MPLAB® C18
C 编译器
入门

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下， Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是 “牢不可破”的。

代码保护功能处于持续发展中。 Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案 （Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip 位于美国亚利桑那州 Chandler 和 Tempe、位于俄勒冈州 Gresham 及位于加利福尼亚州 Mountain View 的全球总部、设计中心和晶圆生产厂均通过了 ISO/TS-16949:2002 认证。公司在 PICmicro® 8 位单片机、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品方面的质量体系流程均符合 ISO/TS-16949:2002。此外，Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

目录

MPLAB® C18 C 编译器入门

前言

客户须知

所有文档均会过时，本文档也不例外。 Microchip 的工具和文档将不断演变以满足客户的需求，因此实际使用中有些对话框和 / 或工具说明可能与本文档所述之内容有所不同。请访问我们的网站（www.microchip.com）获取最新文档。

文档均标记有 “DS”编号。该编号出现在每页底部的页码之前。 DS 编号的命名约定为 “DSXXXXXA”，其中 “XXXXX”为文档编号，“A”为文档版本。

欲了解开发工具的最新信息，请参考 MPLAB® IDE 在线帮助。从 Help （帮助）菜单选择 Topics （主题），打开现有在线帮助文件列表。

简介

本文档旨在帮助嵌入式系统工程师快速学会使用 Microchip 的 MPLAB® C18 C 编译器。通过配合使用 MPLAB C18 C 编译器、 MPLINK™ 链接器、 MPLAB IDE 和 PIC18 PICmicro MCU，可快速开发出 PICmicro® 单片机应用。关于本文所讲述编译器特征的详细介绍，请参阅 《MPLAB® C18 C 编译器用户指南》（DS51288J_CN）。

本文所提供信息适用于有单片机使用背景、理解 8 位单片机基本概念且熟悉 C 编程语言的工程师或学生。

本章内容包括：

- 文档编排
- 本指南使用的约定
- 推荐读物
- Microchip 网站
- 开发系统变更通知客户服务
- 客户支持

文档编排

- **第1章 概述**——对MPLAB C18 C编译器及其组件，以及它与MPLAB集成开发环境（IDE）的集成方面的概述。
- **第 2 章 安装**——一步步地指导 MPLAB C18 C 编译器的安装过程。
- **第3章 项目的基本操作及MPLAB IDE配置**——通过MPLAB项目和MPLAB SIM软件模拟器介绍了使用 MPLAB C18 时的 MPLAB IDE 设置，并提供了运行本指南中示例和应用程序所需 MPLAB IDE 设置方面基本知识的参考信息。
- **第 4 章 简单入门程序**——提供了简单的示例，从最简单的“Hello, world!”入门程序开始，接着提供了点亮连接到 PIC18 单片机的 LED 的程序。
- **第 5 章 特性**——概括介绍了 MPLAB C18 编译器的总体功能，提供了关于优化的代码演示，并举例说明了如何使用 MPLAB Watch （观察）窗口来查看数据元素和结构。
- **第6章 架构**——讲述了PIC18架构，以及MPLAB C18编译器有别于其他C编译器的特殊功能。
- **第7章 疑难解答**——列出了常见的错误消息和技术问题，并提供了处理这些问题的答案和指导。

本指南使用的约定

本手册采用以下文档约定：

文档约定

说明	表示	示例
Arial 字体：		
斜体字符	参考书目	*MPLAB® IDE User's Guide*
	需强调的文字	*... 仅有*的编译器 ...
首字母大写	窗口	Output 窗口
	对话框	Settings 对话框
	菜单选项	选择 Enable Programmer
引用	窗口或对话框中的域名	“Save project before build”
带右尖括号且有下划线的斜体文字	菜单路径	*File>Save*
粗体字	对话框按钮	单击 **OK**
	选项卡	单击 **Power** 选项卡
尖括号 < > 括起的文字	键盘上的按键	按 <Enter>，<F1>
Courier 字体：		
常规 Courier	源代码示例	`#define START`
	文件名	`main.c`
	文件路径	`c:\mcc18\h`
	关键字	`_asm, _endasm, static`
	命令行选项	`-Opa+, -Opa-`
	位值	`0, 1`
	常数	`0xFF, ‘A’`
斜体 Courier	可变参数	*`file`*`.o`，其中 *`file`* 可以是任一有效文件名
`0b`*`nnnn`*	二进制数， *n* 是其中一位	`0b00100, 0b10`
`0x`*`nnnn`*	十六进制数， *n* 是其中一位	0xFFFF, 0x007A
方括号 []	可选参数	`mcc18 [options]` *`file`* `[options]`
花括号和竖线：{|}	选择互斥参数； “或”选择	`errorlevel {0|1}`
省略号 ...	代替重复文字	`var_name [, var_name...]`
	表示由用户提供的代码	`void main (void) { ... }`

推荐读物

PIC18 开发参考读物

要了解更多关于编译器的函数库和预编译目标文件、 MPLAB IDE 及其他工具使用方面的信息，请阅读以下推荐读物。

MPLAB-C18-README.txt

关于使用 MPLAB C18 C 编译器的最新信息，请阅读本软件自带的 MPLAB-C18-README.txt 文件（ASCII 文本）。此 readme 文件包含了本文档可能未提供的更新信息。

Readme for XXX.txt

需要其他 Microchip 工具的最新信息 （MPLAB IDE 和 MPLINK 链接器等），请阅读软件自带的相关 readme 文件 （ASCII 文本文件）。

MPLAB® C18 C 编译器用户指南 （DS51288J_CN）

一个综合指南，讲述了针对 PIC18 器件设计的 Microchip MPLAB C18 C 编译器的使用及特征。

PIC18 Configuration Settings Addendum （DS51537）

给出了 MPLAB C18 C 编译器 `#pragma config` 伪指令和 MPASM `CONFIG` 伪指令支持的 Microchip PIC18 器件的配置位设置。

MPLAB C18 C 编译器函数库 （DS51297F_CN）

关于 MPLAB C18 函数库和预编译目标文件的参考指南。列出了随 MPLAB C18 C 编译器提供的所有库函数，并详细描述了这些库函数的使用。

MPLAB® IDE 用户指南 （DS51519A_CN）

介绍如何安装 MPLAB IDE 软件及如何使用 IDE 创建项目并烧写器件。

MPASM™ 汇编器、MPLINK™ 目标链接器和 MPLIB™ 目标库管理器用户指南 （DS33014J_CN）

这个用户指南描述了如何使用 Microchip 的 PICmicro 单片机汇编器 （MPASM ）、链接器 （MPLINK）和库管理器 （MPLIB）。

PICmicro® 18C 单片机系列参考手册 （DS39500A_CN）

重点介绍 PIC18 系列器件。说明了 PIC18 系列的架构和外设模块的工作原理，但没有涉及到每个器件的具体细节。

PIC18 器件数据手册

讲述 PIC18 器件工作和电气特性的数据手册。

要获得上述任何文档，请访问 Microchip 的网站 （www.microchip.com），获得 Adobe Acrobat （.pdf）格式的文档。

C 语言及其他参考书

有许多有关 C 语言一般知识的参考书和教材，其中一些资料还涉及到使用 Microchip 单片机开发嵌入式应用。

American National Standard for Information Systems – *Programming Language – C*. American National Standards Institute (ANSI), 11 West 42nd. Street, New York, New York, 10036.

此标准规定了用 C 语言编写程序的格式，并对 C 程序进行了解释。其目的是提高 C 程序在多种计算机系统上的可移植性、可靠性、可维护性及执行效率。

Harbison, Samuel P. and Steele, Guy L., *C: A Reference Manual,* Fourth Edition. Prentice-Hall, Englewood Cliffs, New Jersey 07632.

详细地讲述了 C 编程语言。这本书是一本权威性的参考手册，它对 C 语言、运行时库以及 C 编程的风格都进行了完整的描述，C 编程强调正确性、可移植性和可维护性。

Huang, Han-Way. *PIC® Microcontroller: An Introduction to Software & Hardware Interfacing*. Thomson Delmar Learning, Clifton Park, New York 12065.

对 Microchip PIC18 单片机系列进行了全面介绍，包括 PIC 单片机外设功能的编程和接口。这本书可用作大学教科书，其中使用了 PIC 单片机汇编语言和 MPLAB C18 C 编译器。

Kernighan, Brian W. and Ritchie, Dennis M. *The C Programming Language*, Second Edition. Prentice Hall, Englewood Cliffs, New Jersey 07632.

对由 ANSI 标准定义的 C 语言进行了简明阐述。对于 C 程序员来说是一本出色的参考书。

Kochan, Steven G. *Programming In ANSI C*, Revised Edition. Hayden Books, Indianapolis, Indiana 46268.

学习 ANSI C 的另一本出色的参考书，用作大学教材。

Peatman, John B. *Embedded Design with the PIC18F452 Microcontroller*, First Edition. Pearson Education, Inc., Upper Saddle River, New Jersey 07458.

重点介绍 Microchip 公司的 PIC18FXXX 系列单片机以及如何编写优化的应用代码。

Van Sickle, Ted. *Programming Microcontrollers in C*, First Edition. LLH Technology Publishing, Eagle Rock, Virginia 24085.

讲述单片机 C 语言编程的基本原理。

Standards Committee of the IEEE Computer Society – *IEEE Standard for Binary Floating-Point Arithmetic*. The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, New York 10017.

这个标准描述了 MPLAB C18 采用的浮点数格式。

应用笔记

Microchip 提供了丰富的应用笔记，许多应用笔记都与 MPLAB C18 C 编译器兼容。下面列出了其中的部分应用笔记。请查看 Microchip 网站中最新发布的应用笔记。

- AN953 Data Encryption Routines for the PIC18
- AN851 A FLASH Bootloader for PIC16 and PIC18 Devices
- AN937 Implementing a PID Controller Using a PIC18 MCU
- AN914 Dynamic Memory Allocation for the MPLAB C18 C Compiler
- AN991 Using the C18 Compiler and the MSSP to Interface I^2C™ EEPROMs with PIC18 Devices
- AN878 PIC18C ECAN C Routines
- AN738 PIC18C CAN Routines in 'C'
- AN930 J1939 C LIbrary for CAN-Enabled PICmicro® MCUs

设计中心

Microchip 网站 www.microchip.com 中包含许多设计中心，提供针对某个具体行业的指导信息。这些设计中心中包含源代码、应用笔记、网络资源和针对具体应用推荐的 Microchip MCU。

下面是所提供的部分设计中心：

- Microchip 产品入门
- 汽车电子解决方案
- 高引脚数 / 高存储容量单片机
- KEELOQ® 鉴定解决方案
- 电池管理解决方案
- LCD 解决方案
- 网络连接解决方案
  - 物理协议：CAN、 LIN 和 USB
  - 无线协议：ZigBee™、红外和 rfPIC®
  - 互联网协议：TCP/IP
- 低功耗解决方案
- 机电一体化设计
- 电机控制解决方案
- 家电解决方案
- 全球最小的单片机
- 公用仪表解决方案
- EMC 设计
- 3V 系统设计
- 16 位单片机解决方案

MICROCHIP 网站

Microchip 在全球网站 www.microchip.com 上提供在线支持。用户可以在网站上很方便地获得文件和信息。用户可以使用互联网浏览器访问网站。该网站包含以下信息：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南和硬件支持文档、最新的软件版本和归档软件
- **一般技术支持**——常见问题 （FAQ）解答、技术支持请求、在线讨论组以及 Microchip 顾问计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新的 Microchip 新闻、研讨会与活动安排表、 Microchip 销售办事处、代理商及工厂代表列表

开发系统变更通知客户服务

Microchip 的客户通知服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请登录 Microchip 网站 www.microchip.com，点击 “变更通知客户（Customer Change Notification）”服务并按照注册说明完成注册。

开发系统产品的分类如下：

- **编译器**——Microchip C 编译器及其他语言工具的最新信息，包括 MPLAB C18 和 MPLAB C30 C 编译器、MPASM 和 MPLAB ASM30 汇编器、MPLINK 和 MPLAB LINK30 目标链接器，以及 MPLIB 和 MPLAB LIB30 目标库管理器。
- **仿真器**——Microchip 在线仿真器的最新信息，包括 MPLAB ICE 2000 和 MPLAB ICE 4000。
- **在线调试器**——Microchip 在线调试器 MPLAB ICD 2 的最新信息。
- **MPLAB® IDE**——关于支持开发系统工具的 Windows® 集成开发环境 Microchip MPLAB IDE 的最新信息，主要针对 MPLAB IDE、 MPLAB SIM 软件模拟器、MPLAB 项目管理器以及一般编辑和调试功能。
- **编程器**——Microchip 编程器的最新信息，包括 MPLAB PM3 器件编程器和 PIC-START® Plus 开发编程器。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师 （FAE）
- 技术支持

客户应联系其代理商、代表或应用工程师 （FAE）寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过 http://support.microchip.com 获得网上技术支持。

注：

第 1 章 概述

1.1 简介

本章介绍用于嵌入式系统编程的软件工具。讲述了编译器和汇编器的功能及区别，以及 C 语言的优势。还介绍了 MPLAB C18 的目录结构、各种语言工具可执行文件和执行流程。

本章内容包括：

- 嵌入式系统编程工具
- 系统要求
- 目录
- 关于语言工具
- 执行流程

1.2 嵌入式系统编程工具

1.2.1 MPLAB C18 C 编译器

MPLAB C18 C编译器是在PC机上运行的交叉编译器，生成可由Microchip PIC18XXXX系列单片机执行的代码。与汇编器一样， MPLAB C18 编译器将人可理解的语句翻译为单片机可执行的 “1”和 “0”。而与汇编器不同的是，编译器不将机器助记符一对一地翻译为机器码。

MPLAB C18 接受标准 C 语句，如 `if(x==y)` 和 `temp=0x27`，并将其转换为 PIC18XXXX 机器码。编译器在这个过程中融合了很多 “智能”功能。当代码中一个 C 函数采用的子程序也被其他 C 函数使用时，编译器将优化这段代码。编译器能重新排列代码，删除不会执行到的代码，在多个函数间共用公共代码段，且可识别到使用效率低的数据和寄存器并优化对它们的访问。

代码采用标准的 ANSI C 符号编写。源代码被编译为程序代码块和数据块，然后 “链接”到其他的代码和数据块，再存放到 PIC18XXXX 单片机的各存储区中。这个过程称为 “build”，且在编写、测试和调试代码的程序开发过程中经常会进行多次 build。通过使用 “make”程序可使这个过程更为 “智能化”，它仅对上次编译后项目中更改过的 C 源文件调用编译器，因此可缩短项目编译的时间。

可通过命令行调用 MPLAB C18 编译器及其相关工具 （如链接器和汇编器）来生成 .HEX 文件，可将这种文件烧写到 PIC18XXXX 器件中。也可从 MPLAB IDE 中调用 MPLAB C18 及其他工具 。 MPLAB 图形用户界面作为一个单一的环境，可在其中为嵌入式应用编写、编译和调试代码。

MPLAB 对话框和项目管理器完成编译器、汇编器和链接器的大部分具体工作，因而用户可将主要精力集中在编写和调试应用程序的任务上。

由于 MPLAB C18 编译器使用标准 C 语言，因而使得嵌入式系统应用的开发更为容易。有许多教授 C 语言的参考书，本文档的**前言 “推荐读物”**中列出了一些这样的参考书。本指南假定读者已经了解关于 C 编程的基本知识。 C 语言的优势在于，它使用广泛，可在不同的架构之间移植，有许多相关的参考书和教材，且比汇编语言更易于维护和扩展。另外， MPLAB C18 可为 PIC18XXXX 单片机编译出极为高效率的代码。

1.2.2 MPASM 交叉汇编器和 MPLINK 链接器

通常情况下，可使用交叉汇编器和交叉编译器来为项目编写代码。MPASM 是 MPLAB IDE 的一个组件，它与 MPLINK 配合工作，MPLINK 将汇编语言代码段与 MPLAB C18 C 编译器生成的代码链接起来。

汇编语言程序适用于要求非常快运行速度或要求在严格定义的时间内运行的小代码段。

注： 尽管编译器生成代码的执行时间可能与使用汇编语言生成的代码几乎相同，但由于编译是一个翻译转换过程，经过这一翻译转换过程后才能生成可直接从汇编语言生成的机器码，因此编译器生成的代码不可能比汇编语言代码执行速度快。

1.2.3 其他工具

本指南中，利用 MPLAB IDE 的图形化用户界面和开发环境，通过 MPLAB C18 编译器来编写和编译代码示例。《MPLAB® IDE v6.xx 快速入门指南》中提供了教程并一步步指导和帮助您了解和学习使用 MPLAB IDE。其他有关汇编语言和链接器的信息，请参阅 《MPASM™ 汇编器、 MPLINK™ 目标链接器和 MPLIB™ 目标库管理器用户指南 》。

Microchip 的 PICDEM™ 2 Plus 开发板可使用 PIC18F452 作为主单片机，且本文档提供的示例可在此开发板上运行，使板上的 LED 闪烁。

也可使用 MPLAB ICD 2 来对 PICDEM 2 Plus 开发板上的 PIC18F452 编程及调试程序。但运行本指南中的示例可不需要这些硬件工具。可在免费的 MPLAB IDE 中，使用的 MPLAB SIM 的 PIC18XXXX 模拟器来进行调试。

1.3 系统要求

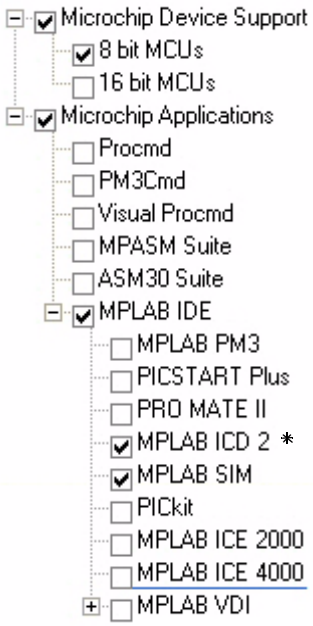
使用 MPLAB C18 和 MPLAB IDE 的建议系统要求为：

- Intel® Pentium® PC，安装 Microsoft® 32 位 Windows 操作系统（Windows 2000、Windows XP 家庭版或 Windows XP 专业版）
- 大约 250 MB 硬盘空间
- 针对本指南中某些示例的可选硬件工具：
  - PICDEM 2 Plus 开发板和电源
  - MPLAB ICD 2 在线调试器 （需要串行或 USB 连接）

尽管 MPLAB C18 可独立于 MPLAB IDE 单独使用，但本指南举例说明了 MPLAB C18 在 MPLAB 集成开发环境中的使用。应该在安装 MPLAB C18 之前安装 MPLAB IDE。MPLAB IDE 的默认安装可能有预先设定的选择。当为使用 MPLAB C18 而安装 MPLAB IDE 时，至少要选择如下组件 （参见图 1-1）：

- MPLAB IDE 器件支持
  - 8 位 MCU
- Microchip 应用程序
  - MPLAB IDE
  - MPLAB SIM
  - MPASM 工具包（这个工具包也可随 MPLAB C18 安装，因此安装 MPLAB IDE 时不必安装它）

图 1-1： MPLAB® IDE 安装菜单



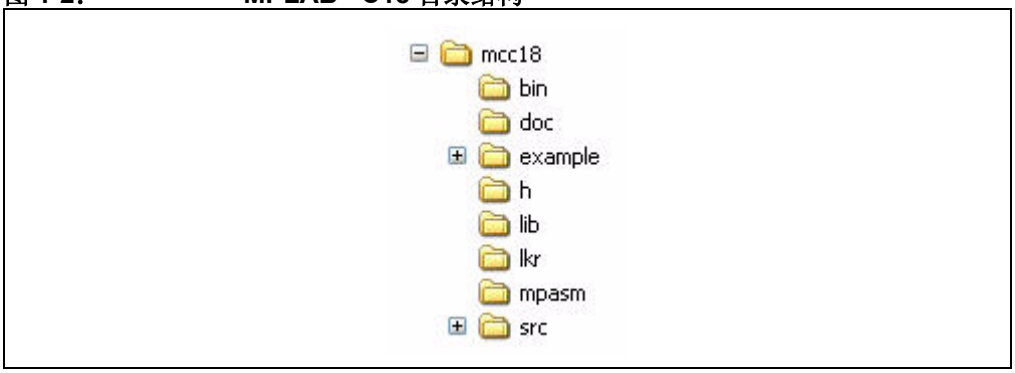
* 可选的组件。如果要使用 MPLAB ICD 2 来进行编程和调试，要选择这个组件。

1.4 目录

可将 MPLAB C18 安装到 PC 的任何目录下。默认安装目录为 C:\mcc18。

图 1-2 示出了 MPLAB C18 典型安装的目录结构：

图 1-2： MPLAB® C18 目录结构



MPLAB C18 安装目录包含编译器、汇编器和链接器的 readme 文件。表 1-1 对其子目录的内容进行了描述。

表 1-1： MPLAB C18 子目录描述

目录	描述
bin	包含编译器和链接器的可执行文件。将在**第 1.5 节 “关于语言工具”**中对这些可执行文件进行更详细的描述。
doc	包含 C18 C 编译器的文档。仅当选择安装文档时，才会安装文档（参见**第 2.2.5 节 “选择组件”**和图 2-5）。
example	包含范例应用程序，帮助用户开始学习使用 MPLAB C18，其中包括本文档中使用的示例。这些代码示例可能和**第 4 章 “简单入门程序”**中使用的代码略微不同。
h	包含标准 C 函数库的头文件和所支持 PICmicro® 单片机的特定处理器函数库的头文件。
lib	包含标准 C 函数库（clib.lib 或 clib_e.lib）、特定处理器的函数库（p18*xxxx*.lib 或 p18*xxxx*_e.lib，其中 *xxxx* 是具体的器件型号）和启动模块（c018.o、c018_e.o、c018i.o、c018i_e.o、c018iz.o 和 c018iz_e.o）。
lkr	包含供 MPLAB C18 使用的链接描述文件。
mpasm	包含 MPASM 汇编器以及 MPLAB C18 所支持器件的汇编头文件（p18xxxx.inc）。
src	包含标准 C 函数库、特定处理器函数库和启动模块的 C 和汇编源代码文件。其中包含针对扩展模式和传统（非扩展）模式的子文件夹。

1.5 关于语言工具

MPLAB C18 编译器安装目录下的 bin 和 mpasm 子目录包含 MPLAB C18 、MPASM 汇编器和 MPLINK 链接器的可执行文件。一般情况下，这些可执行文件中的大多数在编译过程中自动运行。 MPLAB IDE 项目管理器需要知道主编译器、汇编器、链接器和库可执行文件的安装位置 （由 *Project>LanguageToolLocations* 设置）。其中某些工具的简要描述参见表 1-2。

表 1-2： MPLAB® C18、 MPASM™ 汇编器和 MPLINK™ 链接器可执行文件

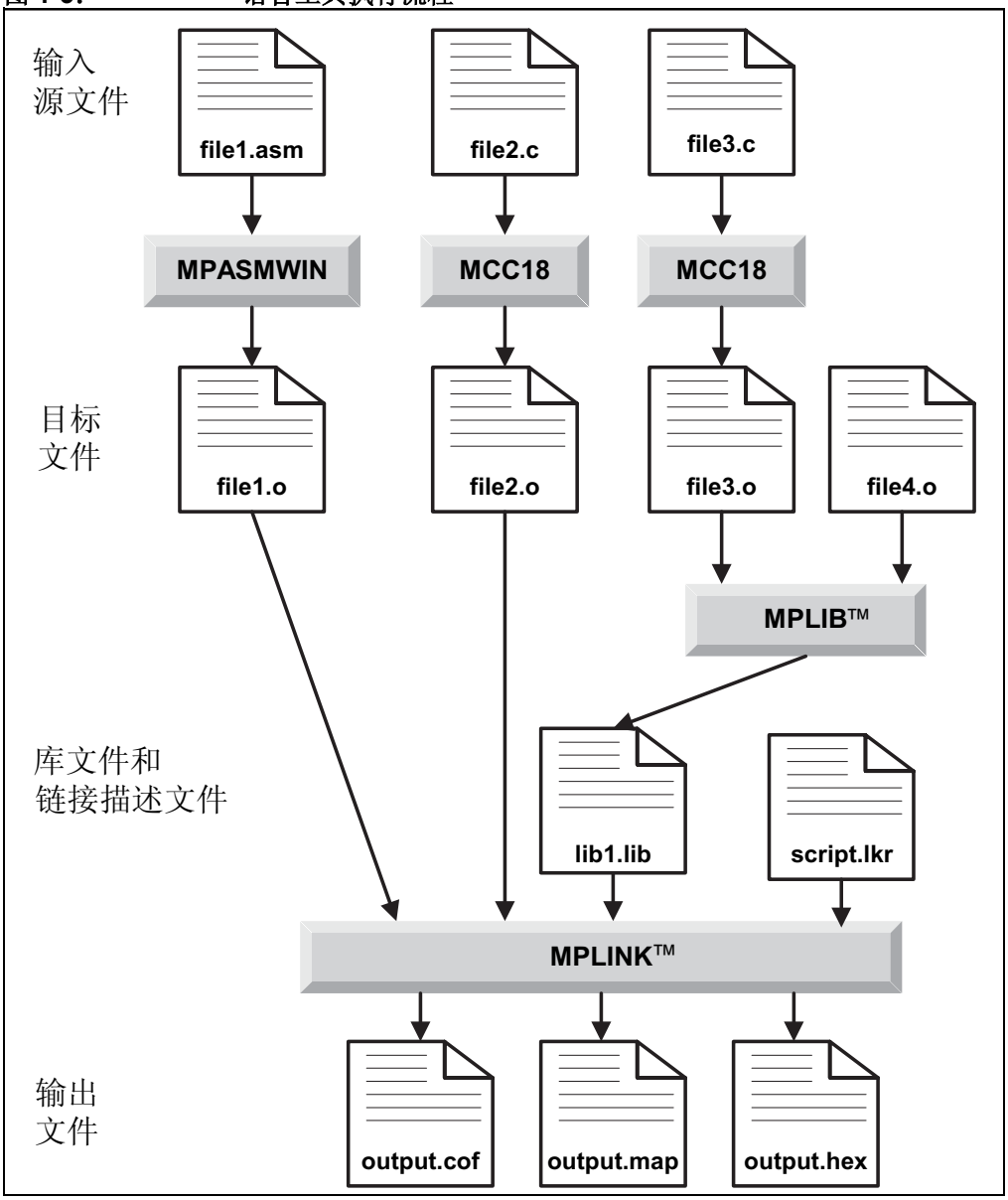
可执行文件	描述
mcc18.exe	编译器 shell。它以 C 文件 （如 file.c）作为输入，调用扩展模式和非扩展模式编译器可执行文件。
mplink.exe	链接器的驱动程序。它以链接描述文件 （如 18F452.lkr）、目标文件和库文件作为输入，并把这些文件传递给 _mplink.exe。然后它会把_mplink.exe输出的COFF文件传递给 mp2hex.exe。
_mplink.exe	链接器。它输入链接描述文件、目标文件和库文件，输出公共目标文件格式 （Common Object File Format， COFF）可执行文件 （如 file.out 或 file.cof）。 COFF 文件是对输入目标文件及从函数库引用的目标文件的数据和代码进行地址分配的结果。 _mplink.exe 也可以选择生成一个映射文件 （如 file.map），这种映射文件包含关于数据和代码分配的详细信息。
mp2hex.exe	把 COFF 文件转化为 hex 文件的文件转换器。 hex 文件是 PICmicro® 单片机编程器（如 PICSTART® Plus 或 PRO MATE® II）可读的文件格式。mp2hex.exe 输入由 _mplink.exe 生成的 COFF 文件，输出 hex 文件 （如 file.hex）。
mplib.exe	库管理器。它允许创建和管理库文件 （如 file.lib），而库文件则充当目标文件的存档文件。库文件用于将目标文件组织成可重用的代码库。
mpasmwin.exe	Windows® 汇编器可执行文件，它以汇编源文件 （如 file.asm）作为输入，输出 COFF 文件（如 file.o）或 hex 文件和 COD 文件 （如 file.hex 和 file.cod）。汇编源文件可能包含汇编头文件 （如 p18f452.inc），汇编头文件也包含汇编源代码。

