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。

PIC12C5XX 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.cn）。

第一章　PIC12C5XX 功能原理

PIC12C5XX 是美国 Microchip 公司推出的 8 位单片机，也是世界上第一个 8 脚封装的 8 位单片机系列。

§1.1　功能特点

一、高性能 RISC 结构 CPU

- 精简指令集，仅 33 条单字节指令，易学易用
- 除地址分支跳转指令为双周期指令外，其余所有指令皆为单周期指令
- 执行速度： DC～1μs
- 二级硬件堆栈
- 直接、间接、相对三种寻址方式

二、功能部件特性

- 8 位定时器/计数器 TIMER0，带 8 位预分频器
- 大驱动电流，I/O 脚可直接驱动数码管（LED）显示
  - 每个 I/O 引脚最大控电流 25mA
  - 每个 I/O 引脚最大灌电流 20mA
- 内置上电复位电路（POR）
- 复位定时器，保障复位正常
- 内部 MCLR 复位端加上拉电路，无需外接上拉
- 内置自振式看门狗，防程序死锁
- 程序保密位，可防止程序代码的非法拷贝
- 低功耗睡眠功能
- I/O 引脚可唤醒睡眠
- 内置 4MHz RC 型振荡源，可省外接振荡
- 可选外接振荡
  - RC： 低成本阻容振荡
  - XT： 标准晶体/陶瓷振荡
  - LP： 低速晶体，低功耗振荡

三、CMOS 工艺特性

- 低功耗

  ＜2mA　@5V，4MHz

  -15μA　@3V，32KHz

  -＜1μA　低功耗睡眠（Sleep）模式下
- 全静态设计
- 宽工作电压范围：2.5V～5.5V
- 宽工作温度范围：

  -商用级： 0℃～＋70℃

  -工业级：－40℃～＋85℃

  -汽车级：－40℃～＋125℃

§1.2 型号及引脚介绍

PIC12C5XX 目前有二种型号，见下表：

型 号	振 荡	EPROM	RAM	定时器	输入线	I/O 线	电压范围	封装（DIP/SOIC）
12C508	DC~4Mhz	512×12	25×8	1	1	5	2.5V-5.5V	8
12C509	DC~4Mhz	1024×12	41× 8	1	1	5	2.5V-5.5V	8

表 1.1 PIC12C5XX 型号功能表

各型号管脚图如下：

PDIP，SOIC，Windowed CERDIP

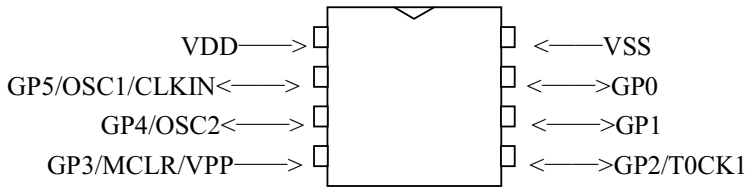


图 1.1 12C508/509 引脚

下表描述了各引脚的功能。

引脚名	引脚序号	属性	缓冲类型	功 能
GP0	7	I/O	TTL/ST	双向 I/O 口线，带可编程弱上拉，并具电平变化唤醒睡眠功能
GP1	6	I/O	TTL/ST	双向 I/O 口线，带可编程弱上拉，并具电平变化唤醒睡眠功能
GP2/T0CK1	5	I/O	ST	双向 I/O 口线，并可设置为计数器 TIMER0 的外部信号输入端
GP3/MCLR	4	I	TTL	单向输入口线，也可设置为芯片复位端。当设为复位端 MCLR 时，低电平有效。当作为输入口线时，带可编程弱上拉及电平变化唤醒睡眠功能
GP4/OSC2	3	I/O	TTL	双向 I/O 口线，（使用片内 RC 振荡源时，也可作为晶振输出端）
GP5/OSC1/CLKIN	2	I/O	TTL/ST	双向 I/O 口线，（使用片内 RC 振荡源时，也可作为晶振输入端或外部振荡输入端）
V_{DD}	1	电源	—	正电源
V_{SS}	8	电源	—	地

注：ST — 斯密特触发器

表 1.2　PIC12C5XX 引脚功能

从上表可看出，PIC12C5XX 最多可以有 5 根 I/O 口线和 1 根输入口线（GP3）。

§1.3　PIC12C5XX 内部结构

PIC12C5XX 的总线结构采用的是数据总线（8 位）和指令总线（12 位）独立分开的”哈佛结构”，所以它具有精简指令集（RISC）的特点，速度快，效率高，并且功耗很低。

PIC12C5XX 在一个芯片上集成了 8 位的算术逻辑运算单元（ALU），0.5K～1K 的 12 位程序存储器，25～41 个 8 位数据寄存器以及 8 位的计数器，上电复位电路，复位定时器，看门狗等等。

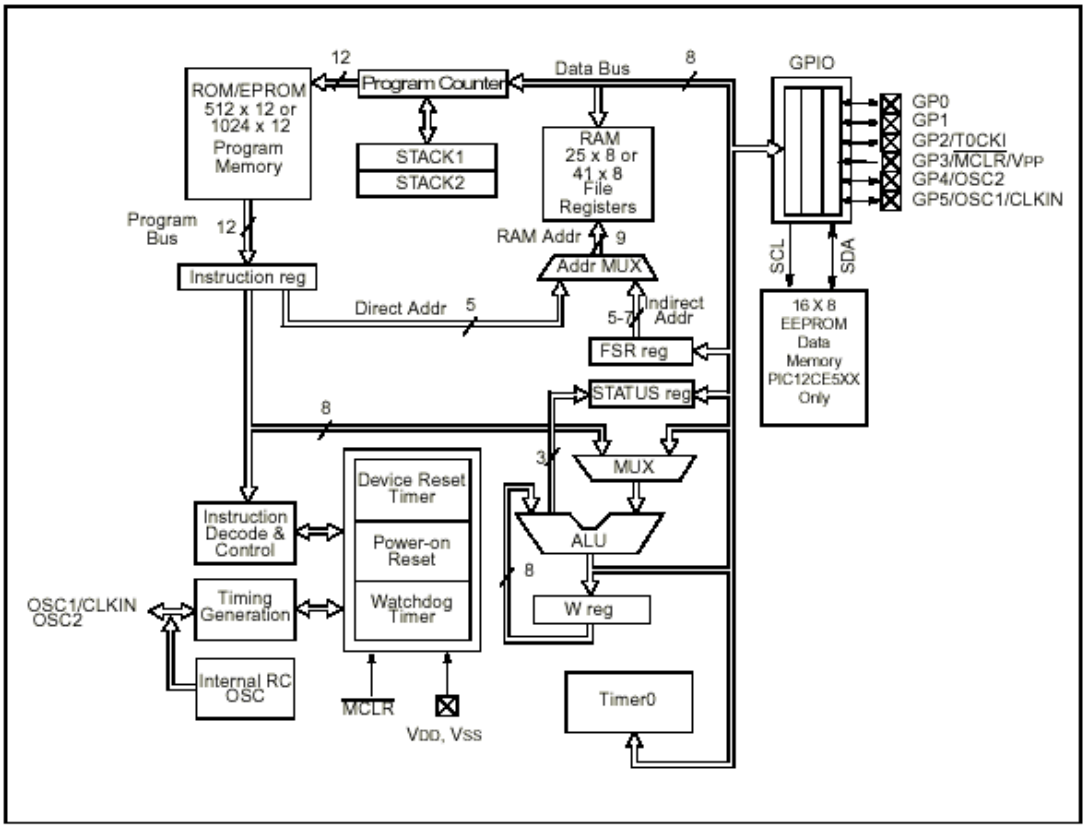


图 1.2　PIC12C5XX 内部结构

§1.4　指令周期和流水作业

PIC12C5XX 的指令周期被分频成 4 个不重叠的节拍 Q1～Q4，程序计数器 PC 在 Q1 节拍增 1，而指令是在 Q4 节拍从程序存储器中取出并置入指令译码器，并在下一个指令周期被执行， 如下图所示：

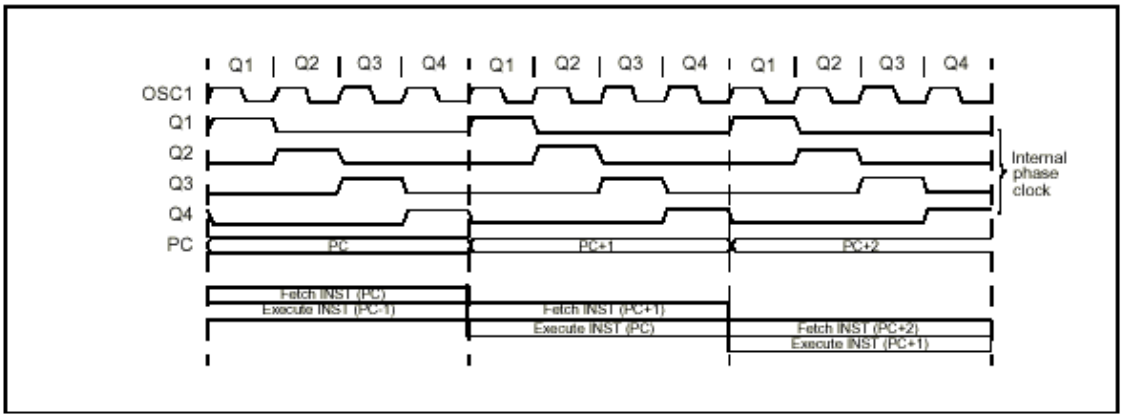


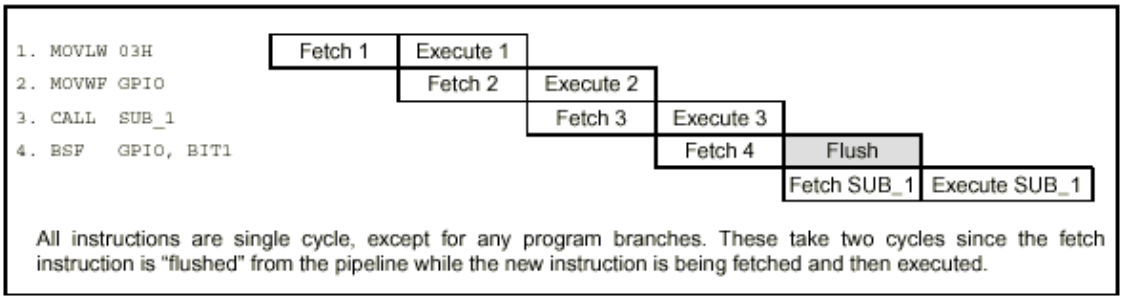
图 1.3　指令周期

指令的执行贯穿 Q1～Q4 节拍。

如上所述，当 CPU 在执行一条指令的同时， 下一条指令的代码也同时被取出置入指令译码器，准备在下一指令周期执行，这就是 PIC 的流水作业方式，也是 RISC 结构单片机的特点，这种特点使单片机的运行速度可以达到很高。

除了地址分支跳转指令的执行周期是 2 个指令周期外，其余所有指令都是单周期指令， 见下图：

图 1.4　流水作业



§1.5　程序存储器和堆栈

PIC12C5XX 的程序存储器为 12 位长，其中 PIC12C508 为 512 字节，而 PIC12C509 为 1024 字节。复位向量为地址 0，因为最后一个字节（PIC12C508 为地址 1FFH，PIC12C509 为地址 3FFH）存放有片内 RC 实际振荡的校正系数，其形式为指令 MOVLW XX，用户不要使用这个字节，所以用户的程序应从地址 000H 开始存放，注意这点和 PIC16C5X 有所不同。

