)
影响状态位：　无
说明：　测试 f 寄有器的有第 b 位，如果位 b 为 1 则跳过下一条指令，否则顺序执行下去。
例：

```
        ┌──BTFSS   8，2      ;测试 F8 的 bit2，如果为 1
bit2=1 ┤   MOVF   5，0      ;则跳到 INCF   9，1
        └──→INCF   9，1
```

§2.5　常数和控制操作类指令

这类指令共有 11 条，其指令码结构为：

（11—8）	（7—0）
OPCODE	K

高 4 位是操作码，低 8 位是常数 K。

23、常数与指令
格式：　ANDLW　K
指令码：

1110	KKKKKKKK

指令周期：　1
操作：　W∧K→W
影响状态位：　Z
说明：　将 W 寄存器和常数 K 做逻辑与运算，结果存入 W。
例：　ANDLW　55H，0　　; W∧55H→W

24、子程序调周指令
格式：　CALL　K
指令码：

1001	KKKKKKKK

指令周期：　2
操作：　PC+1→堆栈;
K→PC(0－7)；‘0’→PC(8)；PA1，PA0→PC(10－9)。
影响状态位：　无

说明：　　将程序计数器 PC 加 1 后推入堆栈、将常数 K(程序地址的低 8 位)置入 PC，同时还把 PC 的第 9 位清为 0。对于 PIC1656/PIC16C57 来论， F3 的 PA1 和 PAO 两位(页面地址)还将被置入 PC 的最高二位(bit10，bit9)。所以子程序可以放在四个页面的任何一个中，但必须放在每页的上半部(低址区)，因为执行 CALL 指令将 PC 的第 9 位清为"0"。

```
例：        CALL    DELAY          ; 调用子程序 DELAY
            :
            :
    DELAY   MOVLW   80H ──┐
            :             │
            :             │ ;子程序的第一条指令须放在每页的上半区
            RETLW   0     ──┘
```

注：有关子程序调用的一些问题，请参考 § 2.7 程序设计技巧。

25、看门狗计数器清零指令

格式：　　CLRWDT

指令码：

0000	0000	0100

指令周期：　1

操作：　　0→WDT ，0→WDT 预设倍数

影响状态位：　1→TO，1→PD

说明：　　清除 WDT ，使之不能计时溢出。

26、无条件跳转指令

格式：　　GOTO　K

指令码：

101	KKKKKKKKK

指令周期：　2

操作：　　K→PC(8－0)，PA1，PA0→PC(10－9)

影响状态位：　无

说明：　　常数 K（地址）置入 PC 低 9 位。对于 PIC16C56/PIC16C57，F3 中的 PA1，PA0 两位页面地址位将同时置入 PC 的最高二位(bit1o，bit9)。所以 GOTO 指令可跳到程序区的任何地方去执行。

```
例：        GOTO    LOOP           ;跳转到 LOOP
             :
             :
    LOOP    MOVLW   10             ;
```

注：　关于跨页面跳转的问题，请参阅 § 2.7 程序设计技巧。

27、常数或指令

格式：　　IORLW　K

指令码：

1101	KKKKKKKK

指令周期：　1

操作：　　W∨K→W

影响状态位：　Z

说明：　　将 W 和常数 K 做逻辑或操作，结果存回 W。

```
例：        IORLW   80H            ;W∨80H→W
```

28、常数传送指令

格式：　　MOVLW　　K

指令码：

1100	KKKKKKKK

指令周期：　1
操作：　K→W
影响状态位：　无
说明：　把常数 K 置入 W 寄存器。注意这条指令并不影响任何状态位。
例：　MOVLW　0　;0→W，注意状态位 Z 不因之
　　MOVLW　88H　;而变化。

29、写 OPTION 寄存器指令
格式：　OPTION

指令码：			
	0000	000000	0

指令周期：　1
操作：　W→OPTION 寄存器
影响状态位：　无
说明：　将 W 内容置入 OPTION 寄存器
例：　MOVLW　07H　;7→W
　　OPTION　;OPTION=W=F

30、子程序返回指令
格式：　RETLW　K

指令码：		
	1000	KKKKKKKK

指令周期：　2
操作：　K→W ，堆栈→PC
影响状态位：　无
说明：　由子程序带参数 K 返回。返回参数存在 W 中。
例：　RETLW　0　;返回，0→W

31、进入低功耗睡眠状态指令
格式：　SLEEP

指令码：			
	0000	0000	0011

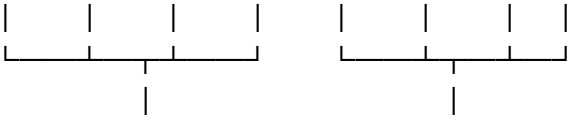
指令周期：　1
操作：　0→PD，1→TO;
　　0→WDT ，0→WDT 的预分频器
影响状态位：　PD，TO
说明：　停止 OSC1 的振荡，使 PIC 进入低功耗睡眠模式。这条指令还会清零 WDT 和预分频器（如果预分频器分配给 WDT 的话），并将 F3 的 PD 位清零，TO 位置 1。进入低功耗模式后，I/O 状态保持不变，WDT 清零后重新计时，一但计时溢出即将把 PIC 从 SLEEP 模式中唤醒（通过 RESET）。

32、设置 I/O 控制寄存器指令
格式：　TRIS　f

指令码：			
	0000	00000	fff

指令周期：　1
操作：　W→I/O 控制寄存器 TRISf(f=5，6，7)
影响状态位：　无
说明：　将 W 寄存器内容置入 I/O 控制寄存器，以设定 I/O 口的输入/输出方向。f=5，6，7，分别对应于 I/O 口 A，B，C。

例：　　MOVLW　0FH　　;W=0FH
　　TRIS　　6　　;TRISB=0FH
　　; RB7　RB6　RB5　RB4　RB3　RB2　RB1 RB0



输出　　输入

33、常数异或指令

格式：　　XORLW　k

指令码：

1111	kkkkkkkk

指令周期：　　W○K→W

影响状态位：　　Z

说明：　　将 W 和常数 K 逻辑异或运算，结果存入 W。

例：　　XORLW　33H　　;W○33H→W

§2.6　特殊指令助记符

PIC16C5X 的一些指令还可以用容易记忆的助记符来表示。PIC15C5X 的汇编程序 PICASM 可以认识这些助记符，在汇编时会将其转译成相应的 PIC16C5X 基本指令。

例如指令“BCF　3，0”（清零 C）也可以写成 CLRC，“BSF　3，0”（置 C=1）也可写成 SETC 等。

表 2.2 列出了这些助记符及其相对应的 PIC165X 指令。

二进制指令代码（Hex）	名称	助记符号	相对运算	状态影响
0100 0000 0011（403）	清除 C 标号	CLRC	BCF 3，0	-
0101 0000 0011（503）	设置 C 标号	SETC	BSF 3，0	-
0100 0010 0011（423）	清除辅助进位标号	CLRDC	BCF 3，1	-
0101 0010 0011（523）	设置辅助进位标号	SETDC	BSF 3，1	-
0100 0100 0011（443）	清除 0 标号	CLRZ	BCF 3，2	-
0101 0100 0011（543）	设置 0 标号	SETZ	BSF 3，2	-
0111 0000 0011（703）	进位则跳	SKPC	BTFSS 3，0	-
0110 0000 0011（603）	无进位则跳	SKPNC	BTFSC 3，0	-
0111 0010 0011（723）	辅助进位为 1 则跳	SKPDC	BTFSS 3，1	-
0110 0010 0011（623）	辅助进位为 0 则跳	SKPNDC	BTFSC 3，1	-
0111 0100 0011（743）	为 0 则跳	SKPZ	BTFSS 3，2	-
0110 0100 0011（643）	不为 0 则跳	SKPNZ	BTFSC 3，2	-
0010 001f ffff（22f）	测试寄存器	TSTF f	MOVF f，1	Z
0010 000f ffff（20f）	搬移寄存器到 W	MOVFW f	MOVF f，0	Z
0010 011f ffff（26f）	寄存器取补码	NEGF f，d	COMF f，1	Z
0010 10df ffff（28f）			INCF f，d	
0110 0000 0011（603） 0010 10df ffff（28f）	加进位到寄存器	ADDCF f，d	BTFSC 3，0 INCF f，d	Z
0110 0000 0011（603） 0000 11df ffff（0cf）	寄存器减进位	SUBCF f，d	BTFSC 3，0 DECF f，d	Z
0110 0010 0011（623） 0010 10df ffff（28f）	加辅助进位到寄存器	ADDDCF f，d	BTFSC 3，1 INCF f，d	Z

0110 0010 0011（623） 0000 11df ffff（0cf）	从寄存器减辅助进位	SUBD CF f，d	BTFSC 3，1	Z
101k kkkk kkkk（akk）	分支	B k	GOTO k	-
0110 0000 0011（603） 101k kkkk kkkk（akk）	依进位分支	BC k	BTFSC 3，0 GOTO k	-
0111 0000 0011（703） 101k kkkk kkkk（akk）	不进位分支	BMC k	BTFSS 3，0 GOTO k	-
0111 0000 0011（703） 101k kkkk kkkk（akk）	辅助进位为 1 分支	BDC k	BTFSC 3，1 GOTO k	-
0110 0100 0011（643） 101k kkkk kkkk（akk）	辅助进位为 0 分支	BNDC k	BTFSS 3，1 GOTO k	-
0111 0100 0011（743） 101k kkkk kkkk（akk）	0 分支	BZ k	BTFSC 3，2 GOTO k	-
0111 0100 0011（743） 101k kkkk kkkk（akk）	不为 0 分支	BNZ k	BTFSS 3，2 GOTO k	-

表 2.2 特殊指令助记符表

在后面的例子里，你将看到程序中使用了很多的特殊指令助记符。特殊指令助记符容易记忆。使用它程序可读性也较好。但这取决于每个人的习惯，你可以只使用一部分你认为好记的助记符，甚至只用基本的指令助记符而不用特殊指令助记符来编写程序。

§2.7 PIC16C5X 程序设计基础

上面我们已经详细介绍了 PIC16C5X 的每条指令。现在我们来总结一下它们的几个特点：

1、各寄存器的每一个位都可单独地被置位、清零或测试，无须通过间接比较，可节省执行时间和程序地址空间。

2、操作寄存器（F1－F7）的使用方法和通用寄存器的方法完全一样，即和通用寄存器一样看待。这样使程序执行和地址空间都简化很多。

3、对于跨页面的 CALL 和 GOTO 操作，要事先设置 F3 中的页面地址位 PA1、PA0，对于 CALL 来说，子程序返回后还要将 F3 的 PA1、PAO 恢复到本页面地址。

§2.7.1 程序的基本格式

先介绍二条伪指令：

（a）EQU——标号赋值伪指令　　（b）ORG——地址定义伪指令

PIC16C5X 一旦 RESET 后指令计数器 PC 被置为全“1”，所以 PIC16C5X 几种型号芯片的复位地址为：

型 号	复位向量
PIC16C52	17FH
PIC16C54/55	1FFH
PIC16C56	3FFH
PIC16C57/58	7FFH

表 2.3 复位地址表

一般说来，PIC 的源程序并没有要求统一的格式，大家可以根据自己的风格来编写。但这里我们推

荐一种清晰明了的格式供参考。

```
        TITLE     This    is.....                          ; 程序标题
        ;------------------
        ;名称定义和变量定义
        ;------------------
F0      EQU     0
RTCC    EQU     1
PC      EQU     2
STATUS  EQU     3
FSR     EQU     4
RA      EQU     5
RB      EQU     6
R7      EQU     7                                  ; 操作寄存器名称定义
                ┆                                  ; 变量定义
RESETADD EQU    17FH/1FFH/3FFH /7FFH    ; 对于 16C52－17FH
;---------------------- ------------------ ----------------    ; 对于 16C54/55-1FFH
        ORG     RESETADD                ; 对于 16C56－3FFH
        GOTO    MAIN                    ; 对于 16C57/58-7FFH
        ORG     0                       ; 从 000H 开始存放程序
;---------------- ------------------ -------- ----------------
;子程序区
;------------------------- -------------------------------
DELAY   MOVLW 255
        ┆
        RETLW  0
;------------------------- -------------------------------
;主程序区
;------------------------------ --------------------------
MAIN    CLRW
        TRIS    RB                      ;RB 已由伪指令定义为 6，即 B 口。
        CALLL   DELAY
        ┆
        END                             ;The  End  of  Program
```

注： MAIN 标号一定要处在 0 页面内。

另一些指令书写注意事项请参阅第五章“汇编程序”。

§2.7.2　程序设计基础

一、设置 I/O 口的输入/输出方向

PIC16C5X 的 I/O 口皆为双向可编程，即每一根 I/O 端线都可分别单独地由程序设置为输入或输出。这个过程由写 I/O 控制寄存器 TRIS　f 来实现，写入值为“1”，则为输入；写入值为“0”，则为输出。

```
MOVLW   0FH             ;00001111 (0FH)
                          ↑   ↑
                          |   └────输入
                          └────────输出
TRIS 6                  ;W 中的 0FH 写入 B 口控制器，B 口高 4 位为输出，低 4
                          位为输入 。
MOVLW   0C0H            ;11000000 (C0H)
MOVWF   6                 ↑ ↑
                          | └──────RB4，RB5 输出 0
                          └────────RB6，RB7 输出 1
```

二、检查寄存器是否为零

如果要判断一个寄存器内容是否为零，很简单：

```
            MOVF    10，1       ;F10→F10，结果影响状态位 Z
        ┌── SKPZ                  ;F10 为零则跳(Z=1 否)
 Z=1    │   GOTO    NZ          ;F10 不为零
（F10=0）└──→:
              :
              :
NZ            MOVLW   55H
              :
              :
```

三、比较二个寄存器的大小

要比较二个寄存器的大小，可以将它们做减法运算，然后根据状态位 C 来判断。注意，相减的结果放入 W，则不会影响二寄存器原有的值。

例如 F8 和 F9 二个寄存器要比较大小：

```
        MOVF    8，0        ;F8→W
        SUBWF   9，0        ;F9－F8→W
        SKPNZ               ;判断 Z=1 否（即 F9=F8 否）
        GOTO    F8=F9       ;F9=F8
        SKPNC               ;C=0 则跳
        GOTO    F9>F8       ;C=1，相减 1 结果为正，F9>F8
        GOTO    F8>F9       ;C=0，相减结果为负，F8>F9
         :
         :
```

四、循环 n 次的程序

如果要使某段程序循环执行 n 次，可以用一个寄存器作计数器。下例以 F10 做计数器，使程序循环 8 次。

```
            COUNT   EQU   10          ;定义 F10 名称为 COUNT（计数器）
          :
          :
            MOVLW   8               ;循环次数→COUNT

            MOVWF   COUNT
┌──→LOOP    CLRW                    ;循环体
│         :
│         :
│   ┌──── DECFSZ  COUNT，1        ;COUNT 减 1，结果为零则跳
└───┼──── GOTO    LOOP                ;结果不为零，继续循环
    └──→  :                       ;结果为零，跳出循环
          :
```

五、“IF.....THEN.....”格式的程序

下面以“IF X=Y　THEN　GOTO　NEXT”格式为例。

```
        X   EQU   ××
        Y   EQU   ××            ;X，Y 值由用户定义（变量）
          :
          :
```

```
            MOVLW     X
            MOVWF     10              ;X→F10
            MOVLW     Y               ;Y→W
            SUBWF     10，0               ;X－Y→W
        ┌── SKPNZ                     ;X=Y 否
X≠Y     │   GOTO      NEXT            ;X=Y，跳到 NEXT 去执行。
        └→  :
            :
```

六、“FOR.....NEXT”格式的程序

“FOR.....NEXT”程序使循环在某个范围内进行。下例是“FOR X=0 TO 5”格式的程序。F10 放 X 的初值，F11 放 X 的终值。

```
            START     EQU     10
            END'      EQU     11
             :
             :
            MOVLW     0
            MOVWF     START           ;0→START(F10)
            MOVLW     5
            MOVWF     END'            ;5→END'(F11)

┌──→LOOP  :                             ;循环体
│           :
│           INCF      START，1        ;START 值加 1
│           MOVF      START，O
│           SUBWF     END'，0         ;START=END'?(X=5 否)
│ X=5 ┌── SKPZ
└─────┼── GOTO      LOOP            ;X<5，继续循环
      └→  :                         ;X=5，结束循环
          :
```

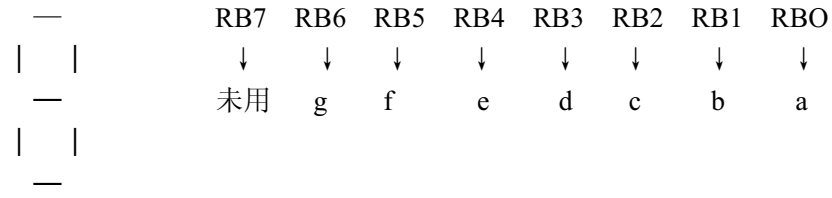
七、“DO WHILE.....END”格式的程序

“DO WHILE.....END”程序是在符合条件下执行循环。下例是“DO WHILE X=1”格式的程序。F10 放 X 的值。

```
            X         EQU       10
                      :
                      :
                      MOVLW     1
                      MOVWF     X           ;1→X(F10)，作为初值
        ┌──→LOOP      :
        │             :
        │                MOVLW     1
        │                SUBWF     X，0
        │ X≠1 ┌──      SKPNZ                 ;X=1 否?
        └─────┼──      GOTO      LOOP        ;X=1 继续循环
              └→       :                     ;X≠1，跳出循环
                       :
```

八、查表程序

查表是程序中经常用到的一种操作。下例是将十进制 0~9 转换成 7 段 LED 数字显示值。若以 B 口的 RB0~RB6 来驱动 LED 的 a~g 线段，则有如下关系：



设 LED 为共阳，则 0－9 数字对应的线段值如下表：

PIC 的查表程序可以利用子程序带值返回的特点来实现。具体是在主程序中先取表数据地址放入 W，接着调用子程序，子程序的第一条指令将 W 置入 PC，则程序跳到数据地址的地方，再由“RETLW”指令将数据放入 W 返回到主程序。

十进数	线段值	十进数	线段值
0	COH	5	92H
1	G9H	6	82H
2	A4H	7	F8H
3	BOH	8	80H
4	99H	9	90H

表 2.4　0－9 LED 线段值

下面程序以 F10 放表头地址。

```
        MOVLW   TABLE           ;表头地址→F10
        MOVWF   10
         :
         :
        MOVLW   1               ;1→W，准备取“1”的线段值
        ADDWF   10，1           ;F10+W=“1”的数据地址
        CALL    CONVERT
        MOVWF   6               ;线段值置到 B 口，点亮 LED。
         :
         :
CONVERT  MOVWF   2                   ;W→PC
TABLE    RETLW   C0H            ;“0”线段值
        RETLW   F9H             ;“1”线段值
        :
        :
       RETLW   90H             ;“9”线段值
```

九、“READ.....DATA，RESTORE”格式程序

“READ.....DATA”程序是每次读取数据表的一个数据，然后将数据指针加 1，准备下一次取下一个数据。下例程序中以 F10 被数据表起始地址，F11 做数据指针。

```
POINTER  EQU     11             ;定义 F11 名称为 POINTER
        :
        :
        MOVLW   DATA
        MOVWF   10              ;数据表头地址→F10
            CLRF        POINTER     ;数据指针清零
        :
        :
        MOVF    POINTER，0
        ADDWF       10，0           ;W=F10+POINTER
        :
        :
```

```
            INCF        POINTER，1      ;指针加 1
            CALL        CONVERT         ;调子程序，取表格数据
            :
            :
CONVERT     MOVWF       2               ;数据地址→PC
DATA        RETLW       20H             ;数据
            :
            RETLW       15H             ;数据
```

如果要执行“RESTORE”，只要执行一条“CLRF　POINTER”即可。

十、延时程序

如果延时时间较短，可以让程序简单地连续执行几条空操作指令“NOP”。如果延时时间长，可以用循环来实现。下例以 F10 计算，使循环重复执行 100 次。

```
                MOVLW     100
                MOVWF     10
┌→LOOP  ┌─── DECFSZ     10，1       ;F10－1→F10，结果为零则跳
└──────┼─── GOTO        LOOP
       └──→  :
             :
```

延时程序中计算指令执行的时间和即为延时时间。如果使用 4MH 振荡，则每个指令周期为 1uS。所以单周期指令执行时间为 1uS，双周期指令为 2uS。在上例的 LOOP 循环延时时间即为：（1+2）*100+2=302（uS）。在循环中插入空操作指令即可延长延时时间：

```
                MOVLW     100
                MOVWF     10
┌──→ LOOP       NOP
│               NOP
│                  NOP
│     ┌─────     DECFWZ    10，1
└─────┼──        GOTO      LOOP
      └──→       :
                 :
```

延时时间=（1+1+1+1+2）*100+2=602（US）。

用几个循环嵌套的方式可以大大延长延时时间。如下例用 2 个循环来做延时。

```
                      MOVLW     100
                MOVWF     10          ;第一计数值(100)→F10
┌→LOOP            MOVLW     16
│                 MOVWF     11          ;第二计数值(16)→F11
│ ┌→LOOP1         DECFSZ    11，1
│ └─────────      GOTO      LOOP1
│           ┌──── DECFSZ    10，1
└───────────┼──── GOTO      LOOP2
            └────→ :
                   :
```

延时时间= [（1+2）*6+2+]+1+2+1+1 +*100+2=5502（US）

十一、RTCC 计数器的使用

RTCC 是一个脉冲计数器，它的计数脉冲有二个来源，一个是从 RTCC 引脚输入的外部信号，一个是内部的指令时钟信号。可以用程序来选择其中一个信号源做为输入。RTCC 可被程序用作计时之用：程序读取 RTCC 寄存器值以计算时间。当 RTCC 作为内部计时器使用时需将 RTCC 管腿接ＶDD 或ＶSS，以减少干扰和耗电流。

下例程序以 RTCC 做延时。

```
        RTCC        EQU         1
                    :
                    :
                    CLRF        RTCC        ;RTCC 清 O
                    MOVLW       07H
                    OPTION                  ;选择预设倍数 1：256→RTCC

┌──→LOOP      MOVLW       255             ;RTCC 计数终位值
│               SUBWF       RTCC，O
│       ┌──    SKPZ                        ;RTCC=255?
└───────┼──    GOTO        LOOP
        │       :
        └──→    :
```

这个延时程序中，每过 256 个指令周期 RTCC 寄存器增 1（Prescaler=1：256），设芯片使用 4MHz 振荡，则：

延时时间=256*256=65536（US）

RTCC 是自振式的，在它计数时，程序可以去做别的事，只要隔一段时间去读取它，检测它的计数值即可。所以用 RTCC 做延时，在延时期间，程序还可以做别的事。这是一般用软件来延时做不到的。

十二、寄存器体（Bank）的寻址

对于 PIC16C52/54/55/56，寄存器有 32 个，只有一个体（bank），故不存在体寻址问题，对于 PIC16C57 来说，寄存器则有 80 个，分为 4 个体（bank0-bank3）。在§1.5.1 中对 F4（FSR）的叙述中已有提及。F4 的 bit6 和 bit5 是寄存器体寻址位，其对应关系如下：

bit6	bit5	Bank	物理地址
0	0	Bank0	10H-1FH
0	1	Bank1	30H-3FH
1	0	Bank2	50H-5FH
1	1	Bank3	70H-7FH

当芯片 RESET 后，F4 的 bit6、bit5 都被清零，所以是指向 Bank0。

下面的例子对 Bank1 和 Bank2 的 30H 及 50H 寄存器写入数据。

例 1：（设目前体选为 Bank0）

```
BSF     4，5            ;置位 Bit5=1，选择 Bank1
MOVLW   55H
MOVWF   10H         ;55H→30H 寄存器（Bank1）
BCF     4，5        ;Bit5=0
BSF     4，6        ;Bit6=1，选择 Bank2
MOVWF   10          ;55H→50 寄存器（Bank2）
```

从上例中我们看到，对某一体（Bank）中的寄存器进行读写，首先要先对 F4 中的体寻址位进行操作。当然，芯片 RESET 后自动选择 Bank0（bit6=0、bits=0），所以如果复位后读写，不需对 PA1 和 PA0 再操作。只有当对 Banko 以外的寄存器体读写，才需先置 PA1 和 PA0 为相应的值。

注意，在例子中对 30H 寄存器（Bank1）和 50H 寄存器（Bank2）写数时， 用的指令“MOVWF 10H”中寄存器地址写的都是“10H”，而不是读者预期的“MOVWF 30H”和“MOVWF 50H”， 为什么?