关于语言工具的更详细信息，包括其命令行用法，请参考 《MPLAB® C18 C 编译器用户指南》（DS51288J_CN）和 《MPASM™ 汇编器、 MPLINK™ 目标链接器和 MPLIB™ 目标库管理器用户指南 》（DS33014J_CN）。

1.6 执行流程

图 1-3 举例说明了语言工具的执行流程。

图 1-3： 语言工具执行流程



在上面的示例中，由 MPLAB C18 编译两个 C 文件 `file2.c` 和 `file3.c`，由 MPASM 汇编汇编文件 `file1.asm`，生成了目标文件 `file1.o`、`file2.o` 和 `file3.o`。

预编译的目标文件 `file4.o` 和 `file3.o` 形成名为 `lib1.lib` 的库文件。最后，链接器将其余的目标文件与库文件组合在一起。

MPLINK 还输入链接描述文件 `script.lkr`。 MPLINK 生成输出文件 `output.cof`、`output.map` 和 HEX 文件 `output.hex`。

第 2 章 安装

2.1 简介

在安装 MPLAB C18 之前，应该先在 PC 上安装 MPLAB IDE。 MPLAB IDE 的安装文件可从 CD-ROM 上获得或从 www.microchip.com 免费下载。 MPLAB IDE 的项目管理器和 MPLAB SIM 软件模拟器都是 MPLAB IDE 的组件，本指南中广泛使用了这两个组件和内置的调试器 （参见**第 1.3 节 “系统要求”**）。

本章详细讲述如何安装 MPLAB C18。有些情况下需要卸载软件，为此也提供了如何卸载软件的指示说明。

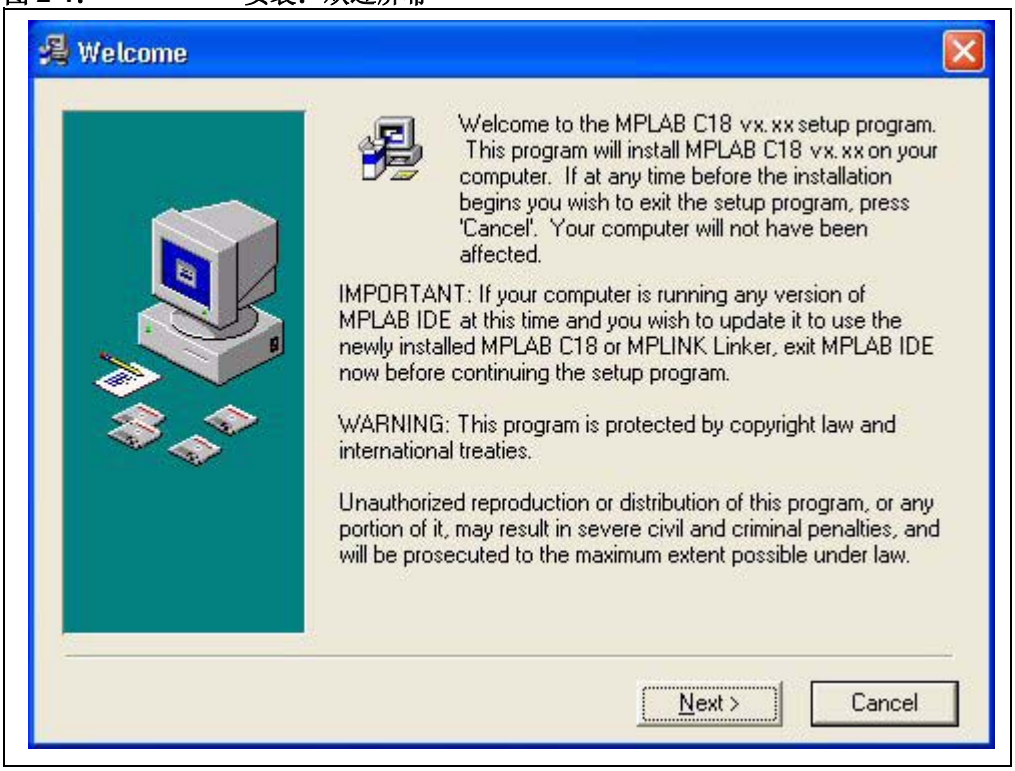
2.2 安装 MPLAB C18

要安装 MPLAB C18，请运行 CD-ROM 中的安装程序。如果要升级 MPLAB C18，请运行从 Microchip 网站上下载的升级安装程序。在整个安装过程中，一系列的对话框将一步步地指导您如何进行安装。

2.2.1 欢迎

首先会出现一个欢迎屏幕 （图 2-1），告知安装程序要安装的 MPLAB C18 的版本号。

图 2-1： 安装：欢迎屏幕



点击 **Next** 继续安装。

2.2.2 许可协议

接着出现 MPLAB C18 许可协议。阅读协议，然后点击 “I Accept”。

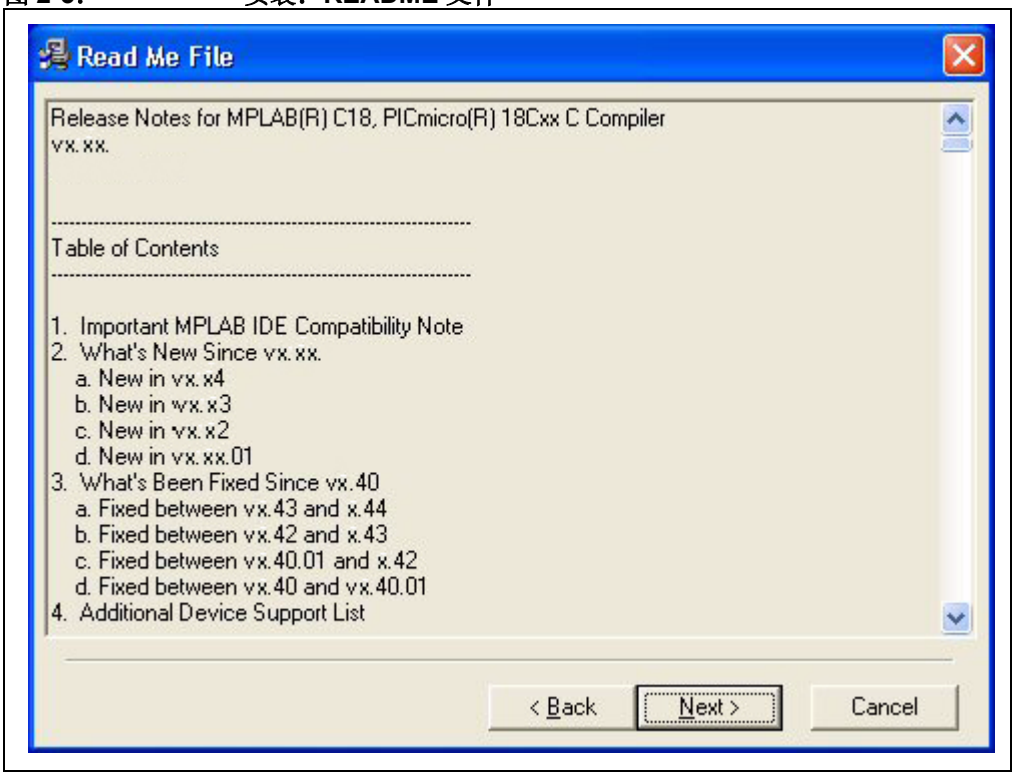
图 2-2： 安装：许可协议

接受许可协议后，点击 **Next>** 继续。

2.2.3 Readme 文件

随后将显示 MPLAB C18 的 readme 文件 （图 2-3）。这个文件包含关于此版本 MPLAB C18 的重要信息，如支持的器件、新特性、已知问题及变通解决方案。每个版本的 readme 文件有所不同，看起来与下图中类似，但内容不同。

图 2-3： 安装：README 文件



看完 readme 文件后，点击 **Next** 继续安装。

2.2.4 选择安装目录

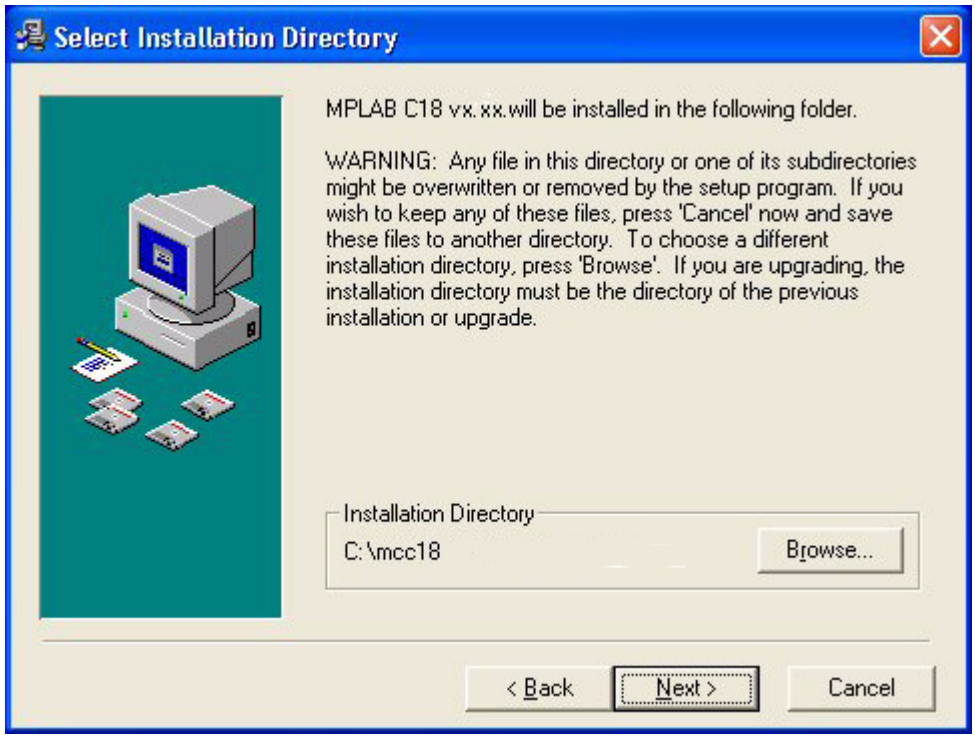
选择要将 MPLAB C18 安装到哪个目录中。

当第一次安装 MPLAB C18 时，默认的安装目录是 C:\mcc18，如图 2-4 所示。如果要安装到其他目录中，请点击 **Browse**。

如果是安装升级版本，安装程序则会将默认安装目录设置为上次安装时的目录。在升级时，所选的安装目录必须是上次安装或升级时的安装目录。

> **注：** 在安装过程中，安装目录和其子目录中的文件可能会被覆盖或删除。若要保存任何文件，例如要保存上次安装修改过的链接描述文件或库源代码，则可在继续进行安装前将这些文件复制到安装目录外的目录中去。

图 2-4： 安装：选择安装目录



点击 **Next**。

> **注：** 安装升级版本时，如果不安装到原有版本所在的目录中，将显示错误消息“No previous installation”。

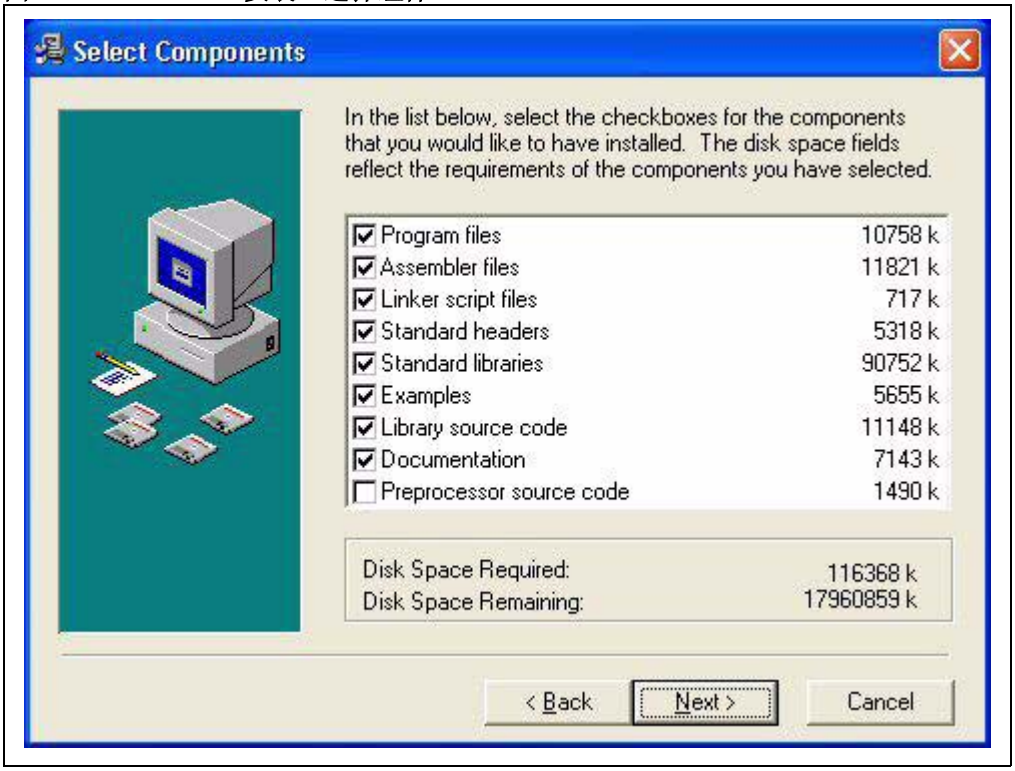
2.2.5 选择组件

选中相应的复选框，选择需要安装的组件（图 2-5）。表 2-1 详细描述了可选择组件。

注： MPASM 和 MPLINK 随 MPLAB IDE 免费提供。这两个工具也可随 MPLAB C18 编译器安装。为确保所有这些工具互相兼容，应该使用 MPASM 和 MPLINK 随 MPLAB C18 编译器提供的版本。

也会随 MPLAB IDE 提供 MPASM 的链接描述文件。**当使用 MPLAB C18 编译器时，要确保使用随 MPLAB C18 安装的链接描述文件，而不是随 MPLAB IDE 安装的链接描述文件。**随 MPLAB C18 提供的链接描述文件具有某些特殊编译器伪指令。

图 2-5： 安装：选择组件



点击 **Next>** 继续。

注： 并非所有的安装都包含文档。对于升级程序和某些从网上下载的安装程序，文档是单独发布的。

表 2-1： MPLAB C18 软件组件

组件	描述
程序文件	编译器和链接器的可执行文件。用户应该选择安装这个组件，除非是仅升级辅助文件（不升级可执行文件）。
汇编器文件	MPASM™ 汇编器以及 MPLAB C18 所支持器件的汇编头文件（p18xxxx.inc）。
链接描述文件	MPLINK™ 链接器需要的文件。每个支持的 PICmicro 单片机都有一个这样的文件。每个文件为处理器提供了一个默认的存储配置，并指引链接器在处理器的存储器中分配代码和数据。 **注：** 这些链接描述文件有别于随 MPLAB IDE 提供的链接描述文件，是专门为 MPLAB C18 设计的。建议安装这个组件。
标准头文件	这些是标准 C 函数库和特定处理器函数库的头文件。建议安装这个组件。
标准函数库	这个组件包含标准 C 函数库、特定处理器函数库和启动模块。请参阅《MPLAB® C18 C 编译器函数库》（DS51297F_CN）和《MPLAB® C18 C 编译器用户指南》（DS51288J_CN），以获取更多关于函数库和启动模块的信息。由于大多数典型的程序都使用函数库和一个启动模块，推荐用户安装这个组件。
示例	这些是范例应用程序，其中包括本文档中讲述的示例，用来帮助用户学习使用 MPLAB C18。
库源代码	标准 C 函数库和特定处理器函数库的源代码。可安装这个组件来查看源代码，以及修改或重建函数库。
预处理器源代码	这是预处理器的源代码，供有兴趣的用户参考。

2.2.6 配置选项

在接下来的 Configuration Options （配置选项）对话框 （图 2-6）中，选择所需要的 MPLAB C18 配置选项。

图 2-6： 安装：配置选项

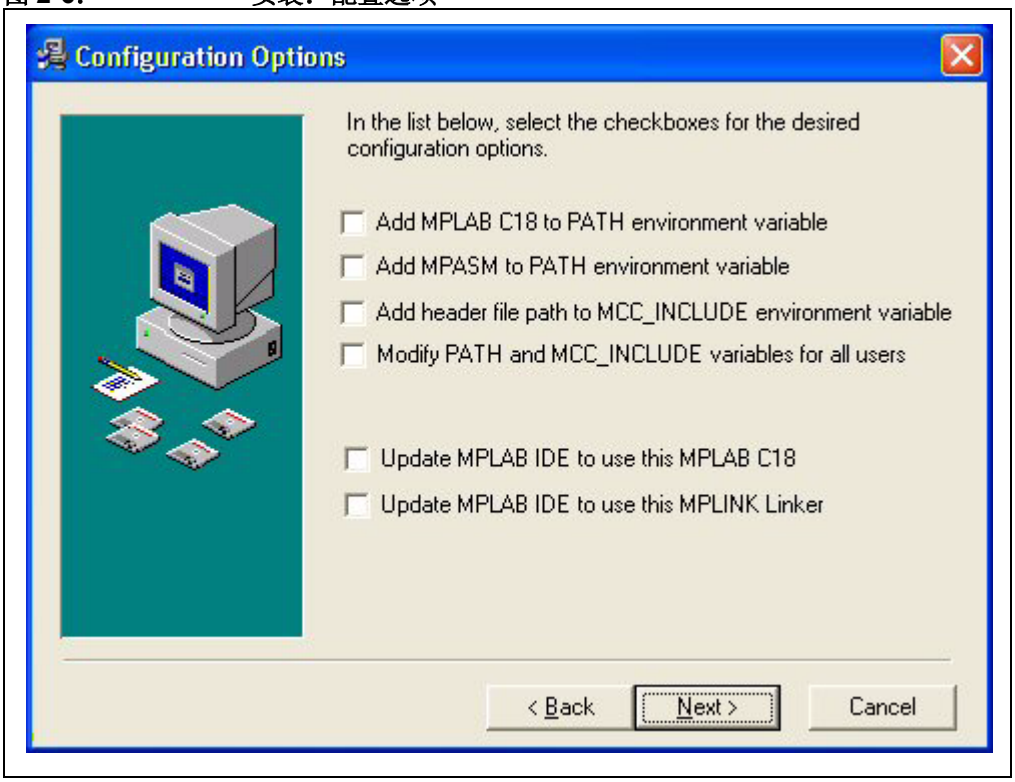


表 2-2 对各配置选项进行了详细描述。点击 **Next**。

表 2-2： MPLAB C18 配置选项

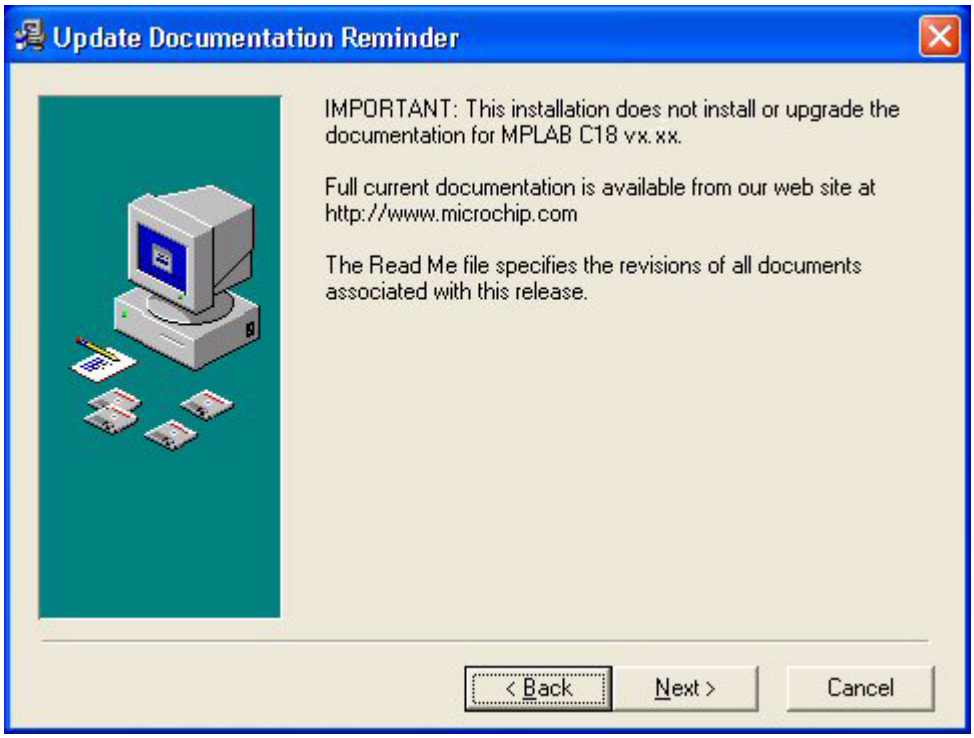
配置	描述
Add MPLAB C18 to PATH environment variable	将 MPLAB C18 可执行文件 （mcc18.exe）和 MPLINK 链接器可执行文件 （mplink.exe）的路径添加到 PATH 环境变量的开头。这样用户就能从任何目录、在命令 shell 提示符下启动 MPLAB C18 编译器和 MPLINK 链接器。不管是否已经包含了目录，都要将这个选项放在路径的最前面。
Add MPASM to PATH environment variable	将 MPASM 可执行文件 （mpasmwin.exe）的路径添加到 PATH 环境变量的开头。这样用户就能从任何目录、在命令 shell 提示符下启动 MPASM 汇编器。不管是否已经包含了目录，都要将这个选项放在路径的最前面。
Add header file path to MCC_INCLUDE environment variable	将 MPLAB C18 头文件目录的路径添加到 MCC_INCLUDE 环境变量的开头。 MCC_INCLUDE 是一个由分号隔开的目录列表。如果 MPLAB C18 在由 -I 命令行选项指定的目录列表中找不到头文件，那么它将在 MCC_INCLUDE 目录列表中搜索头文件。选择该配置选项表明在包含标准头文件时，用户不必使用 -I 命令行选项。如果该变量不存在，就创建它。
Modify PATH and MCC_INCLUDE variables for all users	此选项只在当前用户以管理员身份登录 Windows NT 或 Windows 2000 计算机时出现。如果选择这个配置，则对前三个选项中指定的变量所做的修改会影响到所有的用户，否则只有当前用户的变量会受影响。
Update MPLAB IDE to use this MPLAB C18	只有当系统安装有 MPLAB IDE 时才会出现此选项。选择此选项将把 MPLAB IDE 配置为使用新安装的 MPLAB C18。这包括在 MPLAB IDE 中，使用 MPLAB C18 库目录作为 MPLAB C18 项目的默认库路径。
Update MPLAB IDE to use this MPLINK linker	只有当系统安装有 MPLAB IDE 时才会出现此选项。选择此选项将 MPLAB IDE 配置为使用新安装的 MPLINK™ 链接器。

2.2.7 文档通知

如果可执行文件不安装文档，将出现与图 2-7 类似的通知。可从 MPLAB C18 安装 CD-ROM 和 Microchip 网站上获得文档。

注： 要通过 MPLAB C18 CD-ROM 或从网站上下载的带文档的升级版本自动安装文档，请在 Select Components （选择组件）对话框中选择 Documentation （文档）选项 （见图 2-5）。

图 2-7： 安装：更新文档提示

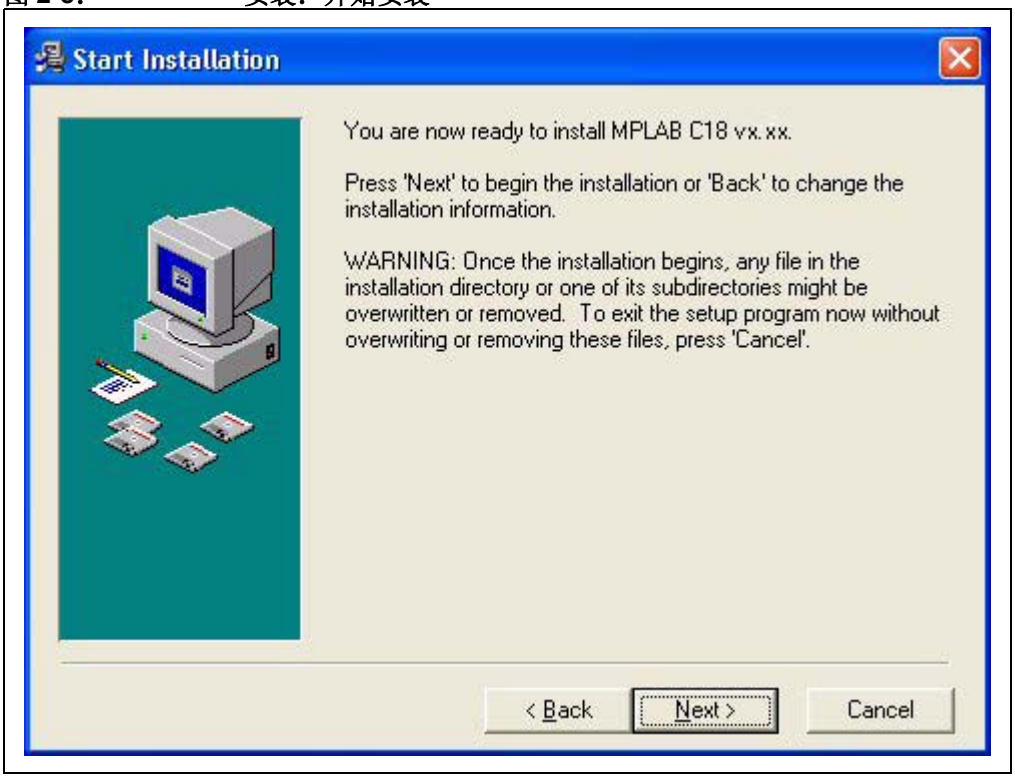


2.2.8 开始安装

在 Start Installation （开始安装）屏幕 （图 2-8）中点击 **Next>** 安装文件。

注： 安装目录及其子目录中的所有文件都会被覆盖或删除。

图 2-8：　　安装：开始安装



2.2.9　完成安装

在 Installation Complete （安装完成）屏幕中，点击 **Finish**。此时 MPLAB C18 已安装成功。

为使 MPLAB C18 能正常运行，可能需要重新启动计算机。如果出现了 “Restart Computer” （“重启计算机”）对话框，可选择 **Yes** 立即重新启动计算机，或选择 **No**，以后再重新启动计算机。