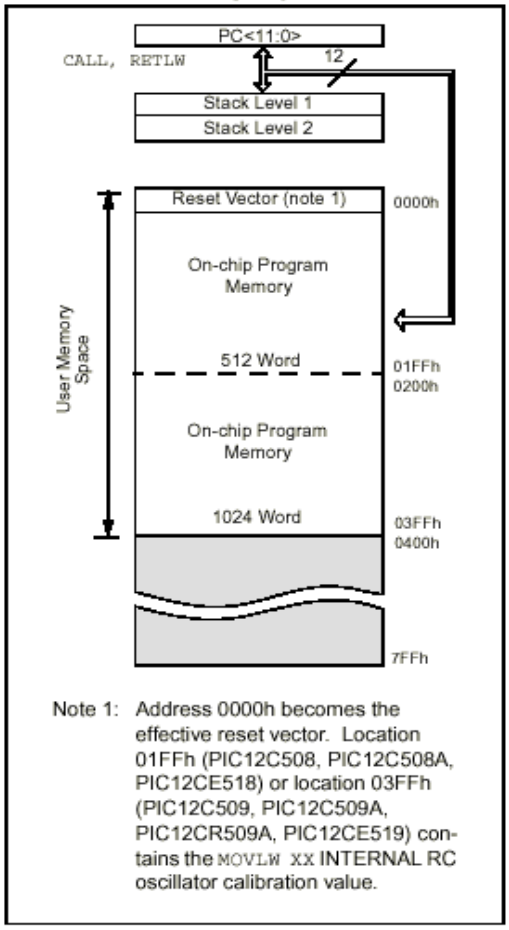


图 1.5　程序存储器和堆栈

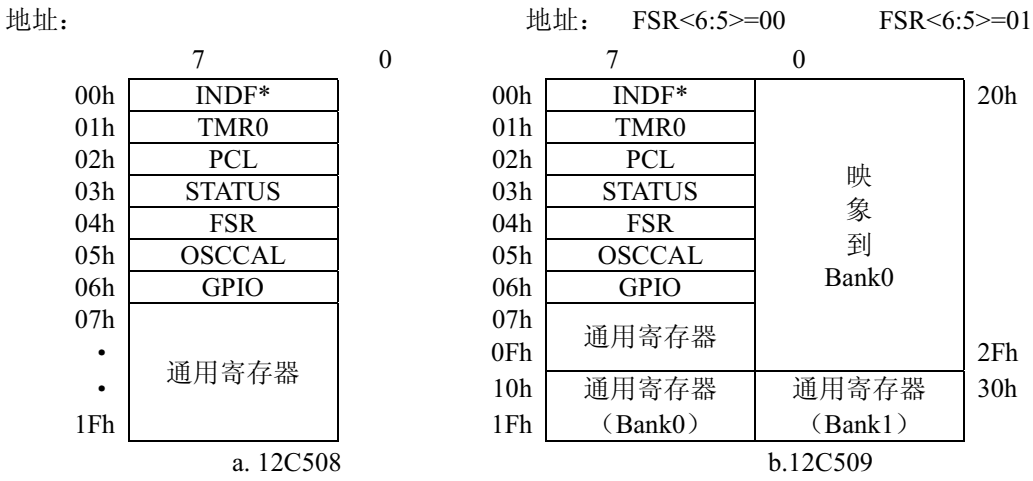
PIC12C5XX 把程序存储器以 512 字节为单位进行分页管理，这样 PIC12C508 有一个页面程序区，而 PIC12C509 有 2 个页面程序区，由状态寄存器 STATUS 中的 PA0 位（STATUS<5>） 确定程序区的页面。这是因为 PIC 是 RISC 结构，所有指令都是单字节，在 PIC12C5XX 中， 一条指令中所包含的地址信息只有 9 位，只能直接寻址一个页面（512 字节）；对于 12C509，则还要由 PA0 位来辅助寻址 2 个页面（1024 字节）的程序空间，即程序当需从一个页面跳转到另一个页面时（CALL、GOTO 指令），应事先根据要跳转去的页面，把 PA0 位置为相应的值，请参阅状态寄存器的描述。

PIC12C5XX 的堆栈有 2 层，有自己独立的空间，不占用程序存储器。注意它只能容纳二层子程序嵌套调用。堆栈的长度是 12 位，和 PC 长度一致，可以存放子程序调用时的 PC 值。

对堆栈的压入操作由子程序调用指令 CALL 完成，出栈操作则由子程序返回指令 RETLW 完成，请参阅第二章中这二条指令的详介。

§1.6 数据存储器

PIC12C5XX 的数据存储器（RAM）由一些寄存器组成，分为特殊寄存器和通用寄存器二种。在 PIC 单片机中，对任何部件的操作都表现为对某一寄存器的操作，所以编程非常简单明了。



* ： 非实际存在的寄存器，参见 §1.6.1 中详介。

图 1.6 寄存器结构

从上图可看到，00h～06h 为特殊寄存器，其余为通用寄存器。PIC12C508 有 25 个通用寄存器，而 PIC12C509 则有 41 个通用寄存器，其中 25 个在 Bank0，另 16 个在 Bank1，关于寄存器的 Bank 方式，请参阅 §1.6.1 的 FSR 寄存器描述。

§1.6.1 特殊寄存器

一、INDF（地址：00h） —— 间址寄存器

INDF 是一个物理上不存在的寄存器，只是一个逻辑寄存器，用来进行间接寻址，实际的寻址地址为 FSR<4:0>的值。

```
例：    MOVLW   10h
        MOVWF   FSR         ；实际地址 10h（F10 寄存器）→FSR
        MOVLW   55h
        MOVWF   INDF        ；数据 55h→F10
        INCF    FSR         ；FSR 增 1（FSR=11h）
        MOVWF   INDF        ；数据 55h→F11
```

参阅后面 FSR 寄存器的描述。

二、TMR0（地址：01h） —— 定时器/计数器寄存器

TMR0 对应于 TIMER0，它是一个 8 位的定时器/计数器（在 PIC16C5X 中称其为 RTCC），请参阅 § 1.8 详介。

三、PCL（地址：02h） —— 程序计数器 PC<7:0>

PIC12C5XX 程序计数器 PC 最多可寻址 1K（1024）程序区：

型　　号	PC 长度	寻址空间	PC 复位值
PIC12C508	9	512	1FFh
PIC12C509	10	1024	3FFh

单片机一复位，PC 值被置为全“1”指向程序区的最后一个字节。前面我们提过，这个地址存放的是芯片出厂时已放入的 MOVLW XX 指令（其中 XX 是片内振荡校正系数），所以单片机复位后会执行这条指令，然后 PC 马上翻转到 000h，开始执行用户的程序代码。注意，页面选择位 PA0 复位时也被清零，所以这时页面处于 0 页，请参阅有关状态寄存器 STATUS 的描述。

对于“GOTO”指令，它的指令码中含有跳转地址的低 9 位，即 PC<8:0>，对于 PIC12C509 来说，状态寄存器的第 5 位（STATUS<5>）还会被置入 PC<9>，以选择程序页面，从而寻址 1K 的程序空间。

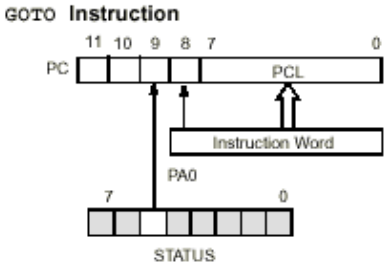


图 1.7　GOTO 指令寻址方式

对于“CALL”指令或其他涉及会修改 PCL 的指令，它们的指令码中仅包含目的地址的低 8 位，即 PC<7:0>，而 PC<8>总是会被硬件自动清零，状态寄存器第 5 位（STATUS<5>）也会被置入 PC<9>以选择程序页面（对于 PIC12C509 而言）。见下图：

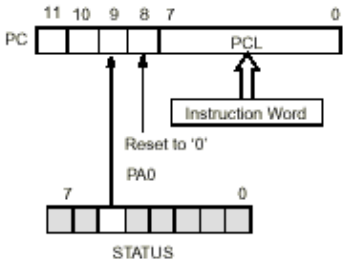


图 1.8　CALL 指令或修改 PCL 的指令寻址方式

从上图可看出，由于执行这些指令硬件总会清 PC<8>=0，所以它们的起始地址都必须限于放在每个程序页面的上半区，即头上的 256 个字节空间内（0h～FFh 或 200h～2FFh）。

四、STATUS（地址：03h） —— 状态寄存器

STATUS 寄存器包含了 ALU 的算术状态、芯片复位状态、程序页面位等信息。STATUS 可以被读/写，但是其中的复位状态位 TO、PD 不能由软件设置，它们的状态如何决定 § 1.12.7 会有详细描述。

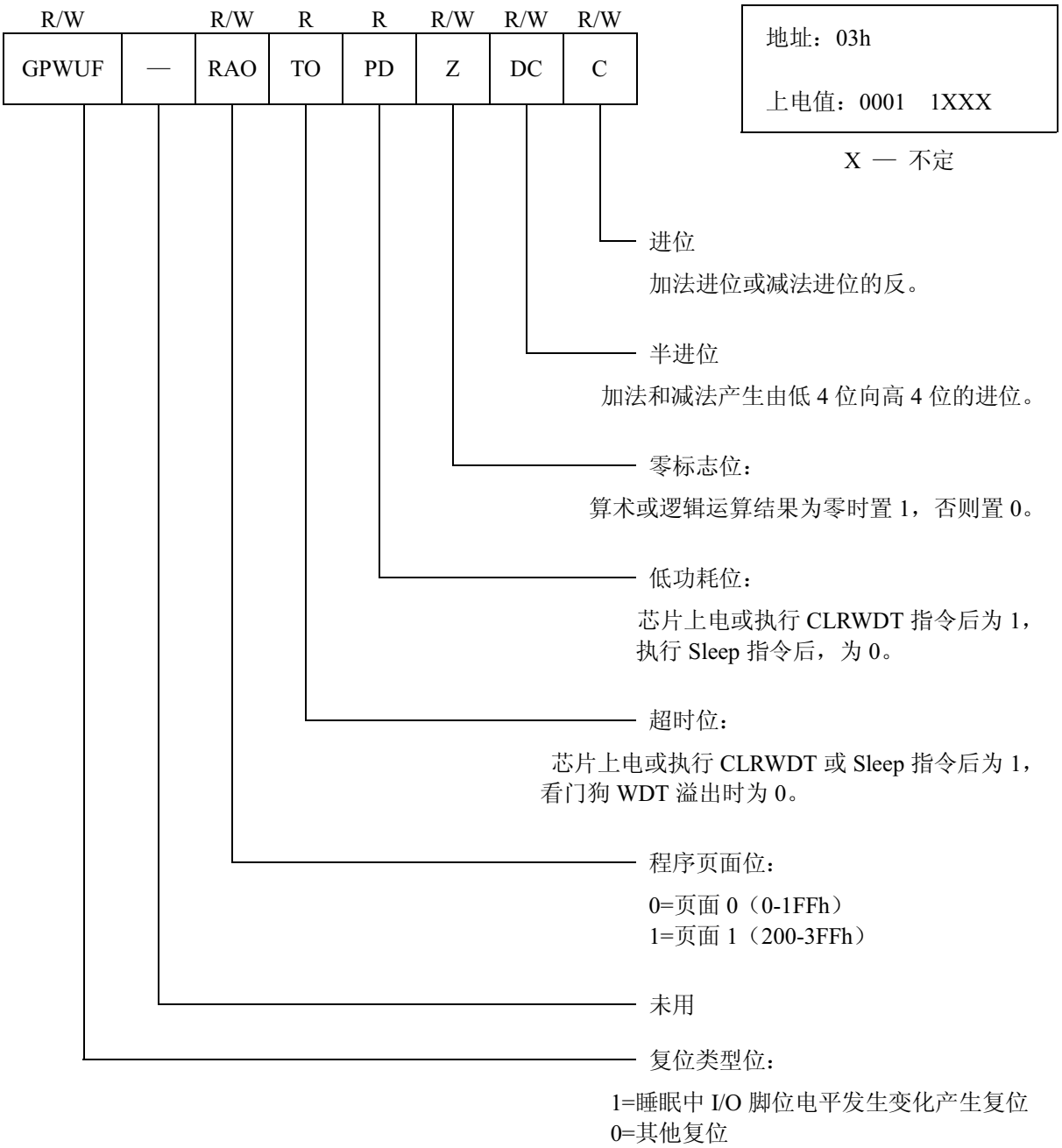


图 1.9　状态寄存器

在加法运算时，C 是进位位；在减法运算时，C 是借位的反。

```
例 a：      CLRF    F10     ；F10=0
            MOVLW   1       ；W=1
            SUBWF   F10     ；F10-W=-1（FFH），C=0（运算结果为负）
例 b：      MOVLW   1       ；W=1
            MOVWF   F10     ；F10=1
            CLRW            ；W=0
            SUBWF   F10     ；F10-W=1，C=1（运算结果为正）
```