让我们回顾一下指令表。在 PIC16C5X 的所有有关到寄存器的指令码中，寄存寻址位都只占 5 个位：fffff，因而只能寻址 32 个（00H－1FH）寄存器。所以要选址 80 个寄存器，还要再用二位体选址位 PA1 和 PA0。当我们设置好体寻址位 PA1 和 PA0，使之指向一个 Bank，那么指令“MOVWF 10H”就是将 W 内容置入这个 Bank 中的相应寄存器内（10H、30H、50H 或 70H）。

有些用户第一次接触体选址的概念，难免理解上有出入，下面是一个例子：

例 2：（设目前体选为 Bank0）

```
MOVLW   55H
MOVWF   30H          ；欲把 55H→30H 寄存器
MOVLW   66H
MOVWF   50H          ；欲把守 6H→50H 寄存器
```

以为“MOVWF 30H”一定能把 W 置入 30H，“MOVWF 50H”一定能把 W 置入 50H，这是错误的。因为这两条指令的实际效果是“MOVWF 10H”，原因上面已经说明过了。所以例 2 这段程序实际上的效果是：1、55H→10H 寄存器；2、66H→10H 寄存器。最后结果是 F10H=66H，而真正的 F30H 和 F50H 并没有被操作到。

讲到这，也许还是有些读者对体寻址感到很麻烦。那么下面我们给出一个建议和例子。

建议：为使体选址的程序清晰明了（对别人和对自己），建议多用名称定义符来写程序，则不易混淆。

例 3：假设在程序中用到 Bank0、Bank1 和 Bank2 的几个寄存器如下：

Bank0	地址	Bank1	地址	Bank2	地址	Bank3	
A	10H	B	30H	C	50H	-	70H
-	-		-	-	-	-	-
-	-		-	-	-	-	-
-	-		-	-	-	-	-

```
A       EQU     10H          ；Bank0
B       EQU     10H          ；Bank1
C       EQU     10H          ；Bank2
        :
        :
FSR     EQU     4
Bit6    EQU     6
Bit5    EQU     5
        :
        :
        MOVLW   55H
        MOVWF   A            ；55H→F10H
        BSF     FSR，Bit5    ；Bit5=1
        MOVWF   B            ；55→F30H
        BCF     FSR，Bit5    ；Bit5=0
        BSF     FSR，Bit6    ；Bit6=1
        MOVWF   C            ；55→F50H
        :
        :
```

程序这样书写，相信体选址就不容易错了。

十三、程序跨页面跳转和调用

在§1.4“程序存贮器”，我们已经谈了 PIC16C5X 的程序存贮区的页面概念和 F3 寄存器中的页面选址位 PA1 和 PA0 两位。下面我们来看实例。

1、“GOTO”跨页面

例：设目前程序在 0 页面（Page0），欲用“GOTO”跳转到 1 页面的某个地方 KEY（Page1）。

```
                :
                :
STATUS      EQU     3
PA1         EQU     6
PA0         EQU     5
                :
                    :
跨页跳转        BSF     STATUS，PA0         ；PA0=1，选择 page 页面

┌──────       GOTO    PAGE1               ；跳转到 1 页面的 KEY
│               :
│               :
└→ KEY        CLRW
                :
                :
```

2、“CALL”跨页面

例：设目前程序在 0 页面（Page0），现要调用—“-”（注：横线）放在 1 页面（Page1）的子程序 DELAY。

```
                :
                :
跨页调用        BSF     STATUS，PA0         ；PA0=1，选择 1 页面
┌──────       CALL    DELAY               ；跨页调用
│                   BCF     STATUS，PA0         ；PA0=0，恢复 0 页面地址!!!
│               :
│               :
│   ; ──────────────────────────
│   ;DELAY 子程序
│   ; ──────────────────────────
└──→DELAY   :
                :
```

注意：程序为跨页 CALL 而设了页面地址，从子程序返回后一定要恢复原来的页面地址。

3、程序跨页跳转和调用的编写

读者看到这里，一定要问：我写源程序（.ASM）时，并不去注意每条指令的存放地址，我怎么知道这个 GOTO 是要跨页面的，那个 CALL 是需跨页面的？

问得好！的确，你开始写源程序时并不知道何时会发生跨页面跳转或调用，不过当你将源程序用 MPASM 汇编时，它会告诉你。当汇编结果显示出：

×××（地址）“GOTO　out　of　Range”

×××（地址）“CALL　out　of　Range”

这表明你的程序发生了跨页面的跳转和调用，而你的程序中在这些跨页 GOTO 和 CALL 之前还未设置好相应的页面地址。这时你应该查看汇编生成的.Lst 文件，找到这些 GOTO 和 CALL，并查看它们要跳转去的地址处在什么页面，然后再回到源程序（.ASM）做必要的修改。一直到你的源程序汇编通过（0 Errors　and　Warnnings）。

4、程序页面的连接

程序 4 个页面连接处应该做一些处理。一般建议采用下面的格式：

地址	指令	
000H	:	Page0(PA1=PA0=0)
:	:	
1FFH		
200H	BSF STATUS，PA0	
201H	:	
:	:	
:	:	Page1(PA1=0，PA0=1)
3FFH	:	
400H	BCF STATUS，PA0	
401H	BSF STATUS，PA1	
:	:	Page2(PA1=1，PA0=0)
:	:	
5FFH	:	
600H	BSF STATUS，PA0	
:	:	
:	:	Page3(PA1=1.PA0=1)
7FFH	GOTO MAIN	

即在进入另一个页面后，马上设置相应的页面地址位（PA1、PA0）。

页面处理是 PIC16C5X 编程中最麻烦的部分，不过并不难。只要做了一次实际的编程练习后，就能掌握了。

第三章　PIC16C5X 系统扩展方法

PIC16C5X 系列单片机在芯片内包括了 CPU,ROM,RAM,I/O 等部件，一片 PIC 即是一完整的微电脑系统。但是在实际应用中，你也可以对它做一些扩充，以满足需要。

§3.1　I/O 的扩展

如果在应用中状态输入/输出较多，可以对 PIC 的 I/O 口做一些扩展。如图 3.1 所示，PIC16C5X 的 B 口即做 8 路输出口，又做 8 路输入口。

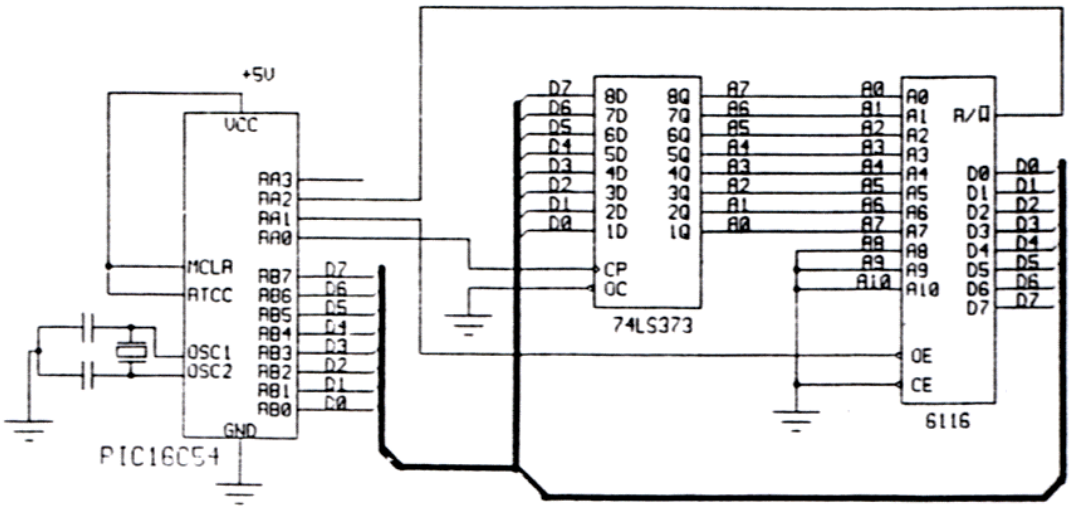


图 3.1　PIC16C54I/O 扩展

设 PIC16C54 从 244 读入 B 口，再从 B 口输出到 374，则程序如下：

```
CLRW
TRIS        5        ；置 RA 为输出口
BCF         5,1      ；置 374CP 端为 0
BSF         5,0      ；置 244G 端为 1
MOVLW       OFFH
TRIS        6        ；置 RB 为输入口
BCF         5,0      ；打开 244（G=0）
NOP
MOVF        6,0      ；RB→W
BSF         5,0      ；关闭 244（G=1）
MOVWF       8        ；把 RB 口内容→F8
CLRW
TRIS        6        ；置 RB 口为输出态
MOVF        8,0      ；F8→W
MOVWF       6        ；W→RB
BSF         5,1
NOP
```

```
BCF     5,1     ；给 374CP 端一锁存脉冲，把
                ；RB 口状态锁入 374。
```

I/O 口的扩展方法很灵活，可根据需要、成本等因素决定怎么做。

§3.2 数据存储器的扩展

PIC16C5X 内部只有 32-80 个数据存储器。所以如果在实际应用中需要大量的 RAM 来存储一些数据（如中间运算结果，测试数据等等），则可外接 RAM 来解决。

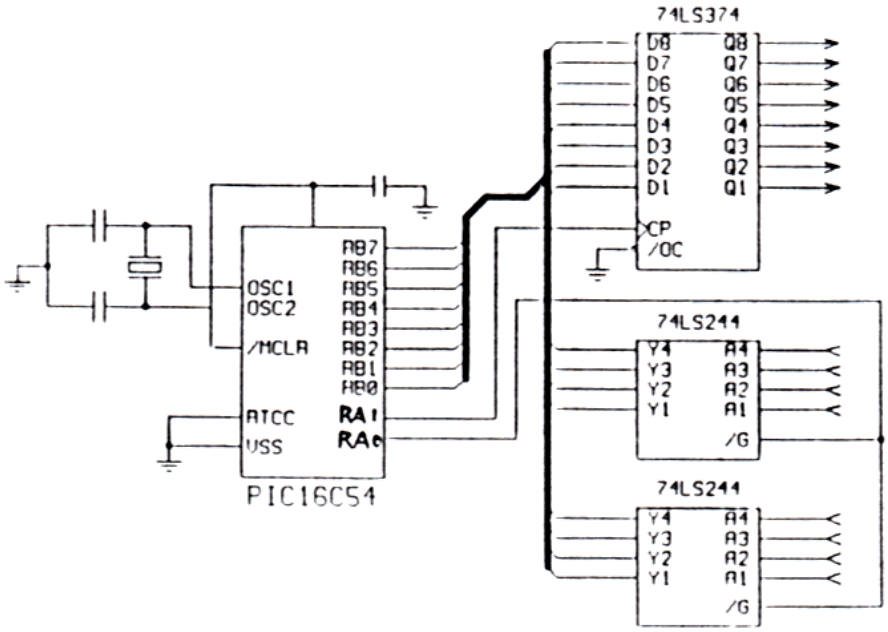


图 3.2　PIC16C5X 数据存储器扩展

假设要从 RAM 的 FFH 单元读出一个数据，再写入 RAM 的 FEH 单元，则程序如下：

```
MOVLW   0
TRIS    6       ；置 RB 为输出态
CLRW
TRIS    5       ；置 RA 为输出态
MOVLW   06
MOVWF   5
MOVLW   0FFH    ；地址 FFH→W
MOVWF   6       ；W→RB
BSF     5,0     ；373LE=1，使地址数据
NOP             ；加到 6116 地址线上。
BCF     5,0
MOVLW   0FFH
TRIS    6       ；置 RB 为输入口
BCF     5,1     ；6116  OE=0，读 RAM
NOP
BSF     5,1     ；OE=1
```

```
MOVE    6,0          ；将 RAM FFH 单元读入 W
MOVWF   8            ；F8=RAM（FFH）
CLRW
TRIS    6            ；置 RB 口为输出态
MOVLW   0FEH
MOVWF   6            ；地址值 FEH→RB
BSF     5,0
NOP
BCF     5,0          ；地址值 FE→6116 地址线上
MOVF    8,0          ；F8→W，F8 为 RAM FFH 单元内容。
MOVWF   6            ；（FFH）→RB 口
BCF     5,2          ；R/W=0
BCF     5,1
NOP
BSF     5,1          ；加 6116 之 OE=0，写 RAM。
BSF     5,2          ；相当于把（FFH）→（FEH）
```

以上这段 RAM 读/写程序经适当修改即可做为一子程序，供主程序需要操作外接 RAM 时调用。同理，读者也可以外接一片 SEEPROM（如 93C××，24C××等），请参阅第四章的实例应用。

第四章　PIC16C5X 设计实例

前面几章已对 PIC16CXX 的硬件构成、指令系统、程序设计基础及系统扩充做了详细的叙述，相信你已经对 PIC16CXX 芯片有了相当的认识，下面几章的主要目的是帮助大家实际应用 PIC。

§4.1　开发步骤流程

当决定选择 PIC 来实现设计后，一般可以采取下面的开发步骤来完成：

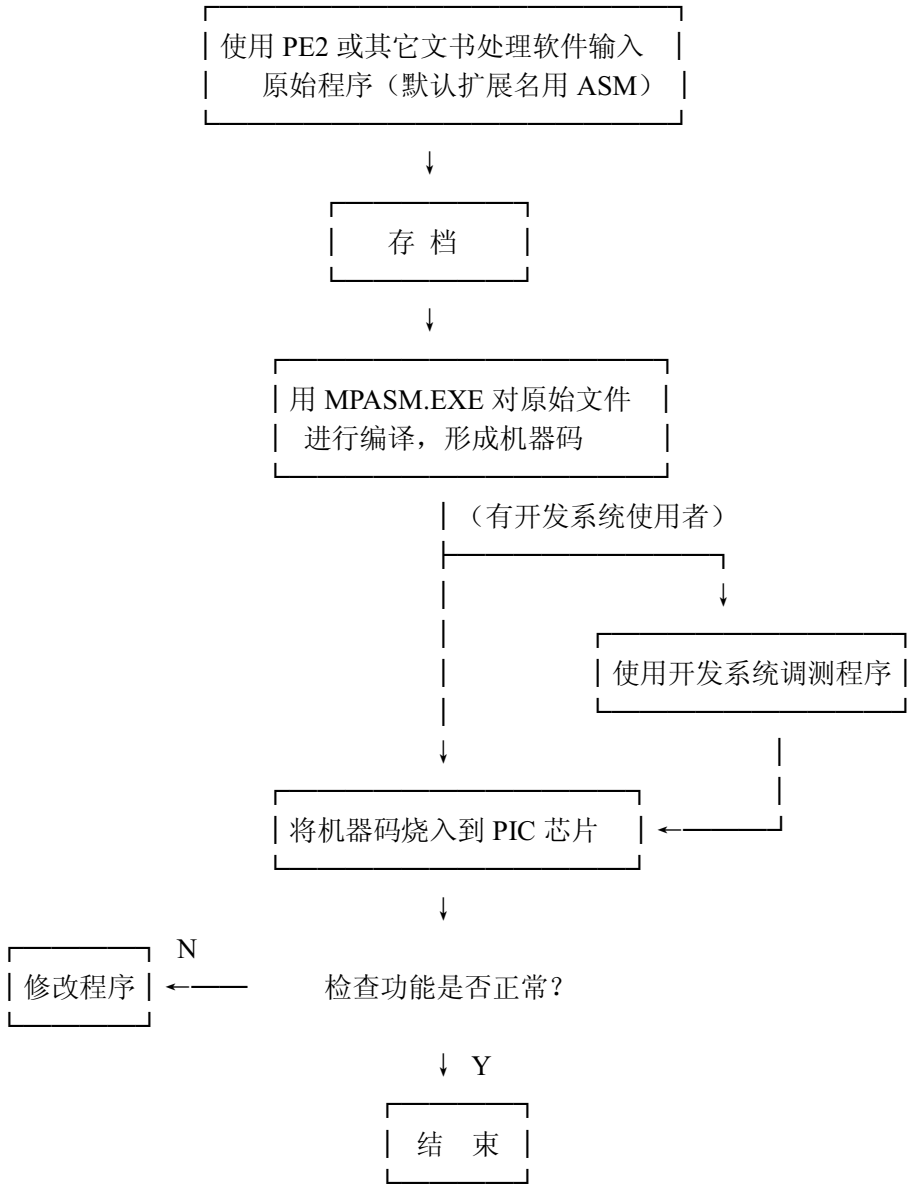


图 1.1　PIC 开发步骤

§4.2　设计实例

下面的这些实例都不启用 WDT。读者如果需要，可以自己加上。这些实例仅向读者展示 PIC16C5X 的一些基本应用，当然其中很多亦具实用价值。

一、数码管 LED 显示（一）

本例要用 PIC 来实现一位 LED 显示，可显示 0H－FH 的十六进制数。程序中用一个计数器由 0 计数到 F，同时将其在 LED 上显示出来。该例给出计数值与显示码的转换方法。

1、电路设计：

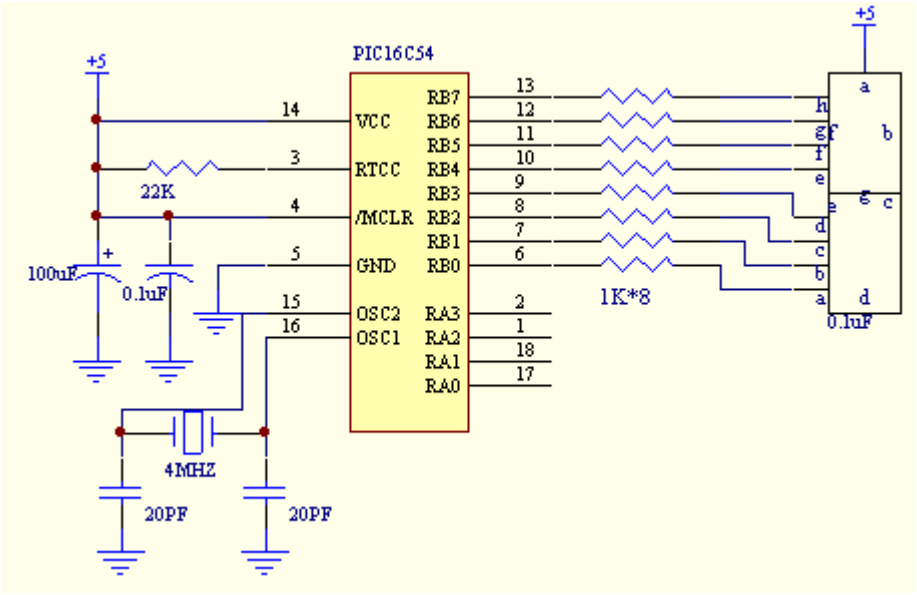
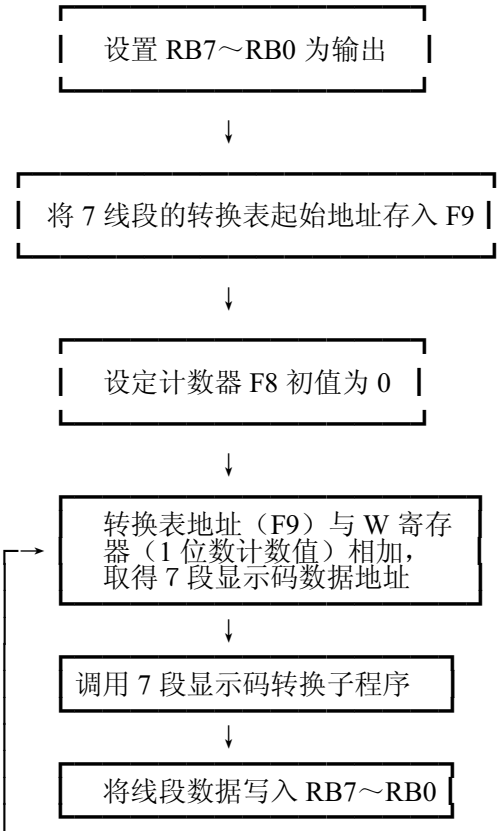
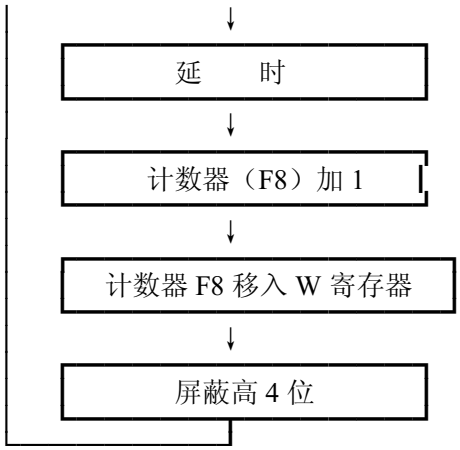


图 1.2　数字显示（一）电路

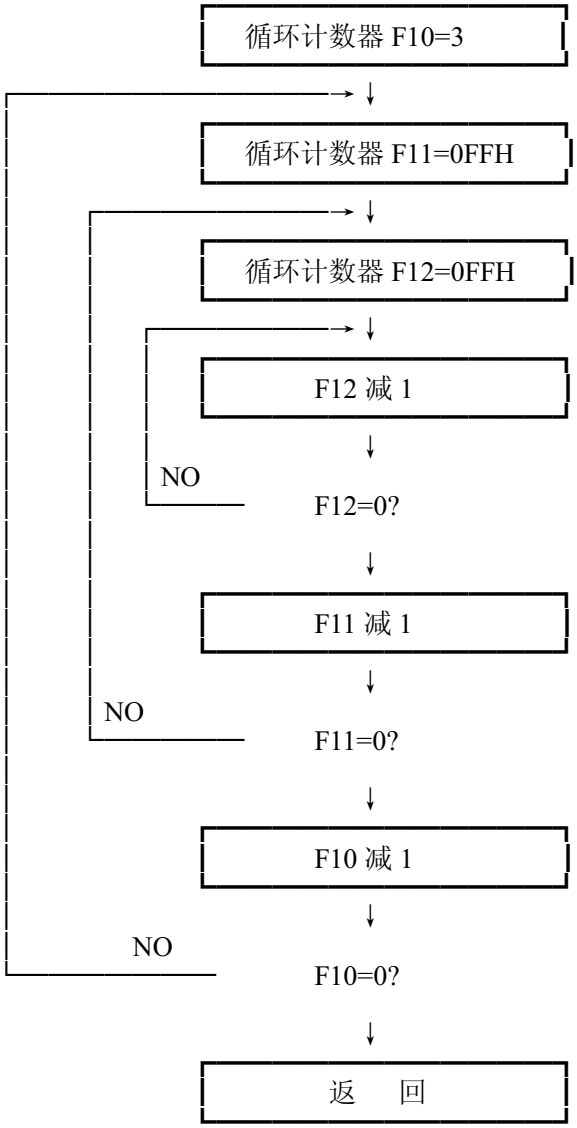
电路中为 PIC16C54 的 RB 口来直接驱动一共阳 LED。RB7－RB0 分别通过限流电阻接到 LED 的 a－g 及 P 段上。

2、程序流程图：

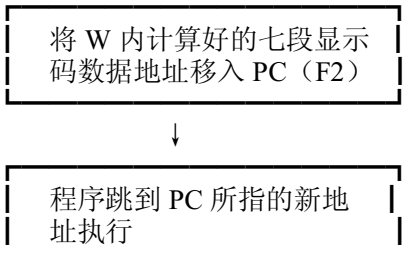


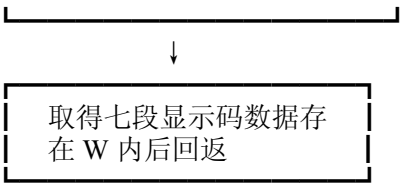


延时子程序：



线段转换子程序：





3、程序清单：

```
PC          EQU         2
RB          EQU         6
COUNTER     EQU         8               ;变量单元定义
TABADD      EQU         9
COUNT0      EQU         10
COUNT1      EQU         11
COUNT2      EQU             12
PIC54       EQU         1FFH            ;PIC16C54 复位地址
SUB         EQU         0               ;子程序存放起始地址(一般从 0000H 开始)
;---- ------------------------------------------------
                ORG         PIC54
                GOTO        MAIN
;---- ------------------------------------------------
                ORG         SUB
;----- -----------------------------------------------
DELAY       MOVLW       3               ;设置延时常数
                MOVWF       COUNT0
L1          MOVLW       255
                MOVWF       COUNT1
L2          MOVLW       255
                MOVWF       COUNT2
L3          DECFSZ      COUNT2          ;递减循环
                GOTO        L3
            DECFSZ      COUNT1
                GOTO        L2
                DECFSZ      COUNT0
            GOTO        L1
                RETLW       0
;------------------- ---------------------------------
CONVERT     MOVWF       PC              ;将 W 寄存器内的 7 段显示码地址放入 PC
TABLE                                       ;PC 执行新地址指令， 跳到相对的地址执行
                                            ;RETLW 指令，将七段显示码存入 W 后返回
                RETLW       0C0H            ;0
                RETLW       0F9H            ;1
                RETLW       0A4H            ;2
                RETLW       0B0H            ;3
                RETLW       099H            ;4
                RETLW       092H            ;5
```