2.3　卸载 MPLAB C18

要卸载 MPLAB C18，请打开 Windows 控制面板并运行 “Add/Remove Programs”（“添加 / 删除程序”）。在程序列表中选择安装的 MPLAB C18 程序，并按照指示来删除程序。这样就会从计算机中删除 MPLAB C18 目录及其中的内容。

注：	如果卸载 MPLAB C18 的升级版本，则全部安装内容都会被删除；MPLAB C18 不能恢复到升级前安装的版本。在卸载升级版本前，要确保有原先的安装盘，以便以后可重新安装 MPLAB C18。

第 3 章 项目的基本操作及 MPLAB IDE 配置

3.1 简介

本章讲述项目的基本操作以及通过 MPLAB SIM 测试本指南中示例和应用程序的配置选项。本章仅对此做了概括性介绍，并以一个普通的应用作为示例。关于器件选择和链接描述文件的详细信息随具体应用而有所不同。如果读者对于这些基本操作已较为熟悉，则可忽略本章内容。

注： 这里未一步步讲述创建和编译项目所需的详细步骤，而仅概述了如何确保正确设置 MPLAB IDE 的关键操作。《MPLAB® IDE 用户指南》中提供了如何创建项目的教程。

本章内容包括：

- 项目概述
- 创建文件
- 创建项目
- 使用项目窗口
- 配置语言工具路径
- 检查安装和编译选项
- 编译和测试

3.2 项目概述

项目由 MPLAB IDE 中与语言工具 （如 MPLAB C18）相关的文件组成。项目由源文件、头文件、目标文件库文件和链接描述文件组成。每个项目都应该有一个或多个源文件及一个链接描述文件。

一般来说，至少需要一个头文件来标识目标单片机的寄存器名。头文件一般被源文件包含进来而不用明确地添加到项目中。

项目的输出文件包括将作为固件装载到目标单片机中的可执行代码。生成的调试文件有助于 MPLAB IDE 将源文件中的符号和函数名与可执行代码和用于存储变量的存储区关联起来。

本指南中的大部分示例和应用程序都只包含一个由一个源文件和一个链接描述文件组成的项目。

更多信息，请参阅 《MPLAB® IDE v6.xx 快速入门指南》（DS51281C_CN）。

3.3 创建文件

启动 MPLAB IDE 并选择 *File>New* 打开一个新的空白源文件。本指南中的示例和应用程序列出了源代码，可通过 MPLAB 编辑器键入或拷贝并粘贴这些源代码到文本文件中。示例源代码在 `mcc18\example\getting started` 中。

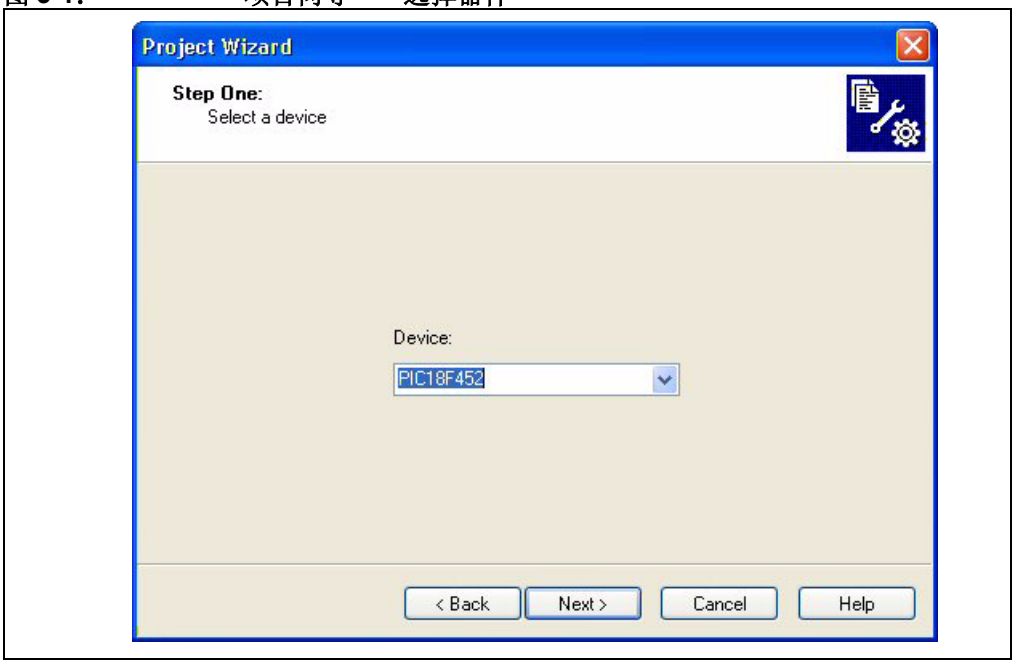
键入或拷贝源代码（在本指南的每个示例中列出）到这个新文件中。（从本文档的示例中拷贝的源代码可能不保留空白。）选择 *File>Save As* 来保存这个文件。浏览至一个文件夹或新建一个文件夹来保存文件。点击 **Save**（保存）。

> **注：** 可在创建新项目之前或之后创建新源文件，这个顺序并不重要。创建新文件并不会自动将该文件添加到当前打开的项目中。

3.4 创建项目

1. 选择 *Project>Project Wizard* 来创建新项目。当显示 Welcome 屏幕时，点击 **Next>** 继续。
2. 在 “Step One: Select a device” 对话框中，用下拉菜单来选择器件。

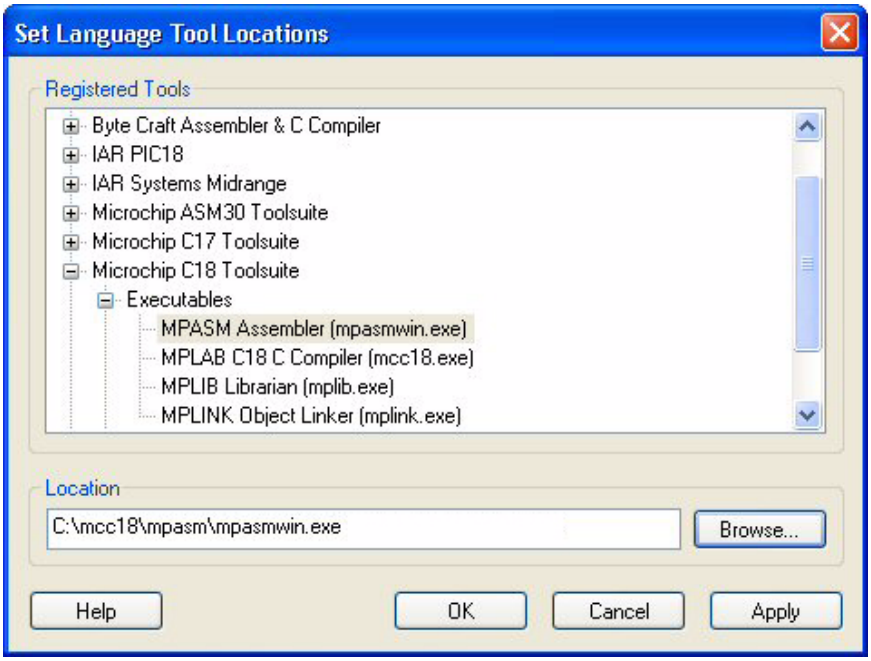
图 3-1： 项目向导——选择器件



点击 **Next>** 继续。

3. 在 “Step Two: Select a language toolsuite”对话框中选择 “Microchip C18 Toolsuite”作为 “Active Toolsuite”。然后点击工具包中 （在 “Toolsuite Contents”下）的每个语言工具并检查或设置与相关可执行文件的路径（图 3-2）。

图 3-2： 项目向导——选择语言工具包



MPASM 汇编器应指向 “Location”下的汇编器可执行文件 `MPASMWIN.exe`。如果没有，应键入或浏览到可执行文件的路径，默认为：

```
C:\mcc18\mpasm\MPASMWIN.exe
```

MPLAB C18 C 编译器应指向 “Location”下的编译器可执行文件 `mcc18.exe`。如果没有，应键入或浏览到可执行文件的路径，默认为：

```
C:\mcc18\bin\mcc18.exe
```

MPLINK 目标链接器应指向 “Location”下的链接器可执行文件 `MPLink.exe`。如果没有，应键入或浏览到可执行文件的路径，默认为：

```
C:\mcc18\bin\MPLink.exe
```

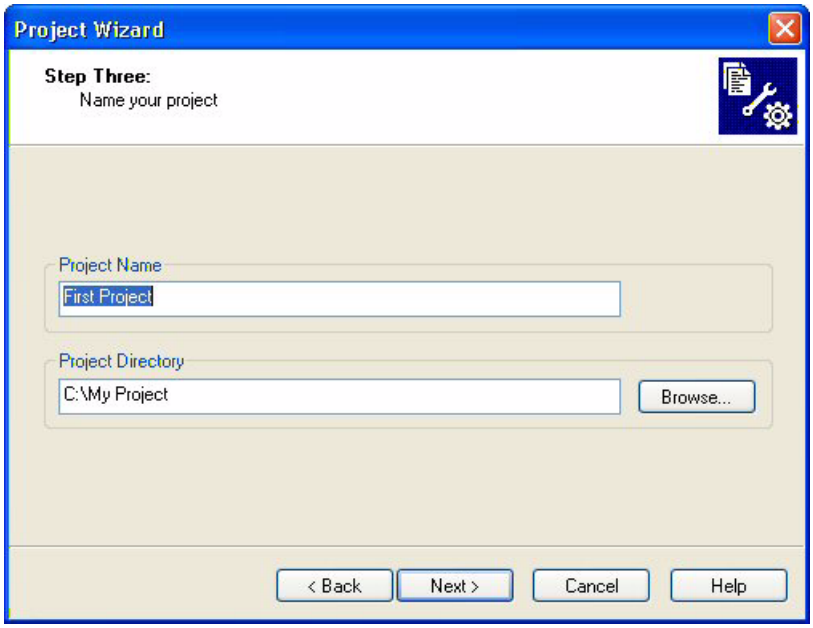
MPLIB 库管理器应指向 “Location”下的库可执行文件 `MPLib.exe`。如果没有，应键入或浏览到可执行文件的路径，默认为：

```
C:\mcc18\bin\MPLib.exe
```

点击 **Next>** 继续。

4. 在 “Step Three: Name your project”（图 3-3）对话框中，键入项目名并点击 **Browse** 选择保存项目的文件夹。然后点击 **Next>** 继续。

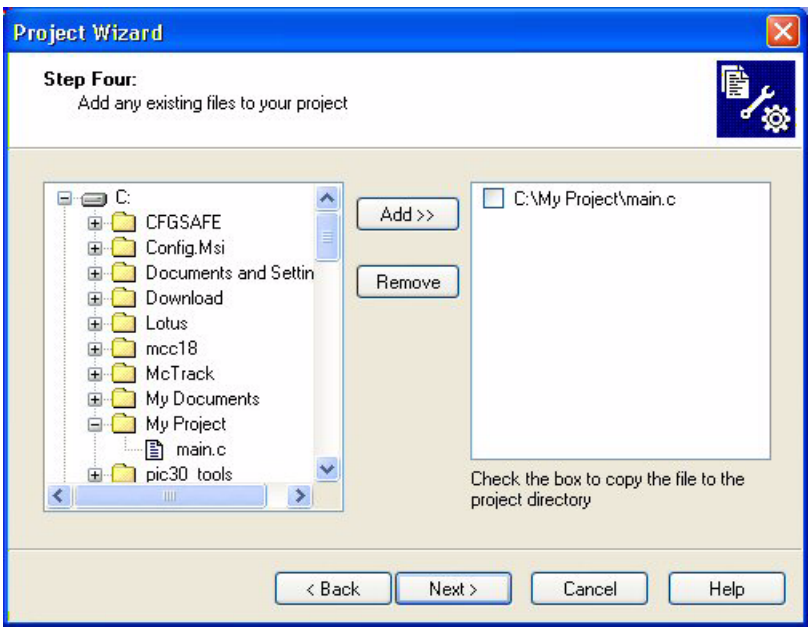
图 3-3： 项目向导——项目名和目录



5. 在 “Step Four: Add any existing files to your project”对话框中，浏览至要添加到项目中的源文件。

   首先，选择原先创建好的源文件。如果还未创建源文件，可以以后后再添加（见图 3-4）。点击 **ADD>>** 将它添加到项目要使用的文件列表中 （在右侧）。

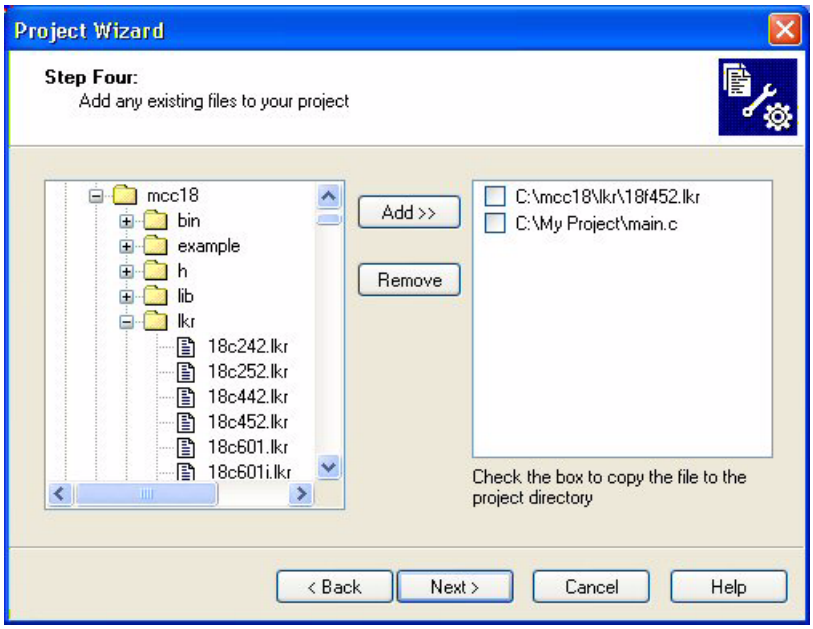
图 3-4： 项目向导——添加 C 源文件



然后，必须添加链接描述文件，告知链接器所选择器件的存储器构成。链接描述文件位于 MPLAB C18 安装目录下的 `lkr` 子文件夹中。向下滚动滚动条找到所选择器件的 `.lkr` 文件，选中它并点击 **ADD>>** 将该文件添加到项目中。请参见图 3-5 中的示例。选择 **Next>** 继续。

注： 当随 MPLAB IDE 安装 MPASM 时，也会随之安装链接描述文件。要确保使用 `\mcc18\lkr` 文件夹中的链接描述文件。

图 3-5：　　项目向导——添加链接描述文件



6. 在 Summary （摘要）屏幕中，重新检查 “Project Parameters”（项目参数），验证器件、工具包以及项目文件的路径是否正确。如果想修改某一项，可以点击 **<Back** 返回到上一个向导对话框。点击 **Finish** 生成新的项目和工作区。点击 **OK** 退出。

3.5 使用项目窗口

在 MPLAB IDE 工作区内找到项目窗口。工作区的文件名应出现在项目窗口顶部的标题栏中，项目的文件名作为项目的顶部节点项目应该看起来与图 3-6 类似。

图 3-6： 项目窗口

注： 如果发生错误，可选中文件名并按删除键或通过鼠标右键菜单来删除文件。将光标移动到 “Source Files”或 “Linker Scripts”上并通过鼠标右键来向项目添加正确的文件。

3.6 配置语言工具路径

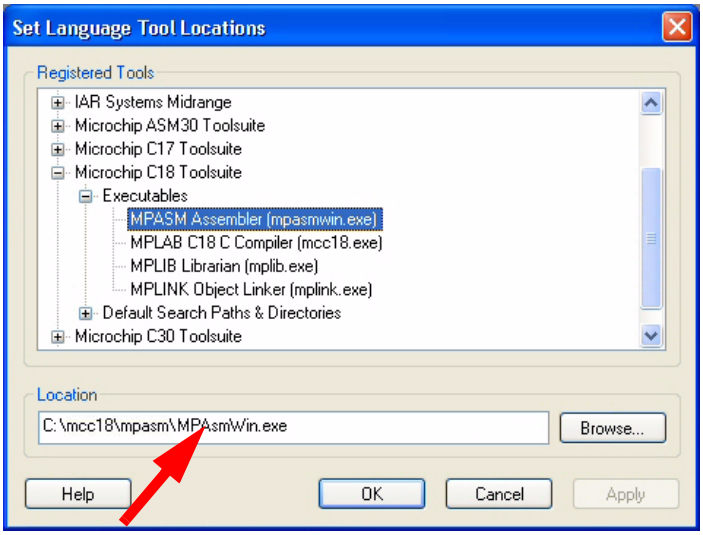
本节讲述如何为 MPLAB 项目设置默认路径。创建默认路径可方便地将这些默认路径应用于新项目。项目创建后也可选择或更改这些默认路径。

通过 MPLAB IDE 程序来选择语言工具路径，因此要启动 MPLAB IDE 来进行。

选择 *Project>Set Language Tool Locations* 打开 Set Language Tool Locations （设置语言工具路径）对话框。点击 **Microchip C18 Toolsuite** 旁边的加号来展开它，然后选中并展开 Executables 文件夹。在展开的列表中点击每个可执行文件来检查其安装路径是否与 **Location** 中所示一致。

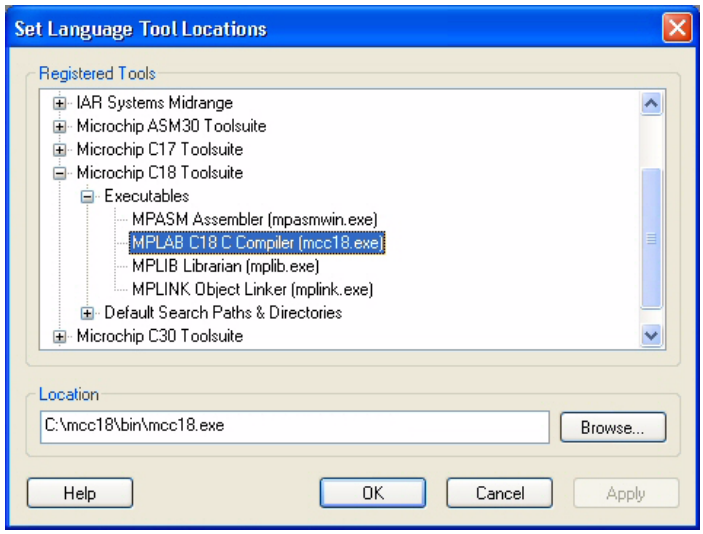
对于 MPASM 汇编器，要检查其路径是否如图 3-7 所示，为 C:\mcc18\mpasm\MPASMWIN.exe。

图 3-7：　　设置语言工具路径：MPASM™ 汇编器



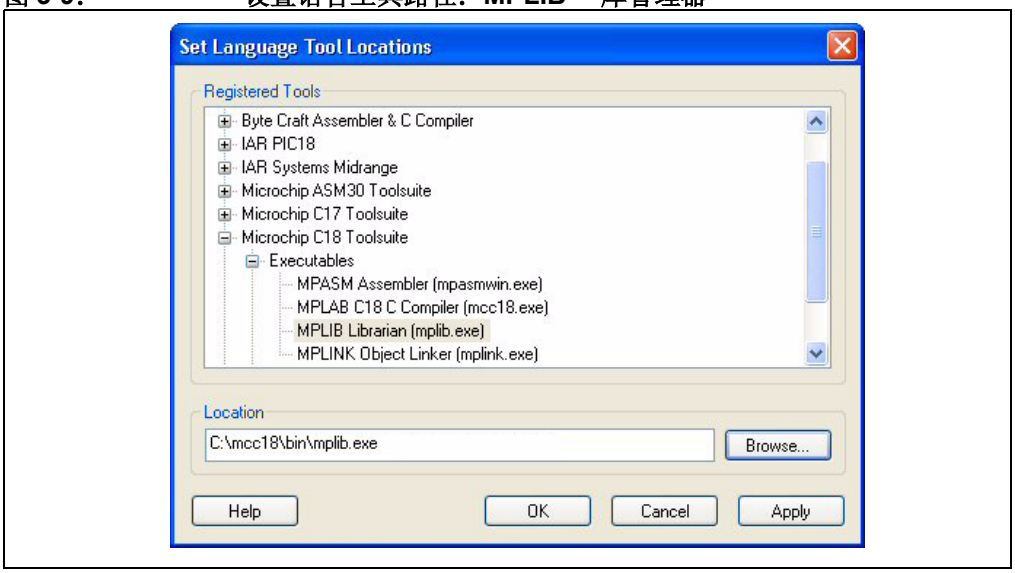
对于 MPLAB C18 编译器可执行文件，要检查其路径是否如图 3-8 所示，为 C:\mcc18\bin\mcc18.exe。

图 3-8：　　设置语言工具路径：MPLAB® C18



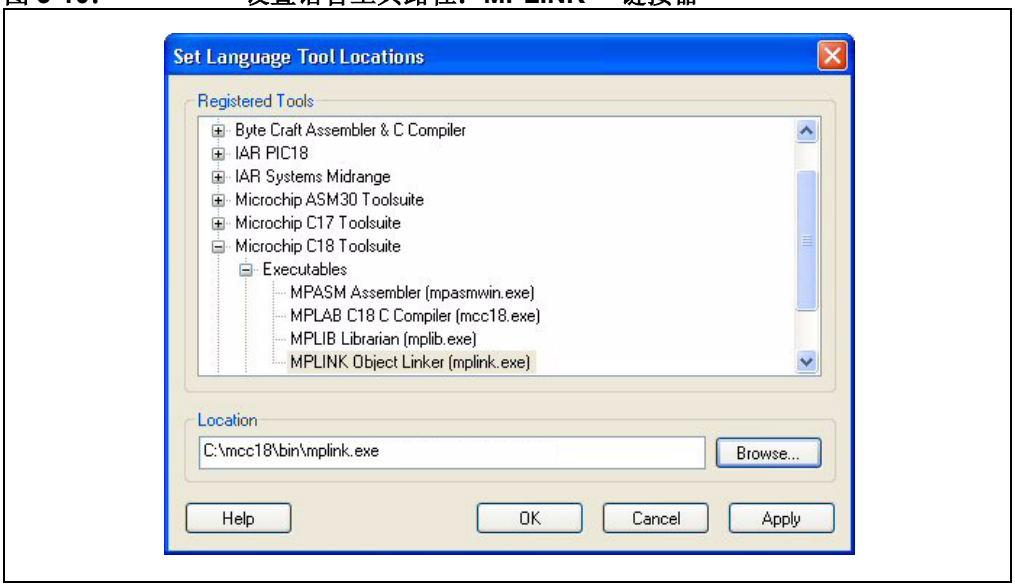
对于 MPLIB 库管理器 （编译器包可执行文件的一部分），要检查其路径是否如图 3-9 所示，为 C:\mcc18\bin\MPLib.exe。

图 3-9： 设置语言工具路径：MPLIB™ 库管理器



对于 MPLINK 链接器，要确保其路径如图 3-10 所示，为 C:\mcc18\bin\MPLink.exe。

图 3-10： 设置语言工具路径：MPLINK™ 链接器



点击 **OK** 保存这些设置并关闭这个对话框。

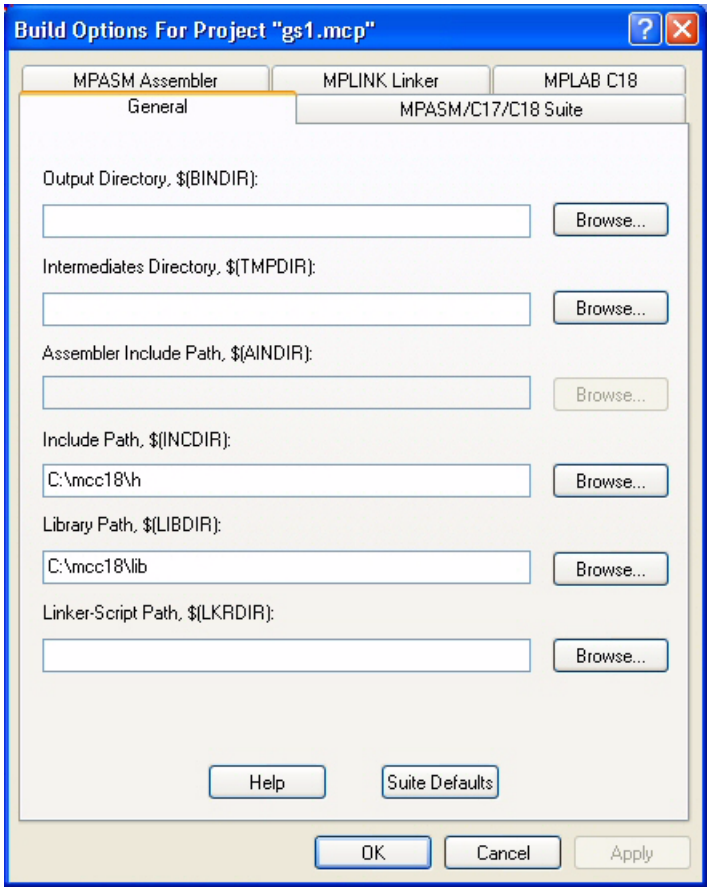
3.7 检查安装和编译选项

在编译和测试程序之前，还应该检查安装和项目设置。

语言工具应该安装正确，且设置对于这些第一批代码示例要正确，否则将发生错误。按照如下所述进行这些检查：

1. 选择 *Project>Build Options...>Project*，并点击 **General** （常规）选项卡。如果 **Include Path** 和 **Library Path** 未如图 3-11 中所示设置，则使用 **Browse** 按钮来在 MPLAB C18 安装目录中找到这些文件夹。