PD 和 TO 两位可用来判断芯片复位的原因，GPWUF 位也是用来判断芯片复位类型，请参阅 § 1.12.7 描述。

五、FSR（地址：04h） —— 选择寄存器

FSR 和 INDF 寄存器（地址：00h）配合完成间接寻址，请参阅前面有关 INDF 寄存器的描述。FSR 寄存器宽度为 5 位，FSR<4:0>用来间接寻址 32 个寄存器，FSR<5> 则用来选择寄存器体（Bank），

见下图：

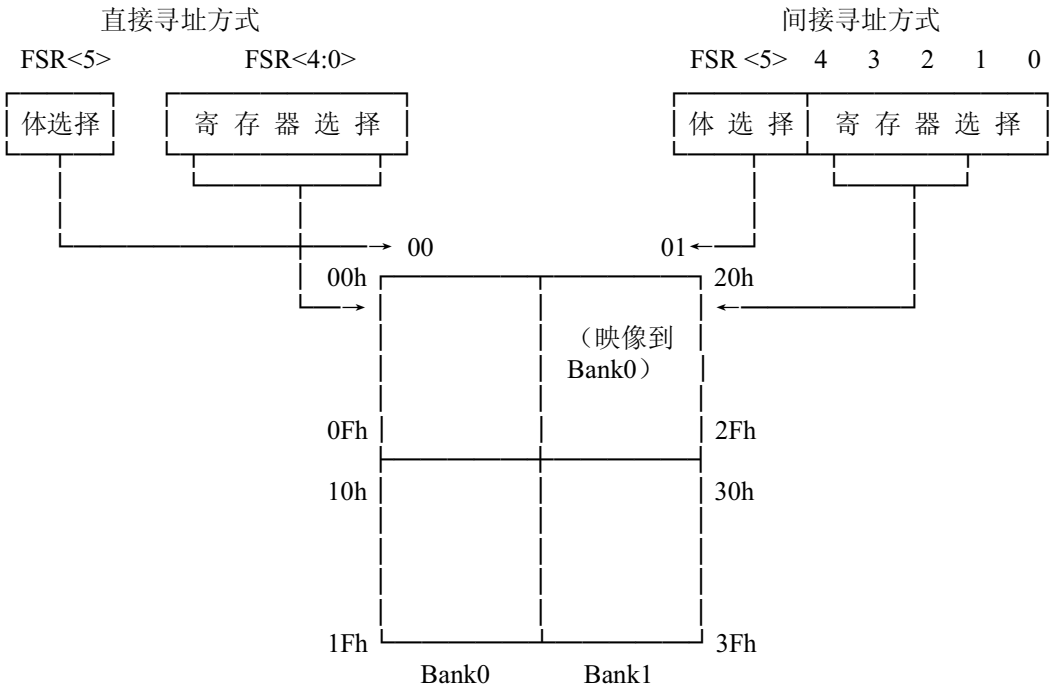


图 1.10　直接/间接寻址方式

a、PIC12C508：　不存在寄存器体选，FSR<5>恒为“1”。

b、PIC12C509：　FSR<5>=1　Bank1，

FSR<5>=0　Bank0。

六、OSCCAL（地址：05h） —— 内部振荡校正系数寄存器

PIC12C5XX 内部集成有 RC 振荡供用户选择使用，OSCCAL<7:4> 包含了该振荡电路的校正系数，其上电初始值为“0111”，请参阅 § 1.11.4 有关内部 RC 振荡的描述。

七、GPIO（地址：06h） —— I/O 寄存器

PIC12C5XX 有一个 6 位的 I/O 口，它在寄存器中的映像就是 GPIO 寄存器，GPIO<5:0>对应于 I/O 口线 GP5:GP0，GPIO<7:6>未用，恒为“0”。

八、TRIS —— I/O 方向控制寄存器

TRIS 是 GP 口线方向控制寄存器，用户不能直接寻址，必须通过执行“TRIS 6”指令来设置它。当执行“TRIS 6”指令后，W 寄存器的内容即会被置入 TRIS 中。“1”将相应的 I/O 口线设为输入态（高阻态），“0”则被设为输出态。但是有二点例外，即 GP3 永远是输入态而 GP2 有可能由 OPTION 寄存器设置为输入态（T0CKI），而不理会 TRIS 中的设置内容。请参阅 § 1.2 关于 I/O 口的描述。

```
例：    MOVLW   0Fh     ；W=“00001111”
        TRIS    6       ；TRIS=“001111”，GP0:GP3 为输入态
                                GP4:GP5 为输出态
```

各种复位都会置 TRIS 为全“1”。

九、OPTION —— 参数定义寄存器

OPTION 用来定义一些芯片工作参数，见下图所示：

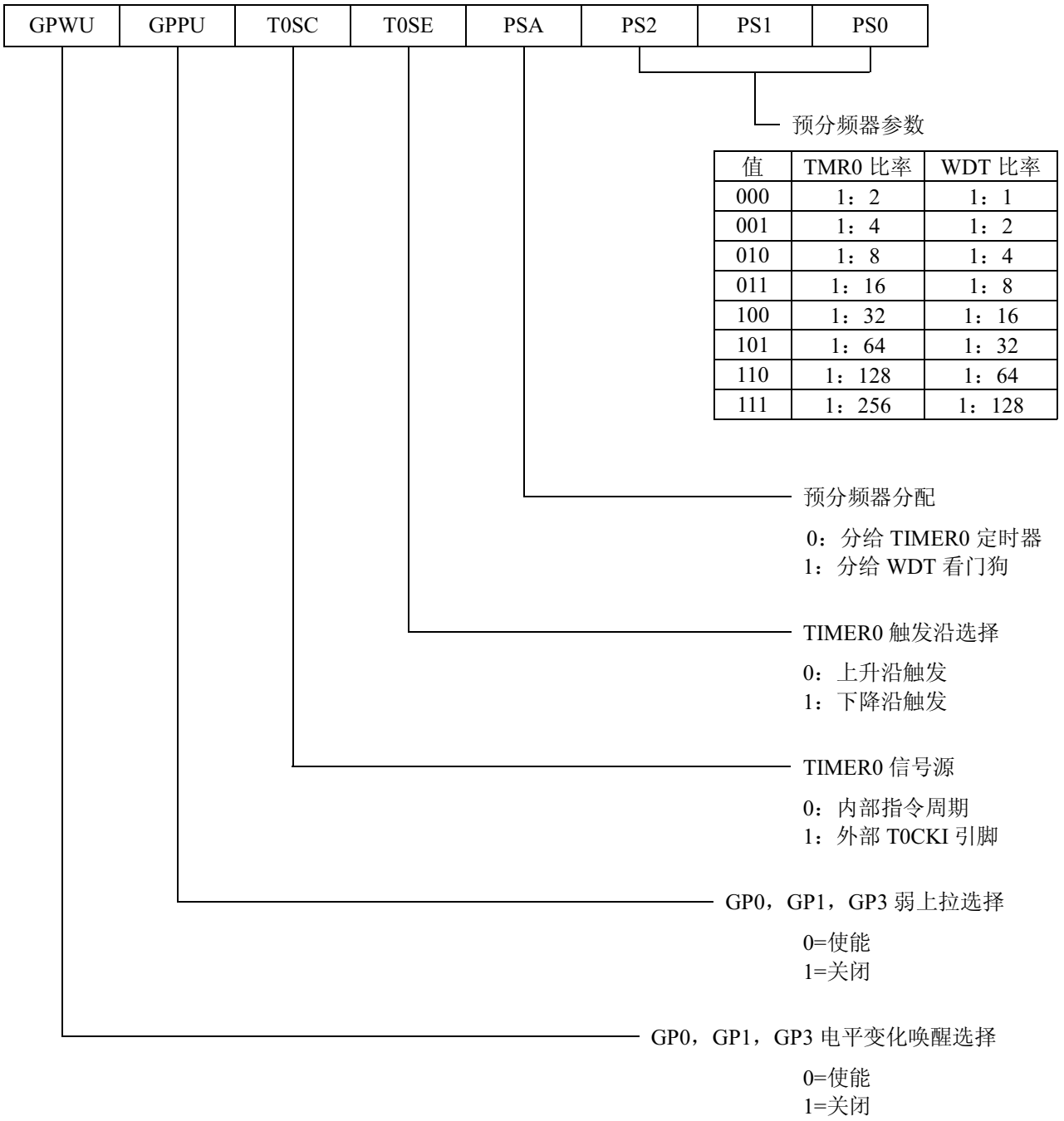


图 1.11　OPTION 寄存器

OPTION 也是不能由用户直接寻址的，必须由执行“OPTION”指令来把 W 寄存器中的内容置入 OPTION 寄存器，如下例：

```
MOVLW     7          ；W=“00000111”
OPTION               ；W→OPTION
```

各种复位都会置 OPTION 为全“1”。

注意即使 TRIS 中相应的 GP2 方向位是“0”，如果将 TOCS 置为“1”，则 GP2 也会被强置为输入态，即为 T0CKI 输入线。

有关 OPTION 各位的定义，请参阅各自相应的章节。

十、W —— 工作寄存器

W 寄存器用来存放指令中的第二个操作数，或用来进行内部数据传送，或存放运算结果，是最常用的寄存器。

§1.6.2　通用寄存器

PIC12C508：　07h — 1Fh　　；Bank0
PIC12C509：　07h — 1Fh　　；Bank0
　　30h — 3Fh　　；Bank1

通用寄存器在上电后的值是随机的，所以它属 RAM 性质。

§1.7　I/O 口

PIC12C5XX 只有一个 I/O 口，对应的映像寄存器为 GPIO（地址：06h），其中 GPIO< 5:0> 对应 GP5:GP0，GPIO<7:6>未用，永为“0”。注意，GP3 仅可作为输入，是单向 I/O 口线。另外，GP5、GP4、GP3 及 GP2 还可以由用户定义成各种特殊功能口线，一旦它们被用作特殊用途，则永远读为“0”。GP0、GP1 和 GP3 还带有可编程的弱上拉和“电平变化唤醒功能”（即唤醒正处于睡眠状态下的芯片），关于这点请参阅 OPTION 寄存器的描述。如果 GP3 被用户定义为复位输入端（MCLR），则它的弱上拉自动有效，但“电平变化唤醒”特性被自动关闭。