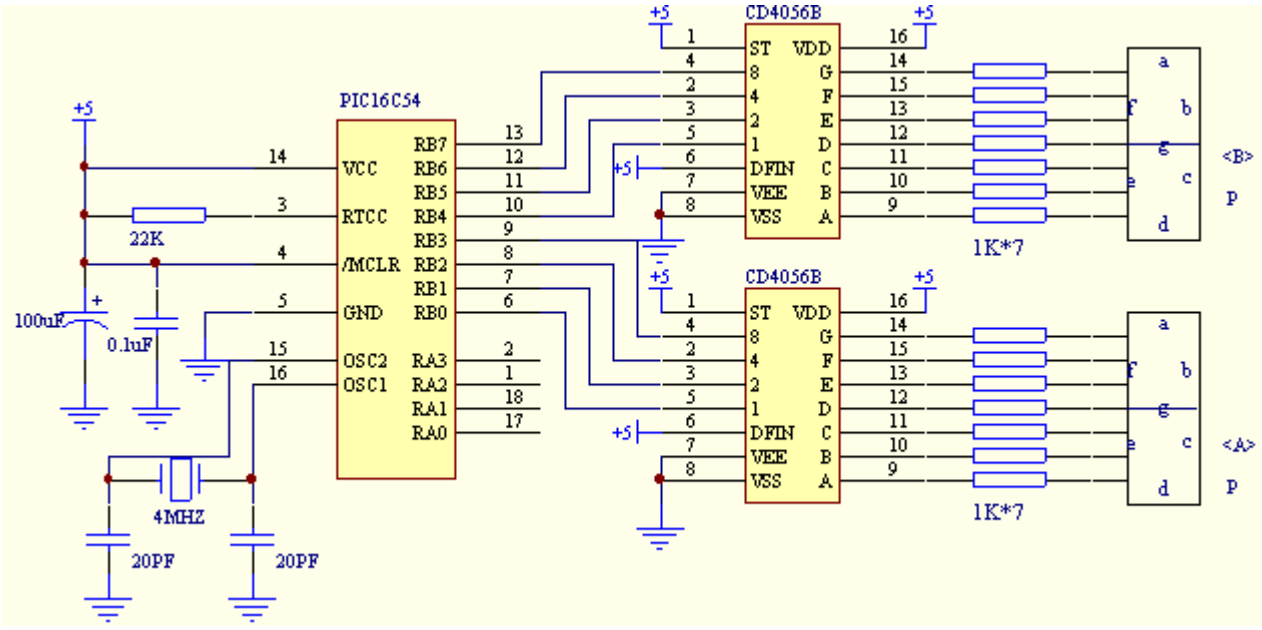
```
                RETLW   082H            ;6
                RETLW   0F8H            ;7
                RETLW   080H            ;8
                RETLW   090H            ;9
                RETLW   088H            ;A
                RETLW   083H            ;B
                RETLW   0C6H            ;C
                RETLW   0A1H            ;D
                RETLW   086H            ;E
                RETLW   08EH            ;F(线段值)
;----- ------------------------------------------------
MAIN        MOVLW   0               ;设置 RB7～RB0 为输出口
                TRIS    R
                MOVLW   TABLE
                MOVWF   TABADD      ;将转换表的首地址存入 TABADD
                CLRF    COUNTER     ;计数器清 0
                CLRW
LOOP        ADDWF   TABADD，W       ;计数值(W)与转换表的起始地址相加
                CALL    CONVERT     ;存入 W 后调用转换表子程序
                MOVWF   RB          ;取出的七段显示码送 RB 口显示
                CALL    DELAY
                INCF    COUNTER     ;计数器加 1 递增(依次显示 0，1，2，…F)
                MOVF    COUNTER，W
                ANDLW   0FH
                GOTO    LOOP        ;循环显示
;-- ------------------------------------------------------
                END
;------------------- ---------------------------------------
```

二、数码管 LED 显示（二）

本例说明显示二位 LED 的方法。二位 LED 显示如果象上例用 I/O 直接驱动，则需二个 I/O 口。如用七段解码器相辅，仅需一个 I/O 口。程序中使用一个寄存器（F20）作计数器，然后将计数结果输出显示。因为 BCD 解码器只能作 0－9 的十进制输出，所以计数器 F20 需转换成十进制再输出，程序中将涉及到将十六进制转成十进制 BCD 码的方法。

1、电路设计：

PIC16C54 将二位数的 BCD 码输出给 BCD 解码器 CD4056B，再由其转换成七段码输出驱动 LED。

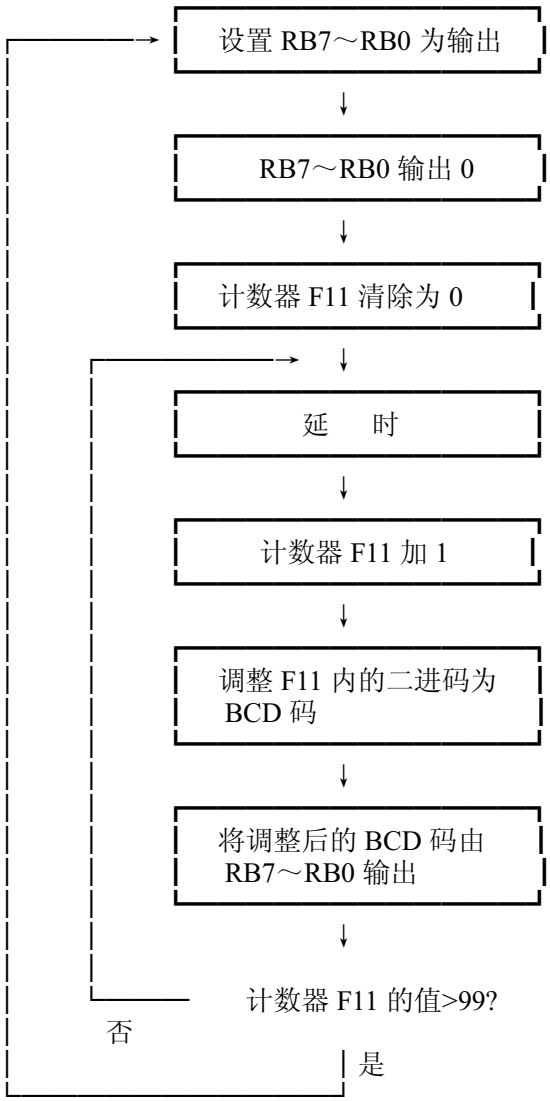


CD4056B 的芯片数据请查阅有关手册。

1．3　数字显示（二）电路

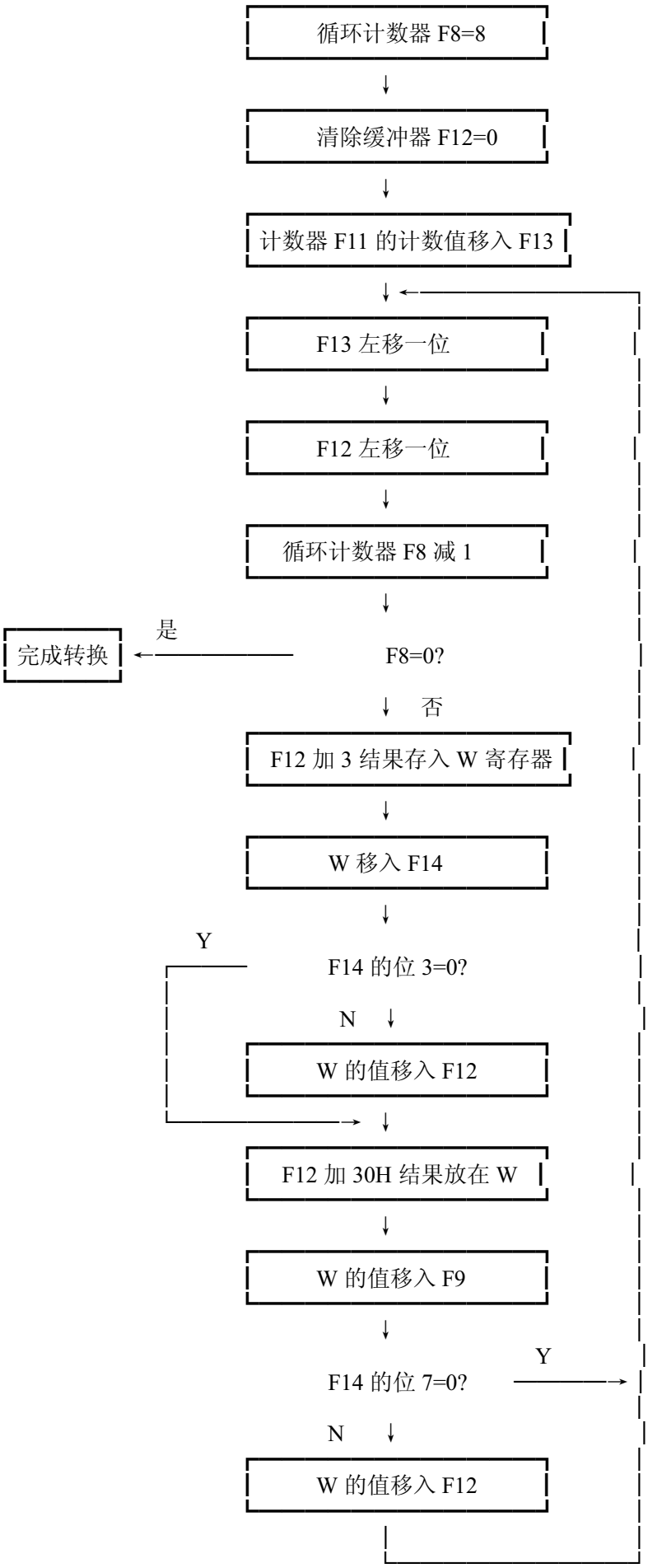
2、流程图

主程序：

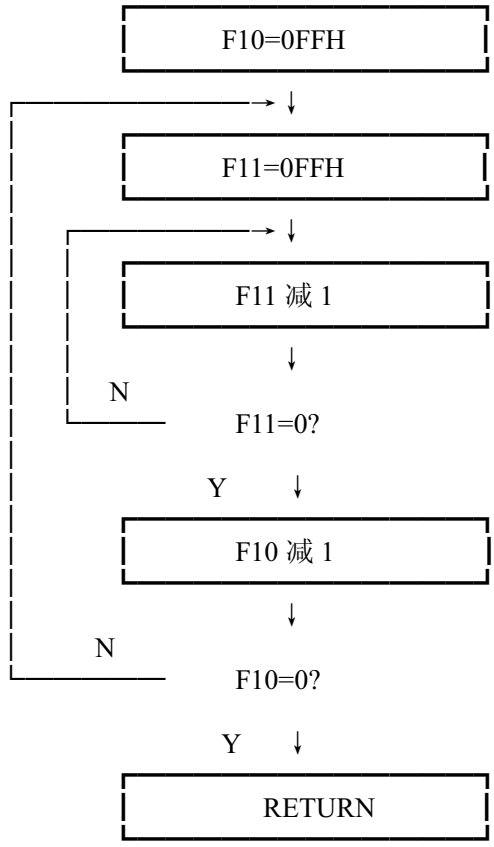


调整程序（ADJ）：

F20 中的二进制调整为 BCD 码的方法是将二进制码左移 8 次，每次移位后都检查低四位 LSD+3 是否大于 7，如果是则再加 3，否则不加。接着再将高四位 MSD 作相同处理。



延时程序：



3、程序清单：

```
PC          EQU         2
RB          EQU         6
COUNTER     EQU         8
COUNT0      EQU         9
COUNT1      EQU         10
DIGIT       EQU         11
BUFFER      EQU         12
BUFFER1     EQU             13
SUM         EQU         14
PIC54       EQU         1FFH
SUB         EQU             0
;-------- ----------------------------------
                ORG         PIC54
                GOTO        MAIN
; ----------------------------------------
                ORG         SUB
;-- ----------------------------------------
DELAY
                MOVLW       255
                MOVWF       COUNT0          ;设置延时常数
L1          MOVLW       255
                MOVWF       COUNT1
```

```
L2          DECFSZ      COUNT1
            GOTO        L2
            DECFSZ      COUNT0
            GOTO        L1
            RETLW       0
;-- ----------------------------------------
MAIN
            MOVLW       0
            TRIS        RB              ;设置 RB 口为输出
            CLRF        RB              ;置 RB 口为 0
            CLRF        DIGIT           ;显示计数器清 0
LOOP1       CALL        DELAY
            INCF        DIGIT
            MOVLW       8
            MOVWF       COUNTER         ;循环计数器置 8
            CLRF        BUFFER
            MOVF        DIGIT，W
            MOVWF       BUFFER1
LOOP2       RLF         BUFFER1         ;二进制数转换成 BCD 码（以便送 4056 显示）
            RLF         BUFFER
            DECFSZ      COUNTER
            GOTO        ADJUST
            MOVF        BUFFER，W
            MOVWF       RB              ;是否有进位（即已达 100），如已计数到 100 则
            SKPC                        ;从 0 开始再计数显示（显示 00～99）
            GOTO        LOOP1
            GOTO        MAIN
ADJUST      MOVLW       3               ;二进制转 BCD 的调整
            ADDWF       BUFFER，W       ;见框图
            MOVWF       SUM
            BTFSC       SUM，3
            MOVWF       BUFFER
            MOVLW       30H
            ADDWF       BUFFER，W
            MOVWF       SUM
            BTFSC       SUM，7
            MOVWF       BUFFER
            GOTO        LOOP2
;- ----------------------------------------
            END
;---------------------- --------------------
```

三、按键显示

本例要 PIC 接收一小健盘输入（0－7 数字），再将其用 LED 显示出来。按健扫描程序若扫描到二个健同时被按下，则不承认其结果。健盘是单片机应用中经常用到的，用户可以加上“去抖动”等处理，也可以用各种方法扩充按健的数目。

1、电路设计：

RB0－RB3 置为输出态，RB4－RB7 置为输入态构成健盘矩阵。扫描到按健按下后，将其健值通过

74LS47 输到 LED 显示。

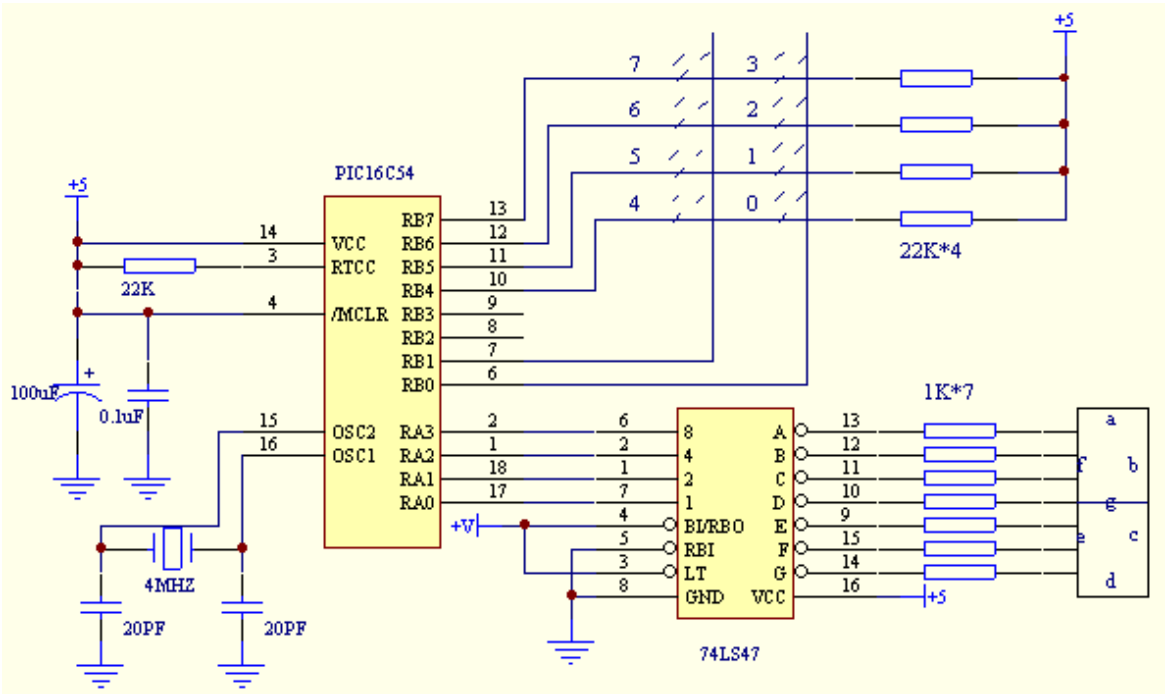
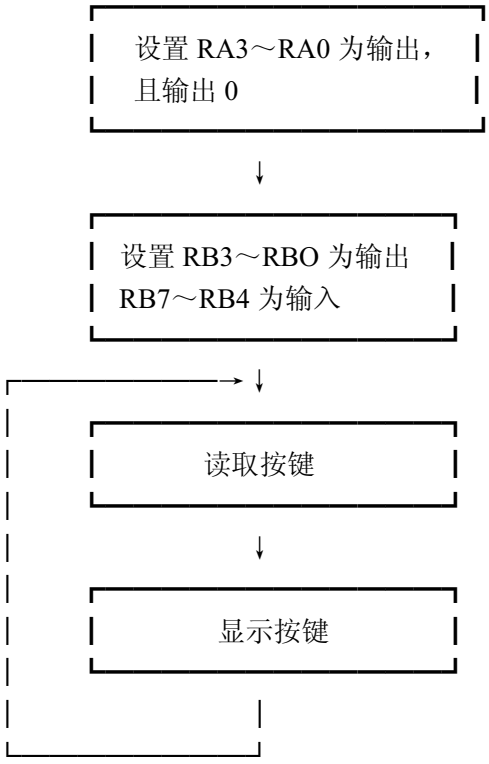


图 1.4　按健显示电路

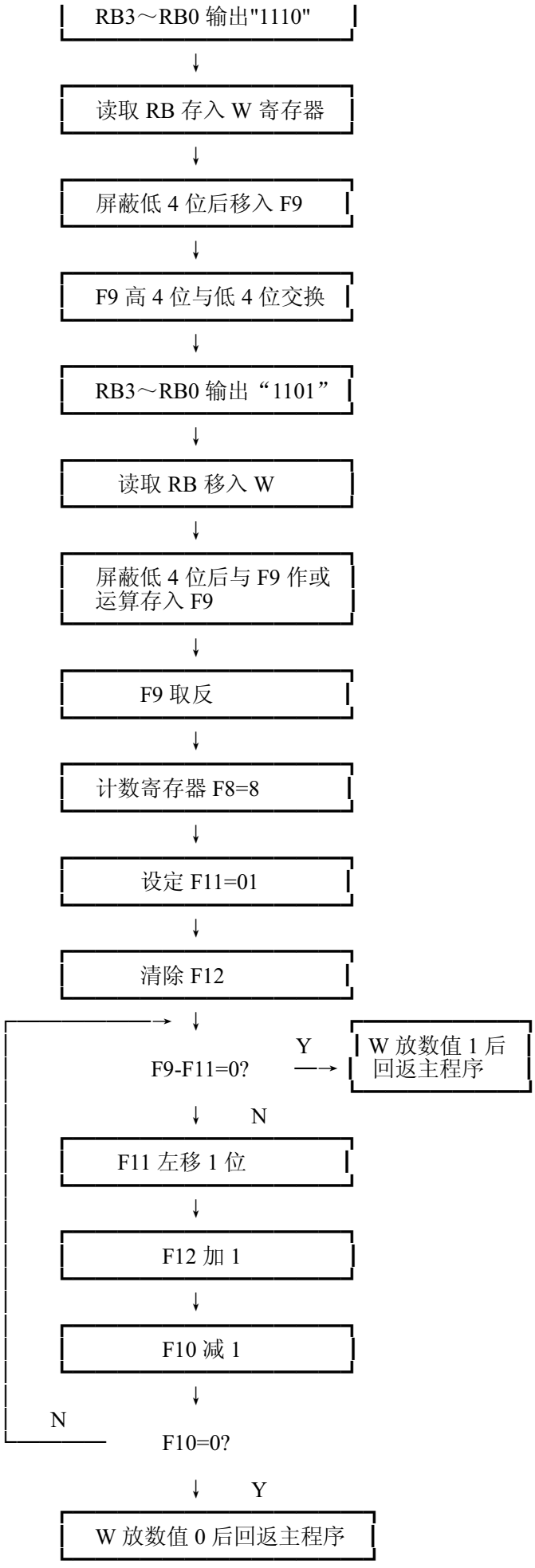
关于 74LS47 的芯片资料请参阅有关手册。

2、程序流程图：

主程序：



键盘扫描程序：


RB3～RB0 输出"1110"
读取 RB 存入 W 寄存器
屏蔽低 4 位后移入 F9
F9 高 4 位与低 4 位交换
RB3～RB0 输出“1101”
读取 RB 移入 W
屏蔽低 4 位后与 F9 作或运算存入 F9
F9 取反
计数寄存器 F8=8
设定 F11=01
清除 F12
F9-F11=0?
Y
W 放数值 1 后回返主程序
N
F11 左移 1 位
F12 加 1
F10 减 1
N
F10=0?
Y
W 放数值 0 后回返主程序

3、程序清单：

```
PC          EQU         2
RA          EQU         5
RB          EQU         6
KEYIN       EQU         7                   ;按键输入存储单元
COUNTER     EQU         8
BUFFER      EQU         9
BUFFER1     EQU             10
PIC54       EQU         1FFH
SUB         EQU         0
; ----------------------------------------
                ORG         PIC54
                GOTO        MAIN
;- ----------------------------------------
                ORG         SUB
;- ----------------------------------------
SCANKEY
                MOVLW       B'11111110'     ;RB0 列扫描
                MOVWF       RB
                MOVF        RB，W
                ANDLW       B'11110000'
                MOVWF       BUFFER
                SWAPF       BUFFER
                MOVLW       B'11111101'     ;RB1 列扫描
                MOVWF       RB
                MOVF        RB，W
                ANDLW       B'11110000'
                IORWF       BUFFER
                COMF        BUFFER
                MOVLW       8
                MOVWF       COUNTER
                CLRC
                MOVLW       1
                MOVWF       BUFFER1
                CLRF        KEYIN
LOOP1       MOVF        BUFFER，W       ;按键值计算
                SUBWF       BUFFER1，W
                SKPNZ
                RETLW       1
                RLF         BUFFER1
                INCF        KEYIN
                DECFSZ      COUNTER
                GOTO        LOOP1
                RETLW       0
;- ----------------------------------------
MAIN
                MOVLW       0
                TRIS        RA
                CLRF        RA
                MOVLW       B'11110000'
                TRIS        RB
LOOP        CALL        SCANKEY         ;调用键盘扫描读取程序
                ANDLW       0FH
```

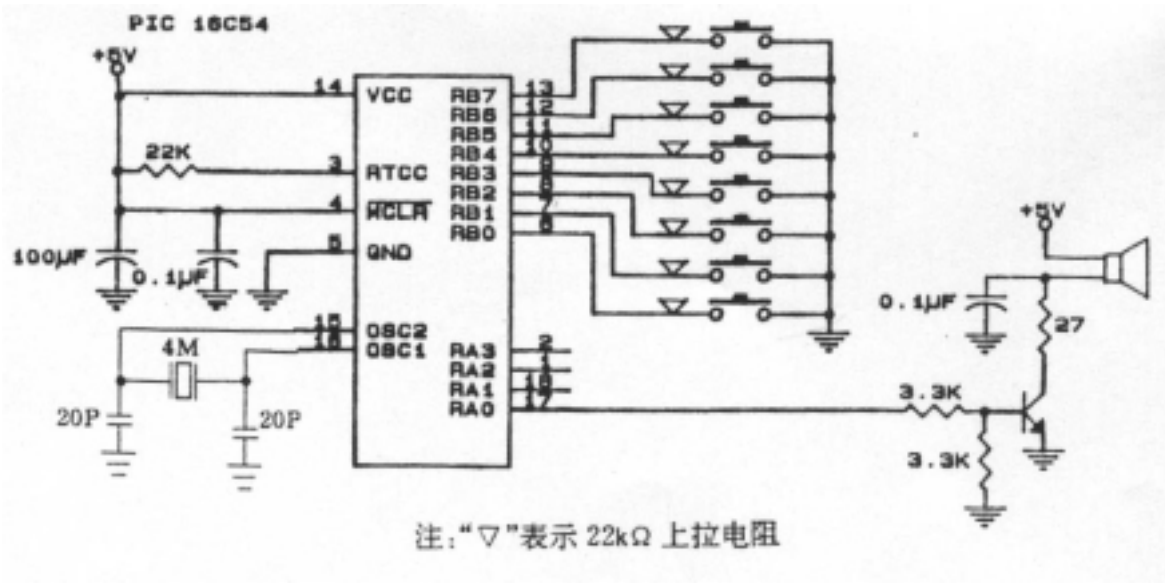
```
                SKPNZ
                GOTO        LOOP
                MOVF        KEYIN，W
                MOVWF       RA                  ;按键值送 RA 口驱动 LED 显示
                GOTO        LOOP
;------ ------------------------------------
                END
;-- ----------------------------------------
```