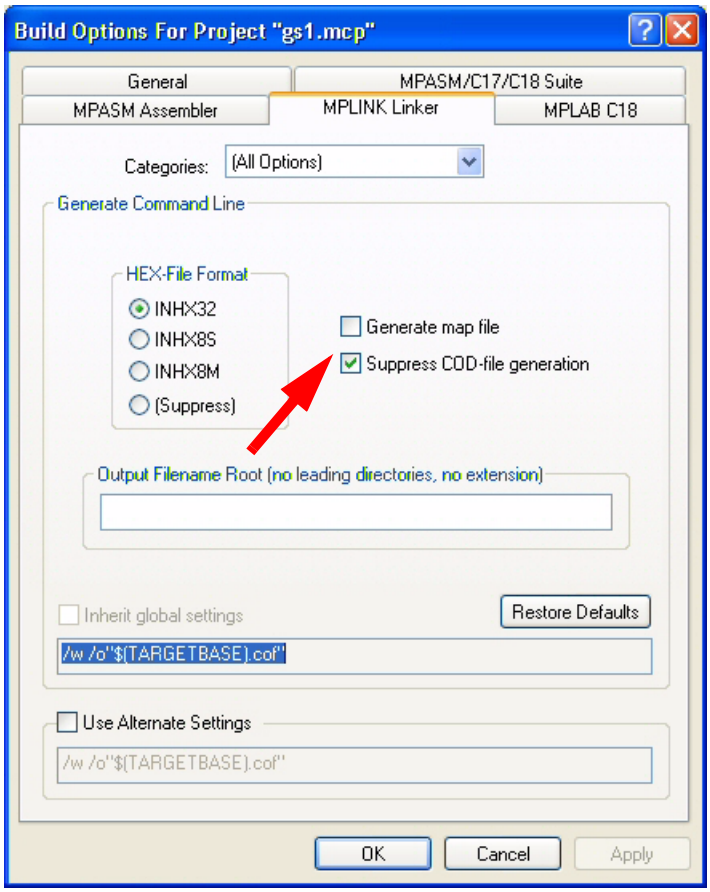
图 3-11： 编译选项：GENERAL



注： 可为一个包含路径或一个库搜索路径输入多个路径，方法是将路径之间用分号分隔开：
`c:\myprojects\h;c:\mcc18\h`。

2. 有一个选项要更改为与默认设置不同。点击 **MPLINK Linker** 选项卡。如果 **Suppress COD-file generation** 复选框没有选中，则选中它：

图 3-12：　　编译选项：MPLINK™ LINKER



注： 如果不选中这个复选框，链接器会生成 MPLAB IDE 不再使用的老 `.cod` 文件类型。这个文件格式具有 62 个字符的文件 / 路径长度限制，将导致错误 “`name exceeds file format maximum of 62 characters`”。

点击 **OK** 关闭这个对话框。

3.8 编译和测试

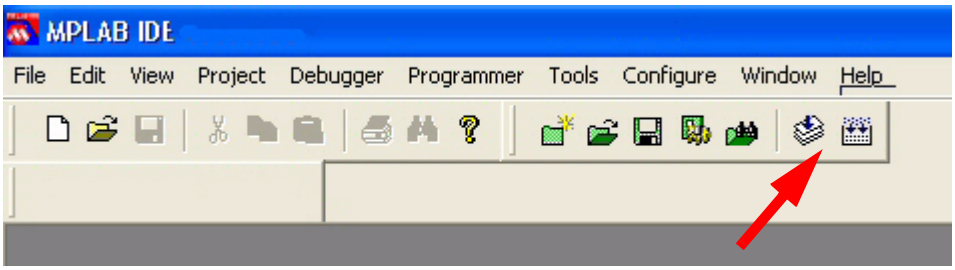
3.8.1 编译项目

如果按照指示完成了安装，就可以选择菜单 *Project>Build All* 或 *Project>Make* 来编译项目了。

> **注：** 编译并链接项目中的所有文件称为“make”或“build”。**Build All** 将重新编译项目中的所有源文件，而 Make 将仅重新编译自上次编译后更改过的源文件，因此 Make 编译速度较快，尤其是项目中包含许多源文件时。

可不通过菜单选择，而使用快捷键 **Ctrl+F10** 和 **F10** 来进行编译。工具栏中也包含这些功能的图标，因此按一下功能键或点击一次鼠标即可编译项目：

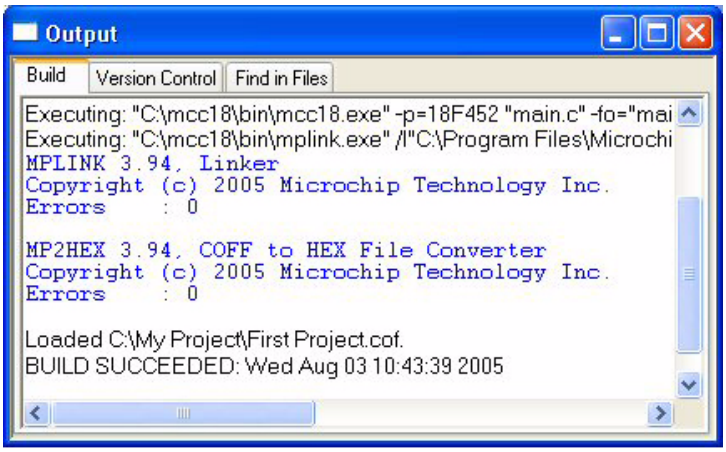
图 3-13： BUILD ALL 和 MAKE 的图标



> **注：** 将光标放在这些图标上，即可弹出文本标识图标的功能。

项目应正确编译，如 Output （输出）窗口中所示：

图 3-14： 编译成功后的 OUTPUT 窗口



如果来自 MPLINK （链接器）和 MP2HEX （.hex 文件转换器）的消息未显示“Errors : 0”，则可能有输入错误。在 Output 窗口中找到第一个错误。如果是输入错误，则在 Output 窗口中的该错误行上双击来在文件 main.c 中编辑错误。如果存在其他错误，请参阅**第 7 章 “疑难解答”**。

3.8.2 通过 MPLAB® SIM 测试程序

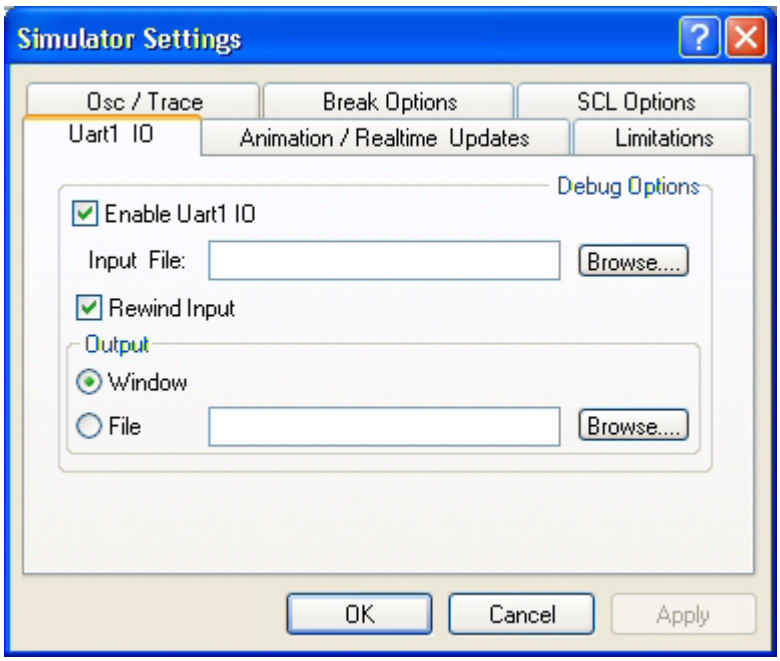
要在 MPLAB IDE 中测试这些程序，可使用内置的软件模拟器 MPLAB SIM。

1. 选择 *Debugger>Select Tool>MPLAB SIM* 来启用模拟器。

   更改调试工具后要重新编译项目，因为程序存储区可能被清除了。

2. 选择 *Debugger>Settings* 并点击 **Uart1 IO** 选项卡。应该选中 **Enable Uart1 IO**（**使能 Uart1 IO**）复选框，且 **Output**（输出）应该设置为 **Window**（窗口），如图 3-15 所示：

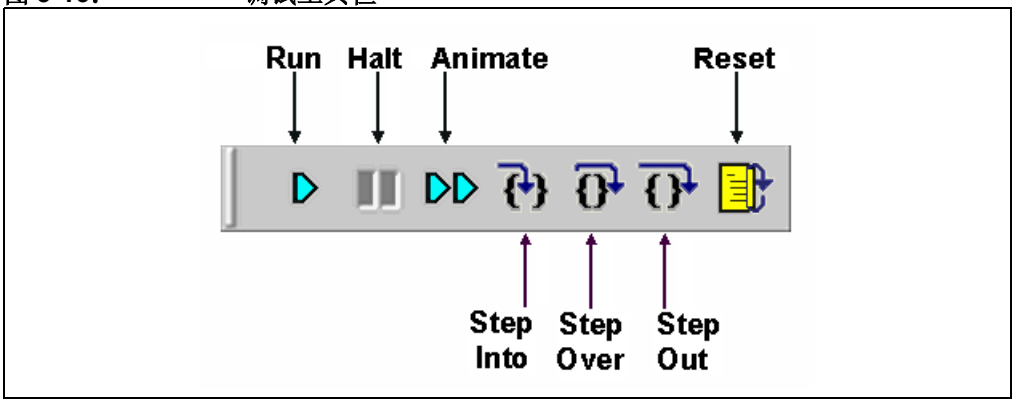
图 3-15： 软件模拟器设置：UART1



注： 这个对话框可使来自 `printf()` 函数的文本传输到软件模拟器的 UART（串行 I/O 外设），然后传输到 MPLAB IDE 的 Output 窗口。

选择了软件模拟器后，将在 MPLAB 菜单下出现 **Debug Toolbar**（调试工具栏，图 3-16）。

图 3-16：　　调试工具栏



图标	功能
Run	运行程序
Halt	暂停程序执行
Animate	连续单步执行指令。使用 *Debugger>Halt* 或按 Halt 图标来暂停。
Step Into	单步执行下一条指令
Step Over	单步跳过下一条指令
Step Out	单步跳出子程序
Reset	执行 $\overline{\text{MCLR}}$ 复位

欲获得关于项目、 MPLAB 配置和调试技巧的更多信息，请参阅 《MPLAB® IDE 用户指南》。

注：

第 4 章 简单入门程序

4.1 简介

假设读者已熟悉 MPLAB 项目。**第 3 章 “项目的基本操作及 MPLAB IDE 配置”** 给出了有关 MPLAB 项目的简要概述。更多深入描述，请参见 《MPLAB® IDE 用户指南》。

下面的章节给出了三个简单的入门程序，目的在于使工程师或学生通过使用 MPLAB 集成开发环境熟悉 MPLAB C18 C 编译器。

- 程序 1： “Hello, world!” —— 输出文本 “Hello, world!”
- 程序 2：使用软件模拟器点亮 LED ——写入模拟的 PIC 18 器件的 I/O 引脚以点亮一个指示灯。
- 程序 3：使用软件模拟器使 LED 闪烁 ——扩展第二个程序，使指示灯闪烁。
- 使用演示板 ——演示了如何使用演示板测试程序。
  如果有 MPLAB ICD 2 和开发硬件，就能在编译程序 3 后使用 MPLAB ICD 2 在开发板上调式程序，从而使开发板上的 LED 闪烁。

4.2 程序 1： “HELLO, WORLD!”

4.2.1 写源代码

典型的 “Hello, world!” 函数包含以下 C 语句来输出一条消息：

```
printf ("Hello, world!\n");
```

函数 `main()` 如例 4-1 那样编写：

例 4-1： HELLO, WORLD! `main()` 代码

```
void main (void)
{
  printf ("Hello, world!\n");

  while (1)
    ;
}
```

注： 由于通常情况下 “Hello, world!” 程序是在 PC 上编译、执行，然后返回操作系统和其他任务的，因此不使用最后的 “`while (1)`” 语句。但是在嵌入式控制器中，目标单片机是持续运行的，必须做一些事情，因此在此例中，采用了一个无限循环来使单片机在完成输出 “Hello, world!” 这一任务后保持忙碌。

要使用 MPLAB C18 进行编译，代码必须如例 4-2 那样编写。

例 4-2：　　程序 1 代码

```
#include <stdio.h>

#pragma config WDT = OFF

void main (void)
{
  printf ("Hello, world!\n");

  while (1)
    ;
}
```

第一行包含头文件 stdio.h，此文件含有 printf() 函数的原型。#pragma 语句是 MPLAB C18 特有的。 #pragma 语句控制目标单片机的看门狗定时器 , 将看门狗定时器禁止以防止其干扰程序的执行。

注： 看门狗定时器是 PIC18 MCU 的外设，它在默认情况下是使能的。使能看门狗定时器时，程序最终会因该定时器超时而复位。在最终应用中，可以使能看门狗定时器来用它检验固件是否正确运行。

4.2.2 创建程序 1

在名为 first project 的新文件夹下创建一个名为 gs1 的新项目。创建一个新文件，将例 4-2 中的代码键入或复制、粘贴到此文件中，将该文件保存为 main.c。然后，将该文件夹中的 main.c 作为源文件添加到项目中，并在项目中添加 18F452.lkr 链接描述文件。

最终的项目应如图 4-1 所示：

图 4-1：　　最终项目窗口

注： 切记通过 *Configure>Select Device*（配置 > 选择器件）选择 PIC18F452 作为当前器件。

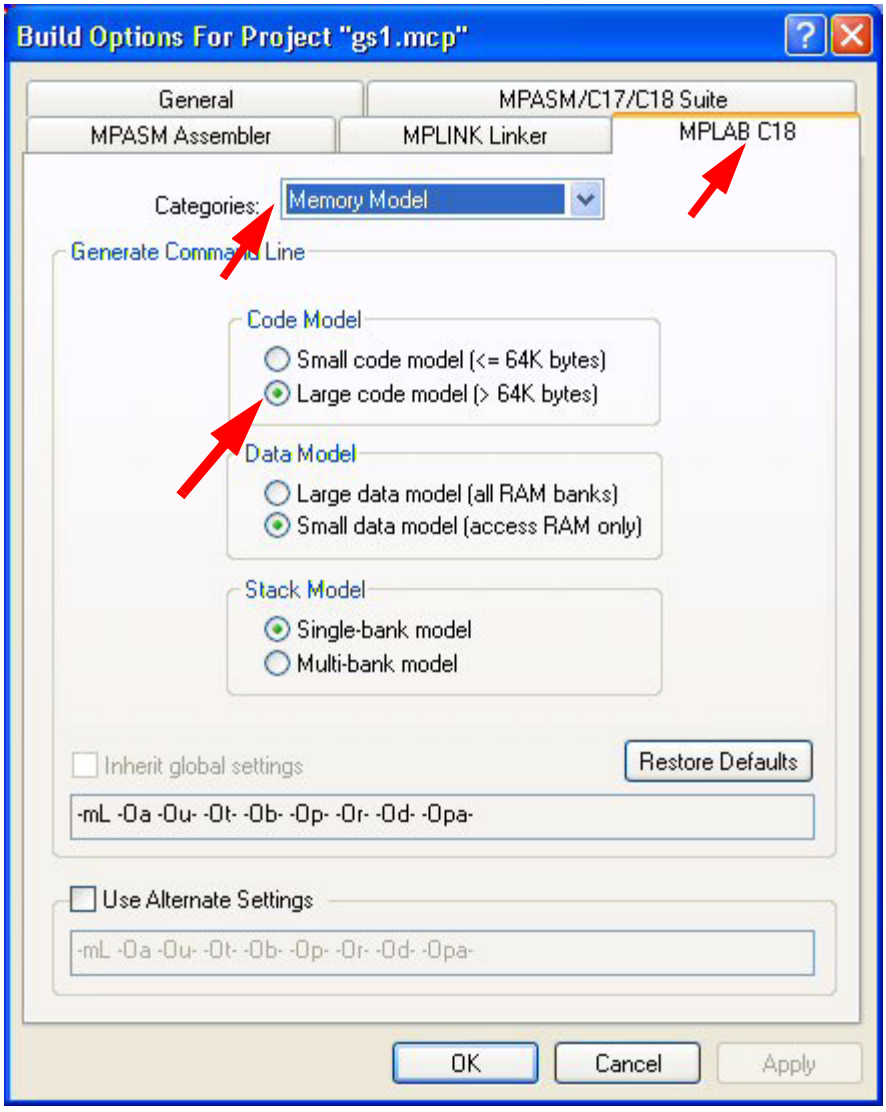
4.2.3 设置存储模型

当通过包含 `stdio.h` 使用标准库时，应为项目选择大代码模型。

> **注：** 标准库是按大代码模型构建的，如果没有选中大代码模型，就会发出类型限定符不匹配的警告。请参见**第 7 章 “疑难解答”**中的 FAQ-4 “为什么会出现 “Warning [2066] type qualifier mismatch in assignment”（警告 [2066] 指定的类型限定符不匹配）？ ”。

转到 *Project>Build Options>Project* 对话框，并选择 **MPLAB C18** 选项卡，然后选择 **Categories: Memory Model** （类别：存储模型）并选中 **Large code model (> 64K bytes)** （大代码模型 （> 64 KB））。

图 4-2： 选择大代码模型

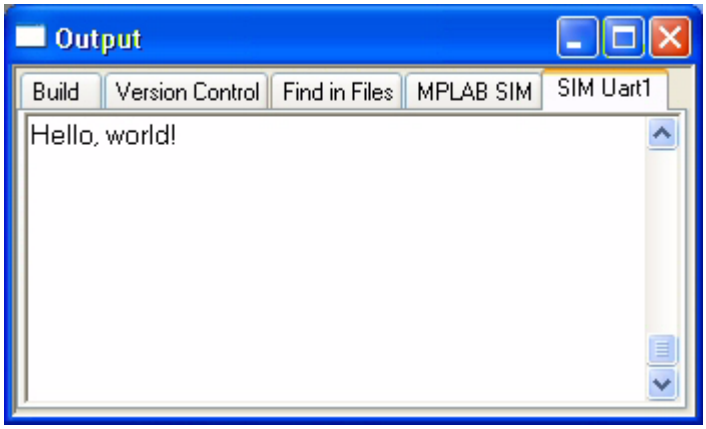


4.2.4 测试程序 1

使用 *Project>Build All*（项目 > 编译所有）或等效的图标来编译项目。

编译成功后， **Run** （运行）图标会变为蓝色，表明程序暂停并准备运行。选择 **Run** 图标后该图标变为灰色，表明程序正在运行。**Halt** （暂停）图标变为蓝色，表明程序正在运行并可被暂停。此外，会在底部的状态栏显示 “Running...”。选择 **Halt** 图标，若此时 Output 窗口未打开，就会将其打开 （见图 4-3）。

图 4-3： OUTPUT 窗口：“HELLO, WORLD!”



“Hello, world!”文本应该会出现在 Output 窗口的 **SIM Uart1** 选项卡中。

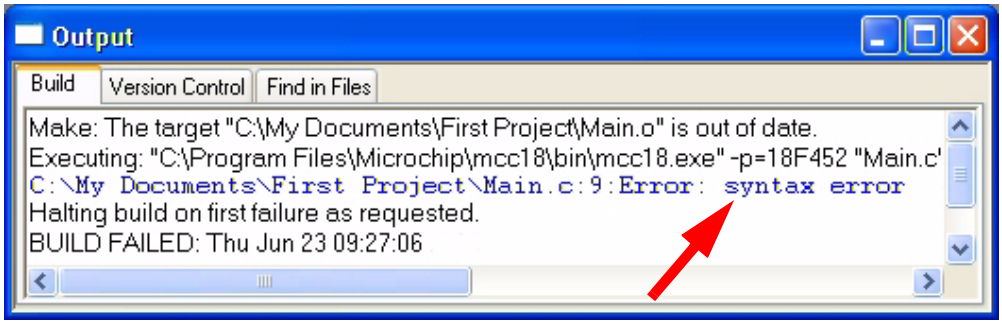
选择 **Reset** （复位）图标复位程序 , 然后再次选择 **Run** 图标将这条消息再次输出到 Output 窗口中。

注： 输出 “Hello, world!”之后，程序继续执行，运行无限循环 `while (1)` 直到被暂停。如果程序暂停后立即执行 **Run**，那么将继续运行无限循环。为了使程序从头开始重新执行，可在程序暂停后选择 **Reset** 图标。

4.2.5 解决问题

如果由于输入错误导致编译项目时出错， Output 窗口的最后几行可能会如图 4-4 所示：

图 4-4： OUTPUT 窗口语法错误



用鼠标双击包含 “syntax error” 的行 （见图 4-4），就会打开 MPLAB 编辑器窗口，光标停留在该窗口中出现该语法错误的行上。

“could not find stdio.h” 错误通常意味着没有设置包含路径。请参见**第 3.7 节 “检查安装和编译选项”** 了解有关设置包含路径的信息。

“type qualifier mismatch in assignment” 警告可能意味着在使用标准 I/O 时未选择大存储模型。请参见**第 4.2.3 节 “设置存储模型”**。

“c018i.o is not found” 错误可能意味着没有正确设置库路径。请参见**第 3.7 节 “检查安装和编译选项”** 了解有关设置库路径的信息。

如果有消息显示在 c018i.o 中无法找到 “main” 的定义，请检查确保 “main” 中的所有字母都是小写的，因为 C 语言是区分大小写的。

“could not find definition of symbol...” 错误通常是由于使用了错误的链接描述文件引起的：

确保使用的是 mcc18\lkr 目录下的 18F452.lkr 文件。在 MPLAB IDE 的某个子目录中也含有一个供仅使用汇编器的项目使用的链接描述文件。请始终将 mcc18\lkr 链接描述文件用于所有使用 MPLAB C18 编译器的项目。

如果 “Hello, world!” 未出现在 Output 窗口中，请尝试下列步骤：

1. 确保选择了软件模拟器 （*Debugger>Select Tool>MPLAB SIM*）（调试器 > 选择工具 >MPLAB SIM）。
2. 确保使能了 **Uart1** 以发送 printf() 文本到 MPLAB IDE 的 Output 窗口，如图 3-15 所示。
3. 选择 **Halt** 图标 （图 3-16）。
4. 再次执行 **Build All**。Output 窗口中应该不会有错误消息，并且在该窗口的最后一行会显示 “Build Succeeded”（编译成功）消息 （图 3-13）。
5. 选择调试工具栏上的 **Reset** 图标 （图 3-16）。
6. 选择调试工具栏上的 **Run** 图标 （图 3-16）。

注： 右击鼠标并选择 Clear Page （清空页面）选项可将 Output 窗口清空。

4.2.6 程序 1 “Hello, world!”总结

第一个程序示例到此结束，本示例涉及以下主题：

- 编写 MPLAB C18 代码
- 编译 （编译和链接）项目
- 用 MPLAB SIM 测试项目
- 解决初学者易犯的错误

4.3 程序 2：使用软件模拟器点亮 LED

第一个示例演示了在 MPLAB IDE 中使用 MPLAB C18 创建、编译和测试项目的基本步骤。它尚未涉及目标处理器将利用该代码完成什么任务的细节。在下一程序中，将生成代码来模拟点亮与 PIC18F452 的某一引脚相连的发光二极管 （Light Emitting Diode,LED）。

4.3.1 创建新项目

在名为 “Second Project”的新文件夹下创建名为 “GS2”的新项目。

确保设置了语言工具并且正确配置了 Build Options，如**第 3.7 节 “检查安装和编译选项”**所示。

4.3.2 编写源代码

创建一个新文件并键入例 4-3 中的代码。将其以 main.c 文件名保存在 “Second Project”文件夹中。

例 4-3： 程序 2 代码：main.c

```
#include <p18cxxx.h>

#pragma config WDT = OFF

void main (void)
{
  TRISB = 0;

  /* Reset the LEDs */
  PORTB = 0;

  /* Light the LEDs */
  PORTB = 0x5A;

  while (1)
    ;
}
```

本代码第一行包含了适用于所有 PIC18XXXX 器件的名为 p18cxxx.h 的通用处理器头文件。此文件会选择与在 MPLAB IDE 中选择的器件相适应的头文件；在本例中为文件名为 p18f542.h 的头文件 （也可以明确地包含该文件）。此文件包含了这些器件中特殊功能寄存器的定义。

“#pragma config WDT = OFF”与第一个程序中相同。

> **注：** 在 MPLAB C18 中， main 函数的返回值声明为 void，因为嵌入式应用程序不会返回到另一个操作系统或函数。

本例将使用 PORTB 寄存器 8 位 I/O 端口的 4 个引脚。

“TRISB = 0”将 PIC18F452 器件的 TRISB 寄存器清零。TRIS 寄存器控制端口上 I/O 引脚的方向。端口引脚可以是输入引脚，也可以是输出引脚。将该寄存器的所有位清零将使 8 个引脚都成为输出引脚。

> **注：** 一个便于记忆如何配置 TRIS 寄存器的方法是：将位设置为 0 表示输出，因为零 （0）像字母 O （Output，输出）（O = 0）。将位设置为 1 表示输入，因为数字一 （1）像字母 I （Input，输入）（I = 1）。

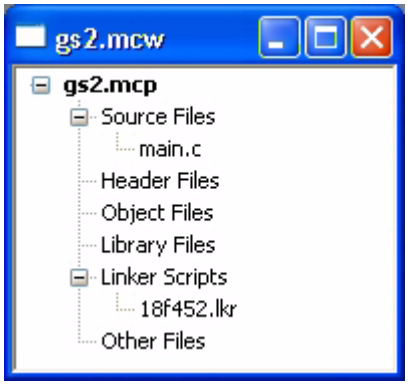
“PORTB = 0”将 PORTB 寄存器的 8 个引脚都设置为 0 或低电压。

“PORTB = 0x5A”将 PORTB 的 4 个引脚设置为 1 或高电压。（0x5A = 0b01011010）。

当在 PIC18F452 中执行此程序时，与其中一个引脚正确连接的 LED 的电平将会升高，点亮 LED。

将 main.c 作为源文件添加到项目中。选择 18F452.lkr 文件作为项目的链接描述文件。项目窗口应如图 4-5 所示。

图 4-5： GS2 项目



4.3.3 编译程序 2

通过 *Project>Build All* 编译项目。如有错误，请检查语言工具的路径以及编译选项是否正确，或者请参见**第 4.2.5 节 “解决问题”**或 **第 7 章 “疑难解答”**。

