

单片机硬件、软件及其应用讲座

第一讲 单片机的基本概念

编者按 为了满足广大读者业余自学单片机技术的要求,本刊从今年开始,开设单片机专栏,并邀请河北省邮电学校的李学海老师主讲单片机知识讲座。讲座以美国 Microchip 公司的 PIC16C873 为例,讲解单片机的硬件、软件及其应用,力争使读者通过本讲座的学习,能够基本掌握单片机的使用,并能利用单片机进行简单的设计。在讲座进行过程中,读者有什么问题或要求,欢迎随时提出。

近几年,国际市场上的单片机性能不断增强,价格却日益下降。随着我国对外开放的力度不断加大,世界上一些著名的微电子公司都在积极开拓我国市场,这使得国内上市的单片机品种型号越来越多,价格也越来越低廉。这给电子爱好者玩单片机提供了丰富廉价的物质基础,因此,有越来越多的电子爱好者对单片机产生浓厚的兴趣。

单片机与常用的 TTL、CMOS 数字集成电路相比掌握起来不太容易,问题在于单片机具有智能化功能,不光要学习其硬件还要学习其软件,而软件设计需有一定的创造性。这虽然给学习它的人带来一定难度,但这也正是它的迷人之处。初学者到底能否在没有专业基础的情况下,通过自学在短暂的时间内掌握单片机技术,事实表明是做得到的!如果再经过反复实践将自己培养成单片机开发应用工程师也是完全可能的!

在当今的生活环境和工作环境中,有越来越多的单片机在为我们服务。可我们却意识不到这些“小精灵”的存在。譬如,当我们用遥控器操纵电视机或 VCD 享受其多彩的画面时,我们并没有意识到这是单片机在接收我们的遥控命令,单片机在 BP 机和大哥大中亦发挥着极其重要的作用;就连曾经一度令许多青少年着迷的电子宠物,也是单片机在大显神威。那么,为什么我们当中的许多人竟然对它的存在熟视无睹呢?原因就是我们对这些“忠心耿耿”地为我们服务的“小精灵”了解甚少。

家用电器和办公设备的智能化、遥控化、模糊控制化已成为世界潮流,而这些高性能无一不是靠单片机来实现的。如果我们不具备单片机方面的知识,对这些电器设备的日常保养和故障维修会形成很大的障碍。

从前的电子爱好者用简陋的器件制作出只能用耳机收听的矿石收音机,后来的电子爱好者用半导体分立元件制作晶

体管收音机,而今天的电子爱好者不仅可以用芯片制作集成电路收音机,还可以用单片机制作许多带智能和遥控等功能的小电器。对一个电子制作的爱好者来说,一旦掌握了单片机技术,就可进入一个神奇而又广阔的新天地。

一只装有专用软件的单片机,配上一只液晶显示屏和几只小按钮,再装入一只小塑料壳,便可构成一只妙趣无穷的电子宠物。其造价只不过 10 余元,但市场售价竟可高达 120 余元。理由何在?原因在于它是高科技产品,技术含量高,其中的软件凝聚着开发者的聪明和智慧。有关专家指出,到 2000 年,一般美国家用系统中应用单片机的数量将增加到 226 个,自动化办公室内将有 42 个,典型的汽车电子系统中将装有 35 个。

当前,单片机的产量正以每年 27% 的速度递增。据统计 1995 年单片机产量已达到 16 亿片,预计到 2000 年将达到 28 亿片。由此可见,单片机技术无疑将是 90 年代乃至 21 世纪最为活跃的新一代电子应用技术。随着微控制技术(以软件代替硬件的高性能控制技术)的日臻完善和发展,单片机的应用必将导致传统控制技术发生巨大变革。换言之,单片机的应用是对传统控制技术的一场革命。因此,学习单片机的原理,掌握单片机的应用技术,具有划时代的意义。

单片机究竟是什么

单片机,亦称单片微电脑或单片微型计算机。它是把中央处理器(CPU)、随机存取存储器(RAM)、只读存储器(ROM)、输入/输出端口(I/O)等主要计算机功能部件都集成在一块集成电路芯片上的微型计算机。这种微型计算机因其制作在一块芯片上而被称为单片机。单片机是大规模集成电路技术发展的产物。单片机具有性能高、速度快、体积小、价格低、稳定可靠、应用广泛、通用性强等突出优点。

单片机的设计目标主要是增强“控制”能力,满足实时控制(就是快速反应)方面的需要。因此,它在硬件结构、指令系统、I/O 端口、功率消耗及可靠性等方面均有其独特之处,其最显著的特点之一就是具有非常有效的控制功能。因此,单片机又常常被人称为微控制器(MCU 或 μC)。

尽管单片机主要是为控制目的而设计的,它仍然具备通用 PC 机的全部特征。既然单片机是一部完整的 PC 机,那么单片机的功能部件和工作原理与 PC 机也是基本相同的。因此,我们可以先通过参照 PC 机的基本组成和工作原理来逐步接近单片机。

图 1 示出了一台普通 PC 机的基本结构。由图 1 可知,一台 PC 机是由运算器、控制器、存储器、输入设备和输出设备五个部分组成的。虽然微型计算机技术得到了最充分的发展,但是 PC 机在体系结构上仍属于经典的计算机结构。这种结构是由计算机的开拓者——数学家约翰·冯·诺依曼最先提出的,所以,就称之为冯·诺依曼计算机体系结构。至今为止,计算机的发展已经经历了四代,尚未冲出诺依曼体系。当前,市场上常见的大多数型号的单片机也还遵循着诺依曼体系的设计思路。

下面让我们来分析 PC 机各部分的作用和 PC 机的工作原理。如果要使 PC 机按照人们的需要解决某个具体问题,并不是把这个问题直接让 PC 机去解决,而是要用 PC 机可以“理解”的语言,编写出一系列解决这个问题的步骤(即程序)并输入到计算机中,命令它按照这些步骤顺序执行,从而使问题得以解决。编写解决问题的步骤,就是人们常说的编写程序(也叫程序设计或软件开发)。计算机是严格按照程序对各种数据或者输入信息进行自动加工处理的,因此必须预先把程序以及数据用“输入设备”送入 PC 机内部的“存储器”中,处理完后还要把结果用“输出设备”输送出来。“运算器”完成程序中规定的各种算术和逻辑运算操作。为了使 PC 机各部件有条不紊地工作,由“控制器”理解程序的意图,并指挥各部件协调完成规定的任务。通常,在 PC 机中,把控制器和运算器制作在一块集成电路内,并称之为中央处理器或中央处理单元(CPU)。

单片机的一般结构可以用图 2 所示的方块图描述。图 2 与图 1 的对应关系是:CPU 包含控制器和运算器;ROM 和 RAM 对应着存储器。ROM 存放程序, RAM 存放数据;I/O 则对应着输入设备和输出设备。用总线实现 CPU、RAM、ROM 和 I/O 各模块之间的信息传递。其实,具体到某一种型号的单片机,其芯片内部集成的程序存储器 ROM 和数据存储器 RAM 可大可小,输入和输出端口 I/O 可多可少,但 CPU 只有一个。

单片机有哪些特点

单片机除了具备体积小、价格低、性能强、速度快、用途广、灵活性强、可靠性高等优点之外,它与通用微型计算机相比,在硬件结构和指令设置上还具有其以下独特之处。

(1) 存储器 ROM 和 RAM 是严格分工的。ROM 用作程序存

储器,只存放程序、常数和数据表格,而 RAM 用作数据存储器,存放临时数据和变量。这样的设计方案使单片机更适用于实时控制(或称为现场控制或者过程控制)系统。配置较大的程序存储空间 ROM,将已调试好的程序固化(或称烧录或者烧写)其中,不仅掉电时程序不丢失,还避免程序被破坏,从而确保了程序的安全性。实时控制仅需容量较小的 RAM,用于存放少量随机数据,这样有利于提高单片机的操作速度。

(2) 采用面向控制的指令系统。在实时控制方面,尤其是在“位”操作方面单片机有着不俗的表现。

(3) 输入/输出(I/O)端口引脚通常设计有多种功能。在设计时,究竟使用多功能引脚的哪一种功能,则可以由用户来随意确定。

(4) 品种规格的系列化。属于同一个产品系列的、不同型号的单片机,通常具有相同的内核、相同或者兼容的指令系统。其主要的差别仅是在片内配置了一些不同种类或不同数量的功能部件,以适用不同的被控对象。

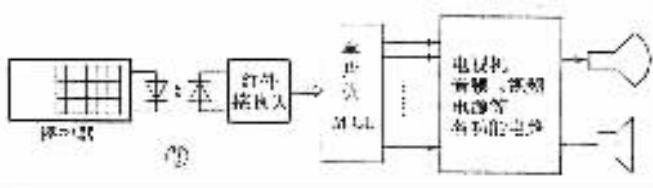
(5) 单片机的硬件功能具有广泛的通用性。同一种单片机可以用在不同的控制系统中,只是其中所配置的软件不同而已。换言之,给单片机固化上不同的软件,便可形成用途不同的专用智能芯片。有时将这种芯片称为“固件”(Firmware)。

单片机的应用

单片机可应用于电话机、寻呼机、手机、对讲机等电信设备,电视机、录像机、摄像机、VCD 机、洗衣机等家用电器,电子玩具,计算机外围设备,办公自动化设备,工业控制设备、仪器仪表,军用设备等等。有人这样说:“凡是能想到的地方,单片机都可以用得上”,这并不夸张。全世界单片机的年产量数以亿计,应用范围之广,花样之多,一时难以详述。

单片机应用的意义不仅仅限于它的广阔范围以及所带来的经济效益,更重要的还在于从根本上改变了传统的控制系统设计思想和设计方法。从前,必须由模拟电路或数字电路实现的大部分控制功能,现在已能使用单片机通过软件方法实现了。这种以软件取代硬件并能提高系统性能的控制技术称之为微控制技术。微控制技术标志着一种全新概念,随着单片机应用的推广普及,微控制技术必将不断发展和日趋完善,而单片机的应用则必将更加深入、更加广泛。

下面结合我们生活中最常用的一种电器——遥控彩电,来简单介绍单片机的一个应用实例。为了突出单片机的作用,将一台遥控彩电的电路概括成如图 3 所示的方块图。当人们按动遥控器的按键时,经过编码和调制的红外遥控信号由发光二极



管发送到接收二极管,并由红外接收头解调出一串二进制码,然后由单片机经输入端口接收、译码并理解遥控命令,再经相应的输出端口输出驱动信号,最终实现调节音量、亮度、对比度、色度以及控制选台、静音、开关机等等一系列智能性很强的操作。

✱

单片机硬件、软件及其应用讲座(2)

· 李学海 ·

第二讲 PIC 系列单片机的特点

随着大规模集成电路(LSI)制造技术的飞速发展,电脑正朝着两个明显的方向(即两大分支)发展:一是微型计算机系统的性能不断提高,以满足高速度大容量的“数据处理”;二是单片微机的功能日益完善,以满足诸多领域各种错综复杂的“现场控制”。

在1975年德克萨斯仪器公司发明的世界上第一个4位单片机TMS-1000诞生后,一些大型微电子公司竞相研制开发了各种单片机系列产品。从字长方面划分,单片机有4位、8位、16位、32位四大类,其中前三类占据了单片机市场的主要份额,在这三类单片机当中,8位机又一直为主流产品。

较具有代表性的4位单片机有美国德克萨斯仪器公司的TMS-1000,日本电气公司(NEC)的 μ PD75 \times 系列,美国国家半导体公司(NS)的COP400系列,美国洛克威尔公司(ROCKWELL)的PPS/1系列,日本松下公司的MN1400系列,日本富士通公司的MB88系列以及日本夏普公司的SM \times 系列等等。

较具有代表性的8位单片机有美国微芯片公司的PIC16C \times 系列、PIC17C \times 系列、PIC1400系列,美国英特尔公司的MCS-48和MCS-51系列,美国摩托罗拉公司的MC68HC05系列和MC68HC11系列,美国齐洛格公司的Z8系列,日本电气公司的 μ PD78 \times 系列,美国莫斯特克公司和仙童公司合作生产的F8(3870)系列等。

较具有代表性的16位单片机有美国莫斯特克公司的MC68200,美国英特尔公司的MCS-96系列,日本电气公司的 μ PC14040系列,美国国家半导体公司(NS)的783 \times 系列等。

目前,单片机正朝着片内存储器RAM和ROM容量大、I/O端口功能多、电源电压范围宽、功率消耗低、操作速度快的方向发展。

在上述各种单片机中,本讲座为什么选用微芯片公司的PIC系列中的PIC16F873单片机作为样板介绍呢?

PIC系列单片机的硬件系统设计简洁,指令系统设计精炼。在所有的单片机品种当中,它是最容易学习、最容易应用的单片机品种之一。对于单片机的初学者来说,若选择PIC单片机作为攻入单片机王国的“突破口”,将是一条最轻松的捷径,定会取得事半功倍的功效。目前已有好几家著名半导体公司仿照PIC系列单片机,开发出与之引脚兼容的系列单片机,比如美国SCENIX公司的SX系列、台湾EMC公司的EM78P系列、台湾MDT公司的MDT系列等。

PIC系列单片机具有以下特点(1)采用哈佛结构。在国内最常见的单片机中,PIC系列单片机是唯一一种在芯片内部采

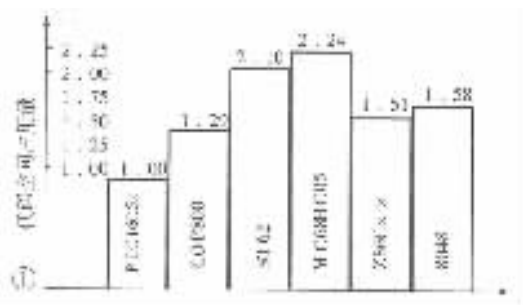
用哈佛结构的机型。这里所说的“哈佛结构”就是在芯片内部将数据总线和指令总线分离,并且采用不同的宽度。这样做的好处是,便于实现“流水作业”,也就是在执行一条指令的同时对下一条指令进行取指操作,而在一般的单片机中,指令总线和数据总线是共用的。

(2)指令的“单字节化”。因为数据总线和指令总线是分离的,并且采用了不同的宽度,所以程序存储器ROM和数据存储器RAM的寻址空间是互相独立的,而且两种存储器宽度也不同。这样设计不仅可以确保数据的安全性,还能提高运行速度和实现全部指令的“单字节化”。在此所说的“字节”,特指PIC单片机的指令字节,而不是常说的8比特字节。例如,PIC12C50 \times /PIC16C5 \times 系列单片机的指令字节为12比特;PIC16C6 \times /PIC16C7 \times /PIC16C8 \times 系列的指令字节为14比特;PIC17C \times 系列的指令字节为16比特。它们的数据存储器全为8位宽。而MCS-51系列单片机的ROM和RAM宽度都是8位,指令长度从1个字节(8位)到3个字节长短不一。

(3)精简指令集(RISC)技术。PIC系列单片机的指令系统只有35条指令。这给指令的学习、记忆、理解带来很大的好处,也给程序的编写、阅读、调试、修改、交流带来极大的便利,真可谓“易学好用”。而MCS-51单片机的指令系统共有111条指令,MC68HC05单片机的指令系统共有89条指令。PIC系列单片机不仅全部指令均为单字节指令,而且绝大多数指令为单周期指令,以利于提高执行速度。

(4)寻址方式简单。寻址方式就是寻找操作数的方法。PIC系列单片机只有4种寻址方式(即寄存器间接寻址、立即数寻址、直接寻址和位寻址,以后将作详细解释),容易掌握,而MCS-51单片机则有7种寻址方式,68HC05单片机有6种。

(5)代码压缩率高。1K字节的存储器空间,对于像MCS-51这样的单片机,大约只能存放600条指令,而对于

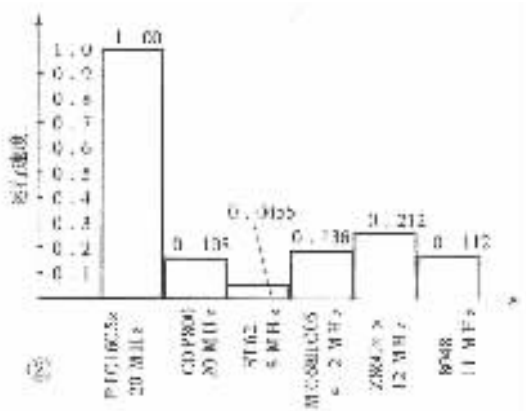


PIC系列单片机则能够存放多达1024条指令条数。从图1中可以看出,与几种典型的单片机相比,PIC16C5 \times 是一种最节省程序存储器空间的单片机。也就是说,完成相同功能的一段



程序所占用的空间,MC68HC05 是 PIC16C5 的 2.24 倍。

(6) 运行速度快。由于采用了哈佛总线结构,又由于指令的读取和执行采用了流水作业方式,PIC 系列单片机的运行速度大大提高。从图 2 中可以看出,PIC 系列单片机的运行速度远高于其它相同档次的单片机。在所有 8 位机中,PIC17C 是目前世界上速度最快的品种之一。



(7) 功耗低。PIC 系列单片机的功率消耗极低,有些型号的单片机在 4MHz 时钟下工作时耗电不超过 2mA,在睡眠模式下低到 1 μ A 以下。

(8) 驱动能力强。I/O 端口驱动负载的能力较强,每个 I/O 引脚吸入和输出电流的最大值可分别达到 25mA 和 20mA,能够直接驱动发光二极管、光电耦合器或者微型继电器等。

(9) 具备 I²C 和 SPI 串行总线接口。PIC 系列单片机的一些型号具备 I²C 和 SPI 串行总线接口。I²C 和 SPI 分别是由 PHILIPS 和 MOTOROLA 公司发明的在芯片之间实现同步串行数据传输的两种串行总线技术。利用单片机串行总线接口可以方便灵活地扩展一些必要的外围器件。串行接口和串行总线的设置,不仅大大地简化了单片机应用系统的结构,而且还极易形成产品电路的模块化结构。目前,松下、日立、索尼、夏普、长虹等公司都在其大屏幕彩电等产品中引入了 I²C 技术。

(10) 寻址空间设计简洁。PIC 系列单片机的程序、堆栈、数据三者各自采用互相独立的寻址(或地址编码)空间,而且前两者的地址安排不需要用户操心,这会受到初学者的欢迎。而 MC68HC05 和 MC68HC11 单片机的寻址空间只有一个,编程时需要用户对程序区、堆栈区、数据区和 I/O 端口所占用的地址空间作精心安排,这样会给高手的设计上带来灵活性,但是也会给初学者带来一些麻烦。

(11) 外围电路简洁。PIC 系列单片机片内集成了上电复位电路、I/O 引脚上拉电路、看门狗定时器等,可以最大程度地减少或免用外围器件,以便实现“纯单片”应用。这样,不仅便于开发,而且还可节省用户的电路板空间和制造成本。

(12) 开发方便。通常,业余条件下学习和应用单片机,最大的障碍是实验开发设备昂贵,使许多初学者望而却步。微芯片公司及其国内多家代理商,为用户的应用开发提供了丰富多彩的硬件和软件支持。有各种档次的烧录器(或称编程器)和硬件仿真器出售,其售价大约从 500 元到 2000 元不等。此外,微芯片公司还研制了多种版本的软件仿真器和软件综合开发环境(MPLAB-IDE),为爱好者学习与实践、应用与开发的实际操作提供了极大的方便。对于 PIC 系列中任一款单片机的开

发,都可以借助于一套免费的软件综合开发环境实现程序编写和模拟仿真,再用任何一种廉价的烧录器完成程序烧写,便形成一套经济实用的开发系统。它特别适合那些不想过多投资购置昂贵开发工具的初学者和业余爱好者。借助于这套廉价的开发系统,用户可以完成一些小型电子产品的研制开发。由此可见,对初级水平的自学者来说,PIC 单片机是一种最为适合、最容易接近的单片机。

(13) C 语言编程。对于掌握了 C 语言的用户,微芯片公司还为其提供了“C 语言编译程序”,这样的用户如果使用 C 语言这种高级语言进行程序设计的话,还可以大大提高工作效率。

(14) 品种丰富。PIC 系列单片机目前已形成三个层次、50 多个型号。片内功能从简单到复杂,封装形式从 8 脚到 68 脚,可以满足各种不同的应用需求。用户总能在其中找到一款适合自己开发目标的单片机。在封装形式多样化方面,不像 MCS-51 系列单片机那样,大都采用 40 脚封装,应用灵活性受到极大的限制。此外,微芯片公司最先开发出世界上第一个最小的 8 脚封装的单片机。

(15) 规格齐全。微芯片公司对其单片机的某一种型号又可提供多种封装工艺的产品:带窗口的 EPROM 型芯片,适合程序反复修改的开发阶段;一次编程(OTP)的 EPROM 芯片,适合于小批量试生产和快速上市的需要;ROM 掩模型芯片,适合大企业大批量定型产品的规模化生产;个别型号具有 EEPROM 或 Flash 程序存储器,特别适合初学者“在线”反复擦写、练习编程。

(16) 程序保密性强。目前尚无办法对 PIC 系列单片机的程序直接进行解密拷贝,可以最大限度地保护用户的程序版权。

在 PIC 系列单片机中,PIC16F873 是微芯片公司于 1998 年底推出的一款特色鲜明的新产品,它除了具有上述特点之外,还有一个最重要的特点,就是它可以实现在线调试和在线编程。这是 MCS-51 和 MC68 系列单片机所不具备的,但却正是广大单片机初学者最需要的。

微芯片公司还专为此款单片机开发了一套小巧廉价的仿真工具套件,以下简称“仿真板”或 MPLAB-ICD。在该仿真板上既可以实现硬件仿真,又可以实现程序烧录,还保留了一块用户可以随意焊接一些元器件的布满焊孔的电路板空间。由此可见,开发 PIC16F873 比前面介绍的开发其它 PIC 型号单片机的手段就更加简便易行。当然,如果只是为了学习,也可以一分钱不花,只用免费的软件综合开发环境(MPLAB-IDE),对 PIC16F873 进行软件模拟仿真来调试程序。该软件约有 24M 字节,并可以从因特网上下载,该公司还在中国开设了中文网站,其网址为 <http://www.microchip.com.cn/>。

其实在 PIC 单片机的家族中,PIC16F873 还有另外三个“近亲弟兄”,它们分别是 PIC16F874、PIC16F876、PIC16F877。微芯片公司将这四兄弟统称为 PIC16F87 \times 。它们之间的差别很小,学会其中一款就基本上等于认识了这四兄弟,所以我们先在其中挑一款相对简单的型号向大家介绍。总体上讲,论本领、威力或者性能的话,在众多的 PIC 单片机家族成员中,PIC16F87 \times 占据着中上等水平。有的初学者可能要问,既然 PIC 系列中还有更简单易学的品种,为什么先给大家引见 PIC16F873 呢?理由就是该型号具备让人接近的良好途径——在线调试和在线编程。借助于这项独特的性能,我们可以边学边练,学用结合。

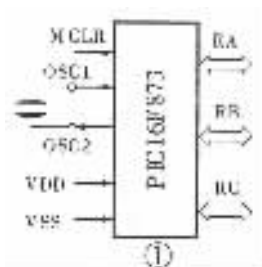


单片机硬件、软件及其应用讲座(3)

· 李学海 ·

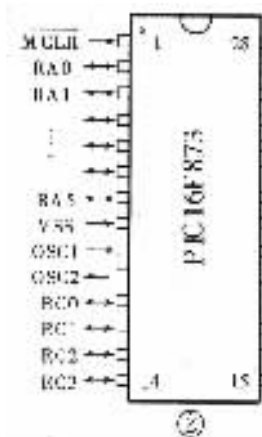
第三讲 PIC168F873 硬件系统 (上)

PIC16F873 型单片机是 PIC 中级单片机中很有特色的一款,其指令字节为 14 比特。它具有 PIC 系列单片机的全部优点,而且片内还带有 128×8 的 EEPROM(也叫 E²PROM)数据存储器,其程序存储器与众不同,采用快闪存储器。快闪存储器可以实现在电路板上快速擦除和写入,最适合于制作仿真板。借助于专为 PIC16F873 制作的仿真板,读者在学习程序编写和调试的过程中,可以方便地烧写程序和修改程序。读者在掌握了 PIC16F873 之后,如果想进一步学习 PIC 系列其它型号的单片机,将会收到触类旁通的功效。



PIC16F873 引脚功能

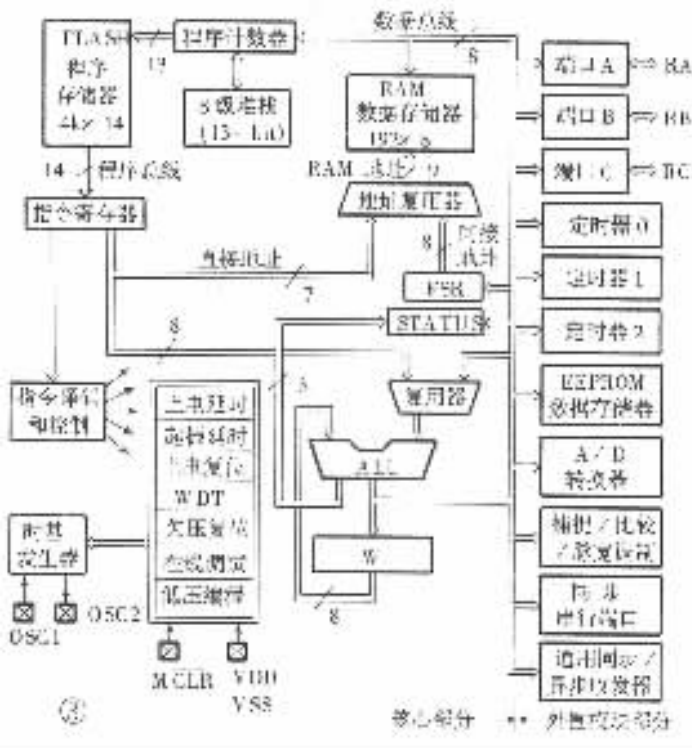
一款单片机无论是软件还是硬件功能多么的强大,当它嵌入到被控系统中,都是通过引脚上的信息吞吐来实现控制功能和体现自身价值的。因此,需要我们对单片机的每一根引脚所能发挥的作用有一个全面地了解。



PIC16F873 采用双列直插和表面贴装两种 28 引脚封装形式,其逻辑符号和引脚排列顺序如图 1 和图 2 所示,各引脚功能的说明见附表。PIC16F873 的许多引脚不仅具有第 1 功能,而且还有第 2 功能,甚至部分引脚还有第 3 功能。为了减少复杂性,便于初学者学习和掌握,在图 1、图 2 和附表中,对于

各条多功能引脚暂时只给出了第 1 功能(即主要功能)。

虽然 PIC16F873 有 28 根引脚,各引脚的功能不仅有多有少而且也千差万别,但是如果按引脚功能的相近程度进行归类的话,不妨可以将所有引脚划分为 4 大类:控制类(包含/MCLR)、时钟类(包含 OSC1 和 OSC2)、电源类(包含 VDD 和 VSS)、端口类(包含 RA、RB 和 RC)。这样一来,就使得图 1 所示的逻辑符号看上去简洁明了。



PIC16F873 内部结构框图

附表

引脚名称	引脚号	引脚类型	功能说明
OSC1	9	I	时钟振荡器输入端,也是晶振连接端。
OSC2	10	O	时钟振荡器输出端,也是晶振连接端。
/MCLR	1	I/P	人工复位输入端(低电平有效)。
RA0~RA5	2~7	I/O	主要功能 RA 端口是一个输入/输出可编程的双向端口,此外还有第 2、第 3 功能。
RB0~RB7	21~28	I/O	主要功能 RB 端口是一个输入/输出可编程的双向端口,作输入时内部有编程的弱上拉电路,此外还有第 2、第 3 功能。
RC0~RC7	11~18	I/O	主要功能 RC 端口是一个输入/输出可编程的双向端口,此外还有第 2、第 3 功能。
VSS	8,19	P	接地端。
VDD	20	P	正电源端。

说明 1. 端口类型中的 I、O、P 分别表示输入、输出和电源;
2. 各引脚的第 2 和第 3 功能暂时不列出,以后用到时再作详解。

PIC16F873 内部结构的功能框图如图 3 所示。为了便于讲解和理解,不妨把整个框图按重要程度划分为核心模块和外围模块两大部分。对于 PIC 系列中的任何一款单片机来说,其核心部分是唯一的也是必不可少的,而外围模块的种类和数量,完全可以由厂家根据单片机的设计目标来确定。本讲介绍 PIC16F873 的核心部分所包含的部件以及各部件的功能。

核心部分包含程序存储器、程序计数器、堆栈、指令寄存器、指令译码和控制器、算术逻辑单元 ALU、工作寄存器 W、状态寄存器 STATUS、复用器、RAM 数据存储器、地址复用器、间接寻址寄存器 FSR、时基发生器、看门狗定时器 WDT、上电复位电路、上电延时电

路、起振延时电路、欠压复位电路、在线调试电路、低压编程电路、数据总线及程序总线等 22 部分。

程序存储器用来存放由用户预先编制好的程序和一些固定不变的数据,程序计数器用来产生并提供对程序存储器进行读出操作所需要的 13 比特地址码,初始状态为零,每执行一条指令,地址码自动加 1。堆栈用于保存程序断点地址。在程序执行过程中,有时需要调用“子程序”,在进入子程序之前,必须保存主程序断点处的地址,以便在子程序执行完后,再恢复断点地址,使主程序得以继续执行。

指令寄存器用来暂存从程序存储器中取出的指令,并将指令按不同的字段分解为操作码(表示计算机执行什么操作)和操作数(表示参加操作的数的本身或者被操作的数所在的地址)两部分,分别送到不同的目的地。

指令译码和控制器可将指令的操作码部分翻译成一系列的微细操作,并控制各功能电路协调运作。

算术逻辑单元 ALU 可实现算术运算和逻辑运算操作。工作寄存器 W 是一个很重要的工作寄存器,许多指令都把它作为操作过程的中转地,比如暂存准备参加运算的一个操作数(称为源操作数),或者暂存运算产生的结果(称为目标操作数)。换句话说,在运算之前,W 是源操作数的出发地,在运算之后,W 是目标操作数的目的地。PIC16F873 中的 W 相当于其它单片机中的“累加器 A”。

状态寄存器 STATUS 可及时反映运算结果的一些算术状态,比如是否产生进位、借位、全零等。该寄存器在其它单片机中又称为标志寄存器或条件码寄存器。

复用器选择和传递参加运算的另一个源操作数。该操作数既可以来源于 RAM 数据存储器,也可以来源于指令码中。

RAM 数据存储器用于存储 CPU 在执行程序过程中所产生的中间数据。普通的 RAM 存储器一般只能实现数据的读出操作和写入操作,而 PIC16F873 中的 RAM 存储器的每个存储单元功能都十分强大,除了具备普通存储器功能之外,还能实现移位、置位、清位、位测试等一系列(只有“寄存器”才能完成的)复杂操作。

地址复用器选择并传递访问(就是进行读取或者写入)数据存储器所需的地址,该地址既可以来源于“间接寻址寄存器 FSR”,也可以来源于指令码。来源于 FSR 的地址叫作间接地址,来源于指令码的地址叫作直接地址。

间接寻址寄存器 FSR 用于存储间接地址。时基发生器可产生芯片内部各功能电路工作所需的时钟脉冲信号。

看门狗定时器 WDT 是一个自带 RC 式振荡器时钟源的定时器,用来监视程序的运行状态。由于意外原因导致 CPU 跑到正常程序之外而出现“死机”时,WDT 将强行把 CPU 复位,使其返回到正常的程序中来。

上电复位电路在芯片加电后 VDD 上升到一定值(一般在 1.6V ~ 1.8V)时产生一个复位脉冲使单片机复位。上电延时电路提供一个固定的 72ms 的上电定时延迟,以使 VDD 有足够时间上升到对芯片合适的电压值。起振延时电路在上电延时之后,再提供 1024 个时钟周期(时钟周期即为时钟频率的倒数)的延迟,目的是让振荡电路有足够的时间产生稳定的时钟信号。欠压复位电路在电源电压 VDD 出现跌落并下降到 4V 以下时产生一个复位信号,使 CPU 进入并保持复位状态,直到 VDD 恢复到正常范围,之后再延迟 72ms,CPU 才从复位状态返回到运行状态。在线调试电路用于实现对焊接在电路板上的 PIC16F873 直接进行程序调试,当然还需要综合开发环境软件 MPLAB 和仿真板的支持。低压编程电路在对 PIC16F873 进行在线串行编程时,允许使用芯片工作电压 VDD 作为编程(即烧写)电压,而不需要加额外的高电压。

数据总线作为数据传输的专用通道,有 8 比特宽。它将各个外围模块以及核心部分的 PC、FSR、STATUS、W、ALU、RAM 等功能部件联系起来。程序总线有 14 比特宽,它作为提取程序指令的高速通道,快速及时地输送从程序存储器到指令寄存器的每一条指令。

PIC16F873 单片机的指令运行过程是这样的:从程序计数器指定的某一程序存储器单元中取出一条指令(指令码的长度为 14 比特),经程序总线送往指令寄存器。在指令寄存器中将指令码的操作码部分和操作数部分分解出来,操作码被送往指令译码和控制电路并被翻译成指挥核心部分中各部件协调工作的一系列控制信号,地址码经地址复用器传递,直接作为数据存储器的地址并选择其中某个单元,操作数经复用器传递到算术逻辑单元 ALU,在 ALU 中完成运算操作,并将运算结果的算术特征及时反映到状态寄存器 STATUS 中,比如运算是否产生进位,结果是否为零等。

有一点需要说明的是,按照单片机所实现的功能不同,其中的指令也有不同类型,因此它们的运行过程也不尽相同。✱

单片机硬件、软件及其应用讲座(4)

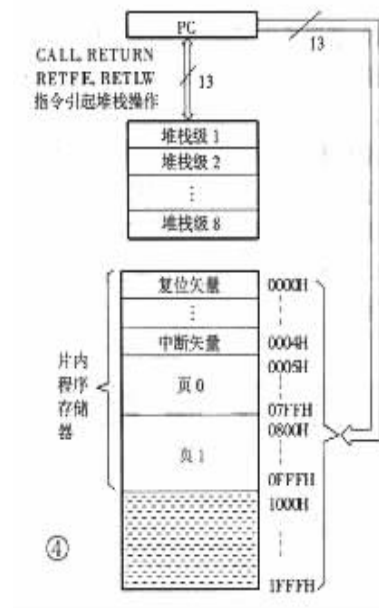
· 李学海 ·

第三讲 PIC16F873 硬件系统 (下)

程序存储器和堆栈

PIC16F873 具有一个 13 位宽的程序计数器 PC, 它所产生的 13 位地址最大可寻址的存储器空间为 $8K \times 14$, 地址编码的最大范围为 0000H ~ 1FFFH。但是,

PIC16F873 只配置了 $4K \times 14$ 的闪存程序存储器(它不仅可以在电路板上直接进行电写入和电擦除, 而且掉电后内容也不会丢失), 其地址范围仅占用了 0000H ~ 0FFFH, 如图 4 所示。整个程序存储器以 2K 为单位进行分页, 所以对于 PIC16F873 来说 4K 程序存储器共分作两页, 分别记为“页 0”和“页 1”。分页到底有何意义后面会讲到。



程序存储器中保留了两个特殊的单元, 不能挪作它用。一个是地址为 0000H 的单元专门存放“复位矢量”, 另一个地址为 0004H 的单元专门存放“中断矢量”。所谓复位矢量就是主程序的入口地址, 无论由于何种原因引起的单片机复位, 它都将从该地址入口从头执行主程序。所谓中断矢量就是中断服务子程序的入口地址, 无论由于何种原因引起的单片机中断, 它都将从该地址进入中断服务子程序。

PIC16F873 的堆栈具有 8×13 的独立空间, 不占用程序存储器和数据存储器区域, 也不需要进栈和出栈之类的堆栈操作专用指令。只有当执行“调用指令 CALL”或者 CPU 响应中断而发生程序跳转时, 才把当前程序计数器 PC 的值(即被中断的程序的断点地址)自动压入堆栈; 也只有当执行“返回指令 RETURN、RETFIE 或 RETLW”时, 才会从堆栈中弹出并恢复程序计数器 PC 原先的值。

堆栈的操作, 遵循一种“后进先出”的规则, 即最先进栈的数据最后出栈, 最后进栈的数据最先出栈。

普通的 RAM 存储器(即随机读取和写入存储器)一般只能实现数据的读出和写入操作。而 PIC16F873 中用于存储数据的 RAM 存储器功能十分强大, 除了具备普通存储器功能之外, 还能实现移位、置位、清位、位测试等一系列寄存器才能完成的复杂操作。

PIC16F873 的 RAM 数据存储器替代了其它单片机的通用寄存器和专用寄存器以及片内 RAM 的全部功能。其内容可读可写, 掉电后自动消失。为了与 PIC16F873 片内配置的另一种数据存储器 EEPROM 区分, 在此把它叫作 RAM 数据存储器。

PIC16F873 的数据存储器, 按功能分为“特殊功能寄存器”和“文件寄存器”两个区域, 两者按纵向排列。特殊功能寄存器占据了上半部的低地址部分, 而文件寄存器占据了下半部的高地址部分。其中有一些寄存器单元实际上是同一个寄存器单元, 却具备 4 个不同的地址。例如, 状态寄存器 STATUS 的 4 个地址是 03H、83H、103H 和 183H。还有一些寄存器单元是同一个寄存器单元, 具备 2 个不同的地址。例如, 选项寄存器 OPTION-REG 的两个地址是 81H 和 181H。又比如, 整个文件寄存器也是如此。

文件寄存器是用于通用目的, 即由用户自由安排和存放随机数据(单片机上电复位后其内容是不确定的), 因此又可以把文件寄存器称为“通用寄存器”。文件寄存器共有 192 个 8 比特宽的单元, PIC16F873 的文件寄存器扮演了其它单片机中的通用寄存器和片内 RAM 的双重角色。

特殊功能寄存器(FSR)作为专用寄存器, 其每个寄存器单元都有一个固定的用途, 所以又可以把特殊功能寄存器称为“专用寄存器”。由于特殊功能寄存器专门用于控制 CPU 内核的性能配置和各种外围功能模块的操作, 因此又可分成两类: 一类是与 CPU 内核相关的寄存器, 另一类是与外围模块相关的寄存器。以下仅介绍与 CPU 内核相关并且关系密切的几个寄存器, 即状态寄存器、实现间接寻址的寄存器(INDF 和 FSR)以及与程序计数器 PC 相关的寄存器(PCL 和 PCLATH), 而其余的寄存器则在介绍各种外围模块时予以讲解。

状态寄存器与通用寄存器不完全一样, 其中某些位(TO 和 PD)只能读不能写, 另一些位的状态会根据运算结果而随时变化。状态寄存器各位(见表 1)的含义如下: (1) C 为进位/借位标志位。执行加法指令时, $C = 1$