四、电子音阶

本例让 PIC 扫描几个按健的输入，同时将按健转换成相应的声音频率发音。每种频率的声音其延时时间不同。发音的延时程序使用 RTCC 计时器计时。

1、电路设计：

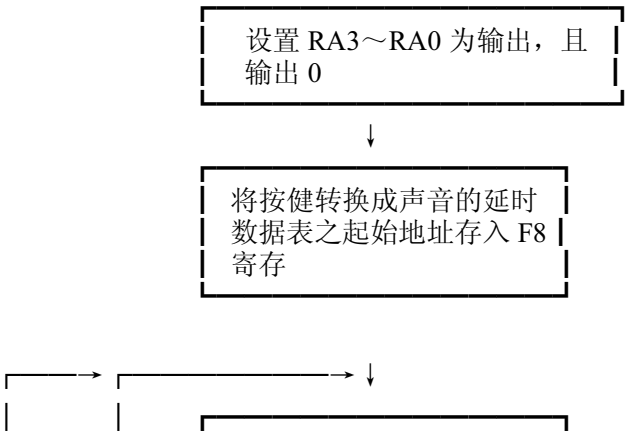
如同所示，RB7－RB0 设置为输入态，同时用上拉使之为高电平，当按健按下时则降为低电平。BA0 用来驱动喇叭发声。

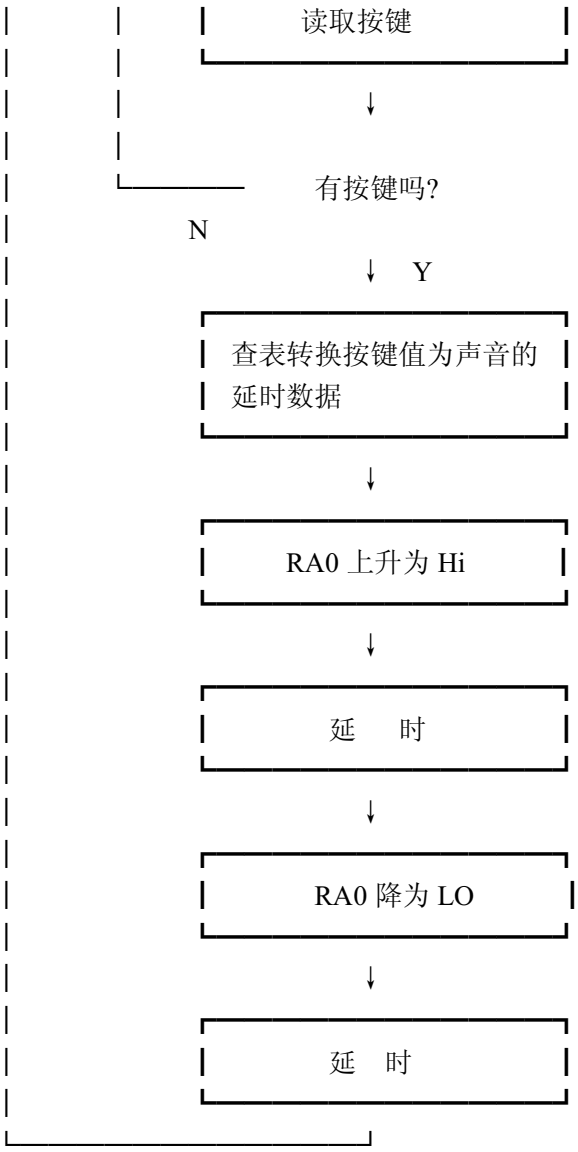


1.5　电子琴电路

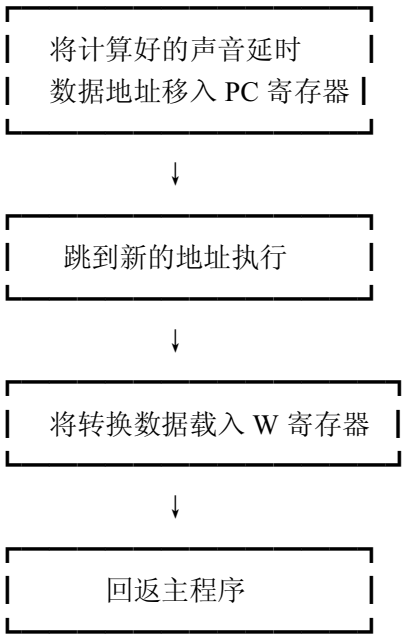
2、程序流程图；

主程序：

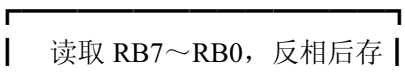


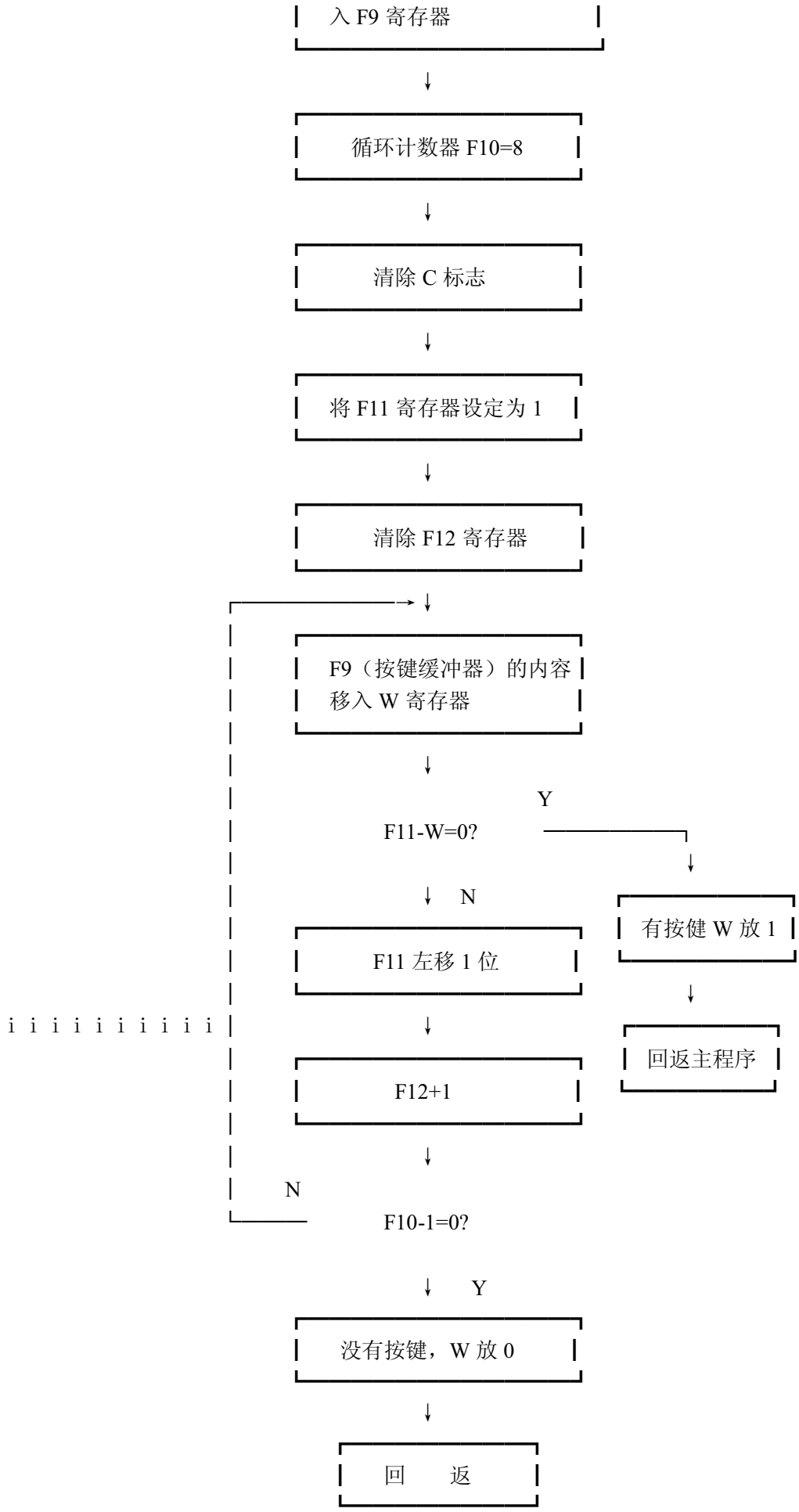


声频转换子程序：

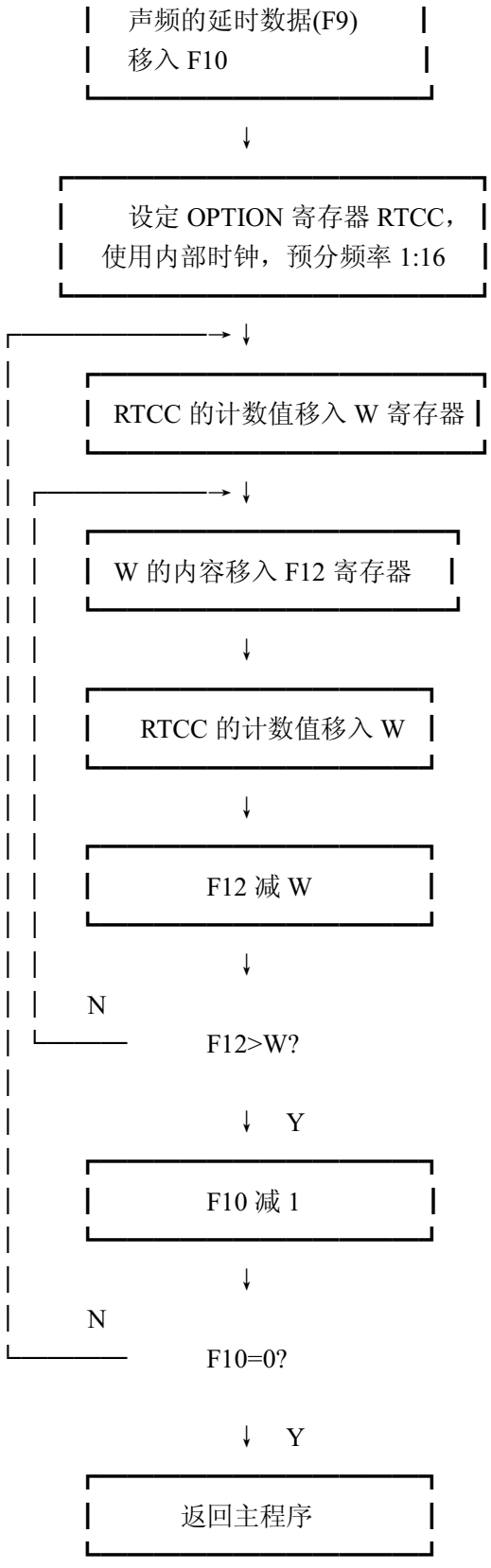


读取按键子程序：




入 F9 寄存器
循环计数器 F10=8
清除 C 标志
将 F11 寄存器设定为 1
清除 F12 寄存器
F9（按键缓冲器）的内容
移入 W 寄存器
Y
F11-W=0?
N
有按健 W 放 1
F11 左移 1 位
回返主程序
F12+1
N
F10-1=0?
Y
没有按键，W 放 0
回 返

延时子程序：



3、程序清单：

```
RTCC        EQU         1
PC          EQU         2
RA          EQU         5
RB          EQU         6
KEYIN       EQU         7
```

```
COUNTER    EQU        8
BUFFER     EQU        9
BUFFER1    EQU        10
TABADD     EQU        11
PIC54      EQU        1FFH
SUB        EQU            0
;- - ---------- ----------
           ORG        PIC54
           GOTO       MAIN
; - ----------------------------------------
           ORG        SUB
;-- - ----------------------------------------
CONVERT    MOVWF      PC              ;声调(频率)数据转换
TABLE
           RETLW      106             ;1
           RETLW      94              ;2
           RETLW      85              ;3
           RETLW      79              ;4
           RETLW      71              ;5
           RETLW      64              ;6
           RETLW      56              ;7
           RETLW      53              ;i
SCANKEY
           COMF       RB，W           ;按键扫描读取程序
           MOVWF      BUFFER
           MOVLW      8
           MOVWF      COUNTER
           CLRC
           MOVLW      1
           MOVWF      BUFFER1
           CLRF       KEYIN
L1         MOVF       BUFFER，W
           SUBWF      BUFFER1，W
           SKPNZ
           RETLW      1
           RLF        BUFFER1
           INCF       KEYIN
           DECFSZ     COUNTER
           GOTO       L1
           RETLW      0
DELAY                              ;发音频率程序
           MOVF       BUFFER，W       ;以频率数据作为延时常数
           MOVWF      COUNTER
           MOVLW      3
           OPTION
```

```
L2          MOVF      RTCC，W
L3          MOVWF     BUFFER1
            MOVF      RTCC，W
            SUBWF     BUFFER1
            SKPNC
            GOTO      L3
            DECFSZ    COUNTER
            GOTO      L2
            RETLW     0
;- - ----------------------------------------
MAIN
            MOVLW     0
            TRIS      RA
            CLRF      RA
            MOVLW     TABLE
            MOVWF     TABADD
LOOP
            CALL      SCANKEY       ;调按键扫描读取子程序
            MOVWF     BUFFER
            MOVF      BUFFER，W
            SKPNZ                   ;是否有按键输入
            GOTO      LOOP          ;否
            MOVF      KEYIN，W      ;是
            ADDWF     TABADD，W     ;读取发音频率数据
            CALL      CONVERT
            MOVWF     BUFFER
            BSF       RA，0         ;发音
            CALL      DELAY
            BCF       RA，0
            CALL      DELAY
            GOTO      LOOP          ;循环
;--------- - --------------------------------
            END
;- - ----------------------------------------
```

五、交流电电源控制

使用单片机做强电控制是单片机经常要涉及的事情。通常单片机要控制的强电设备都是使用 220V AC，单片机的 I/O 输出和其相比小得多，不能直接驱动这些电路。所以要用单片机做强电控制，一般采取二种方法：（一）用晶体管入大控制信号；（二）使用光电藕合元件。光藕的输入和输出完全隔离，可防干扰。输入端只要用小电流驱动一个 LED 作 ON/OFF 就可以控制输出。如果输出端的光电元件是 SCR 或 TRIAC（可控硅）等交流元件时，就可以直接控制输出端的 AC 电流。本例要用 PIC 来控制交流电灯泡的明灭。

1、电路设计：

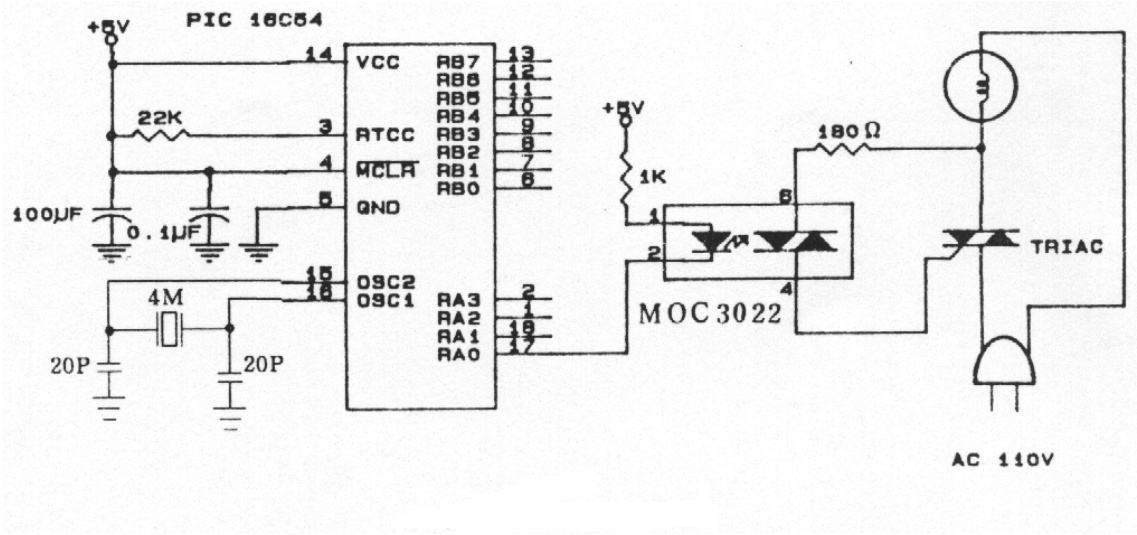


图 1.6　AC 电源控制

用 PIC16C54 的 RA0 输出控制光藕 MOC3022 输入端的 LED 作 ON/OFF 转换，以控制输出端的交流电源。MOC3022 输入端的 LED 点亮常态电流是 5MA，所以使用 1K 电阻连至+5V 电源。程序使 RA0 不断的输出高→低→高的电平变换，于是 TRIAC 控制的灯泡即作明灭的闪动。注意因为灯泡是靠钵钨丝加热发光的，所以闪动的速度不能太快，必须使钨丝有足够的加热时间。

如果负载是感性元件，则 TRIAC 的电路如下：

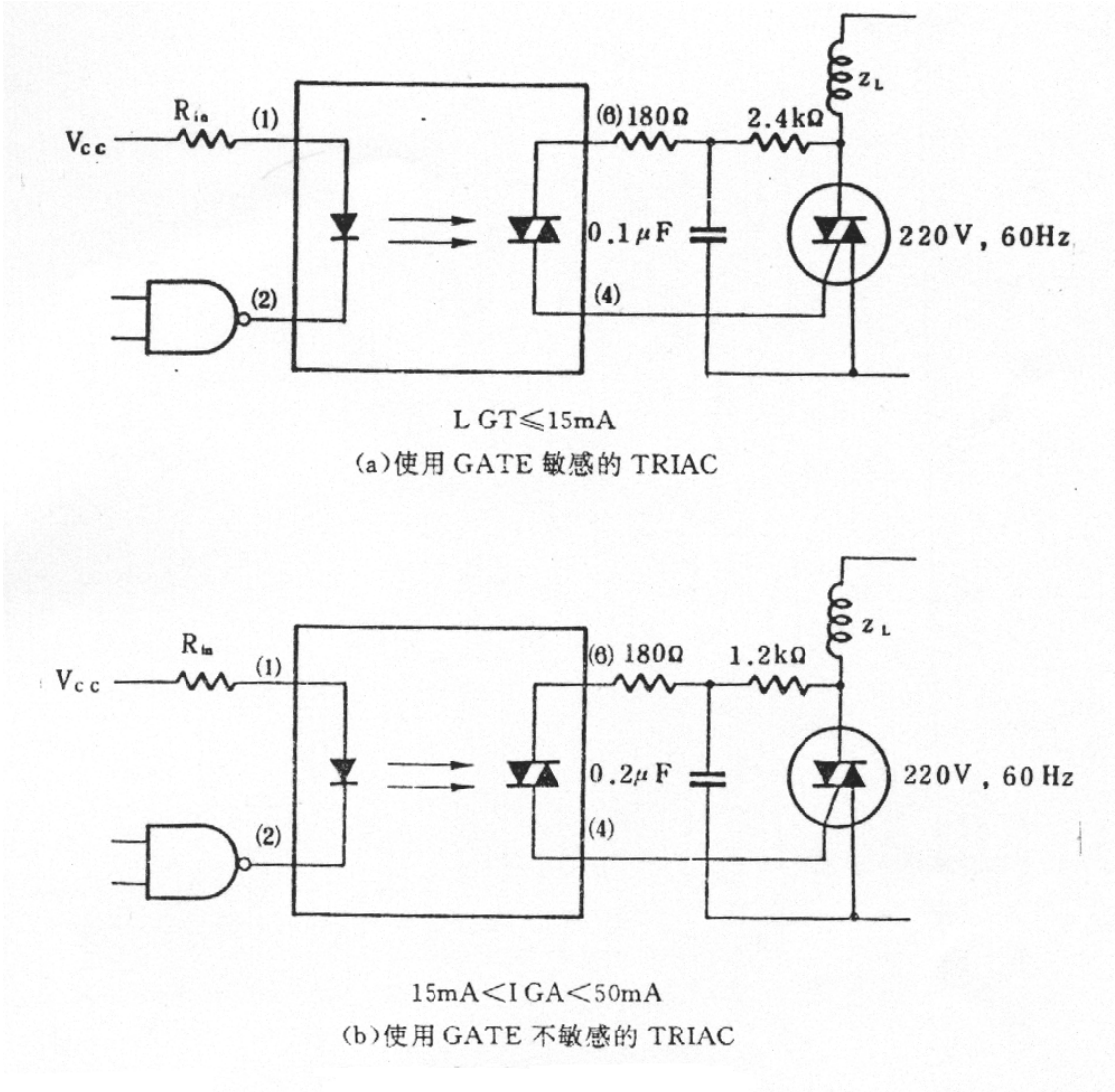
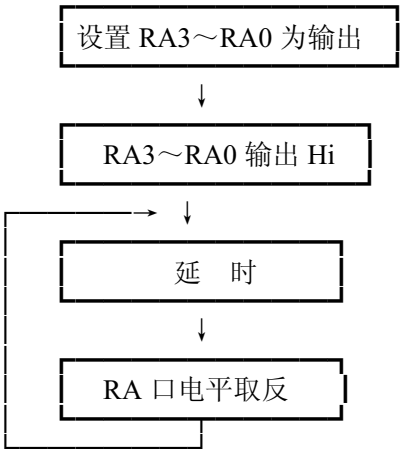


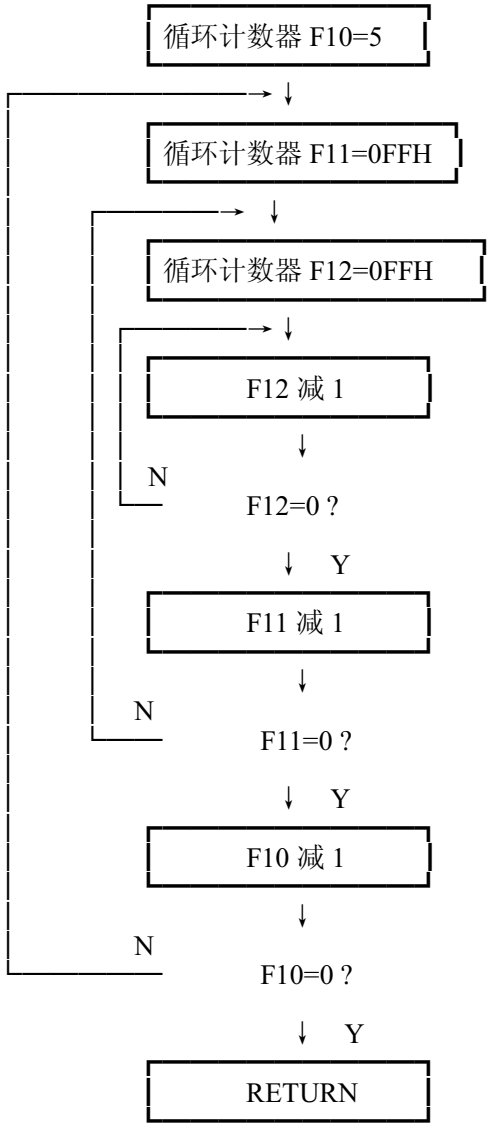
图 1.7　感性负载的 TRIAC 电路

2、程序流程图：

主程序：



延时子程序：



3、程序清单：

```
RA          EQU         5
COUNT0      EQU         8
COUNT1      EQU         9
COUNT2      EQU         10
PIC54       EQU         1FFH
```

```
SUB             EQU             0
;- ----------------------------------------
                ORG             PIC54
                GOTO            MAIN
;-- ----------------------------------------
                ORG             SUB
;--------- ------------------------------------
DELAY
                MOVLW       5
                MOVWF       COUNT0
L1          MOVLW       255
                MOVWF       COUNT1
L2          MOVLW       255
                MOVWF       COUNT2
L3          DECFSZ      COUNT2
                GOTO        L3
                DECFSZ      COUNT1
                GOTO        L2
                DECFSZ      COUNT0
                GOTO        L1
                RETLW       0
;----- ----------------------------------------
MAIN        MOVLW       0
                TRIS        RA
                BSF         RA，0
LOOP        CALL        DELAY           ;闪动延时
                COMF        RA              ;亮—灭—亮—灭…控制
                GOTO        LOOP
;-- ----------------------------------------
                END
;------ --------------------------------------
```

六、电力线参数测量

在某些应用中（如功率因素调整、电力测量、电力线监视等），我们需要测量电力线（AC Power Line）的一些参数，如零相位、频率、相对相位等等。

这里我们介绍一种利用 PIC I/O 口的特点而设计的电力线检测电路，其特点是极其简单：只要用一个外部电阻，并且比那些使用电容或笨重的变压器的方法更可靠。见图 1.8，PIC 的所有端口都有静态保护电路。这些保护电路的作用是当 PIC 的输入端口被加上一超压信号时，就会将其和电源线（VDD 或 VSS）短路。这样就能使其免受静电脉冲的危害。每个保护电路使用两个大 P－N 结二极管。这些二极管起钳位作用，可以经受几个毫安的电流。所以只要处于电流安全值内，高压信号可以直接加到 PIC 的输入口上。