4.3.4 测试程序 2

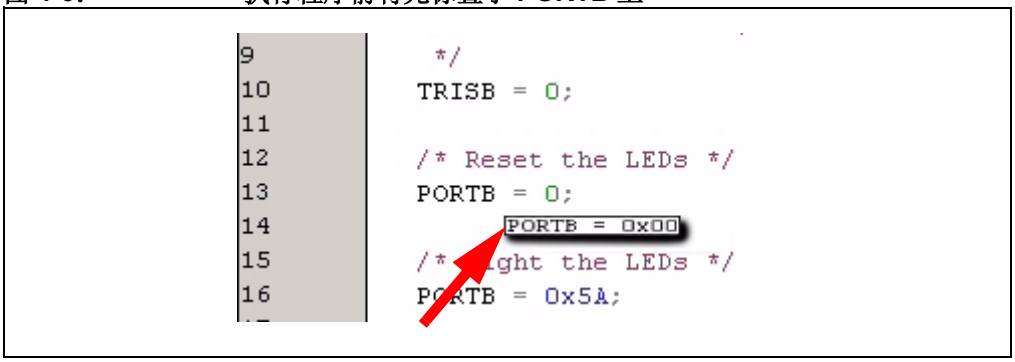
如同第一个程序，将使用 MPLAB IDE 中的软件模拟器测试此代码。确保使能了软件模拟器。如果没有预先选择软件模拟器，就需要重新编译项目。

要测试代码，必须监视 PORTB 上的引脚状态。MPLAB IDE 中有两种方法可以完成此任务。

4.3.4.1 将光标置于变量之上

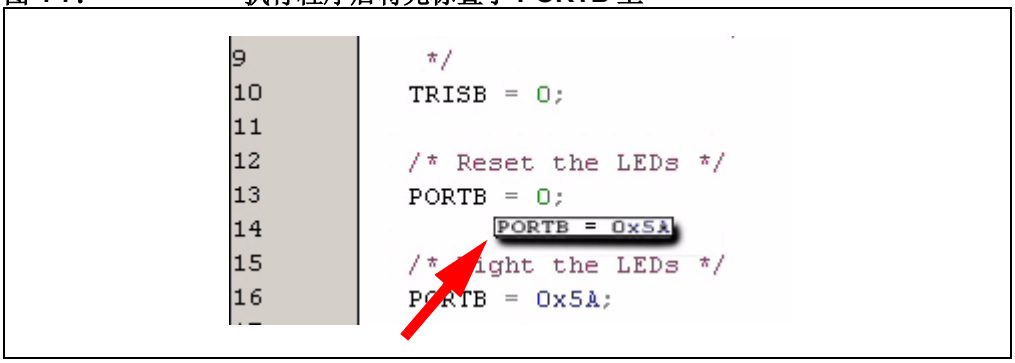
成功编译项目后，使用鼠标将光标置于编辑器窗口中的变量名之上，以显示变量的当前值。在运行程序之前，将光标放在 PORTB 上应该显示其值为零 （见图 4-6）：

图 4-6： 执行程序前将光标置于 PORTB 上



单击 **Run** 图标 （或选择 *Debug>Run* （调试 > 运行）），然后单击 **Halt** 图标并再次将光标置于该变量上。此时其值应为 0x5A （见图 4-7）：

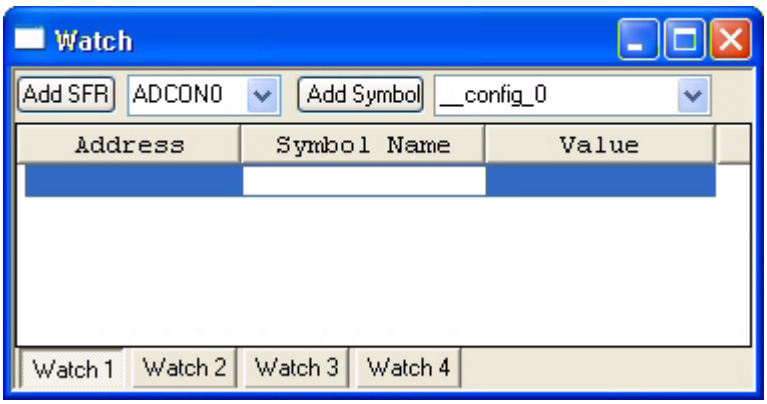
图 4-7： 执行程序后将光标置于 PORTB 上



4.3.4.2 使用 WATCH （观察）窗口

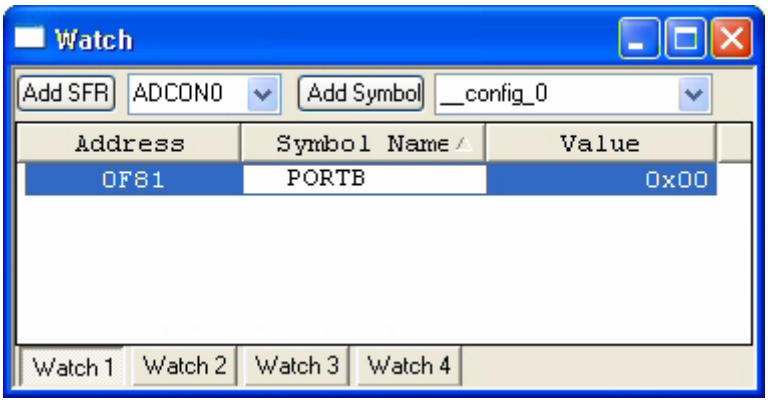
检查变量值的第二种方法是将其拖放到 Watch 窗口中。选择 *View>Watch* （视图 > 观察）打开一个新的 Watch 窗口 （见图 4-8）。

图 4-8： 新 Watch 窗口



现在将 Watch 窗口拖离源文件窗口直到两窗口无任何重叠。选中 `main.c` 中的单词 PORTB。当该单词处于高亮状态时，将其拖动到 Watch 窗口的空白区域。 Watch 窗口现在与图 4-9 相似。

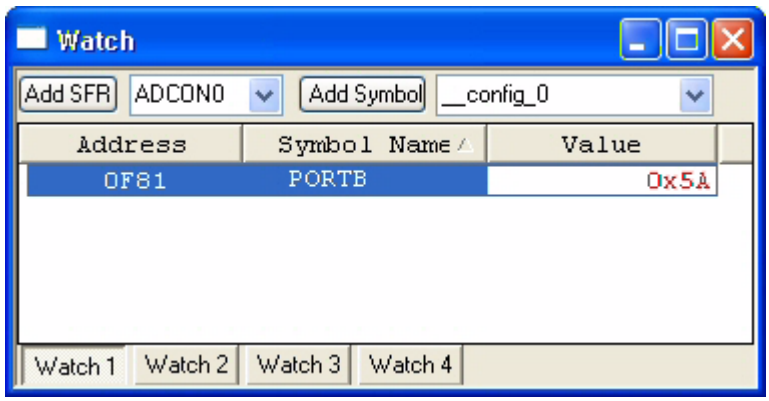
图 4-9： WATCH 窗口中的 PORTB 变量



注： 如果 PORTB 的值为 0x5A，则表明此前已执行了程序。双击 Watch 窗口中的值，键入零使之清零。

选择 **Run** 图标，数秒后再选择 **Halt** 图标。Watch 窗口中应该显示 PORTB 的值为 0x5A （见图 4-10）。

图 4-10： 程序执行后的 WATCH 窗口



双击 Watch 窗口中 PORTB 的值 0x5A 使之高亮，然后键入任意其他 8 位值。依次选择 **Reset** 和 **Run**，等待数秒后按下 **Halt**，将会看到该值返回到 0x5A。

4.3.5 程序 2 总结

第二个程序示例到此结束。该示例演示了以下主题：

- 使用带有特定处理器的寄存器定义的包含文件
- 编写代码设置 PIC18F452 中 PORTB 寄存器的位
- 使用鼠标查看寄存器的值
- 通过拖放向 Watch 窗口添加变量或寄存器
- 使用 Watch 窗口查看变量或寄存器的内容
- 在 Watch 窗口中改变变量或寄存器的值

4.4 程序 3：使用软件模拟器使 LED 闪烁

4.4.1 修改源代码

本程序将在上一程序的基础上使 PORTB 上的 LED 闪烁。修改上一程序使之循环运行，以交替地将引脚设置为高电平或低电平。将程序 2 的代码修改为例 4-4 所示：

例 4-4： 程序 3 代码

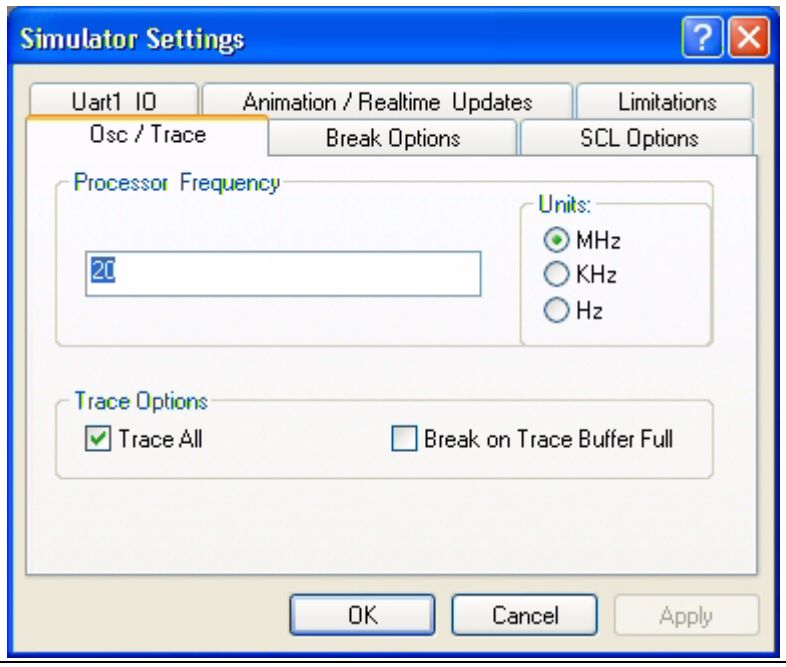
```
#include <p18cxxx.h>
#pragma config WDT = OFF

void main (void)
{
  TRISB = 0;

  while (1)
    {
      /* Reset the LEDs */
      PORTB = 0;

      /* Light the LEDs */
      PORTB = 0x5A;
    }
```

现在 `while()` 无限循环中的代码会不断地将 PORTB 的 4 个引脚置 1 和清零。

这样会产生使 LED 闪烁的效果吗？

PIC18F452 指令的执行速度非常快，通常小于一微秒，具体取决于时钟速度。LED 可能在不断地点亮－熄灭，但是非常快，以至于肉眼无法察觉它们是在闪烁的。处理器的时钟频率可由软件模拟器控制。选择 ***Debugger>Settings*** （调试器 > 设置）显示 Simulator Settings （软件模拟器设置）对话框 （见图 4-11）：

图 4-11： 软件模拟器设置

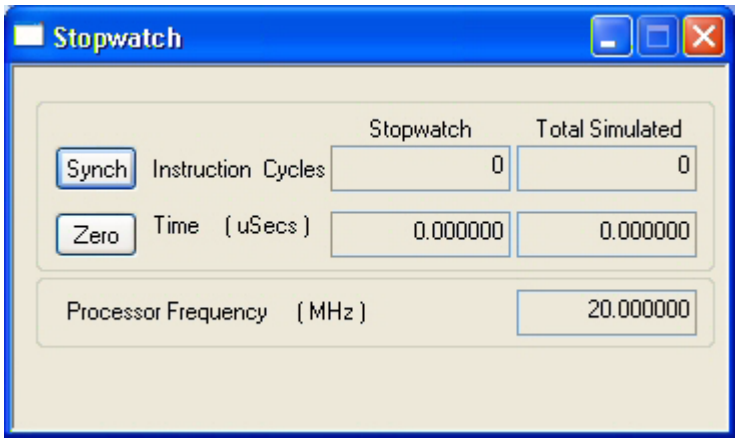


4.4.2 选择跑表 （Stopwatch）

在 **Osc/Trace** （振荡器 / 跟踪）选项卡上，处理器频率的默认设置为 20 MHz。如果显示的不是 20 MHz，请将其修改到与图 4-11 中的设置相匹配。然后单击 **OK**。

引脚电平变高和变低之间的时间间隔可用 MPLAB 跑表测定。选择 *Debugger>Stopwatch* （调试器 > 跑表）显示 MPLAB 跑表 （见图 4-12）。

图 4-12： 跑表

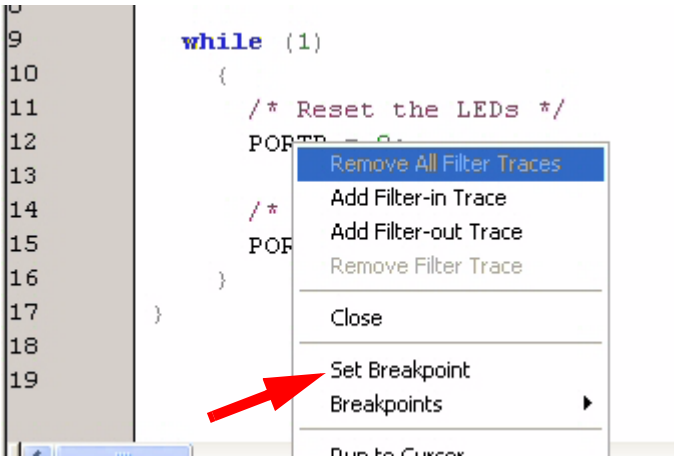


Stopwatch 窗口也显示当前的处理器频率设置为 20 MHz。为了测定 LED 点亮和熄灭之间的时间间隔，要在代码的恰当位置设置断点。使用鼠标右键来设置断点。

注： 如果无法从下拉菜单中选择 Stopwatch，可能是由于没有设置软件模拟器（*Debugger>Select Tool>MPLAB SIM*）。

将光标放在包含 PORTB 的第 12 行上并右击鼠标，将显示如图 4-13 所示的调式菜单。

图 4-13： 鼠标右键菜单

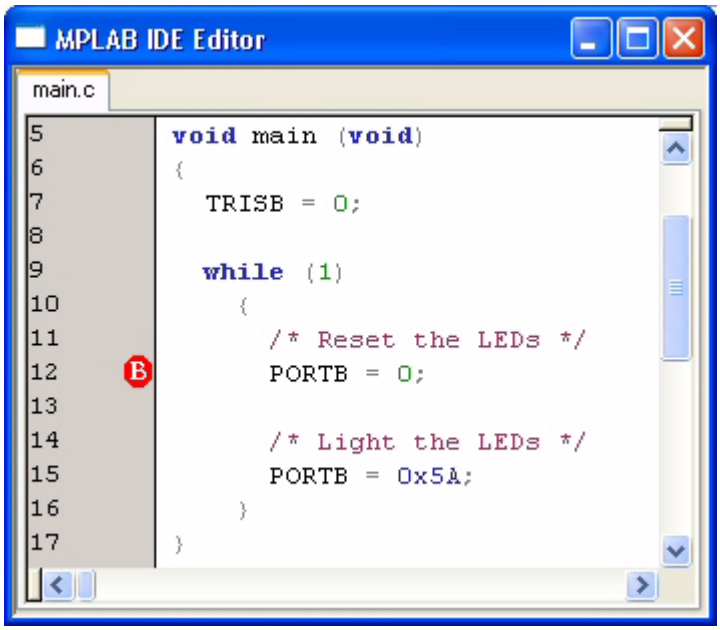


4.4.3 设置断点

从右键弹出菜单中选择 “Set Breakpoint”（设置断点），应该在屏幕的当前行显示一个断点，由左侧空白区域中的红色 “**B**”图标表示 （见图 4-14）。

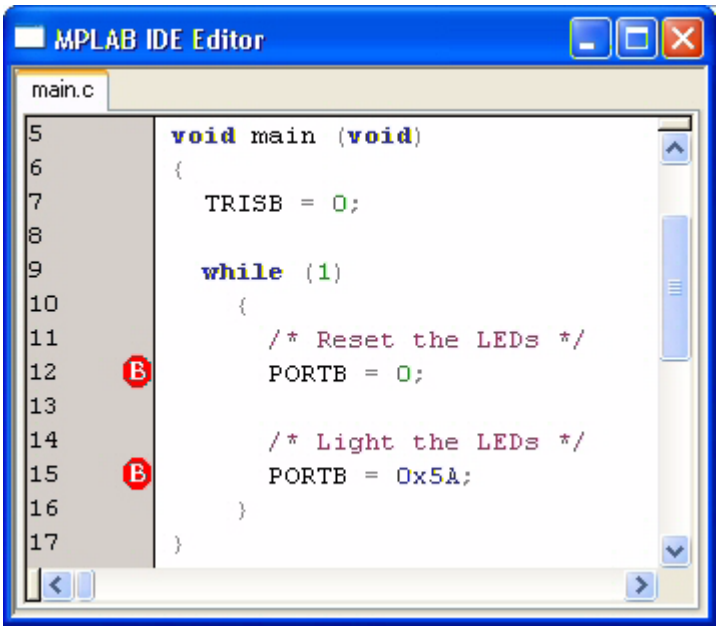
> **注：** 如果无法设置断点，可能是因为尚未编译项目。选择 *Project>Build All* 并在随后尝试重新设置断点。

图 4-14： 断点



在第 15 行 （此行将值 0x5A 发送到 PORTB）设置第二个断点。此时编辑器窗口中应该有两个断点，与图 4-15 相似：

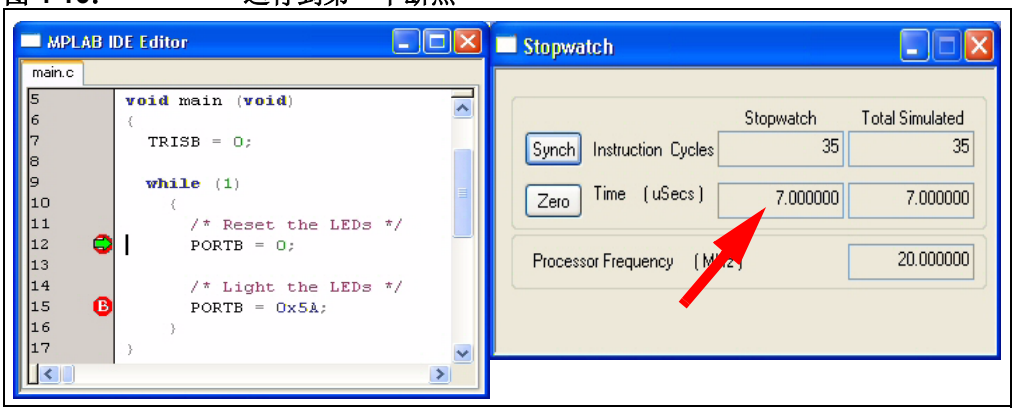
图 4-15： 第二个断点



4.4.4 运行程序 3

选择 **Run** 图标，程序应被执行，并随后在第一个断点处停止，由绿色的箭头指示。注意跑表已经测量了到达第一个断点所花的时间 （见图 4-16）。

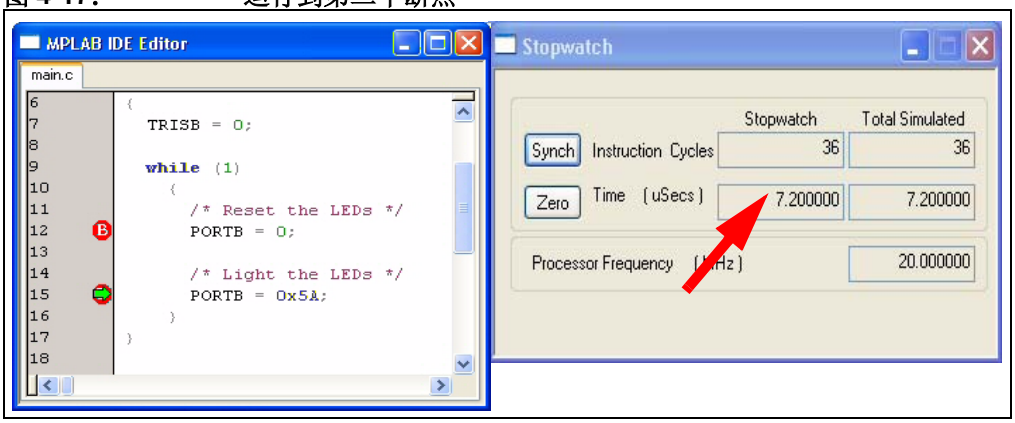
图 4-16： 运行到第一个断点



跑表读数为 7.000000 微秒，表示程序从复位到运行至该断点用了 7 微秒。

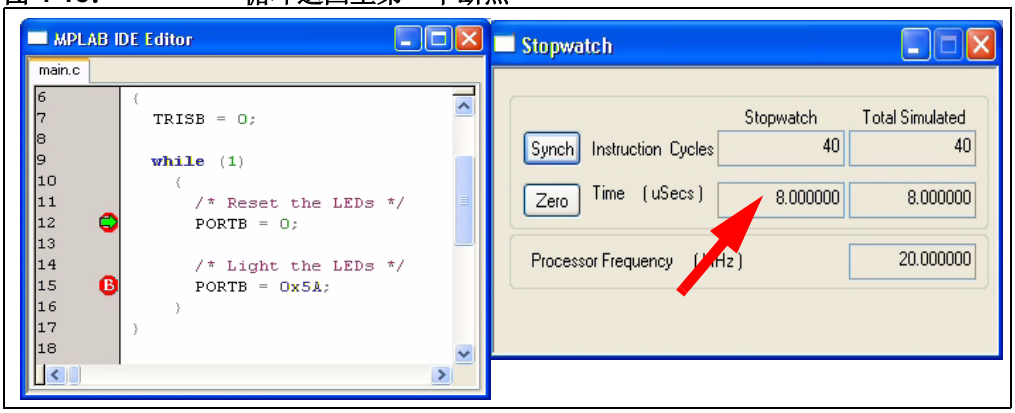
再次选择 **Run** 使程序运行至第二个断点 （见图 4-17）：

图 4-17： 运行到第二个断点



此时跑表读数为 7.200000 微秒，表明从上一断点运行到此处用了 0.2 微秒。再次选择 **Run** 循环返回至第一个断点 （见图 4-18）：

图 4-18： 循环返回至第一个断点



4.4.5 分析程序 3

我们可以回答前面提出的问题了。跑表读数为 8.000000 微秒，因此执行一次循环的用时为 8.0 –7.2 = 0.8 微秒。如果 LED 每微秒点亮并熄灭的次数多于一次，对于肉眼来说太快而无法分辨。要使 LED 以可被感知的速率闪烁，就必须降低处理器频率，或者必须添加一些延时。

如果应用程序所需要做的只是使这些 LED 闪烁的话，就可以降低处理器频率。因为这样做会使所有的代码都运行得非常慢，完成其他任务而不是使 LED 闪烁的代码也会运行得很慢。更好的解决方案是添加一段延时。

4.4.6 添加一段延时

延时可以是一个使某个变量递减很多次的简单子程序。对本程序而言，延时子程序可按例 4-5 编写：

例 4-5： 延时程序

```
void delay (void)
{
  int i;
  for (i = 0; i < 10000; i++)
    ;
}
```

将其添加到 main.c 的代码中，并在 LED 熄灭和点亮后分别插入对该延时函数的调用（见例 4-6）。

例 4-6： 加入了延时的程序 3 代码

```
#include <p18cxxx.h>
#pragma config WDT = OFF

void delay (void)
{
  unsigned int i;
  for (i = 0; i < 10000; i++)
    ;
}

void main (void)
{
  TRISB = 0;

  while (1)
    {
      /* Reset the LEDs */
      PORTB = 0;
      /* Delay so human eye can see change */
      delay ();

      /* Light the LEDs */
      PORTB = 0x5A;
      /* Delay so human eye can see change */
      delay ();
```

4.4.7 编译程序 3

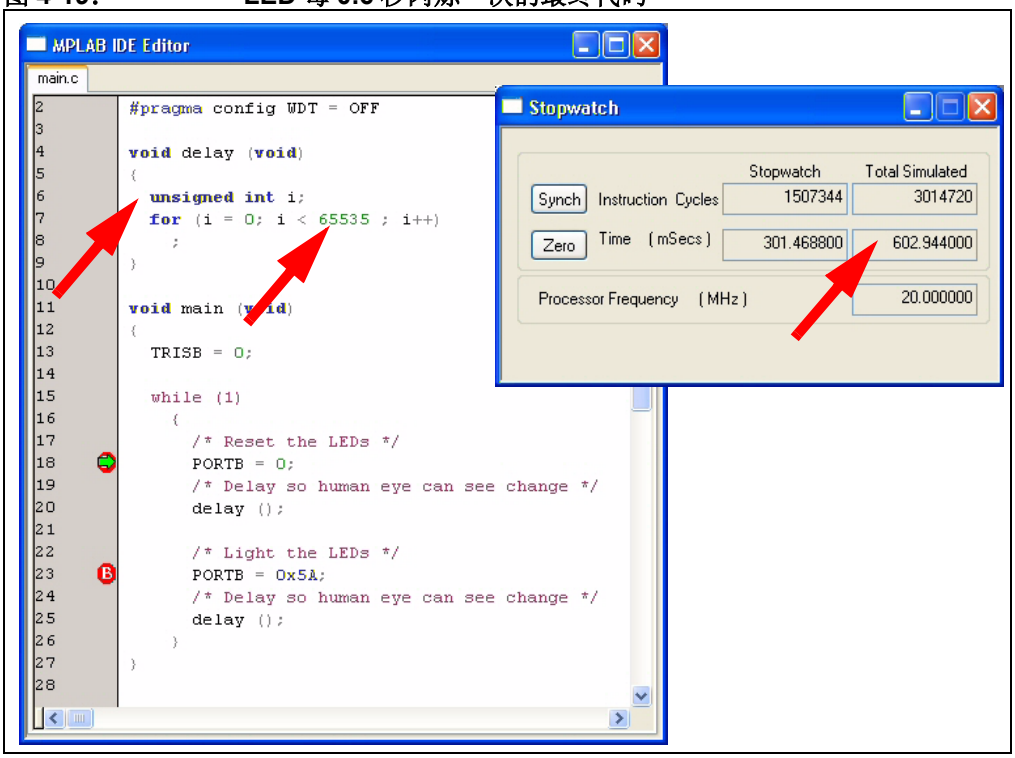
在将源代码作了如上修改后再次选择 *Project>Build All* 来重新编译项目，并向写有 PORTB 的第 18 行和第 23 行添加断点。使用跑表测试代码。