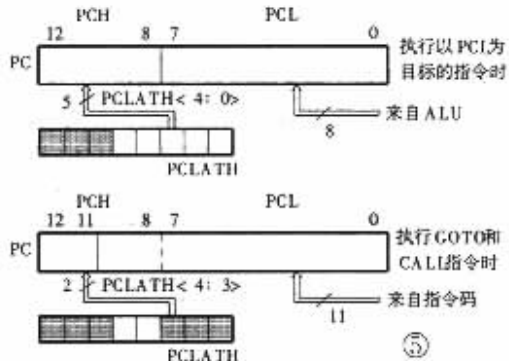
表 1

IRP	RP1	RP0	TO	PD	Z	DC	C
第 7 位	第 6 位	第 5 位	第 4 位	第 3 位	第 2 位	第 1 位	第 0 位

表示发生进位, $C=0$ 表示不发生进位。执行减法指令时, $C=1$ 表示不发生借位, $C=0$ 表示发生借位; (2) DC 为辅助进位/借位标志位, 执行加法指令时, $DC=1$ 表示低四位向高四位发生进位, $DC=0$ 表示低四位向高四位不发生进位; (3) Z 为零标志位。 $Z=1$ 表示算术或逻辑运算的结果为零, $Z=0$ 表示算术或逻辑运算的结果不为零; (4) PD 为掉电位。初始加电或 CLR-WDT(看门狗清零)指令执行后该位置 1, SLEEP(睡眠)指令执行后该位清零; (5) TO 为超时位。上电或 CLRWDT 及 SLEEP 指令执行后该位置 1, 若看门狗发生超时该位清零; (6) RPO 和 RPI 为数据存储器 RAM 体选位, 仅用于直接寻址; (7) IRP 也是数据存储器 RAM 体选位, 仅用于间接寻址。

位于 RAM 数据存储器最顶端、地址码最小的 INDF 寄存器, 其实是一个空寄存器, 它只有地址编码, 并不存在一个真正(或者物理上)的寄存器单元。INDF 寄存器与 FSR 寄存器配合, 可实现间接寻址。当寻址 INDF 时, 实际上是访问以 FSR 内容为地址的数据存储器 RAM 单元。在 PIC 系列单片机中所采用的这种独特而又巧妙的构想, 可以大大简化指令系统, 也就是使指令集得到很大程度地精简。

程序计数器 PC 是一个 13 位宽的专门提供程序存储器地址的寄存器。为了与其它 8 位宽的寄存器进行数据交换, 将它分成 PCL 和 PCH 两部分: 低 8 位 PCL 有自己的地址, 可读可写, 而高 5 位 PCH 却没有自己的地址, 不能直接写入, 只能用寄存器 PCLATH 装载的方式来进行间接写入。对于高 5 位



PCH 的装载又分两种情况(如图 5 所示)。一是当执行以 PCL 为目标(即目的地)的指令时, PC 的低 8 位来自算术逻辑单元 ALU, PC 的高 5 位来自 PCLATH; 二是当执行跳转指令 GOTO 或调用子程序指令 CALL 时, PC 的低 11 位来自指令码中直接携带的 11 位地址, 高 2 位来自 PCLATH。关于在此提到的 CALL 和 GOTO 等指令会在后面的指令系统中作详细介绍。

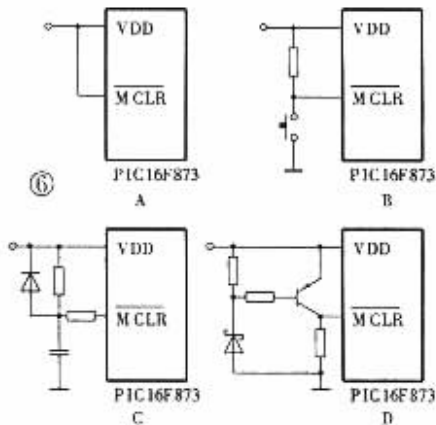
复位与系统时钟

复位功能 电子爱好者都知道, 在通用数字集成电路中, 比如 CMOS 和 TTL 系列, 有各式各样的计数器。这些计数器一般都具备一个复位端, 在计数过程中一旦该脚加入有效电平, 就会强迫计数器回零, 再从 0 开始计数。与此类似, 单片机也有一个复位端, 以便于人为地输入复位信号。除了人工复位之外, 单片机还有其它几种自动实现复位的途径。

PIC16F873 的复位功能设计得比较完善, 实现复位或者说引起复位的条件和原因可以归纳成以下 4 类: (1) 人工复位。无论是单片机在正常运行程序, 还是处在睡眠状态, 还是出现死机, 只要在人工复位端 MCLR 加入低电平信号, 就会令其复

位; (2) 上电复位。每次单片机加电时, 上电复位电路都要对电源电压 VDD 的上升过程进行检测, 当 VDD 上升到规定值 $1.6 \sim 1.8V$, 就产生一个有效的复位信号, 需经 $72ms + 1024$ 个时钟周期的延时, 才会使单片机复位; (3) 看门狗复位。不论何种原因, 只要没有对看门狗定时器 WDT 进行周期性及时地清零, WDT 就会出现超时溢出, 也就会引发单片机复位; (4) 欠压复位。前面已经作过介绍, 此处不再重述。

外部复位电路的几种接法如图 6 所示。最简单的接法是将 MCLR 直接接 VDD(图 6-A)。接一只按钮开关和一只电阻则便于手工复位操作(图 6-B)。针对



VDD 上升缓慢的应用场合, 图 6-C 可以延长复位时间, 确保单片机运行可靠。在另一些应用中可能需要对 VDD 严密监视, 一旦发现 VDD 跌落到某一门限值时(即在 $VDD < VZ + 0.7V$ 时, VZ 是稳压管的稳定电压), 就会使芯片复位, 以免系统失控, 这时可按图 6-D 设计复位电路。

图 6 不仅给出了 MCLR 脚的几种不同接法, 也给出了几种不同的对复位功能的开发利用方法。

系统时钟 单片机内部的各种功能电路几乎全部是由数字电路组成的。数字电路的工作离不开时钟信号, 每一步细微动作都是在一个共同的时间基准信号驱动之下完成的。作为时基发生器的时钟振荡电路, 为整个单片机芯片的工作提供系统时钟信号, 也为单片机与其它外接芯片之间的通信提供可靠的同步时钟信号。

PIC16F873 的时钟电路是由片内的一只反相器和一只反馈电阻, 与外接的一只石英晶体和两只电容器共同构成的一个自激多谐振荡器, 其电路如图 7 所示。

构成振荡器的一只反相器是一只具有受控端的三态门, 当执行睡眠指令 SLEEP 时, 三态门输出端呈现高阻状态, 令时钟电路停振, 从而迫使单片机的绝大部分片内电路停止工作, 进入低功耗模式, 达到节电的目的。时钟信号

是经过另一只反相器缓冲后, 被输送到内部各功能电路的。外接电路中的各元器件参数如表 2 所列。图 7 也给出了 OSC1 和 OSC2 两脚的一般接法。

XTAL	C1	C2
200kHz	47 ~ 68pF	47 ~ 68pF
1MHz	15pF	15pF
4MHz	15pF	15pF

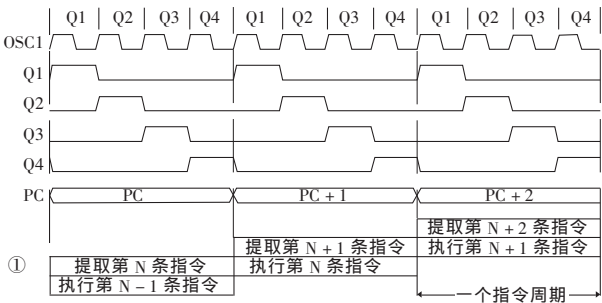
编者附记 本刊将与 MICROCHIP 公司合作, 配套供应 PIC16F87x 在线调试工具, 价格约在 400 ~ 450 元左右, 详情请读者留意近期《电子世界》本栏目。

另: 作者李学海老师的通信地址为: 石家庄市河北省邮电

第四讲 指令时序与指令系统 (上)

指令时序

时钟振荡器电路产生的时钟信号,经内部4分频后形成4个不重叠的方波信号Q1~Q4,叫作4个节拍。由4个节拍构成一个“指令周期”,所以,一个指令周期内部包含4个时钟周期,如图1所示。



在前一个指令周期内完成取指操作,在后一个指令周期内完成指令的译码和执行。当Q1节拍的上升沿出现时,程序计数器PC增1,指令码是在Q4节拍取出并放入指令寄存器的。指令码的译码和执行贯穿下一个指令周期的Q1~Q4节拍。乍看起来,一条指令的全部运行时间似乎是两个指令周期,但是,由于PIC单片机内部采用哈佛总线结构,使得它在执行一条指令的同时,就可以提取下一条指令,从而实现“流水作业”。这样就使每一条指令的运行时间平均为一个指令周期,因此,人们习惯于说PIC单片机采用的是“单周期”指令。严格地说,大多数指令的运行时间都是一个指令周期,但是,少数引起程序执行顺序发生跳转的指令则是两个周期,其原因分析放到指令系统部分去介绍。

指令系统

一种单片机所能识别的全部指令的集合,就称为该单片机的指令系统。不同厂家生产的单片机,或者基于不同CPU内核的单片机,一般具有不同的指令系统。指令系统中的每一条指令都完成一种特定的操作,如数据传送操作、加法操作、减法操作、逻辑“与”操作、逻辑“或”操作、左移操作、右移操作,等等。无论要求单片机实现多么复杂的控制操作,都可以由这些简单操作拼凑而成。换言之,无论多么复杂的操作任务,都可以分解成一系列简单的操作。将若干条实现简单操作的指令语句,按照一定的规则排列组合在一起,就构成了一个可以完成复杂功能的程序。

如果要为某种单片机编写程序,就要学习、记忆和应用该单片机指令系统的每一条指令,至少应了解每一条指令的功能,使用时会查阅指令表。为了便于学习和掌握,通常每一条指令都用指意性很强的英文单词或缩写来代表。例如,MOVWF,就是由“Move W to f.”一句英文缩略而成,中文含义

是“将寄存器W的内容移动到寄存器f中。”所以,人们又通常将代表一条指令的一个字符串称为“助记符”。假如按指令被使用的频繁程度划分整个指令系统的话,其实只有一部分指令在编写程序时经常用到,而另一部分指令却较少使用,还有一部分指令极少用到。每条指令一般都由操作码和操作数组成,也有个别指令不带操作数。操作码是指令操作功能的记述,而操作数描述操作的对象和操作的范围。

PIC16F873共有35条指令,均是长度为14位的单字节指令。所有指令按其操作对象的不同又可分为三类,即面向字节操作类(17条),面向位操作类(4条)以及常数操作和控制操作类(14条)。指令代码中所用到描述符号及其说明见表1。

表 1	
符号	说 明
W	代表工作寄存器(即累加器)
f	代表7位寄存器单元地址,最多可区分128个单元($2^7=128$)
b	表示某一比特在一个寄存器内部8比特数据中的位置(即伴地址),由3位组成($2^3=8$)
k	代表8比特数据常数,或者代表11比特地址常数
d	代表目标寄存器 $d=0$,目标寄存器为W; $d=1$,目标寄存器为f
→	表示运算结果送入目标寄存器
∧	代表逻辑“或”运算符
∨	代表逻辑“或”运算符
⊕	代表逻辑“异或”运算符

上述三种类型指令代码分配格式如下:

- (1) 面向字节操作类指令代码分配格式(如右):

13~8	7	6~0
操作码	d	f(寄存器地址)
- (2) 面向位操作类指令代码分配格式(如右):

13~8	9~7	6~0
操作码	b	f(寄存器地址)
- (3) 常数操作和控制操作类指令代码分配格式,又分3种情况:
携带8比特常数的指令代码分配格式(如右):

13~8	7~0
操作码	k(数据)

携带13比特常数的CALL和GOTO指令代码分配格式(如右):

13~11	10~0
操作码	k(程序地址)

不携带常数的指令代码分配格式(如右):

13~0
操作码

面向字节操作类指令与面向位操作类指令

1. 面向字节操作类指令 面向字节操作类指令有17条:
(1) 寄存器加法指令 格式:ADDWF f,d ;W寄存器内容和f寄存器内容相加,结果存入f($d=1$)或W($d=0$)。操作: $W+f \rightarrow d$,影响状态位C、DC和Z。
(2) 寄存器减法指令 格式:SUBWF f,d ;f寄存器的内容减去W寄存器的内容,结果存入W($d=0$)或f($d=1$)。操作: $f-W \rightarrow d$,影响状态位C、DC和Z。
(3) 寄存器加1指令 格式:INCF f,d ;f寄存器内容加1后,结果送寄存器W($d=0$)或f($d=1$)。操作: $f+1 \rightarrow d$,影响状态位Z。

(4)寄存器减1指令 格式 :DECF f, d ;f 寄存器内容减1后 ,结果存入 W(d=0)或 f(d=1)。操作 $f-1 \rightarrow d$,影响状态位 Z。

(5)寄存器逻辑与指令 格式 :ANDWF f, d ;W 寄存器内容和 f 寄存器内容相与 ,结果存入 f(d=1)或 W(d=0)。操作 : $W \wedge f \rightarrow d$,影响状态位 Z。

(6)寄存器逻辑或指令 格式 :IORWF f, d ;W 寄存器内容和 f 寄存器内容相或 ,结果存入 f(d=1)或 W(d=0)。操作 : $W \vee f \rightarrow d$,影响状态位 Z。

(7)寄存器逻辑异或指令 格式 :XORWF f, d ;W 寄存器内容和 f 寄存器内容相异或 ,结果存入 f(d=1)或 W(d=0)。操作 : $W \oplus f \rightarrow d$,影响状态位 Z。

(8)寄存器取反指令 格式 :COMF f, d ;f 寄存器内容取反后 ,结果存入 f(d=1)或 W(d=0)。操作 f 取反 $\rightarrow d$,影响状态位 Z。

(9)寄存器清零指令 格式 :CLRF f ;f 寄存器内容被清为零。操作 $0 \rightarrow f$,使状态位 Z=1。

(10)W 清零指令 格式 :CLRWF ;W 寄存器内容被清为零。操作 $0 \rightarrow W$,使状态位 Z=1。

(11)寄存器传送指令 格式 :MOVF f, d ;将 f 寄存器内容传送到 f 本身(d=1)或 W(d=0)。操作 $f \rightarrow d$,影响状态位 Z。

(12)W 寄存器传送指令 格式 :MOVWF f ;将 W 寄存器内容传送到 f, W 内容不变。操作 : $W \rightarrow f$;不影响状态位。

(13)递增跳转指令 格式 :INCFSZ f, d ;f 寄存器内容加1 ,结果存入 f 本身(d=1)或 W(d=0) ,如果结果为 0 则跳过下一条指令 ,否则顺序执行。操作 : $f+1 \rightarrow d$, $f+1=0$ 则 $PC+1 \rightarrow PC$,影响状态位 Z。

(14)递减跳转指令 格式 :DECFSZ f, d ;f 寄存器内容减1 ,结果存入 f 本身(d=1)或 W(d=0) ,如果结果为 0 则跳过下一条指令 ,否则顺序执行。操作 $f-1 \rightarrow d$, $f-1=0$ 则 $PC+1 \rightarrow PC$,影响状态位 Z。

(15)寄存器带进位位循环左移指令 格式 :RLF f, d ;将 f 寄存器带 C 循环左移 ,结果存入 f 本身(d=1)或 W(d=0) ,如图 2。操作 $f(n) \rightarrow f(n+1)$, $f(7) \rightarrow C$, $C \rightarrow d(0)$,影响状态位 C。

(16)寄存器带进位位循环右移指令 格式 :RRF f, d ;将 f 寄存器带 C 循环右移 ,结果存入 f 本身(d=1)或 W(d=0) ,如

图 3。操作 $f(n) \rightarrow f(n-1)$, $f(0) \rightarrow C$, $C \rightarrow d(7)$;影响状态位 C。

(17)寄存器半字节交换指令 格式 :SWAPF f, d ;f 寄存器高 4 位和低 4 位交换位置后 ,结果存入 f 本身(d=1)或 W(d=0) ,如图

4。操作 $f(7:4) \rightarrow f(3:0)$, $f(3:0) \rightarrow f(7:4)$;不影响状态位。

为了便于读者查找 ,表 2 列出了面向字节操作类的指令。

表 2

助记符	操作说明	影响状态寄存器的位
ADDWF f, d	$w+f \rightarrow d$	C, DC, Z
INCF f, d	$f+1 \rightarrow d$	Z
SUBWF f, d	$f-w \rightarrow d$	C, DC, Z
DECF f, d	$f-1 \rightarrow d$	Z
ANDWF f, d	$w \wedge f \rightarrow d$	Z
IORWF f, d	$w \vee f \rightarrow d$	Z
XORWF f, d	$w \oplus f \rightarrow d$	Z
COMF f, d	f 取反 $\rightarrow d$	Z
CLRF f, d	$0 \rightarrow f$	Z
CLRW f, d	$0 \rightarrow w$	Z
MOVF f, d	$f \rightarrow d$	Z
MOVWF f, d	$w \rightarrow f$	-
INCFSZ f, d	$f+1 \rightarrow d$,结果若为 0 则跳一步	-
DECFSZ f, d	$f-1 \rightarrow d$,结果若为 0 则跳一步	-
RLF f, d	f 带 C 左移 $\rightarrow d$	C
RRF f, d	f 带 C 右移 $\rightarrow d$	C
SWAPF f, d	f 半字节交换 $\rightarrow d$	-

2. 面向位操作类指令 面向位操作类指令有以下 4 条 :

(1)位清零指令 格式 :BCF f, b ;将寄存器的第 b 位清零。操作 $0 \rightarrow f(b)$;不影响状态位。

(2)位置 1 指令 格式 :BSF f, b ;将寄存器的第 b 位置 1。操作 $1 \rightarrow f(b)$;不影响状态位。

(3)位测试为 0 跳转指令 格式 :BTFSC f, b ;测试 f 寄存器的第 b 位 ,若 $f(b)=0$ 则跳过下一条指令 ,否则顺序执行。操作 :检测 $f(b)=0$ 则 $PC+1 \rightarrow PC$;不影响状态位。

(4)位测试为 1 跳转指令 格式 :BTFSS f, b ;测试 f 寄存器的第 b 位 ,若 $f(b)=1$ 则跳过下一条指令 ,否则顺序执行。操作 :检测 $f(b)=1$ 则 $PC+1 \rightarrow PC$;不影响状态位。

表 3

助记符	操作说明	影响状态寄存器的位
BCF f, b	将 f 中第 b 位清 0	-
BSF f, b	将 f 中第 b 位置 1	-
BTFSC f, b	f 中第 b 位为 0 ,则跳一步	-
BTFSS f, b	f 中第 b 位为 1 ,则跳一步	-



单片机硬件、软件及其应用讲座(6)

· 李学海 ·

第四讲 指令时序与指令系统 (下)

面向常数操作和控制操作类指令

面向常数操作和控制操作类指令共有以下 14 条:

(1) 常数加法指令 格式: $\text{ADDLW } k$; W 寄存器内容和 8 位立即数相加, 结果存入 W 。操作: $W + k \rightarrow W$; 影响状态位 C 、 DC 和 Z 。

(2) 常数减法指令 格式: $\text{SUBLW } k$; 8 位立即数减掉 W 寄存器内容, 结果存入寄存器 W 。操作: $k - W \rightarrow W$; 影响状态位 C 、 DC 和 Z 。

(3) 常数逻辑与指令 格式: $\text{ANDLW } k$; W 寄存器内容和 8 位立即数相与, 结果存入寄存器 W 。操作: $W \wedge k \rightarrow W$; 影响状态位 Z 。

(4) 常数逻辑或指令 格式: $\text{IORLW } k$; W 寄存器内容和 8 位立即数相或, 结果存入寄存器 W 。操作: $W \vee k \rightarrow W$; 影响状态位 Z 。

(5) 常数逻辑异或指令 格式: $\text{XORLW } k$; W 寄存器内容和 8 位立即数相异或, 结果存入寄存器 W 。操作: $W \oplus k \rightarrow W$; 影响状态位 Z 。

(6) 看门狗定时器清零指令 格式: CLRWDWT ; 将 WDT 寄存器和分配给它的预分频器同时清为全零。操作: $0 \rightarrow WDT$, $0 \rightarrow WDT$ 预分频器, 影响状态位 $1 \rightarrow \overline{TO}$, $1 \rightarrow \overline{PD}$ 。

(7) 常数传送指令 格式: $\text{MOVLW } k$; 将 8 位立即数传送到 W 寄存器。操作: $k \rightarrow W$; 不影响状态位。

(8) 子程序调用指令 格式: $\text{CALL } k$; 首先将 $PC + 1$ 推入堆栈, 然后将 11 位常数 k 送入 $PC(10 \sim 0)$, 同时将 $PCLATH(4, 3) \rightarrow PC(12, 11)$, 从而使 $PC =$ 子程序入口地址。操作: $PC + 1 \rightarrow$ 堆栈, $k \rightarrow PC(10 \sim 0)$, $PCLATH(4, 3) \rightarrow PC(12, 11)$; 不影响状态位。

(9) 无条件跳转指令 格式: $\text{GOTO } k$; 将 11 位常数 k 送入 $PC(10 \sim 0)$, 同时将 $PCLATH(4, 3) \rightarrow PC(12, 11)$, 从而使 $PC =$

新地址。操作: $k \rightarrow PC(10 \sim 0)$, $PCLATH(4, 3) \rightarrow PC(12, 11)$; 不影响状态位。

(10) 子程序返回指令 格式: RETURN ; 将堆栈顶端单元内容弹出并送入 PC , 从而返回主程序断点处。操作: 栈顶 $\rightarrow PC$; 不影响状态位。

(11) 子程序带参数返回指令 格式: $\text{RETLW } k$; 将堆栈顶端单元内容弹出并送入 PC , 同时 8 位常数 $k \rightarrow W$, 从而带着参数返回主程序断点处。操作: 栈顶 $\rightarrow PC$, $k \rightarrow W$; 不影响状态位。

(12) 中断服务子程序返回指令 格式: RETFIE ; 将堆栈顶端单元内容弹出并送入 PC , 从而返回主程序断点处, 同时将“全局中断使能位 GIE (该位位于中断控制寄存器 $INTCON$, 以后介绍)”置 1, 重新开放中断。操作: 栈顶 $\rightarrow PC$, $1 \rightarrow GIE$; 不影响状态位。

(13) 睡眠指令 格式: SLEEP ; 该指令执行后, 单片机进入低功耗睡眠模式, 时基电路停振。操作: $0 \rightarrow \overline{PD}$, $1 \rightarrow \overline{TO}$, $0 \rightarrow WDT$, $0 \rightarrow WDT$ 预分频器, 影响状态位 \overline{TO} , \overline{PD} 。

(14) 空操作指令 格式: NOP ; 不产生任何操作, 仅使 PC 加 1, 消耗一个指令周期。操作: 空操作; 不影响状态位。

面向常数操作和控制操作类指令见表 4。

寻址方式

指令的一个重要组成部分就是操作数, 由它指定参与运算的数据或者数据所在的单元地址。所谓寻址方式, 就是寻找操作数的方法, 就是给操作数定位的过程, 就是获取操作数所在单元的地址的途径。

在 PIC16F873 的指令系统中, 根据操作数的来源不同, 设计了四种寻址方式, 即立即寻址、直接寻址和寄存器间接寻址、位寻址。

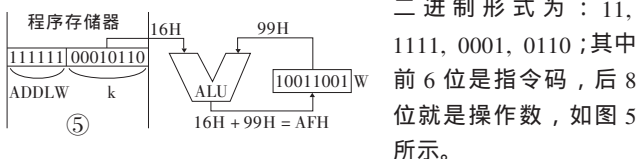
1. 立即寻址 在这种寻址方式中, 指令码中携带着实际



表 4

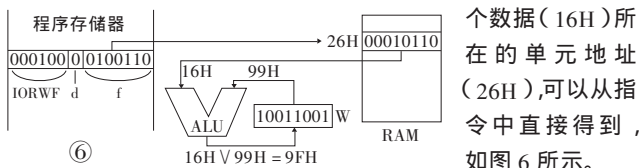
助记符	操作说明	影响状态寄存器的位
ADDLW k	$w + k \rightarrow w$	C, DC, Z
SUBLW k	$k - w \rightarrow w$	C, DC, Z
ANDLW k	$w \wedge k \rightarrow w$	Z
IORLW k	$w \vee k \rightarrow w$	Z
XORLW k	$k \oplus w \rightarrow w$	Z
CLRWDI -	$0 \rightarrow WDT$	TO, PD
MOVLW k	$k \rightarrow w$	-
CALL k	调用子程序	-
GOTO k	无条件跳转	-
RETURN -	从子程序返回	-
RETLW k	w 带参数子程序返回	-
RETFIE -	从中断服务子程序返回	-
SLEEP -	进入睡眠方式	TO, PD
NOP -	空操作	-

操作数(就称立即数),换言之,操作数可以在指令码中立即获得,而不用到别处去寻觅。举例:ADDLW 16H;将立即数 16H 与 W 内容(假设为 99H)相加,结果(AFH)送到 W。其指令码的



二进制形式为: 11, 1111, 0001, 0110;其中前 6 位是指令码,后 8 位就是操作数,如图 5 所示。

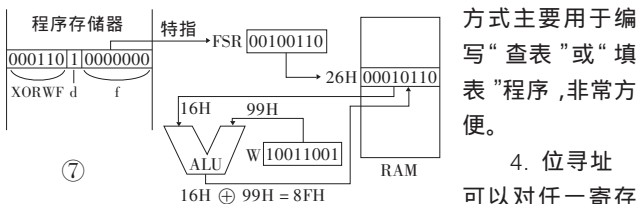
2. 直接寻址 采用直接寻址方式的指令,可以直接获取任一寄存器单元的地址,即指令码中包含着被访问寄存器的单元地址。举例:IORWF 26H, 0, 实现将地址为 26H 的 RAM 单元的内容(假设为 16H)与 W 的内容(假设为 99H)相“或”后,结果(9FH)送入 W 中,因为 d=0。参加逻辑“或”运算的一个数据(16H)所



一个数据(16H)所在的单元地址(26H),可以从指令中直接得到,如图 6 所示。

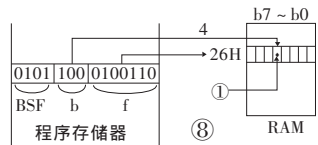
3. 寄存器间接寻址 在采用寄存器间接寻址方式的指令码中,7 位寄存器地址必为全 0。利用 7 位 0 这个专用地址,特指 FSR 寄存器,并且以 FSR 寄存器内容为地址的 RAM 单元中存放着参加运算或操作的数据。从表面上看,指令码中的 7 位 0 指定的是 INDF 单元,其实 INDF 仅仅是一个假想的、不存在的寄存器单元,只不过是将其地址编码给专用了。这样做的理由是可以大大简化指令系统。

为了便于理解,我们不妨换一个角度来讲解,把 FSR 看成一个特殊的具有两个地址(00H 和 04H)的寄存器单元,而可以不提 INDF。当用 04H 访问 FSR 时,它像普通寄存器的一个单元一样,可以直接对 FSR 进行读或写;而用 00H 访问 FSR 时,它就是一个间接寻址寄存器,不是对 FSR 进行读或写,而是把 FSR 的内容作为地址使用。举例:XORWF 0, 1, 将 26H 号 RAM 单元的内容(假设为 16H)与 W 内容(假设为 99H)相“异或”,运算结果(8FH)送回 26H 号 RAM 单元,因为 d=1。参加“异或”运算的一个数据可以从指令码中间接得到,如图 7 所示。FSR、W 和 26H 号 RAM 单元的内容都是预先存入的。间接寻址方式主要用于编写“查表”或“填表”程序,非常方便。



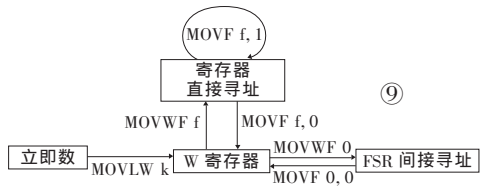
4. 位寻址 可以对任一寄存

器中的任一比特位直接寻址访问,即指令码中既包含着被访问寄存器的地址,又包含着该寄存器中某一比特位的位地址。如果将 RAM 存储器看成一个阵列的话,那么在这个阵列中寻找某一个比特,就需要一个纵坐标和一个横坐标。纵坐标就相当于单元地址,横坐标就相当于比特地址。举例:BSF 05H, 4;把地址为 05H 的寄存器单元内的第 4 位设置为 1,如图 8 所示。



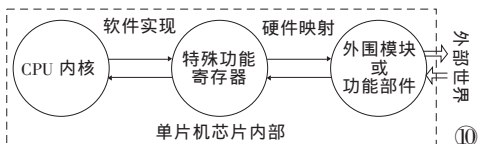
数据传递关系

数据的逻辑运算和算术运算过程,以及控制信号的输入和输出过程,在单片机内部都可以看成是“数据传递”的过程。根据 PIC16F873 的硬件系统和软件系统(即指令系统和寻址方式)的规划特点,我们可以总结出在单片机内部能够实现数据传递的几种途径。图 9 是对这几种数据传递的形象化描述,图中还给出了分别实现每一种数据传递所用指令的一个实例。



“内核-寄存器-外围模块”相互关系

单片机的开发和应用的主要任务有两项:一是软件设计,二是硬件设计。硬件设计我们暂且不提,在此只用软件设计的观点,从不同角度剖析单片机内部的组织关系。软件设计实际上就是运用指令编制程序。PIC 单片机的指令的作用范围非常集中,作用的对象也很单一,仅仅限制在(包含着文件寄存器和特殊功能寄存器的)RAM 数据存储器的范围之内。或者说,指令的操作对象主要就是 RAM 数据存储器的各个寄存器单元。单片机的工作就是用一条条的指令指挥各部分硬件的动作,而这种“指挥”就是通过给特殊功能寄存器填写相应的内容来实现的。因此,我们不妨将 PIC 单片机画成图 10 所示的形式,内核与寄存器之间存在灵活的“软件上的对应”关系,而寄存器与外围模块之间存在着固定的“硬件上的映射”关系。



寄存器与外围模块之间存在着固定的“硬件上的映射”关系。特殊功能寄存器在中间扮演着桥梁或者界面的角色。各个外围模块从外部世界采集的现场信息,经过硬件电路立即反映到与自己对应的特殊功能寄存器单元上,CPU 通过执行指令从该寄存器单元里获取相应的信息。相反,CPU 通过填写与某一外围模块对应的特殊功能寄存器单元,由该寄存器单元经过硬件电路将控制信息映射到外围模块上,再由外围模块驱动外接电路完成相应的动作,从而将 CPU 的命令落到实处。

我们在用指令编写程序时,不仅应搞清指令系统中每条指令的功能,还应弄清特殊功能寄存器与外围模块或功能部件之间的对应关系。这些对应关系在后面讲解各外围模块的专题中会陆续介绍。

单片机硬件、软件及其应用讲座(7)

· 李学海 ·

第五讲 PIC 汇编语言程序设计基础 (上)

单片机内部的电路基本上都是用数字逻辑电路构成的,而数字逻辑电路只能处理二进制代码“0”和“1”,因此,单片机仅能够识别二进制形式的机器语言程序(也称机器码程序)。所谓“机器语言”就是用二进制代码表示的能为计算机直接识别和执行的指令的集合,它是计算机的一种最低级的语言形式。前述指令系统中的每一条指令都有自己相应的机器语言形式。比如,睡眠指令“SLEEP”和加法指令“ADDWF f,d”的机器码(或称机器指令)分别为“00000001100011”和“000111df6~f0”(其中d和f6、f5……f0均代表一位二进制数码)。在机器指令中,操作码、操作数和地址码等都是用二进制代码表示的。如果直接使用机器语言来设计程序,编写起来很繁琐,容易出错,给程序的阅读、修改、调试等环节也都会带来极大的困难。为了克服这些困难,人们在开发应用单片机的实际工作中通常都使用汇编语言进行程序设计。

汇编语言是对机器语言的改进,它采用便于人们记忆的一些符号(例如简化的英文单词)来表示操作码、操作数和地址码等。汇编语言的语句通常与机器语言指令是一一对应的。用汇编语言编写的程序就称为汇编语言源程序(下文简称源程序)。由于单片机不认识汇编语言程序,所以开发人员需要在微机上运行一个由单片机制造商免费配套提供的、称作“汇编器”(或汇编程序)的软件,将自己编写的源程序翻译成机器语言程序。这种机器语言程序就称为“目标程序”。烧写到单片机程序存储器中的程序就是这种程序。利用汇编器将源程序翻译成目标程序的过程称为“汇编”。

汇编语言提供了一种不涉及实际存储器地址和机器指令编码的编写源程序的有效方法。为了掌握这种方法,我们需要了解汇编器所约定的一些内容(汇编语言的语句格式、伪指令及程序格式等),以及程序的4种基本结构(顺序、分支、循环和子程序)等。

在编制PIC系列单片机的程序时会遇到几个特有问题的,即RAM数据存储器的体选寻址、程序存储器的跨页跳转、复位矢量和中断矢量的安排等问题。在实际工作中常用的几种程序有延时程序、查表程序等。

汇编语言的语句格式

为了利用能在微机上的汇编器对源程序进行自动汇编,在给PIC系列单片机编写源程序时,必须依照所用汇编器的一些约定进行书写。例如使用微芯公司提供的汇编器“MPASM”对源程序进行汇编时,典型的汇编语言语句格式由以下4部分组成:

标号:操作码(指令助记符) 操作数 注释

这4部分的顺序不能颠倒。标号必须从最左边第一列开始书写,其后至少用一个空格与操作码隔离,在没有标号的语句中,指令操作码前面必须保留一个或一个以上的空格。操作码与

操作数之间也必须保留一个或一个以上的空格。操作码后面如果跟随两个操作数,则操作数之间必须用(半角)逗号隔开。在必要时可以加注释,注释可以跟在操作码、操作数或标号之后,并用分号引导,甚至可以单独占用一行且可以从任何一列开始。汇编语言源程序既可以用大写字母书写,也可以用小写字母书写,还可以大写小写混用,以便于阅读。一个语句行最多允许有225个(半角)字符。

1. 标号 用在指令助记符之前的标号就是该指令的符号地址,在程序汇编时,它被赋以该指令在程序存储器中所存放的具体地址。并不是每一条语句都需要加标号,只有那些欲被其他语句引用的语句之前才需要加标号。标号可以单独作为一行。

标号最多可以由32个字母、数字和其他一些字符组成,且第一个字符必须是字母或下划线“_”,必须从一行的第一列开始写,后面用空格、制表符或换行符与操作码隔开。标号不要用指令助记符、寄存器代号或其他在系统中已有固定用途的字符串。一个标号在程序中只能定义一次。

2. 操作码 操作码可以是指令系统中的助记符,也可以是用于控制汇编器的伪指令。操作码前面至少保留一个空格,以便与标号区别。

3. 操作数 操作数是操作对象,也就是数据或者地址,可以用常数或符号两种形式表示。其中,“常数”可以是二进制、八进制、十进制、十六进制数

或者字符(各种数制的表示法见表1,其中每一种数制又有几种可以通用的描述法);“符号”可以是在此之前经过定义(或者赋值)的代表数据或地址的标号或字符串。如果操作数有两个,中间应该用逗号分开。与许多其它单片机的汇编器不同的是,MPASM的默认进制不是十进制,而是十六进制。

数制	格 式	举例
十六进制	H'十六进制'	H'9E'
	十六进制 H	9EH
	Qx 十六进制	0x9E
八进制	O'八进制'	O'87'
	Q'八进制	Q'87'
	八进制 O	87O
	八进制 Q	87O
十进制	D'十进制'	D'123'
	十进制	.123
二进制	B'十进制'	B'11001010'
	二进制 B	11001010B
字符	'字符'	'G'
	A'字符'	A'G'

对于表1需要作以下两点说明:(1)十六进制数由数字0~9和字母A~F组成。当在源程序中采用后缀“H”表示一个以A~F打头的16进制数时,则必须在它的前面增添一个“0”作为引导,以便于汇编器将其与标号或符号名相区别。如16进制数FF应表示为0FFH。(2)用字符代表的常数就是该字符对应的ASCII码(即“美国标准信息交换码”,其长度为7比特,许多计算机原理书或高级语言程序设计书中都能找到ASCII表)。例如,‘1’,‘A’,‘\$’和‘{’的ASCII码分别为32H,41H,24H和7BH。

4. 注释 注释部分可有可无,但是最好养成附带注释的

习惯。注释用来对程序作一些注解和说明,便于人们阅读、交流、修改和调试程序。注释不是程序的功能部分,通常用半角分号引导或与指令部分隔开,汇编器对该部分不作任何处理。加注释时,一般应该说说明指令的作用和执行的条件,尤其要说明程序在做什么;在用到了子程序时,要说明子程序的入口条件和出口条件以及该程序完成的功能。

伪指令

用来编写汇编语言源程序的语句,主要是指指令助记符,其次就是伪指令。所谓伪指令就是“假”指令的意思,不是单片机的指令系统中的真实指令。它与指令系统中的助记符的不同之处是没有机器码与它对应。当源程序被汇编成目标程序时,目标程序中并不出现这些伪指令的代码,它们仅在汇编过程中起作用。伪指令是程序设计人员向汇编器发布的控制命令,告诉汇编器如何完成汇编过程和一些规定的操作,以及控制汇编器的输入、输出和数据定位等。汇编器 MPASM 可以使用的伪指令多达数十条,不过,在此仅仅介绍以下几条最常用的伪指令:

(1) EQU(符号名赋值伪指令) 格式:符号名 EQU nn EQU 的意思是使 EQU 两端的值相等,即给符号名赋予一个特定值或者说是给符号名定义一个数值。格式中的符号名通常是代表寄存器名称或专用常数的一个字符串,“nn”通常是一个不大于 8 比特二进制数的数值。一个符号名一旦由 EQU 赋值,其值就固定下来了,不能再被重新赋值。对符号名的要求类似于对标号的要求,比如符号名应从一行的第一列开始书写,其后至少保留一个空格与 EQU 隔离。

(2) ORG(程序起始地址定义伪指令) 格式: ORG nnnn ORG 用于指定该伪指令后面的源程序存放的起始地址,也就是汇编后的机器码目标程序在单片机的程序存储器中开始存放的首地址。nnnn 是一个 13 比特长的地址参数。

(3) END(程序结束伪指令) 格式:END END 伪指令通知汇编器 MPASM 结束对源程序的汇编。在一个源程序中必须要有并且只有一条 END 伪指令,放在整个程序的末尾。

程序流程图和程序格式

1. 程序流程图 通常在编写程序之前,需要画程序流程图。程序流程图是一种图解表示方法,比用文字和数学表达式来描述程序的基本思路要直观得多,使设计者可以直接了解整个系统及各部分之间的相互关系。它被很多没有程序设计基础的人所理解。程序流程图反映出操作顺序,因而有助于分析出错的原因。可以说,流程图是一种图形语言,亦即它用各种图形符号来说明程序的执行过程。常用的图形符号有圆角矩形框、矩形框、菱形框及指向线四种。圆角矩形框为起始/终止框,表示一个程序的开始或结束。矩形框为任务框,表示要处理的任务。菱形框为判断框,表示要判断的因素,判断结果将导致程序走入不同的分支。指向线为带有箭头的线段,表示程序的走向。当所设计的程序较小或者较简单时,流程图也可以不用画到纸上,但在头脑中应形成清晰的执行顺序。