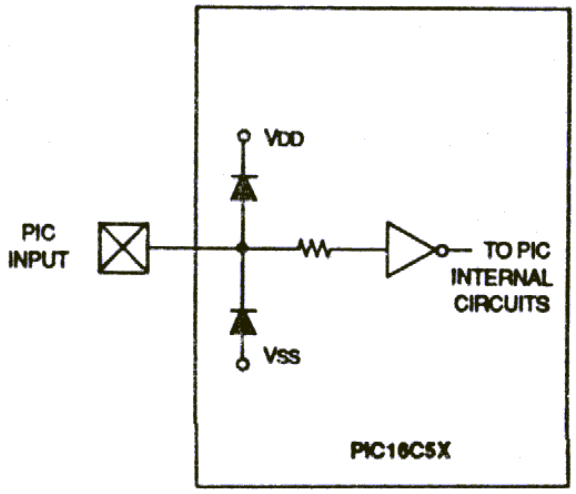


图 1.8　PIC16C5X 输入保护电路

基于上述原理我们可以用一个高值电阻把电力线和 PIC 输入口连接起来如图 1.9 所示：

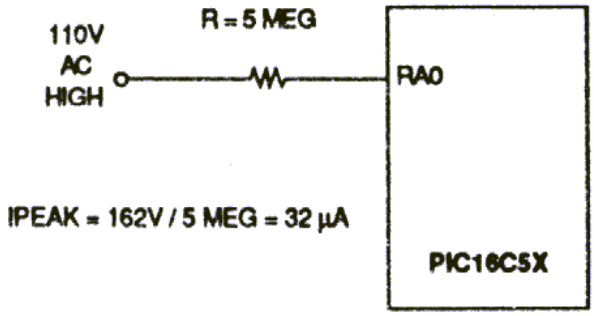


图 1.9　PIC 限流电阻连接图

电力线电压通过限流电阻加到 PIC 输入口，再由 PIC 输入口的保护电路的二极管钳位输入到 PIC 内部。一个典型的输入波形如图 1.10 所示：

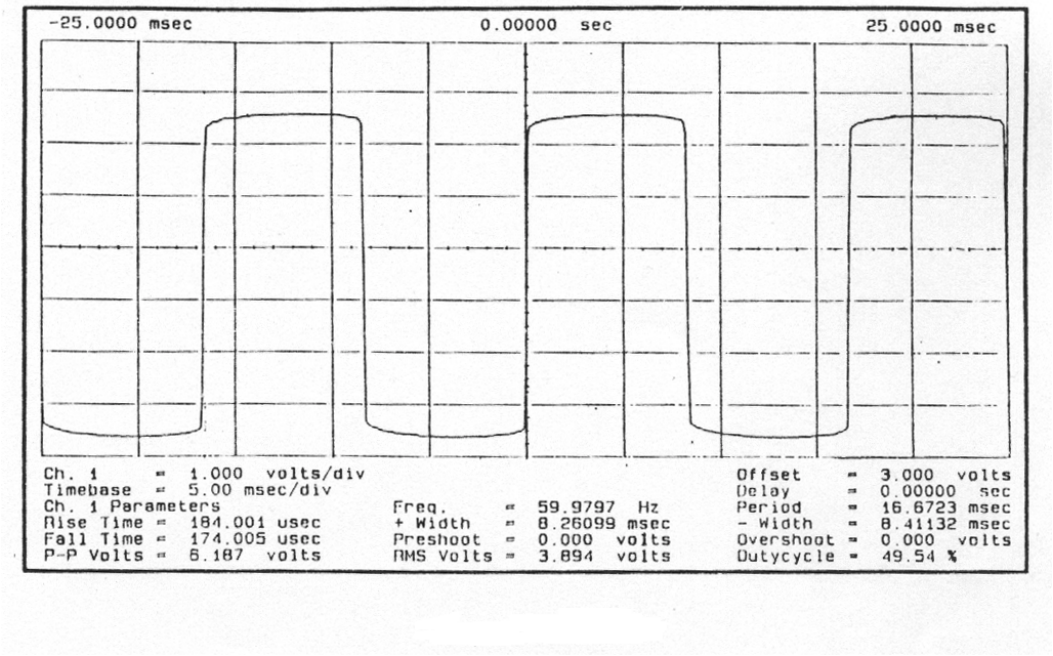


图 1.10　　输入波形

对于 115V AC、60HZ（美国标准）的正弦波电力输入电压值将在 32us 内由 0V 上升到 2V，所以如果要测量“零点”，PIC I/ O 端上 2V 的门槛电压将使“零点”测量有大约 30us 的精度。典型的 PIC I/O

端的电容为 5PF，应取 R=6M（T=RC）或更小一点，有利于测量“零点”的精度。假如取 R=5M，那么 115AC 加上后电流被限于 32uA。这个值对 PIC 是非常安全的。

对于这种电路，有二种情况会损坏 PIC，应加以注意：1、限流电阻 R 短路，但这一般不易发生；2、限流不够造成大电流加于 PIC 端口。PIC 的任一端口只能经受 20mA 的电流。

七、阶梯波产生器（D/A 转换）

D/A 转换是最常使用的一种技术。D/A 转换有多种方法。这里我们要用 R－2R 分压网来产生对应的模拟电压输出。这种分压网原理如下图所示：

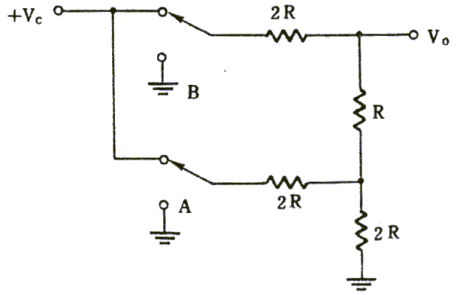


图 1.11　D/A 分压网络

其中 A、B 二个开关可切换在“高”及“低”二个状态，在各种不同的状态下 V0 和 A、B 开关的关系如下：

B	A	V0
低	低	0V
低	高	1/4V
高	低	2/4V
高	高	3/4V

二个开关可组成 4 个台阶波（2=4），8 个开关可组成 256（2）个台阶波。此例即要以 PIC16C54 的 RB7－RB0 作为电子开关产生高/低电平转换，驱动一个基本 R－2R 分压电路，产生相对应的模拟电压量输出。

1、电路设计：

电路如图 1.12 所示：

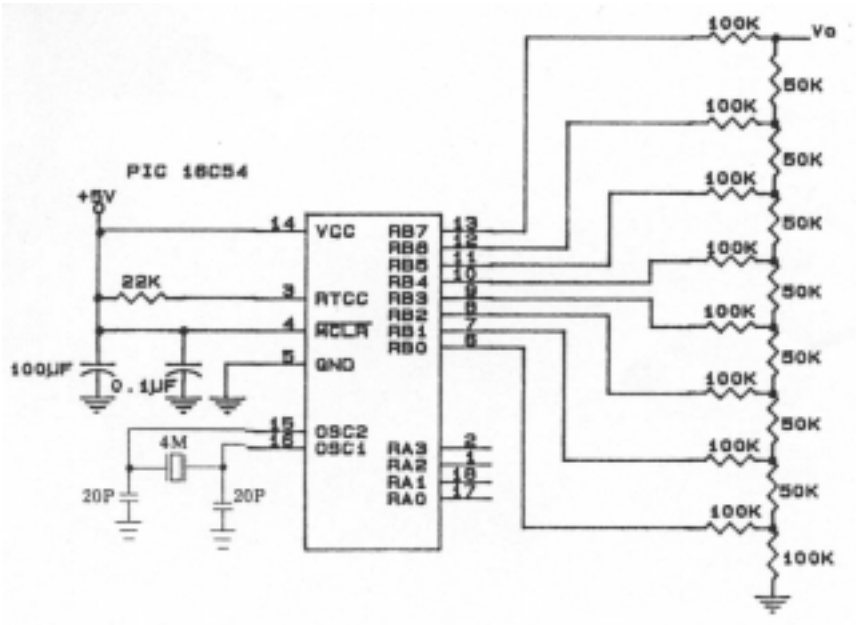


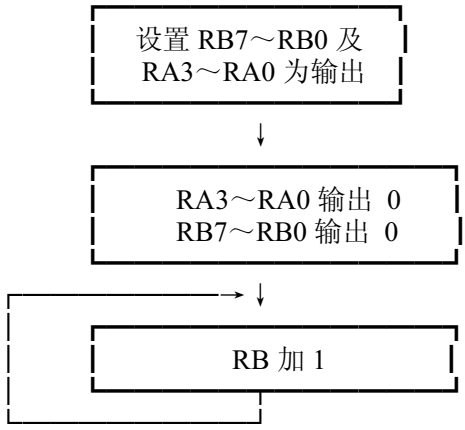
图 1.12　　阶梯波产生器

R－2R 分压网络的电阻值不宜太大，但亦不宜太小以免使输出电流太大而影响输出电压。因为 RB7－RB0 在低电平状态时电压并未达到 OV，为了修正这个误差，可将最后一个接地电阻（100K）接到 RA0，并且使 RA0 保持在低电平输出状态作为模拟的接地点。如果忽略这个误差，也可将 100K 电阻直接接地。

D/A 转换后的输出电压 V0 必须使用高输入阻抗放大器（OPA 等）作缓冲才可输出使用。

使用示波器监测 V0 端可看到连续的锯齿波，如果将其放大可看到锯齿波是由不连续的台阶波组成的。

2、程序流程图：



3、程序清单：

```
RA          EQU         5
RB          EQU         6
PIC54       EQU             1FFH
SUB         EQU         0
;-- ----------------------------------------
                ORG         PIC54
                GOTO        MAIN
;---- ----------------------------------------
                ORG         SUB
;- ----------------------------------------
MAIN        MOVLW       0
                TRIS        RA
                TRIS        RB
                CLRF        RA
                CLRF        RB
LOOP        INCF        RB          ;RB 递增，驱动分压网络输出阶梯波
                GOTO        LOOP
;---- ----------------------------------
                END
;------------------ ----------------------
```

八、A/D 转换

本例要用 PIC16C54 来实现一个 A/D 转换器。其特点是成本很低，仅需 4 个低廉的外围元件。精度可由软硬件调节，分辨率为 6～10 位，转换时间为 250us。利用软件校准还可以补偿时间和温度飘移及消除元件误差。

1、电路设计：

电路如图 1.13 所示：

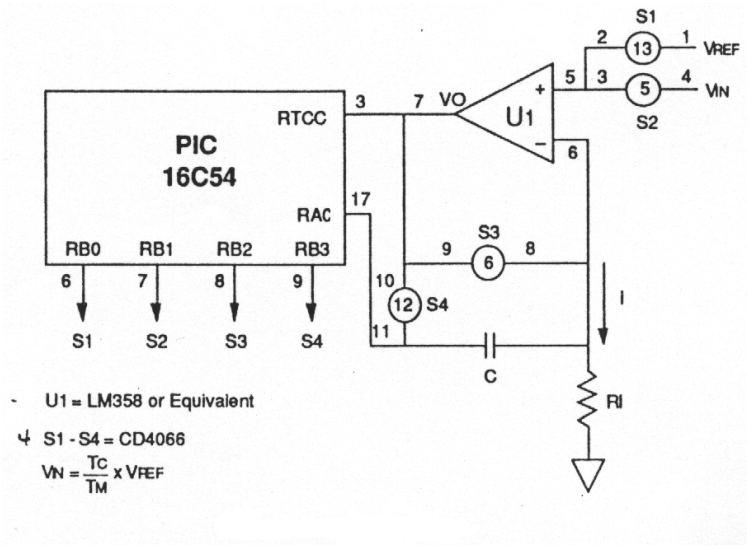


图 1.13　A/D 转换电路

电路中采用一个 RC 充电电路及一个电压/电流转换电路，把输入电压转换成时间量。先算出参考电压 Vref 对应的时间量 Tc，再算出输入电压 Vin 的时间量 Tm，则可由下式算出 Vin：

Vin=Tc/Tm.Vref

具体可按下到步骤求解：

（1）先由 PIC16C54 的 RB 口控制模拟开关 CD4066，把电路接成线放电形式，如图 1.14 所示：

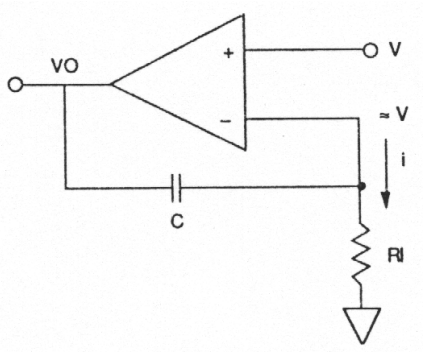


图 1.14　放电电路

（2）再把电路接成图 4.14 所示的充电形式。充电将持续至 V0 上升到 PIC16C54 之 RTCC 的阈门电压值，并引起 RTCC 寄存器值变化（加 1）。程序在此期间所记录的充电时间即为输入电压 V0 对应的时间量 T。

按以上两个步骤，分别对 Vref 和 Vin 求解其对应的时间量，即可按公式算出 Vin，完成 A/D 转换。RC 电路参数可下列公式计算：

RC=（Vi*T）/Vt

式中　Vi——输入转换的最小电压值

T————转换时间

Vt——PIC 输入端口的门槛电压（一般为 3V）

RC 的实际取值应比计算出来的值稍小一点，以免 PIC 在测量过程中计时过头（充电时间）。

注意，由于一般 PIC 的输入门阈电压为 3V，则需用分压电路使 Vin 不超过 3V。实验表明该 A/D 转换器精度达±1%。去掉 u1，则可作为电流型的 A/D 转换器。

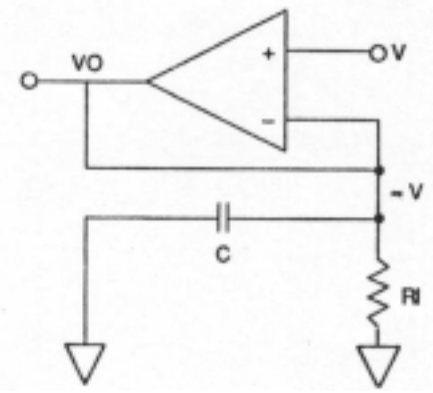
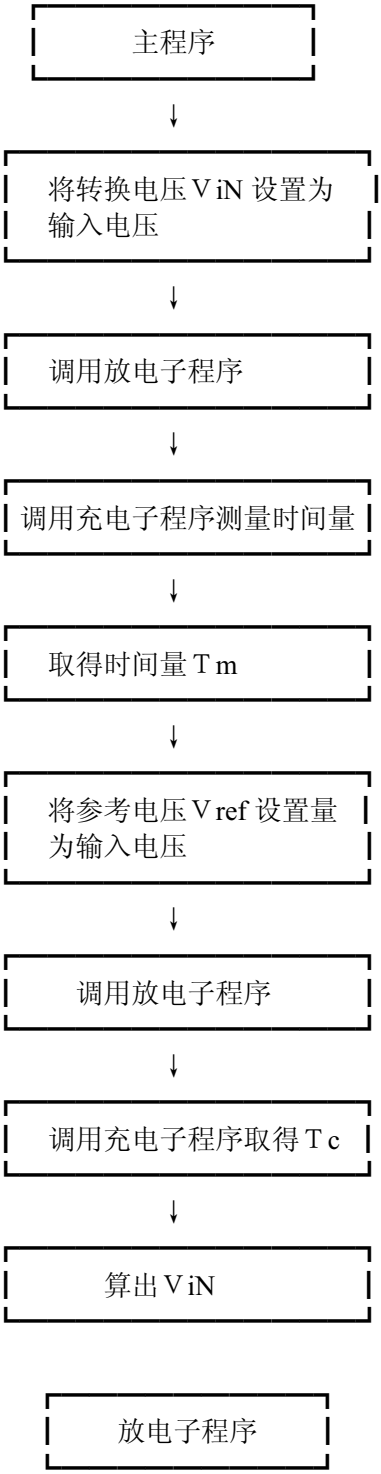
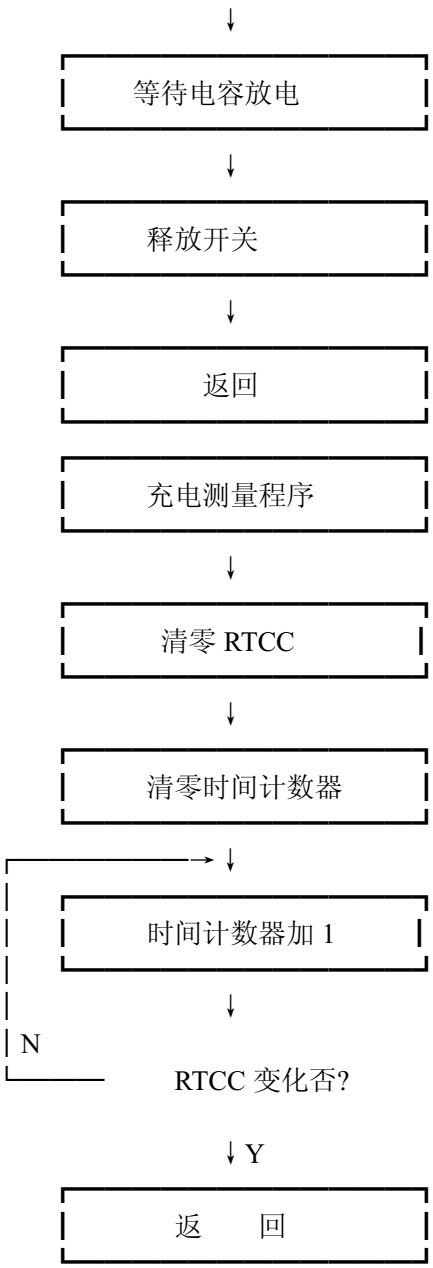


图 1.15　充电电路

2、程序流程图：





3、程序清单：

```
            TITLE     `VOLTMETER/AD CONVERTER
            PROGRAM REV 3-29-90'
            LIST      P=16C54

ACCA        EQU       8
ACCB        EQU       0A
ACCC        EQU       0C
ACCD        EQU       0E
ACCE        EQU       10
TMEAS       EQU       12
TEMP        EQU       14

VCALMS      EQU       60              ; VCAL MSB VALUE IN HEX
VCALLS      EQU       0A4             ; VCAL LSB VALUE IN HEX
            ORG       1FF
                GOTO      VOLTS           ; PROGRAM CODE
```

```
            ORG     0               ; SUBROUTINES

MADD        MOVF    ACCA+1，W
                ADDWF   ACCB+1          ; ADD MSB
            RETLW   0
MPY         CALL    SETUP           ; RESULTS IN B（16 MSB'S）
                                    ; AND C（16 LSB'S）
MLOOP       RRF     ACCD            ; ROTATE D RIGHT
                RRF     ACCD+1
                SKPNC                   ; NEED TO ADD?
            CALL    MADD
            RRF     ACCB
            RRF     ACCB+1
                RRF     ACCC
                RRF     ACCC+1
                DECFSZ  TEMP            ; LOOP UNTIL ALL BITS CHECKED
            GOTO    MLOOP
            RETLW   0

            NOP
SETUP       MOVLW   16
            MOVWF    TEMP
            MOVF    ACCB，W         ; MOVE B TO D
            MOVWF   ACCD
            MOVF    ACCB+1，W
            MOVWF   ACCD+1
            MOVF    ACCC，W
            MOVWF   ACCE
            MOVF    ACCC+1，W
            MOVWF   ACCE+1，W
            CLRF    ACCB
                CLRF    ACCB+1
                RETLW   0
DIV         CALL    SETUP
            MOVLW   32
            MOVWF   TEMP
            CLRF    ACCC
            CLRF    ACCC+1
            DLOOP   CLRC
            RLF     ACCE+1
            RLF     ACCE
            RLF     ACCD+1
            RLF     ACCD
            RLF     ACCC+1
                RLF     ACCC
            MOVF    ACCA，W
                SUBWF   ACCC，W         ; CHECK IF A>C
            SKPZ
```

```
            GOTO        NOCHK
            MOVF        ACCA+1，W
            SUBWF       ACCC+1，W       ; IF MSB EQUAL THEN CHECK LSB
MOCHK       SKPC                        ; CARRY SET IF C>A
            GOTO        NOGO
            MOVF        ACCA+1，W       ; C-A INTO C
            SUBWF       ACCC+1
            BTFSS       3，0
            DECF        ACCC
            MOVF        ACCA，W
            SUBWF       ACCC
                SETC                    ; SHIFT A 1 INTO B（RESULT）
NOGO        RLF         ACCB+1
            RLF         ACCB
            DECFSZ      TEMP            ; LOOP UNTILL ALL BITS CHECKED
            GOTO        DLOOP
                RETLW       0

DSCHRG      MOVLW       B`00001110'     ; DISCHARGE C（RA0 ON）
            TRIS        5
            MOVLW       OFF
            MOVWF       TEMP
LOOP        DECFSZ      TEMP            ; WAIT
            OTO         LOOP
            MOVLW       B`00001111'     ; ALL RA HIGH Z
                TRIS        5
            RETLW       0

M_TIME      CLRF        1               ; CLEAR RTCC REGISTER
            CLRF        ACCA+1          ; CLEAR 16 BIT COUNTER
            CLRF        ACCA
TLOOP       INCFSZ      ACCA+1
            GOTO        ENDCHK
            INCFSZ      ACCA
            GOTO        ENDCHK
            GOTO        END_M
ENDCHK      BTFSS       1，0            ; CHECK FOR RTCC TRIP
            GOTO        TLOOP
END_M       MOVF        1，W
            RETLW       0

VOLTS       MOVLW       B`00000110'     ; SET S2 AND S3 HIGH（ON WHEN
                                        ; ACTIVATED）
            MOVWF           6
            MOVLW       B`11110000'     ; ACTIVATE SWITCHES S1-S4
            TRIS        6
                MOVLW       B`00101000'     ; SELECT POSITIVE EDGE FOR RTCC
```

```
                OPTION
                MOVLW     B`00000000'
                MOVWF     5                 ; SET RA0 LOW（ON WHEN ACTIVATED）
MEAS            CALL      DSCHRG            ; CHARGE CAPACITOR TO VIN
                MOVLW     B`00001010'       ; S2 AND S4 ON
                MOVWF     6
                    CALL      M_TIME            ; MEASURE TIME
                MOVF      ACCA+1，W
                MOVWF     TMEAS+1           ; STORE LSB
                    MOVF      ACCA，W
                    MOVWF     TMEAS             ; STORE MSB

CAL             MOVLW     B`00000101'       ; S1 AND S3 ON
                MOVWF     6
                    CALL      DSCHRG                ; CHARGE CAPACITOR TO VREF
                MOVLW     B`00001001        ; S1 AND S4 ON
                MOVWF     6
                CALL      M_TIME            ; MEASURE TIME

                MOVLW     VCAALLS
                    MOVWF     ACCB+1
                MOVLW     VCALMS
                MOVWF     ACCB

                CALL      MPY               ; MULTIPLY ACCA（TCAL）
                                            ; * ACCB(VREF)
                MOVF      TMEAS+1，W
                MOVWF     ACCA+1
                    MOVF      TMEAS，W
                    MOVWF     ACCA

                CALL      DIV               ; DIVIDE ACCB（TCAL * V）
                                            ; BY ACCA（TMEAS）
                GOTO      VOLTS
                END
```

九、异步串行通讯

在 PIC16C5X 系列的内部没有异步串行通讯口，所以必须用软件来完成这个工作。实际上我们已在很多应用中采用软件实现了 RS－232 标准的异步串行通讯，结果证明工作非常可靠、稳定。用软件来完成串行通讯，降低了芯片的硬件成本。