代码中其他位置可能还会显示前面设置的断点。使用鼠标右键菜单 **Remove Breakpoint**（移除断点），只留下需要的第 18 行和第 23 行的断点。

再次测定断点间的时间间隔。停止在第一个断点后，按下跑表上的 **Zero**（清零）按钮来从此断点开始测量。

如果变量 `i` 从 10000 开始向下计数，当其为零时测得的时间间隔约为 36 毫秒。记住当变量 `i` 被定义为 `int` 时，其范围为 -32768 至 32767（见《MPLAB® C18 C 编译器用户指南》）。其最大值（32767）将使延时增长约 3 倍。如果 `i` 被声明为 `unsigned int`，其范围可扩展至 65535，如图 4-19 所示。当设置为此值时，测得的延时将约为 301 毫秒，这意味着执行一次循环（有两段延时）需用约 602 毫秒。刚超过半秒，因此 LED 每秒约闪烁两次。请参见图 4-19。

图 4-19： LED 每 0.6 秒闪烁一次的最终代码



4.4.8 程序 3 总结

第三个程序示例到此结束。该示例涉及以下主题：

- 使用特定处理器的包含文件
- 设置软件模拟器处理器频率
- 设置断点
- 使用 MPLAB 跑表测量时间

4.5 使用演示板

本节使用硬件来演示前面的程序，而并非模拟。如果没有适合硬件，可跳过本节。本节所要求的硬件如下：

- MPLAB ICD 2 在线调试器
- J6 上安装有跳线的 PICDEM 2 Plus 演示板

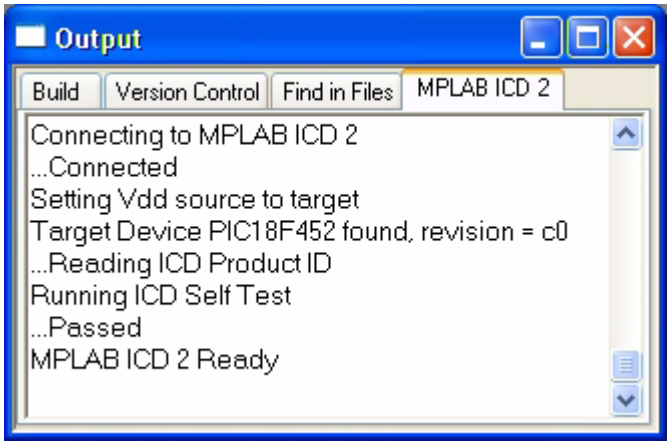
4.5.1 选择 MPLAB ICD 2

按照 MPLAB IDE 安装向导上的指导安装 MPLAB ICD 2。将 PICDEM 2 Plus 板与电源相连，将 MPLAB ICD 2 上的 ICD 电缆与演示板相连。

使用 *Debugger>Select Tool>MPLAB ICD 2* （调试器 > 选择工具 >MPLAB ICD 2）选择 MPLAB ICD 2 作为硬件调试器。选择 *Debugger>Connect* （调试器 > 连接）确保 MPLAB ICD 2 已经与 MPLAB IDE 建立了通信。

如果一切均已安装并正确连接， Output 窗口应与图 4-20 相似：

图 4-20： MPLAB® ICD 2 OUTPUT 窗口



注： 如果因为未找到 MPLAB ICD 2 或者 USB 端口无法打开而导致操作失败，请检查 MPLAB IDE 文档查看器 （MPDocViewer.exe，在 MPLAB 安装目录下的 Utilities 文件夹里），其中包含有安装 MPLAB ICD 2 的 USB 驱动程序的相关信息。

4.5.2 烧写代码以便用 MPLAB ICD 2 进行测试

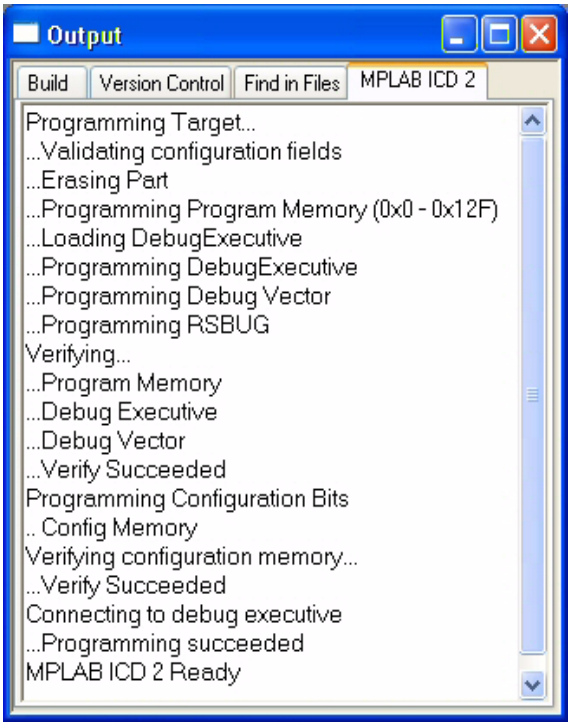
当改变调试器时，必须重新编译项目。点击 **Build All** 图标。

选择 *Debugger>Program* （调试器 > 编程）将程序下载到 PICDEM 2 Plus 演示板。

注： 当使用 MPLAB ICD 2 时，会提供特殊的链接描述文件，这样应用代码就不会使用供 MPLAB ICD 2 调试用的小存储区。这些链接描述文件的名称都以字符 “i” 结尾。对当前项目，使用名为 `18f452i.lkr` 的链接描述文件。在使用 MPLAB ICD 2 调试时总是使用名称含有 “i” 的链接描述文件。

Output 窗口应该显示与图 4-21 相似的文本：

图 4-21：　　编程后 MPLAB® ICD 2 的输出

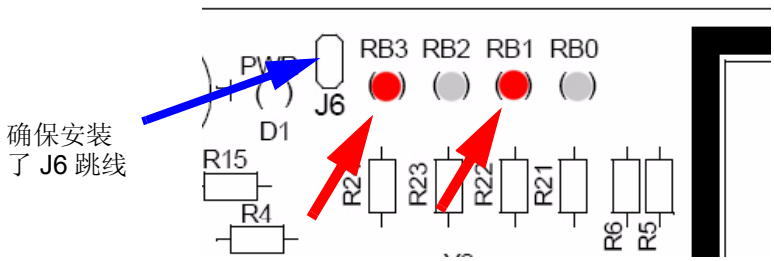


4.5.3　　在演示板上测试程序 3

选择 **Run** 图标。标识为 RB3 和 RB1 的 LED 应该会开始闪烁 （见图 4-22）。

注：　这些 LED 被标识为 RB0、 RB1、 RB2 及 RB3 是因为当 J6 上安装了跳线时它们与 PORTB 的引脚相连。PORTB 的 8 个端口引脚被称为 RBn，其中 n 的范围为 0 到 7。这些引脚中只有 4 个连接了 LED。

图 4-22：　　PICDEM™ 2 PLUS 演示板的顶部



由于 PORTB 的值在 0x5A 与 0x00 之间交替变换以控制 4 个 LED， RB1 和 RB3 应该闪烁，以表示 0x5A 的低半字节或值 0xA （二进制数 1010）。

LED 很可能闪烁得比前面在程序 3 中使用软件模拟器中的跑表时慢。这是因为 PICDEM 2 Plus 演示板带有的振荡器是 4 MHz 的，而使用软件模拟器时时钟频率为 20 MHz。延时循环的值可以减小到 10000 以加快闪烁。

注： 跑表是软件模拟器的一种调试功能。MPLAB ICD 2 不具备等效的功能。

4.5.4 对演示板上的处理器编程

当 MPLAB ICD 2 作为调试器工作时，可以单步执行程序，可将变量输入到 Watch 窗口，可以像在软件模拟器中一样运行和暂停程序。

当程序运行完全正常后，就可以将它烧写到目标器件中，从而无需连接 MPLAB ICD 2 和 PC 即可运行。

注： 当使用 MPLAB ICD 2 作为调试器时，会将特殊的代码下载到目标器件中，器件进入在线调试模式。对于最终应用，应关闭 MPLAB ICD 2 功能。

4.5.5 取消选定 MPLAB ICD 2 作为调试器

要取消 MPLAB ICD 2 作为调试器，请选择 *Debugger>Select Tool>None*（调试器 > 选择工具 > 无）。

4.5.6 设置 MPLAB ICD 2 作为编程器

选择 *Programmer>Select Programmer>MPLAB ICD 2*（编程器 > 选择编程器 >MPLAB ICD2）将 MPLAB ICD 2 使能为编程器。

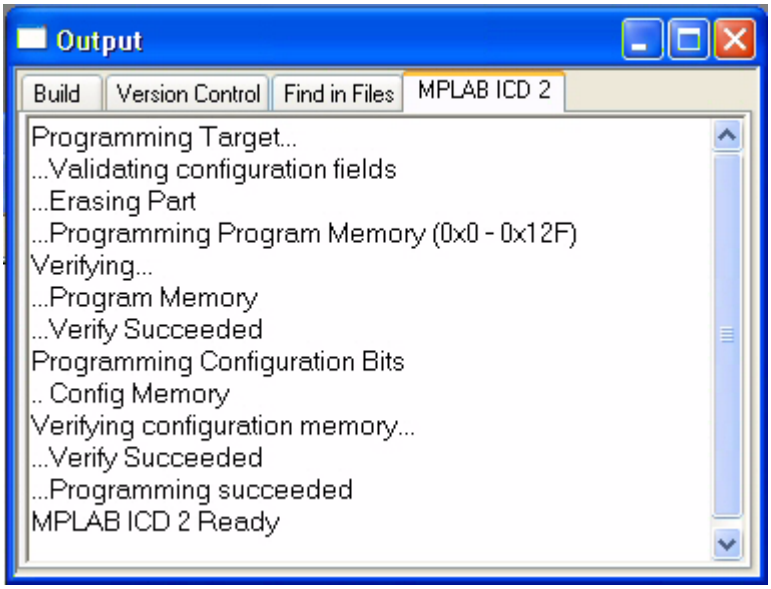
4.5.7 烧写器件

> **注：** 现在调试已完成，可使用原来的链接描述文件 `18f452.lkr` 来代替在使用 MPLAB ICD 2 进行调试时的 `18f452i.lkr`。

要将我们的代码下载并烧写到 PIC18F452 中，请选择 *Programmer>Program* （编程器 > 烧写）。

Output 窗口应该会显示结果 （见图 4-23）：

图 4-23： 烧写后 MPLAB® ICD 2 的输出



现在可以将 MPLAB ICD 2 与 PICDEM 2 Plus 断开。如果按下 PICDEM Plus 板上的 **RESET** （复位）按钮 （S1）， LED 就会像之前一样开始闪烁，表明固件已被成功地烧写到了最终应用中。

4.5.8 演示板使用总结

在演示板上实现较短的 C 程序的说明到此结束。本程序示例涉及以下主题：

- 选择 MPLAB ICD 2 作为调试器
- 使用 MPLAB ICD 2 调试演示板
- 选择 MPLAB ICD 2 作为编程器
- 使用 MPLAB ICD 2 将最终固件烧写到应用中。

第 5 章 特性

5.1 概述

本章将讲述可通过 MPLAB 用户界面控制的 MPLAB C18 的很多特性。范例项目将展示 MPLAB C18 和 MPLAB 调试器的一些特性。可通过将范例中的源代码复制并粘贴到 MPLAB 编辑器中创建所有这些项目。

本章涉及以下主题：

- MPLAB 项目编译选项
  - General 选项
  - Memory Model 选项
  - Optimization 选项
- 演示：代码优化
- 演示：在 Watch 窗口中显示数据
  - 基本数据类型
  - 数组
  - 结构
  - 指针
  - 映射文件

5.2 MPLAB 项目编译选项

MPLAB 项目管理器中含有控制 MPLAB C18 编译器、MPASM 汇编器及 MPLINK 链接器的设置。可为整个项目设置项目选项，也可为每个源文件分别调整项目选项。

项目编译选项具有如下选项卡来控制项目的语言工具选项。

- **General**（常规）——为项目设置路径。
- **MPASM/C17/C18 Suite**（MPASM/C17/C18 工具包）——将编译目标设置为标准或库。
- **MPASM Assembler**（MPASM 汇编器）——控制 MPASM 开关选项，如是否区分大小写、是否启用 PIC18XXXX 扩展模式、十六进制文件格式、警告及错误消息。
- **MPLINK Linker**（MPLINK 链接器）——确定 HEX 文件的格式，以及映射文件和调试输出文件的生成。
- **MPLAB C18** —— 设置 general、 memory model 及 optimization 选项。

> **注：** 可为每个文件分别定制编译设置。选择 *Project>Build Options...><file name>*（项目 > 编译选项 > 文件名）显示项目中各文件的选项。

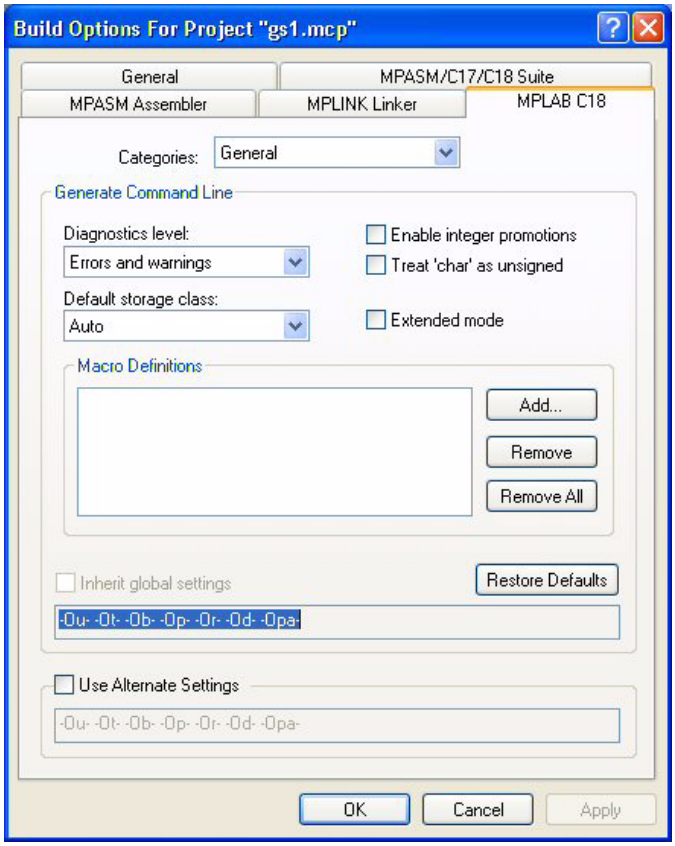
MPLAB C18 对话框具有 3 个分类，可从 Categories （分类）下拉菜单中选择。

- General 选项
- Memory Model 选项
- Optimization 选项

5.2.1 General 选项

选择 *Project>Build Options...>Project* 显示控制整个项目选项的对话框。**MPLAB C18** 选项卡具有以下设置 （见图 5-1）：

图 5-1： 项目的 General 选项对话框



Diagnostic level （诊断级别）——通过以下三个设置控制诊断输出：

- 仅输出错误
- 输出错误及警告
- 输出错误、警告及消息

Default storage class （默认存储类别）——设置局部变量的默认存储类别。可通过在定义各局部变量时将其声明为所希望的类别改写。

- Auto （自动）——这是默认设置，允许可重入的代码。这是在扩展模式中允许使用的惟一存储类别。
- Static （静态）——局部变量和参数将被静态分配，因而存取这些变量和参数所需代码量较小。仅在非扩展模式中才允许使用。
- Overlay （重叠）——局部变量及参数将被静态分配。此外，可能的话，局部变量将会与其他函数的局部变量重叠。仅在非扩展模式中才允许使用。

Enable integer promotions（启用整型提升）—— ANSI C 标准要求算术运算以 int 精度（16 位）或更高精度执行。禁用此选项有利于缩短应用程序的代码长度。若要求与 ANSI C 兼容，应选中此复选框。

Treat 'char' as unsigned（将 char 视为无符号型）—— 因为 PIC18XXXX 器件的数据总线是 8 位的，因而计算时通常使用 0 至 255（0xFF）之间的值。正常情况下， char 定义一个值在 -128 至 127 之间的变量。将普通的 char 视为无符号型（unsigned）将只允许在 0 至 255 之间的正值，在一些用 8 位单片机处理长度较短变量的应用中，这样做可能更适合于计算。

Extended mode（扩展模式）——允许使用 PIC18XXXX 扩展模式进行编译。当使用支持扩展模式的 PIC18XXXX 器件时，必须使用恰当的链接描述文件。扩展模式链接描述文件的名称以 "_e" 结尾，如 18f2520_e.lkr。

Macro Definitions（宏定义）——可通过 **Add**（添加）按钮向宏定义段添加宏。这与《MPLAB® C18 C 编译器用户指南》中 "简介" 部分描述的 -D 命令行选项是等效的。

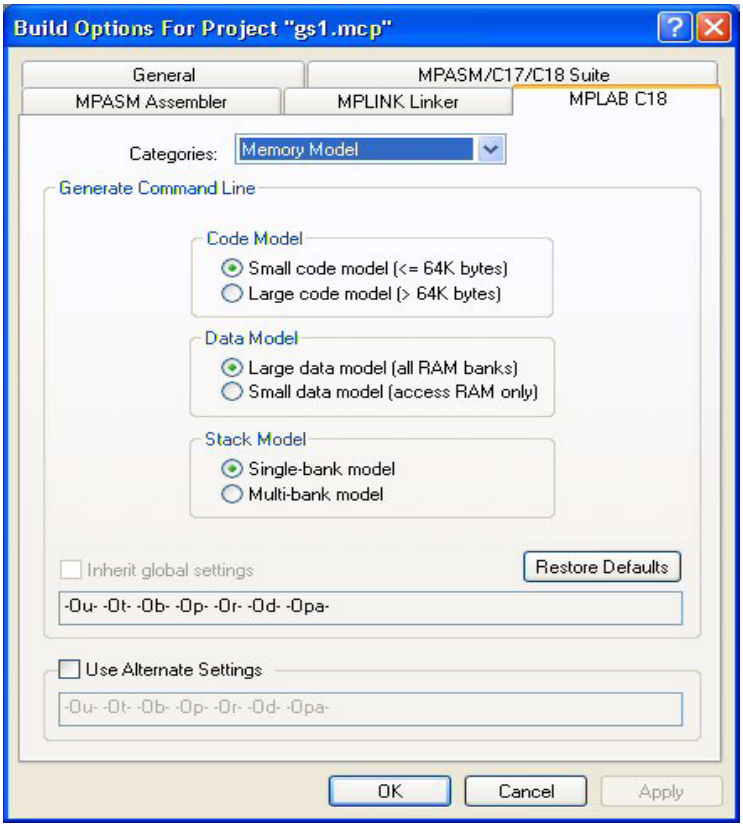
Inherit global settings（继承全局设置）——当选中此复选框时，文件会继承项目的所有设置。

Use Alternate Settings（使用其他设置）——当选中此复选框时，设置仅适用于当前文件。允许设置当前 MPLAB 对话框不支持的其他编译器命令行选项。更多有关编译器开关选项的信息请参见《MPLAB® C18 C 编译器用户指南》。

5.2.2 Memory Model 选项

该对话框可单独控制编译器的存储模型 （见图 5-2）。

图 5-2： MEMORY MODEL 选项对话框



Code Model （代码模型）—— 将程序存储器指针的默认长度设置为 16 位或 24 位。可通过声明该指针为 `near` （16 位）或 `far` （24 位）改写默认设置。使用 16 位指针（小代码模型）可提高代码效率，但如果指针指向的是程序存储器大于 64 KB 的器件中的程序存储数据 （`romdata`），就应该使用 24 位指针 （大代码模型）。

Data Model （数据模型）—— 默认数据段 （`idata` 和 `udata`）位于快速操作 RAM （Access RAM ）中 （小代码模型）或分区 RAM （大代码模型）中。可通过将各变量声明为 `near` 或 `far` 并在正确的存储区中创建段来改写特定变量的位置。

> **注：** 小代码模型只能用于非扩展模式。

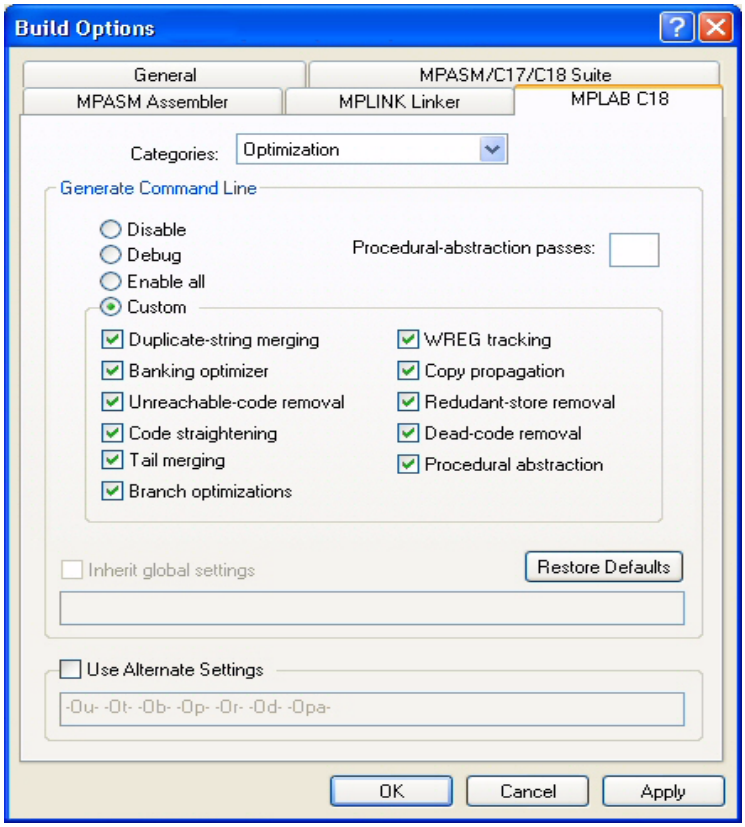
Stack Model （堆栈模型）——设置数据堆栈是否能延伸到一个存储区之外。堆栈的大小和位置在链接描述文件中设置。如果链接描述文件将堆栈定义为延伸到单个存储区外，此堆栈应该被设置为 “Multi-bank model”（多存储区模型）。如果使用的堆栈较大，由于数据堆栈指针需要按 16 位而不是 8 位进行算术运算，因而会使性能略微下降。

5.2.3 Optimization 选项

该对话框可单独控制编译器的各种优化（见图 5-3）。有关每种优化的详细信息，请参见《MPLAB® C18 C 编译器用户指南》中的“优化”一章。

通常来说，在调试代码时，推荐使用 **Debug**（调试）设置。

图 5-3： OPTIMIZATION 选项对话框



可通过 **Generate Command Line**（生成命令行）下的单选按钮控制优化。有关各种优化的详细信息，请参见《MPLAB® C18 C 编译器用户指南》中的“优化”一章。有四种设置：

Disable（禁止）——禁止所有优化

Debug —— 启用大多数优化，但禁止一些不利于调试的优化，特别是合并相同字符串、代码排序及 WREG 跟踪。

Enable all（启用所有）——启用所有优化。

Custom（定制）—— 启用选中的优化。

Procedural-abstraction passes（过程抽象次数）——可多次执行过程抽象优化。默认情况下运行 4 次过程抽象。也可尝试运行更多次过程抽象以进一步缩短代码，但这样做可能会产生太多同时被抽象的函数，导致运行时返回堆栈溢出。可设置过程抽象次数少于 4 次以使对返回堆栈的影响最小。

5.3 演示：代码优化

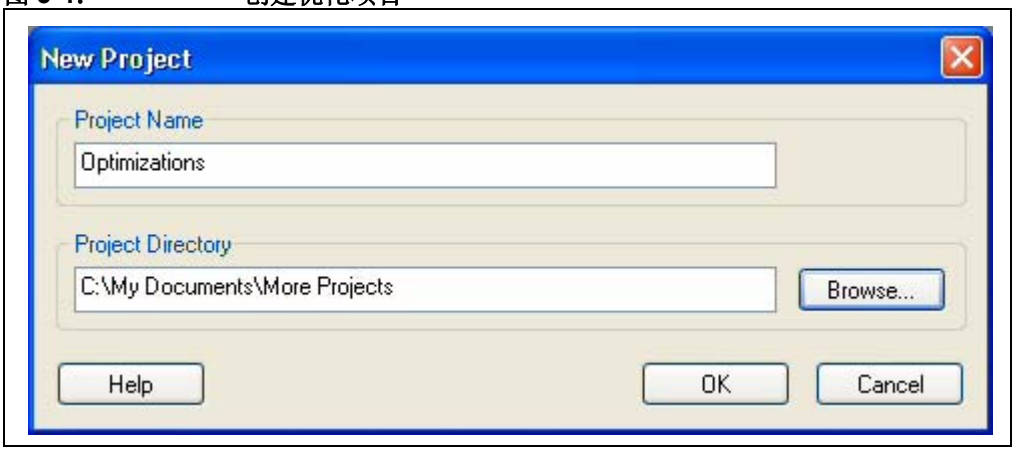
本节将通过一个示例讲述代码优化如何影响项目调试。在无优化的情况下创建并编译代码。单步执行代码演示代码的预期行为。

之后，代码将被优化并会看到单步执行仍能产生正确的操作，但是代码的执行流程发生了改变（被优化），使得调试更加困难。

5.3.1 创建优化项目

要进行演示，通过 *Project>New...*（项目 > 新建）在 MPLAB IDE 中创建一个新项目（见图 5-4）。将其命名为 “Optimizations” 并创建一个名为 “More Projects” 的新项目目录。

图 5-4： 创建优化项目



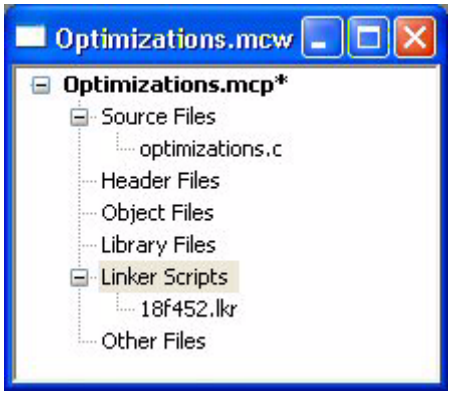
使用 *File>New*（文件 > 新建）创建一个新文件并复制或键入以下代码（例 5-1）。使用 *File>Save*（文件 > 保存）将其以文件名 optimizations.c 保存在 More Projects 目录中。

例 5-1：　　优化代码

```
#include <stdio.h>

void main (void)
{
  int j = 0;
  int i;

  for (i = 0; i < 10; i++)
    {
      printf ("%d:\t", i);

      if (i % 2)
        {
          printf ("ODD");
          j += i;
        }
      else
        {
          printf ("EVEN");
          j += i;
        }

      printf ("\tj = %d\n", j);
    }

  while (1)
    ;
}
```