2. 程序格式 一般来说,PIC 的源程序并没有规定的统一格式,大家可以根据自己的风格来编写。对于一个完整程序的总体布局,在这里推荐一种格式供大家参考。

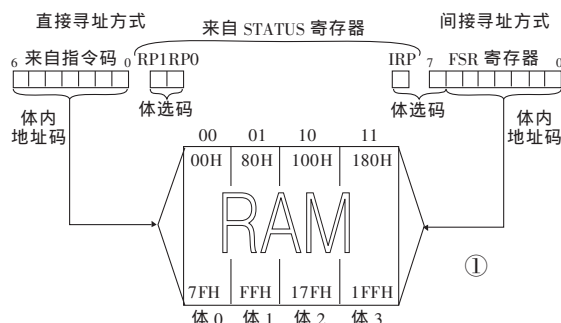
```
;- - - - -
;
;- - - - -
;符号名定义和变量定义
;- - - - -
INDF EQU 00H ; 各操作寄存器地址定义,也就是说,把
TMR0 EQU 01H ; 后面程序的指令中将要用到的寄存器
PCL EQU 02H ; 单元地址和位地址用表义性很强的符
STATUS EQU 03H 号名预先定义
FSR EQU 04H
PORTA EQU 05H
TRISA EQU 85H
X EQU 20H ;对程序所需变量预先进行定义
Y EQU 21H
;- - - - -
;- - - - -
;复位矢量和中断矢量安排(对于 PIC16F873)
;- - - - -
;- - - - -
ORG 0000H ;地址 0000H 为复位矢量
GOTO MAIN ;跳转到主程序
ORG 0004H ;地址 0004H 为中断矢量
GOTO INT_BODY ;跳转到中断服务程序
;- - - - -
;- - - - -
;主程序区
;- - - - -
;- - - - -
ORG 0005H ;从 0005H 开始存放主程序
MAIN CLRW
CALL SUB
.....
GOTO MAIN
;- - - - -
;- - - - -
;子程序和中断服务程序区
;- - - - -
;- - - - -
SUB MOVLW 01H ;子程序
.....
RETURN ;子程序返回
;
INT_BODY ;中断服务程序
MOVLW 0FFH
.....
RETIE ;中断服务程序返回
;- - - - -
;- - - - -
END ;全部程序结束
```

RAM 数据存储器的体选寻址

在讲解后面的内容之前,必须搞清 PIC16F873 单片机中

RAM 数据存储器的布局图和“体 (Bank)”的概念。第四讲业已讲过,所有面向字节操作和面向位操作的指令,其指令代码中均包含一个 7 比特长的 RAM 数据存储器单元地址 f_6 。因为 2 的 7 次方等于 128,所以 f_6 最多可以区分 128 个存储器单元。但是,事实上 PIC16F873 内部的 RAM 配置了 512 个单元的地址空间,是 128 的 4 倍,地址编码长度需要 9 比特,从 000H 到 1FFH。如果想用 7 比特地址码(从 00H 到 7FH)实现对 512 个单元的寻址,就必须对 RAM 采取一种新的组织方法。这就是将长度为 512 的 RAM 均匀划分为 4 等份,每一等份称作一个“体”,按地址从小到大的顺序分别记为“体 0”、“体 1”、“体 2”和“体 3”。区分 4 个体需要 2 比特地址码(从 00B 到 11B),可以称该地址码为“体选码”。

图 1 是关于体选寻址的示意图。通常把体 0~体 3 这 4 个体按横向顺序排列,每个体内含有 128 个单元。当访问 RAM 中的某一个单元时,首先要确定包含该单元的体作为“当前体”(开机后单片机自动将“体 0”默认为当前体),然后用包含在指



令码中的 7 比特地址码 (即体内地址码) 对指定单元进行定位。对 RAM 直接寻址时,两位体选码来自状态寄存器 STATUS 的 RP0 和 RP1 位;对 RAM 进行间接寻址时,两位体选码来自状态寄存器 STATUS 的 IRP 位和 FSR 寄存器的最高位。也可以这样认为,一个单元的位置是由体选码和体内地址码两部分确定的。

RAM 数据存储单元中的各个寄存器单元的功能和地址分配情况如图 2 所示。从该图中可以看出 (1) 有些寄存器单元具有 4 个不同的地址,比如 STATUS,不论当前体是哪一个, $f_6 = 03H$ 的 7 位地址码都能找到同一个单元 STATUS。也就是说,可以忽略体选码而只用 7 位体内地址码,即可对 STATUS 单元寻址。这样会给这类寄存器单元的寻址带来很大的方便。(2) 根据够用即可的原则, PIC16F873 在 000H 到 1FFH 的地址空间里,实际并没有配置 512 个真实的单元,阴影标出部分没有配置。(3) 特殊功能寄存器安排在 4 个体的上半部分,而文件寄存器占据着下半部分。(4) 总共 192 个单元的文件寄存器分作两半,分别安排在体 0 和体 1 上,并且其中 96 个单元在体 0 和体 2 之间是互相映射的,即不论当前体是体 0 还是体 2,一个 7 位体内地址码都会找到体 0 中的一个真实单元。同理,另外 96 个单元在体 1 和体 3 之间是互相映射的。(5) 在 PIC16F873 的 RAM 中实际配置的特殊功能寄存器单元多达数十个,暂时用不到的在该图中没有全部画出来,用到时再作描述。

为了便于观察和理解,还可将图 2 进一步简化成图 3 的样子。从图 3 中可以看出,在 RAM 中寻找某一单元需用两个地址码,这就像在直角坐标系表示的平面上确定一个点一样,需要用到横坐标和纵坐标两个数据。

体 0	体 1	体 2	体 3
00H INDF	80H INDF	100H INDF	180H INDF
10H TMR0	81H OPTION-REG	101H TMR0	181H OPTION-REG
02H PCL	82H PCL	102H PCL	182H PCL
03H STATUS	83H STATUS	103H STATUS	183H STATUS
04H FSR	84H FSR	104H FSR	184H FSR
05H PORTA	85H TRISA	105H	185H
06H PORTB	86H TRISB	106H PORTB	186H TRISB
07H PORTC	87H TRISC	107H	187H
08H	88H	108H	188H
09H	89H	109H	189H
0AH PCLATH	8AH PCLATH	10AH PCLATH	18AH PCLATH
0BH INTCON	8BH INTCON	10BH INTCON	18BH INTCON
0CH	8CH	10CH	18CH
1FH	9FH	11FH	19FH
20H 96 字节文件寄存器	A0H 96 字节文件寄存器	120H 访问该区域时将自动影射到体 0 中	1A0H 访问该区域时将自动影射到体 1 中
7FH	FFH	17FH	1FFH

寄存器单元功能分配和地址分配

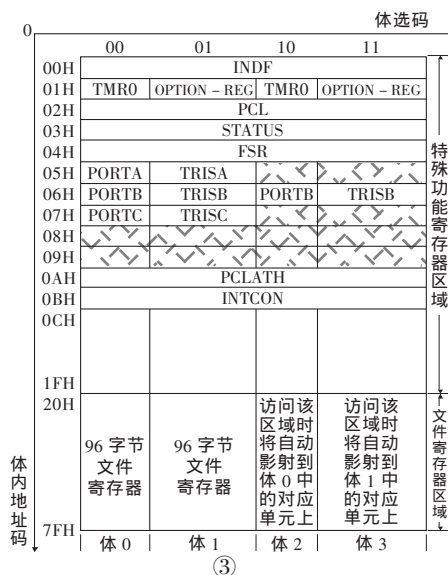
②

通常将“体 0”作为当前体,开机后单片机默认的当前体也是“体 0”。如果打算对“体 1”中的 TRISA 寄存器进行写入操作,那么事先需要设定 STATUS 寄存器中的体选码,事后还需要恢复该体选码。程序段的编写如下:

STATUS EQU 03H	将符号名 STATUS 定义为 03H,因 STATUS 寄存器地址为 03H
RP0 EQU 5H	将符号名 RP0 定义为 5H,因 RP0 比特的位地址为 5
HTRISA EQU 85H	将符号名 TRISA 定义为 85H,因 TRISA 寄存器地址为 85H
BSF STATUS, RP0	将状态寄存器的第 5 比特置 1,以选择体 1
MOVLW 0FH	将 0FH 送入工作寄存器 W
MOVWF TRISA	将 W 中的 0FH 转送到 TRISA 寄存器中
BCF STATUS, RP0	将状态寄存器的第 5 比特清零,以恢复体 0 为当前体。✱

编者附记 本刊与福州高齐科技有限公司合作,供应 PIC16F87x 在线调试工具 ICD,凡《电子世界》读者优惠价 350.00(含邮费)。需购买的读者请汇款到福州市东大路 8 号花开富贵 A 座 20D 福州高齐科技有限公司 邮编 350001。

(汇款时请注明“《电子世界》读者”)



③

单片机硬件、软件及其应用讲座(8)

· 李学海 ·

第五讲 PIC 汇编语言程序设计基础 (中)

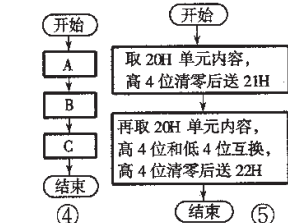
程序按其执行顺序或者行进路线可分为顺序结构、分支结构、循环结构和子程序结构四种基本结构。可以这么说,无论多么庞大复杂的程序均可看成由这4种基本结构组合而成。

顺序程序结构

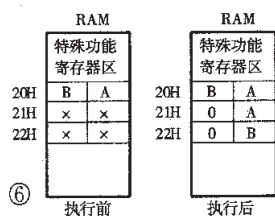
顺序程序结构是最简单的一种结构,在流程图中表示为任务框一个个地串行连接。在计算机执行程序时表现为从头至尾严格按照次序一条语句一条语句地顺序执行,并且每一条语句均被执行一遍。顺序程序流程图如图4所示。图中的A、B和C分别代表的可以是一条语句,也可以是一段程序。

[例1]当用LED数码管对某一RAM存储器单元的内容进行

显示时,通常需要将该单元的数据拆分成高4位和低4位两个“半字节”。本例中将20H单元的数据分解后,依次将低、高半字节放入21H和22H单元,并将这两个单元的高4位补零。图5是其流程图。下面就是实现这一功能的程序片段。



MOVWF 20H,0 将 20H 单元的内容送入 W
ADDLW 0FH ;W 高 4 位清零,低 4 位保持不变
MOVWF 21H 将拆分后的低 4 位送 21H 单元
SWAPF 20H,0 将 20H 单元内容高、低半字节换位后送 W
ADDLW 0FH ;再将 W 高 4 位清零,低 4 位保持不变
MOVWF 22H 将拆分后的高 4 位送 22H 单元



程序中的第2条语句中采用了“ANDLW”指令,将一个8比特数据同常数0FH相“与”,以实现清零高4位和保留低4位的目的,但这一操作只能在工作寄存器W内才能完成。程序执行前,假设原先20H单元的内容为BAH,21H和22H单元的内容是随机的或不确定的,程序执行后,20H单元内容不变,而21H和22H单元的内容变成了0AH和0BH,如图6所示。

需要说明的是,实现同一功能的程序不是唯一的,可以有多种不同的编写方法。

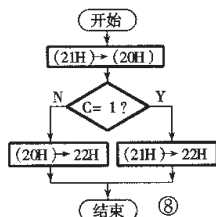
分支程序结构

分支程序流程图都包含一个判断框,该判断框具有一个入口和两个出口,从而形成程序的两个分支,如图7所示。在程序运行时究竟是执行B还是C,要由判断框内的条件判为“是”或“否”来决定。语句A执行完之后通常产生一个条件码cc,当条件cc判为“是”(记为YES或Y)时进入B分支;当条件cc判为“否”(记为NO或N)时进入C分支。由此可见,只有一个分支

中的程序被执行了一遍,而另一分支中的程序没有得到执行。在实际编程时,不仅会用到上述的二分支程序结构,还会用到分支数多于两个的多分支程序结构。不过,多分支结构可以看作由二分支结构嵌套而成,即分支中又包含分支。下面举一个二分支结构的例子。

[例2]将RAM的20H单元和21H单元中存放的两个数做比较,把其中的大者找出并存入22H单元。

该例子的程序的编程方法是将两个参与比较的数做减法运算,如果被减数小于减数,就会发生借位(C=0),否则,不发生借位(C=1)。判断标志位C的值,就可以挑出大数。该程序的流程图见图8。值得注意的是,图中带括号的数与不带括号的数含义不同,比如(21H)表示以21H为地址的单元的内容,而22H表示以22H为地址指定的单元。该例子的程序如下。

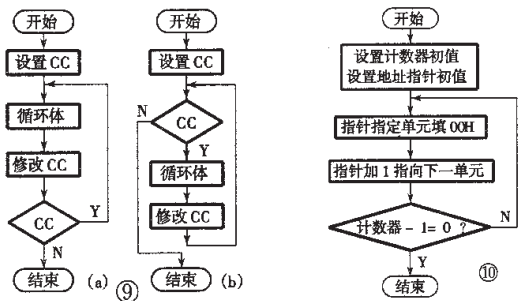


STATUS EQU 03H ;定义 STATUS 寄存器地址为 03H
C EQU 0H ;定义进位/借位标志 C 在 STATUS 中的位地址为 0
MOVWF 20H,0 将 20H 单元的内容送入 W
SUBWF 21H,0 21H 单元内容减去 W 内容,结果留在 W 中
BTFSF STATUS,C ;若 C=1 没借位,21H 中的数较大,则跳转到 F21BIG 处
GOTO F20BIG ;若 C=0 有借位,20H 中的数较大,则跳转至 F20BIG 处
F21BIG MOVF 21H 将 21H 中的数送入 W
MOVWF 22H ;再将它转存到 22H 单元
GOTO STOP ;跳过下面两条指令到程序末尾
F20BIG MOVF 20H 将 20H 单元的内容送入 W
MOVWF 22H ;再将它转存到 22H 单元
STOP GOTO STOP ;任务完成,停机

循环程序结构

在程序设计过程中,有时要求对某一段程序重复执行多遍,此时若用循环程序结构,有助于缩短程序。一个循环程序结构包含循环初设置、循环体和循环控制三部分。循环初设置就是在循环开始时,指定或定义一个循环变量cc(可以是循环次数计数器、地址指针等),并且给它设置一个初始值。循环体为要求重复执行的程序段。循环控制就是根据循环结束条件,判断是否结束循环。在循环程序中必须给出循环结束条件,否则就成为“死循环”。

图9为循环程序流程图。该图有两种画法。在图9(a)中循环体至少执行一次,这是因为先执行循环体,后判断循环结束条件;而在图9(b)中,先判断结束条件,再执行循环体,如果一开始就满足结束条件,则循环体将一次也不被执行。在实际编



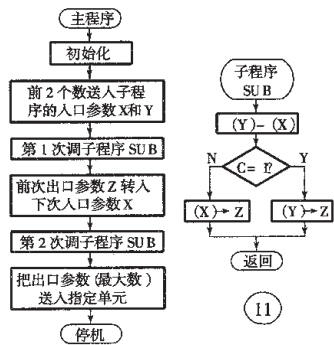
COUNT	EQU	20H	指定 20H 单元作为循环次数计数器(即循环变量)
FSR	EQU	04H	定义 FSR 寄存器地址为 04H
INDF	EQU	00H	定义 INDF 寄存器地址为 00H
	MOVLW	D'50'	把计数器初始值 50 送入 W
	MOVWF	COUNT	再把 50 转入计数器(作为循环变量的初始值)
	MOVLW	30H	将 30H(起始地址)送入 W
	MOVWF	FSR	再把 30H 转入寄存器 FSR(用作地址指针)
NEXT	CLRF	INDF	把以 FSR 内容为地址所指定的单元清零
	INCF	FSR, 1	地址指针内容加 1 指向下一个单元
	DECFSZ	COUNT, 1	计数值减 1 结果为 0 就跳过下一指令到 STOP 处
GOTO	GOTO	NEXT	跳转回去并执行下一次循环
STOP	GOTO	STOP	结束循环之后执行该语句, 实现停机

子程序结构

之间传递 8 位参数,用工作寄存器 W 是理想的选择。

[例 4] 将 RAM 中

在编写例 4 的程序时，我们可以把 [例 3] 中比较两个数的程序段设计成一个子程序 SUB，编写一个主程序 MAIN 来反复调用它。图 11 是该程序的流程图。



主程序：

;- - - - -

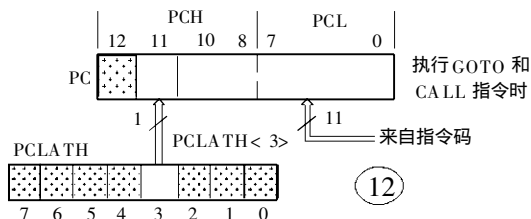
子程序:(入口参数:X和Y,出口参数:Z)

SUB	MOVF	X, 0	将 X 的内容送入 W
	SUBWF	Y, 0	:Y 内容减去 W 内容, 结果留在 W 中
	BTFSS	STATUS, C	若测得 C = 1, 即没发生借位, 意味 Y 中数较大, 则跳一步, 即跳转到程序的 Y_BIG 处
	GOTO	X_BIG	否则, C = 0 发生了借位, 意味着 X 中的数较大, 则跳转
Y_BIG	MOVF	Y	将 Y 中的数送入 W
	MOVWF	Z	再将它转存到 Z
	GOTO	THEEND	跳过下面两条指令到程序末尾
X_BIG	MOVF	X	将 X 中的数送入 W
	MOVWF	Z	再将它转存到 Z
THEEND	RETURN		子程序返回

第五讲 PIC 汇编语言程序设计基础 (下)

程序跨页跳转和调用

对于 PIC16F873, 实际配置的程序存储器容量为 $4K \times 14$ ($1K = 1024 = 2^{10}$), 其地址编码长度需要 12 比特, 2 的 12 次方等于 4096, 即 4K。但是, 两条引起程序跳转的指令 GOTO 和 CALL 所携带的地址码仅有 11 位, 2 的 11 次方等于 2K, 因而也就只能在 2K 的地址范围内跳转。所以就把 4K 的程序空间分为两页, 每页 2K, 页面 0 的地址范围为 0000H ~ 07FFH, 页面 1 的地址范围为 0800H ~ 1FFFH。再把 PCLATH <3> 位 (即该寄存器的第 3 位) 作为页面选择位, 简称页选位, 这样就可以在 4K 的地址范围内自由跳转了。方法是: 当发生跨越页面的



跳转时, 事先预置 PCLATH <3>, 使其指向所希望的页面 (0 或 1), 见图 12。在程序的执行过程中, 当遇到 GOTO 和 CALL 指令时, 程序计数器的低 11 位, 即 PC <10: 0> (代表 PC 的比特 10 ~ 比特 0) 由指令码携带的 11 位地址装载 (即刷新), 而 PC <11> 由 PCLATH <3> 自动装载。从而使程序的执行顺序发生改变。对于只配置了 4K 程序存储器的 PIC16F873, PC <12> 位没有用到。

【例 5】编制一个程序, 使其主程序部分放置在页面 0 (0000H ~ 07FFH) 内, 子程序部分放置在页面 1 (0800H ~ 0FFFH) 内。主程序对子程序的调用是一个跨页跳转。

```
PCLATH EQU 0AH      ;将符号名 PCLATH 定义为 0AH 因
                      ;PCLATH 寄存器地址为 0AH
                      ;地址为 0 的单元专门用作复位矢量
ORG 0000H            ;存放一条到主程序的跳转指令
GOTO MAIN            ;从页面 0 的 500H 单元开始存放主程序
ORG 0500H
MAIN BSF PCLATH, 3    ;预置页选位, 准备选择页面 1
CALL SUB             ;调用子程序, 引起程序跳转
LOOP .....           ;从子程序返回后执行的第一条指令
;-----
ORG 0900H            ;从页面 1 的 900H 单元开始存放
SUB MOVLW 00H         ;子程序
.....
RETURN              ;返回到位于页面 0 的主程序的 LOOP 处
END                  ;源程序结束
```

从程序中可以看出, 在子程序的返回指令 RETURN 之前不必理会页选位 PCLATH <3>, 这是因为在发生跳转时, 程序计数器 PC 的值先被压入堆栈中保留, 然后才用来自 PCLATH 的页选位和来自 CALL 指令的 11 位地址码填充。在子程序执

行完毕返回时, 从堆栈中弹出 PC 的原值, 从而可以使程序返回到原先的页面中。

人们在着手编写程序时, 往往不考虑指令在程序存储器中的实际存放位置, 也不会注意到哪个 CALL 或 GOTO 指令会引起跨页跳转。好在这些也用不着人们去操心, 在用汇编器 MPASM 对源程序进行汇编时, 会自动发出警示信息。这时可根据该信息的提示对源程序稍作修改, 即作相应的页面预置处理, 然后再重新进行汇编即可。

实际上, 并不是每次编程时都会遇到跨页跳转问题。只有当程序跨页存放时, 才有可能遇到。比如程序较长并且超过 2K 时, 必然会跨页存放。

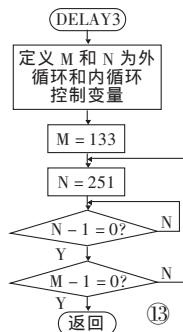
延时程序

在程序运行时, 为了某种原因, 经常要用某种方法来延迟程序的执行。这程序执行的方法有两种: 一是利用片内的硬件资源——可编程定时器; 二是采用软件手段——插入一段延时程序。本文仅介绍后面一种方法。如果延迟时间较短, 则可以连续插入几条空操作指令 NOP; 如果延迟时间较长, 则可以插入一段 (单一循环或者多重循环) 的循环结构延时程序。

在编写延时程序之前, 必须了解 PIC 单片机指令系统中的每一条指令的执行时间 (即指令周期 T_{INS})。在 35 条指令构成的指令系统中, 5 条实现无条件跳转的 (必然引起程序执行顺序发生改变的) 指令 (即 GOTO, CALL, RETURN, RETLW, RETFIE) 占用 2 个指令周期; 而有可能引起程序执行顺序发生改变的 4 条条件跳转指令 (即 DECFSZ, INCFSZ, BTFSZ, BTFSF) 的执行时间随着条件而定, 当因条件为真而发生跳转时需要占用 2 个指令周期, 当因条件为假而不发生跳转时仅占用 1 个指令周期; 除此之外的所有指令, 每一条仅占用一个指令周期。另外, 还应搞清单片机时基振荡器外接晶振的频率 f_{osc} , 以便确定时钟周期 ($T_{osc} = 1/f_{osc}$)、指令周期 ($T_{INS} = 4T_{osc}$) 以及程序执行时间的累计。

【例 6】采用不同方法编制三个延时子程序, 要求延时时间分别为 0.3ms、0.8ms 及 100ms。

对于这种程序, 首先定义两个循环控制变量 N 和 M, 用于保存决定延时长短的时间常数。为便于计算, 每条指令后面的注释部分中都给出了指令周期数。图 13 是延时 100ms 子程序 (DELAY3) 的流程图。



```
程序清单      (指令周期数) 注释
N EQU 20H
M EQU 21H
;{0.3ms 延时子程序}
DELAY1 MOVLW X      ;循环变量初始值 X (待定) 经 W 转送 N
MOVWF N
```



```

LOOP    DECFSZ  N,1    ;N-1 送 N 并判断结果 = 0 ?是 !跳出循环
        GOTO    LOOP  ;否! 循环回去
        RETURN      返回调用程序
;-----
;{0.8ms 延时子程序}
DELAY2  MOVLW   D'100' ;循环变量初始值 100 经 W 转送 N
        MOVWF   N      ;
LOOP    NOP      ;填充空操作
        NOP      ;
        NOP      ;
        NOP      ;
        DECFSZ  N,1    ;N-1 送 N 并判断结果 = 0 ?是 !跳出循环
        GOTO    LOOP  ;否! 循环回去
        RETURN      返回调用程序
;-----
;{100ms 延时子程序}

```

```

DELAY3  MOVLW   D'133' ;外循环变量初始值经 W 转送 M
        MOVWF   M      ;
LOOP1   MOVLW   D'251' ;内循环变量初始值经 W 转送 N
        MOVWF   N      ;
LOOP2   DECFSZ  N,1    ;N-1 = 0 ? 是! 跳出内层循环
        GOTO    LOOP2 ;否! 循环回去
        DECFSZ  M,1    ;M-1 = 0 ? 是! 跳出循环
        GOTO    LOOP1 ;否! 循环回去
        RETURN      返回调用程序
;-----
;{100ms 延时子程序}

```

```

DELAY3  MOVLW   D'133' ;外循环变量初始值经 W 转送 M
        MOVWF   M      ;
LOOP1   MOVLW   D'251' ;内循环变量初始值经 W 转送 N
        MOVWF   N      ;
LOOP2   DECFSZ  N,1    ;N-1 = 0 ? 是! 跳出内层循环
        GOTO    LOOP2 ;否! 循环回去
        DECFSZ  M,1    ;M-1 = 0 ? 是! 跳出循环
        GOTO    LOOP1 ;否! 循环回去
        RETURN      返回调用程序
;-----
;{100ms 延时子程序}

```

执行上述延时子程序 DELAY1 所需指令周期个数 = $1 + [1 + 2] \times (X - 1) + 2 + 2$ 。式中的“1+1”对应两条 MOV 指令，“1+2”对应 DECFSZ (非跳转) 和 GOTO 指令，“X”是循环变量递减的次数，“X-1”是循环次数，由于 DECFSZ 指令的执行过程是先递减后判断再跳转，所以循环的次数比递减的次数小 1；接下来的“2”对应 DECFSZ (成功跳转) 指令；最后的“2”对应 RETURN 指令。当时钟晶振选用 4MHz 时，每个指令周期 T_{INS} 为 $1\mu s$ ($T_{INS} = 4T_{OSC} = 4/f_{OSC}$)。在上面的计算式中，当时间常数 $X = 1$ 时，延时 = $1 + 1 + 2 + 2 = 6T_{INS} = 6\mu s$ ；当 $X = 99$ 时，延时 = $1 + 1 + (1 + 2) \times (99 - 1) + 2 + 2 = 300T_{INS} = 300\mu s = 0.3ms$ 。

在延时子程序 DELAY1 中，由于保存 X 的是一个 RAM 单元，X 的取值范围只能为 0~255，这就使得最大延迟时长受到限制。可以用在循环体内填充 NOP 指令的方法来加大延时，从而构成延时子程序 DELAY2。它的延时 = $1 + 1 + 5 + (1 + 2 + 5) \times (100 - 1) + 2 + 2 = 803T_{INS} = 803\mu s \approx 0.8ms$ 。

如果希望既要加大延迟时间又要程序尽量短小，则最好采用多重循环程序结构。延时子程序 DELAY3 就是一个二重循环结构，其延时 = $2 + [2 + (1 + 2)(251 - 1) + 2 + 1 + 2](133 - 1) + 2 + 2 = 99930 = 99.930ms \approx 100ms$ 。

在利用上述各子程序实现延时，若将调用它的 CALL 指令执行时间也考虑在内，则延迟时间又多了 2 个指令周期。

查表程序

在单片机的开发应用中，经常用查表程序来实现代码转换、索引、翻译等。下面就以 LED 数码管显示驱动程序设计作为讲解的例子。LED 数码管内部包含 8 只发光二极管，其中 7 只发光二极管构成字型笔段，1 只发光二极管构成小数点。根据各二极管公共端连接方式的不同，又有共阴极和共阳极之分，如图 14 所示。当某个发光二极管的阳极为高电平、阴极为低电平时该段点亮。驱动 LED 点亮的笔段码和 LED 所显字符

之间的关系如表 2 所示。

PIC 单片机的查表程序可以利用其“子程序带值返回指令 RETLW”来实现。我们

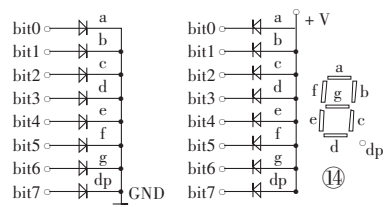
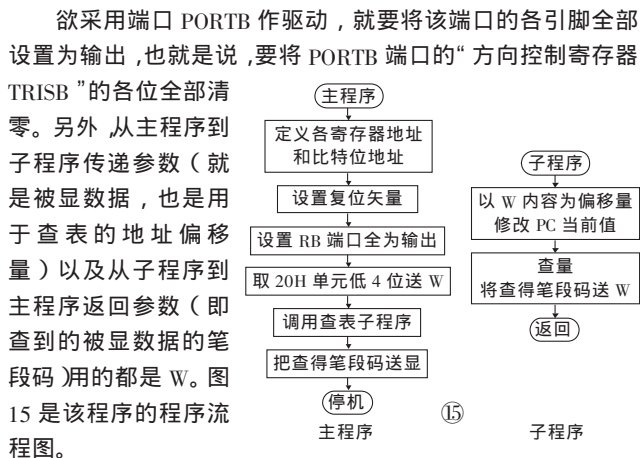


表 2

显示字符	共阴极笔段码	共阳极笔段码	显示字符	共阴极笔段码	共阳极笔段码
0	3FH	C0H	A	77H	88H
1	06H	F9H	B	7CH	83H
2	5BH	A4H	C	39H	C6H
3	4FH	B0H	D	5EH	A1H
4	66H	99H	E	79H	86H
5	6DH	92H	F	71H	8EH
6	7DH	82H	P	73H	8CH
7	07H	F8H	U	3EH	C1H
8	7FH	80H	全熄	00H	FFH
9	6FH	90H	全亮	FFH	00H

可以采用若干条携带着笔段码的 RETLW 指令按顺序排列在一起，来构成一张一维数据表。然后以表头为参照，以被显数字为索引值，到表中查找对应被显数字的笔段码。具体方法是，采用带有入口参数和出口参数的子程序结构，用数据表来构成子程序的主体部分。在子程序的开头安放一条修改程序计数器 PC 值的指令，来实现子程序内部的跳转。查表时，在主程序中先把被显数据（索引值）作为笔段码在数据表中的地址偏移量存入 W，以便向子程序传递参数，接着调用子程序。子程序的第一条指令将 W 中的地址偏移量取出并与程序计数器 PC 的当前值叠加，则程序就会跳到携带着所需笔段码的 RETLW 指令处。由该指令将笔段码装入 W 中，以便向主程序传递参数，然后返回主程序。

[例 7] 假设用 8 位端口 PORTB 作为一只“共阴极”LED 数码管的驱动端口。编一段程序，要求把寄存器单元 20H 中的数据的低 4 位送到 LED 显示。



```

CPL    EQU    02H    ;定义寄存器 PCL 的地址为 02H
STATUS EQU    03H    ;定义寄存器 STATUS 的地址为 03H
RP0    EQU    06H    ;定义 RP0 比特的位地址为 06H
PORTB  EQU    06H    ;定义寄存器 PORTB 的地址为 06H
TRISB  EQU    86H    ;定义寄存器 TRISB 的地址为 86H

        ORG    0000H
        GOTO    MAIN  ;设置复位矢量
        ORG    0005H
        ;设置主程序起始地址

MAIN    BSF    STATUS, RP0 ;选择体 1 为当前体
        CLRF   TRISB      ;定义 PORTB 端口各脚全部为输出
        BCF    STATUS, RP0 ;恢复体 0 为当前体

```

```

MOVWF 20H, 0 ;把 20H 单元的数据送 W
ANDLW 0FH ;屏蔽掉高 4 位后作为查表地址偏移量
CALL CONVERT ;调用数码转换子程序
MOVWF PORTB ;送到 PORTB 端口显示
STOP GOTO STOP ;停机

```

;-----

```

CONVERT ;子程序名称

```

```

ADDWF PCL, 1 W 内容叠加到 PC 的低 8 位上

```

```

TABLE RETLW 3FH ; 0 的笔段码

```

```

RETLW 06H ; 1 的笔段码

```

```

RETLW 5BH ; 2 的笔段码

```

```

RETLW 4FH ; 3 的笔段码

```

```

RETLW 66H ; 4 的笔段码

```

```

RETLW 6DH ; 5 的笔段码

```

```

RETLW 7DH ; 6 的笔段码

```

```

RETLW 07H ; 7 的笔段码

```

```

RETLW 7FH ; 8 的笔段码

```

```

RETLW 6FH ; 9 的笔段码

```

```

RETLW 77H ; A 的笔段码

```

```

RETLW 7CH ; B 的笔段码

```

```

RETLW 39H ; C 的笔段码

```

```

RETLW 5EH ; D 的笔段码

```

```

RETLW 79H ; E 的笔段码

```

```

RETLW 71H ; F 的笔段码

```

```

END

```

数据表可以看成是由 16 条 RETLW 指令构成，表头为“TABLE”。当程序跳转到子程序时，便开始执行 ADDWF 指令，同时，程序计数器的当前值已经指向表头。在此基础上再叠加预存在 W 中的表内地址偏移量，假设该偏移量为 8，则叠加后的 PC 值指向“RETLW 7FH”指令（其中 7FH 即为数字 8 的显示笔段码）。使程序跳转到该指令并执行它，执行后便返回到主程序，同时将 7FH 装入 W 中。





单片机硬件、软件及其应用讲座(10)

· 李学海 王国联 ·

第六讲 软件集成开发环境 MPLAB (上)

Microchip 公司为 PIC 系列单片机配备了功能强大的软件集成开发环境 MPLAB, 该软件可以在 Microchip 公司的网站上下载, 也可以向 Microchip 公司的办事处免费索取载有该软件的光盘。有了 MPLAB, 您就能在自己的微机系统上对 PIC 系列单片机进行程序的创建、录入、编辑以及汇编, 甚至还能实现程序的模拟运行和动态调试 (Debug) 之类的虚拟实战演练, 而且调试可以采用连续运行、单步运行、自动单步运行、设置断点运行等多种运行方式。MPLAB 的功能非常丰富, 限于篇幅, 不可能对其所有功能全部介绍, 我们只想从简单实用角度出发, 以一个实例程序从创建到调试的完成过程为例, 向读者展示 MPLAB 的基本用法, 以起到抛砖引玉的作用。为了下文描述方便起见, 我们用带有“>”的单词或词组代表菜单命令, 如果其中的英文单词或词组在文中首次出现时用括号中的译文加以说明, 比如 Options> Development Mode(选项> 开发模式) 菜单命令; 以“钮”字代表屏幕上出现的图标按钮或者文字按钮, 以“键”字代表计算机 101 标准键盘上的按键。

软件集成开发环境 MPLAB 的组成

MPLAB 是一个集多种单片机应用开发工具软件于一体的功能完备的“软件包”。在此仅对其中的工程项目管理器、源程序编辑器、汇编器、软件模拟器及在线调试工具 ICD 的支持程序这 5 种工具软件做一简要介绍。

1. 工程项目管理器 (Project Manager) 工程项目管理器是 MPLAB 的核心部分, 用于创建和管理工程项目, 为开发人员提供和定制一个自动化程度高、操作简便的符号化调试的工作平台。“符号化”是指对于指令、指令地址、常数、变量、寄存器等, 在屏幕上均用表义性和可读性很强的符号来显示。

2. 源程序编辑器 (MPLAB Editor) 源程序编辑器是一个全屏幕文本编辑器, 用于创建和修改汇编语言源程序文件。源程序文件以纯文本格式保存, 其文件扩展名为“.asm”。

3. 汇编器 (MPASM Assembler) 它用于将汇编语言源程序文件(.asm) 汇编成机器语言的目标程序文件(.hex), 并负责查找语法错误和格式错误等一些(浅层次)简单错误。

4. 软件模拟器 (MPLAB -SIM Software Simulator) 软件模拟器是一种用来代替价格较贵的硬件仿真器的调试工

具, 它是一种非实时、非在线的纯软件的调试工具。借助于这个在微机系统上运行的工具软件, 我们可以不需要任何额外的附加硬件, 仅用软件的手段, 来模仿 PIC 系列单片机的指令的执行和 I/O 端口信号的输入/输出, 从而实现对用户自编单片机源程序的模拟运行、功能调试和(深层次)逻辑错误查找。因此可以说, 微芯片公司为学习和应用 PIC 系列单片机的人们提供了一种虚拟的实战环境。对于单片机初学者来说, 不用花钱, 仅利用 MPLAB 也可以边学边练; 而对于单片机开发者来说, 利用 MPLAB 可以缩短开发周期和降低开发成本。总之, 它是许多其他型号系列单片机很少配备的、性能价格比极高的一种程序调试工具。

不过, MPLAB -SIM 也存在一定的局限性: (1) 它还不能模拟 PIC16F87X 片内少数功能特殊的外围模块; (2) 它不能帮我们查找目标板上的电路错误(目标板指的是演示板或用户板。演示板是 Microchip 公司为了便于单片机新用户的学习, 也为了向用户展示其单片机的功能, 更是为了产品设计者前期的开发评估而设计生产的一种电路板成品, 也叫 DEMO 板; 用户板是单片机开发者为了某项电子产品而自己设计和焊装了除单片机之外的元器件的电路板); (3) 它执行速度慢, 因而只适合调试那些实时性要求不高的程序。

5. 在线调试工具 ICD 的支持程序 (MPLAB -ICD Debugger) 这是一种专门与 ICD 配合使用的支持程序。ICD 是 Microchip 公司专为 PIC16F87X 设计的一种廉价在线调试和在线编程(即烧录)工具套件, 我们在下一讲中将作专题介绍。

另外, MPLAB 还包含一些其他工具软件, 如 C 语言编译器、硬件在线仿真器的支持程序、目标程序烧录器的支持程序、运行于 DOS 操作系统下的汇编器等等。

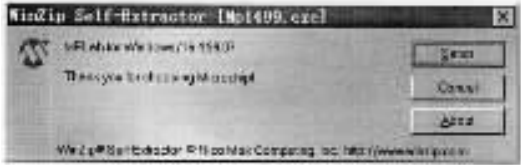
软件集成开发环境 MPLAB 的安装

为了使 MPLAB 能够在您的微机系统上顺利地安装和可靠地运行, 您的微机系统的最小配置为: CPU 为 Intel 486 或以上; 操作系统为 Windows 3. X 或 Windows 95/98; 显示器为 VGA (建议使用 SVGA); 内存容量应不小于 8MB (建议 32MB), 可利用的硬盘空间不小于 20MB; 配有鼠标。

下面介绍在 Window 98 操作系统之下, 版本为 V4. 99. 07 的

MPLAB 的安装过程。

首先从 Microchip 公司提供的光盘上的 MPLAB 目录下找到名为“mpl499.exe”的一个自解压文件,用鼠标左键双击该文件(下文中若不声明,点击或双击均指鼠标左键),会出现图 1 所示的对话框。按回车键或点击 Setup(建立)按钮,运行该文



①

件,就会自动解压并且自动启动安装过程,依照安装向导的指引选择操作,即可完成安装过程。为了简便起见,对于初学者可以不作任何选择而连续按动回车键,也可完成安装过程。不过这样会占用较大的硬盘空间(约 27MB)。如果您的电脑硬盘空间较小的话,可以采用“定制安装”的方法,只装当时必要的



②(a)



②(b)

程序文件,其他程序文件等用到时再安装。例如,在安装中我们可以暂时去掉如图 2(a) MPLAB 组件选择对话框及图 2(b) MPLAB 语言工具选择对话框中的 6 个程序,这大约可节省 8MB。如果再将(英文的)联机帮助文件 Help Files 也去掉的话,则剩下的只需要约 12MB 的硬盘空间。