GPIO 口线的方向由 TRIS 寄存器控制，详情参见§1.6.1 中有关 TRIS 寄存器的描述。

§1.7.1　I/O 口结构

一根 I/O 口线的结构如下图所示：

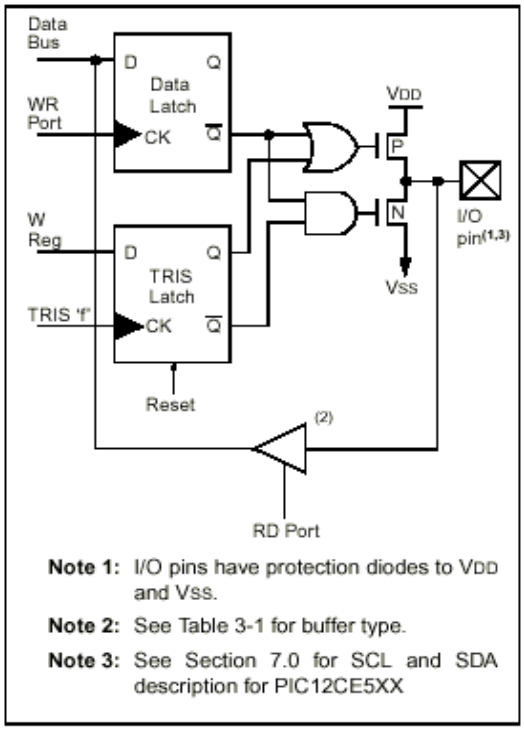


图 1.12　I/O 口结构

除了 GP3 只能单向作为输入口外，其余的 GPIO 口皆可由用户定义为输入/输出态。作为输入口时没有锁存，外部信号必须保持到让 CPU 读入为止（例如：MOVF　GPIO，W）。作为输出则有锁存，可以保持直到被新的值取代为止。

I/O 端的输入/输出态由 TRIS 寄存器的值控制，当 TRIS 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 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 1.13 所示。

§1.7.2 I/O 口使用注意事项

1、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）。

举个例子来说：“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 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。

2、I/O 的“线或”和“线与”

从图 1.12 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“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 端最大电流来决定。

3、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 1.13）。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
```

4、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。最保险的方法当然是对 I/O 读写前都定义一下 I/O 控制寄存器（但是实践证明对于大多数的应用都不必做到这样，只是提请你注意噪声干扰）。

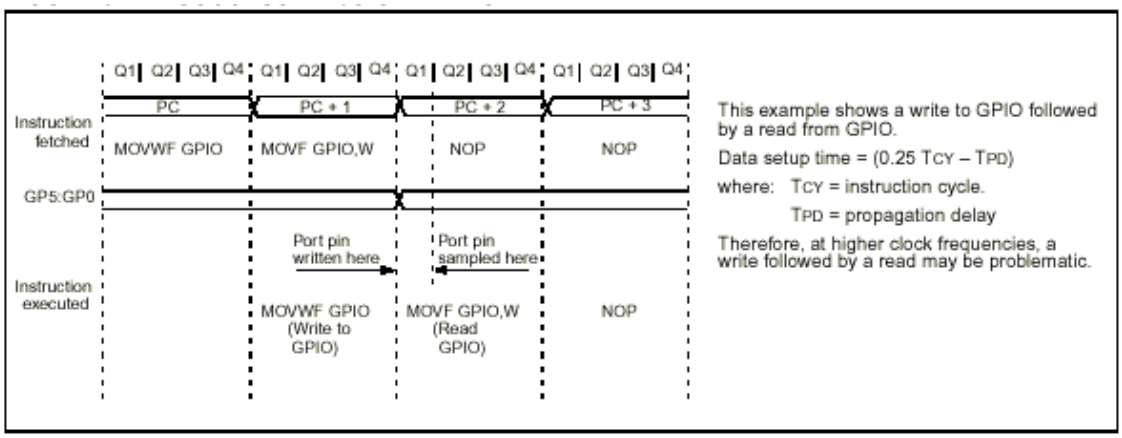


图 1.13　I/O 口连续读/写时序

§1.8　定时器/计数器 TIMER0

TIMER0 是一个 8 位的定时器/计数器，其对应的映像寄存器是 TMR0（地址：01h），可由用户程序直接读写，并且可带有 8 位的预分频器。它用于对外加在 GP2/T0CKI 引脚上的外部信号进行计数（计数器）或对内部指令时钟进行计时（定时器），在 PIC16C5X 中它被称为 RTCC。

TIMER0 及其相关电路如图 1.14 所示。从图中可看出 TIMER0 工作状态由 OPTION 寄存器控制，其中 OPTION 寄存器的 T0SC 位用来选择 TIMER0 的计数信号源，当 T0SC 为“1”时，信号源为内部时钟，T0SC 为“0”时，信号源为来自 T0CKI 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”， 分配给 TIMER0，即外部或内部信号经过预分频器分频后再输出给 TIMER0。预分频器的分频比率由 OPTION 内的 PS0～PS2 决定。这时涉及写 TMR0 寄存器的指令均同时将预分频器清零，OPTION 寄存器内容保持不变，即分配对象、分频比率等均不变。OPTION 的 T0SE 位用于选择外部计数脉冲触发沿。当 T0SE 为“1”时为下降沿触发，为“0”时则上升沿触发。

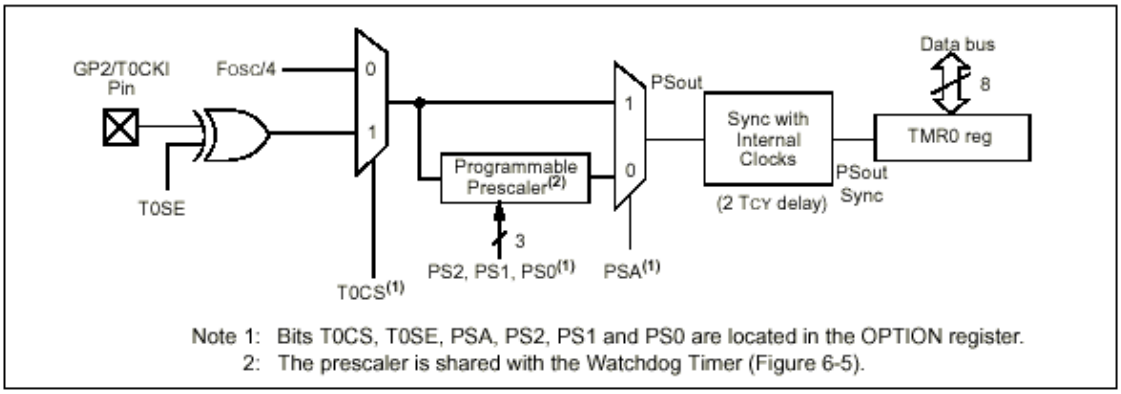


图 1.14　TIMER0 方块图

TIMER0 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，从此一直循环下去。TIMER0 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 1.15。

TIMER0 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 T0CKI 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。同时，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。

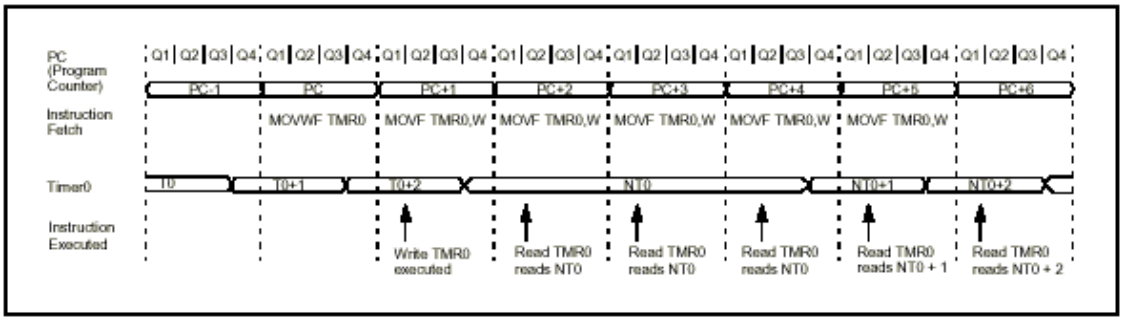


图 1.15a.　TIMER0 时序图：内部时钟/无预分频器

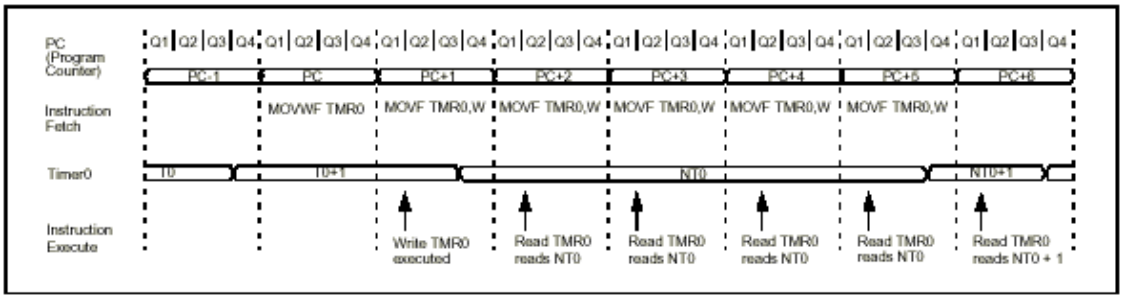


图 1.15b.　TIMER0 时序图：内部时钟/预分频比 1：2

应注意的是尽管 PIC 对外部加于 T0CKI 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 TIMER0 检测不到这个信号。一般要求信号宽度要大于 10ns。

§1.9　预分频器

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 1.14 所示上节对预分频参数已有描述，这里不再赘述。

预分预器的分配对象完全由程序控制，可以在程序中改变 Prescaler 分配对象。

1、从 TIMER0 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    1              ；='001'，则暂时先选一个另外的值
        MOVLW   B'XXXX1XXX'    ；清零 TMR0 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'    ；选择新的预分频器
        OPTION
```

2、从 WDT 到 TIMER0 的改变

```
        CLRWDT                 ；清 WDT 及 Prescaler
        MOVLW   B'XXXX0XXX'    ；选择 TIMER0
        OPTION
```