右击项目窗口中的 Source Files 并将源文件 optimizations.c 添加到项目中。右击项目窗口中的 Linker Scripts，添加链接描述文件 18F452.lkr （见图 5-5）。

图 5-5：　　优化项目



5.3.2　　启用软件模拟器

按如下方法设置软件模拟器：

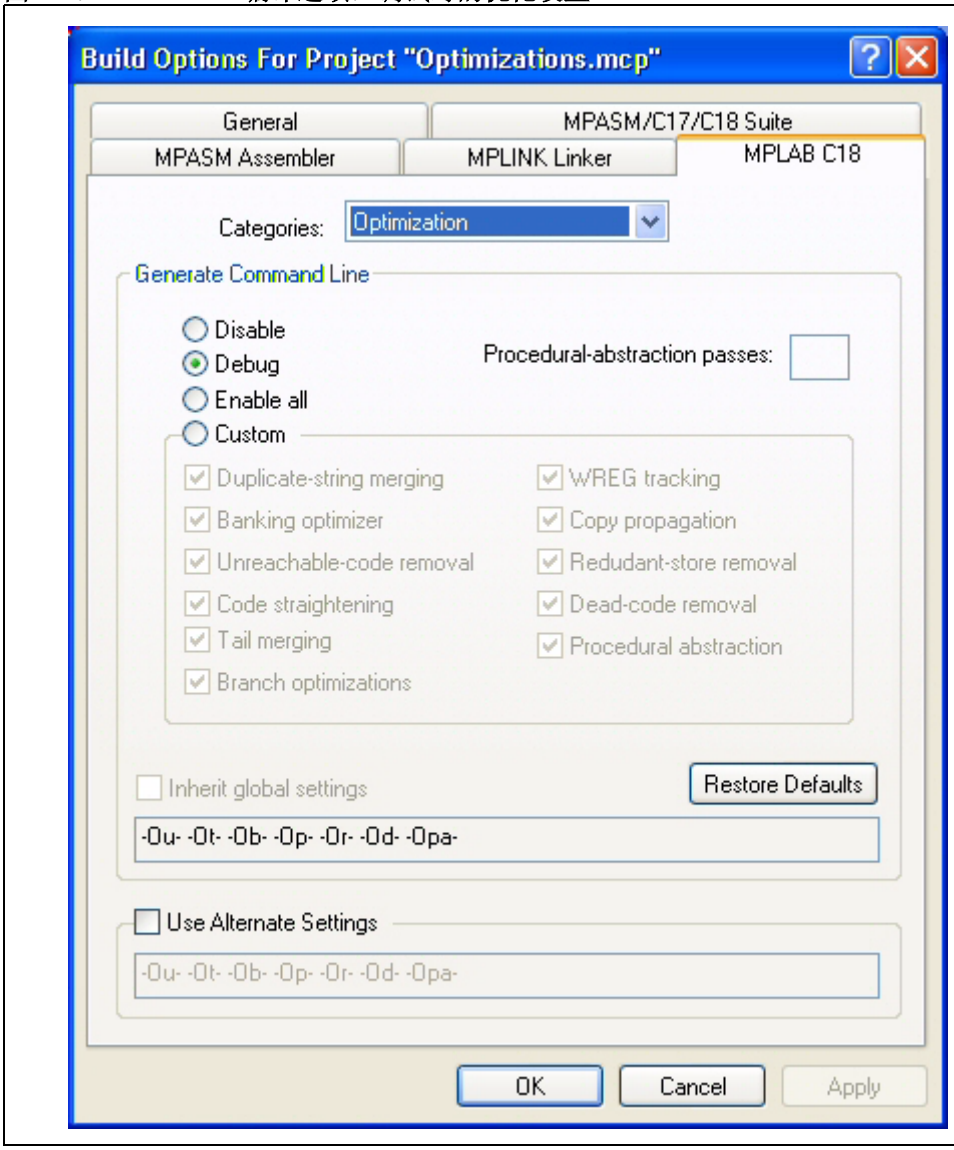
- 通过 *Debugger>Select Tool>MPLAB SIM* 将软件模拟器设置为当前调试器。

5.3.3 关闭优化

按以下步骤设置调试所需的编译选项：

- 选择 *Project>Build Options>Project* 弹出编译对话框。
- 选择 **MPLAB C18** 选项卡并选择 **Categories: Optimizations** 显示如下所示的对话框。
- 像图 5-6 所示选中 **Debug** 以禁止不利于调试的优化。

图 5-6： 编译选项：调试时的优化设置



5.3.4 检查设置

- 双击 **General** 选项卡并查看包含路径及库路径是否如**第 3.7 节 “检查安装和编译选项”**那样正确设置。
- 还要检查 *Debugger>Settings* （调试器 > 设置）并单击 **Uart1 IO** 选项卡。确保选中了 **Enable Uart1 IO** （使能 Uart1 IO）复选框并且 **Output** （输出）被设置为 **Window** （窗口）。

5.3.5 编译并测试项目

使用 *Project>Build All* 或工具栏中的 **Build All** 图标来编译项目。

单击 **Run** 图标并检查 Output 窗口以查看代码是否正确执行。Output 窗口应该显示：

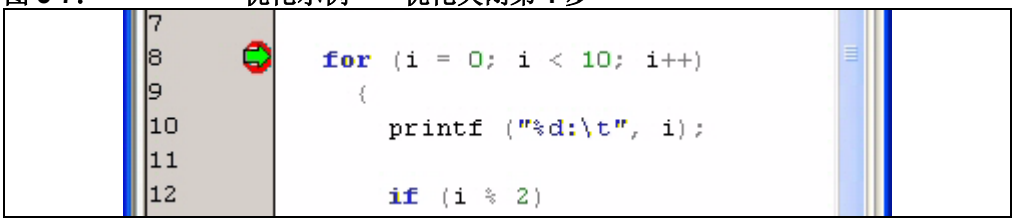
```
0:  EVEN    j = 0
1:  ODD     j = 1
2:  EVEN    j = 3
3:  ODD     j = 6
4:  EVEN    j = 10
5:  ODD     j = 15
6:  EVEN    j = 21
7:  ODD     j = 28
8:  EVEN    j = 36
9:  ODD     j = 45
```

5.3.6 单步执行代码

先后单击 **Halt** 图标和 **Reset** 图标以确保准备从头开始执行代码。

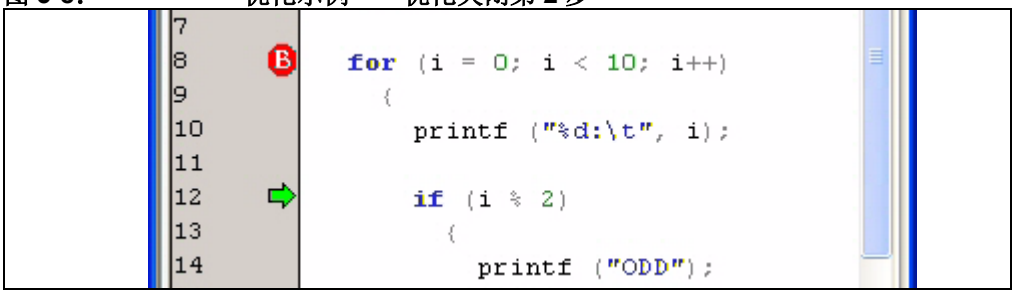
在 “for” 循环处设置断点并单击 **Run** 图标使程序在主循环开始处暂停，如图 5-7 所示：

图 5-7：　　优化示例——优化关闭第 1 步



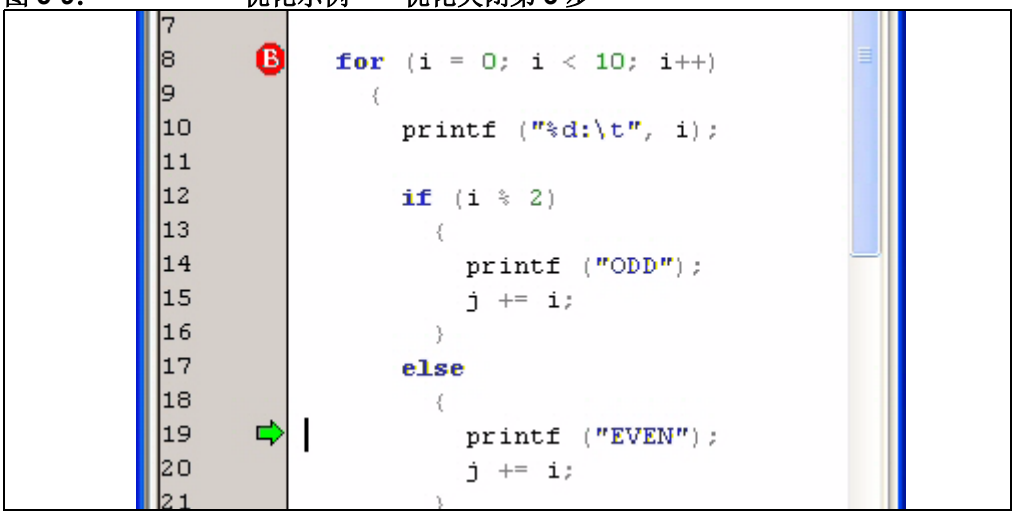
单击 **Step Over** 按钮开始单步执行代码。再次单击 **Step Over** 使代码执行到 “if” 语句 （见图 5-8）。

图 5-8：　　优化示例——优化关闭第 2 步



再次单击 **Step Over** 使代码执行到该语句的 “else” 部分。模运算 （%）的结果在第一次循环时为假，如 Output 窗口所示， Output 窗口的第一行显示为 “Even”（见图 5-9）。

图 5-9：　　优化示例——优化关闭第 3 步



继续单击 **Step Over** 使代码再次执行到 “if” 语句 （见图 5-10、图 5-12、图 5-13 和图 5-14）。

图 5-10：　　优化示例——优化关闭第 4 步

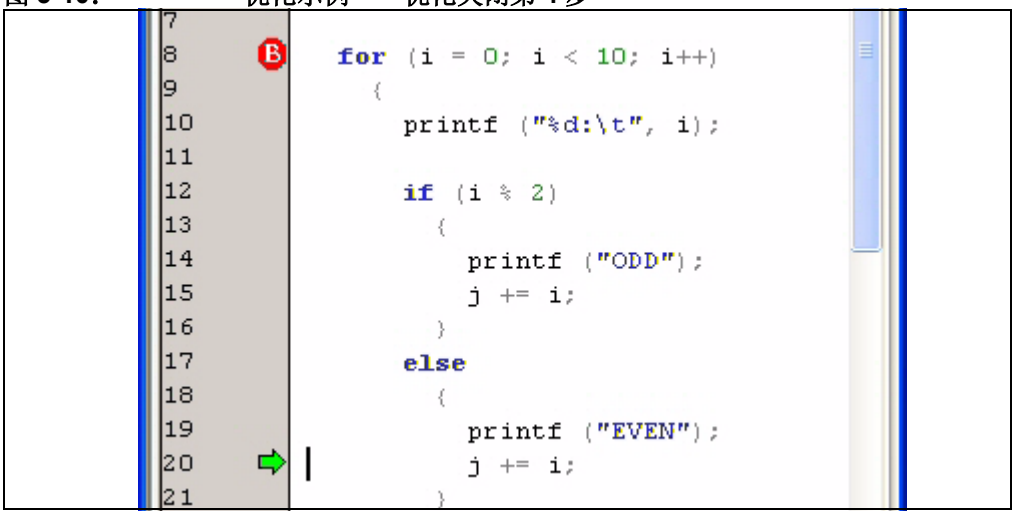


图 5-11：　　优化示例——优化关闭第 5 步

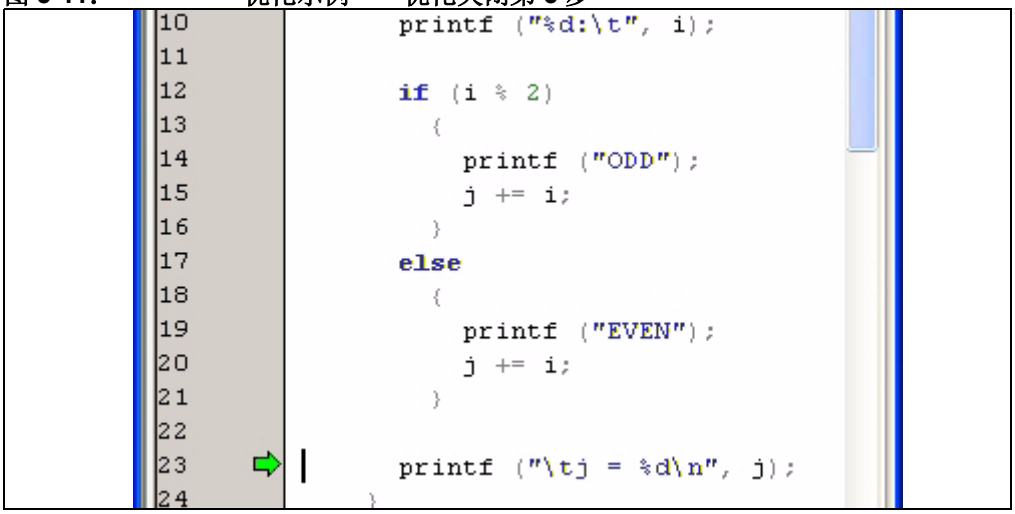


图 5-12：　　优化示例——优化关闭第 6 步

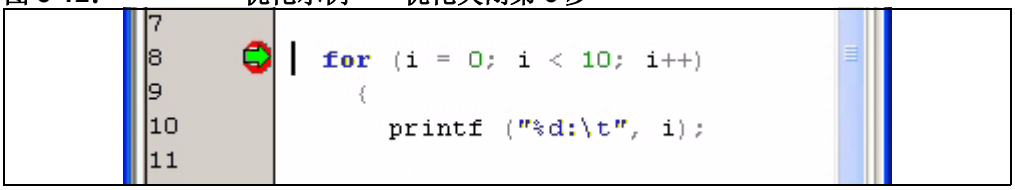


图 5-13：　　优化示例——优化关闭第 7 步

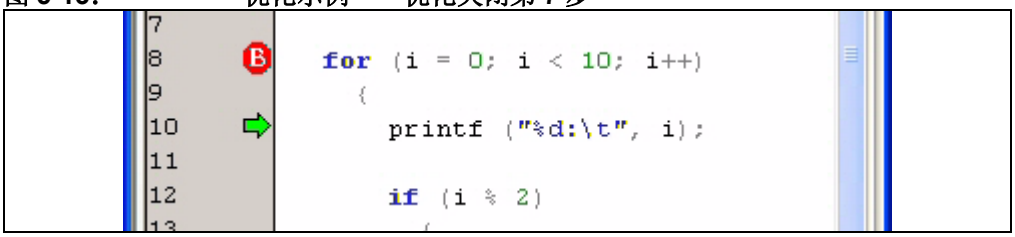
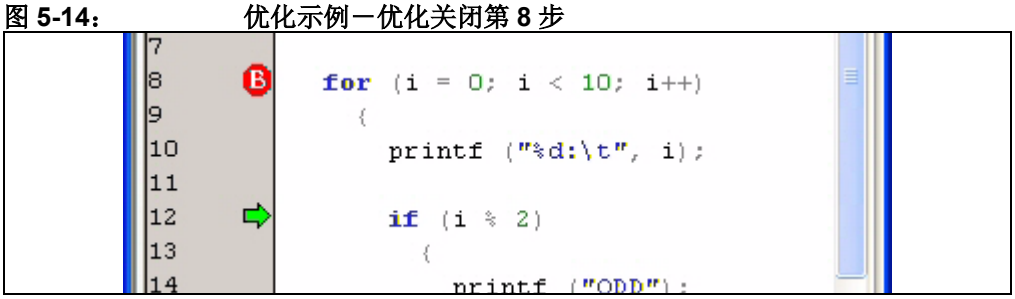


图 5-14：　　优化示例－优化关闭第 8 步



继续单击 **Step Over** 使代码执行到第 15 行，即函数 “`if`” 部分的 “`j += i`” 语句（见图 5-15 和图 5-14）。

图 5-15：　　优化示例——优化关闭第 9 步

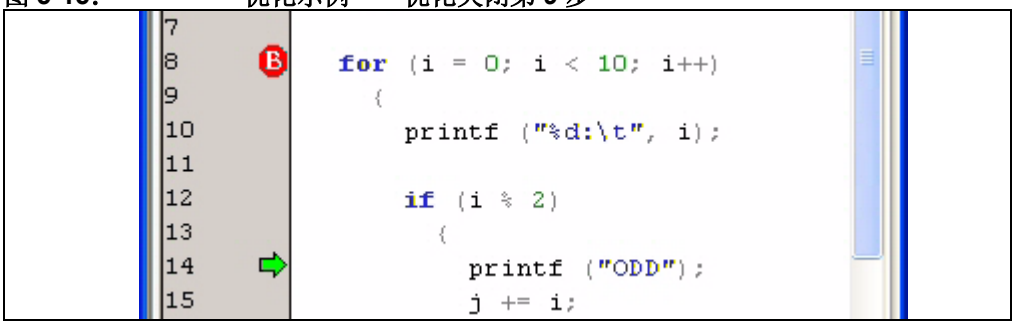
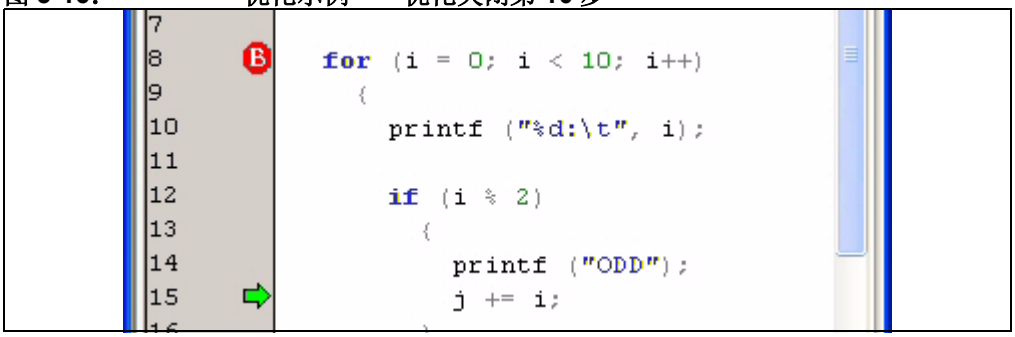


图 5-16：　　优化示例——优化关闭第 10 步

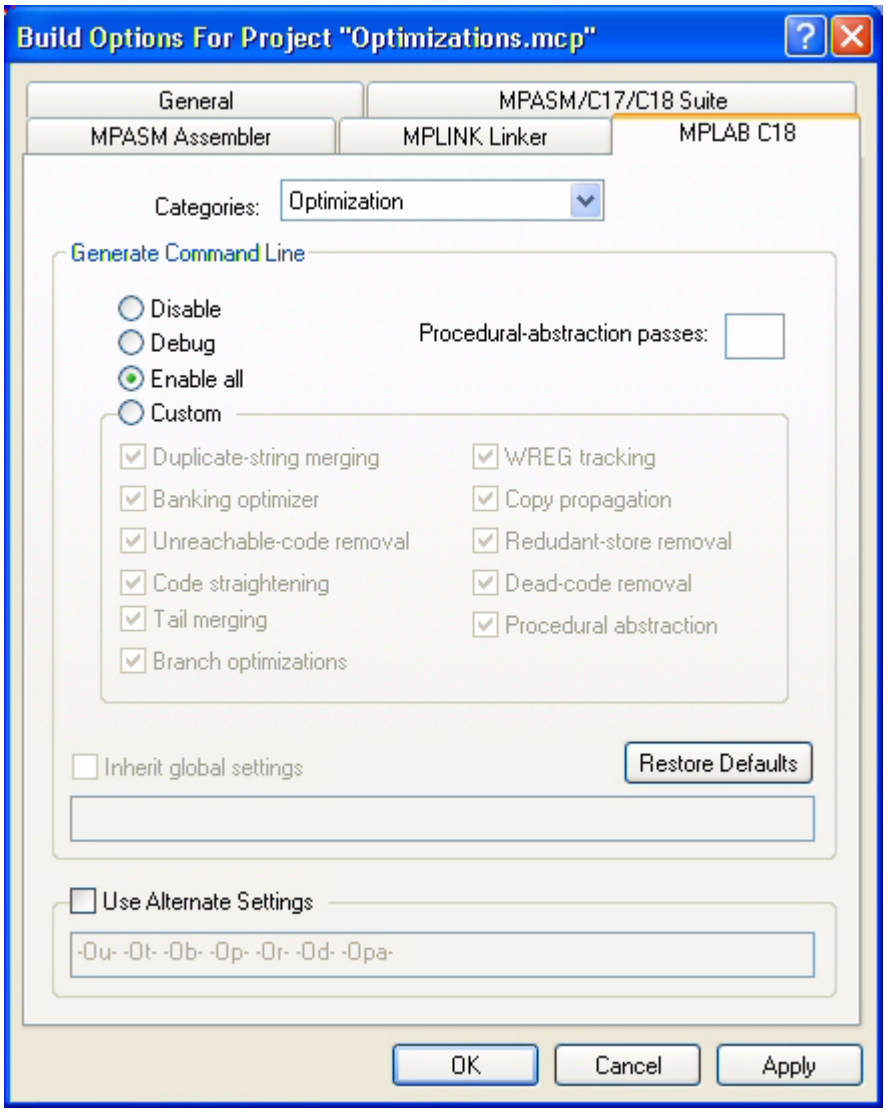


这一节演示了无优化情况下的代码运行。如同预期的那样，单步代码执行符合逻辑地进行着。下一节将展示优化后的代码行为。

5.3.7 启用优化

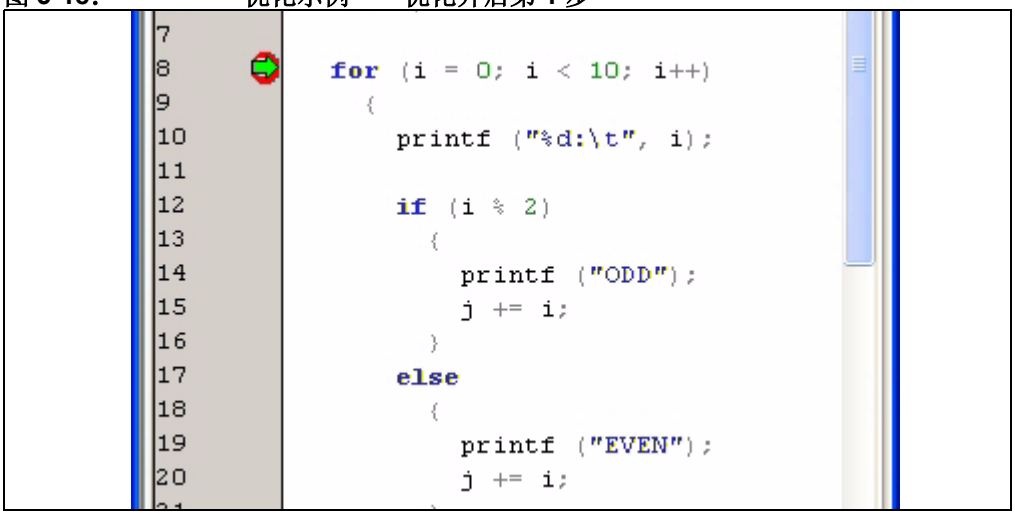
选择 *Project>Build Options>Project* 对话框，然后选择 **MPLAB C18** 选项卡（见图 5-17）。选择 **Categories: Optimization** 下拉菜单，然后选中 **Enable all** 单选按钮。

图 5-17： MPLAB® C18 编译选项：优化开启



使用 **Build All** 图标重新编译项目。如果还没有设置断点的话，在第 8 行的 “for” 语句处设置断点，然后使用 **Step Over** 图标单步执行代码 （见图 5-18 到图 5-26）。