安装完成后 MPLAB 会自动在 Windows98“开始”

按钮的“程序”组中,建立一个“Microchip MPLAB”程序组。至此 MPLAB 的系统文件已安装完毕。您可在硬盘驱动器 C:\Program Files\Mplab 目录下,建立一个新的子目录 Work 作为您的工作目录,存放您在学习和操练过程中产生的各种文件。

MPLAB 自带了卸载程序 C:\Program Files\Mplab\unwise32.exe,如果您日后不打算再使用 MPLAB,运行这一卸载程序即可将 MPLAB 彻底删除干净。

如果您想从 Microchip 公司的网站下载 MPLAB 的话,可登录到 <http://www.microchip.com>(英文网址)或 <http://www.microchip.com.cn>(中文网址)。当进入中文网站时,会出现如图 3(a)所示的公司主页。从中可以查看到一条“免费下载 MPLAB 最新试用版(v5.00.00)”的信息。点击它就会进入包含着“MPLAB(Disks 1 to 7)”信息的下一页,点击之又会进入包含着“MPL5000.zip”(8.96Mb)信息的下一页,再点击则会打开如图 3(b)所示的对话框,点击“确定”按钮,然后选定存盘路径即可下载一个 MPLAB 的压缩文件。

用 WINZIP 等压缩/解压工具软件,将下载的压缩文件展



③(a)



③(b)

开后会得到 MPVvvvvv.EXE、MPVvvvvv.W02、MPVvvvvv.W03、MPVvvvvv.W04、MPVvvvvv.W05、MPVvvvvv.W06、MPVvvvvv.W077 个文件。其中几位阿拉伯数字 vvvvv 是 MPLAB 的版本号(2000 年 7 月间上网时下载的新版本为 5.00.00,同时还有一个 4.12.00 版本供人们下载)。执行 MPVvvvvv.EXE,即可开始与上述自光盘安装相似的安装过程。由于 Microchip 公司对于 MPLAB 的版本更新比较频繁,因此网上下载的版本以及内容,可能会与在此介绍的版本之间存在一些微小的差异。

软件集成开发环境 MPLAB 的使用

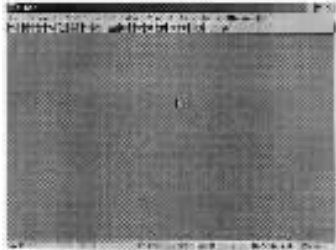
下面将引导您利用 MPLAB 建立一个简单的项目,并且进行一些基本的调试,以便您尽快地熟悉集成开发环境 MPLAB 典型的使用方法。

启动 MPLAB 在 Windows98 桌面上,依次选取开始>程序>Microchip MPLAB>MPLAB,便可启动和进入 MPLAB 集成开发环境。MPLAB 的桌面是一个标准的 Windows 应用程序窗体,它由标题栏、菜单栏、工具栏、工作区窗口和状态栏五部分组成,如图 4(a)所示。

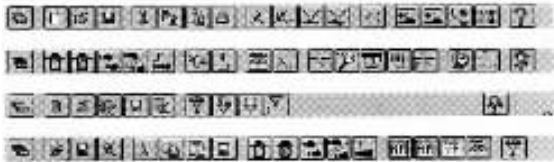
标题栏显示项目文件或源文件的名称及其所在的目录。

菜单栏(在此以 V4.99.07 版本的 MPLAB 为例)共有八个选项,从左到右依次为 File(文件)、Project(项目)、Edit(编辑)、Debug(调试)、Picstart Plus(一种烧录器的型号)、Options(选项)、Tools(工具)、Windows(窗口)和 Help(帮助)。每个菜单项下都有相应的子菜单,所有 MPLAB 的功能都可以通过操纵菜单命令来实现。

工具栏为用户提供了一种执行日常任务的便捷手段。它包含编辑工具栏、调试工具栏、项目工具栏和用户定义工具栏四个相互重叠的工具栏,如图 4(b)所示。在所有四个工具栏的最



④(a)



④(b)

左边有一个相同的图标按钮——工具栏切换按钮。根据当前的工作需要,点击该按钮可以在四个工具栏之间循环切换。点击工具栏上的图标按钮,可以快速实现对应的菜单命令功能。当把光标移至工具栏的任何一个图标按钮上时,在状态栏上就会自动显示该图标按钮的功能提示。

工作区窗口是 MPLAB 的主体,用于源程序的输入、编辑和



汇编,以及显示各种对话框和调试窗口的工作区域。

状态栏位于 MPLAB 桌面的最下端,其中包含 14 个字段的的信息,用于显示 MPLAB 当前的运行状态,从左到右排列的字段依次为:(1)显示 MPLAB 版本号(仅在 MPLAB 桌面刚打开时)或者光标在当前窗口中的位置——行号(Ln)/列号(Col);(2)当前窗口中所显内容的总行数;(3)当前窗口打开之后是否被改动过(是则显#号;否则显空白);(4)文件是可编辑的(显 WR)还是只读的(显 RO);(5)文本文件的行是否卷绕以及从第多少列开始卷绕;(6)输入模式是插入(显 INS)还是覆盖(显 OVR);(7)选中的待开发单片机型号;(8)程序计数器 PC 的当前值;(9)工作寄存器 W 的当前值;(10)状态寄存器的当前值(用大写字母表示逻辑 1,用小写字母表示逻辑 0);(11)全局断点的开关状态(Bk On 表示打开,Bk Off 表示关闭);(12)当前工作模式(Sim 表示软件模拟器,ICD 表示在线调试器,E0 表示只能编辑等);(13)选定的单片机的时钟频率;(14)当前所使用的工具栏(Edit——编辑,Proj——项目,Debug——调试,User——用户定义)。其中有的字段双击时,还能直接更改设置。

MPLAB 的设置 对 MPLAB 作某些设置的目的是让 MPLAB 按照用户的各项具体要求进行有针对性的程序调试。方法是:从 MPLAB 的菜单栏中选择菜单命令 Options> Development Mod(选项>开发模式),打开 Development Mod(开发模式)对话框(见图 5),其中包含 6 张卡片。我们只需对 Tools



⑤

介绍的单片机型号 PIC16F873 作为我们的学习和应用对象,对于其他选项卡则可维持原先的默认值(或称缺省值),最后点击 OK 按钮完成系统设置和复位。这时您会发现 MPLAB 桌面底部状态栏中的单片机型号变成了“PIC16F873”,开发模式变成了“SIM”,见图 4(a)。

创建简单的项目 当我们选择 MPLAB 的菜单命令 File> New 时,MPLAB 将会自动调用源程序编辑器(MPLAB Editor)



⑥



⑦

(工具)卡片中的选项进行设定:选择 MPLAB-SIM Simulator(软件模拟器)作为我们这一讲的程序调试工具;从 Processor(处理器)下拉列表中选择本讲座中

介绍的单片机型号 PIC16F873 作为我们的学习和应用对象,对于其他选项卡则可维持原先的默认值(或称缺省值),最后点击 OK 按钮完成系统设置和复位。这时您会发现 MPLAB 桌面底部状态栏中的单片机型号变成了“PIC16F873”,开发模式变成了“SIM”,见图 4(a)。

创建简单的项目 当我们选择 MPLAB 的菜单命令 File> New 时,MPLAB 将会自动调用源程序编辑器(MPLAB Editor)和工程项目管理器(Project Manager)。工作区内会出现一个图 6 所示的文本编辑窗口和一个 Create Project(建立项目)对话框。点击 Yes(确定)按钮,又会出现图 7 所示的 New Project(新项目)对话框。先在右边的 Directories(目录)窗口中双击选定我们先前创建的工

作目录 C:\Program Files\Mplab\Work,然后再在左边的 File Name(文件名)条框中输入一个自己喜欢的文件名,本讲座中选用“lxh.pjt”作为我们新建的项目文件名。点击 OK 按钮后,出现图 8 所示的 Edit Project(编辑项目)对话框,让我们开始编辑项目文件的内容。从该对话框中我们可以看到,系统自动将目标文件名定义为“lxh.hex”。点击选中 Project Files(项目文件)框内的 lxh.[hex],使其反白显示。这时 Node Properties...(节点属性)按钮由不可用状态(灰色)变为可用状态(黑色)。



⑧

点击 Node Properties...(节点属性)按钮,这时出现图 9 所示的 Node Properties(节点属性)对话框。在该对话框中完成关于汇编器工具的选择以及对于所选汇编器的参数设置。对初学者来说,可以不作任何改动,全部维持原先的默认设置即可。



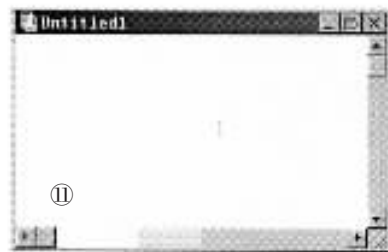
⑨

点击 Edit Project(编辑项目)对话框(图 8)的 Add Node(添加节点)按钮,可以看到图 10 所示的 Add Node(添加节点)对话框,并且工作目录与项目文件的目录保持一致。在文件名下输入汇编语言源文件名 lxh.asm,点击确定按钮,则又回到了 Edit Project(编辑项目)对话框,不过这时会看到源文件名 lxh.asm 出现在目标文件名 lxh.hex 的下面,并且缩进一点。在本讲中介绍的这种单文件项目,其项目文件名、目标文件名和源文件名必须保持一致。点击 OK 按钮完成项目文件的编辑,这时您会发现



⑩

MPLAB 桌面上,将图 11 所示的文本编辑窗口重新一览无余地显现出来,表明 MPLAB Editor 编辑器为创建源文件作好了准备。它的标题栏中显示 Untitled 表示源文件尚未命名。



⑪

编者附记 本刊与福州高齐公司合作,优惠供应 PIC16F87x 在线调试工具 ICD(350 元/套,含邮费),需购买的读者请汇款到高齐公司(具体地址见广告页,汇款时请注明“《电子世界》读者”)。

第六讲 软件集成开发环境 MPLAB（下）

• 李学海 王国联 •

新建和汇编一个简单的源文件 在图 11 中，若欲为即将输入的源文件预先命名的话，那就选用菜单命令 File>Save As（文件>保存为），随即会出现一个如图 12 所示的 Save File As（文件保存为）对话框。

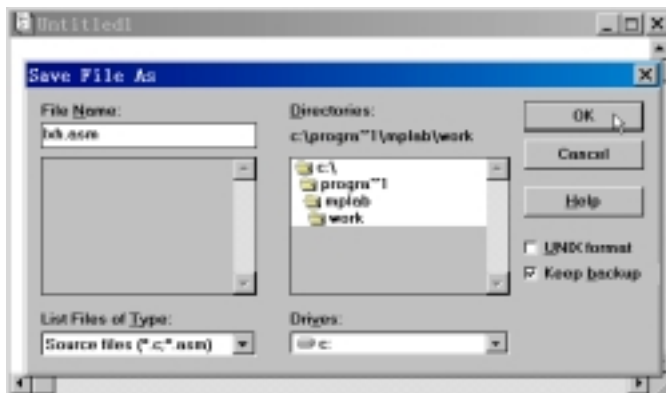


图 12

在图 12 所示的 Save File As（文件保存为）对话框中的 File Name（文件名）下面的条形框内输入 lxh.asm，并且将其存放在与项目文件相同的目录下（本例为 C:\Program Files\Mplab\Work）。这样文本编辑窗口的标题栏就会换成刚定义的文件名 lxh.asm 及其所在的目录路径，参见图 13(b)的标题栏。这时我们就可以在文本编辑窗口内输入源程序了。MPLAB Editor 编辑器不支持汉字，分号后面的注释部分可以用英语或者汉语拼音。由于汇编语言源文件[.asm]属于纯文本格式的文件，当然也可以选用 Windows98 附件中的“记事本”或“写字板”等其他可以编辑纯文本文件的软件作为源文件编辑器，这样输入汉语注释就不存在任何障碍了。不过，保存时文件扩展名要用“.asm”，并且必须保存为纯文本格式，存放路径应与项目文件一致，以便被 MPLAB 调用。

下面我们编制一段双重循环结构的简单程序，以作为编辑、汇编和调试的范例。该程序实现的功能是：定义一个寄存器变量（用作计数器）并赋给一个不等于 0 的任意值作为初值（在此假设选 10，也就是 0aH 或 0x0a），然后循环递减，直到结果为 0。这时再次给计数器赋初值，再循环递减，减到 0 时再次赋初值……，循环往复。该程序的流程图如图 13 所示。范例程序如下：

```
temp equ 0x20      ;定义 RAM 的 20H 单元为计数器变量 temp
f     equ 1        ;令 f 等于 1，用 f 指定目标寄存器
org   0x000        ;设置复位矢量
reset goto start   ;放置一条到主程序的跳转指令
org   0x008        ;指定主程序在程序存储器的起始地址
```

```

start    movlw 0x0a      ;经 W 预置一个非 0 值为 temp 的初值
movwf    temp           ;初始化计数器变量 temp
loop     decfsz temp, f   ;temp 递减，其结果决定是否跳一步
goto     loop           ;结果不为 0，则跳转到 loop 处
goto     stat           ;结果为 0，则跳转到 start 处
end       ;源程序结束

```

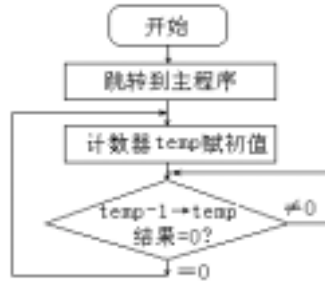


图 13

源程序输完之后（如图 14 所示，简练起见，在此没有输入注释部分），应该用菜单命令 **File>Save**（文件>保存）及时保存到 C:\Program Files\Mplab\Work\lxh.asm 路径及文件名下。对于较长的源程序，如果一次不能输入完毕而需要中途退出，也应及时保存，以便在下一次能继续前次的工作。

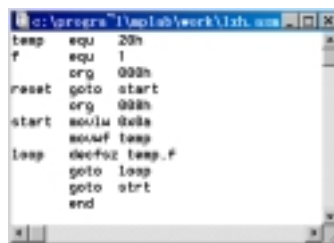


图 14

接下来需要启动汇编器 MPASM 对汇编语言源程序进行汇编。完成汇编过程的方法有多种，在此使用的方法是选择菜单命令 **Project>Build All**（项目>建立所有文件），MPLAB 将自动调用 MPASM 将项目文件 lxh.pjt 管理下的源文件 lxh.asm 汇编成（16 进制的）目标文件 lxh.hex。这时会出现一个 Build Results（建立结果）窗口，见图 15（a），其中最后一行告诉我们：建立失败（Build failed）。失败的原因在提示信息的第 4 行“Error[113]”中作了说明，是源文件的第 10 行中存在没有经过预先定义的符号。回过头来检查源文件，果然在第 10 行中将“start”错误地录入为“stat”。的确“stat”在此前没有被定义过。其实，这是笔者事先有意安排的一处错误，以便创造一次查错和纠错的的实践机会。



图 15（a）

纠正错误的方法有多种，最简便的方法是，双击图 15（a）中以“Error”打头的错误信


息提示行，即第 4 行。MPLAB 会再次自动调用编辑器，随即文本编辑窗口呈现在眼前，并且光标自动停留在存在错误的语句行上。你可将错误的“stat”纠正为“start”，然后采用上述方法重新进行汇编。这时会再次出现一个 Build Results（建立结果）窗口，见图 15（b）。其中最后一行告诉我们：建立完全成功（Build completed successfully）。点击右上角的叉号按钮，关闭 Build Results（建立结果）窗口。至此，我们就建好了一个可以用软件模拟器（MPLAB-SIM）进行调试的、完整的项目。对于建好的项目应及时地保存。选用菜单命令 Project>Save project（项目>保存项目）即可保存到项目文件 lxh.pjt 中。另外，对于作了修改的源程序文件也需要重新保存，选用菜单命令 File>Save 即可保存到 lxh.asm 中。




图 15（b）

程序调试 程序调试就是检验我们设计的程序是否能够正常运行，是否存在逻辑错误，算法（可以通俗地理解为，用计算机的思想解决实际问题的方法）设计是否合理，是否能够准确地控制各种硬件资源，是否产生正确的结果，是否能够实现预期的功能等等。

MPLAB 软件包中的软件模拟器（MPLAB-SIM）提供了多种运行程序的方式或调试程序的手段，比较常用的有连续运行、设置观察窗、设置观察寄存器变量、单步运行、自动单步运行和设置断点运行等几种。下面结合前面的范例程序分别介绍各种调试手段的使用方法。

1. **连续运行方式** 首先应使虚拟的“单片机”复位，方法是选择菜单命令

Debug>Run>Reset（调试>运行>复位）或者按一下键盘上的 F6 键，也可以点击工具栏上的图标按钮。这时光标会停留在主程序的第 1 条指令语句上，并使其反白显示，程序计数器 PC 回 0，如图 16 所示。当我们选择菜单命令 Debug>Run>Run（调试>运行>连续运行方式）或者按动 F9

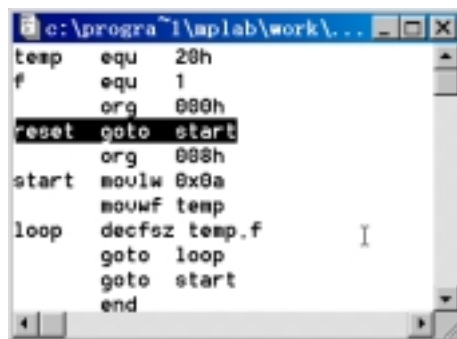




图 16

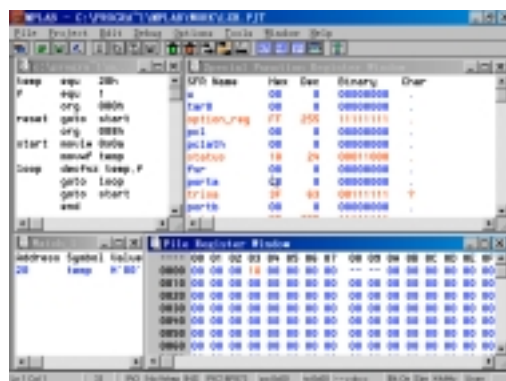
键，也可以点击工具栏上的绿灯按钮，均可令程序进入连续运行状态。当程序连续运行时，可以通过观察状态栏是否改变颜色来判断运行情况。如果状态栏变为黄色，则表明程序正在运

行。由于这段程序的结构是“无限循环”结构，自己不会停止。欲想中止程序运行，请按一下工具栏上的 红灯按钮（等效于 F5 键或菜单命令 **Debug>Run>Halt**）。随即，状态栏的颜色复原，表明程序停止运行，并随机地停留在程序的某一行上。在程序连续运行过程中，状态栏中的字段得不到及时更新，只有当程序停止运行之后，状态栏信息才被更新一次。可见，采用这种调试手段，不便于及时了解程序的运行状态，也不便于控制程序的运行过程。不过，对于调试顺序结构和分支结构的程序，比如算术运算程序，连续运行方式效率较高，可以很快将全部程序运行一遍，并能立刻得到程序的处理结果。而对于调试我们在此设计的这段循环结构的程序，连续运行方式就不是一种很有效的手段。因此，需要我们进一步探讨和选择其他的调试手段。

2. 设置观察窗口 大家知道 PIC16F87X 系列单片机内部的存储区域可分为程序存储器、堆栈、文件寄存器、特殊功能寄存器、EEPROM 数据存储器等多种。程序在运行过程中总要读出、写入或修改这些存储区域当中的内容。因此，我们可以通过观察各种存储区域中的内容随着程序的运行而发生的变化，来了解程序的运行情况，进而达到调试程序的目的。

当我们用 MPLAB 进行单片机程序开发与调试时，可以在 MPLAB 桌面上同时观察各种存储器和寄存器的内容，使开发者更直观地分析程序的逻辑功能和运行状况。究竟开设几个观察窗以及观察哪些存储区域为好，完全由开发者根据被调试程序的调试过程的需要和个人意愿而定。

本讲中调试的这个程序很简单，其中用到的寄存器除了 W、STATUS 和 PC 之外，还定义了一个寄存器变量“temp”，位于文件寄存器区域的 20H 单元，并且在程序运行过程中不停地改写该变量。我们可以根据这种情况来设置观察窗。除了桌面上已有的一个源程序窗口之外（见图 16），我们可以再增设文件寄存器观察窗、特殊功能寄存器观察窗和寄存器变量观察窗三个窗口，如图 17 所示。另外，在状态栏中也可以直接观察到 PC、W 和 STATUS 三个特殊功能寄存器的内容。



四种格式同时显示；(3) 打开观察寄存器变量窗口和选定被观察寄存器变量。选择菜单命令 **Window>Watch Windows>New Watch Window** (窗口>观察窗>新建观察窗)，同时出现一个 Watch_1 观察寄存器变量窗口和一个 Add Watch Symbol (添加观察符号) 对话框。从该对话框中的 Symbol (符号) 下面的空白条形框中输入打算观察的变量名“temp”，或者在(按字母顺序)罗列寄存器名和变量名的方形框中直接(滚动)选中“temp”，如图 18 所示，然后点击 Add (添加) 按钮，即可在“Watch_1”窗口增加一个待观察的寄存器变量(如图 19 所示)。如果有必要，用同样的方法还可以在“Watch_1”窗口中添加多个寄存器变量。完成在“Watch_1”窗口中变量的添加之后，点击 Add Watch Symbol (添加观察符号) 对话框的 Close (关闭) 按钮，关闭该对话框而留下寄存器变量观察窗。在该观察窗中可以查看到以兰色显示的变量的物理地址(十六进制)、符号名和数值。在完成各个观察窗的设置后，可以用菜单命令 **Windows>Tile Horizontal** (窗口>水平排列) 或 **Windows>Tile Vertical** (窗口>垂直排列) 或 **Windows>Cascade** (窗口>重叠) 等，大致调整各个观察窗在 MPLAB 桌面上的布局(实现粗调)；还可以通过任意拖拽和拉伸操作，进一步把各个观察窗调整到自己喜欢的位置和大小(实现细调)，以达到如图 17 所示的效果。

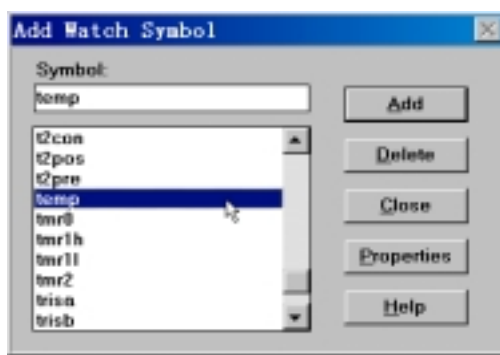


图 18

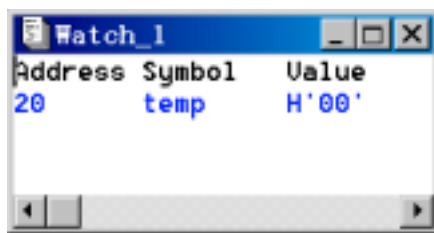






图 19

3. 单步运行方式 单步运行是一种控制程序运行过程的有效方法，而且能够及时观察到程序的运行状态。每输入一次控制命令，程序就会被执行一条指令，并且立刻更新 MPLAB 桌面上各个观察窗中的显示信息以及状态栏中的显示信息，也就是及时显示该指令的执行结果。具体操作方法如下：在图 17 所示的 MPLAB 桌面上，应保持源程序观察窗为当前窗口，也就是使其标题栏呈现兰色。首先应点击工具栏上的  图标按钮，使(虚拟的)“单片机”复位。然后选择菜单命令 **Debug>Run>Step** (调试>运行>单步运行方式) 或者按动 F7 键、或者点击工具栏上的  足迹按钮，均可令程序进入单步运行状态。一次次地点击  按钮的同

时，可以看到观察窗中出现变红的寄存器或变量，并且其中的值会不断地及时更新。MPLAB 将一条指令执行过程中所涉及的寄存器或变量，用红色标出以引人注目。可见，采用单步运行方式，不仅可以及时地了解程序的运行状态，还能很好地控制程序的运行过程。程序的单步运行方式与连续运行方式相比，两者具有很强的互补性。但是，单步运行方式效率较低，尤其是对于调试长程序和循环程序。

4. 自动单步运行方式 自动单步运行方式既象连续运行方式那样自动控制程序的运行过程，又象单步运行方式那样在每条指令执行过后刷新屏幕显示信息。程序不停，刷新不止，显示效果类似于播放动画片，因此又称为动画运行方式。

在图 17 所示的 MPLAB 桌面上，首先应保持源程序观察窗为当前窗口，并让“单片机”复位。然后选择菜单命令 **Debug>Run>Animate**（调试>运行>自动单步运行方式）或者按动一次 **Cntl+F9** 组合键，令程序进入自动单步运行状态。同时应注意观察寄存器变量“temp”的变化规律，是否符合我们的设计要求。若想让程序停止，点击一下工具栏红灯按钮即可。

5. 设置断点（Break）运行方式 在调试长程序的过程中，我们可以控制连续运行那些简单的或者已调通的程序片段，而控制单步运行那些复杂的或者待调的程序片段。这样就可以将连续运行方式和单步运行方式的优势结合起来交替使用。还有的时候，希望执行一个程序片段之后暂停下来，观察各寄存器变量的值，以便分析中间结果。这些均可以通过在程序的预定行上设置断点的方法来实现。可以说，设置断点是控制程序运行过程的另一种有效方法。一种最简便的设置断点的方法是，首先确保当前窗口为源程序观察窗，再用鼠标右键点击该窗口中计划设置断点的源程序语句行。这时会弹出一个如图 20 所示的按钮菜单，从中选择 **Break Point(s)** 断点选项即可。在源程序窗口中设有断点的语句行将以红色显示，以区别于其他语句行。用这种方法可以在同一程序中设置多个断点。消除某一语句行上所设断点的操作方法是，用右键点击该语句，在弹出的按钮菜单中再次选择 **Break Point(s)** 断点选项即可。

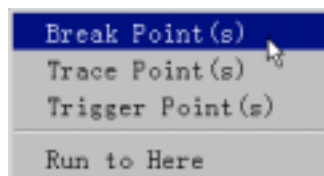




图 20

对已设置断点的程序进行调试时，一般采用连续运行方式。即先让“单片机”复位，再点击绿灯按钮使程序连续运行直到遇上第一个断点或者第一次遇上断点才会暂停；对于运行结果完成观察之后，再次点击绿灯按钮使程序连续运行直到遇上第二个断点或者第二次遇上断点又会暂停。等等。

以上分别介绍了 5 种调试手段的使用方法，实际运用时常常需要将这些调试手段混合使用，以达到最佳的调试效果。在调试过程中经常需要中途退出，这时请不要忘记保存项目文件，以便下次在打开 MPLAB 继续原来的调试工作时，能够原封不动地恢复自己设定的调试工作平台。从菜单栏中选择 **Project>Save Project**(项目>保存项目)即可实现对于 **lxh.pjt** 项目文件的保存。

第七讲 MPLAB-ICD 在线调试工具套件及其应用（上）

• 李学海 张秀芳 王国联 张先迅 •

MPLAB-ICD 是微芯公司对其 PIC 系列单片机中具有片内 FLASH 程序存储器的 PIC16F87X 所研制的一套廉价的学习和开发工具套件。MPLAB-ICD 可以用作实验阶段的评估和辅助调试。它既是一个编程器（即程序烧写器），又是一个实时在线调试器。用 MPLAB-ICD 可以代替在单片机应用项目开发过程中常用的硬件在线实时仿真器和程序烧写器，它利用了 PIC16F87X 片内集成的在线调试（In-Circuit Debugger）能力和微芯公司的在线串行编程技术（In-Circuit Serial Programming™）。MPLAB-ICD 工作于 MPLAB 集成开发环境软件包之下，其仿真头直接连接到目标电路板上，如同将一片 PIC16F87X 插入到目标板内一样去运行用户编制的程序。

MPLAB-ICD 的功能

1. MPLAB-ICD 的功能 MPLAB-ICD 在线调试工具能以实时或单步方式运行用户程序；具有断点设置、在线调试、在线烧写功能；用 MPLAB-ICD 可对源程序直接进行代码级的调试；MPLAB-ICD 以 RS-232 串行接口方式与微机系统相连，可以工作于 MPLAB 集成开发环境下。它的工作电压范围为 3.0～5.5V，可以从目标板上获取工作电源，工作频率范围为 32kHz 到 20MHz；

用户借助于 MPLAB-ICD 工具套件可以实现：软件调试——在自己设计的 PIC16F87X 的应用电路中实时运行和调试自己的源程序；硬件调试——用自己编制的程序来调试和检验自制目标板上的电路；程序固化——利用“在线串行编程技术”将自己设计的目标程序烧写到插在目标板上的 PIC16F87X 单片机（又称目标单片机）中。

2. MPLAB-ICD 的局限性 对于这套小巧廉价的、电路简洁的 MPLAB-ICD

开发工具套件，由于在实现在线调试和在线编程的过程中采用了 PIC16F87X 集成在片内的在线调试功能和微芯公司的在线串行编程协议，因此在用 MPLAB-ICD 仿真目标单片机时会存在一定的局限性。具体地说，MPLAB-ICD 工作过程中将会占用目标单片机的片内和引脚中的部分资源，故这部分资源用户就不能再使用了，这对于一般的项目开发没有太大的影响。与使用价格昂贵的专业级全功能在线实时仿真器相比，用 MPLAB-ICD 来调试和烧写 PIC16F87X 系列单片机，具有极高性能价格比，所以它非常适合初级开发者的学习和实践。MPLAB-ICD 所占用的目标单片机的部分资源如下：

系统复位脚/MCLR 被用作编程电压 VPP（约为 13V）输入脚；禁止采用低电压方式进行在线串行编程；RB6 和 RB7 脚保留为在线编程和在线调试时的通信专用；程序存储器中的首条指令（地址 0X0000）必须放置一条空操作指令 NOP；占用了（8 级堆栈中的）一级堆栈；SLEEP 指令在调试期间不能使用；在调试程序的过程中只能设置一个程序断点；六个通用文件寄存器单元保留给“调试监控程序”使用；程序存储器的最后 256 或 288 个单元被保留用来存放调试监控程序（根据单片机型号的不同所占的单元地址和单元个数也不同）。表 1 为 MPLAB-ICD 占用的 PIC16F87X 目标单片机的文件寄存器和程序存储器的地址。表中的列出的 PIC16F870/871/872 三种新型号的单片机，是微芯公司前不久刚推出的，功能和引脚兼容的简化版本。

表 1

处理器类型	被占用的文件寄存器	被占用的程序存储器
PIC16F870/871/872	0x70, 0x0BB-0x0BF	0x06E0-0x07FF
PIC16F873/874	0x70, 0x0EB-0x0F0	0x0EE0-0x0FFF
PIC16F876/877	0x70, 0x1EB-0x1EF	0x1F00-0x1FFF

MPLAB-ICD 工具套件的构成

MPLAB-ICD 套件中包括 MPLAB-ICD 仿真头、MPLAB-ICD 模块、MPLAB-ICD 演示板、RS-232 串行通信电缆、14 脚和 20 脚的连接插针各两条、20 厘米长的六芯电缆、包含所有 MPLAB 软件包和文档资料的光盘、《MPLAB-ICD 用户指南》

的中文翻译版本及直流电源适配器（这是高齐公司在原有套件基础上额外添加的一件，图中没有）等 9 种，如图 1 所示。下面对各主要部件的功能作一简要介绍。



MPLAB-ICD 模块 MPLAB-ICD 模块包括所有的调试、编程和控制逻辑。MPLAB-ICD 模块的上游是通过一条九芯串行电缆连接到微机系统的串行口（COM）上；MPLAB-ICD 模块的下游是用一条六芯扁平电缆连接到 MPLAB-ICD 仿真头上，或者直接连接到 MPLAB-ICD 演示板（或其它目标板）上。

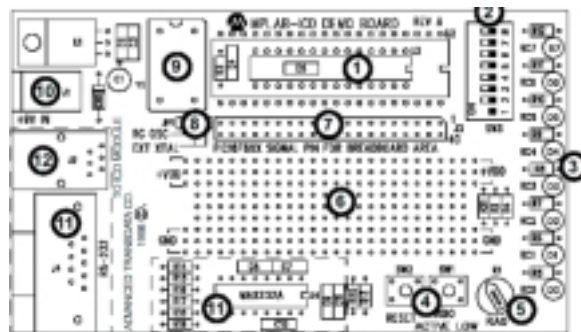
集成在 MPLAB 开发环境软件包内的 MPLAB-ICD 模块的支持程序，支持该模块与微机之间的串行通讯以及 MPLAB-ICD 模块与仿真头（或演示板）之间的串行通讯，以实现插在目标板上的 PIC16F87X 进行在线调试和在线编程。这一切都是在 MPLAB 集成开发环境的管理之下完成的。该模块的电源取自目标板（或演示板），所需要的电流最大为 70mA（不包括目标板上的器件所消耗的电流）。

MPLAB-ICD 仿真头 为方便用户使用，MPLAB-ICD 仿真头可以非常方便地插入到用户目标板（或演示板）上的 PIC16F87X 芯片插座上。MPLAB-ICD 模块和 MPLAB-ICD 仿真头之间有一条约 20 厘米长的 6 芯电缆相连。对于在线仿真，需要在仿真头上插入具有 40 引脚的 PIC16F87X 单片机（比如在该套件中选用的是 PIC16F87X 子系列中功能最强大的 PIC16F877，其功能也最齐全，覆盖了该子系列中其它几款单片机的所有功能）。然后将该仿真头插入到用户目标板或演示板上的 28 引脚或者 40 引脚的 PIC16F87X 双列直插（DIP）IC 插座上。MPLAB-ICD 仿真头的电源电压取自用户目标板，电压适应范围从 3~5.5V。

MPLAB-ICD 演示板 厂家提供的定型的演示板，用于在用户还没有制作自

己的目标板的情况下，预先进行演示和开发评估。它可以通过 MPLAB-ICD 仿真头以及 MPLAB-ICD 模块连接到微机系统，并与之进行通信。调试结果满意之后，可以将 PIC16F877 从仿真头上拔出来，直接插入到演示板上独立运行用户自己的程序。

演示板（见图 2）由下列部件构成：（1）一只 40 引脚和一只 28 引脚 IC 插座，用于插接仿真头或单片机。40 管脚插座适应 PIC16F871/874/877；28 管脚插座适应 PIC16F870/872/873/876；（2）一只 8 位双列插脚的拨动开关，可以独立地进行切换，以便实现将 8 路负载分别连接到单片机的端口 C 的 8 条管脚上，或者与这些管脚断开；（3）8 只红色的 LED 发光二级管充当 8 路负载和显示器，经过 8 位拨动开关连接到端口 C 上，用于显示八位二进制信息；（4）两只按钮开关，一只用于产生人工系统复位（/MCLR）信号，另一只连接到 RB0/INT 引脚作为外部开关量信号输入或者外部中断信号输入；（5）一个电位器连接到 RA0/AN0 引脚上，用来作为外部模拟信号的输入；（6）一小块焊孔密布的区域留给用户自由使用，随意焊装用户设计方案中的电路元件；（7）一个扩展插座位置，便于把 PIC16F87X 的 I/O 引脚信号引伸出来，比如将其连接到其他扩展板上去；（8）一个选择跳线，用于选择 RC 振荡方式（大约 2MHz）或者外部晶体振荡器振荡方式；（9）连接外部晶体振荡器插座的位置，该振荡器是一个有源的、独立的，双列直插式振荡器；（10）直流电源适配器的输入插座；（11）RS232 串行通信接口及其相应的配套电路（为可选部分）；（12）6 芯电缆连接插座，可以通过电缆直接连接到 MPLAB-ICD 模块（为可选部分）。



六芯电缆 两端带有插头的，约 20 厘米长的六芯电缆，将 MPLAB-ICD 模

块连接到 MPLAB-ICD 仿真头或者直接连接到演示板上，以便输送电源和建立双向通信。六芯电缆内部各芯的功能分配如表 2 所示，这些信号为 MPLAB-ICD 提供了调试和编程所需的所有功能。

表 2

六芯电缆各芯编号	信号
6	RB3
5	RB6
4	RB7
3	接地
2	+VDD
1	VPP

连接插针 两条 14 脚插针和两条 20 脚插针的用途是在仿真 40 脚封装的 PIC16F87X 单片机型号时，用两条 20 脚插针来将仿真头的 40 芯插座与演示板的 40 芯插座对应插接在一起；在仿真 28 脚封装的 PIC16F87X 单片机型号时，用两条 14 脚插针来将仿真头的 28 芯插座与演示板的 28 芯插座对应插接在一起。

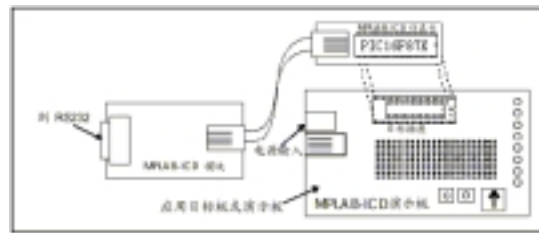
MPLAB 集成开发环境软件包 录制在一张光盘中的 MPLAB 集成开发环境软件包可以运行于 Win31~Win2000 等操作系统上。MPLAB-ICD 开发工具套件的在线调试和在线编程的功能，都必须在 MPLAB 集成开发环境的管理之下才能实现。

直流电源适配器 高齐公司选配的直流电源适配器，可以将 220V/50Hz 的交流市电转换为 9V/0.5A 的直流电源。在演示板上装有的三端稳压器 7805，可以将 9V 电压再进一步转换为 5V 电压。为演示板、MPLAB-ICD 仿真头和 MPLAB-ICD 模块提供直流电源。

MPLAB-ICD 在线调试工具的安装

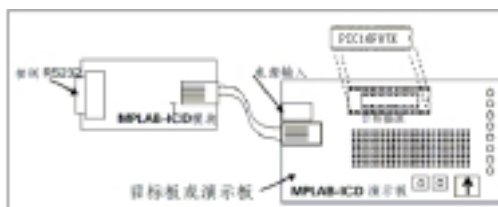
MPLAB 软件包应安装在具有奔腾以上的 CPU、16MB 的系统内存（推荐使用 32MB 系统内存）、45MB 可使用的磁盘空间、一个空余的串行通讯口、带有光盘驱动器并装有 Win3X~Win2000 的微机系统中。

硬件安装方法一 图 3 为 MPLAB-ICD 硬件连接示意图。连接步骤如下：



(1) 将一片 40 脚封装的单片机 PIC16F87X 作为目标单片机，插入到 MPLAB-ICD 仿真头的 40 管脚 DIP 插座里面（无论被仿真的单片机型号是 40 脚封装的还是 28 脚封装的，都要求在仿真头上插入 40 脚封装的型号）；(2) 用 20 厘米长的扁平 6 芯电缆把 MPLAB-ICD 模块和 MPLAB-ICD 仿真头连接起来；(3) 如果打算仿真的单片机是 40 脚封装的型号，就用两条 20 脚插针，将仿真头上的 40 引脚插座与演示板上的 40 脚 IC 插座对接在一起；如果打算仿真的单片机是 28 脚封装的型号，就用两条 14 脚插针，将仿真头上的 28 引脚插座与演示板上的 28 脚 IC 插座对接在一起；(4) 将 RS-232 串行口专用电缆连接到 PC 主机的串行口 COM 和 MPLAB-ICD 模块之间；(5) 打开主机电源；(6) 接通 MPLAB-ICD 演示板的电源（同时供给 MPLAB-ICD 模块电源），其上安装的一只红色 LED 便开始闪烁，表示该模块与微机之间的通信尚未建立起来。当该通信建立起来之后，LED 停止闪烁而维持常亮。