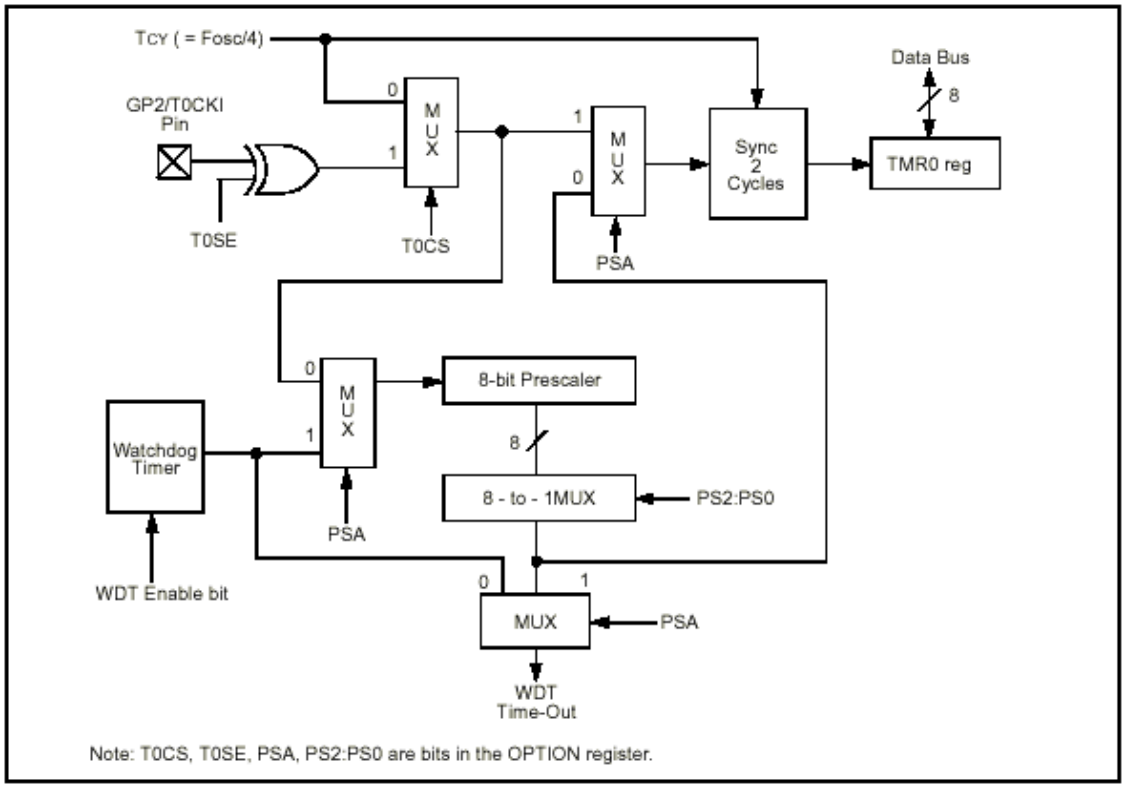


图 1.16　预分频器方块图

注意，预分频器只能分配给 TIMER0 或 WDT 其中之一使用，而不能同时分配给二者。

§1.10　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 PIC12C5XX 芯片内有一个特殊的谓之“系统定义字”（Configuration EPROM）的单元，其中的一个位是用于定义 WDT 的，可以将其置“0”来抑制 WDT 使之永远不起作用。这将在第六章的烧写器介绍部分详细说明。

1、WDT 周期

WDT 有一个基本的溢出周期 18ms（无预设倍数），如果需要更长的 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（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“失控”。在正常情况下，WDT 应在计时溢出前被 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 在溢出前能被清零，否则会造成芯片不停地产生 RESET，使系统无法正常工作。

在噪声工作环境下，OPTION 寄存器可能会因受干扰而改变，所以最好每隔一段时间就将其重新设置一下。

§1.11 振荡

PIC12C5XX 可以运行在以下四种振荡方式下：

a、LP 低功耗低速晶体振荡
b、XT 标准晶体/陶瓷振荡
c、INTRC 内部 4MHz RC 振荡
d、EXTRC 外部 RC 振荡

以上四种振荡方式可由“系统定义字”中的 Fosc1：Fosc2 两位来选择，请读者参阅后面 §1.12.9 的详述。

§1.11.1 晶体/陶瓷振荡

这种振荡包括 XT 和 LP，其电路连接是在 GP5/OSC1/CLKIN 和 GP4/OSC2 两端加一晶体/陶瓷振荡，如下图所示：

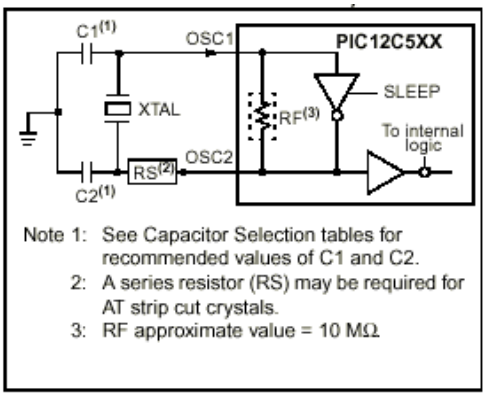


图 1.17 晶体/陶瓷振荡电路

下表是使用各种频率的晶体和陶瓷振荡所需的 C1、C2 电容值。

振荡类型	频率	C1	C2
XT	455KHz	68-100P	68-100P
	2MHz	15-33P	15-33P
	4MHz	10-22P	10-22P

a. 陶瓷振荡

振荡类型	频率	C1	C2
LP	32KHz	15P	15P
XT	100KHz	15-30P	200-300P
	200KHz	15-30P	100-200P
	1MHz	15-30P	15-30P
	2MHz	15P	15P
	4MHz	15P	15P

b.晶体振荡

表 1.3 各种振荡下的 C1 和 C2 值

§1.11.2 外部 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 1.18 所示。

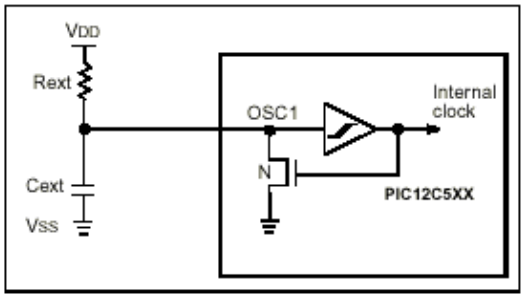


图 1.18 RC 振荡电路

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 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 1.4 所示。RC 振荡时 OSC2 端输出一 OSC1 的 4 分频脉冲（f=1/4OSC1）。

Rest	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	480MHz
10kΩ	290PF	5.0	245MHz
100kΩ	300PF	3.5	30MHz

表 1.4　RC 与频率的关系

§1.11.3　外部振荡

PIC12C5XX 也可以接受外部振荡源（仅适合于 XT 和 LP 类型振荡），连接时将外部振荡接入 GP5/OSC1/CLKIN 端，见下图：

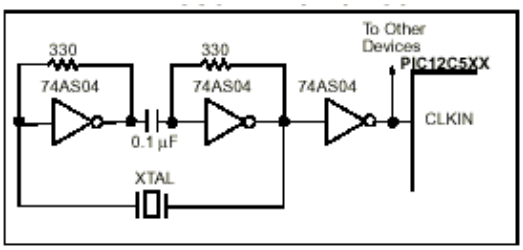


图 1.19　外部振荡源输入电路

§1.11.4　内部 RC 振荡

PIC12C5XX 内部提供有 4MHz 的 RC 振荡源供用户选择使用，选择振荡方式和振荡源的方法见§1.12.9 详介。

在 PIC12C5XX 的程序区最顶端（12C508：1FFh，12C509：3FFh）放了一条 MOVLW XX 的指令， XX 是内部 RC 振荡的校正系数。芯片上电后，PC 指针指向程序区最顶端，执行完这条指令后 PC 值加 1 变为 000h。这时 W 寄存器中存放即是内部 RC 振荡的校正系数，用户可以把这个系数置入 OSCCAL 寄存器（05h）以便使其起校正作用，也可以忽略不管它。

```
例：        ORG     0               ；定义存储区地址 0
            MOVWF   OSCCAL          ；把 W 中的校正系数置入 OSCCAL。
```

§1.12　复 位（RESET）

PIC12C5XX 有各种各样原因造成的芯片复位：

1、芯片上电

2.、MCLR 端加低电平

3、看门狗 WDT 超时溢出

4、睡眠中某些 I/O 口线电平发生变化

当芯片处于复位状态时，所有 I/O 口线都处于输入状态（高阻态），看门狗 WDT 和预分频器都被清零。

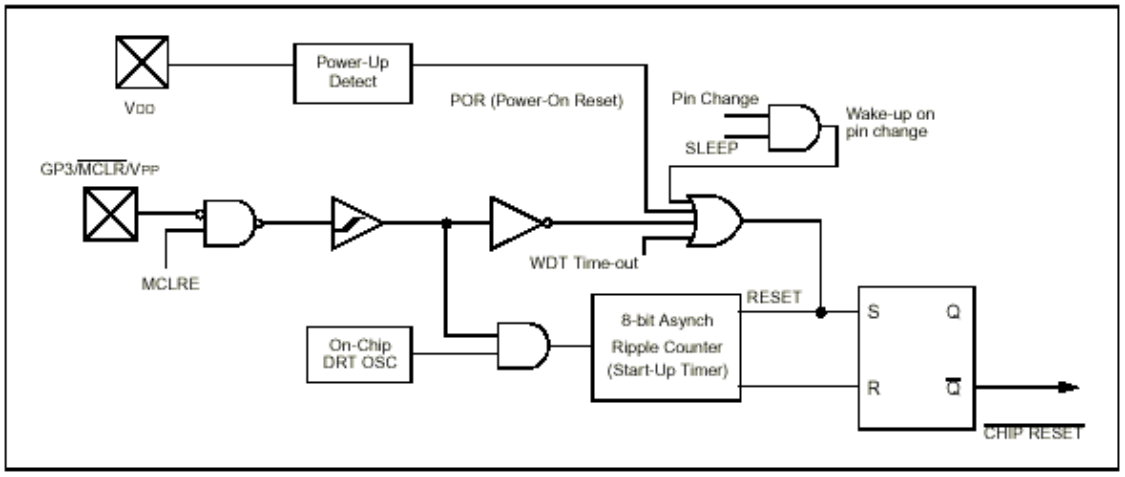


图 1.20　片内复位电路

§ 1.12.1　复位定时器（DRT）

复位定时器 DRT（在 PIC16C5X 中我们称其为 OST）是为了使芯片的复位可靠安全而设计。在 PIC12C5XX 中，对于 XT 和 LP 振荡方式，上电后它们还需要一定的时间来建立稳定的振荡。有鉴于此，PIC12C5XX 内部设计了一个复位定时器 DRT。DRT 在 MCLR 端到达高电平（VIHMC）后，即启动计时 18ms，这样可以使芯片保持在复位状态约 18ms 以便让振荡电路起振及稳定下来，然后芯片即脱离复位状态进入正常运行状态。DRT 的振荡源是芯片内专有的 RC 振荡电路，所以外围电路并不能改变其 18ms 的计时时间。

当 WDT 计时溢出后，DRT 也是启动 18ms 使芯片保持在复位状态，然后再重新开始运行程序。

注意，在振荡方式是外部 RC 或内部 RC 时，DRT 都关闭不起作用。

§ 1.12.2　芯片上电复位（POR）

PIC12C5XX 在芯片内集成有上电复位电路，见图 1.20 所示。当芯片电源电压 VDD 上升到一定值时（1.5V-2.1V），检测电路即会发出复位脉冲使芯片复位。

§ 1.12.3　MCLR 复位

PIC12C5XX 的 GP3/MCLR 端可以由用户定义为普通输入口 GP3 或复位端 MCLR，如下图：

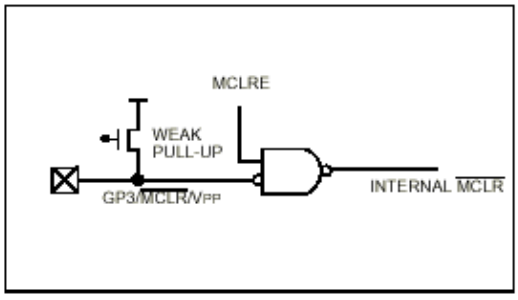


图 1.21　GP3/MCLR 端电路

具体方法参见 § 1.12.9 有关描述。

一旦用户选择 MCLR 功能，则该端输入低电平会使芯片进入复位状态。

§1.12.4　外部复位电路

在某种情况下，DRT 计时 18ms 后，芯片的振荡电路还不能稳定或供电电压（VDD）还不能达到标准值，这时如果芯片脱离复位状态进入运行，则芯片就有可能失控或运行不正常。为了使芯片脱离复位状态时各部分都处于正常，可以在 MCLR 端上加外部 RC 复位电路来延长复位时间，如下图：

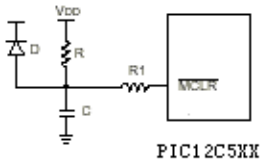


图 1.22　外部复位电路

这个电路可以使 VDD 上升到标准值一段时间后，MCLR 才会上升到高电平，从而启动 DRT 计时 18ms 后才进入运行。这样可以延长整个复位过程，保障芯片复位后进入正常运行。