本例给出 RS－232 的发送和接收电路及其程序，可以作为 PIC16C5X 用户使用串行通讯的参考。

1、电路设计：

如图 1.16 所示：

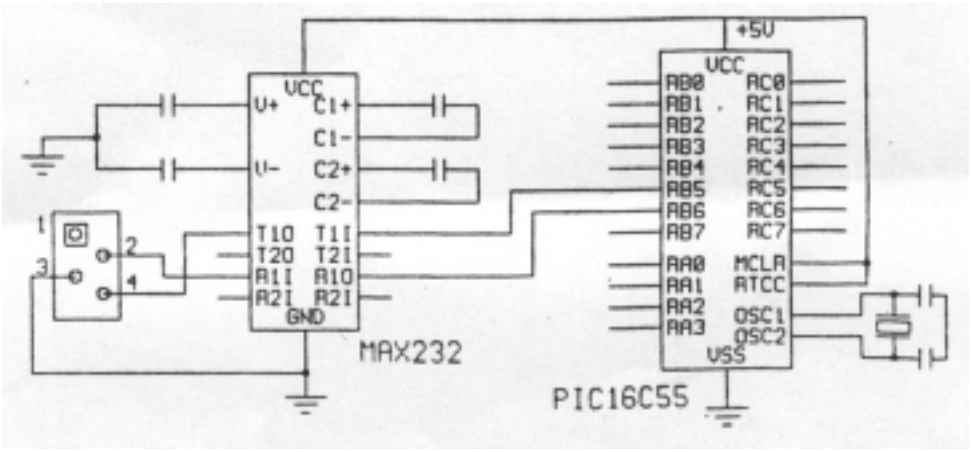
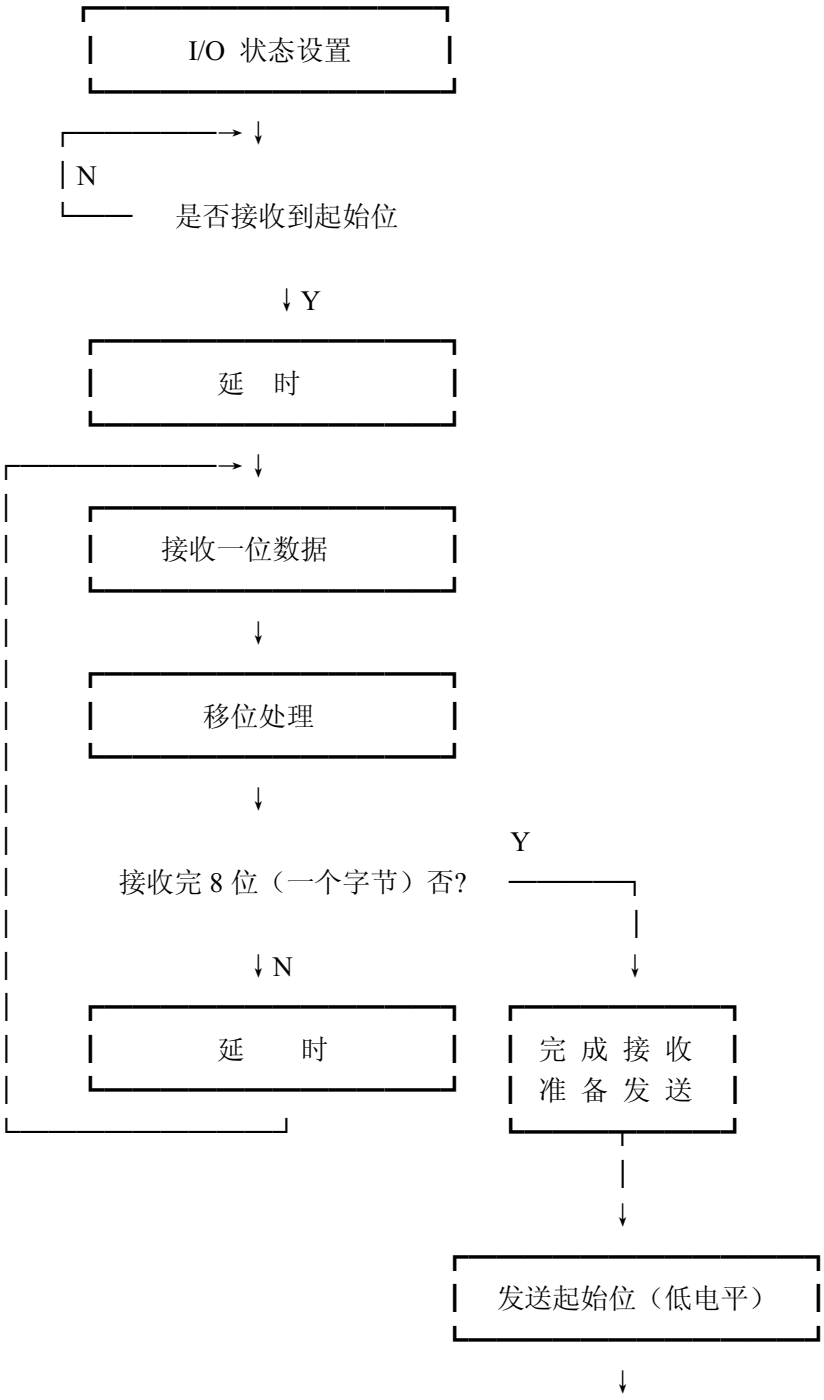
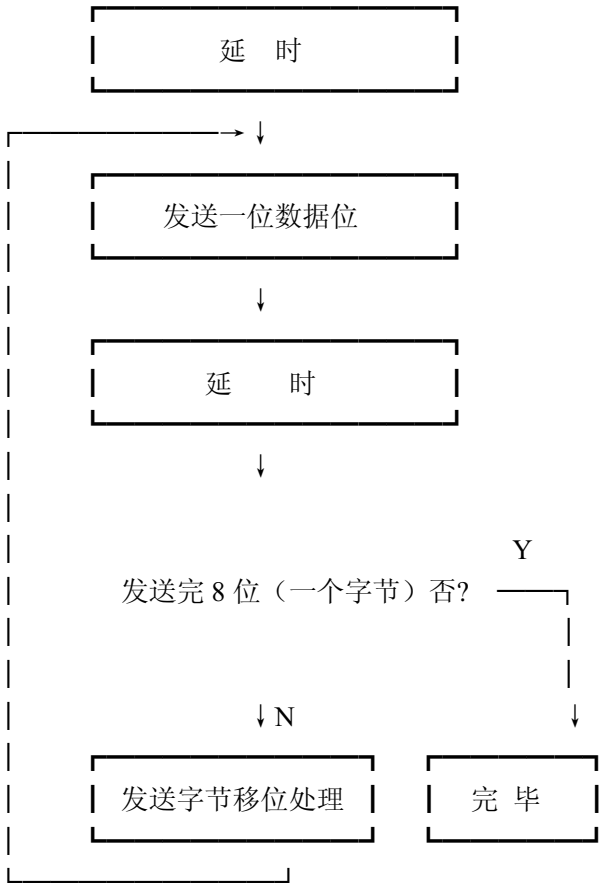


图 1.16 异步串行通讯电路图

2、程序流程图：

1.接收





3、程序清单

```
STATUS      EQU         3
RA          EQU         5
RB          EQU             6
RC          EQU         7
BUFFER      EQU         8
COUNT       EQU         9
BYTE        EQU         10
PIC55       EQU         1FFH
SUB         EQU         0           ;每字节数据 8 位
BITNUM      EQU         8
C           EQU         0
;--- ----------------------------------------
                ORG         PIC55
                GOTO        MAIN
;---- ---------------------------------------
                ORG         SUB
;------------- -------------------------------
DELAY       MOVLW       23          ;接收时每位的延时常数
                MOVWF       BUFFER          ;在 4MHZ 振荡下，9600 波特率比延时常数为 23
LOOP        DECFSZ      BUFFER
                GOTO        LOOP
                RETLW       0
DELAY1      MOVLW       30          ;发送时每位的延时常数（4MHZ，9600 波特率）
                MOVWF       BUFFER
```

```
LOOP        DECFSZ      BUFFER
                GOTO        LOOP
                RETLW       0
;------- -------------------------------------
MAIN        MOVLW       40H
                TRIS        RB              ;置 RB5 为输出口，RB6 为输入口
                CLRF        BYTE
                MOVLW       BITNUM
                MOVWF       COUNT           ;传送 8 位/字节
TEST        BTFSC       RB，6           ;测是否收到起始位（RS232 起始位为一低电平）
                GOTO        TEST            ;未测到
                CALL        DELLAY          ;测到起始位，下面准备接收
                CALL        DELLAY          ;延时
START       BTFSC       RB，6
                BSF         BYTE，7         ;收到高电平位
                BTFSS       RB，6
                BCF         BYTE，7         ;收到低电平位
                DECFSZ      COUNT
                GOTO        L1              ;一个字节未收完，继续收下一位
                CALL        DELAY
                GOTO        TRANSMIT        ;一个字节收完，准备发送
L1          RRF         BYTE            ;移位处理
                CALL        DELAY
                GOTO        START               ;收下一位
TRANSMIT
                MOVLW       BITNUM
                MOVWF       COUNT
BEGIN       BCF         RB，5           ;发送起始位
                CALL        DELAY1          ;延时
L2          RRF         BYTE            ;发送一位数据
                BTFSC       STATUS，C
                BSF         RB，5
                BTFSS       STATUS，C
                BCF         RB，5
                CALL        DELAY           ;延时
                DECFSZ      COUNT           ;一个字节 8 位发送完毕否?
                GOTO        L2              ;否，继续发下一位
                BSF         RB，5           ;是
                CALL        DELAY
                GOTO        TEST            ;循环
;- ----------------------------------------
                END
;--------- -----------------------------------
```

十、I2C 串行总线的控制

I^2C 总线（Intergrated Circuit bus）是 Philips 公司首先推出的一种二线制串行传输总线。它具有占有 I/O 口少、控制方式简单、信号传输速度快、配套功能芯片种类多，因此非常适于单片机系统设计中。I^2C 总线的结构如图 1.11 所示：总线为两根线构成，数据线（SDA）、时钟线（SCL），所有 I^2C 总线接口芯片以线与方式连接在一起。I^2C 总线的数据传输过程为图 1.12 所示，具体过程为：

1、主控者发出开始信号。

2、主控者接着传送出 1byte 的受控者地址信息，其中最低位为读/ 写控制码，“1”为读即主控者将从受控者接收数据；“0”为写即主控者将数据传送给受控者，高 7 位为受控者器件地址码。

3、受控者发出认可信号。

4、发送者开始发送信息每发完 1byte 后接收者发出认可信号给发送者。

5、主控者发出停止信号。

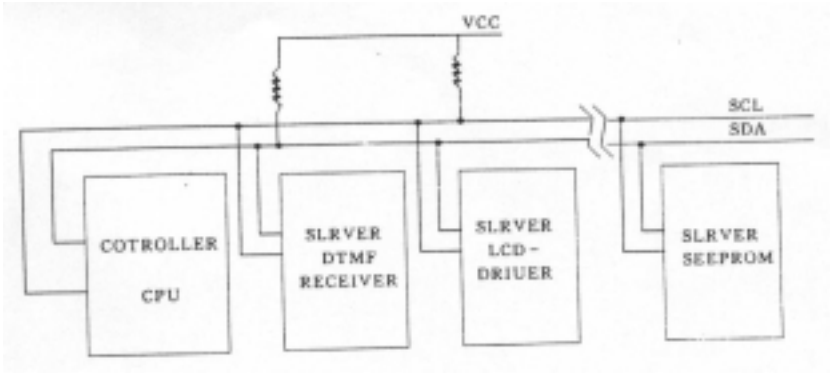


图 1.17 I²C 总线结构

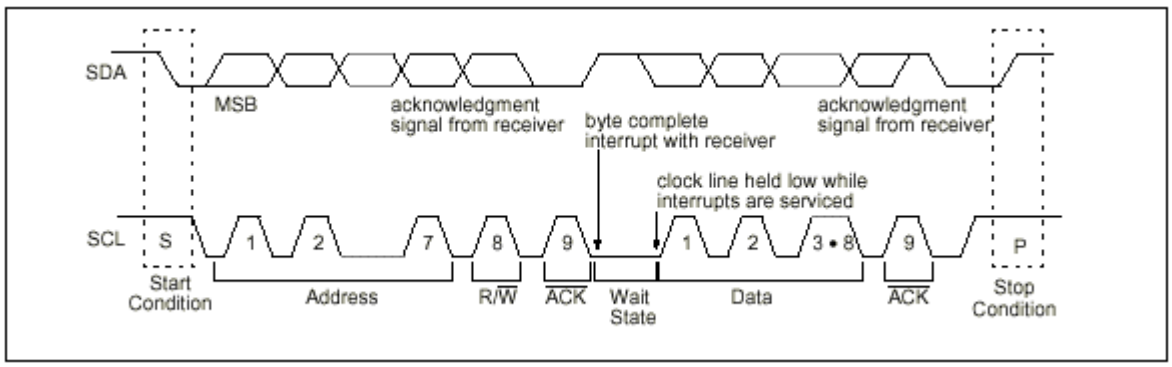


图 1.18 I²C 总线数据传送格式

从图中可看出，数据信号是在时钟信号（SCL）为低电平时发出的。传输开始/停止用时钟信号（SCL）为高电平时，数据信号（SDA）的负跳/正跳来表示。因此在传输数据期间：数据信号（SDA），在时钟信号为高时必须保持稳定。

下面以 24LCXX 系列串行 E²PROM 为例介绍 16C5X 系例单片如何设计 I²C 总线接口。

1、电路设计：

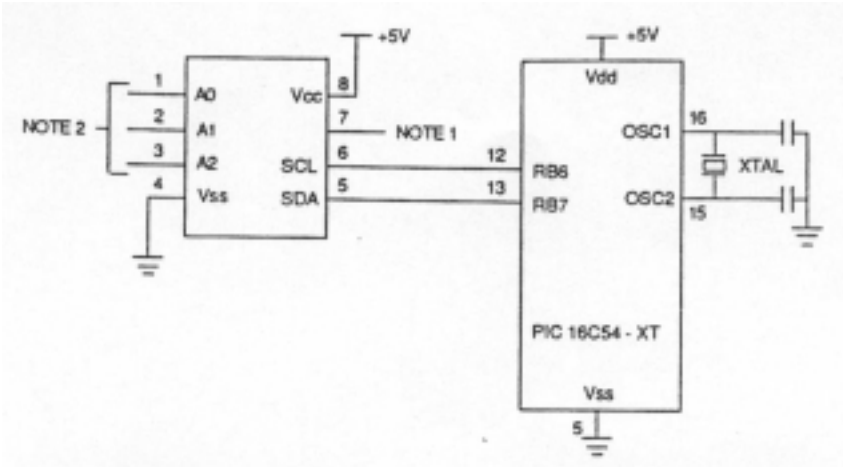


图 1.19　串行 E²PROM 电路图

注 1：对 85CXX 为空脚，对 24LCXX 为写保护端。

注 2：片选控制线、接地或电源，且必须与受控者地址字节中 b3～b1 位逻辑状态相对应。

2、程序流程图：

图 1.20、1.21 分别为串行 E^2PROM 写入，读出的数据传送格式，依据这格式的写、读程序流程图。

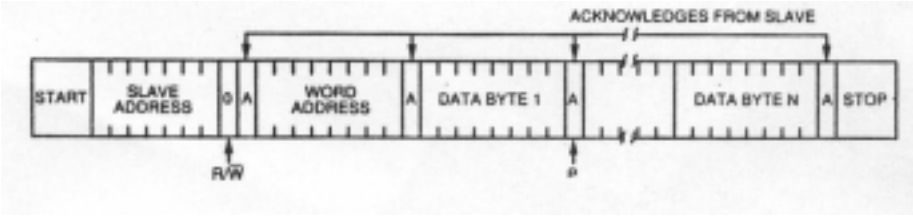


图 1.20 E^2PROM 写入数据传送格式

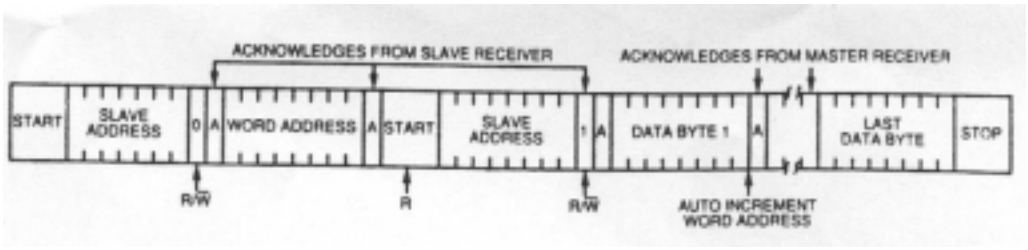
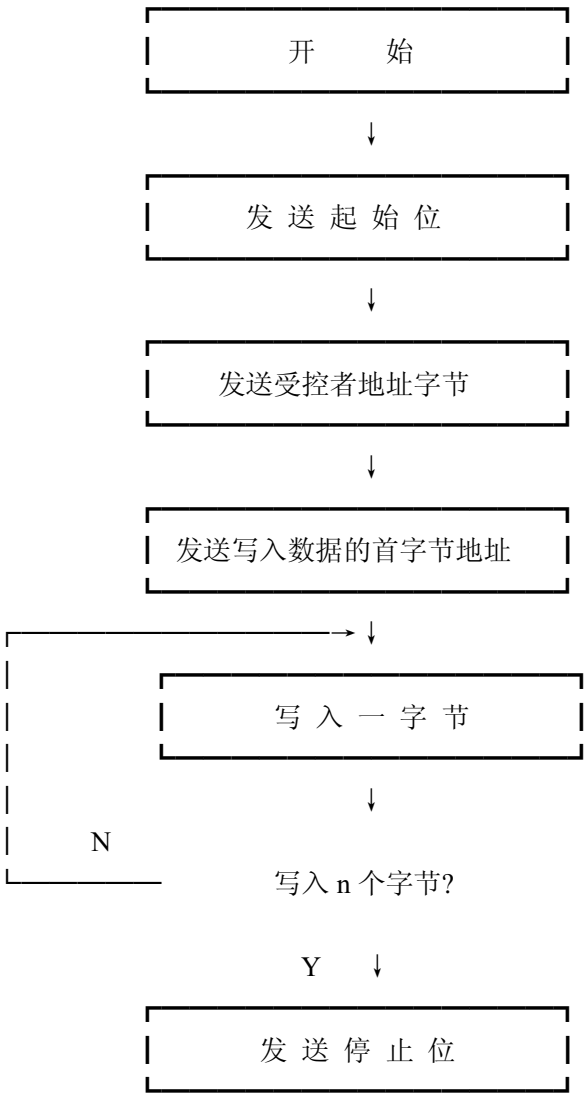
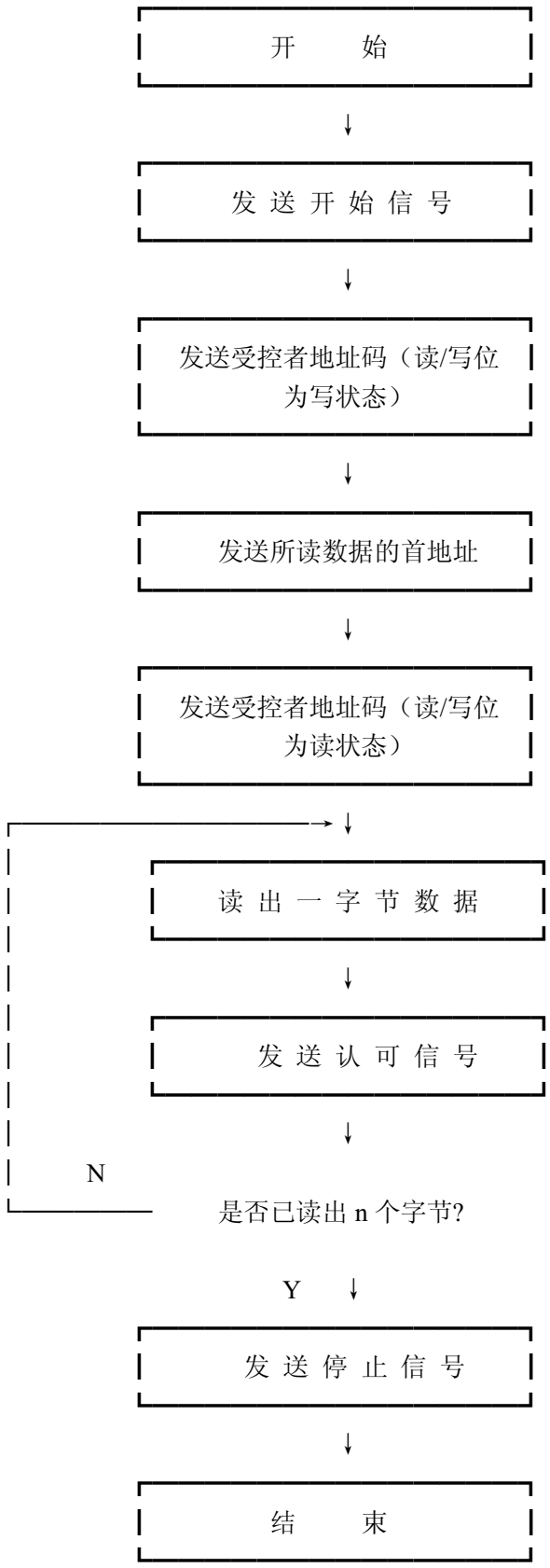


图 1.21 E^2PROM 读出数据传送格式

写入程序：



注：n 不能大于 E^2PROM 每页的字节数。并且 n 个字节必须在同一页内。
读入 n 字节：



3、程序清单：

```
;  ——————————————————————————
;   写入 N 字节数据程序:
```

```
;  把 N 个字节数据写入起始地址为 ADDR 的 E2PROM
;   内。（待写入字节数不得大于 E2PROM 的页容量。）
; ————————————————————————
DIGITS      EQU         0AH             ; 需写入的字节数
ADDR        EQU         0BH             ; 待写入数据的首地址
SLAVE       EQU         0CH             ; E2PROM 的 I2C 总线地址（1010 A2 A1 A0 0）
DATAI0      EQU         10H             ; 写入数据的寄存器
      :
DATAIn      EQU         1XH             ; 写入数据的寄存器

WR
            MOVLW       DIGITS          ; 设置写入字节数
            MOVWF       B_COUNT
            MOVLW       SLAVE           ; 把受控器件的 I2C 总线地
            MOVWF       TXBUF           ; 址存入发送缓冲寄存器
            CALL        BSTART          ; 发送开始命令
            CALL        TX              ; 发送受控器件的 I2C 总线地址
            MOVFW       ADDR
            MOVWF       TXBUF
            CALL        TX              ; 发送写入起始地址
            MOVLW       DATAO0          ; 设置写入数据指针
            MOVWF       FSR

WR0：
            MOVFW       0H              ; 用间址寻址方法取出待写入
            MOVWF       TXBUF           ; 的数据到发送缓冲寄存器
            CALL        TX              ; 写入一字节数据
            INCF        FSR             ; 指针指向下一个数据
            DECFSZ      B_COUNT         ; 是否全部发送完。
            GOTO        WR0
            CALL        BSTOP               ; 发出结束命令
;
            END

; ————————————————————————
; 读出 N 字节数据程序:
; 从 E2PROM 中连续读出 N 字节。
; ————————————————————————
DIGITS      EQU         0AH             ; 需读出的字节数
ADDR        EQU         0BH             ; 待读出数据的首地址
SLAVE       EQU         0CH             ; E2PROM 的 I2C 总线地址（1010 A2 A1 A0 0）
DATAI0      EQU         10H             ; 读出数据的寄存器
;
DATAIn      EQU         1XH             ; 读出数据的寄存器
RD
            MOVLW       DIGITS
            MOVWF       B－COUNT        ; 设置读出字节数
            MOVLW       SLAVE
```

```
                MOVWF     TXBWF          ；放入受控者 I2C 总线地址
                CALL      BSTART         ；发送开始命令
                CALL      TX             ；发送 I2C 总线地址
                MOVFW     ADDR           ；
                CALL      TX             ；
                MOVFW     ADDR           ；
                MOVWF     TXBUF          ；发送待读出数据
                CALL      TX             ；的起始地址
                MOVLW     SLAVE
                MOVWF     TXBUF
                BSF       TXBUF，0       ；设置“读”命令
                CALL      BSTART         ；发出开始命令
                CALL      TX             ；发送“读”命令
                MOVLW     DATAIO         ；设置读出数据
                MOVWF     FSR            ；存储区起始指针
RDO
                CALL      RX             ；读一字节数据
                MOVFW     RXBUF          ；
                MOVWF     0H             ；存入寄存器
                INCF      FSR
                DECFSZ    R－COUNT
                GOTO      RDO
                CALL      BSTOP
                END
；————————————————————
; 发送数据子程序
；————————————————————
TXBUF     EQU       1CH              ；
TX
                MOVLW     8
                MOVWF     COUNT
TXLP
                MOVLW     B`00111111'
                TRIS      RB
                BCF       EEPROM，DO     ；确定发
                BTFSC     TXBUF，7       ；送数据
                BSF       EEPROM，DO     ；
                CALL      BITOUT         ；串行数据输出
                BCF       3，0           ；
                RLF       TXBUF          ；移位，准备发送下一位数据
                DECFSZ    COUNT          ；是否发完一字节
                GOTO      TXLP
                CALL      BITIN          ；接收认可信号。
                RETLW     0
;               END       SUB
；————————————————————
;接收数据子程序
；————————————————————
```

```
RXBUF       EQU         1DH             ；接收数据寄存器
RX：
            MOVLW       8               ；设置字节长度
            MOVWR       COUNT           ；
            CLRF        RXBUF
RXLP
            CALL        BITIN
            BTFSC       EEPROM，DI
            BSF         RXBUF，O
            BCF         3，0
            RLF         RXBUF
            DECFSZ      COUNT
            GOTO        RXLP
            BSF         EEPROM，DO
            CALL        BLTOUT
            RETLW       0
;           END         SUB
; ——————————————————————————
; PIC 发送一位数据给 E2PROM
; ——————————————————————————
BITOUT
            MOVLW       B`00111111'     ；设 RB，SDA
            TRIS        RB              ；RB，SCL 为输
            BTFSS       EEPROM，DO      ；出状态
            GOTO        BITO            ；确定待发送
            BSF         RB，SDA         ；位状态
            NOP
            GOTO        CLK1
BITO
            BCF         RB，SDA
            NOP
            NOP
CLK1
            BSF         RB，SCL
            NOP
            NOP
            NOP
            NOP
            BCF         RB，SCL
            RETLW       0
;           END         SUB
; ——————————————————————————
; 从 E2PROM 中读出 1 位数据
; ——————————————————————————
BIGIN
            MOVLW       B`10111111'     ；RB， SDA 为输入端
            TRIS        RB
            BCF         EEPROM，DI
```

```
                BSF         RB，SCL         ；时钟置“1”
                NOP
                BTFSC       RB，SDA         ；读 SDA
                BSF         EEPROM，DI
                NOP
                BCF         RB，SCL         ；时钟置“0”
                RETLW       OH
;               END         SUB
; ——————————————————————————————
;“开始命令”程序
; ——————————————————————————————
BSTART
                MOVLW       B`00111111'     ；置 SCL，
                TRIS        RB              ；SDA 为输出状态
                BSF         RB，SCL
                NOP
                NOP
                NOP
                BCF         RB，SDA          ；发出“开始”命令
                NOP
                NOP
                NOP
                BCF         RB，SCL
                RETLW       0
;
; ——————————————————————————————
;“停止”命令发送子程序
; ——————————————————————————————
BSTOP
                BCF         RB，SDA
                NOP
                NOP
                NOP
                BSF         RB，SCL
                NOP
                NOP
                NOP
                BSF         RB，SDA         ；发出“停止”命令
                RETLW       0
```