硬件安装方法二 假如在演示板的预留位置上安装一只与仿真头相同的 6 芯电缆插座，那么就可以方便地将 MPLAB-ICD 模块和演示板直接相连，而免用仿真头，参见图 4。其连接步骤基本与方法一相同，只是第 2 步用 20 厘米扁平电缆直接将 MPLAB-ICD 模块与 MPLAM-ICD 演示板相连，而不是连到仿真头。



软件安装 关于 MPLAB-ICD 的软件安装，被包含在 MPLAB 集成开发环境软件包的安装之内。原因是 MPLAB-ICD 的支持程序是 MPLAB 里的一个插件，MPLAB-ICD

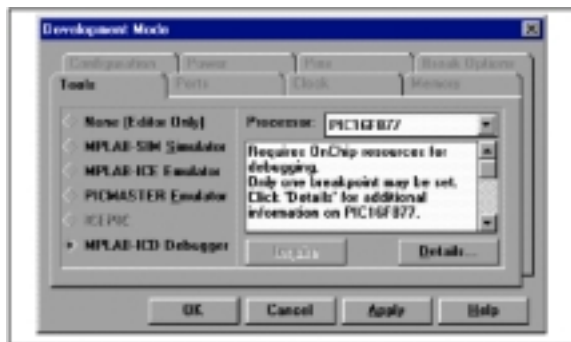
工具是一个运行于 MPLAB 集成开发环境之下的可插入工具，其安装过程详见上一讲。

第七讲 MPLAB-ICD 在线调试工具套件及其应用（下）

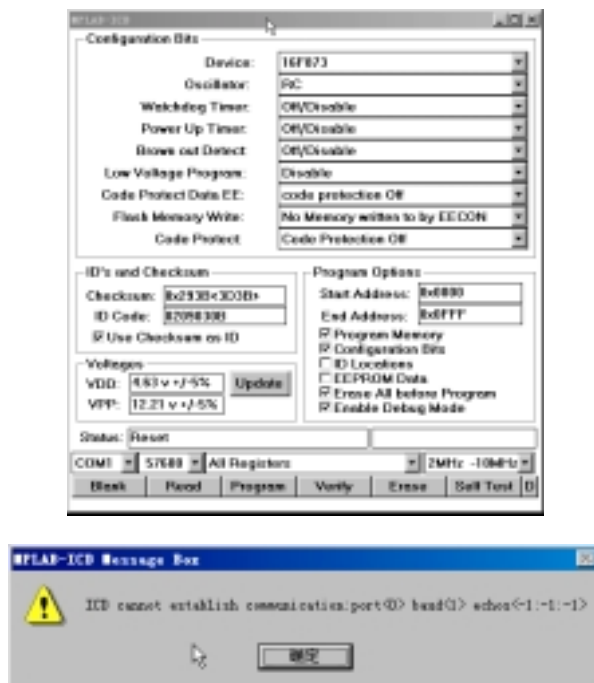
• 李学海 张秀芳 王国联 张先迅 •

MPLAB-ICD 在线调试工具的使用

微机与 MPLAB-ICD 建立通信 在确保 MPLAB-ICD 模块和微机系统之间用 RS-232 串行电缆连接可靠；确保演示板的电源已经接通后，MPLAB-ICD 模块会从演示板得到 5V 电源，LED 指示灯被点亮；在 Win98 桌面上，依次选取**开始>程序>Microchip MPLAB>MPLAB**命令，便可以启动和进入 MPLAB 的集成开发环境，并出现 MPLAB 的桌面（见上一讲的图 4a）；选择菜单命令 **Options>Development Mode**，打开 Development Mode（开发模式）对话框，然后点击“**Tools（工具）**”选项卡（见图 5）；



在该卡的 Processor（处理器）下拉菜单中选择一种您将要开发的 PIC16F87X 系列单片机的型号（在本讲座中以 PIC16F873 为例），并且选择“MPLAB-ICD Debugger”作为仿真工具的开发模式，点击“**OK（确认）**”按钮，则开发模式对话框关闭，系统自动开始查找 MPLAB-ICD 调试器并与之建立通信。若找到 MPLAB-ICD 调试器并与微机系统之间的通信良好，将有一个如图 6 所示的 MPLAB-ICD 窗口打开。假如找不到 MPLAB-ICD 调试器或者通信不畅，您会得到一条无法找到 MPLAB-ICD 的屏幕提示信息（如图 7 所示）。



MPLAB-ICD 的设置 当 MPLAB-ICD 被设置为启用状态时，图 6 所示的窗口总是打开着的(该窗口因 MPLAB 软件包版本的不同可能不同，在此以 MPLAB V4.99.07 版为例)。注意在微机与 MPLAB-ICD 调试器进行通信时不能关闭该窗口，因为若关闭该窗口，将会关掉微机与 MPLAB-ICD 调试器之间的通信。

在 MPLAB-ICD 工作窗口里，包含了在调试和编程过程中所需的 MPLAB-ICD 调试器支持程序的软件设置选项和对于目标单片机的硬件设置选项（在本讲中设置的结果如图 6 所示）。将包含着显示信息和选项设置等内容的整个窗口划分为配置位、用户识别码和校验和、电压值显示、编程选项、状态信息、通信选项及功能按钮 7 个区域。其各选项的含义和设置方法如下。

配置位（含单片机型号选择） 在该区域里不仅可以选定您即将使用单片机的型号，还允许您设置其内部的各种配置位（configuration bits）。这些配置位被存在一个地址为 2007H 的 14 比特的特殊程序存储器单元中，它不占用单片机程序存储器地址空间，用户不能用程序计数器 PC 访问，只能借助于“烧写器”对其进行编程。您想要更改选项设置，点击下拉箭头，会看到选项列表，然后从该列表选取合适的参数（见表 3）。

表 3

选项类别	选项设置
Device(单片机型号)	该条形窗口中显示的是在开发模式对话框里选择的单片机型号，假如想更改设置，任何打开的 MPLAB “项目” 都将关闭（处于纯编辑模式下例外），所有程序存储器、配置位、ID 位将会被清除。在本讲座中选择的是 PIC16F873。
Oscillator (振荡模式)	可选 RC、LP、XT 或者 HS 模式。在此选择 RC 模式，因为演示板上没装晶振，所以目标单片机只能工作于 RC 模式。
Watchdog Timer (看门狗定时器)	选择 “ON”（允许）或 “OFF”（禁止）该项功能。在调试状态下通常要将其设置为 “OFF”。
Power Up Timer (上电延时定时器)	选择 “ON”（允许）或 “OFF”（禁止）该项功能。在调试状态下通常要将其设置为 “OFF”。
Brown Out Detect (电源跌落检测)	选择 “ON”（允许）或 “OFF”（禁止）该项功能。在调试状态下必须要将其设置为 “OFF”。
Low Voltage Program (低电压烧写) 注：MPLAB-ICD 不支持该项功能	选择 “Enable”（使能）为低电压烧写方式，烧写电压从 RB3 脚送入；选择 “Disable”（禁止）为高电压烧写方式，片外电荷泵电路产生的 12V 烧写电压从 MCLR 脚送入，这时 RB3 可用作普通数字 I/O 脚。在 ICD 调试状态下，低电压烧写方式必须禁止。
Code Protect Data EE (程序保护 EEPROM 数据存储器)	选择 “ON”（允许）或 “OFF”（禁止）以设定是否保护 EEPROM 数据存储器。在调试状态下必须要将其设置为 “OFF”。
Flash Memory Write (FLASH 存储器烧写)	选择是否允许由 EECON 寄存器(该寄存器尚未讲到)控制烧写未保护的程序存储器。在调试状态下必须将其禁止。
Code Protect (程序保护)	选择是否保护程序存储器，以及欲保护的地址范围。一旦设置为保护，程序就不能再被读出。在调试状态下必须将其禁止。

ID（用户识别码）位与检验和 在 PIC16F87X 单片机内部还有一块 16 比特的特殊存储区域，让用户烧入自定义的 4 位 16 进制码，以作单片机的识别码（即 ID 码）。该码不论烧写什么内容，都不影响单片机的正常工作，只起识别作用。该存储区域实际是由 4 个 14 比特长的专用程序存储器单元构成的，其地址分布在 2000H~2003H，按厂家建议仅使用这 4 个单元的低 4 位，所以共有 16 位。与 2007H 单元相似，它不占用单片机程序存储器地址空间，用户不能用程序计数器 PC 访问，只能借助于“烧写器”对其进行编程（见表 4）。

表 4

选项类别	选项设置
Checksum (检验和)	显示程序代码的检验和。将程序存储器中的代码累加求和，并且只保留低 16 比特。
ID Code(用户识别码)	显示 ID 码，并且可更改 ID 码。
Use Checksum as ID (用检验和充当 ID 码)	若把检验和作为 ID 码，选中 <input checked="" type="checkbox"/> 号即可。

电压值显示 该区域向用户及时反映目标板的电源电压和对目标单片机的烧写电压（见表 5）。

表 5

选项类别	选项设置
VDD(芯片主电源)	显示单片机当前工作电压 VDD，通常为 5V 左右。
VPP(烧写电压)	显示当前 MPLAB-ICD 模块提供的编程电压 VPP 值(大约为 12V)。VPP 由来自目标板的 VDD 经 MPLAB-ICD 模块上的电荷泵电路升压后得到。
Update(更新按钮)	用于检查当前电压值。当目标板上的电压变动时，点击更新按钮，允许用户检查目标板上现时的 VDD 和 VPP 的电压值。

编程选项 (Program Options) 用户可以通过用鼠标点击选项前的方框来选中该选项，选中该选项时，会在方框中看到一个对勾符号。在程序调试阶段可以只选中 Program Memory (程序存储器)、Configuration Bits (配置位)、Erase All before Program (烧写前全部擦除) 和 Enable Debug Mode (允许调试模式) 四项。在程序调试完成后，应该清除“允许调试模式”选项，以避免将调试监控程序再烧入到单片机程序存储器里 (见表 6)。

表 6

选项类别	选项设置
Start Address, End Address (起始地址, 终了地址)	为烧写、读出、检验操作设置程序存储器的起始和终了地址。其默认的存储器地址范围是用户选定的单片机型号内部固有的程序存储器的最大范围。ICD 占用一些文件寄存器和程序存储空间，详细情况请参考“ICD 局限性”部分。
Program Memory (程序存储器)	选择是否烧写程序存储器。假如您只想烧写程序存储器，就只选“Program Memory”，将其他关于存储器类型的选择盒全部清除即可。
Configuration Bits (配置位)	烧写配置位。即把在 ICD 工作窗口上部用户选定的各个配置位，烧入 2007H 单元内。
ID Locations (用户识别码存储单元)	烧写 ID 码。在 ICD 工作窗口里的 ID 码与检验和区域设置的 ID 码，烧入 2000H~2003H 单元内。
EEPROM Data (EEPROM 数据)	对于 PIC16F87X 单片机，把 EEPROM 存储器窗口(该窗口在 MPLAB 桌面上可以打开和修改其中的数据)里给定的数据烧入 EEPROM 数据存储器内。
Erase All before Program (烧写前全部擦除)	选中此项，用户在点击功能按钮“ Program ”时，会先将所有程序存储器里的数据擦除，然后才开始烧写。
Enable Debug Mode (允许调试模式)	选中此项，用户在点击功能按钮“ Program ”时，会将“调试监控程序”预先写入单片机里，然后才允许调试操作。若只使用 MPLAB-ICD 的“烧写器”功能对单片机烧写，则可清除该选项，调试监控程序就不再下载到程序存储器里，MPLAB-ICD 也就没有了调试功能。

状态信息 (Status) 状态信息栏在调试过程中随时更新其内容，这对于用户很有帮助 (见表 7)。

表 7

条目	内容含义
Status	该状态栏显示被执行的 MPLAB-ICD 功能和状态信息，以及各种提示信息。

(状态信息)	
--------	--

通信选项 该区域中一些选项设置，直接影响微机系统与 MPLAB-ICD 之间的通信速率，甚至影响到是否能够进行正常的通信（见表 8）。

表

8

选项类别	选项设置
串行口	为 MPLAB-ICD 选择微机系统上的串行通讯口 COM1、COM2、COM3 或 COM4。
波特率	为 MPLAB-ICD 通讯选择串行口波特率。
上传内容选项 (以串行通信方式向微机系统上传的反映单片机运行状态的数据量越大，屏幕刷新就越慢)	上传的数据量选择：只上传最小数据量（FSR、W、Status、PCLATH 寄存器），速度最快；只上传特殊功能寄存器，速度较快；上传最小数据量和观察窗口内容，速度较慢；上传所有的寄存器内容，速度最慢。 如果您使用单步、自动单步或断点方式运行程序，可以减少所上传的数据量，以提高速度。第一种选择有大约 1 秒的延迟，最后一种选择有大约 2 秒的延迟。所选数据将在一个单步、一个断点或一个停止操作执行后被上传到微机系统，并刷新屏幕显示。
工作频率范围	在 MPLAB-ICD 工作频率范围的选项内作出选择：32 kHz ~ 500 kHz；500 kHz ~ 2 MHz；2 MHz ~ 10 MHz（在此选择了该项）；10 MHz ~ 20 MHz。

功能按钮 功能按钮包括存储器空白检查（Blank check）、存储器读出（Read）、存储器烧写（Program）、存储器检验（Verify）、存储器擦除（Erase）、自检测（Self Test）等。点击某一功能按钮可以令 MPLAB-ICD 执行相应的功能。如果您在“编程选项”里选择了地址范围，则指定的功能只适用于所选的存储器地址范围，但擦除功能例外（见表 9）。

表

9

选项类别	选项设置
Blank (空白检查)	检查单片机的存储器区域是否为空白。
Read (读出)	读出指定的存储器区域内的：程序存储器、配置位、ID 码和 EEPROM 中的内容。
Program (烧写)	烧写指定的存储器区域内的：程序存储器、配置位、ID 码和 EEPROM 的内容。如果“调试模式”被允许的话，还将把调试监控程序下载到程序存储器的底部。
Verify (检验)	检验指定的存储器区域内的：程序存储器、配置位、ID 码和 EEPROM 的内容。
Erase (擦除)	擦除在 PIC16F87X 芯片内部的所有内容，包括：程序存储器、配置位、ID 位。
Self Test (自检测)	点击该钮，对 MPLAB-ICD 执行自检测，约需 10 秒，在此期间状态信息显示“Please wait...”。如果正常就会显示如图 6.7 所示的信息。
D (诊断)	点击该按钮时，会出现一个诊断对话框（如图 8 所示）。



用 MPLAB-ICD 统调用户程序和用户电路

下面通过一个小程序，来演示利用 MPLAB-ICD 在线调试工具套件，进行项目软硬件的统调。在此演示板暂时充当用户电路。该程序实现的功能是，把端口 RC 的 8 条引脚全部设置为输出模式，依次从引脚 RC0 到 RC7 送出高电平，然后再依次从引脚 RC7 到 RC0 送出高电平，并且周而复始，从而使得与该端口 C 相连的 8 只发光二极管 LED 循环往复依次点亮。即依次循环点亮“LED0→LED1→LED2→……→LED7→全熄→LED7→LED6→……→LED0→全熄”。程序如下：

```

;*****
;文件名为 "1112.asm".
;*****
status    equ 3h          ;定义状态寄存器地址
portc     equ 7h          ;定义端口 C 的数据寄存器地址
trisc     equ 87h         ;定义端口 C 的方向控制寄存器地址
flag      equ 25h         ;定义一个控制左移/右移的标志寄存器
org 000h   ;定义程序存放区域的起始地址
nop        ;放置一条 ICD 必需的空操作指令
bsf        status,5       ;设置文件寄存器的体 1
movlw      00h            ;将端口 C 的方向控制码 00H 先送 W
movwf      trisc          ;再由 W 转移到方向控制寄存器
bcf        status,5       ;恢复到文件寄存器的体 0
movlw      01h            ;将 00000001B 先送 W
movwf      portc          ;再由 W 转移到数据寄存器
loop       btfs          status,0 ;测试进位/借位，是 1 则修改标志
          goto          loop1    ;是 0 则不修改标志
          incf          flag,1   ;FLAG 的 BIT0 作为标志位，并修改它
loop1      btfs          flag,0  ;判断标志位，是 1 则跳到循环左移
          goto          loop2    ;是 0 则跳到循环右移
          rlf          portc,0   ;循环左移端口 C 数据寄存器，结果送 W
          movwf         portc    ;将结果再送回端口 C 的数据寄存器
          goto          loop3    ;跳过下面两条指令
loop2      rrf          portc,0  ;循环右移端口 C 数据寄存器，结果送 W
          movwf         portc    ;将结果再送回端口 C 的数据寄存器
loop3      call          delay   ;调用延时子程序
          goto          loop     ;返回
;延时子程序

```


delay			;子程序名，也是子程序入口地址
	movlw	0ffh	;将外层循环参数值 FFH 经过 W
	movwf	20h	;送入用作外循环变量的 20H 单元
lp0	movlw	0ffh	;将内层循环参数值 FFH 经过
	movwf	21h	;送入用作内循环变量的 21H 单元
lp1	decfsz	21h, 1	;变量 21H 内容递减，若为 0 跳跃
	goto	lp1	;跳转到 lp1 处
	decfsz	20h, 1	;变量 20H 内容递减，若为 0 跳跃
	goto	lp0	;跳转到 lp0 处
	return		;返回主程序
	end		;源程序结束

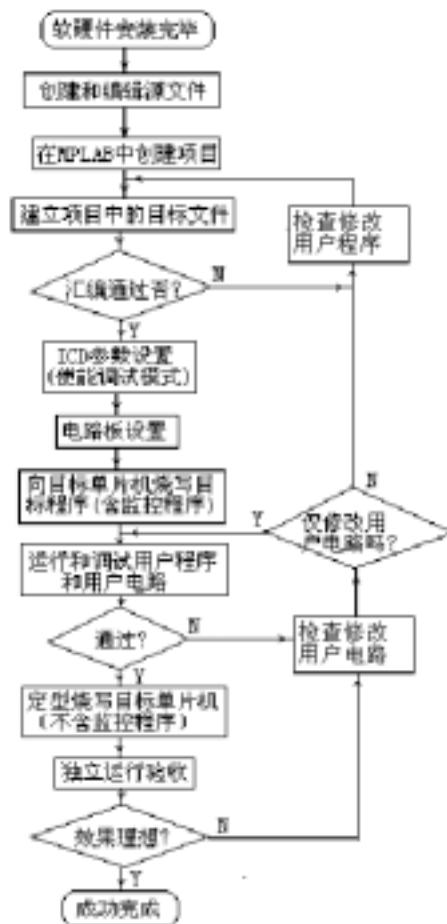



图 9 为软硬件调试流程图。以下对流程图中的各个步骤，进行一些必要的说明：（1）软硬件的安装：按照本讲介绍的方法操作即可；（2）创建源文件和编辑源文件：在此介绍另一种创建源文件的方法：用 Windows 附件中的“记事本”作为文本编辑器，可以方便地加入中文注释。不过应注意注释前面的分号“;”必须用西文半角输入，并且必须用“.asm”扩展名存储到事先建立的一个专用子目录下；（3）打开 MPLAB 集成开发环境；（4）创建项目：选用菜单命令

“File>New”或“Project>New Project”，在事先建立的一个专用子目录下创建一个新项目，将用“记事本”创建的源文件加入到该项目中来；（5）建立项目中的目标文件：选择菜单命令 Project>Build All(项目>建立所有文件)，MPLAB 将自动调用 MPASM，将项目文件管理下的源文件（.asm）汇编成（16 进制的）目标文件（.hex）；（6）ICD 参数设置：通过菜单命令 Project>Edit Project 或者 Option>Development Mode，将开发模式设置为“MPLAB ICD Debugger”，点击“OK”按钮，打开 ICD 的工作窗口；需要说明的是，选中“Enable Debug Mode”（使能调试模式）选项，在向目标单片机烧写机器码程序时，会将“调试监控程序”同时写入单片机内的指定程序存储器区域，然后才允许用 ICD 方式调试；（7）电路设置：将演示板上的 8 位拨动开关全部拨到“ON”的位置上，以使 8 只发光二极管与端口 C 接通；将用于选择频率的插接跳线插到“RC OSC”位置上，可免用外接独立的三端晶体振荡器（演示板上保留了该器件的安装位置）；（8）向目标单片机烧写目标程序：用户在点击功能按钮“**Program**”向目标单片机烧写机器码程序时，会等待一段时间；（9）运行和调试用户程序和用户电路：在将各项参数设置好之后，可以用 ICD 的工作窗口右上角的按钮把该窗口最小化（如图 10 所示）。利用上一讲中介绍的几种调试方法进行调试。当用自动单步方式调试时，建议临时禁止延时子程序发挥作用，具体方法，可在 CALL 指令前添加一个分号。为了学习目的，在调试过程中可以人为地加入一些软件漏洞（Bug）或硬件故障。加入软件漏洞的方法见上一讲；加入硬件故障的方法可以将 8 位拨动开关中的一只拨到“OFF”位置，来模仿单片机端口引脚的片内或片外故障等。在图 10 中可以看出，在此打开了源文件窗口、列表文件窗口、文件寄存器窗口和特殊功能寄存器窗口；在 MPLAB 的环境下可以观察中文注释，但是不能编辑和输入中文注释；（10）定型烧写目标单片机：当程序调试完成后，可以进行定型烧写，即将 ICD 窗口中的“Enable Debug Mode”（使能调试模式）选项消除，不再将调试监控程序写入单片机中；（11）独立运行验收：烧写过程完成后，即可将 ICD 模块和 ICD 仿真头（或演示板）之间的 6 芯电缆断开，让单片机在演示

板上独立运行，观察实际效果。

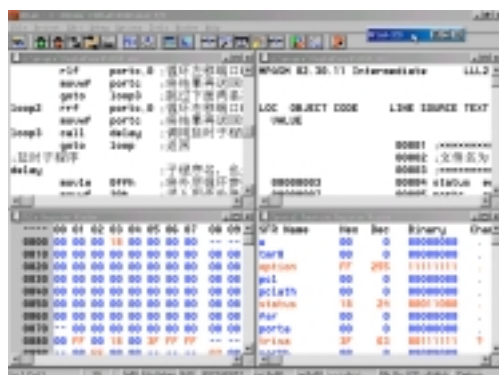


图 10

第八讲 输入/输出端口（上）

• 李学海 •

输入/输出端口（也可简称为 I/O 口）是单片机内部电路与外部世界交换信息的通道。输入端口负责从外界接收检测信号、键盘信号等各种开关量信号。输出端口负责向外界输送由内部电路产生的处理结果、显示信息、控制命令、驱动信号等。如果将单片机看作是一个为人服务的“奴仆”的话，那么单片机的 I/O 口也就是“主-仆对话”或称“人-机对话”的渠道。由此可见，输入/输出端口对于单片机来说是一种极其重要的外围模块，以至于对任何一个厂家生产的任何一种型号的单片来说，输入/输出端口模块都是必不可少的。

在 PIC16F87X 系列单片机中，28 脚封装的型号具有 3 个输入/输出端口，40 脚封装的型号具有 5 个输入/输出端口。由于 PIC16F87X 属于 8 位单片机，因此每个端口都由数量不超过 8 条的端口引脚（或称口线）构成。每个端口中的每条引脚都可以用软件的方式，由用户按需要单独编程，设定为输出引脚或者输入引脚。“端口引脚”与“端口”这两个概念之间的关系就是一种“个体”与“整体”的关系。

在本讲座中选定为样机的 PIC16F873 有三个端口，分别是 RA、RB 和 RC。RA 包含 6 个引脚，RB 和 RC 都包含 8 个引脚。其中有些 I/O 引脚和某些单片机内部的功能部件或其它外围模块的外接信号线进行了复用，也就是说，既可以作为普通 I/O 引脚，又可以作为某些功能部件或外围模块的外接引脚，这可以由用户以软件方式定义。比如端口 RC 中的引脚 RC4 既用作一般的输入/输出引脚，又可以作为“SPI 串行通信模式”的数据输入引脚，还可以作为“I²C 串行通信模式”的数据双向传送引脚。集 3 种功能于一脚，给用户的开发带来很大的灵活性。

与输入/输出端口相关的寄存器

在 PIC 系列单片机中，每个端口都至少对应着两个在 RAM 数据存储器中统一编址的寄存器，分别是数据寄存器和方向控制寄存器。也就是 PIC 单片机把端口都当作寄存器来访问（即读出或写入），这样有利于减少指令集中指令的类型和数量，给用户的记忆和编程带来了方便。PIC16F873 的三个端口 RA、RB 和 RC，分别与寄存器的对应关系如附表所示。从附表中可以看出：（1）端口 RA 对应的数据寄存器和方向控制寄存器分别是 PORTA

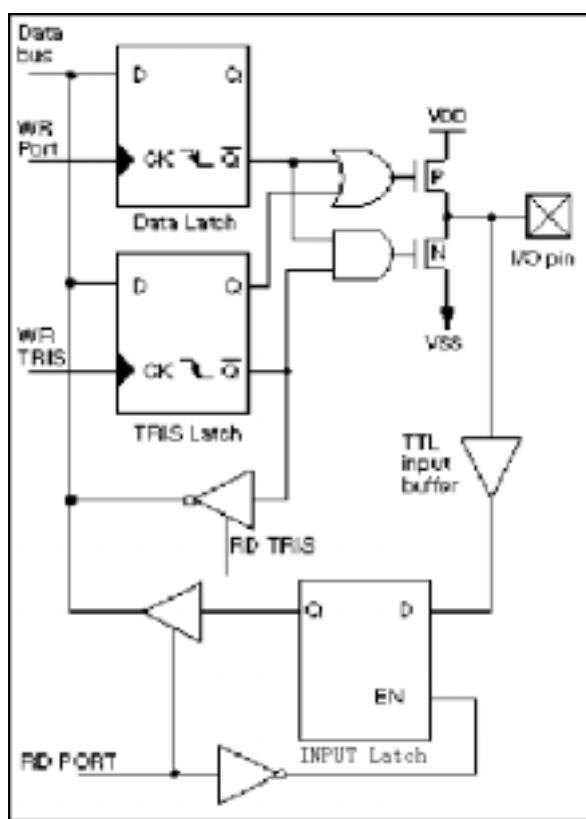
和 TRISA，地址分别是 05H 和 85H，分别位于体 0 和体 1 的相同位置上，并且都只用到了低 6 个比特，每个寄存器中阴影标出的两个比特没有被利用；（2）端口 RB 对应的数据寄存器和方向控制寄存器分别是 PORTB 和 TRISB。PORTB 的地址有两个，分别是 06H 和 106H，这就是说在体 0 和体 2 的相同位置上都能访问到它；TRISB 的地址也有两个，分别是 086H 和 186H，这就是说在体 1 和体 3 的相同位置上也都能访问到它；（3）端口 RC 对应的数据寄存器和方向控制寄存器分别是 PORTC 和 TRISC，地址分别是 07H 和 87H，分别位于体 0 和体 1 的相同位置上；（4）由于端口 RB 具备一项独有的“内部上拉”功能，因此与端口 RB 相关的寄存器中就多了一个“选项寄存器 POTION_REG”，不过 RB 端口仅仅用到了选项寄存器 POTION_REG 的一个比特位/RBPU，后面有说明。其余用阴影标出的 7 个比特位另有它用，留到以后的相关章节中用到时再作解释；（5）由于端口 RB 还具备另一项独有的“输入电平变化中断”功能，因此与端口 RB 相关的寄存器中就又多了一个“中断控制寄存器 INTCON”，不过 RB 端口仅仅用到了该寄存器的两个比特位 RBIF 和 RBIE，后面有说明。其余用阴影标出的 6 个比特位另有它用，留到以后的相关章节中用到时再作解释。

附表

端口名称	寄存器名称	寄存器地址	寄存器内容							
RA	PORTA	05H	—	—	RA5	RA4	RA3	RA2	RA1	RA0
	TRISA	85H	—	—	6 比特方向控制数据					
RB	PORTB	06H, 106H	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
	TRISB	86H, 186H	8 比特方向控制数据							
	POTION_REG	81H, 181H	/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
	INTCON	0BH, 8BH	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
RC	PORTC	07H	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
	TRISC	87H	8 比特方向控制数据							

输入/输出端口的内部结构和工作原理

PIC16F873 的三个端口 RA、RB 和 RC 之间不仅存在内部结构上的差异，而且同属于一个端口的各条引脚的内部结构也不尽相同，对此不再详述。我们用一个有代表性的“基本结构模型”，来向读者阐述一条 I/O 引脚的基本电路结构、硬件工作原理和软件编程方法。



图一

输入/输出端口的内部结构 图 1 所示的是可代表各个端口及其各个引脚共性的一个 I/O 引脚内部的基本结构模型。其中包括 3 只 D 触发器、两个受控三态门、一只反相器、一只 TTL 电平缓冲器、二输入端或门和与门各一只、能承受大电流的（20~25mA）构成互补推挽输出级（即驱动级）的 PMOS 管（P 沟道场效应管）和 NMOS 管（N 沟道场效应管）各一只。8 个图 1 所示的引脚结构图并列在一起，就构成了一个 8 位宽的 I/O 端口的内部结构图。

图 1 中右侧连接单片机的外接引脚；左侧连接到单片机的内部的数据总线（Data Bus）、写端口数据的控制线（WR Port）、写端口方向的控制线（WR TRIS）、读端口数据的控制线（RD PORT）和读端口方向的控制线（RD TRIS）。

数据输出的途径是，来自内部数据总线 Data Bus 的数据，送入数据锁存器 Data Latch 的 D 输入端，由 \bar{Q} 端送出反相后的数据，经二输入端“或”门和“与”门构成的“双门”及 PMOS 管和 NMOS 管构成的“对管”，数据再次被反相，最终送到引脚 I/O pin 上。如果为了更好地理解该“对管”的工作原理，用人们更熟悉的普通三极管来类比的话，上面的 PMOS 管相当于一只发射极接电源正极的 PNP 三极管，下面的 NMOS 管相当于一只发射极接电源负极的 NPN 三极管。将上面的 PNP 三极管的集电极和下面的 NPN 三极管的集电极连接在一起，并引出作为 I/O 引脚。

数据输入的途径是, 来自 I/O pin 脚上的数据, 经 TTL 电平缓冲器送到 D 触发器 INPUT Latch 的 D 输入端, 由其 Q 端送出, 再经受控三态门送到内部数据总线 Data Bus 上。

对于 I/O 端口的基本操作无外乎有四种: (1) 设置端口的输入/输出状态: 向端口的方向控制寄存器 TRIS Latch 写控制信息; (2) 经端口输出数据: 将打算输出的数据写入端口数据寄存器 Data Latch 中; (3) 经端口输入数据: 读取端口上的 (逻辑电平) 状态信息; (4) 检查端口的输入/输出状态: 从端口的方向控制寄存器读取控制信息。

以上这些操作都是通过对三只与每根 I/O 引脚关联的 D 触发器 (或称锁存器) Data Latch、TRIS Latch 和 INPUT Latch 的读/写操作来实现的。共同构成一个 8 位端口的引脚通常有 8 条, 与这 8 条引脚相关的 8 只 D 触发器 Data Latch 就构成了该端口的数据吞吐寄存器 (比如 PORTA、PORTB、PORTC, 可简称数据寄存器); 与这 8 条引脚相关的 8 只 D 触发器 TRIS Latch 就构成了该端口的 I/O 方向控制寄存器 (比如 TRISA、TRISB、TRISC, 可简称方向寄存器)。

图 1 中 3 只 D 触发器的作用分别是: (1) 对于端口的 I/O 方向控制寄存器 TRIS Latch: 写入 “1” 时, 对应的引脚被设置为 “输入”, 并且对外呈现高阻状态; 写入 “0” 时, 该引脚被设置为 “输出”, 引脚上的逻辑电平取决于数据寄存器 Data Latch 的内容。(2) 对于端口的数据寄存器 Data Latch: 经端口进行输出操作时, 将需要输出的数据写入该寄存器即可。(3) INPUT Latch 为端口状态锁存器。从端口引脚输入数据时, 该锁存器负责锁存端口引脚上的逻辑状态。

输入/输出端口的工作原理 对于一个端口引脚进行的 4 种基本操作分别说明如下:

(1) 写 I/O 方向控制寄存器 TRIS Latch 根据向方向寄存器中写入的内容不同, 又可以分为两种情况: 写入 “1” 则对应引脚被设置为 “输入”; 写入 “0” 则对应引脚被设置为 “输出”。为了便于大家记忆的更加牢固, 这里向读者介绍一个小诀窍: 与 “输入” 对应的英文单词和控制数据分别为 “Input” 和 “1”; 与 “输出” 对应的英文单词和控制数据分别为 “Output” 和 “0”。

将引脚设定为输入状态: 经 Data Bus 送来 “1”, 同时由 WR TRIS 送来脉冲下降沿, “1” 被锁入 TRIS Latch 中, 其 Q 端输出高电平封住 “或” 门, 其 \bar{Q} 端输出低电平封住 “与” 门。“或” 门输出的高电平使 PMOS 管截止, “与” 门输出的低电平使 NMOS 管截止, 数据输出的路径被阻断。因此, 该引脚 I/O pin 被设置为 “输入” 状态, 并且引脚对外呈现高阻状态; 将引脚设定为输出状态: 经 Data Bus 送来 “0”, 同时由 WR TRIS 送来脉冲下降

沿，“0”被锁入 TRIS Latch 中，其 Q 端输出低电平打开“或门”，其 \overline{Q} 端输出高电平打开“与”门。“或”门和“与”门的输出电平取决于 Data Latch 的 \overline{Q} 端电平，互补“对管”构成一级反相器，其导通与截止受控于“双门”的输出状态。因此，该引脚 I/O pin 被设置为“输出”状态。

(2) 经端口引脚输出数据 经端口引脚输出数据的前提是，该端口引脚必须预先已被设定为“输出”态。然后把欲输出的数据“X（为 0 或 1）”放到数据总线 Data Bus 上，接着由控制线 WR Port 送来脉冲下降沿作为触发信号，“X”被锁入 Data Latch 中。

当 X=0 时，Data Latch 的 \overline{Q} 端输出高电平，“或”门和“与”门同时输出高电平，使得 P 管截止，N 管导通。引脚 I/O pin 上出现低电平，即数据“0”被输出；当 X=1 时，Data Latch 的 \overline{Q} 端输出低电平，“或门”和“与门”同时输出低电平，使得 N 管截止，P 管导通。引脚 I/O pin 上出现高电平，即数据“1”被输出。

(3)从端口引脚输入数据 对于这种操作，根据方向控制寄存器的内容不同，又分为两种情况：

方向控制寄存器的内容为“1”时，读取的是引脚逻辑电平。“双门”被封住，“对管”处于截止状态。此时经 RD Port 送来“读脉冲”，一是加到 D 触发器 INPUT Latch 的 EN 端上，控制 D 触发器锁存此刻 I/O pin 脚经过输入缓冲器送来的逻辑电平；二是打开三态门将 D 触发器锁存的数据转移到内部数据总线 Data Bus 上。D 触发器的功能可以展宽输入信号的脉冲宽度。

方向控制寄存器的内容为“0”时，读取的是端口数据寄存器中锁存的数据：“双门”被放开，“对管”的输出状态取决于数据寄存器内容，进而使 I/O 脚上的逻辑电平也就取决于数据寄存器的内容。所以此时会将数据寄存器的内容读回到内部数据总线 Data Bus 上。

(4)读取 I/O 方向控制寄存器 TRIS Latch 由 RD TRIS 送来“读脉冲”，打开三态门将 TRIS Latch 锁存的数据转移到内部数据总线 Data Bus 上。例如，若其中锁存的内容为“1”，则其 \overline{Q} 端输出低电平，经过三态门反相后变成逻辑 1，送到数据总线上。

除了以上讲述的各个端口以及各条引脚的共性之外，还需要对于端口 RB 多作些介绍。端口 RB 具有弱上拉功能，即每条引脚内部都有一个可编程的弱上拉电路。这相当于在芯片内部每条引脚上，都有一个可由程序控制的开关连接一只高阻值电阻到 VDD 上。该开关闭合，则弱上拉功能被启用；该开关断开，则弱上拉功能被禁止。整个端口 RB 的 8 只

弱上拉开关，又都受控于一个总控制位 `POTCON_REG<7>`，即寄存器 `POTCON_REG` 的 `RBPU` (bit7) 位 (见附表)。因此，与端口 `RB` 相关的寄存器就又多了一个 `POTCON_REG`，但是在此仅用到了其中的一位。`RBPU=0` 时，设定弱上拉功能可以启用；`RBPU=1` 时，设置弱上拉功能被禁止。此外，还有一点需要说明：对于其中某一个具体引脚而言，仅当它被定义为输入方式，并且同时 `RBPU=0`，弱上拉功能才确实被启用；反之，如果该脚被定义为输出时，无论 `RBPU` 为何值，该脚上的弱上拉功能都自动被取消。对于那些需要外接上拉电阻的单片机应用场合，如果利用这项功能，就可以简化外部电路。

另外 `RB` 端口还有一个重要特性，即 `RB7~RB4` 具有输入“电平变化中断”功能，就是当 `RB7~RB4` 引脚上输入的电平发生变化（电平由低变高或者由高变低）时，可以引起 `CPU` 的中断（关于 `CPU` 的中断功能，将在以后的章节中作介绍）。但是，仅当 `I/O` 引脚被定义为输入时该中断功能才有效。因此，与端口 `RB` 相关的寄存器就又多了一个 `INTCON`，但是在此仅用到了其中的两位 `RBIF` 和 `RBIE`。其中 `RBIF` 为 `RB` 端口的输入电平变化中断标志位，`RBIF=1` 表示 4 条引脚 `RB<7:4>` 中有电平变化；`RBIF=0` 表示 4 条引脚 `RB<7:4>` 中无电平变化。其中 `RBIE` 为 `RB` 端口的输入电平变化中断屏蔽位，`RBIE=1` 表示允许 `RB` 端口向 `CPU` 发出中断请求；`RBIE=0` 表示禁止 `RB` 端口向 `CPU` 发出中断请求。

通常将端口 `RB` 的“电平变化中断”功能，与弱上拉功能配合使用，可以非常方便地构成一个键盘矩阵输入端口。特别适用于那些用于电池供电，而追求降低能耗的应用场合，可以平时让 `CPU` 处于“睡眠”状态（通过执行 `SLEEP` 指令）以降低能耗，需要时以键盘的按动来唤醒 `CPU`，使其进入正常工作状态。例如，手持遥控器、计算器、移动电话机、寻呼机等。采用这种设计方案，是一种理想的选择。