§1.12.5　掉电复位锁定

当单片机的供电电压掉到最小标准值以下后，可能会使芯片的运行出现异常，从而扰乱整个控制系统，所以在某些应用中，我们希望一旦 VDD 掉到某个值时使芯片自动进入复位状态（所有 I/O 口都变成高阻态）以免扰乱系统，下面是一个 PIC12C5XX 掉电复位锁定的电路：

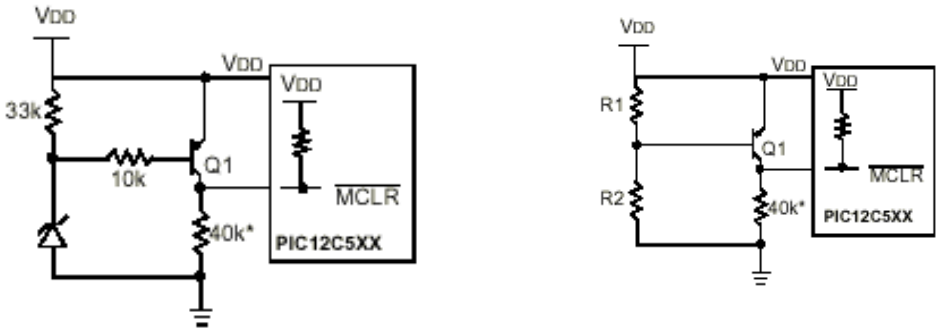


图 1.23　掉电复位锁定

当 VDD 电压恢复上升到标准值以上后，MCLR 端恢复为高，从而使芯片恢复正常运行。

§1.12.6　复位对寄存器的影响

对于通用寄存器来说，上电复位后它们的值是随机不定的，其他类型的复位后则保持原值不变。对于特殊寄存器，各种复位后它们都会等于一个固定的复位值，见以下二表：

寄存器	地址	上电复位值	MCLR 复位 WDT 复位 引脚变化唤起复位
W	—	qqqq　xxxx（注 1）	qqqq　uuuu（注 1）
INDF	00h	xxxx　xxxx	uuuu　uuuu
TMR0	01h	xxxx　xxxx	uuuu　uuuu

PC	02h	1111 1111	1111 1111
STATUS	03h	0001 1xxx	?00? ?uuu（注 2）
FSR（12C508）	04h	111x xxxx	111u uuuu
FSR（12C509）	04h	110x xxxx	11uu uuuu
OSCCAL	05h	0111 ----	Uuuu ----
GPIO	06h	--xx xxxx	--uu -----
OPTION	—	1111 1111	1111 1111
TRIS	—	--11 1111	--11 1111

u： 未变； x： 随机值； -： 未用； ?： 其值取决于复位方式

注 1：由于在复位向量处存放着 MOVLW XX 指令，其中 XX 为内部 RC 振荡校正系数，所以复位后 W<7:4>即会等于这个值。

注 2：参见表 1.6。

a. 各特殊寄存器复位后的值

复 位 类 型	状态寄存器 STATUS	程序计数器 PC
芯片上电复位	0000 1xxx	1111 1111
运行时 MCLR 端加低电平复位	000u uuuu	1111 1111
睡眠时 MCLR 端加低电平复位	0001 0uuuu	1111 1111
睡眠时看门狗 WDT 超时复位	0000 0uuu	1111 1111
运行时看门狗 WDT 超时复位	0000 1uuu	1111 1111
睡眠时 I/O 脚电平变化唤醒复位	1001 0uuuu	1111 1111

u：未变； x：随机.

b. 复位对 STATUS 和 PC 的影响

表 1.5 各种复位对特殊寄存器的影响

§1.12.7 复位的鉴别

PIC12C5XX 有多种原因都可引起芯片复位。在程序中判断芯片复位的原因有时是非常必要的，例如上电复位后程序一般都要做一些寄存器初始化工作，而别的复位后则可以不做初始化而直接进入控制运行。

在状态寄存器 STATUS 有三个位（GRWUF、TO、PD）可用来标识各种复位状态，见下表：

GPWUF	TO	PD	复 位 原 因
0	0	0	睡眠中 WDT 超时溢出
0	0	1	运行时 WDT 超时溢出
0	1	0	睡眠中 MCLR 拉低
0	1	1	芯片上电
0	u	u	运行时 MCLR 拉低
1	1	0	睡眠中 GP0，GP1 或 GP3 电平变化

u：未变

a. 复位后 TO、PD 及 GPWUF 的状态

事 件	GRWUF	TO	PD	注
芯片上电	0	1	1	
WDT 超时溢出	0	0	u	不影响 PD 位

执行 Sleep 指令（进入睡眠）	u	1	0	
执行 CLRWDT 指令（清看门狗）	u	1	1	
睡眠中 GP0，GP1 或 GP3 电平发生变化	1	1	0	

u：未变

b.　影响 TO、PD 及 GPWUF 位状态的事件

表 1.6　复位对 STATUS 的影响

例：要判断是否芯片上电。

```
        BTFSS    STATUS，TO        ；TO=1？
        GOTO     NO_POWERUP
        BTFSS    STATUS，PD        ；PD=1？
        GOTO     NO_POWERUP
INIT    …                          ；TO=1，PD=1。芯片上电，做初始化。
```

§1.12.8　睡眠模式（Sleep）

1、进入 SLEEP

执行一条“SLEEP”指令即可进入低功耗睡眠模式。当进入 SLEEP 后，WDT 被清零，然后重新开始计数。状态寄存器 STATUS 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。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）。

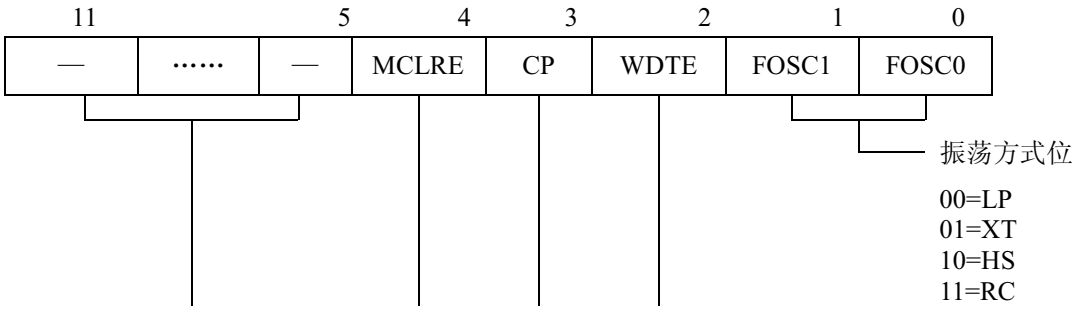
MCLR 必须处于高电平状态。

2、唤醒 SLEEP

SLEEP 可被 WDT 溢出唤醒，或在 MCLR 端加低电平唤醒 SLEEP 或 GP0、GP1、GP3 电平发生变化。第二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。第三种方法则在需要使用系统时唤醒睡眠中的单片机，它常通过按键输入来实现。系统上电时，STATUS 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。STATUS 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。

§1.12.9　系统定义字（Configuration）

在 PIC12C5XX 中有一个 12 位长的系统定义字单元，其中只用了前 5 位（bit0～bit4），用来定义单片机的一些系统性能选择，如下图：



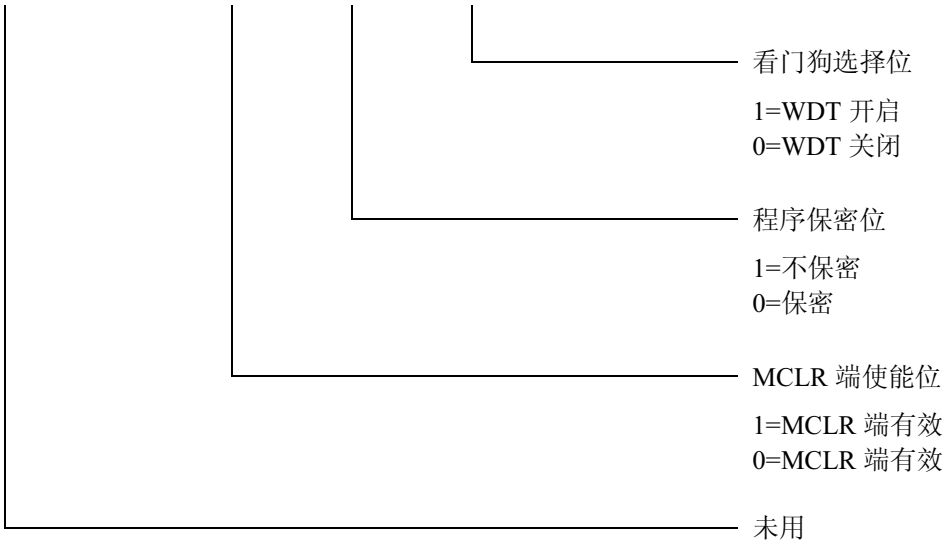


图 1.24　系统定义字

系统定义字属特殊的空间，不占用芯片的程序存储器，不能由程序指针（用户程序）访问，用户可以用烧写器对其进行编程，参见烧写器章节中的描述。

程序保密位被置为“0”后，程序存储区中的程序代码（12 位）中的高 8 位将被遮没。具体地说，就是加密后再用烧写器读该芯片的程序区时，每一个程序代码都呈现 00X 的形式，这样别人就无法恢复这些被加密的代码，因此也就无法进行复制拷贝。加密后的单片机的功能不会受任何影响，加密后的程序代码并不影响其在芯片内的运行，而只是不能再被还原读出来。

§1.12.10　ID 码

PIC12C5XX 芯片中有一个 16 位的标识码（称为 ID 码），用来作芯片标识。ID 码仅起芯片识别作用，用户可在烧写器上将其烧入和读出作芯片识别（如烧入日期等），但不会对芯片功能产生任何影响，即不使用它也没有关系。

第二章　PIC12C5XX 指令集及程序设计技巧

§2.1　PIC12C5XX 指令概述

PIC12C5XX 每条指令长 12 位，指令由操作码和操作数组成。PIC12C5XX 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。

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 0 则跳	BTFSS f，b	Test bit(b) in file(f):Skip if clear	Z	

常数操作和控制操作类指令				（11-8） OPCODE	（7-0） 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 PIC12C5XX 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 GP 口上的状态值，而非 GP 口输出锁存器里的值。

3、指令“TRIS 6”将 W 寄存器中的内容写入 GP 的 I/O 口控制寄存器中：“1”关断对应端口的输出缓冲器，使其为输入（高阻）状态，“0”则使其为输出态。

4、当预分频器（Prescaler）分配给 TIMER0 后，任何对 TMR0 寄存器（F1）写操作的指令都将使预分频器清零。

§2.2 PIC12C5XX 指令寻址方式

PIC12C5XX 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（INDF）、F4（FSR）来实现。实际的寄存器地址放在 FSR 中，通过 INDF 来进行间接寻址。

例：
```
FSR     EQU     4
INDF    EQU     0
        MOVLW   05H             ; W=5
        MOVWF   FSR             ; W(=5)→F4
        MOVLW   55H             ; W=55H
        MOVWF   INDF            ; 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 PIC12C508，寄存器地址（5 位）直接包括在指令中，对 PIC12C509，寄存器地址中最高 1 位由 FSR（F4）寄存器中的 bit5 决定，即体选位。

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 PIC12C509，则还要参考寄存器体选择器 FSR 的 bit5 选择存入哪一个寄存器体（bank0 或 bank1）。

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 ，1→z
影响状态位：z
说明：　　将 f 寄存器清零，状态位 Z 将被置为 1。
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，1→Z
影响状态位：　Z
说明：　　将 W 寄存器清零，状态位 Z 将被置为 1。
例：　　CLRW　　；W 清为零，Z 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：　C，DC，Z
说明：　　f 寄存器内容减 1 存入 f（d=1）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：

0010	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 寄存器内容减 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：　C，DC，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：
```
LOOP    ┌─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 MOVWFF9 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 寄存器内容和 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 寄存器传送指令
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