十一、液晶 LCD 显示驱动

迄今为止，LCD 显示器是功耗最低的显示器件。在要求低功耗的单片机系统中，普遍均采用 LCD 显示器。

由于 LCD 显示器要求采用交流驱动方式，因此单片机一般通过 LCD 驱动器来驱动 LCD 显示器。下面介绍一种采用 AY0438 驱动 4 位 LCD 显示电路及控制软件。

AY0438 是美国 Microchip 公司生产的一种 LCD 驱动器。它采用静态方式，可驱动 32 段。显示数据

采用串行方式输入。

其电路结构和引脚排列如图 1.22、图 1.23。

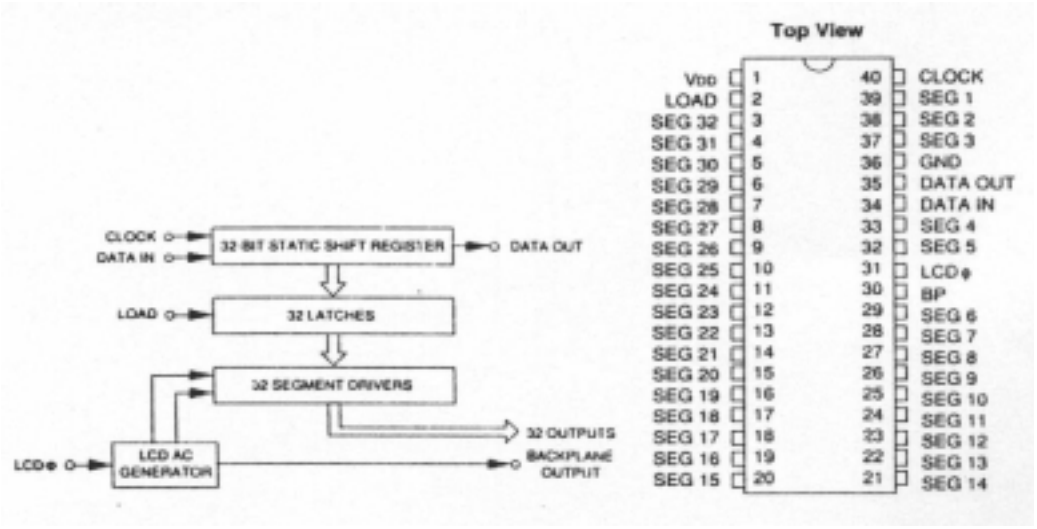


图 1.22　AY0438 电路结构图　　图 1.23　AY0438 引脚图

其中：SEG1～SEG32 为段驱动信号引脚。

BP 为背极信号驱动脚。

LCD0 为背极信号输入脚/振荡器外接电容引脚，做为背极信号输入脚时，输入信号即做为背极驱动信号；做为振荡器外接电容引脚时（电容另一端接地），振荡器信号经 2 分频后，做为背极信号。外接电容容量与背极驱动信号频率的关系如图 1.24。

Data in 为串行数据输入端 。

Data out 为串行数据输出端。Clock 为串行数据时钟输入端。Load 锁存信号输入端。

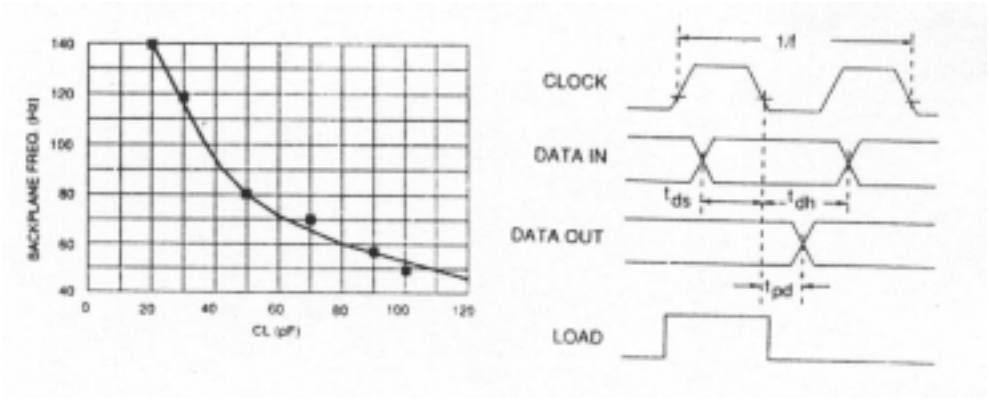
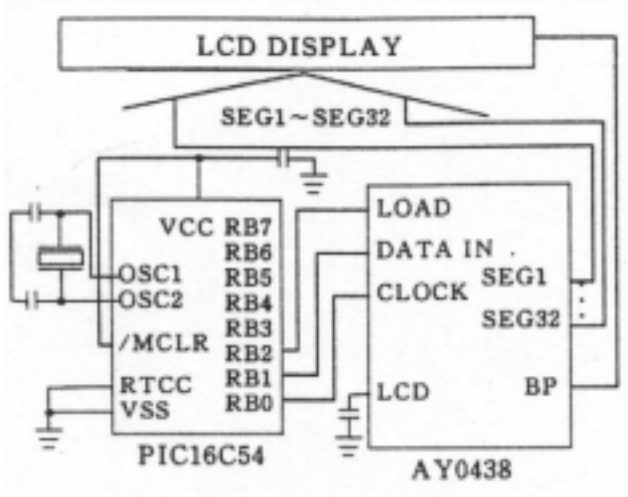


图 1.24　外接电容容量与背极驱动信号频率关系图　　图 1.25　数据传送时序图

1、电路设计：



1.26 显示控制电路图

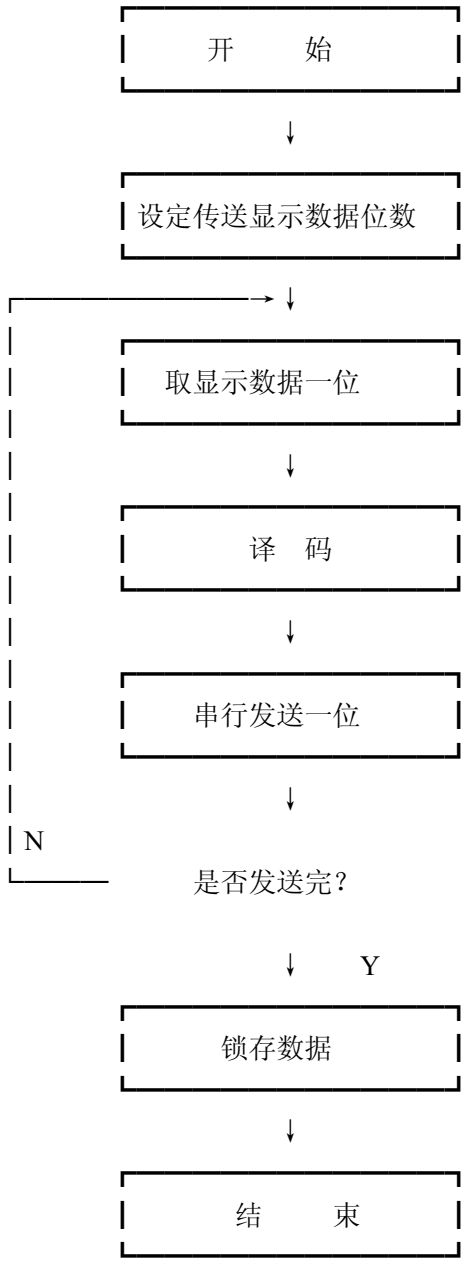
串行信号时序关系为图 1.25 所示，输入数据的锁存、移位、输出均由时钟信号的下降沿控制。

显示控制电路为图 1.26 所示，LCD 显示器为 4 位 8 段数码（含小数点）。LCD 显示器引脚与显示段的关系如表 1.1。

	a	b	c	d	e	f	g	p
第一位	Seg32	Seg31	Seg30	Seg29	Seg28	Seg27	Seg26	Seg25
第二位	Seg24	Seg23	Seg22	Seg21	Seg20	Seg19	Seg18	Seg17
第三位	Seg16	Seg15	Seg14	Seg13	Seg12	Seg11	Seg10	Seg9
第四位	Seg8	Seg7	Seg6	Seg5	Seg4	Seg3	Seg2	Seg1

表 1.1

2、程序流程图：



3、程序清单：

```
INDEX    EQU    0
STATUS   EQU    3
FSR      EQU    4
```

```
RA          EQU         5
RB          EQU         6
BUFFER      EQU         8
COUNT_D     EQU         9
COUNT_B     EQU         10
TEMP        EQU         11
DP0         EQU         12                  ;显示数据
DP1         EQU         13
DP2         EQU         14
DP3         EQU         15
PIC56       EQU         3FFH
SUB         EQU         0
CLK         EQU         0
DATA        EQU         1
LOAD        EQU         2
C           EQU         0
DIGITNUM    EQU         4
;---- ------------------------------------------
                ORG         PIC56
                GOTO        MAIN
;- ------------------------------------------
                ORG         SUB
;-- ------------------------------------------
CONVERT     ADDWF       PC                  ;显示数码数据表
                RETLW       03FH            ;0
                RETLW       006H            ;1
                RETLW       05BH            ;2
                RETLW       04FH            ;3
                RETLW       055H            ;4
                RETLW       05DH            ;5
                RETLW       07DH            ;6
                RETLW       007H            ;7
                RETLW       07FH            ;8
                RETLW       06FH            ;9
;--- ------------------------------------------
MAIN        MOVLW       B'11111000'
                TRIS        RB
                ANDWF       RB              ;置 RB0～RB2 为输出口，并置初值 0
LOOP
                MOVLW       DP0
                MOVWF       FSR             ;设置欲显示数据的指针
                MOVLW       DIGITNUM        ;4 位显示
                MOVWF       COUNT_D
TR
                MOVFW       INDEX           ;取显示数据（DP0～DP3）
                CALL        CONVERT         ;调用段码数据转换程序
                MOVWF       TEMP
                MOVLW       8
                MOVWF       COUNT_B         ;8 段显示
TR0         BSF         RB，CLK
```

```
                RRF         TEMP
                BTFSS       STATUS，C       ;显示段码右移一位
                GOTO        TR1
                BSF         RB，DATA
                GOTO        TR2
TR1         BCF         RB，DATA
TR2         NOP
                NOP
                BCF         RB，CLK
                DECFSZ      COUNT_B         ;8 段移位完成否
                GOTO        TR0
                INCF        FSR             ;一位数据已显示完毕，准备取下一位
                DECFSZ      COUNT_D         ;4 位数显示完否?
                GOTO        TR              ;未显示完，继续下一位显示操作
                BSF         RB，LOAD        ;锁存
                NOP
                NOP
                BCF         RB，LOAD        ;循环
                GOTO        LOOP
;----- ----------------------------------------
                END
;-- ----------------------------------------
```

十二、PIC16C5X 模拟 EPLD、PLD 电路

EPLD、PLD 较通用逻辑器件具有逻辑设计灵活、可减小芯片使用数量、可加密等优点。由于 PIC16C5X 系列单片机运行速度很高，因此在很多情况下可模拟 EPLD、PLD 器件的功能，其保密性能更好，使用更灵活。

下面以-8 输入端、8 输出端 24 个积项的与—或可编程阵列为例介绍模拟方法。

可编程阵列电路结构如图 1.27 所示。PIC16C5X 模拟电路如图 1.28。其中 RB 口设置为输入端 RC 口为输出端。可编程阵列要实现的电路功能的真值表如表 1.2。

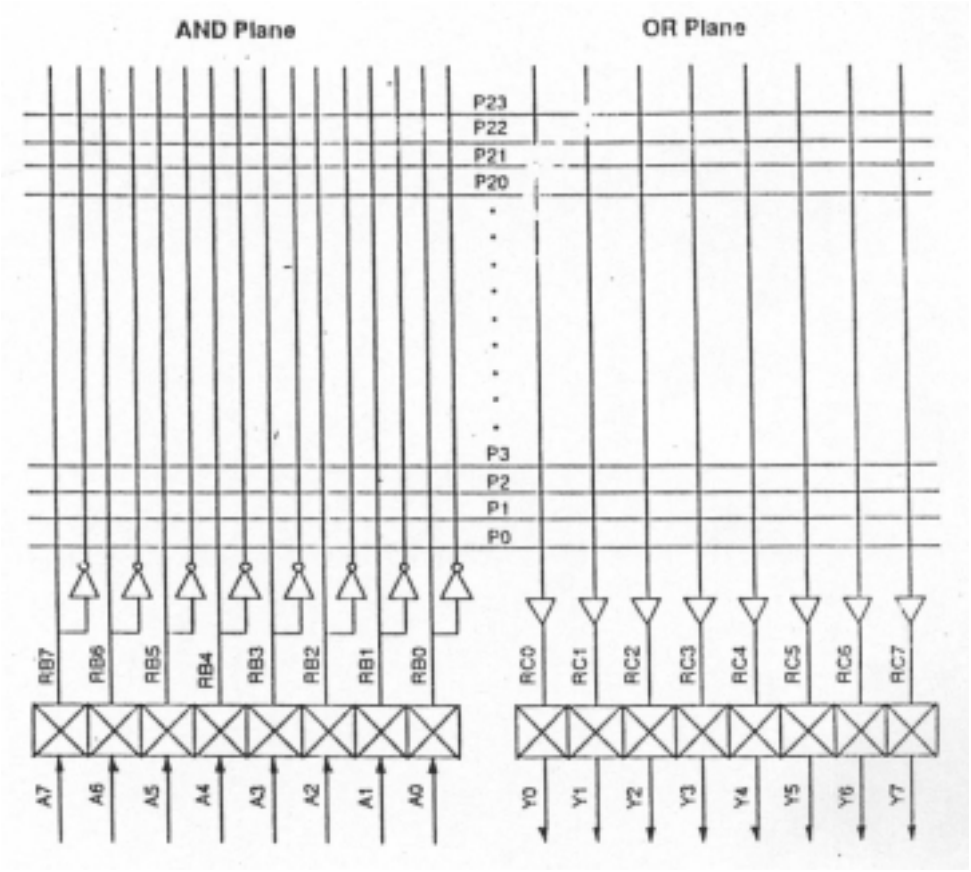


图 1.27　与/或可编程阵列电路图

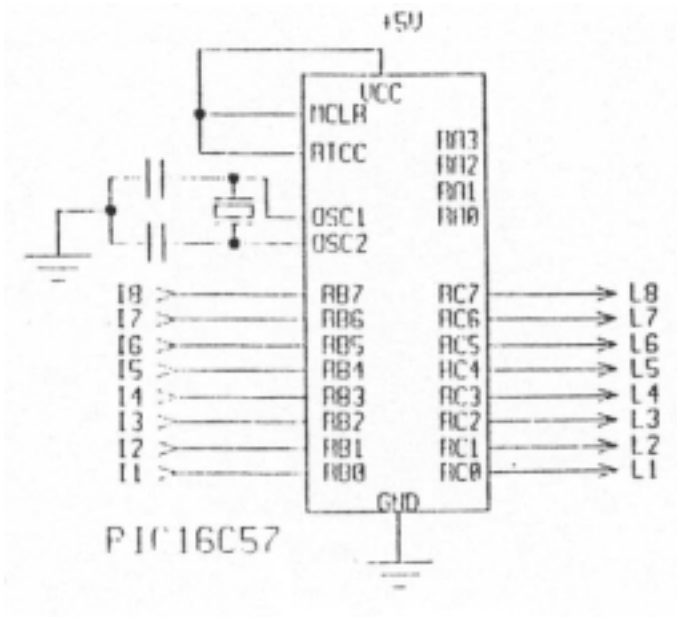
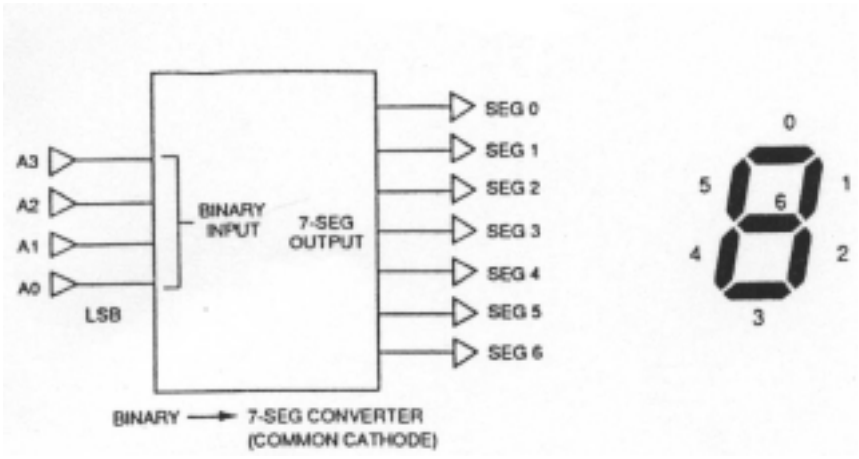


图 1.28 16C5X 模拟电路图

1、电路设计：

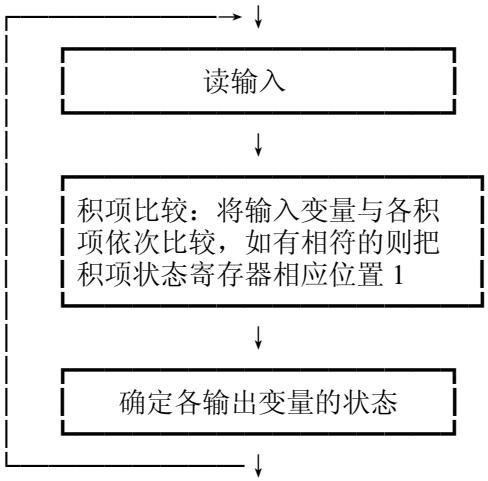


Truth Table:　　Product Terms

HEX	A3	A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	
0	0	0	0	0	1	1	1	1	1	1	0	P1 = $\bar{A3}$, $\bar{A2}$, $\bar{A1}$, $\bar{A0}$
1	0	0	0	1	0	1	1	0	0	0	0	P1
2	0	0	1	0	1	1	0	1	1	0	1	P2
3	0	1	1	1	1	1	1	1	0	0	1	P3
4	0	1	0	0	0	1	1	0	0	1	1	P4
5	0	1	0	1	1	0	1	1	0	1	1	P5
6	0	1	1	0	1	0	1	1	1	1	1	P6
7	0	1	1	1	1	1	1	0	0	0	0	P7
8	1	0	0	0	1	1	1	1	1	1	1	P8
9	1	0	0	1	1	1	1	0	0	1	1	P9
A	1	0	1	0	1	1	1	0	1	1	1	P10
B	1	0	1	1	0	0	0	1	1	1	1	P11
C	1	1	0	0	1	0	0	1	1	1	0	P12
D	1	1	0	1	0	1	1	1	1	0	1	P13
E	1	1	1	0	1	0	0	1	1	1	1	P14
F	1	1	1	1	1	0	0	0	1	1	1	P15 = A3, A2, A1, A0

表 1.2　真值表

2、流程图：



上算法中，积项比较以下逻辑表达式进行

<RB(0:7).XOR.Pn-X>.AND.Mn-a

其中：RB（0:7）为从输入端读入的输入值；Pn-x 为需比较第 n 个积项，即真值表中第 n 项，或输出逻辑函数与-或表达式中的第 n 个积项，如表 1.2 中 P1- x= ××××A3 A2 A1 A0=××××0 0 0 0 如果不相等则跳，相等则置 1，Mn-a 项为第 n 个积项屏蔽参数，用于屏蔽多余的输入端的值，如此例中仅用 RB 口低四位输入端，因此 Mn-a 为 b`00001111'。

如上表达式结果为“0”则把积项状态寄存器中对应位置“1”，反之置“0”。

输出状态由下逻辑表达式确定：

(Preg-a.AND.OR-a).OR.(Preg-b.AND.OR-bn).OR.(Preg-c.AND.OR.cn)

其中 Preg-a.Preg-b.Preg-c 为积项状态寄存器，积项比较的结果。OR-an. OR-bn.OR-cn 为第 n 个输出端输出值的比较参数，根据真值表或“与——或”表达式确定。

如表 1.2 中

Y0=P0+P1+P2+P3+P4+P5+P6+P7+P8+P9+P10+P11+P12+P13+P14

则 OR-A0=B`11101101'

OR-B0=B`011101111'

OR-C0=B`00000000'