第八讲 输入/输出端口（下）

• 李学海 •

输入/输出端口应用举例

硬件电路规划 针对本讲中介绍的端口功能，并且以“ICD 在线调试器套件”中提供的演示板及其上面的现有硬件资源为基础，笔者设计了一个利用 I/O 端口实现数据输入和输出的应用实例。在该实例中将要用到的演示板上的部分硬件电路如图 2 和图 3 所示。

图 2 是端口 RC 外接的 8 条支路，由一只（8 路封装在一起的）拨动开关 SW3 分别控制各路接通还是断开。这 8 条支路既构成了端口 RC 的输出负载电路，又构成了端口 RC 的输出显示电路。其中 8 只电阻（R5~R12）起限流作用，保护端口引脚和发光二极管 LED（D0~D7）；8 只 LED 在相应引脚送出高电平时分别被点亮（当然是在开关闭合时）。图 3 是端口 RB 的外接电路，其中只应用了一只引脚 RB0，作为输入引脚。电阻 R4 为限流电阻，当 RB0 引脚误设为输出时起保护作用；电阻 R21 为上拉电阻，平时将 RB0 引脚电平拉高；微动开关 SW1 用来人工输入低电平脉冲信号。

程序设计思路 为该实例设计的程序取名为“PORT.asm”，其功能是把演示板当作一个按键计数器。刚刚接通电源时，8 只发光二极管都不亮，表示计数器的初始值为 0，即二进制数的 00000000B；当按下开关 SW1 时，计数器的值加 1，发光二极管 D0 点亮，表示二进制数的 00000001B，然后松开按钮；再次按下开关 SW1 时，计数器的值又加 1，发光二极管 D1 点亮，表示二进制数的 00000010B，然后再松开按钮；依次类推……。直到按动了 255 次按钮开关时，发光二极管 D7~D0 会全部点亮，其后的一次开关按动将使计数器回零。就这样循环往复。

在设计按钮或者键盘输入程序时，有一点特别需要提起注意，就是按钮在按下或者松开时均存在抖动现象。必须采用硬件或者软件措施加以处理，以避免产生误判而造成误动作或者重复输入。另外，对于输入信号在按钮断开期间和闭合期间可能存在的干扰窄脉冲信号，也最好一并做出统筹考虑。在本例中只对人们常用的（防干扰和去抖动）软件措施加以介绍。

将与按钮开关相连的端口引脚 RB0 上的信号波形描绘成如图 4 所示的电压波形。图中的 t1 表示按钮按下之前，RB0 引脚上呈现高电平，在此期间可能受到的干扰是很窄的负脉冲；t2 为按钮按下过程中出现的闭合抖动期；t3 表示按钮按下期间，RB0 引脚上会呈现

低电平，在此期间可能受到的干扰是很窄的正脉冲。这段时间的长短由操作人员的按键动作所确定，一般为零点几秒~几秒； t_4 为按钮松开过程中出现的断开抖动期； t_5 表示按钮松开之后，**RB0** 引脚上呈现高电平，在此期间可能受到的干扰也是很窄的负脉冲。图中的 **T** 表示用户程序对端口引脚逻辑状态进行周期性的扫描，所采用的扫描周期，大约为两个指令周期的时间（当晶振频率为 4MHz， $T=2\mu s$ ）； τ 为用户程序首次扫描到 **RB0** 引脚时电平变化，到再次扫描确认之间的延迟时间，约为 10ms。为了保证单片机对一次按键操作（完成按下和松开两个动作），作一次且仅仅作一次接收处理，就必须去除抖动。体现在程序设计上的思路，就是在扫描到 **RB0** 上的首次电平变化，延迟 τ （=10ms）待 **RB0** 上的状态稳定后，再次扫描确认，果真是按键动作（被按下或者松开），方认定为有效。否则，判为干扰脉冲，将其忽略。从而采用同一段程序，既有效地克服了抖动的影响，又避免了干扰脉冲带来的危害，可谓是一举两得。基于以上思想设计的程序流程图如图 5 所示，程序如下：

```

;*****
;文件名为 "PORT.asm".
;*****
status    equ 3h          ;定义状态寄存器地址
portb     equ 6h          ;定义端口 B 的数据寄存器地址
trisb     equ 86h         ;定义端口 B 的方向控制寄存器地址
portc     equ 7h          ;定义端口 C 的数据寄存器地址
trisc     equ 87h         ;定义端口 C 的方向控制寄存器地址
data1     equ 20h         ;定义一个延时变量寄存器
data2     equ 21h         ;定义另一个延时变量寄存器
n1        equ d'13'       ;定义一个外层循环延时常数
n2        equ 0ffh        ;定义一个内层循环延时常数
rp0       equ 5h          ;定义状态寄存器中的页选位 RP0
org 000h ;定义程序存放区域的起始地址
nop        ;放置一条 ICD 必需的空操作指令
bsf status, rp0;设置文件寄存器的体 1
movlw 00h   ;将端口 C 的方向控制码 00H 先送 W
movwf trisc ;再转到方向寄存器，将其设为输出
movlw 0ffh ;同理，将端口 B 设置为输入
movwf trisb ;
bcf status, rp0 ;恢复到文件寄存器的体 0
movlw 00h   ;将 00000000B 先送 W
movwf portc ;再由 W 转移到数据寄存器，并送 LED 显示
check      btfsc portb, 0 ;测试 SW1 按下否？是！跳过下条指令
            goto check    ;否！则循环检测
            call delay     ;调用延时子程序，消除接通抖动的影响
            btfsc portb, 0 ;再次测试 SW1 按下否？是！跳过下条指令
            goto check    ;否！则循环检测
            incf portc     ;端口 C 数据寄存器加 1，并送 LED 显示
check1     btfss portb, 0 ;测试 SW1 断开否？是！跳过下条指令
            goto check1   ;否！则循环检测
            call delay     ;调用延时子程序，消除断开抖动的影响
            btfss portb, 0 ;再次测试 SW1 断开否？是！跳过下条指令
            goto check1   ;否！则循环检测
            goto check     ;返回

```

```

;10ms 延时子程序
delay                                ;子程序名，也是子程序入口地址
    movlw    n1                      ;将外层循环参数值 n1 经过 W
    movwf    data1                  ;送入用作外循环变量的 data1 单元
lp0    movlw    n2                      ;将内层循环参数值 n2 经过
    movwf    data2                  ;送入用作内循环变量的 data2 单元
lp1    decfsz   data2, 1              ;变量 data2 内容递减，若为 0 跳跃
    goto     lp1                    ;跳转到 lp1 处
    decfsz   data1, 1              ;变量 data1 内容递减，若为 0 跳跃
    goto     lp0                    ;跳转到 lp0 处
    return                                ;返回主程序
end                                ;源程序结束

```

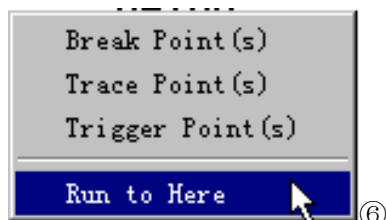
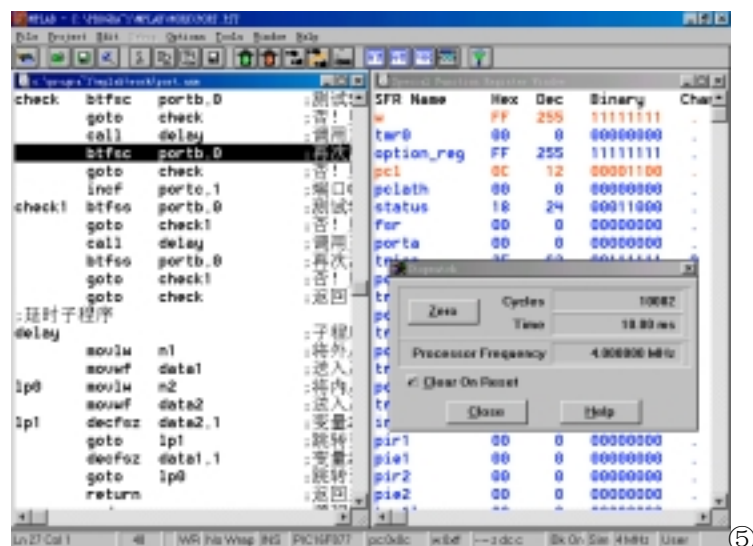
程序调试方法 在本例的程序调试过程中，可以采用软件模拟调试和硬件仿真调试两种手段相结合的方法。也就是，先用软件模拟器 **MPLAB-SIM** 将用户程序大致调通，然后再用 **MPLAB-ICD** 在线调试器对用户程序 and 用户电路一并进行实时运行统调，或者将用软件模拟器调通的用户程序直接烧入目标单片机，进行独立运行试验。这样做至少有两个好处，一是可以提高工作效率，二是可以减少目标单片机被擦写的次数，以延长其使用寿命。


用“软件模拟器 **MPLAB-SIM**”调试程序时，可以利用以前介绍的单步运行、中止运行、连续运行（也就是全速运行，但这不是实时运行，原因是实际执行指令的速度并不取决于目标板上的时钟晶振，而是取决于微机系统模拟单片机指令的快慢）、自动单步运行、设置断点运行等多种运行方式。

根据笔者在实践过程中积累的点滴经验，在这里给大家一点小提示：在采用“自动单步运行”方式（或称动画运行方式）调试带有延时子程序的程序时，为了提高效率，最好将延时子程序暂时屏蔽掉。具体实现的方法是，可以在调用延时子程序的指令（**call delay**）前暂时加上一个分号“;”，这样该语句就会被认为是注释行而被跳过。

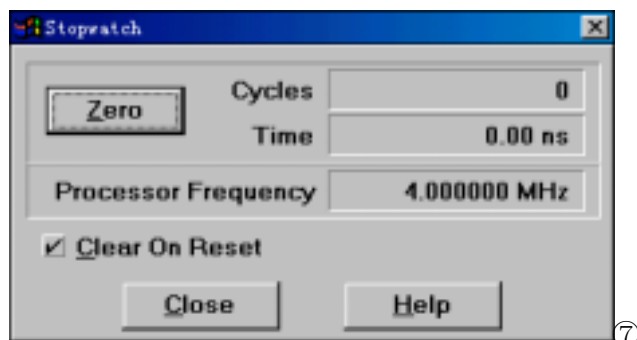
除了在前面的“软件集成开发环境 **MPLAB**”章节中已经介绍的几种调试手段之外，这里向大家再介绍几种调试手段或技巧。

“运行到此”方式 可以实现从当前程序计数器 **PC** 指定的指令行开始，执行到您指定的任意行上停止。在某些情况下，这种运行方式可以实现连续运行和断点设置运行的双重功效，但更加方便灵活好用。具体操作方法是，在图 5 所示的 **MPLAB** 桌面上（确定汇编语言源文件窗口为当前窗口的前提下），在指定行上点击鼠标右键，会弹出一个浮动的小型命令菜单（如图 6 所示）。选取其中的“运行到此（**Run to Here**）”命令项，程序开始从当前 **PC** 指定处连续运行，随后自动停止在该指令行上。



“单步跨越运行”方式 在没有遇到子程序调用指令的前提下，该方式与单步运行方式相同；而在遇到子程序调用指令时，会将子程序一口气执行完毕，然后停止在调用指令下面相邻的一条指令上。也就是用连续运行方式一步跨过了子程序。具体操作方法是，既可以选用菜单命令 **Debug>Run>Step Over**（调试>运行>单步跨越运行方式），也可以用快捷键 **“F8”**，还可以直接用鼠标左键点击工具栏中的带有红黄条的足迹图标按钮。对于那些带有子程序的程序，如果您只想集中精力调试主程序的话，利用这项功能会给您带来很大的方便。

“跑表观察窗” 在设计和编写延时子程序时，为了精确计算整个子程序的延迟时间，常常需要逐条分析每一条指令被执行时占用的指令周期数。在此结合本例中延时子程序的设计过程，介绍一种快捷方法——“跑表观察窗”法。



这里的跑表功能类似于田径场上赛跑计时用的跑表或称秒表。打开跑表观察窗的操作方法是，在 MPLAB 桌面上，选用菜单命令 Windows>Stopwatch...（窗口>跑表观察窗...），即可出现一个如图 7 所示的小型信息观察窗——跑表观察窗。该窗口可以记录一段程序被执行的过程，所占用的时间和指令周期数。跑表观察窗中包含 3 条显示信息、3 个功能按钮和一个可选项：（1）3 条显示信息分别为单片机设定的工作频率（Processor Frequency）、执行程序时占用的指令周期数（Cycles）和执行程序时占用的时间（Time）。（2）3 个功能按钮分别是“清零按钮（Zero）”用于将指令周期数和时间显示信息归零；“关闭按钮（Close）”用于关闭该观察窗；“帮助按钮（Help）”用于调出联机帮助信息。（3）可选项（Clear On Reset）一旦被选中，就会在每次复位单片机时，自动清除指令周期数和时间显示信息中的累计值。

结合本程序实例中延时子程序的参数确定方法，在 MPLAB 桌面上跑表功能的具体操作步骤为：（1）在如图 5 所示的 MPLAB 桌面上，首先令（虚拟的）单片机复位，此时会出现一个“反白显示行”并且停留在第一条指令（nop）上。反白显示行表明当前的程序计数器 PC 值，指向该行指令所在的程序存储器的单元地址，也表明该行是即将被执行的指令。（2）接着在汇编语言源文件窗口被确立为当前窗口的前提下，在第一条调用延时子程序的指令（call delay）上点击鼠标右键，会弹出一个小型浮动的命令菜单（如图 6 所示）。（3）选取其中的“运行到此（Run to Here）”命令项，程序开始被执行，此后反白显示行自动停留在那条调用指令上。这时跑表观察窗中有了时间显示，显示的是此前被执行过的 11 条指令所占用的时间和指令周期数。（4）这个显示值不是我们所关心的，点击“Zero”按钮将其清零。（5）再点击一次“单步跨越运行”按钮，“跑表”便开始累计记录程序运行的时间和指令周期数。此时 MPLAB 桌面上的状态栏变黄，表明程序正在被执行。一段时间过后，状态栏恢复原色，程序停止运行并且停留在与调用指令相邻的下一条指令上。跑表观察窗中所显示出的信息就是执行延时子程序所花费的时间和指令周期总数。（6）如果该时间值不符合设计者的要求，可以修改延时子程序中的内循环参数 n2 或者外循环参数 n1。（7）然后再用同样的方法运行程序，并观察延时子程序消耗的时间。反复这一过程，直到延迟时间满意为止。

经过几次参数选配试验，最后确定出 $n1=13=d'13'$ ， $n2=255=ffH$ 时，延时子程序的延迟时间为 10.00ms，占用的指令周期总数为 10002 个。

第九讲 定时器/计数器（上）

• 李学海 • E-mail:lixuehai@163.net

在许多世界著名的半导体制造公司开发的型号繁多的单片机芯片内部，定时器/计数器模块是一种基本上普遍配置的常用外围设备模块，只是配备的数量和规格不同而已。其中，规格的不同指的是：宽度的不同、是否附带着预分频器、是否附带着后分频器、是否同时附带着预分频器和后分频器、以及预分频器或后分频器的分频比的不同，等等。

定时器/计数器模块的基本用途

在单片机芯片内部配置的各种外围设备模块中，定时器/计数器模块是一种应用比较灵活的外设模块。那么，定时器/计数器模块究竟有什么用途呢？归纳起来大致可适用于以下三类：

（1）向外部电路“送出”一系列符合一定时序规范的方波信号。在一些单片机的应用项目中，有时要求单片机在其端口引脚上，向外部电路“送出”一系列符合一定时序规范的方波信号。例如，空调机中的变频控制，VCD、光盘驱动器、照相机、打印机、传真机中的步进电动机的驱动，电器设备的提示音的音调产生，PWM 脉冲宽度调制信号的形成等等。在对这些应用项目的单片机进行编程时，需要在程序的执行过程中，插入一定时长的延时。对此有两种方案可供选择，一是利用芯片内部现成的硬件资源——可编程定时器，来精确控制输出“事先预定”的时间间隔；二是采用软件手段——插入一段延时程序。关于软件手段延时，在“PIC 汇编语言程序设计基础”章节中已经作过介绍，其缺点是需要占用“机时”。在此仅对第一种方法中用到的硬件资源进行讲解。

（2）检测外部电路“送来”的一系列方波信号的脉宽、周期或频率，以便单片机接收外部电路的输入信号或通信信号。在一些单片机的应用项目中，经常要求单片机在其引脚上，检测外部电路“送来”的一系列方波信号的脉宽、周期或频率，以便单片机接收外部电路的输入信号或通信信号。例如，遥控电视机中的红外遥控信号的接收，速度里程表中的转速检测，超声波测距仪中发射波与反射波之间的时间间隔的精确测量等等。这类应用程序的编写，可用定时器来对“事先未知”的时间间隔进行精确计时。

（3）对“触发信号”进行准确地计数 在一些单片机应用项目中，需要单片机对其端口引脚上输入的由外部事件产生的“触发信号”进行准确地计数，依据计数结果来控制完成相应的动作。例如，在饮料的生产和包装车间里，传送带上的易拉罐在移动时，可以借助于红外线透射或者反射方式，获得触发信号并且送入单片机的相应引脚，由单片机内部的可编程计数器来对移过红外探头的易拉罐数量进行计数。每当计数器的累加值达到 24 时（在此假设每箱易拉罐包装 24 听），就控制相应装置完成封箱操作。

PIC 系列单片机中定时器/计数器 TMR0 模块的特性

微芯公司产生的 PIC 系列单片机各款产品片内全部配备有定时器/计数器模块，并且配备的数量不尽相同。早期研制的 PIC 单片机产品系列，例如 PIC12CXXX/CEXXX 系列、PIC16C5X/5XX 系列、PIC16C8X/F8X 系列中的全部产品、PIC16C62X/CE62X 系列中的部分产品，只配置了一个定时器/计数器模块。除了最早的 PIC16C5X 系列单片机中，把该模块叫做 RTCC 模块之外，在其余所有 PIC 单片机中都把该模块叫做 TMR0 模块。近期新研制的 PIC 单片机产品系列中，大都配置了多个定时器/计数器模块，例如 PIC17CXXX 系列和 PIC18CXXX 系列就都配置了 4 个定时器/计数器模块。

在本讲座中当作样板讲解的 PIC16F87X 系列单片机都配置了 3 个定时器/计数器模块，分别记为 TMR0、TMR1 和 TMR2。需要事先声明的是，TMR0、TMR1 和 TMR2 三个定时器/计数器模块，不仅电路结构上均不相同，而且设计的初衷也各有所异（TMR0 为 8 位宽，有一个可选的预分频器，用于通用目的；TMR1 为 16 位宽，附带一个可编程的预分频器，

还附带一个可选的低频时基振荡器，适合与 CCP（捕捉/比较/脉宽调制）模块配合使用来实现输入捕捉或输出比较功能；TMR2 为 8 位宽，同时附带一个可编程的预分频器和一个可编程的后分频器及一个周期寄存器和比较器，适合与 CCP 模块配合使用来实现 PWM 脉冲宽度调制信号的产生。三者的核心部分都是一个按递增方式，即累加方式工作的由时钟信号触发的循环计数器，都是从“0”开始计起，并且在累计到最大值（或规定值）后溢出时都会建立一个相应的溢出标志，对它们的编程方法基本相同。因此，本文选择一个最有代表性的，也是各款 PIC 单片机内部都配置了的定时器/计数器 TMR0 模块，作为讲解的模型。

定时器/计数器 TMR0 具有以下特性：（1）核心是一个 8 位宽的由时钟信号上升沿触发的循环累加计数寄存器 TMR0；（2）TMR0 也是一个在文件寄存器区域内统一编址的寄存器，地址为 01H 或 101H；（3）用户用软件方式可直接读出或写入计数器的内容；（4）具有一个可选用的 8 位可编程预分频器；（5）用于累加计数的信号源可选择内部或外部时钟信号源；（6）当使用外部触发信号作为时钟信号源时可由程序定义上升或下降沿触发有效；（7）具有溢出中断功能。

定时器/计数器 TMR0 模块的电路结构和工作原理

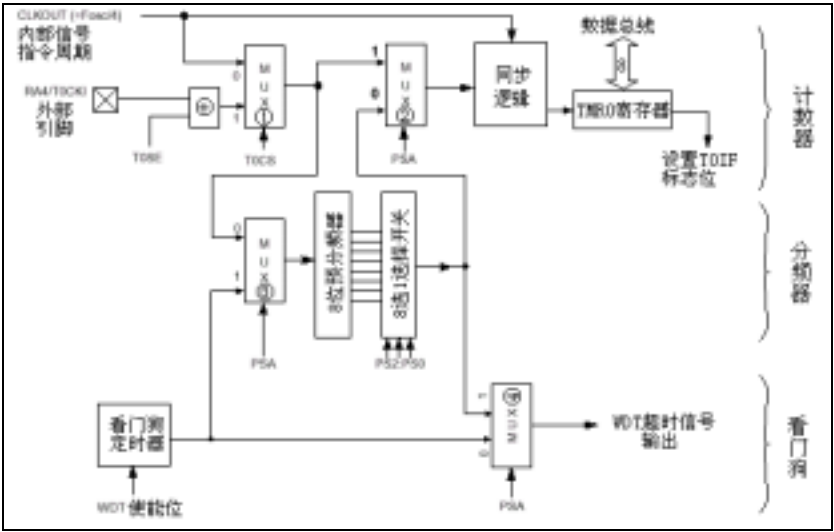


图 1 TMR0+分频器+看门狗结构图

定时器/计数器 TMR0 模块的结构方框图如图 1 所示。我们将整个电路按功能简化为 3 个相对独立的主要组成部分：计数寄存器 TMR0、分频器和看门狗定时器 WDT。图 2 所示为其简化方框图。其中，看门狗定时器 WDT 在以后的章节中将作专题介绍，图 3 是将看门狗定时器 WDT 剔除之后的方框图。

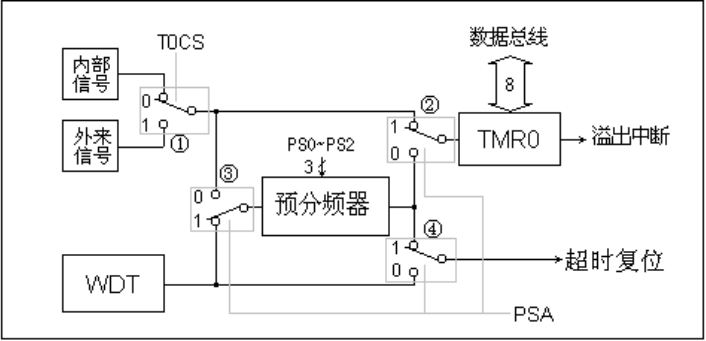


图 2 简化方框图

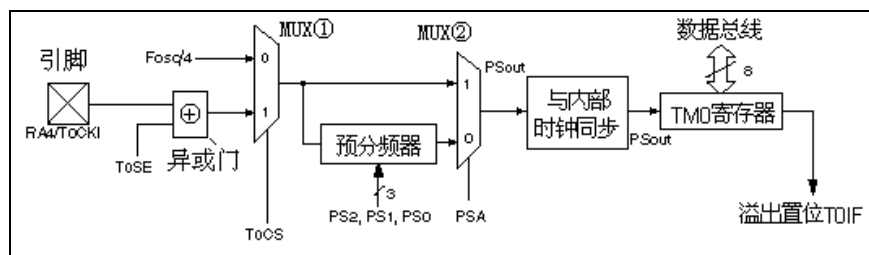


图 3 带有可编程预分频器的 TMR0 模块方框图

在图 2 中的 3 个组成部分之间借助于 3 只由同一个 PSA 信号控制的切换开关 MUX②、MUX③和 MUX④相互连接在一起。为便于理解，不妨把切换开关就看成是电子制作中常用的 2 选 1 选择开关。MUX②、MUX③和 MUX④3 只切换开关还可以理解为一只有 3 组单刀双掷转换开关的继电器的 3 组触点（如图 4 所示）。当 PSA 控制端送来（逻辑 0）低电平信号时，继电器保持静止，3 组开关靠自身弹力倒向静合触点“0”一侧；而当 PSA 端送来（逻辑 1）高电平信号时，继电器得电吸合，3 组开关靠磁力转换到动合触点“1”一侧。

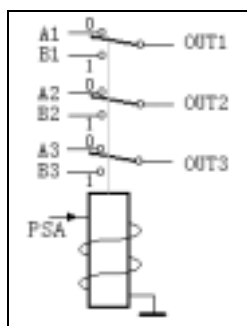


图 4 带有 3 组单刀双掷转换开关的继电器

1. 分频器 看门狗定时器 WDT 与 TMR0 共同分享同一个分频器，但两者不能同时使用。也就是说，在某一时刻分频器只能分配给两者中的一个。与 TMR0 配合使用时，它是以一个“预”分频器的角色出现在 TMR0 的输入信号路径中的；而与 WDT 配合使用时，它是以一个“后”分频器的角色出现在 WDT 的输出信号路径中的。分频器实际上也是一个 8 位累加计数器，不过它不能象 TMR0 那样通过内部数据总线用程序进行读、写操作，并且它只能配合 TMR0 或 WDT 起分频作用。由于它主要用来与 TMR0 配合工作，因此在厂家提供的产品手册中总是习惯地把它称作“预分频器”，分频器的电路结构示意图如图 5 所示。可以把分频器看作由两片 CMOS 通用集成电路构成，一片是 12 位二进制计数器 CD4040（在此仅使用低 8 位），一片是 8 选 1 模拟开关 CD4051（或者是一片 8 选 1 数据选择器 74LS151）。当开关切换到 Q1 点时，时钟信号 CLOCK 经过 1 级二进制分频后送到 OUT 端，分频比为 1: 2；当开关切换到 Q2 点时，时钟信号 CLOCK 经过 2 级二进制分频后送到 OUT 端，分频比为 1: 4；当开关切换到 Q3 点时，时钟信号 CLOCK 经过 3 级二进制分频后送到 OUT 端，分频比为 1: 8；……；当开关切换到 Q8 点时，时钟信号 CLOCK 经过 8 级二进制分频后送到 OUT 端，分频比为 1: 256。开关的切换位置取决于 PS2~PS0 的值，也就是由 PS2~PS0 设定分频比。

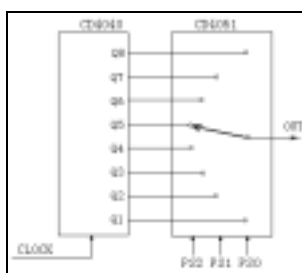


图 5 分频器等效电路

分频器的功能就是将进入 TMR0 的时钟信号（或从 WDT 送出的时钟信号）频率除以一个指定的倍数，这个倍数就是分频比，由 OPTION_REG 寄存器中的 PS2~PS0 比特决定。

究竟将分频器配置给 WDT 还是 TMR0，这要由控制信号 PSA 的逻辑电平来决定。当 PSA 为低电平时，分频器归 TMR0 所有，进入累加计数器 TMR0 的时钟信号，都要先经过分频器；而当 PSA 为高电平时，分频器与 TMR0 隔离，进入 TMR0 的时钟信号，不再经过分频器。

应注意，当分频器分配给 TMR0 时，任何以 TMR0 为目标的写操作指令（如 CLRF 1，MOVWF 1）都会同时将分频器清零。同理，当分频器分配给 WDT 时，一条清 WDT 的指令(CLRWDT)将会同时清零其分频器。这里指的是分频器清零，而分频比和分配对象并不会改变。

TMR0 累加计数寄存器 顾名思义，定时器/计数器 TMR0 模块既可以作为定时器使用，也可以作为计数器使用，或者说，TMR0 具有定时器和计数器两种工作模式。实际上，两种模式之间的主要差异就是送入累加计数寄存器 TMR0 的时钟信号的来源不同而已。TMR0 的工作模式由 T0CS 位，即选项寄存器 OPTION_REG 比特 5（厂家提供的技术资料中记为 OPTION_REG <5>）决定，表 1 为 TMR0 的工作模式。

表 1

T0CS	TMR0 工作模式	触发信号的来源
0	定时器	计数器的触发信号取自内部指令周期
1	计数器	计数器的触发信号取自外部引脚 T0CKI 电平的上升沿/下降沿

定时器模式 当 T0CS (OPTION_REG <5>) = “0” 时，TMR0 模块被设定为定时器模式，触发信号源取自于芯片内部的指令周期信号。也常被说成是，指令周期信号作为累加计数器的时钟信号源。在定时器工作模式下，一旦往计数寄存器中写入初始值后，TMR0 便重新启动累加计数。在没有使用分频器的情况下，TMR0 会在每个指令周期信号（等于晶体振荡器产生的主时钟周期的 4 倍）到来时自动加 1。在配置了分频器的情况下，TMR0 会在每次收到由分频器将指令周期信号分频一个固定倍数后产生的信号时自动加 1。如果 TMR0 在累加计数的过程中，CPU 执行一条往 TMR0 中写入数据的指令，则累加计数器的加 1 操作将被推迟两个指令周期，重新开始计数。这两个指令周期的偏差在用户编写时间精度要求较高的程序时应引起注意，可以通过在每次写入 TMR0 时给一个调整值的方法来解决。

计数器模式 当 T0CS (OPTION_REG <5>) = “1” 时，TMR0 模块被设定为计数器模式，触发信号源取自于芯片外部引脚 RA4/T0CKI 上的输入信号。也常被说成是，外部输入信号作为累加计数器的时钟信号源。当工作在计数器模式时，T0SE(OPTION<4>)比特决定外部时钟信号的触发边沿：T0SE=1，下降沿触发；T0SE=0，上升沿触发。这是因为控制信号和引脚信号成逻辑“异或”关系，即经过一只异或门之后形成触发信号，见图 1。“1”与“1”异或后得“0”，相当于输入信号多经过一级“非门”；“0”与“1”异或得“1”，相当于输入信号被直接送入。当 TMR0 工作于计数器模式下，一旦往计数寄存器中写入初始值后，TMR0 便立即开始新一轮的累加计数。在没有使用分频器的情况下，TMR0 会在每个 T0CKI 信号的上升沿或下降沿到来时自动加 1。在此模式下，外部随机送入的触发脉冲信号和内部的工作时钟之间存在一个同步的问题。也就是说，并不是外部触发信号的跳变沿一送入，TMR0 就立即进行加 1 操作，而是要经过一个同步逻辑，该触发信号与系统时钟进行同步之后，方能进入累加计数器 TMR0，引发一次加 1 操作。

与定时器/计数器 TMR0 模块相关的寄存器

在 PIC16F87X 单片机的 RAM 数据存储区域，与定时器/计数器 TMR0 模块有关的特殊功能寄存器共有 4 个，分别是：8 位宽的累加计数寄存器 TMR0、中断控制寄存器 INTCON、选项寄存器 OPTION_REG 和端口 RA 方向控制寄存器 TRISA（如表 2 所示）。这 4 个寄存器都具有在 RAM 数据存储区域中统一编码的地址，也就是说，PIC 单片机可以把 TMR0 当作一个普通寄存器单元来访问（即读出或写入），这样有利于减少指令集的指令类型和指令数

量，也便于用户的学习、记忆和编程。

表 2

寄存器名称	寄存器符号	寄存器地址	寄存器内容							
			Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
累加计数寄存器	TMR0	01H/ 101H	8 位累加计数寄存器							
中断控制寄存器	INTCON	0BH/8BH/ 10BH/18BH	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
选项寄存器	OPTION_REG	81H/ 181H	/RBPUP	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
端口 RA 方向控制寄存器	TRISA	85H	-	-	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0

1.选项寄存器 OPTION_REG 选项寄存器是一个可读/写的寄存器。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
/RBPUP	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

与 TMR0 有关各位的含义如下：

(1) PS0~PS2：分频器分频比选择位，见表 3。

表 3

PS0~PS2	TMR0 比率	WDT 比率
0 0 0	1:2	1:1
0 0 1	1:4	1:2
0 1 0	1:8	1:4
0 1 1	1:16	1:8
1 0 0	1:32	1:16
1 0 1	1:64	1:32
1 1 0	1:128	1:64
1 1 1	1:256	1:128

(2) PSA：分频器分配位。1=分频器分配给 WDT；0=分频器分配给 TMR0。

(3) T0SE：TMR0 的时钟源触发边沿选择位。只有当 TMR0 工作于计数器模式时，该位才发挥作用。1=外部时钟 T0CKI 下降沿触发 TMR0 递增；0=外部时钟 T0CKI 上升沿触发 TMR0 递增。

(4) T0CS：TMR0 的时钟源选择位。1=由 T0CKI 外部引脚输入的脉冲信号作为计数器 TMR0 时钟源；0=由内部提供的指令周期信号作为定时器 TMR0 时钟源。

2.中断控制寄存器 INTCON 中断控制寄存器也是一个可读/写的寄存器。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
GIE	EEIE	T0IE	INTE	RBIF	T0IF	INTF	RBIF

与 TMR0 有关的各位的含义如下：

(1) T0IF：TMR0 溢出标志位（也就是溢出中断标志）。1=TMR0 发生溢出；0=TMR0 未发生溢出。

(2) T0IE：TMR0 溢出中断使能位。1=允许 TMR0 溢出后产生中断；0=屏蔽 TMR0 溢出后产生中断。

(3) GIE：全局中断总使能位。1=允许 CPU 响应所有外围设备模块产生的中断请求；0=禁止 CPU 响应所有外围设备模块产生的中断请求。

3. 端口 RA 方向控制寄存器 TRISA 与 TMR0 有关的只有一个比特位。由于 TMR0 模块的外部输入信号 T0CKI 与端口引脚 RA4 是复合在同一条引脚上的，当 TMR0 工作于计数器模式时，要求该脚必须设定为输入方式，作为 T0CKI 信号专用输入引

脚。即：1=端口引脚 RA4 设定为输入，以便从该脚送进 T0CKI 信号。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	-	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0

第九讲 定时器/计数器（下）

• 李学海 • E-mail:lixuehai@163.net

定时器/计数器 TMR0 模块的应用举例

由于定时器/计数器模块是一种应用比较灵活的外设模块，因此我们在本节中充分利用 ICD 配套演示板上有限的硬件资源，尽可能多地列出几个实验范例，以展示定时器/计数器 TMR0 模块的各种应用。实验条件如下：硬件环境：CPU 为赛扬 466 的 PC 机；MPLAB-ICD 在线调试器及其演示板；SR8 双踪示波器；MF500A 型万用表；25W 内热式烙铁；常用电子元器件以及连接导线。软件环境：WINDOWS98；MPLAB-IDE 综合开发环境应用软件包 V4.99.07 版；WORD2000。

1. TMR0 用作硬件定时器 能否以“查询”方式利用可编程定时器 TMR0 模块产生延时？这与采用软件手段产生延时的方法相比有什么好处？在下面的实验范例中将得到验证。可以看到硬件方式延时，使得程序更精练。实际上，如果以“中断”方式利用 TMR0 延时还会节省 CPU 的时间。

[实验 1] 队列灯 实验实现的功能是：把演示板上的 8 只 LED 发光二极管，设计为轮流发光。也就是在图 6（a）所示的 16 个显示状态之间轮流切换，并且在各个状态之间切换时，插入一个 256 毫秒（ms）的延时。

硬件电路规划：队列灯电路如图 6（b）所示。图中的电路元件标号尽量与《MPLAB-ICD 用户指南》中的电路图保持了一致，其他实验范例的电路图中元件标号的标法也是如此。

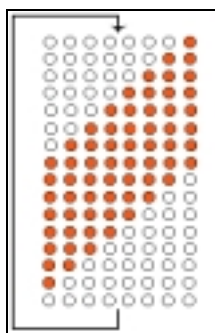


图 6（a） 队列灯

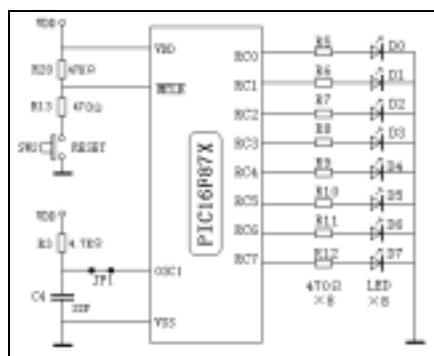


图 6（b） 队列灯电路

软件设计思路：我们曾经学习过利用软件手段，在程序执行过程中插入一段延时的方法。现在来看看硬件手段延时具体是如何实现的。在本例中，将分频器配置给 TMR0 使用，并且分频比设定为最大（1: 256）。利用 TMR0 编制了一段 64 毫秒（ms）的延时子程序。LED 显示驱动码的获取采用了查表法，在表中预先存储了设定好的编码。并列摆放的 8 只 LED 的亮灭规律符合“先入队列的亮灯，最先移出队列”，所以我们就给它取一个雅致的名字叫“队列灯”。

程序流程图见图 7，包含主程序和子程序的流程图。

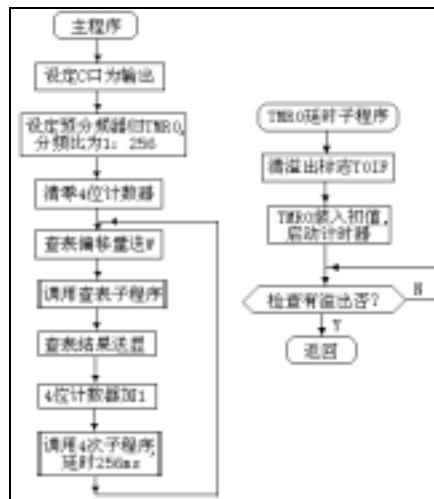


图 7 流程图

程序清单:

```

;*****
; 《队列灯》
;程序文件名为 "TMR0EXP1.ASM".
;*****

tmr0equ 01h      ;定义定时器/计数器 0 寄存器地址
pcl      equ 02h    ;定义程序计数器低字节寄存器地址
status   equ 3h     ;定义状态寄存器地址
option_reg equ 81h  ;定义选项寄存器地址
intcon    equ 0bh   ;定义中断控制寄存器地址
portc     equ 7h    ;定义端口 C 的数据寄存器地址
trisc     equ 87h   ;定义端口 C 的方向控制寄存器地址
tmr0b     equ 6      ;定义 TMR0 寄存器初始值 (250=256-6)
count     equ 20h    ;定义一个计数器变量寄存器
rp0       equ 5h     ;定义状态寄存器中的页选位 RP0
;***** 主程序 *****
        org 000h;定义程序存放区域的起始地址
main
        nop                ;设置一条 ICD 必需的空操作指令
        bsf status, rp0;设置文件寄存器的体 1
        movlw 00h          ;将端口 C 的方向控制码 00h 先送 W
        movwf trisc        ;再转到方向寄存器, RC 全部设为输出
        movlw 07h          ;设置选项寄存器内容: 分频器给 TMR0;
        movwf option_reg;分频比值设为 "1: 256"
        bcf status, rp0;恢复到文件寄存器的体 0
        clrfcount          ;清零计数器
loop
        movfcount, 0 ;count 作为查表地址偏移量送入 W
        callread;调用读取显示信息子程序
        movwf portc      ;将查表得到的驱动码送显
        incfcount, 1 ;计数器加 1
        movlw 0fh        ;屏蔽掉高 4 位(count 等效为 4 位计数器),