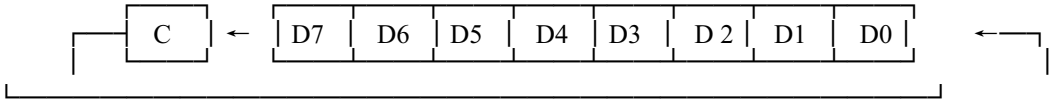
指令周期：　1

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：

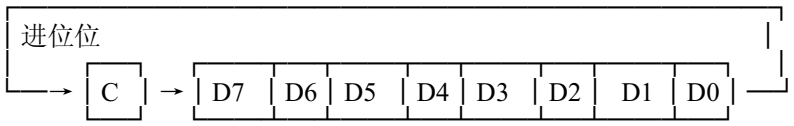
001100	d	fffff

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，2

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　面向位操作类指令

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)清为 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 INCF9，1
          └──→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 PIC12C509 来论，STATUS 的 PA0 位(页面地址)还将被置入 PC 的最高位(bit9)。所以子程序可以放在二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(bit8)清为"0"。

```
例：        CALL DELAY          ；调用子程序 DELAY
            :
            :
    DELAY   MOVLW   80H ──┐
            :             │
            :             │ ；子程序其第一条指令须放在每页面的上半区。
          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 PIC12C509，STATUS 中的 PA0 位(页面地址位)将同时置入 PC 的最高位(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： 2
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、由入低功耗状态指令

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： 停止芯片振荡，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 STATUS 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 |

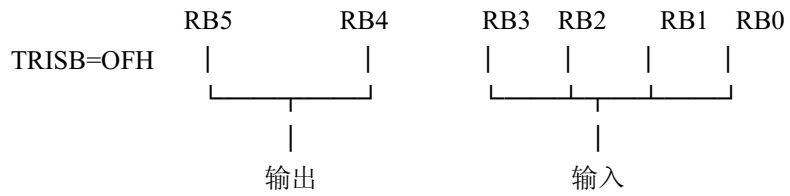
指令周期： 1

操作： W→I/O 控制寄存器 TRISf(f=6)

影响状态位： 无

说明： 将 W 寄存器内容置入 GP 口控制寄存器，以设定 GP 口的输入/输出方向。f=6，对应于 GP 口。

例：　　MOVLW　0FH　　; W=OFH
　　TRIS　　6　　; TRISB=OFH



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

PIC12C5XX 的一些指令还可以用容易记忆的助记符来表示。PIC12C5XX 的汇编 MPASM 可以认识这些助记符，在汇编时会将其转译成相应的 PIC12C5XX 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 PIC12C5XX 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 PIC12C5XX 程序设计基础

上面我们已经详细介绍了 PIC12C5XX 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、特殊功能寄存器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 PIC12C509 跨页面的 CALL 和 GOTO 操作，要事先设置 STATUS 中的页面地址位 PA0，对于 CALL 来论，子程序返回后还要将 STATUS 中的 PA0 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

PIC12C5XX 一旦复位后指令计数器 PC 被置为全“1”，所以 PIC12C5XX 几种型号芯片的复位地址为：

型 号	RESET 地址
PIC12C508	1FFh
PIC12C509	3FFh

表 2.3 复位地址表

一般说来，PIC12C5XX 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
;------------------
INDF        EQU        0
TMR0        EQU        1
PC          EQU        2
STATUS      EQU        3
FSR         EQU        4
GP          EQU        6
;--------------------------------
            ORG        0                    ; 从 000H 开始存放程序
            GOTO       MAIN
;--------------------------------
;子程序区
;--------------------------------
 DELAY      MOVLW      255
            ┆
            RETLW      0
;--------------------------------
;主程序区
;--------------------------------
 MAIN       CLRW
            TRIS       GP                   ; GP 已由伪指令定义为 6。
            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

PIC12C5XX 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 TRISf 来实现，写入值为“1”，则为输入；写入值为“0”， 则为输出。

```
MOVLW       0FH              ; 00 1111 (0FH)
                               ↑   ↑
                               |   └──输入
                               └──────输出
TRIS        6                ; W 中的 0FH 写入 GP 口控制器，GP 口高 2 位为输出，
                               低 4 位为输入 。
```

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。

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。

下面程序以 F10 放表头地址。

```
            MOVLW     TABLE            ; 表头地址→F10
            MOVWF     10
```

```
                :
                :
                MOVLW     1              ; 1→W，准备取表格中的数据。
                ADDWF     10，1          ; F10+W=表中第二个数据的地址。
                CALL      CONVERT
                MOVWF     6              ; 送到 GP 口上。
                :
                :
CONVERT     MOVWF     2                  ; W→PC
TABLE       RETLW     C0H
            RETLW     F9H
             :
             :
            RETLW     90H
```

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 4MHz 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
        MOVLW    10          ;第一计数值(100)→F10
LOOP    MOVLW    16
        MOVLW    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、TIMER0 计数器的使用

TIMER0 是一个脉冲计数器，它的计数脉冲有二个来源，一个是从 T0CKI 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。TIMER0 可被程序用作计时之用：程序读取 TMR0 寄存器值以计算时间。

下例程序以 TIMER0 做延时。

```
TMR0    EQU     1
  :
  :
CLRF    TMR0        ; TMR0 清 O
MOVLW   07H
OPTION              ; 选择预设倍数 1：256→RTCC
LOOP    MOVLW   255     ; TMR0 计数终位值
        SUBWF   TMR0，0
        SKPZ            ; TMR0=255?
        GOTO    LOOP
        :
        :
```

这个延时程序中，每过 256 个指令周期 TMR0 寄存器增 1（预分频率=1：256，设芯片使用 4MHz 振荡，则：

延时时间=256*256=65536（US）

TIMER0 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 TIMER0 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 PIC12C508，寄存器有 32 个，只有一个体（bank0），故不存在体寻址问题，对于 PIC12C509 来说，寄存器则分为 2 个体（bank0/bank1）。FSR<5>用来进行体选择，其对应关系如下：

FSR<5>	Bank	物理地址
0 1	Bank0 Bank1	10H-1FH 30H-3FH

对于 PIC12C509，当芯片上电复位后，FSR<5>被清为“0”，所以是指向 Bank0，而对于其他类型的复位则 FSR<5>保持原值不变。

下面的例子对 Bank0 和 Bank1 的 10H 及 30H 寄存器写入数据。

例 1：（设目前体选为 Bank0）

```
        BCF     FSR，5          ；置位 FSR<5>=1，选择 Bank0
        MOVLW   55H
        MOVWF   10H             ; 55H→10H 寄存器(Bank0)
        BSF     FSR，5          ; FSR<5>=1，选 Bank1
        MOVWF   10              ; 55H→30 寄存器(Bank1)
```

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 PIC12C509 的程序存贮区的页面概念和 STATUS 寄存器中的页面选址位 PA0。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
        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（Page 0），现要调用—-放在 1 页面（Page 1）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)。

4、程序页面的连接

PIC12C509 程序 2 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（PA0=0）
：	：
1FFH	
200H	BSF　STATUS，PA0
201H	： ；Page1（PA0=1）

即在进入另一个页面后，马上设置相应的页面地址位（PA0）。

第三章　两个 PIC12C5XX 应用电路

PIC12C508/509 是 8 脚封装的 8 位单片机，极适合于嵌入到各种电子装置中做智能开发，下面介绍二个 Microchip 公司给出的实例电路。

§3.1　灯光亮度调节器

- 根据房间亮度自动调节电灯亮度
- 手动调节电灯亮度

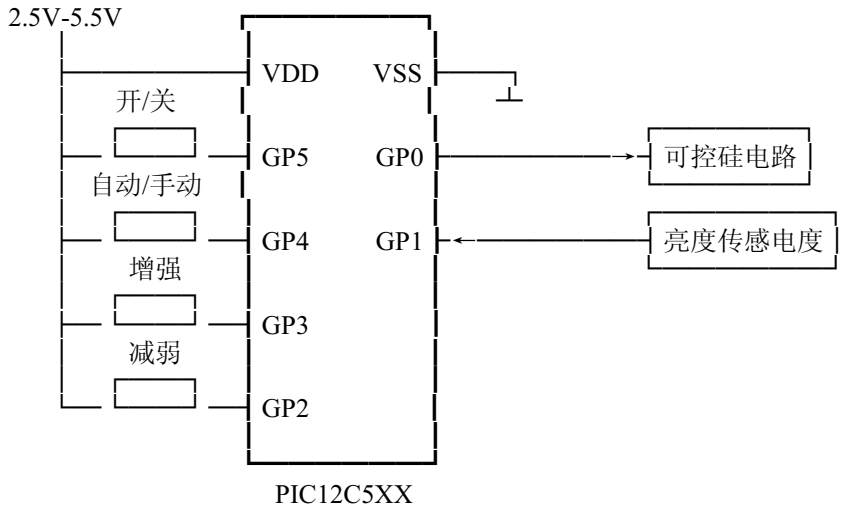


图 3.1　灯光调节器

§3.2　家庭防盗传感器

- 非法进入声/光报警
- 单片机自动报警/状态保存
- 手动开启/关闭系统

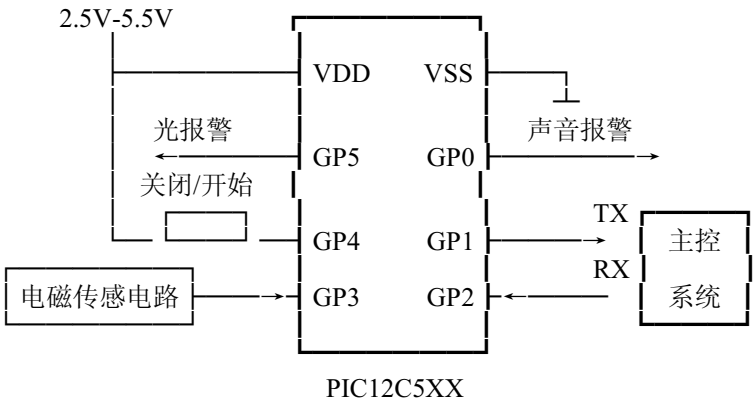


图 3.2　家庭防盗传感器

以上二例程序省略。

第四章　宏汇编器 MPASM

MPASM 是 Microchip 公司推出的可适用于其 PIC16/17 全部单片机的宏汇编器，功能齐全，全屏幕操作。

§4.1　启动和操作

MPASM 的启动很简单，在 DOS 状态下：

> MPASM <Enter>　（注意 MPASM 后面不要跟文件名）

屏幕上即显示：

```
MPASM 01.11 Released      (c)1993，94      Byte Craft Limited/Microchip Technology Inc.