当该逻辑表达式不为“0”时，输出为“1”，反之为“0”。

3、程序清单：

```
Y-reg       EQU     10H             ; 输出结果寄存器
Input       EQU     11H             ; 输入值寄存器
Preg-a      EQU     12H             ; 积项状态寄存器。
Preg-b      EQU     13H             ; 积项状态寄存器。
Preg-c.     EQU     14H             ; 积项状态寄存器。
;
;
P0_X        EQU     B"00000000"     ;
P0_A        EQU     B"00001111"     ;
P1_X        EQU     B"00000001"     ;
P1_A        EQU     B"00001111"     ;
P2_X        EQU     B"00000011"     ;
P2_A        EQU     B"00001111"     ;
P3_X        EQU     B"00000100"     ;
P3_A        EQU     B"00001111"     ;
P4_X        EQU     B"00000100"     ;
P4_A        EQU     B"00001111"     ;
```

```
P5_X        EQU         B"00000101"         ;
P5_A        EQU         B"00001111"         ;
P6_X        EQU         B"00000110"         ;
P6_A        EQU         B"00001111"         ;
P7_X        EQU         B"00000111"         ;
P7_A        EQU         B"00001111"         ;
P8_X        EQU         B"00001000"         ;
P8_A        EQU         B"00001111"         ;
P9_X        EQU         B"00001001"         ;
P9_A        EQU         B"00001111"         ;
P10_X       EQU         B"00001010"         ;
P10_A       EQU         B"00001111"         ;
P11_X       EQU         B"00001011"         ;
P11_A       EQU         B"00001111"         ;
P12_X       EQU         B"00001100"         ;
P12_A       EQU         B"00001111"         ;
P13_X       EQU         B"00001101"         ;
P13_A       EQU         B"00001111"         ;
P14_X       EQU         B"00001110"         ;
P14_A       EQU         B"00001111"         ;
P15_X       EQU         B"00001111"         ;
P15_A       EQU         B"00001111"         ;
P16_X       EQU         B"00000000"         ;
P16_A       EQU         B"00000000"         ;
P17_X       EQU         B"00000000"         ;
P17_A       EQU         B"00000000"         ;
P18_X       EQU         B"00000000"         ;
P18_A       EQU         B"00000000"         ;
P19_X       EQU         B"00000000"         ;
P19_A       EQU         B"00000000"         ;
P20_X       EQU         B"00000000"         ;
P20_A       EQU         B"00000000"         ;
P21_X       EQU         B"00000000"         ;
P21_A       EQU         B"00000000"         ;
P22_X       EQU         B"00000000"         ;
P22_A       EQU         B"00000000"         ;
P23_X       EQU         B"00000000"         ;
P23_A       EQU         B"00000000"         ;
;
;
;
OR_A0       EQU         B"11101101"         ; FOR OUPUT Y0
OR_B0       EQU         B"11010111"         ;
OR_C0       EQU         B"00000000"         ;
OR_A1       EQU         B"10011111"         ; FOR OUPUT Y1
OR_B1       EQU         B"00100111"         ;
OR_C1       EQU         B"00000000"         ;
OR_A2       EQU         B"11111101"         ; FOR OUPUT Y2
OR_B2       EQU         B"01111011"         ;
OR_C2       EQU         B"00000000"         ;
OR_A3       EQU         B"01101101"         ; FOR OUPUT Y3
OR_B3       EQU         B"01111001"         ;
OR_C3       EQU         B"00000000"         ;
```

```
OR_A4       EQU         B"01000101"          ; FOR OUPUT Y4
OR_B4       EQU         B"11111101"          ;
OR_C4       EQU         B"00000000"          ;
OR_A5       EQU         B"01110001"          ; FOR OUPUT Y5
OR_B5       EQU         B"11011111"          ;
OR_C5       EQU         B"00000000"          ;
OR_A6       EQU         B"01111100"          ; FOR OUPUT Y6
OR_B6       EQU         B"11101111"          ;
OR_C6       EQU         B"00000000"          ;
OR_A7       EQU         B"00000000"          ; FOR OUPUT Y7
OR_B7       EQU         B"00000000"          ;
OR_C7       EQU         B"00000000"          ;

            ORG         01FFH                ;
BEGIN       GOTO        MAIN                 ;

            ORG         000H                 ;

MAIN        CALL        PLA88                ;
            GOTO        MAIN                 ;
; 宏定义: 输入值与积项比较
EVAL-P      MACRO       PREG-X，BIT-N，  PN-X，PN-D
            MOVF        INPUT，W             ;
                XORLW       PN-X                 ;
            ANDLW       PN-A
            BTFSC       STATUS，BIT2         ;
                BSF         PREG-X，BIT-N        ;PRODUCT TERM=1
            ENDM

;
; 宏定义: 确定输出值
EVAL-Y      MACRO       OR-AN，OR-BN，O  R-CN，BIT-N
            LOCAL       SETBIT               ; 伪指令

            MOVF        PREG-A，W                ;
                ANDLW       OR-AN                ;
            BTFSS       STATUS，BIT2         ;
            GOTO        SETBIT               ;

            MOVF        PREG-B，W            ;
                ANDLW       OR-BN                ;
                BTFSS       STATUS，BIT2         ;
            GOTO        SETBIT               ;

                MOVF        PREG-C，W            ;
                ANDLW       OR-CN                ;
            BTFSS       STATUS，BIT2         ;
SETBIT      BSF         Y-REG，BIT-N         ;
            ENDM

; 可编程阵列 PLA 模拟程序
;
PLA88       MOVLW       0FFH                 ;
```

```
            TRIS    6                   ; PORT-B=INPUT
            MOVF    PORT-B，W           ; READ INPUT
            MOVWF   INPUT               ; STORE INPUT IN A REGISTER
            CLRF    PREG-A              ; CLEAR PRODUCT REGISTER A
            CLRF    PREG-B              ; CLEAR PRODUCT REGISTER B
            CLRF    PREG-C              ; CLEAR PRODUCT REGISTER C
            CLRF    Y-REG               ; CLEAR OUTPUT REGISTER

AND-PL      EVAL-P  PREG-A，BIT0，P0-X，P0-A      ；依次比较 24 个积项
            EVAL-P  PREG-A，BIT1，P1-X，P1-A
            EVAL-P  PREG-A，BIT2，P2-X，P2-A
            EVAL-P  PREG-A，BIT3，P3-X，P3-A
            EVAL-P  PREG-A，BIT4，P4-X，P4-A
            EVAL-P  PREG-A，BIT5，P5-X，P5-A
            EVAL-P  PREG-A，BIT6，P6-X，P6-A
            EVAL-P  PREG-A，BIT7，P7-X，P7-A

            EVAL-P  PREG-A，BIT0，P8-X，P8-A
            EVAL-P  PREG-A，BIT1，P9-X，P9-A
                EVAL-P  PREG-A，BIT2，P10-X，P10-A
            EVAL-P  PREG-A，BIT3，P11-X，P11-A
            EVAL-P  PREG-A，BIT4，P12-X，P12-A
            EVAL-P  PREG-A，BIT5，P13-X，P13-A
            EVAL-P  PREG-A，BIT6，P14-X，P14-A
            EVAL-P  PREG-A，BIT7，P15-X，P15-A

            EVAL-P  PREG-A，BIT0，P16-X，P16-A
            EVAL-P  PREG-A，BIT1，P17-X，P17-A
            EVAL-P  PREG-A，BIT2，P18-X，P18-A
            EVAL-P  PREG-A，BIT3，P19-X，P19-A
            EVAL-P  PREG-A，BIT4，P20-X，P20-A
            EVAL-P  PREG-A，BIT5，P21-X，P21-A
                EVAL-P  PREG-A，BIT6，P22-X，P22-A
            EVAL-P  PREG-A，BIT7，P23-X，P23-A

OR-PL       EVAL-Y  OR-A0，OR-B0，OR-C0，BIT0      ；确定 8 个输出变量的输出值
            EVAL-Y  OR-A1，OR-B1，OR-C1，BIT1
            EVAL-Y  OR-A2，OR-B2，OR-C2，BIT2
                EVAL-Y  OR-A3，OR-B3，OR-C3，BIT3
            EVAL-Y  OR-A4，OR-B4，OR-C4，BIT4
            EVAL-Y  OR-A5，OR-B5，OR-C5，BIT5
            EVAL-Y  OR-A6，OR-B6，OR-C6，BIT6
            EVAL-Y  OR-A7，OR-B7，OR-C7，BIT7

WR-OUT      CLRW                        ;
            TRIS    7                       ；置 RC 口为输出
                MOVF    Y-REG，W            ;
            MOVWF   PORT-C              ; RC 口
            RETLW   0                       ;
```

§4.3　算术例程

一、　无符号的 BCD 加法

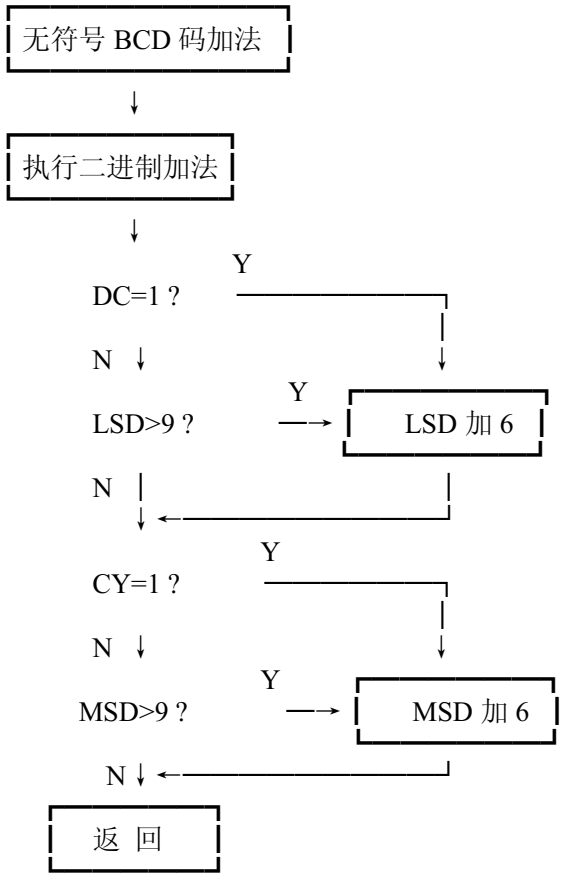
使用二进制数做 BCD 的加法时，相加结果必须调整，以转换成 BCD 的位数。下列程序使用二个步骤来完成这个调整。

1、如果和的最低有效 1 位是一个>9 的数，或 DC 位=1 时将和加 6（产生 DC 标志）。

2、完成步骤 1 后，如果最高有效 1 位表示>9 的数，或者相加后 C 标志为 1，将和加 60H（即加 6 到 MSD）。

注：扩展程序到 2 位数以上时，所有的加法都必须连 C 标志（或 DC）一起相加，而且上面的原则都用到每个位数，从原数相加，步骤 1、步骤 2 的加法所产生的进位都必须进位到下一位数。

1、流程图：



2、程序清单：

```
TITLE`无符号 BCD 加法'
;----------------------------------------
;F12  放被加数，F11 放加数
;运行后和放在 F12，进位放在 F11
```

```
                ;----------------------------------------
UBCDAD      MOVF        11，W                   ;
            ADDWF       12                      ; 作二进制加法
            CLRF        11                      ;
            RLF         11                      ; 保存进位
            SKPNDC                              ; DC=1?
            GOTO        ADJST                   ; 是，调整 LSD
            MOVLW       6                       ; 测 LSD>9 否（通过 LSD 加 6
            ADDWF       12                      ; 并判断 DC=1?）
            SKPNDC                              ;
            GOTO        OVR1                    ;
            SUBWF       12                      ; LSD<9
            CLRC                                ; LSD 不作调整
            GOTO        OVR1                    ;
ADJST       MOVLW       6                       ; LSD 加 6 调整
            ADDWF       12                      ;
OVR1        RLF         11                      ; 保存进位
            MOVLW       $60                     ; MSD 加 6 调整
            ADDWF       12                      ;
            SKPNC                               ; MSD>9 否
            GOTO        OVR2                    ; YES
            BTFSC       11，0                   ; 测试保存的进位位
            GOTO        OVR3                    ;
            SUBWF       12                      ;
            GOTO        OVR                     ;
OVR3        MOVLW       1                       ; 保存和进位
            MOVWF       11                      ;
            GOTO        OVR                     ;
OVR2        CLRF        11                      ; 保存和进位
            RLF         11                      ;
OVR         RETLW       0                       ;
```

二、 无符号的 BCD 减法

使用二进制数做 2 位数 BCD 减法（2 补码的加法），必须调整其相减结果成为正确的 BCD 数。调整步骤如下：

1、如果差的低 4 位（LSD）>9，则从 LSD 减去 6（产生的 DC 被加到下一位数）。

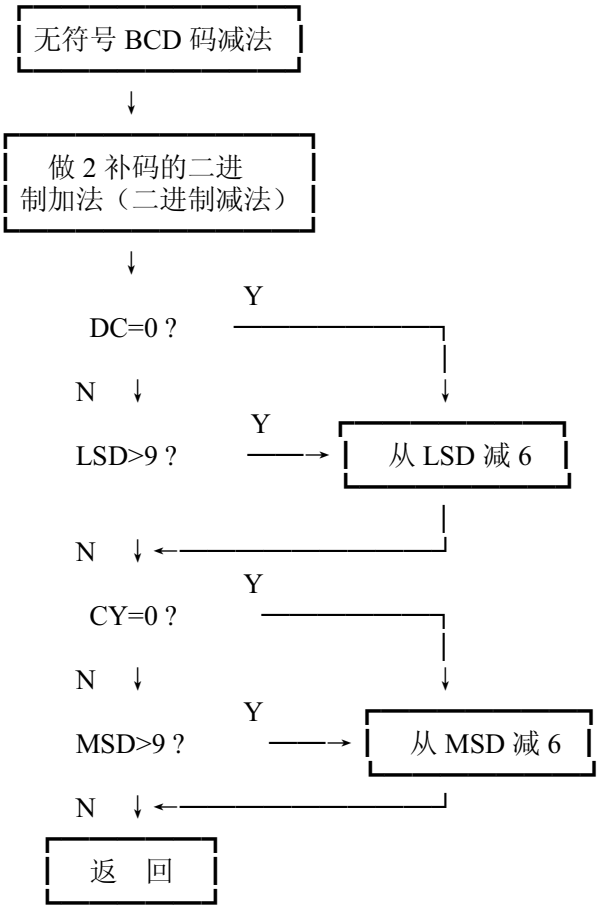
2、步骤 1 完成后，如果差的高 4 位（MSD）>9，则从 MSD 减去 6。

注：（1）程序扩展到二位数以上时每个 BCD 位数都使用以上的原则。

（2）C 标志的测试（步骤 2）是在二补码的加法完成后进行。

（3）当 F11=9，结果是 VE，取十补码以取得它的值。

1、流程图：



2、程序清单：

```
            TITLE`无符号 BCD 码减法 '
                ;- ----------------------------------------------------
            ;F11  放被减数
            ;F12  放减数
            ;运算结果差放在 F12，借位放在 F11
            ;------------------ ---------------------------------
UBCDAD      MOVF        11，W                   ; 做二进制减法
            SUBWF       12                      ;
            CLRF        11                      ;
            RLF         15                      ; 保存进位
            SKPDC                               ; DC=0?
            GOTO        ADJST1                  ; 是，LSD 调整
            MOVLW       6                       ; 测 LSD>9 否
            ADDWF       12，W                   ;
            SKPDC                               ;
            GOTO        OVR1                    ;
ADJST1      MOVLW       6                       ; LSD 调整（减 6）
            SUBWF       12                      ;
OVR1        CLRF        11
            RLF         11                      ; 保存进位
            BTFSS       15，0
            GOTO        ADJST2
```

```
                BTFSS       11，0
                GOTO        ADJST2
                MOVLW       $60                 ; MSD>9?
                ADDWF       12，W
                SKPC
                GOTO        OVER
ADJST2          MOVLW       $60                 ; MSD 调整
                SUBWF       12
                CLRF        11
                GOTO        OVR
                MOVLW       1
                MOVWF       11
OVR             RETLW       0                   ;
```

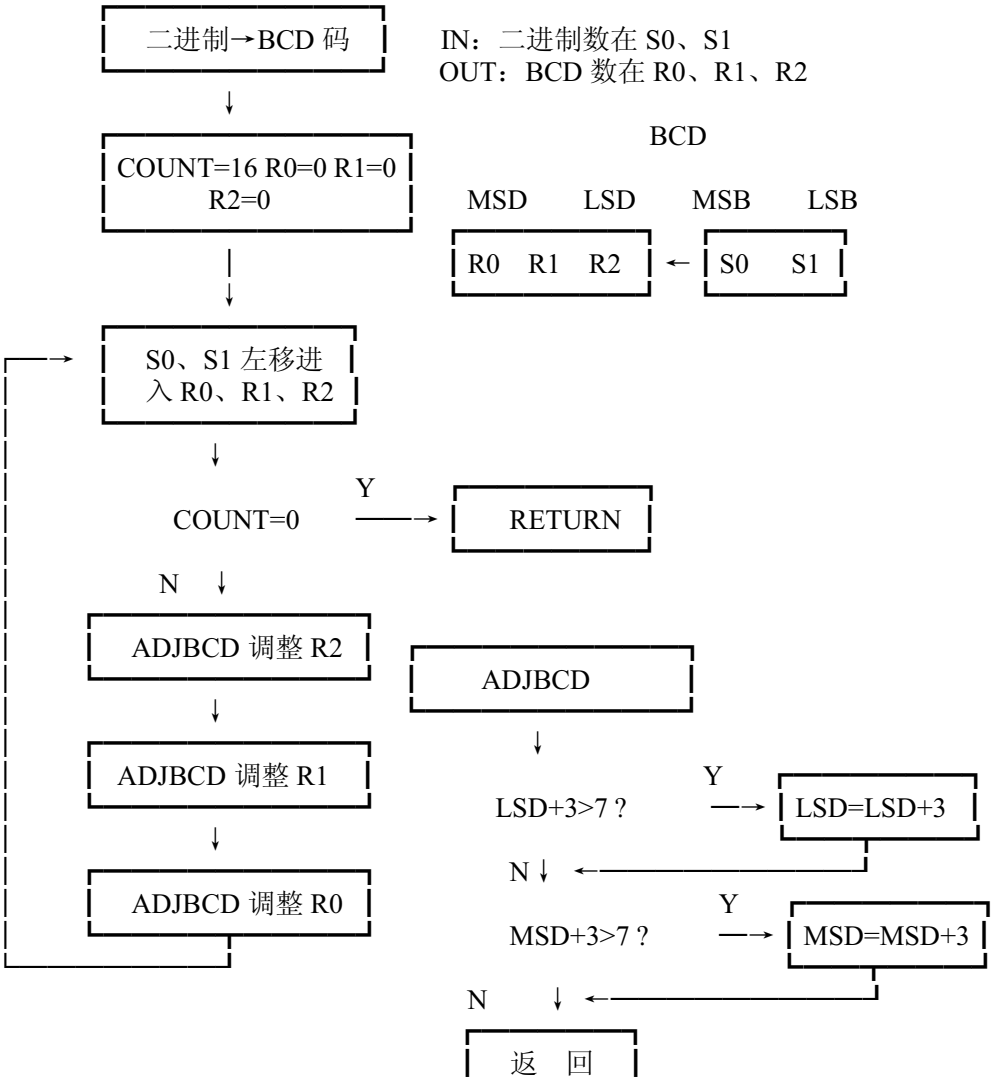
三、 二进制转换成 BCD 数

本程序将一个 16 位的二进数转换成一个 5 位数的 BCD 数。16 位的二进制数放在寄存器 S0、S1。S0 放高位。5 位数的 BCD 数输出到寄存器 R0、R1、R2。MSD 放在 R0，LSD 放在 R2。

一个很简单的演算用来完成转换，二进制数向右移一位，一直到 16 位全部移完即完成转换。

否则每个 BCD 数被检查是否大于 4，如果是则加 3 到该位数。上述的过程一直重复执行。

1、流程图：



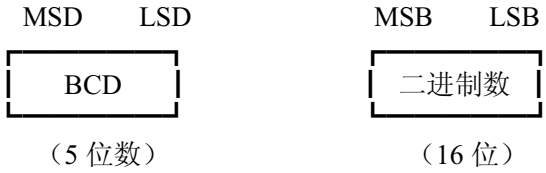
2、程序清单：

```
                Title'二进制数转换成 BCD 数'
                ;-----------------------------------
                ;二进制数放在 S0、S1
                ;转换结果 BCD 数放在 R0、R1、R2
                ;-----------------------------------
BINTOB      MOVLW     16
            MOVWF     COUNT
            CLRF      R0
            CLRF      R1
            CLRF      R2
LOOPC       RLF       S1
            RLF       S0
            RLF       R2
            RLF       R1
            RLF       R0
            DECFSZ    COUNT
            GOTO      ADJDEC
            RETLW     0                ; 完成转换
ADJDEC      MOVLW     R2
            MOVWF     FSR
            CALL      ADJBCD           ; 调整 R2
            MOVLW     R1
            MOVWF     FSR
            CALL      ADJBCD           ; 调整 R1
            MOVLW     R0
            MOVWF     FSR
            CALL      ADJBCD           ; 调整 R0
            GOTO      LOOPC
                                       ;
ADJBCD      MOVLW     X`03'
            ADDWF     0，W             ; LSD 加 3
            MOVWF     TEMP
            BTFSC     TEMP，3          ; 结果>7
            MOVWF     0                ; 存到 LSD
            MOVLW     X`30'
            ADDWF     0，W             ; LSD 加 3
            MOVWF     TEMP
            BTFSC     TEMP，7          ; 结果>7
            MOVWF     0                ; 存到 MSD
            RETLW     0
```

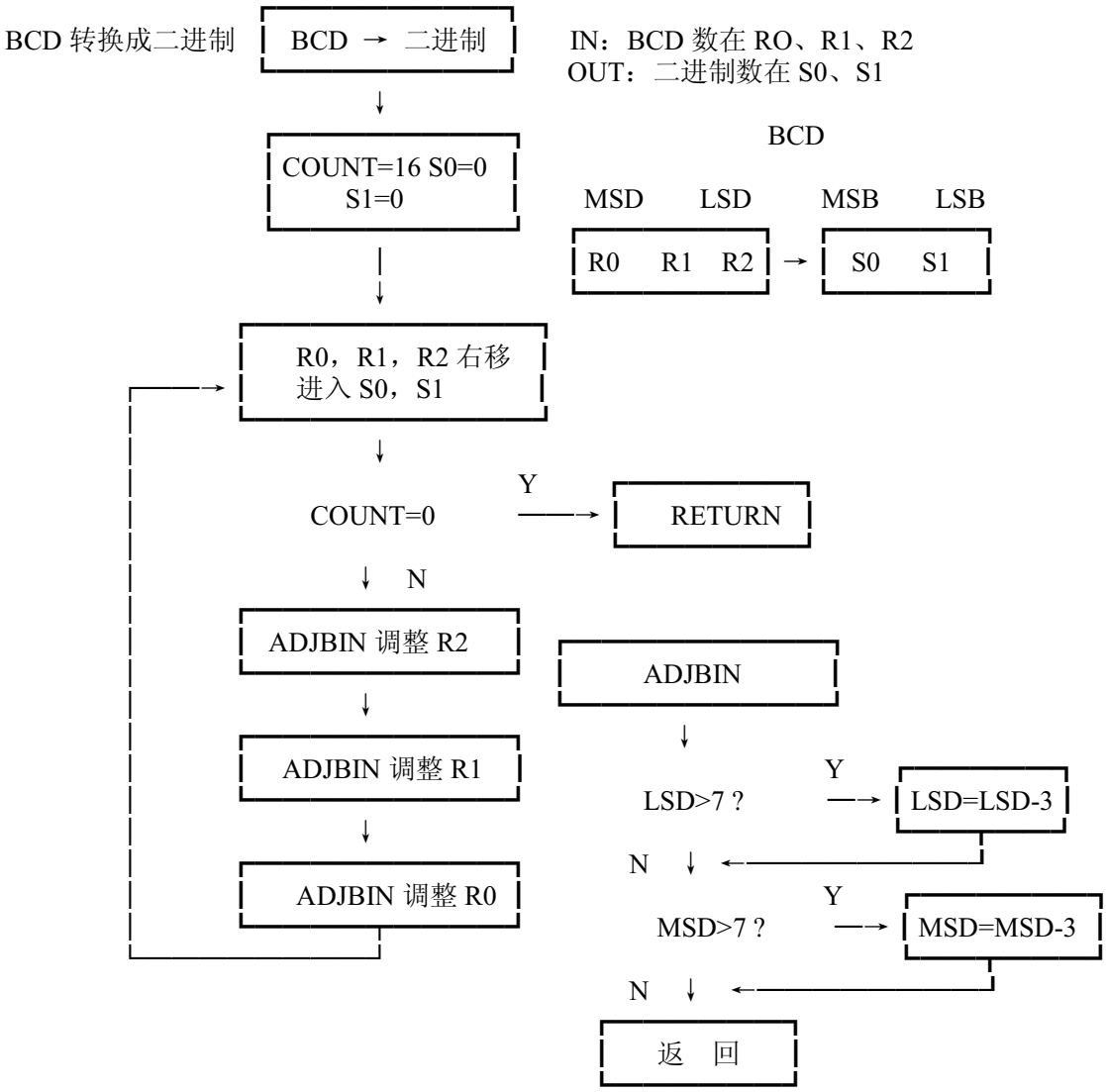
四、 BCD 转换成二进制数

本程序将 5 位数的 BCD 转换成 16 位的二进制数。5 位的 BCD 数 R0、R1、R2。MSD 放在 R0 的最右位。16 位的二进制数输出到寄存器 S0、S1。S0 存放高位。

程序使用简单的演算完成转换。BCD 数向右移一位到二进制数内，直到 16 位全部移位完才跳出程序。否则每一个 BCD 数被检查是否大于 7，如果是则将该位数减 3，重复上述过程到结束。



1、流程图：



2、程序清单：

```
                Title 'BCD 数转换成二进制数 '
                ;--------------------------------
                ;BCD 数放在 R0、R1、R2
                ;转换结果二进制数放在 S0、S1
                ;--------------------------------
FSR             EQU         4
                ;
BINTOB          MOVLW       16
                MOVWF       COUNT
                CLRF        S0                  ;
                CLRF        S1
LOOPC           CLRC                            ;
```

```
            RRF     R0          ; BCD 码移位进二进制单元
            RRF     R1
            RRF     R2
            RRF     S0
            RRF     S1
            DECFSZ  COUNT
            GOTO    ADJDEC
            RETLW   0           ; 完成转换
                                ;
ADJDCT      MOVLW   R2
            MOVWF   FSR
              CALL      ADJBIN        ; 调整 R2
            MOVLW   R1
            MOVWF   FSR
            CALL    ADJBIN      ; 调整 R1
            MOVLW   R0
            MOVWF   FSR
            CALL    ADJBIN      ; 调整 R0
            GOTO    LOOPD
ADJBIN      MOVLW   X`03'
            BTFSC   0，3        ; IF >7
            SUBWF   0           ; LSD 减 3
            MOVLW   X`30'
            BTFSC   0，7        ; IF >7
            SUBWF   0           ; MSD 减 3
            RETLW   0
```