```



```

andwf    count,1 ; 以免偏移量超出表格范围
calldelay ;调用延时子程序
calldelay ;调用延时子程序
calldelay ;调用延时子程序
calldelay ;调用延时子程序 4 次共延时 256ms
gotoloop;跳转到
;***** TMR0 延时子程序 (64ms) *****
delay
    bcf    intcon,2;清楚 TMR0 溢出标志位
    movlw  tmr0b ;TMR0 赋初值,
    movwf  tmr0; 并重新启动定时计数
loop1
    btfss  intcon,2;检测 TMR0 溢出标志位
    gotoloop1 ;没有溢出,循环检测
    return ;子程序返回
;***** 读取显示信息的查表子程序 *****
read
    addwf  pcl,1 ;地址偏移量加当前 PC 值
    retlw  b'00000001';显示信息码,下同
    retlw  b'00000011';
    retlw  b'00000111';
    retlw  b'00001111';
    retlw  b'00011111';
    retlw  b'00111111';
    retlw  b'01111111';
    retlw  b'11111111';
    retlw  b'11111110';
    retlw  b'11111100';
    retlw  b'11111000';
    retlw  b'11110000';
    retlw  b'11100000';
    retlw  b'11000000';
    retlw  b'10000000';
    retlw  b'00000000';
end ;通知汇编器源程序结束

```

小结：在本例中，旨在让读者学习以下内容：

1. 程序中“movwf tmr0”语句一旦被执行，将同时完成 3 个任务：向 TMR0 写入初值；将分频器清零（请注意：分频比并不改变）；重新启动计数器 TMR0 开始工作。
2. 分频器的使用方法。
3. 硬件方法延时同软件方法延时一样，延迟时间的计算结果与单片机的时钟晶体振荡器的工作频率有着密切的关系。在下面的延时计算式中，假设的是时钟频率为 4MHz。这样假设的理由是，在 ICD 套件中提供的演示板上，方便可用的时钟振荡方式为 RC 振荡方式，并且电阻选的是 4.7kΩ，电容选的是 22pF。RC 振荡器工作频率范围在 3.5MHz 到 6MHz 之间，会在一定程度上受工作电压和环境温度的影响。在本例中就按 4MHz 来计算，因此，一个指令周期就是一个微秒（μs）的时间。

4. 延时时长的计算方法。延时计算式为 $256 \times (256 - 6) = 64000$ 指令周期 $= 64000 \mu s = 64ms$ 。其中，前面的 256 是分频比；括号内的 256 是 TMR0 产生溢出时累加计数的最大值；“6”是每次循环累加计数开始时需要向 TMR0 填写的初始值。也就是 TMR0 初始值应为“6”，即在“6”的基础上开始递增，直到计数到 256 时产生溢出。

5. 查表程序的应用方法。

6. 常数标号的定义和引用方法。比如，程序中的“`tmr0b equ 6`”为定义语句；“`movlw tmr0b`”为引用语句。如果同一个常数标号，在程序中多处被引用，这种做法会给常数值值的修改带来极大的方便。

2. TMR0 多次被引用 如何实现在同一用户程序中的两个不同的地方，利用同一可编程定时器 TMR0 模块，来产生两个不同时长的延时，并且避免发生冲突？在下面的实验中，试图为解决这类问题提供一条可行途径。

[实验 2]单键循环切换方波信号发生器 实验实现的功能：利用定时器/计数器 TMR0 模块作为硬件定时器，来控制端口引脚 RC0 上产生频率分别为 15.625Hz、31.25 Hz、62.5 Hz、125 Hz、250 Hz、500 Hz、1000Hz 和 2000Hz 的 8 种方波信号。加电伊始，该信号发生器自动输出的是最低档的 15.625Hz 的方波信号，每按动一次接在端口引脚 RB0 上的按钮开关 SW1 后，输出的方波信号的频率就会提高一倍，直到频率到达 2000Hz，再次按动 SW1 后，输出方波信号的频率就回到 15.625Hz，从而可以实现单键控制循环切换。输出的方波信号可以借助于示波器观察和验证，也可以接上一只蜂鸣器进行监听，不过其中较低的频率已经超出了人耳听觉的能力范围。

硬件电路规划：单键循环切换方波信号发生器电路见图 8（a）。

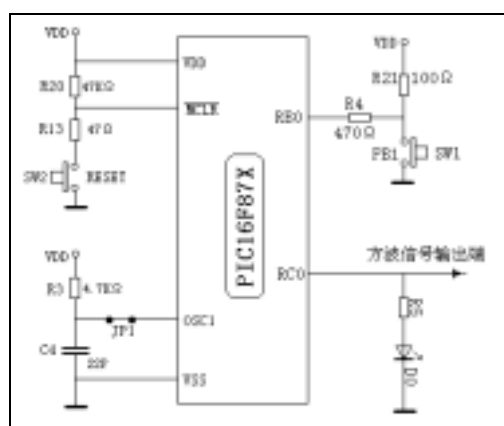


图 8（a）方波信号发生器电路

软件设计思路：令 TMR0 初始值为 131 ($=256-125$)，计数 125 个指令周期就产生溢出。也就是，TMR0 从计数开始到溢出需要 125 微秒 (μs)。通过改变分频器的分频比，来达到产生不同频率的目的。TMR0 溢出一次就使端口引脚 RC0 的输出电平反转一次，每反转两次引脚电平就形成方波信号的一个周期。当分频比为 1: 256 时，TMR0 的延时为 $125 \times 256 = 32000 \mu s$ ，方波信号周期应为 $32000 \mu s \times 2 = 64000 \mu s = 64ms$ ，对应的方波信号频率为最低挡的 15.625Hz；当分频比为 1: 2 时，TMR0 的延时为 $125 \times 2 = 250 \mu s$ ，方波信号周期应为 $250 \mu s \times 2 = 500 \mu s = 0.5ms$ ，对应的方波信号频率为最高挡的 2000Hz。在以上的计算方法中，假设时钟振荡器频率为 4MHz，并且忽略了一些附属操作指令占用的时间，实践证明这样做在本例中是可行的。按键检测接收防抖动延时子程序，也可以利用同一个 TMR0，但必须注意在调用子程序时，要避免主程序和子程序中都会用到的选项寄存器 OPTION_REG 内容发生冲突而被覆盖掉。

程序流程图：主程序和子程序的流程图见图 8（b）。


```

check    btfss    portb,0 ;测试 SW1 断开否? 是! 跳过下条指令
          goto     check ;否! 则循环检测
          call     delay ;调用延时子程序, 消除断开抖动的影响
          btfss    portb,0 ;再次测试 SW1 断开否? 是! 跳过下条指令
          goto     check ;否! 则循环检测

bsf       status, rp0;是! 设置文件寄存器的体 1
decf      option_reg,1;选项寄存器递减
movlw     07h      ;为了确保 bit7~bit3 是 0
          andwf    option_reg;屏蔽选项寄存器高 5 位
movf      option_reg,1;自己传到自己为的是影响 Z 标志位
bcf       status, rp0;恢复到文件寄存器的体 0
btfsc     status,2 ; 检查选项寄存器减到 0 否? 否! 跳一步
goto      main     ;是! 回到起始处开始下一轮的频率切换

loop      movlw    01h      ;用“异或”算法  $1 \oplus A = \overline{A}$ ;  $0 \oplus A = A$ 
          xorwf    portc    ;将 RC0 引脚电平取反
          bcf      intcon,2 ;清除 TMR0 溢出标志位
          movlw    tmr0b     ;TMR0 赋初值,
          movwf    tmr0; 并重新启动定时器开始计数

test      btfss    intcon,2 ;检测 TMR0 溢出标志位
          goto     test     ;没有溢出, 循环检测
          goto     keyin    ;有溢出, 检测键盘输入

;***** TMR0 延时子程序 8ms *****
delay
          bsf      status, rp0;设置文件寄存器的体 1
          movf     option_reg,0;保护选项寄存器的内容,
          movwf    option_b; 到备份寄存器
          movlw    04h      ;设置选项寄存器: 分频器给 TMR0;
          movwf    option_reg;分频比初值设为 1: 32
          bcf      status, rp0;恢复到文件寄存器的体 0
          bcf      intcon,2 ;清楚 TMR0 溢出标志位
          movlw    06h      ;TMR0 赋初值 250=256-6, 便启动定时计数
          movwf    tmr0;

loop1     btfss    intcon,2 ;检测 TMR0 溢出标志位
          goto     loop1    ;没有溢出, 循环检测
          bsf      status, rp0;设置文件寄存器的体 1
          movf     option_b,0; 从备份寄存器,
          movwf    option_reg; 恢复选项寄存器的内容
          bcf      status, rp0;恢复到文件寄存器的体 0
          return      ;子程序返回
end        ;通知汇编器源程序结束

```

小结: 在本例中, 读者学习了以下主要内容:

1. 在同一软件程序的两个不同之处引用同一可编程定时器 TMR0 模块的方法, 及其避免冲突的措施。可以在进入子程序之前的主程序内, 安排对可能发生冲突的寄存器保留备份的指令; 也可以在进入子程序之后的子程序内, 安排对可能发生冲突的寄存器保留备份的指令。本例中的方式属于后者。

2. “表达式”的应用会给程序设计者带来很多方便，尤其是对于那些复杂的计算。在本例采用了一个简单的表达式，赋值伪指令“`tmr0b equ d'256'-d'125'`”语句中，将“`tmr0b`”定义为一个常数标号。可是“`equ`”后面并没有直接给出一个 8 位常数，而是需要汇编器在汇编过程中，自动计算出结果 131 (=256-125)。在后面的程序中凡是出现“`tmr0b`”的地方，汇编器就自动用“131”来取代。

3. 用“异或”算法 ($1 \oplus b = \overline{b}$; $0 \oplus b = b$) 来将一个寄存器内的某一比特位求反而不影响其它比特位。在此仅用了两条指令：

```
movlw    01h      ; 将工作寄存器 W 的“比特 0”置成“1”
xorwf    portc,1   ; 把 PORTC 寄存器中的“比特 0”取反，即可实现将
                  ; RC0 引脚电平取反
```

当然如果在该寄存器的其它比特位不怕受到影响的情况下，也可以仅采用一条指令“`comf portc,1`”或者“`incf portc,1`”，同样能将 RC0 引脚电平取反。这也算是小小的编程技巧吧。

4. 判断一个寄存器的内容是否为“0”的编程技巧。采用一条寄存器自传指令，即传送操作的“源”和“目标”是同一个寄存器，然后跟一条“Z”标志位判别指令：

```
movf     option_reg,1 ;自己传给自己仅仅为的是影响 Z 标志位
btfsc    status,2     ;检查 Z 标志位=? 如果是“0”则跳过下一条指令
当 Z=0 表明寄存器 OPTION_REG 的内容不等于 0；而当 Z=1 表明寄存器
OPTION_REG 的内容等于 0。
```

5. 在程序中对于那些涉及目标寄存器存在不同选择的指令，比如“`decf option_reg,1`”和“`movf option_reg,1`”指令，其中代表目标寄存器（为非 W）的比特位“1”可以省缺。也就是以上两条指令可以写成“`decf option_reg`”和“`movf option_reg`”，这样汇编器在进行汇编时，会自动认定目标寄存器是 `option_reg` 寄存器，而不是 W 工作寄存器。但是当目标寄存器为 W（即该比特位为“0”）时，则不能省略。

3. TMR0 用作硬件计数器 如何利用 TMR0 模块作为一个硬件计数器？这种用法又有什么实用价值？在下面的实验范例中将得到验证。

[实验 3]车辆里程表 实验实现的功能：假设某摩托车厂生产的摩托车，车轮直径为 43 厘米，那么，该车行走 1 公里需要车轮运转约 734 圈。即 $734 \text{ 圈/公里} \approx 1000 \text{ 米/公里} \div (0.43 \text{ 米} \times \pi)$ 。在车体上找一个能够检测车轮转动的适当位置，安装一个磁敏传感器（比如廉价易购的 3 脚霍尔器件）。在与磁敏传感器位置相对的摩托车转动部件上，安装一块小磁铁。这样车轮运转时形成磁敏传感器与小磁铁之间的相对位移，从而产生一系列的电脉冲信号。将该信号作为单片机内部可编程计数器的外接引脚输入的触发信号，供计数器累加计数。充分利用现有的演示板资源，可以进行开发产品的前期模拟实验。

硬件电路设计：电路如图 9 所示。用演示板上的 8 只发光二极管 LED，作为简易的里程累加值输出显示器件，以 8 位二进制形式显示公里数。提供给计数器的外来触发信号，用一只通用 CMOS 数字集成电路——六反相器 CD4069 构成的多谐振荡器产生，振荡频率可以用一只电位器调节，模拟摩托车的调速器。频率越高就相当于摩托车跑得越快。按图中给定的电路参数，调节电位器时，模拟的摩托车速度可以在 9~87 公里/小时之间变化。也可以利用信号发生器或示波器上的 1000Hz 校准信号输出等，作为触发信号源；还可以利用一只按钮开关来产生触发脉冲信号。这两种情况可以参考图 10 提供的按钮输入和脉冲输入防抖动电路，在演示板上布满焊孔的区域内进行安装和连线，这也是对图 9 中振荡器电路部分的变通。图 10 中带有正反馈回路的两只反相器，等效于一只可调滞后电压（取决于两只电阻的阻值）的施密特触发器。有这只施密特触发器的存在，可以对外来信号源起整形、缓冲和保护单片机引脚免遭过压冲击的作用。对于按钮脉冲输入方式，该施密特触发器再配合一只电容，可以很好地起到以硬件方式消除抖动和抗干扰的作用。可编程计数器 TMR0 的触发信号，从外接引脚 T0CK 送入，这是一条与端口引脚 RA4 复用的引脚。

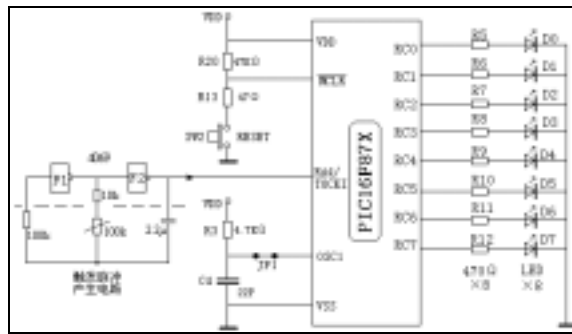


图 9 里程表

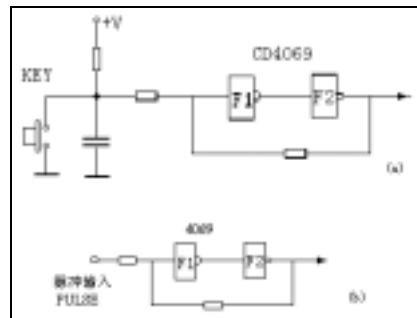


图 10 按钮输入和脉冲输入防抖动电路

软件设计思路:在前面两个实验中,都是将 **TMR0** 当作定时器使用的。在本例中,让 **TMR0** 工作在计数器模式。要求当送入 **740** 个脉冲时,计数器产生一次溢出,令里程表累加器作一次加 **1** 操作。可是 **740** 超出了 **TMR0** 的计数范围,这就需要借助于分频器来帮忙。将分频器配置给 **TMR0** 使用,并且分频比设定为 **1: 4**。**740** 除以 **4** 等于 **185**,这样以来就落到了 **TMR0** 的计数范围 (**0~255**) 之内了。里程表累加器由 **PORTC** 寄存器充当,它会将里程累加值直接映射到作为显示部件的 **8** 只发光二极管上,从而省去了输出送显指令。程序流程图如图 **11** 所示。

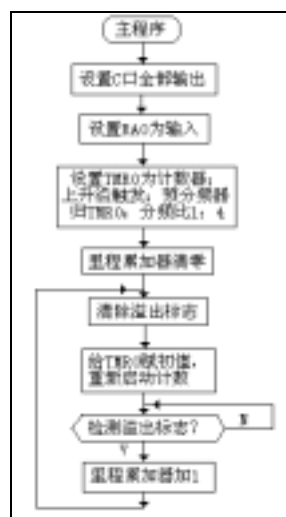


图 8.11 流程图

程序清单:

```

;*****
; 《里程表》
;程序文件名为 "TMR0EXP3.ASM".
;*****

```

```

tmr0equ 01h      ;定义定时器/计数器 0 寄存器地址
status equ 3h    ;定义状态寄存器地址
option_reg equ 81h ;定义选项寄存器地址
intcon equ 0bh   ;定义中断控制寄存器地址
portc equ 7h     ;定义端口 C 的数据寄存器地址
trisc equ 87h    ;定义端口 C 的方向控制寄存器地址
trisa equ 85h    ;定义端口 A 的方向控制寄存器地址
tmr0b equ d'71'  ;定义 TMRO 寄存器初始值(71=256-740/4)
rp0 equ 5h       ;定义状态寄存器中的页选位 RP0
t0ifequ 2        ;定义 TMRO 溢出标志位的位地址
f equ 1          ;定义目标寄存器指示标号
;***** 主程序 *****
org 000h;定义程序存放区域的起始地址
main
    nop          ;设置一条 ICD 必需的空操作指令
    bsf status, rp0;设置文件寄存器的体 1
    movlw 00h    ;将端口 C 的方向控制码 00h 先送 W
    movwf trisc  ;再转到方向寄存器, RC 全部设为输出
    movlw 0ffh;将端口 A 的方向控制码 FFh 先送 W
    movwf trisa  ;再转到方向寄存器, 主要设 RA0 为输入
    movlw 31h    ;设置选项寄存器内容: 分频器给 TMRO;
                ;触发信号来自引脚 TOCKI; 上升沿触发;
    movwf option_reg;分频比值设为"1: 4"
    bcf status, rp0;恢复到文件寄存器的体 0
    clrfportc    ;将里程累加器也是显示缓冲区清零
loop
    bcf intcon, t0if ;清楚 TMRO 溢出标志位
    movlw tmr0b      ;给 TMRO 赋初值,
    movwf tmr0; 并重新启动 TMRO 开始计数
test
    btfss intcon, t0if ;检测 TMRO 溢出标志位?
    gototest;没有溢出, 循环检测
    incfportc, f ;有溢出, 里程值加 1, 并送显
    gotoloop;跳回, 开始累计下一公里
end
;通知汇编器源程序结束

```

程序调试方法: 对于 MPLAB-SIM 软件模拟器而言, 当被调试的用户程序, 在执行过程中, 有时需要检测和接收单片机引脚上的输入信号。遇到这种情况时, 仅用前面介绍的调试手段, 就显得力不从心了。下面就结合本例再介绍几种关于 MPLAB-SIM 软件模拟器的新的调试手段。

引脚信号异步激励 首先打开和建立一个如图 12 所示的 MPLAB 工作台面。选用菜单命令 **Debug>Simulator Stimulus>Asynchronous Stimulus...** (调试>模拟器激励>异步激励...), 便可以打开一个如图 13 所示的“异步激励按钮框 (尚未进行分配)”。其中有 12 只按钮 **Stim1(P)~ Stim12(P)**供用户随意安排。操作方法举例: 将光标箭头移到其中一个按钮上, 比如 **Stim1(P)**上, 点击鼠标右键, 会弹出一个如图 14 所示的命令菜单。选中其中一条 **Assign Pin...** (分配引脚...) 命令, 就会出现一个如图 15 所示的 **Pin Selection(引脚选择)** 对话框。

借助于它的滑动条可以选择到所有引脚符号，我们这里需要选中其中的“T0CKI”，并且双击鼠标左键。这时图 13 中的按钮 Stim1(P)就变成了图 16 所示的“异步激励按钮框（已经作了分配）”样子。按钮 T0CKI(P)括号里的“P”代表系统默认的引脚激励信号形式是“脉冲激励”。也就是说，在用户程序运行过程中，每次用鼠标点击该按钮时，自动模拟一个 T0CKI 引脚上的脉冲输入信号。如果想改变激励信号形式，在右键点击该按钮弹出的命令菜单里（如图 14），可以选择到“脉冲激励”、“低电平激励”、“高电平激励”和“电平反转激励”4 种形式。

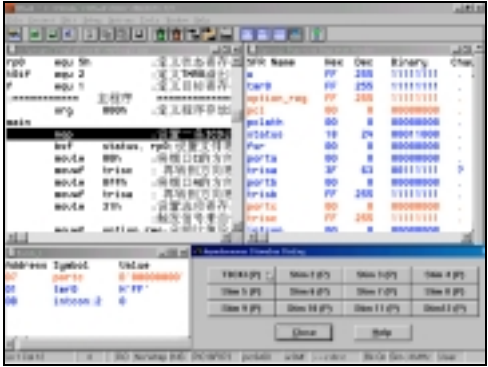


图 8.12 MPLAB 工作台面

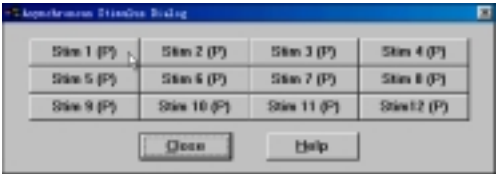


图 8.13 异步激励按钮框（尚未进行分配）



图 8.14 命令菜单

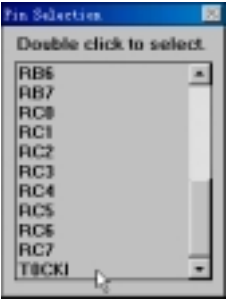


图 8.15 引脚选择对话框

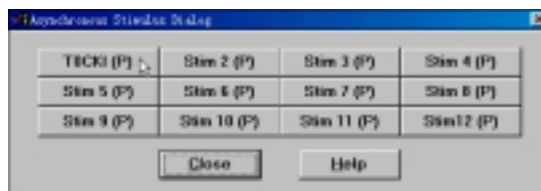


图 8.16 异步激励按钮框（已经作了分配）

编辑寄存器观察窗 打开和建立一个如图 17 所示的未经过编辑的观察窗，其中各个寄存器的内容以默认的十六进制格式显示。这样在某些情况下还显得不够方便，比如在本例中观察中断寄存器 **INTCON** 时，只关心其中的 **bit2**，希望像图 18（a）那样仅仅显示 **bit2** 的内容；又比如当作里程表累加器的寄存器 **PORTC**，其内容被直接显示到演示板的 8 只发光二极管 **LED** 上，如果像图 18（a）那样以二进制格式显示的话，就更象排列成一行的 8 只发光二极管了。那么怎样才能将图 17 的显示格式编辑成图 18（a）的显示格式呢？方法是：点击该窗口左上角的系统按钮，会弹出一个命令菜单，选择其中的 **Edit Watch...(编辑观察窗...)** 命令，就打开一个图 8.19 所示的编辑观察标号对话框，然后点击选中欲被编辑的寄存器标号（例如 **portc**），再点击刚刚变得可用的 **Properties(属性)** 按钮，出现一个图 8.20 所示的 **Properties(属性)** 选项框。在该对话框的 **Format(格式)** 区域选取 **Binary（二进制）** 项，其它保持不动，然后点击 **OK（确认）** 按钮，对于寄存器 **PORTC** 显示格式的修改就完成了。以同样的操作步骤，在图 8.19 对话框中选中 **INTCON** 寄存器，在图 8.20 选项框的 **Size(位数)** 区域选取 **bit(比特)** 项，其它维持原样，然后点击 **OK（确认）** 按钮，对于寄存器 **INTCON** 显示格式的修改就实现了。另外，通过细心观察可以发现，在 **MPLAB** 的工作台（或者称为主窗口）内被打开的各个窗口上，它们的左上角都有一个系统按钮。点击该按钮后都会弹出一个命令菜单，但是随着窗口不同，该菜单中包含的命令也不尽相同，不过都包含一条 **Toggle line numbers(行号触发开关)**。选中这条命令后，就会给相应窗口里罗列的内容自动添加行号，如图 8.18（b）所示。如果想取消行号，则按同样的方式再次选择一次该命令即可。

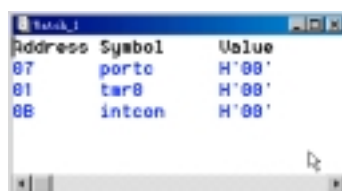


图 8.17 未经过编辑的观察窗

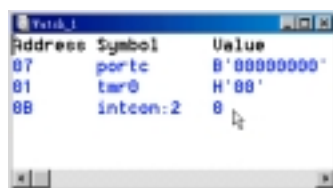


图 8.18（a） 编辑之后的观察窗

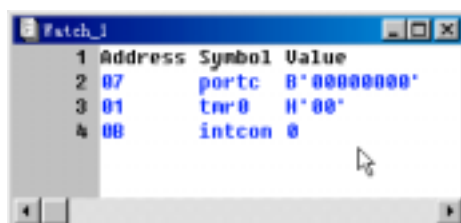


图 8.18（b） 编辑之后的观察窗（带行号）

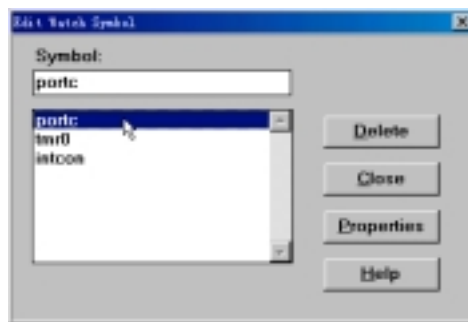


图 19 编辑观察标号对话框

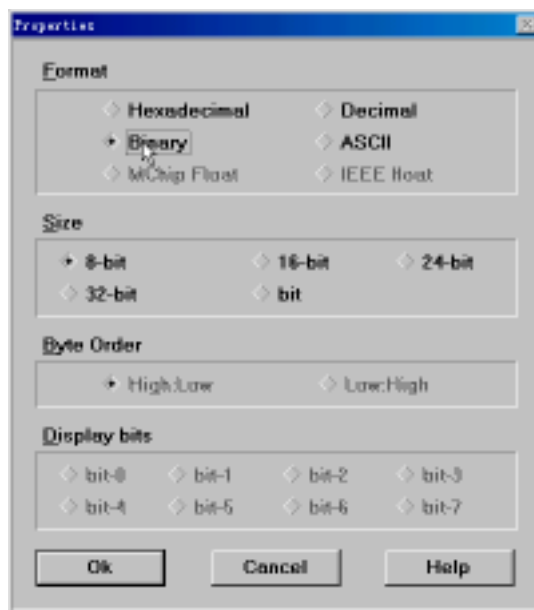


图 8.20 属性选项框

利用以上两种调试手段，调试本例程序如下：在图 12 所示的 MPLAB 工作台上，先将虚拟的“单片机”复位，接着启动“自动单步运行方式”运行程序。在使用该方式运行程序之前，最好将伪指令语句“tmr0b equ d'71”中的 71 改为 255，以便提高调试效率。理由是把 TMR0 开始计数的初始值如此修改后，就会在 T0CKI 引脚每输入 4 个脉冲（即点击 T0CKI（P）按钮 4 次）时，计数器 TMR0 便产生一次溢出，引发一次里程值加 1 操作。所以，在以自动单步运行程序的过程中，连续地点击图 8.16 中的 T0CKI（P）按钮。同时注意观察图 8.18 所示的观察窗中的寄存器 PORTC 内容，每点击 4 次按钮，PORTC 的值就会加 1 一次。

小结：在本例中，读者可以学习到以下主要内容：

1. 利用 TMR0 模块当作一个可定制累加计数器的方法。
2. 设计可以用作各种机动车简易里程表的基本思路。
3. 用硬件方式实现消除按钮抖动和抗干扰的电路设计方法。
4. 在 MPLAB-SIM 模拟调试环境下，如何注入引脚异步激励信号。
5. 对观察窗中被观察的变量或寄存器的显示格式如何进行编辑。
6. 采用 MPLAB-Editor 编辑器录入和编辑源程序，不能输入中文；采用 WINDOWS 的“记事本”录入和编辑源程序，虽然能输入中文但是编辑能力较弱；如果利用 WORD 作为录入和编辑源程序的工具，可以克服上述两者的缺点，并且它强大的编辑功能会大大提高我们的工作效率。

本实验范例只是简单模仿了一个里程表的基本工作原理，仅仅是为了满足通俗易懂的教学目的而设计的。因此，它与实用产品之间还有相当的距离，比如还存在以下有待解决的问题：

1. 累加里程值在里程表断电后不应丢失。这可以利用 **PIC16F87X** 内部的，掉电后内容不挥发的 **EEPROM** 数据存储器，就能很容易地实现这项要求。只是我们还没有讲到 **EEPROM** 数据存储器的使用方法。

2. 显示方式应该以人们习惯的十进制形式。这需要增加硬件电路，将显示方式改进为多位 7 段数码管或 **LCD** 液晶显示器；软件方面将累加结果调整成二进制表示的十进制码（计算机理论中称为 **BCD** 码），然后才能输出显示。

3. 通常，摩托车的里程表至少能够记录百万公里的累计值。本例中的里程表累加器，也是显示缓冲区，容量太小，不能满足实际需要。这需要随着显示器件位数的加宽而加大。利用单片机内部现有的 **RAM** 数据存储器，很容易实现。

第十讲 中断系统（上）

• 李学海 • E-mail:Lixuehai@163.net

1. 中断的基本概念

当计算机系统正在执行程序时，出现了某种特殊状况，比如定时时间到、有键盘信号输入等，此时 CPU 需要暂时停止当前的工作，转去处理这些特殊工作，待这些工作执行完毕后，再回到原先的工作中去，这就形成了一次中断过程。如果计算机系统缺少中断功能，则计算机的工作效率将会很低。因此，中断(Interrupt)是提高计算机工作效率的一项重要功能，中断功能是所有的微处理器（CPU 或 MPU）和微控制器（MCU 或单片机）几乎都会配置的一项功能。中断功能的强弱已经成了衡量一种微处理器和微控制器的功能是否强大的重要指标之一。

“中断源”就是引起中断的原因，一种型号单片机中断源的种类和数量，与这种型号单片机的片内包含的外围设备模块的种类和数量存在着很大程度的关联性。在 PIC 系列单片机内部，中断是作为一个功能部件而不是作为一个外围设备模块配置的。每一个 PIC16F87×系列单片机的芯片内部都集成了 10~15 个外围设备模块（型号不同，外围设备模块数略有不同）。这些外围设备模块在工作时都或多或少的需要 CPU 参与控制，那么，一个 CPU 如何“照顾”得过来这么多的外围设备模块呢？这就要用到——中断。

一次中断处理的过程是这样的：当某一中断源发出中断请求时，CPU 决定是否响应这个中断请求，假如 CPU 目前正在执行更紧急的任务，可采用中断控制方法（称为中断屏蔽）暂时不响应中断请求；如果 CPU 响应这次中断请求，必须在现行的指令执行完毕之后，把断点处的程序计数器 PC 值，也就是下一条即将被执行的指令的地址压入堆栈保留起来（称为保护断点）、同时将各重要寄存器的内容和状态标志位也保留起来（称为保护现场）。然后转到需要处理的中断服务子程序，在子程序中首先应该查清发出中断请求的具体中断源，然后清除该中断源对应的中断标志位以避免 CPU 对同一个中断重复响应多次，接下来就跳转到与该中断源对应的程序分支去执行。当中断处理完毕之后，先恢复事先被保留下来的各个寄存器的内容和状态标志位（称为恢复现场）、再恢复程序计数器 PC 的值（称为恢复断点），使 CPU 能够返回断点处，继续执行被打断的主程序。

由此可见，一次中断过程大致经历了中断请求、中断屏蔽、中断响应、中断处理和中断返回 5 个阶段。CPU 响应中断后转入中断服务子程序的处理方法，与“PIC 汇编语言程序设计基础”部分中讲到的子程序调用的处理过程有类似之处。

2. PIC16F87X 的中断源

PIC16F87X 系列单片机具备的中断源多达 14 种（具体见表 1），但中断矢量只有一个，并且各个中断源之间也没有优先级别之分，不具备非屏蔽中断。从表 9.1 中可以看出，各中断源基本上都是与各个外围设备模块相对应的。

表 1 PIC16F87X 单片机的中断源

中断源 种类	中断源 标志位	中断源 屏蔽位	873/ 876	874/ 877	870	871	872
外部触发中断 INT	INTF	INTE	√	√	√	√	√
TMRO 溢出中断	TOIF	TOIE	√	√	√	√	√
RB 端口电平变化中断	RBIF	RBIE	√	√	√	√	√

TMR1 溢出中断	TMR1IF	TMR1IE	√	√	√	√	√
TMR2 中断	TMR2IF	TMR2IE	√	√	√	√	√
CCP1 中断	CCP1IF	CCP1IE	√	√	√	√	√
CCP2 中断	CCP2IF	CCP2IE	√	√			
SCI 同步发送中断	TXIF	TXIE	√	√	√	√	
SCI 同步接收中断	RCIF	RCIE	√	√	√	√	
SSP 中断	SSPIF	SSPIE	√	√			√
SSP I ² C 总线冲突中断	BCLIF	BCLIE	√	√			√
并行端口中断	PSPIF	PSPIE		√		√	
A/D 转换中断	ADIF	ADIE	√	√	√	√	√
EEPROM 中断	EEIF	EEIE	√	√	√	√	√
			14 种	13 种	10 种	11 种	10 种

3. PIC16F87X 的中断硬件逻辑

PIC16F87X 系列单片机的中断系统的逻辑电路如图 1 所示。每一种中断源对应一个中断标志位（记为 XXXF）和一个中断屏蔽位或者叫中断使能位（记为 XXXE）。中断源产生的中断标志信号是否得以向前传递，将受控于对应的中断屏蔽位。每一个中断标志位都对应一个触发器，当中断源申请 CPU 中断时，与之对应的触发器就由硬件自动置位，而该触发器的清位是由用户安排程序来实现的。

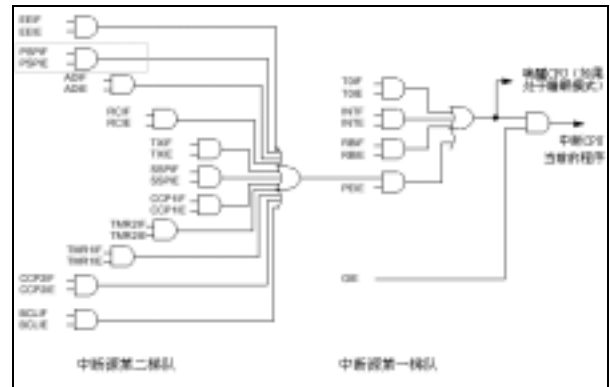


图 1 中断逻辑

图 1 中描绘的逻辑电路是一个由简单的门电路构成的组合逻辑电路。将全部 14 个中断源按两个梯队并列排开，所有的中断源都受中断屏蔽位的控制。

4. 中断相关的寄存器

与中断功能有关的特殊功能寄存器共有 6 个，分别是选项寄存器 POTION_REG、中断控制寄存器 INTCON、第一外围设备中断标志寄存器 PIR1、第一外围设备中断屏蔽寄存器 PIE1（中断屏蔽寄存器也可称为中断使能寄存器）、第二外围设备中断标志寄存器 PIR2 和第二外围设备中断屏蔽寄存器 PIE2，如表 2 所示。其中后 5 个寄存器，共有 40 个比特位，其中仅使用了 30 个比特位。分别与图 1 中的中断逻辑电路的输入逻辑信号成严格对应关系。这 6 个寄存器都具有在 RAM 数据存储器中统一编码的地址，也就是说，PIC 单片机可以把这 6 个特殊寄存器当作普通寄存器单元来访问（即读出或写入操作），这样有利于减少指令集的指令类型和指令数量，也便于用户的学习、记忆和编程。

表 9.2 与中断功能相关的寄存器

寄存器名称	寄存器符号	寄存器地址	寄存器内容							
			Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
选项寄存器	POTION_REG	81H/ 181H	/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

中断控制寄存器	INTCON	0BH/8BH/ 10BH/18BH	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
第一外设中断标志寄存器	PIR1	0CH	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
第一外设中断屏蔽寄存器	PIE1	8CH	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
第二外设中断标志寄存器	PIR2	0DH	-	-	-	EEIF	BCLIF	-	-	CCP2IF
第二外设中断屏蔽寄存器	PIE2	8DH	-	-	-	EEIE	BCLIE	-	-	CCP2IE

选项寄存器 POTION_REG POTION_REG 选项寄存器是一个可读/可写的寄存器。该寄存器包含着与定时器/计数器 TMR0、分频器和端口 RB 有关的控制位。端口引脚 RB0 和外部中断 INT 共用一脚，与该脚有关的一个控制位 INTEDG 为外部中断 INT 触发信号边沿选择位。INTEDG=1，选择 RB0/INT 上升沿触发有效；INTEDG=0，选择 RB0/INT 下降沿触发有效。

中断控制寄存器 INTCON 中断控制寄存器也是一个可读/可写的寄存器，它将第一列中的 3 个中断源的标志位和屏蔽位及 PEIE 和 GIE 囊括其中：RBIF 为端口 RB 的引脚 RB4~RB7 电平变化中断标志位。RBIF=1，RB4~RB7 已经发生了电平变化（必须用软件清 0）。RBIF=0，RB4~RB7 尚未发生了电平变化；RBIE 为端口 RB 的引脚 RB4~RB7 电平变化中断屏蔽位。RBIE=1，允许端口 RB 产生的中断。RBIE=0，屏蔽端口 RB 产生的中断；INTF 为外部 INT 引脚中断标志位。INTF=1，外部 INT 引脚有中断触发信号（必须用软件清 0）。INTF=0，外部 INT 引脚无中断触发信号；INTE 为外部 INT 引脚中断屏蔽位。INTE=1，允许外部 INT 引脚产生的中断。INTE=0，屏蔽外部 INT 引脚产生的中断；TOIF 为 TMR0 溢出中断标志位。TOIF=1，TMR0 已经发生了溢出（必须用软件清 0）。TOIF=0，TMR0 尚未发生溢出；TOIE 为 TMR0 溢出中断屏蔽位。TOIE=1，允许 TMR0 溢出后产生的中断。TOIE=0，屏蔽 TMR0 溢出后产生的中断；PEIE 为外设中断屏蔽位。PEIE=1，允许 CPU 响应来自第二梯队的中断请求。PEIE=0，禁止 CPU 响应来自第二梯队的中断请求；GIE 为全局中断屏蔽位（也可以叫作总屏蔽位或总使能位）。GIE=1，允许 CPU 响应所有中断源产生的中断请求。GIE=0，禁止 CPU 响应所有中断源产生的中断请求。

第一外围设备中断标志寄存器 PIR1 第一外围设备中断标志寄存器 PIR1 也是一个可读/可写的寄存器。该寄存器包含第一批扩展的外围设备模块的中断标志位，各位的含义如下：TMR1IF 为定时器/计数器 TMR1 模块溢出中断标志位。TMR1IF=1，发生了 TMR1 溢出（必须用软件清 0）。TMR1IF=0，尚未发生 TMR1 溢出；TMR2IF 为定时器/计数器 TMR2 模块溢出中断标志位。TMR2IF=1，发生了 TMR2 溢出（必须用软件清 0）。TMR2IF=0，尚未发生 TMR2 溢出；CCPIF 为输入捕捉/输出比较/脉宽调制 CCP1 模块中断标志位。在输入捕捉模式下：CCPIF=1，发生了捕捉中断请求（必须用软件清 0）。CCPIF=0，未发生捕捉中断请求。在输出比较模式下：CCPIF=1，发生了比较输出中断请求（必须用软件清 0）。CCPIF=0，未发生比较输出中断请求。在脉宽调制模式下：无用；SSPIF 为同步串行端口（SSP）中断标志位。SSPIF=1，发送/接收完毕产生的中断请求（必须用软件清 0）。SSPIF=0，等待发送/接收；TXIF 为串行通信接口（SCI）发送中断标志位。