                    Source File : SAMPLE.ASM

                  Processor Type : 12C509

                     Error File : Yes

               Cross Refernece File : No

                    Listing File : Yes

                 Hex Dump Type : INHX8M .HEX

             Assemble to Object File : No

──────────────────────────────────────────────────────────────────────────────

↑ ↓，Tab : Move Cursor      Esc : Quit          Press Enter to  change  value.
F1       : Help             F10 : Assemble
```

图 4.1　MPASM 画面

Source File:	源程序文件名。可以带路径和通配符(*)。
Processor Type:	芯片型号。可通过 Enter 键来选择用户所需的型号。
Error File:	汇编后自动产生一个.ERR 文件，该文件记录了汇编中产生的错误语句和警告信息。
Cross Reference File:	产生一个参考文件.XRF。
Listing File:	产生一个列表文件.LST。该文件中包含了各种仿真环境中需要的参数，主要用于仿真调试。
Hex Dump Type:	产生的代码烧写文件，一般选择 INHX8M 格式，可适应众多的烧写器。
Assemble to Object File:	注意这里产生的.OBJ 文件不是通常认为的机器代码文件，而是预留给链接器(Linker)的可重定位文件。选择 NO 则汇编不产生任何.OBJ 文件。

§4.2 汇编语言格式

PIC 汇编语句的格式为:

〔标号〕〈指令助记符〉〔操作数〕 ;〔注释〕

指令助记符与标号间至少应有一个空格。若一行语句没有标号，则指令助记符前必须至少有一个空格，否则会当成是标号。一条语句最多字符个数为 255。

```
;
; Sample MPASM Source Code. It is for illustration only.
;
        List        p=12C509，r=HEX

        org         0h                  ; 程序从 0h 处开始放
start
        movlw       0x0a
        movlw       0x0b                ;
        goto        start               ; loop

        end
```

图 4.2 汇编语言范例

一、标号

标号须由第一格起始写，最多可达 31 个字符，且第一个字符必须是字母。标号后可跟冒号（:）、空格或行结束符。除非使用选择项/C，否则标号中的字母大小写是不一样的，如：

START

start

是二个不同的标号。

二、指令助记符

指 PIC 的指令或伪指令，宏定义符等。具体参阅有关各章节和资料。

三、操作数

操作数可以是常数，符号或表达式。两个操作数之间必须由逗号(，)分开。

（1）符号——各种定义的符号、宏定义等。

例：MOVWF F10 ;F10 为操作数，是定义的代表寄存器 10 的符号。

（2）常数——在 MPASM 中，常数可以是如下：

进制	书写格式	例子
十进制	D'<数字>'	D'255'
十六进制	H'<16 进制数字>' 或 0x<16 进制数字>	H'A8' 0xA8
二进制	B'<二进制数字>'	B'00111001'
八进制	O'<八进制数字>'	O'777'
字符 ASCⅡ码	'< 字 符 >'	'C'

注：MPASM 默认进制为 16 进制。

表 4.1

（3）表达式——由常数、符号和各种算术运算符号按一定顺序组成。

MPASM 中的算术符号如表 4.2 所示。

运 算 符		例 子
(	左括号	1+（d*4）
)	右括号	同上

!	非	IF !(a-b)
+	加	a+b
-	减	a-b
*	乘	a*b
/	除	a/b
%	取模	a%2
<<	左移	<<a
>>	右移	>>a
>	大于	IF a>b
<	小于	IF a<b
<=	小于或等于	IF a<=b
==	等于	IF a==b
!=	不等于	IF a!=b
&	与	a & b
^	异或	a ^ b
|	或	a | b
~	取反	
& &	逻辑与	IF (a=2) && (b=3)
| |	逻辑或	IF (a=2) | | (b=3)
=	等于	a=b
+=	加，然后等于	a+=1
-=	减，然后等于	a-=1
*=	乘，然后等于	a*=5
/=	除，然后等于	a/=5
<<=	左移，然后等于	a<<=5
>>=	右移，然后等于	a>>=5
&=	与，然后等于	a&=5
|=	或，然后等于	a |=5
^=	异或，然后等于	a^=5
￥	返回当前 PC 值	GOTO ￥+3

表 4.2

四、注释

以分号（;）起始，用户可以注释程序。

```
CLRF    F10      ; 清 F10 寄存器
```

§4.3 伪指令

所谓伪指令，是一些用来控制汇编器的命令。它们可放在源程序（.ASM）中，但不是被翻译成可执行的机器代码，而是用来控制汇编器的输入/输出以及数据的定位等。

在 MPASM 中，有四类伪指令：

1、数据伪指令：用于控制程序存储器的定位，定义数据的名称等。

2、列表伪指令：用于控制 MPASM 产生的列表文件（.LST）的格式等。

3、控制伪指令：用于控制汇编的路径，如条件汇编等。

4、宏汇编指令：用于控制宏定义体中的运行和数据定位。

一、数据伪指令

1. DATA——定义程序存储器的值。

格式：〔<标号>〕 DATA <操作数>，〔<操作数>…〕

例：DATA 1，2+AB，“Test”

2. DEFINE——定义字符串变量。

格式：#DEFINE<变量名>〔<字符串>〕

例：#DEFINE　Length　　20

#DEFINE　control　　0x19，7

#DEFINE　position　(X，Y，Z)　(Y-(2×Z+X))

…

…

test_Lable　DATA　　position(1，Length，52)

bsf　　control　　；置 0X19 寄存器的 bit7

3. SET——对标号赋值。

格式：　<标号>　SET　<表达式>

例：　width　SET　9

area　　SET　0x16

width　SET　area+8

用 SET 可对标号任意重新赋值，见上例 3。这和下面的另一条标号赋不同。

4. EQU——对标号赋值。

格式：<标号>　EQU　<表达式>

例：　lable　EQU　0x16

标号一旦由 EQU 赋值后，其值便不能再重新定义，参考上面 SET 命令。

5. RES——保留某段程序存贮区。

格式：　RES　<单元个数>

例：　RES　　10

保留 10 个空白字节。

6. INCLUDE——调入外部文件，通常是定义文件，对一些标号和变量进行定义。

MPASM 提供一个名为 PICREG.EQU 的定义文件，读文件中定义了 PIC 寄存器的地址，复位向量及状态位址。

格式：　INCLUDE　“文件名”

例：　INCLUDE　“picreg.egu”

7. Radix——进制定义指令。

格式：　RADIX　<进制表达式>

例：　RADIX　dec　　；十进制

RADIX　Hex　　；十六进制

RADIX　oct　　；八进制

二、列表伪指令

1. LIST——列表选择指令，设置各种汇编参数。

格式：　LIST〔<选择项>…<选择项>〕

例：　LIST F=INHX8M，　R=DEC，　P=16C84

以下是 LIST 选项表：

选　项	默 认 值	作　　用
C=nnn	80	行宽

N=nnn	59	每页的行数
T=ON/OFF	OFF	ON 截去超长行的超出部分
P=<type>	无	PIC12C/16C/17C
R=<radix>	hex	常数进制选择:hex，dec，oct
F=<format>	INHX8M	烧写文件格式：INHX16，INHX32 和 INHX8M

表 4.3

2. PAGE——分页命令。
格式： PAGE
在列表文件中（.lst）中产生分页效果，即下面的文件输出将从新页面开始。

3. TITLE——程序标头命令。
格式： TITLE 'program header'
例： TITLE 'This is for PIC12C50X demo'
标头最长不超过 60 个字符。TITLE 令会造成分页，即标头总是在一页的第一行上。

三、控制伪指令

1. ORG——定义程序存放起始地址。
格式：<label> ORG <address expression>
例： ORG 0h ; 起始程序存放地址
START: MOVWF OSCCAL
…
…
若 ORG 不带地址参数，则默认为 0。若 ORG 带标号，则地址参数也赋值给该标号。

2. END——程序结束命令。
格式： END
例： END
这条指令告诉 MPASM 这是源程序（.ASM）的结束行，后面若还有语句将被视为无效。

3. IF——条件汇编命令。
格式： IF <条件表达式>
<源程序行>
<ELSE>
<源程序行>
ENDIF
例： IF VER==100
MOVLW 5
WOVWF F11
ELSE
MOVLW 6
MOVWF F11
ENDIF
…
…

如果条件表达式为真，MPASM 将汇编 IF 和 ELSE 之间的语句，反之汇编 ELSE 和 ENDIF 之间的语句。ELSE 可以缺省，这样条件为真则汇编，反之不汇编。

4. WHILE——条件循环命令。

格式： WHILE <条件表达式>

…

…

ENSW

例：
```
VARIABLE  i
WHILE     i<count
MOVLW     i
i=i+1
ENDW
```

注：VARIABLE 也是一条定义变量的伪指令，和 EQU 及 SET 不同的是它不要求变量在定义时必须赋值给初值，如上例中的变量 i。关于这条伪指令不再赘述。

四、宏定义伪指令

1. MACRO——宏定义命令。宏是一段指令，可以插在源程序中。宏必须事先定义好，宏之间可以互相调用，也可以自己递归调用。宏本身不会产生代码，只是在调用它时把宏体插入源程序，这点和子程序调用有本质不同，即宏并不会节省程序空间，它主要的好处是令程序书写简洁明了。

格式： <标号> MACRO〔<参数 1>…<参数 N>〕

(宏体)

ENDM

例：
```
GET   MACRO   X，Y，Z
      MOVWF   X
      Y
    Z MOVLW   10
      GOTO    Z
      ENDM
```

宏调用可以下为：

…

GET F0，(INCF F17，W)， ENTRY

…

则汇编后这句宏调用产生的源代码为：

```
      GET     F0，(INF17，W)，  ENTRY
+     MOVWF   F0
+     INCF    F17，W
+ENTRY MOVLW  10
+     GOTO    ENTRY
```

前面带+号表示是宏体中定义的程序。

§4.4 错误/警告信息

MPASM 汇编一个源程序后，可以产生一个.ERR 文件，该文件用来存放汇编后可能产生的错误或警告信息。必须强调的是错误信息（Error）是指出源程序中出现“致命”（fatal）的错误，用户必须修改直至汇编后 Errors= 0。而警告信息（Warnings）是指出源程序中可能有问题的地方，但并不一定是“致命”错误，只是提醒用户去注意这些被警告的地方。如果用户可以确认无误，便可以不理会产生的 Warnings。

一、错误信息

1.Address exceeds maximum limit avaiable

程序存储器地址溢出（超出）有效范围。

2.Attempt to redefine reserved word

MPASM 中的保留字如“END”、“ERROR”、“HIGH”、“LOW”和“PAGE”被重定义，用户必须避免再将其用做标号或变量。

3.Branch or jump out of range

程序跳转指令如“GOTO”、“CALL”等超出规定的范围。

4.Couldn't open…

TMPASM 不能打开“.OBJ”、“.map”、“.Hex”、“.Err”、“.Lst”或“.ref”文件。一般是电脑已没有足够的磁盘空间。

5.Couldn't open source file…

汇编的源程序文件不存在。

6.Duplicate lable or redefininy symbol that cannot be redefined

标号或变量名重复定义。

7.Error in parameter

参数错误。

8.Expected…

源程序行有错。

9.File not found

指定的文件找不到。

10.Illegal argument

非法参数。

11.Illegal condition

IF 语句中的条件符号出错。

12.Illegal condition，EOF encountered before END or conditional end directive

IF、WHILE 或 MACRO 语句中缺少相应的 ENDIF、ENDW 和 ENDM。

13.Illegal conditional compile

IF/ELSE/ENDIF 结构书写有错。

14.Illegal character…in label…

在标号字符中出现非法字符。合法的字符是“-”、“ •”、“A”～“Z”、“a”～“z”、“0”～“9”。

15.Illegal digit

非法数字。如在十进制数中出现十六进制符等。

16.Illegal opcode

非法操作数。

17.Include file not found

Include 指令中的文件找不到。

18.Include files nested too cleep

Include 文件嵌套太多。Include 文件嵌套最多的为 5 重。

19.Macro name missing

缺少宏定义名称。

20.Marco nested too deep

宏体嵌套太多。宏体中最多可嵌套 8 重。

21.Missing arguments

缺少参数，如指令中缺少操作数等。

22.Missing terminator

缺少配对符，如各种括号“)”、“〕”、“}”或“ •”、空格等。

23.Nested forward reference not allowed.