TXIF=1, 发送完成, 即发送缓冲区空。TXIF=0, 正在发送, 即发送缓冲区未空; RCIF 为串行通信接口 (SCI) 接收中断标志位。RCIF=1, 接收完成, 即接收缓冲区满。RCIF=0, 正在准备接收, 即接收缓冲区空; ADIF 为模拟/数字 (A/D) 转换中断标志位。ADIF=1, 发生了 A/D 转换中断。ADIF=0, 未发生 A/D 转换中断; PSPIF 为并行端口中断标志位, 只有 40 脚封装型号具备, 对于 28 脚封装型号总保持 0。PSPIF=1, 并行端口发生了读/写中断请求。PSPIF=0, 并行端口未发生读/写中断请求。

第一外围设备中断屏蔽寄存器 PIE1 第一外围设备中断屏蔽寄存器 PIE1 也是一个可读/可写的寄存器。该寄存器包含第一批扩展的外围设备模块的中断屏蔽位 (也可称为使能位), 各位的含义如下: TMR1IE 为定时器/计数器 TMR1 模块溢出中断屏蔽位。TMR1IE=1, 开放 TMR1 溢出发生的中断。TMR1IE=0, 屏蔽 TMR1 溢出发生的中断; TMR2IE 为定时器/计数器 TMR2 模块溢出中断屏蔽位。TMR2IE=1, 开放 TMR2 溢出发生的中断。TMR2IE=0, 屏蔽 TMR2 溢出发生的中断; CCP1 为输入捕捉/输出比较/脉宽调制 CCP1 模块中断屏蔽位。CCP1IE=1, 开放 CCP1 模块产生的中断请求。CCP1IE=0, 屏蔽 CCP1 模块产生的中断请求; SSPIE 为同步串行端口 (SSP) 中断屏蔽位。SSPIE=1, 开放 SSP 模块产生的中断请求。SSPIE=0, 屏蔽 SSP 模块产生的中断请求; TXIE 为串行通信接口 (SCI) 发送中断屏蔽位。TXIE=1, 开放 SCI 发送中断请求。TXIE=0, 屏蔽 SCI 发送中断请求; RCIE 为串行通信接口 (SCI) 接收中断屏蔽位。RCIE=1, 开放 SCI 接收中断请求。RCIE=0, 屏蔽 SCI 接收中断请求; ADIE 为模拟/数字 (A/D) 转换中断屏蔽位。ADIE=1, 开放 A/D 转换器的中断请求。ADIE=0, 屏蔽 A/D 转换器的中断请求; PSPIE 为并行端口中断屏蔽位, 只有 40 脚封装型号具备, 对于 28 脚封装型号总保持 0。PSPIE=1, 开放并行端口读/写发生的中断请求。PSPIE=0, 屏蔽并行端口读/写发生的中断请求。

第二外围设备中断标志寄存器 PIR2 第二外围设备中断标志寄存器 PIR2 也是一个可读/可写的寄存器。该寄存器包含第二批扩展的外围设备模块的中断标志位, 各位的含义如下: CCP2F 为输入捕捉/输出比较/脉宽调制 CCP2 模块中断标志位。在输入捕捉模式下: CCP2F=1, 发生了捕捉中断请求 (必须用软件清 0)。CCP2F=0, 未发生捕捉中断请求。在输出比较模式下: CCP2F=1, 发生了比较输出中断请求 (必须用软件清 0)。CCP2F=0, 未发生比较输出中断请求。在脉宽调制模式下无用; BCLIF 为总线冲突中断标志位。BCLIF=1, 当同步串行端口 SSP 模块被配置成 I²C 的主模式时, 发生了总线冲突。BCLIF=0, 未发生总线冲突; EEIF 为 EEPROM 写操作中断标志位。EEIF=1, EEIF 写操作完成 (必须用软件清 0); EEIF=0, 写操作未完成或尚未开始进行。

第二外围设备中断屏蔽寄存器 PIE2 第二外围设备中断屏蔽寄存器 PIE2 也是一个可读/可写的寄存器。该寄存器包含第二批扩展的外围设备模块的中断屏蔽位, 各位的含义如下: CCP2E 为输入捕捉/输出比较/脉宽调制 CCP2 模块中断屏蔽位。CCP2E=1, 开放 CCP2 模块产生的中断请求。CCP2E=0, 屏蔽 CCP2 模块产生的中断请求; BCLIE 为总线冲突中断屏蔽位。BCLIE=1, 开放总线冲突产生的中断请求。BCLIE=0, 屏蔽总线冲突产生的中断请求; EEIE 为 EEPROM 写操作中断屏蔽位。EEIE=1, 开放 EEPROM 写操作产生的中断请求。EEIE=0, 屏蔽 EEPROM 写操作产生的中断请求。

5. 中断的处理

单片机复位后, 由硬件自动对全局中断屏蔽位进行设置 (GIE=0), 将屏蔽所有的中断源; 中断返回指令 “RETFIE” 执行后, 也由硬件自动对总屏蔽位进行设置 (GIE=1), 重新开放所有的中断源。不论各种中断屏蔽位和全局中断屏蔽位 GIE 处于何种状态, 某一中断源的中断条件满足时, 都会发出中断请求, 相应的中断标志位都会被置位 (=1)。但是, 是否能够得到 CPU 的响应, 则要根据该中断源所涉及到的中断屏蔽位的状态而定。CPU 响应中断后, 由硬件自动对全局中断屏蔽位进行清位 (GIE=0), 屏蔽所有的中断源, 以免发

生重复中断响应，然后由硬件自动把当前的程序计数器 PC 值（即程序断点地址）压入堆栈（实际为硬件堆栈），并且把 PC 寄存器置以中断向量地址（0004H），从而转向并开始执行中断服务子程序（也可叫作中断服务程序）。进入中断服务子程序后，程序中必须安排指令检查发出请求的中断源（如果同时开放多个中断源的话），这可以通过检查各个中断源的标志位来实现。一旦确定发出申请的中断源，就用软件把该中断源的标志位人为地清除（=0），否则，执行中断返回指令“RETFIE”重开中断后，由于中断标志位仍为“1”而引起 CPU 重复响应同一个中断请求。中断服务子程序的末尾必须放置一条中断返回指令“RETFIE”，执行该条指令后，不仅可以重开中断，而且还可以由硬件自动将保留在堆栈顶部的断点地址弹出，并放回到程序计数器 PC 中，使 CPU 返回和继续执行被中断的主程序。INT 外部中断时序图如图 2 所示。

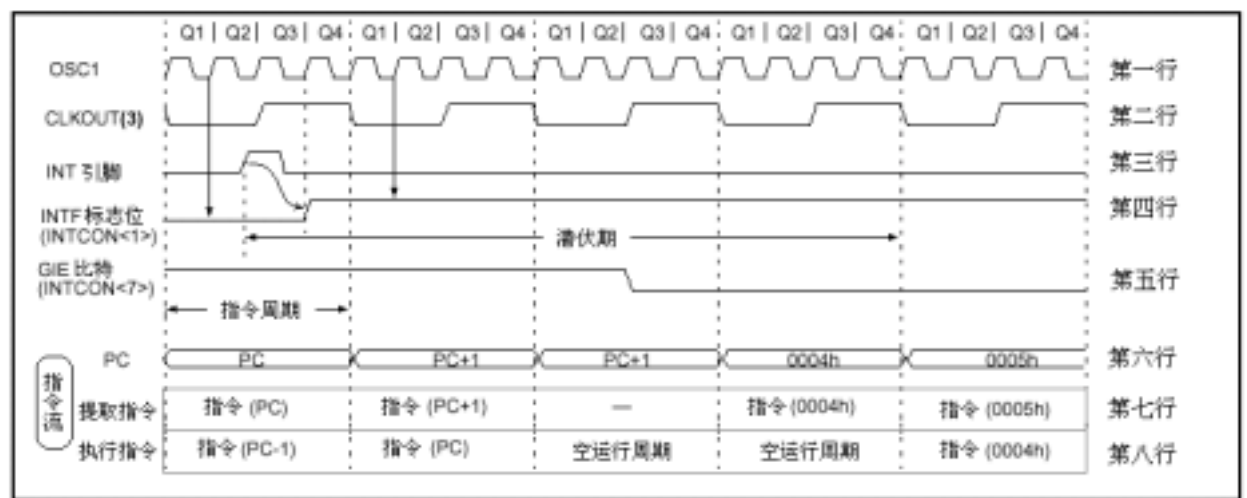


图 9.2 INT 外部中断时序图

由于具有中断功能的 PIC 系列单片机采用的是“多源中断”的设计方案（即一个中断向量对应着多个中断源），只有唯一的一个中断向量，或者说只有一个中断服务子程序入口地址。这就意味着，此类单片机的中断服务子程序只能编写一个。这类单片机的硬件结构得到简化了，那么相应的软件设计上就得多开销一些。在一个中断服务子程序中，若想对多个中断源作出处理，就必须在进入中断服务子程序后，首先执行调查具体中断源的一条或多条指令，其后才能对查到的中断源作出有针对性的服务。另外，PIC 单片机中采用的是硬件堆栈结构，其好处是既不占用程序存储器空间，也不占用数据存储器空间，并且不用用户去操作堆栈指针。但 PIC 单片机不具备象其它单片机指令系统中的压栈（PUSH）和出栈（POP）指令，所以要用一段用户程序来实现类似的功能。而这段程序的执行有可能会影响到 W 寄存器和 STATUS 寄存器，所以应该首先把这两个寄存器保护起来，然后再去保存其他用户认为有必要保护的寄存器。在 PIC 单片机中，中断现场数据不是保留在芯片的堆栈存储区中，而是保留在用户自己选择的一些文件寄存器（即 RAM 数据存储器单元）中，当然一般应该选择通用寄存器来保护现场。下面给出的是一段原厂家最新提供的实现保护中断现场的范例程序片段。

```

; 将 W, STATUS 和 PCLATH 寄存器的内容保存到临时备份寄存器中
[1] MOVWF      W_TEMP      ; 复制 W 到它的临时备份寄存器 W_TEMP 中
[2] SWAPF      STATUS, W    ; 将 STATUS 寄存器高低半字节交换后放入 W
[3] CLRF       STATUS      ; 不管当前处在哪个体，都设置体 0 作当前体
[4] MOVWF      STATUS_TEMP  ; 保存 STATUS 到体 0 上的临时寄存器 STATUS_TEMP
[5] MOVF        PCLATH, W    ; 把寄存器 PCLATH 内容复制到 W 中
[6] MOVWF      PCLATH_TEMP  ; 经 W 将 PCLATH 内容转到临时寄存器 PCLATH_TEMP
[7] CLRF       PCLATH      ; 不管当前处在哪页，都把 PCLATH 设置成指向页 0

```



```

:
: (中断服务子程序的核心部分)
:
[8] MOVF      PCLATH_TEMP, W;经过W转移
[9] MOVWF    PCLATH      ;恢复PCLATH内容
[10] SWAPF    STATUS_TEMP, W;将STATUS_TEMP寄存器高低半字节交换后放入W
[11] MOVWF    STATUS      ;把W内容移动到STATUS寄存器,
                        ; (同时也把当前体恢复到原先的体上)
[12] SWAPF    W_TEMP, F   ;将W_TEMP内容高低半字节交换后放回
[13] SWAPF    W_TEMP, W   ;再次将W_TEMP内容高低半字节交换后放入W

```

这段程序适用于 PIC16CXX 系列中的各款型号的单片机。在这段程序之前，假设预先对于待保留的各个寄存器都分别定义了相应的临时备份寄存器。用后缀 “_TEMP” 表示临时备份寄存器，例如 “W” 的临时备份寄存器记为 “W_TEMP”。

中断中需要注意的几个问题

1. 中断标志位的状态与该中断源是否被屏蔽无关。换句话说，不管是否允许 CPU 响应该中断源的请求，只要满足了中断的条件，中断标志位就会被置位 (=1)。另外，也可以利用软件将中断标志位置 “1” 或清 “0”；

2. 当开放某一中断源时，该中断源就是通过中断标志位向 CPU 申请中断的。无论什么原因，只要将中断标志位置位 (=1)，就会产生中断响应。如果用软件强行将中断标志位置位，也会产生中断响应；

3. 如果在中断被屏蔽（或禁止）的情况下，中断标志位被置位，只要不被清除就会一直保持，那么一旦解除屏蔽，就会立即产生中断响应；

4. 在中断被禁止的情况下，若中断标志位已被置位，而在允许其中断之前将它清除，那么即使解除禁止，它也不会产生中断响应；

5. 当 CPU 响应任何一个中断时，全局中断屏蔽位 GIE 将会自动清 0；当中断返回时它又会自动恢复为 1。如果在中断处理期间用软件将已经复位的 GIE 重新置位，这时再出现中断请求，就可以形成中断嵌套。也就是说，如果在响应某一中断期间又响应了其他中断请求，就形成了中断嵌套。发生中断嵌套时，前一中断处理过程被暂停而进入后一中断处理，当后一中断过程被处理完毕之后，才会继续处理前一中断。照此方式，还可以形成多级嵌套，甚至自身嵌套。不过嵌套的级数绝对不能超过硬件堆栈的深度；

6. 对于中断响应和处理时间有严格要求的应用，保护现场的指令安排也应考虑延时问题；

7. 如果同时发生多个中断请求，到底哪个中断会优先得到处理，完全取决于在中断服务子程序中检查中断源的顺序，原因是各个中断源之间不存在优先级别之分；

8. 如果清除中断标志位的指令安排在中断服务子程序的尾部，就有可能丢失在处理该中断期间该中断源第 2 次中断请求的机会（如果出现的话）。

第十讲 中断系统（下）

• 李学海 • E-mail:Lixuehai@163.net

中断功能的应用

由于中断功能是一种应用比较广泛的功能，在绝大多数的单片机控制项目中几乎都会用到此功能，因此我们在本节中尽量利用 ICD 配套演示板上不算很多的硬件资源，通过灵活配置和辅以必要的附加器件，尽可能多样化的设计几个实验范例，以便充分展现中断功能的不同用法和编程技巧。设计和验证这些实验范例所使用的实验条件如下：清华同方真爱 1000 微机系统，CPU 为赛扬 466；MPLAB-ICD 在线调试器及其演示板；MF500A 型万用表；25W 内热式烙铁；电子元器件以及连接导线。PC 机中应装有 Windows98；MPLAB-IDE 综合开发环境应用软件包；WORD2000。

1. TMR0 溢出中断 如何以“中断”方式利用 TMR0 模块产生延时？与查询方式相比，以中断方式利用 TMR0 模块产生延时是如何分解 CPU 负担的？试图利用下面的实验范例使读者得到答案。

[实验 1] 闪烁式跑马灯。把演示板上的 8 只 LED 发光二极管，规划为以跑马灯方式轮流闪烁发光。也就是 8 只 LED 中只有一只点亮，亮灯的位置以循环方式不停地移动，移动的速度取决于在各个位置上停留的时间，即在两步之间插入一个约 196ms 的延时，并且在每一个位置上 LED 都保持快速闪烁。

硬件电路规划 流水灯电路如图 3 所示。利用端口 RC 上现有的 8 只发光二极管 LED（D10~D7）作为显示部件，每只 LED 均有限流电阻（R5~R12），主要是对单片机端口内部电路起保护作用。单片机的时钟振荡器工作模式选用 RC 阻容振荡方式（将跳线 JP1 插接到演示板标有 RC 一侧的接线针上），根据电路中给定的 R3 和 C4 值，时钟频率大约为 4MHz 左右。SW2 为复位按钮，可对单片机实施人工强行复位。利用片内的定时器/计数器 TMR0 模块和中断逻辑功能部件，让 TMR0 工作于定时器模式，并且在超时溢出时向 CPU 发送中断请求信号。

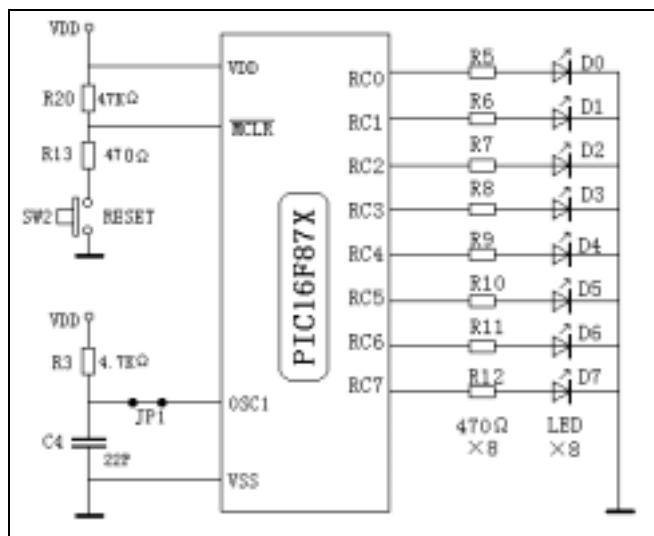


图 3

软件设计思路 驱动 8 只 LED 的显示码的形成，采用一次性向端口寄存器赋初值，然后循环移动的方式。在本例的程序中，需要加入两段延时：一个是 LED 灯每向前移动一步时（记为 T1），另一个是 LED 在亮态和灭态切换时（记为 T2）。在此 T1 延时用软件手段实现，T2 延时以硬件措施完成。将分频器配置给 TMR0 使用，并且分频比设定为最大（1：256）。利用 TMR0 编制一段大约 66 毫秒（ms）的延时子程序。

TMR0 延时时长的计算式为 $256 \times (256 - 0) = 65536$ 指令周期 $= 65536 \mu s = 65.536ms$ 。其中，前面的 256 是分频比；括号内的 256 是 TMR0 的最大计数值；“0”是每次循环累加计数开始时需要向 TMR0 填写的初始值。也就是 TMR0 在“0”的基础上开始递增，直到计数至 256 时产生溢出。即从 00H 开始经过 256 次加 1 后，累计到 100H 便产生高位溢出，并且发出中断请求。图 4 为其程序流程图。

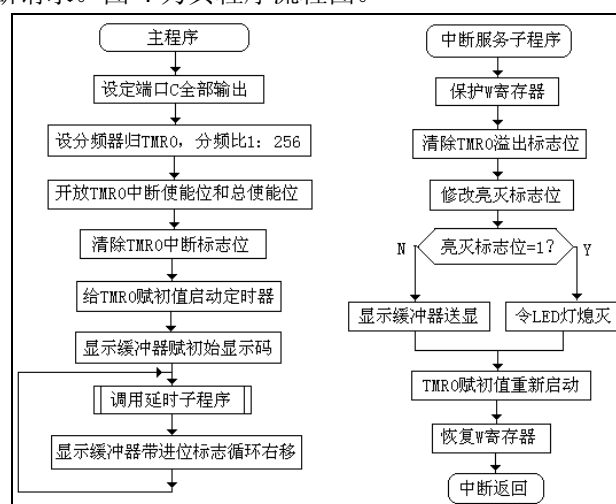


图 4

程序的调试可以在图 5 所示的调试工作台上完成。比如，利用跑表观察窗可以核对和修改延时子程序的延时时间；利用寄存器观察窗可以观察输出到 RC 端口寄存器中的显示结果。

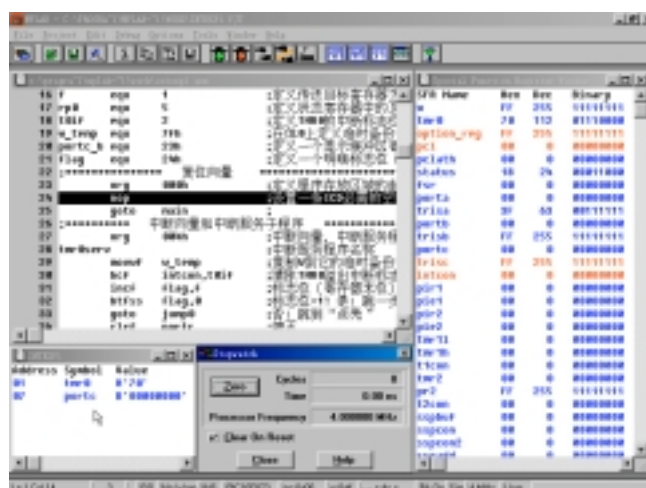


图 5

在图 5 中的寄存器观察窗里只有两个变量。如果感觉不够用，还可以把在本例用作“显示缓冲寄存器”的 PORTC_B 变量也添加到寄存器观察窗中来，方法是点击“Watch_1”观察窗左上角的系统按钮，在弹出的命令菜单中选择“Add watch...”（添加观察变量）命令，即可。

对于 PIC 单片机初学者（尤其是具备 MCS-51 单片机基础的人）来说，往往不习惯 PIC 单片机的这种与众不同的指令格式。特别是对于那些存在不同传送目标的算术运算、逻辑运算或传送操作指令（PIC16 系列单片机的指令系统中具备 14 条这样的指令）。可能常常会把表示传送目标寄存器是 W（d=0）还是文件寄存器即 RAM 单元（d=1）的指示位（d）忽略掉。

如果您漏掉的是 d=1，汇编器将所有缺省的 d 自动默认为 1，这样您的程序在运行时不会出现问题；如果您漏掉的是 d=0 的话，那肯定就会出现麻烦，您原本打算送入工作寄存器 W 中的数据，却没有送达。不管您漏掉的 d 是何值，其汇编过程都可以顺利通过。例如，本例的服务中断子程序中有一条“jump0 movf portc_b,w”指令，其中的“w”如果漏掉的话，运行程序时就会出现这样的征兆：8 只灯一起不停地闪烁。依据这些提示，您可以检查源程序，如果您发现漏掉的“d”属于错误的省略，就在补充之后重新汇编即可。

2. INT 外部中断 外部中断 INT 引脚有什么用途？以及如何利用？读者或许从下面这个实验范例中获得一点启发和思路。

【实验 2】 带电源故障报警和备用电池切换功能的流水式灯箱控制器。利用演示板上现成的 8 只 LED 发光二极管，来模拟指定的被控对象。其中，LED0 模拟光电耦合器驱动信号；LED1~LED7 模拟 7 组或 7 只灯箱。让它们以流水方式轮流发光。也就是以图 6 所示的 4 个显示状态之间循环切换，形成流水的视觉效果。并且当电源出现故障（即电源电压跌落或掉电）时，自动接通备用电池，并令流水灯暂时停下来，令 7 只 LED1~LED7 一起快速闪烁，以提醒维护人员及时处理。直到电源恢复正常后，7 只 LED1~LED7 才从被打断的显示状态上继续开始流水式发光。

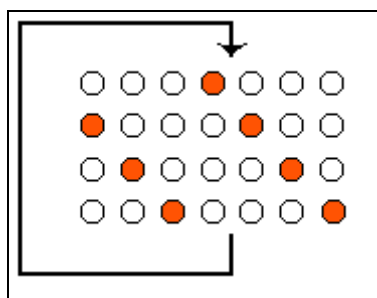


图 6

硬件电路规划 流水灯控制电路如图 7 所示。利用端口 RC 上现有的 8 只发光二极管 LED 作为流水灯模拟发光部件。利用片内的定时器/计数器 TMR0 模块和中断逻辑功能部件，让 TMR0 工作于定时器模式，并且在超时溢出时向 CPU 发送中断请求信号。利用外部中断信号输入脚 INT，作为电源故障检测端。

如图 8 所示（其中的元器件 Ra、Rb、Rc、Da、Ua 和 Ea 是在演示板原来基础上额外添加的），三端稳压器 7805 的输入电压约 9V 左右，输出电压为 5V。从 7805 的输入端连接一个电阻分压支路到地，中间抽头连接到单片机的 INT 脚。由于单片机 INT 引脚的内部设有施密特触发器，因此可以将分压信号直接接到 INT 脚。当分压信号刚刚开始下降时，首先由 INT 感测到并由其内部施密特触发器整形形成下降沿，作为中断请求信号发给 CPU。当 CPU 响应中断请求后，首先从 RC0 引脚送出高电平信号，驱动光电耦合器将备用电池接通（用点亮的 LED0 来模拟，如图 7），然后令 LED1~LED7 一起闪烁，发出告警信号，提醒人们及时检修电源。为了简单易行，也可以将演示板上现有的一只微型按钮开关 SW1（SW1 按下时相当于掉电，松开相当于电源恢复正常），用于模拟电源电压跌落时形成的故障信号（见图 7）。

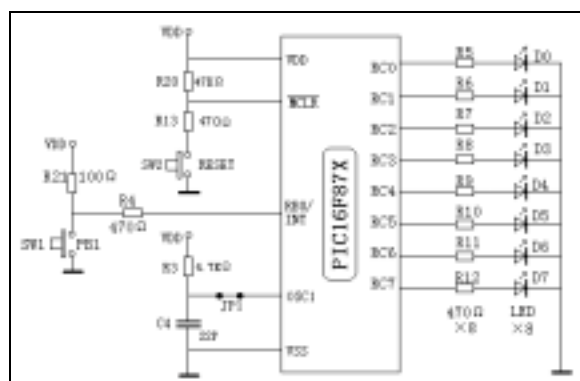


图 7

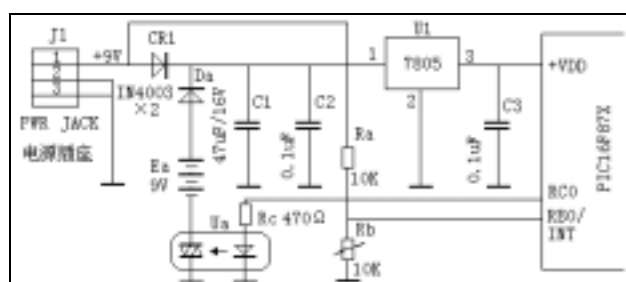


图 8

软件设计思路 驱动 7 只 LED 的显示码的形成，采用查表法获得。在本例程序中的软件设计思路与上例相同，延时时长记为 T1 和 T2，延时时长都用（定时器/计数器 TMR0 模块）硬件手段，并用查询方式实现。延时时长 T1 的计算式为 $256 \times (256 - 2) \times 4 = 260096$ 指令周期 = $260096 \mu s \approx 260ms$ 。其中，前面的 256 是分频比；括号内的 256 是 TMR0 最大的计数范围；括号内的 2 就是每次循环累加计数开始时需要向 TMR0 填写的初始值；后面的 4 是循环利用 TMR0 的次数。同理，延时时长 T2 的计算式为 $256 \times (256 - 6) = 64000$ 指令周期 = $64000 \mu s = 64ms$ 。图 9 为其程序流程图。

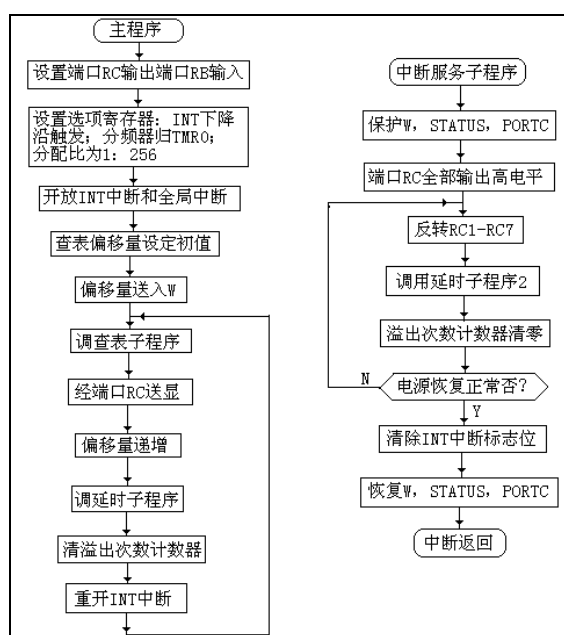


图 9 (a)

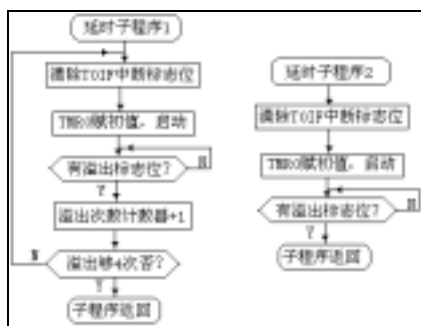


图 9 (b)

3. 端口 RB 电平变化中断 我们知道, RB 端口的 RB4~RB7 引脚具有电平变化中断功能, 那么, 如何有效地利用这个功能呢?

[实验 3] 简易四路抢答器 该抢答器供不多于四个的抢答比赛使用。每个选手的座位前安装一只抢答按钮开关和一只信号灯。主持人的座位前安装一只复原按钮开关、一只蜂鸣器和一只抢答器工作状态指示灯。每当主持人口头发出允许抢答的号令之后, 哪个队先按下座位上的按钮开关, 该座位的信号灯就先被点亮, 同时封锁其他按钮开关的活动。并且熄灭主持人座位上的状态指示灯和发出 3 声类似于电话振铃的提示声, 以“声明”此次抢答动作已经完成。在主持人确认后, 按下复原按钮, 状态指示灯重新点亮, 并且同时发出“笛——笛——”声, 为下一次的抢答作好准备。

硬件电路规划 电路原理图如图 10 所示。电路中的蜂鸣器 FM 是一只带有助音腔的压电陶瓷蜂鸣器, 用于模拟发出报警声的功率放大器和喇叭。在 FM 发声的同时, 灯 D6 也在发光。FM 可以看作是一个电容性负载, 本身不能流过直流电流。发声的原理是, 作用在两个电极极板的电位在发生变化时, 陶瓷材料就发生弯曲, 从而振动空气发出声音。FM 和 4 只按钮开关 SWa~SWd 以及 4 只电阻 Ra~Rd 都是在演示板的基础上额外添加的。由于 RB 端口内部具有上拉电阻, 只要用软件设置其有效, 即可省略在 4 只端口引脚上外接上拉电阻。按钮开关和指示灯与座位的对应关系如表 3 所示。

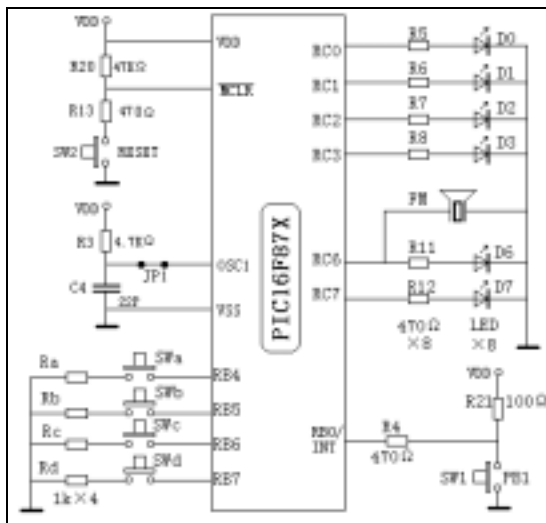


图 10

表 3 按钮开关和指示灯与座位的对应关系

座位	主持人席	座位 1	座位 2	座位 3	座位 4
按钮开关	SW1	SWa	SWb	SWc	SWd
指示灯	D7	D0	D1	D2	D3
蜂鸣器	有	无	无	无	无

软件设计思路 本例的电话振铃的提示声用软件实现。在此设计的提示音是以发声 1 秒、停顿 1 秒的间歇方式，连发 3 声。我们可以用 3 个不同工作频率的振荡器（频率为 500Hz 的低音发生器、频率为 630Hz 的高音发生器和频率为 10Hz 的用于高、低音切换的振荡器）产生出悦耳的铃声。

500Hz 音频对应的周期为 $2000\mu\text{S}$ 。如果利用定时器/计数器 TMRO 模块产生该延迟时间，可以找出恰当的分频比和初始值，分别是 1: 8 和 131。同理，产生 630Hz 音频应该采用的分频比和初始值，分别是 1: 8 和 157。其中，500Hz 计算式为 $T_L=1/500\text{Hz}=0.002\text{s}=2000\mu\text{S}$ ， $1000\mu\text{S} \div 8=125=256-131$ ；630Hz 计算式为 $T_H=1/630\text{Hz}=0.0015873\text{s}=1587.3\mu\text{S}$ ， $793.65\mu\text{S} \div 8 \approx 99=256-157$ 。这两个音频信号再以每秒 10 次的频率轮流切换，从单片机的一条口线（RC6）上送出，每个信号持续的时间是 50ms。那么，对于 500Hz 来说，50ms 之内包含信号周期的个数为 $500\text{Hz} \times 0.05\text{s}=25$ 个，包含（从高到低和从低到高）的电平切换次数为 $25 \times 2=50$ 次。同理，对于 630Hz 来说，电平切换次数为 $31.5 \times 2=63$ 次。因此，发声 1 秒需要轮流调用 500Hz 和 630Hz 的产生程序各 10 次。图 11 是其程序流程图，其中，11（a）为主程序流程图，11（b）为中断服务子程序流程图，11（c）为低音发声子程序流程图，11（d）为高音发声子程序流程图，11（e）为发声 1 秒子程序流程图，11（f）为 3 声发声子程序流程图，图 11（g）为用 TMRO 查询方式延时 1 秒子程序流程图。



图 11（a）

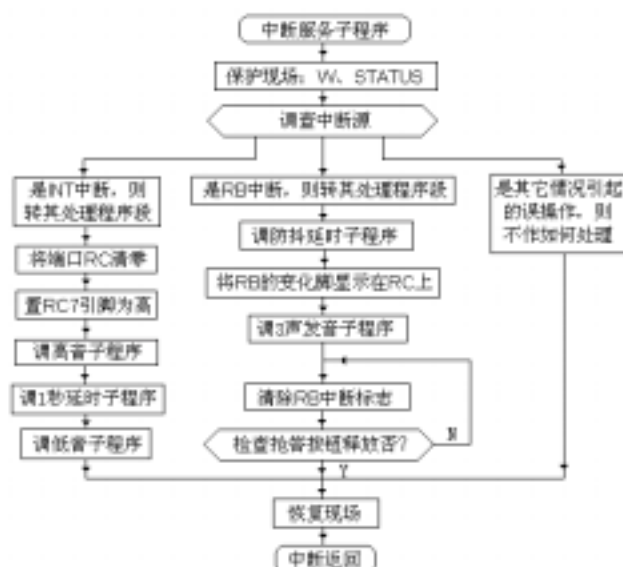


图 11（b）

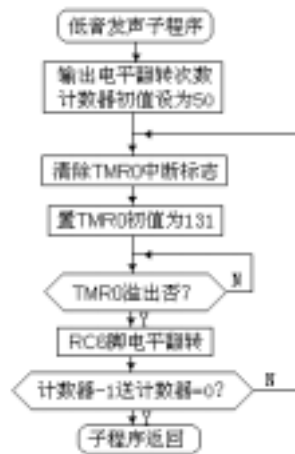


图 11 (c)

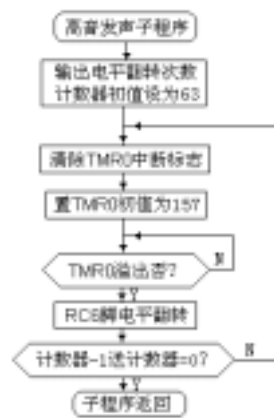


图 11 (d)

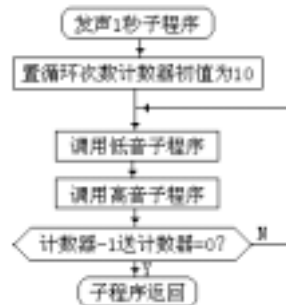


图 11 (e)

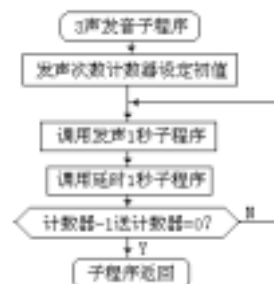


图 11 (f)

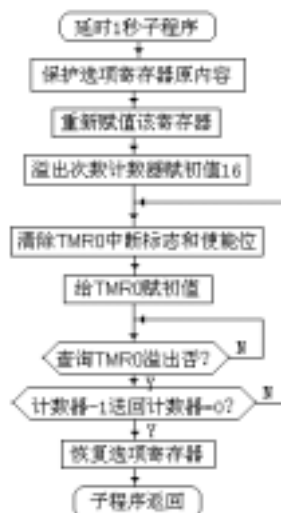


图 11 (g)

在调试一个很复杂的用户程序时，常常希望先挑选出某一段程序或者某一个子程序，单独进行调试。实现这一目的的方法可以有几种，在本例中采用修改程序计数器 PC 值的方法，现介绍如下。

首先建立并且布置如图 12 所示的 MPLAB 工作台，在此打开了源文件、程序存储器、特殊功能寄存器和一个变量观察窗共四个窗口。点击工具栏中的复位按钮时，源文件和程序存储器窗口中的光标自动停留在程序的第一行上，也就是即将被执行的指令。此时的 PC=0000H。

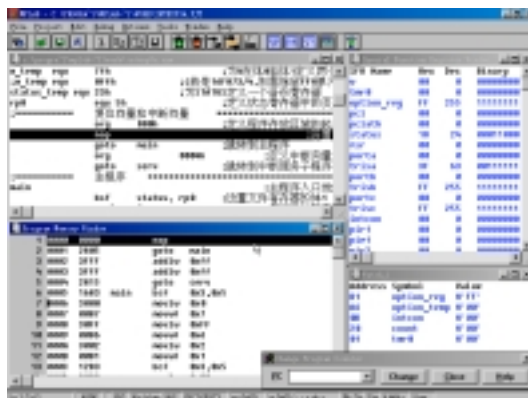


图 12

若想让单片机一开始就执行“延时子程序 DELAY”，则需要先选取菜单命令“Debug>Run>Change Program Counter...”或者直接双击工作台底部状态栏中的“pc:0x00”项，打开如图 13 的“ChangeProgram(更改程序计数器)”对话框，再输入 DELAY 延时子程序的“起始地址”。在如图 14 所示的“Program Memory Window (程序存储器窗口)”中查找，以确定 DELAY 延时子程序的“起始地址”，在此查得的结果应该是 0056H。

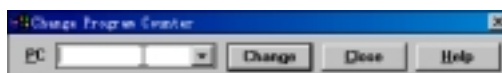


图 13



图 14

如果想把 PC 的值改变为 0056H，就在图 13 的对话框中 PC 的后面的条框中键入“56”，然后点击字符按钮“Change”，随即源文件和程序存储器窗口中的光标自动停留在 DELAY 延时子程序的第一行上，也就是即将被执行的首条指令（如图 15）。



图 15

如果被调试的程序段的起始处具有地址标号（没有的话也可以临时添加一个），借助于地址标号（本例中为“DELAY”）来更改 PC 值会更加方便快捷。方法是点击图 13 对话框中字符 PC 条框后面的下拉箭头，在出现的下拉菜单中，选取“delay”。

如果用户程序很大，地址标号很多，还可以在 PC 字符后的条框中直接键入地址标号“delay”的字头部分或者全部字符。在这一键入操作的进行过程中，随着键入字符个数的增加，下拉菜单中显示的地址标号会不断地发生变化。一旦所要的地址标号出现在视线中，立即用鼠标选中。