图 5-18：　　优化示例——优化开启第 1 步



和以前一样，使用 **Step Over** 图标单步执行代码。

图 5-19：　　优化示例——优化开启第 2 步

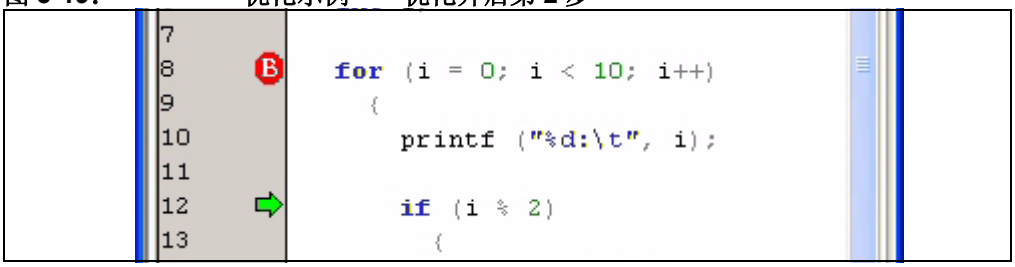


图 5-20：　　优化示例——优化开启第 3 步

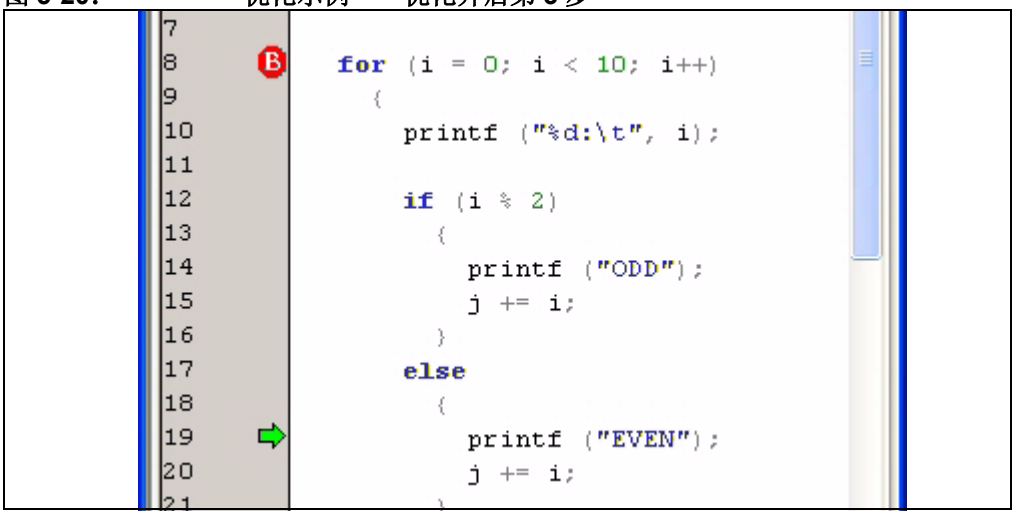


图 5-21：　　优化示例——优化开启第 4 步

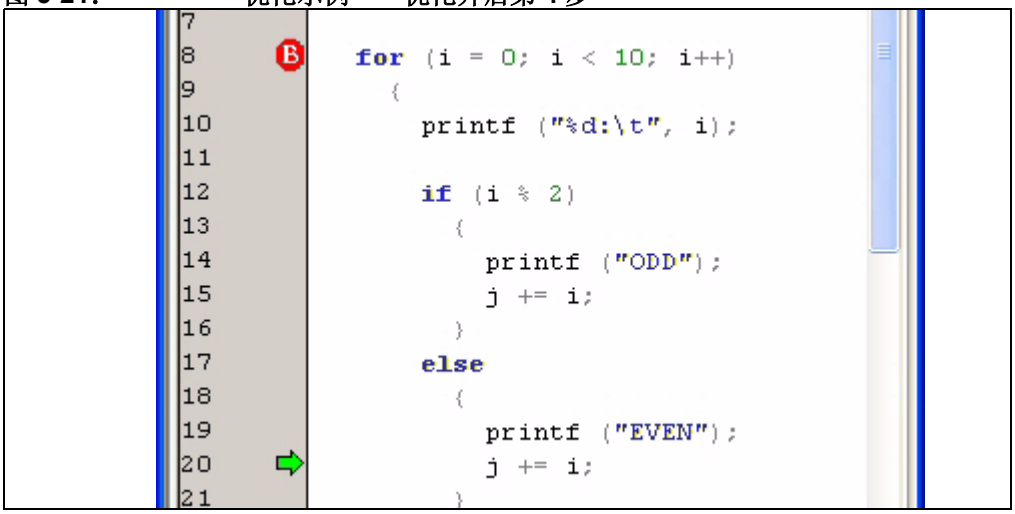


图 5-22：　　优化示例——优化开启第 5 步

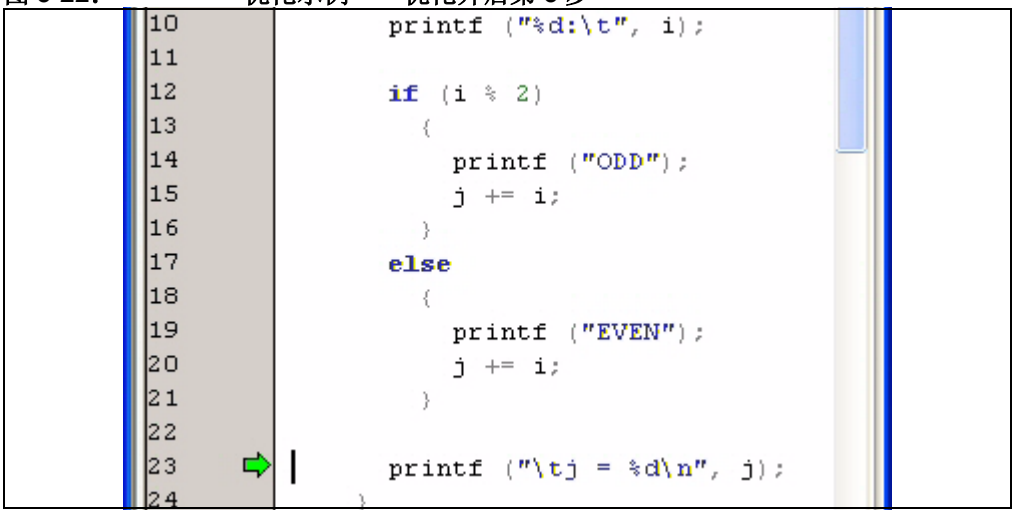


图 5-23：　　优化示例——优化开启第 6 步

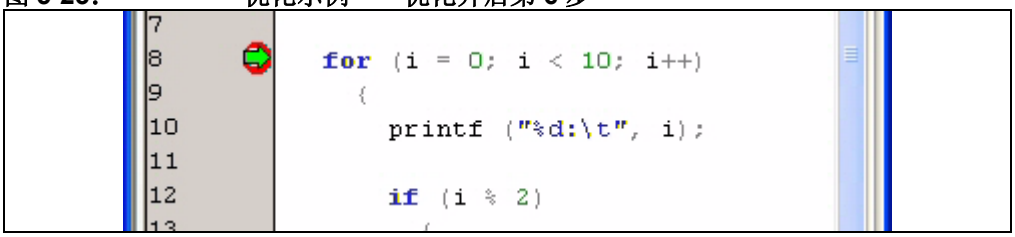


图 5-24：　　优化示例——优化开启第 7 步

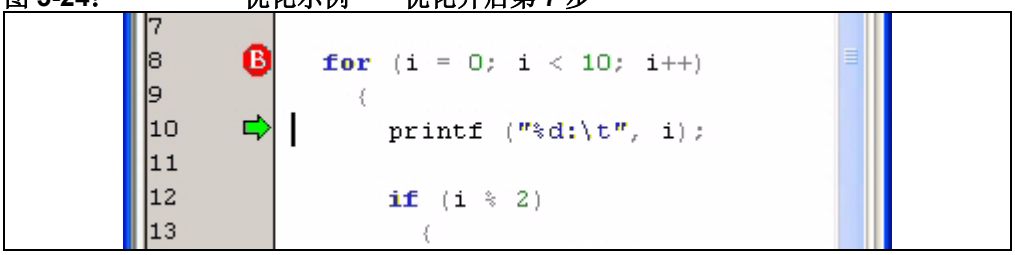


图 5-25：　　优化示例——优化开启第 8 步

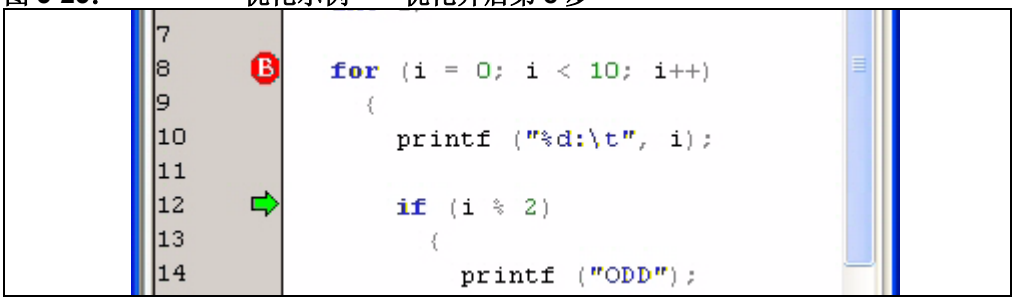
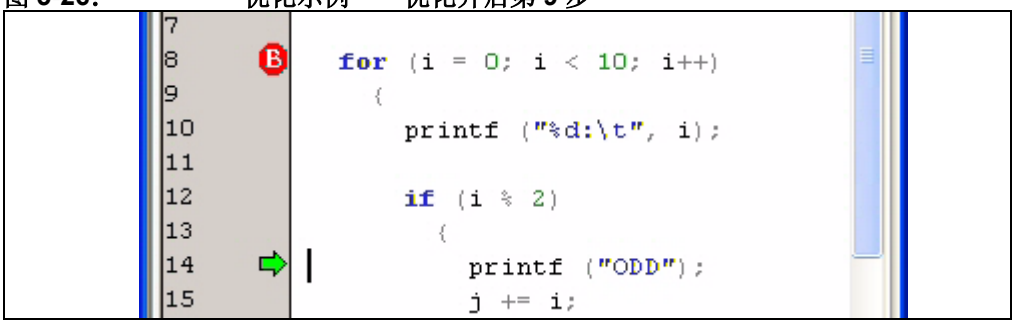
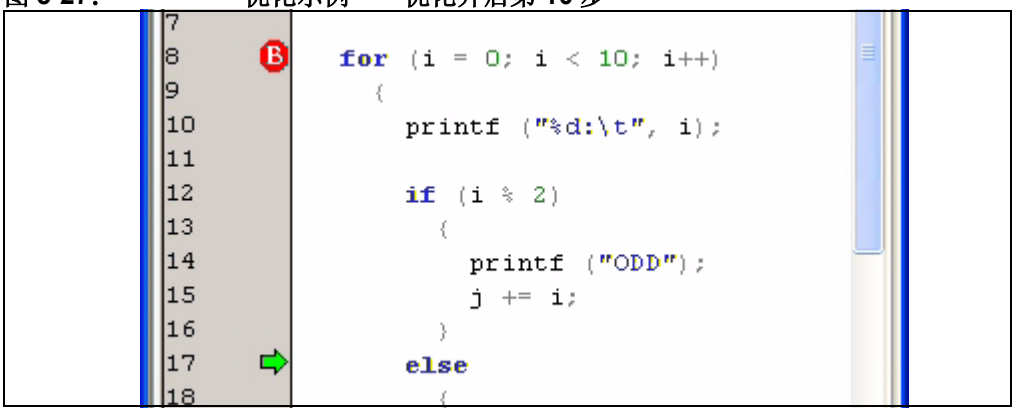


图 5-26：　　优化示例——优化开启第 9 步



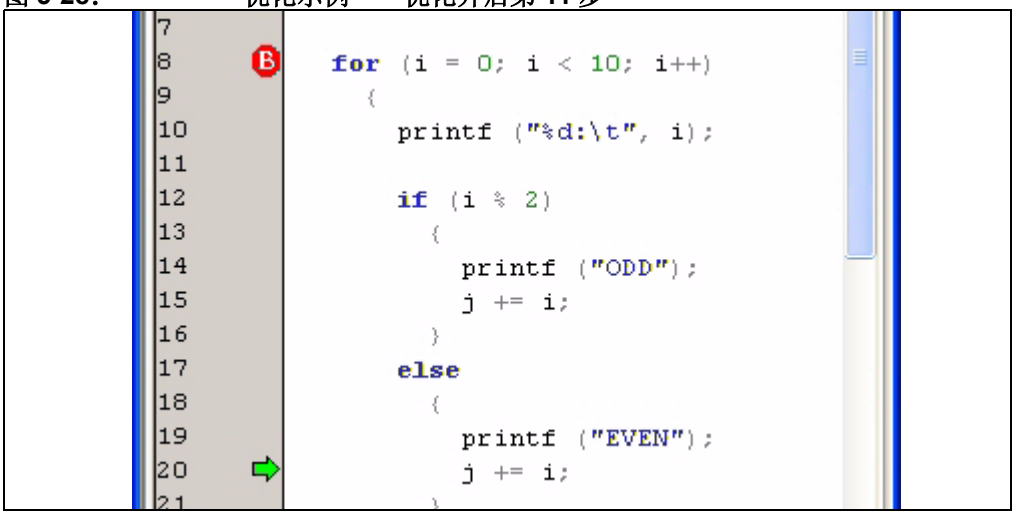
再一次 **Step Over** 会产生令人吃惊的结果。上图中，程序计数器箭头在函数的 “if” 部分，而现在程序计数器却跳到了函数的 “else” 部分 （见图 5-27）。

图 5-27：　　优化示例——优化开启第 10 步



再次单击 **Step Over** 会看到将要执行代码中 “else” 部分的 “j += i” 语句（见图 5-28）。

图 5-28：　　优化示例——优化开启第 11 步



这一节演示了代码已经使用 “尾部合并 （tail merging）” 技术进行了优化。代码 “if” 与 “else” 部分的 “j += i” 语句由原来分离的两组代码优化为了一组代码。

单步执行代码时，代码跳到了 “else” 子句部分，但实际执行的仍是代码的 “if” 部分。上图中第 15 行的第一个 “j += i” 语句已被删除。这一优化在调试时会导致令人困惑的结果。代码仍按照设计的那样执行，但是编译器将其重新组织了以产生更少的指令。

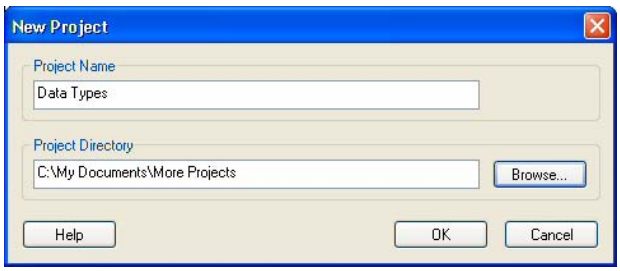
5.4　演示：在 WATCH 窗口中显示数据

5.4.1　基本数据类型

MPLAB C18 中的变量可以被添加到 Watch 窗口中。MPLAB IDE 可用适合每种数据类型的格式正确显示它们的值。

可以使用一个新的项目完成此演示。选择 *Project>New Project*，将项目名称设置为 “Data Types” 并将项目目录设置为在上一演示中使用的同一目录 （见图 5-29）。

图 5-29：　　演示：数据类型



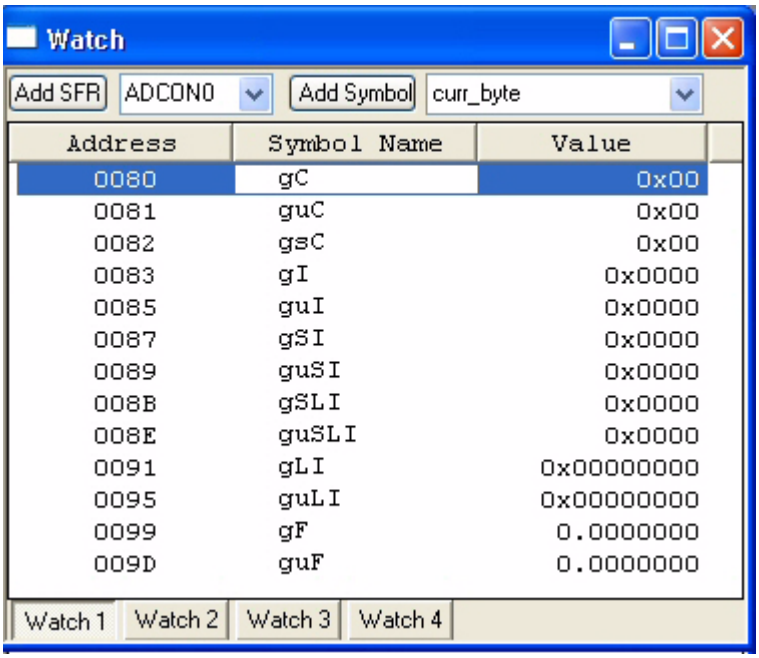
该示例代码（例 5-2）使用 MPLAB C18 的基本数据类型。使用 *File>New* 创建新文件，将其保存为 “basic_types.c”，并将这个文件和 18F452.lkr 链接描述文件添加到项目中。

例 5-2：　　数据类型代码

```
char gC;
unsigned char guC;
signed char gsC;

int gI;
unsigned int guI;

short int gSI;
unsigned short int guSI;

short long int gSLI;
unsigned short long int guSLI;

long int gLI;
unsigned long int guLI;

float gF;
unsigned float guF;

void main (void)
{
  gC  = 'a';
  guC = 'b';
  gsC = 'c';

  gI  = 10;
  guI = 0xA;

  gSI  = 0b1010;
  guSI = 10u;

  gLI  = 0x1234;
  guLI = 0xFA5A;

  gF  = -1.395;
  guF = 3.14;

  while (1)
    ;
}
```

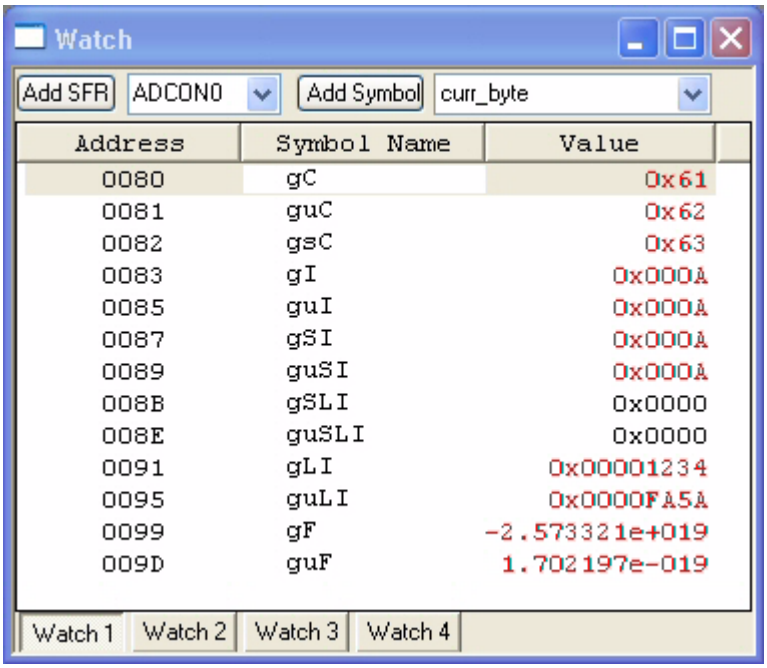
选择 *View>Watch* 显示 Watch 窗口，并通过选中源代码中的变量并将其拖放到 Watch 窗口来添加变量 （见图 5-30）。

图 5-30： 数据类型 WATCH 窗口



编译项目；先后单击 **Run** 和 **Halt**。将会按照 `basic_types.c` 中的设置显示变量的值。那些发生变化的变量将以红色显示，如图 5-31 所示。

图 5-31： 运行后的数据类型 WATCH 窗口



5.4.2 数组

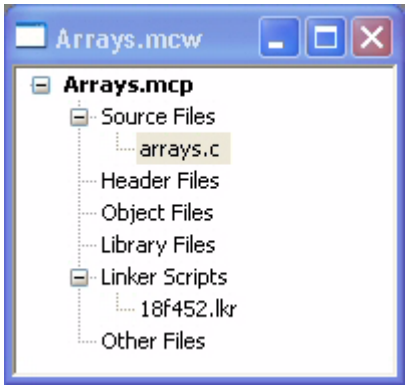
数组在 MPLAB C18 Watch 窗口中以可折叠的项显示，允许其在需要被查看时展开，而在观察其他变量时折叠起来，以提供更多的空间。为了演示，使用下列代码（例 5-3）创建一个新的名为 arrays.c 的源文件和一个名为 “Arrays” 的新项目。

例 5-3： 数组代码

```
char x[] = "abc";
int i[] = { 1, 2, 3, 4, 5};

void main (void)
{
  while (1);
    ;
}
```

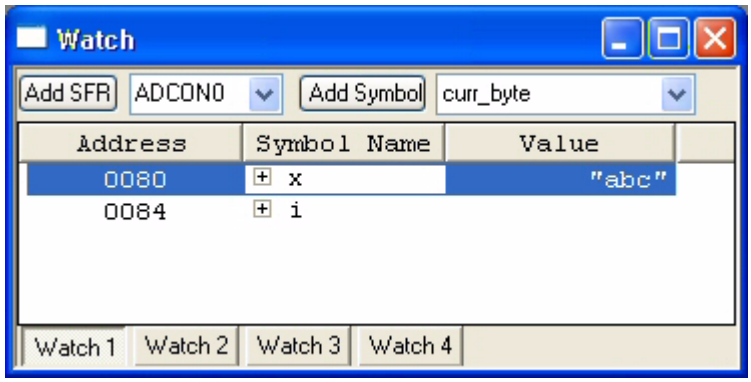
将名为 array.c 的文件和链接描述文件 18F452.lkr 添加到这个项目中 （见图 5-32）。

图 5-32： 数组项目



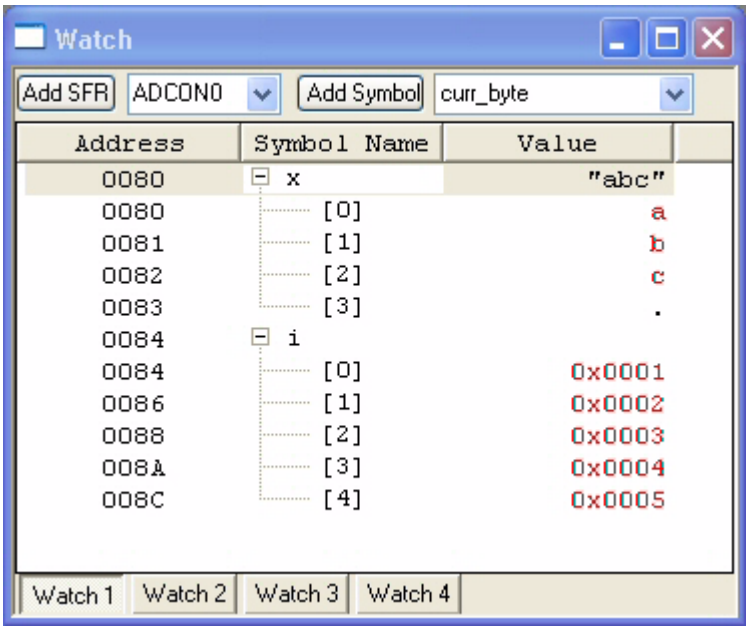
选择 *View>Watch* 打开一个 Watch 窗口并将名为 “x” 和 “i” 的数组拖放到 Watch 窗口中 （见图 5-33）。

图 5-33： 数组 Watch 窗口



确保选择了软件模拟器作为调试器，编译项目并运行程序。随后，单击 **Halt** 可查看数组。在图 5-34 中，每个数组旁的 “+” 号被展开了。注意程序执行后数组中值。

图 5-34：　　展开的数组



5.4.3　　结构

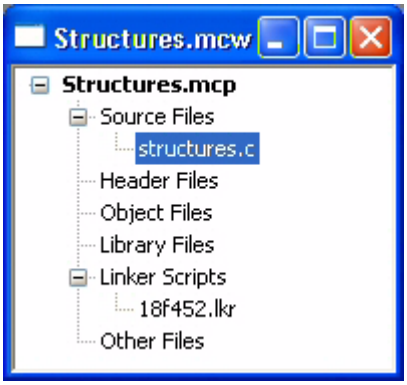
和数组一样，MPLAB C18 中的结构在 Watch 窗口中也显示为可展开 / 可折叠的成员。例 5-4 中的演示代码将用来演示结构在 MPLAB C18 Watch 窗口中的显示方式。

例 5-4：　　结构代码

```
struct {
  int x;
  char y[4];
} s1 = { 0x5A, “abc” };

struct {
  int x[5];
  int y;
} s2 = { { 10, 22, 30, 40, 50 }, 0xA5 };

void main (void)
{
  while (1)
    ;
}
```

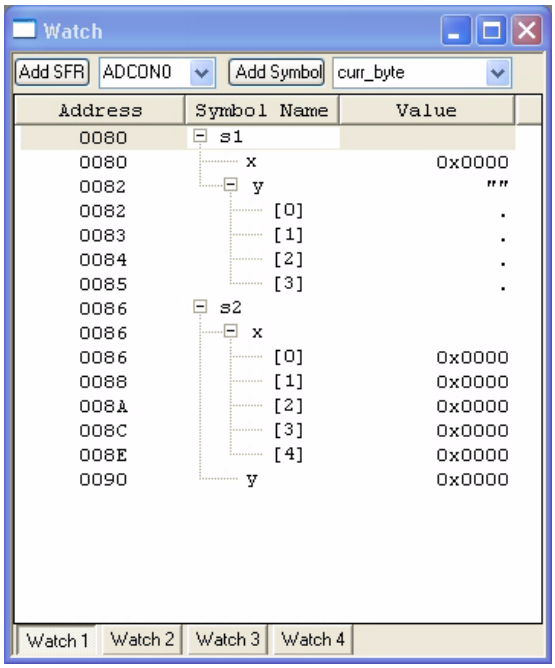
用该源文件创建一个项目 （见图 5-35），向其中添加 18F452.lkr 链接描述文件，然后将软件模拟器设置为调试器并编译项目。

图 5-35： 结构：项目



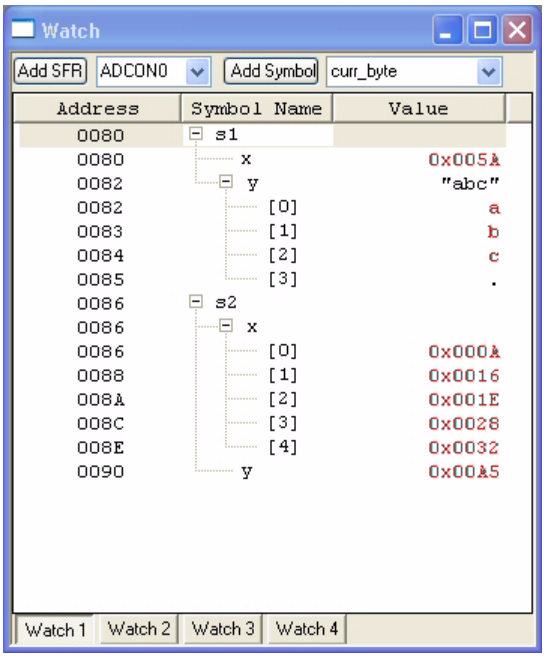
程序运行之前， Watch 窗口应如图 5-36 所示，此时所有的成员都是展开的：

图 5-36： 结构：WATCH 窗口



在单击 **Run** 之后单击 **Halt**， Watch 窗口应该显示储存在结构中的值 （见图 5-37）：

图 5-37： 结构：代码执行后的 WATCH 窗口



5.4.4 指针

MPLAB C18 中的指针可用来指向 ROM 或 RAM 中的数据。本演示使用 3 个指针，说明了它们在 PIC 18 架构中的使用方法。

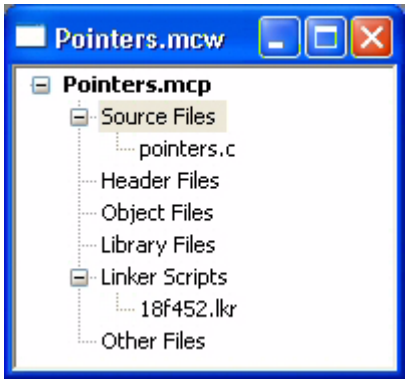
源代码如例 5-5 所示。将其输入到 MPLAB IDE 中的新文件中，并以文件名 “pointers.c” 保存在 “More Projects” 文件夹中。

例 5-5： 指针代码

```
ram char * ram_ptr;
near rom char * near_rom_ptr;
far rom char * far_rom_ptr;

char ram_array[] = "this is RAM";
rom char rom_array[] = "this is ROM";

void main (void)
{
  ram_ptr = &ram_array[0];
  near_rom_ptr = &rom_array[0];
  far_rom_ptr = (far rom char *)&rom_array[0];

  while (1)
    ;
}
```

创建一个名为 “Pointers”的新项目，将 pointers.c 文件作为源文件添加到项目中，并在项目中添加 18F452.lkr 链接描述文件。项目应如图 5-38 所示：

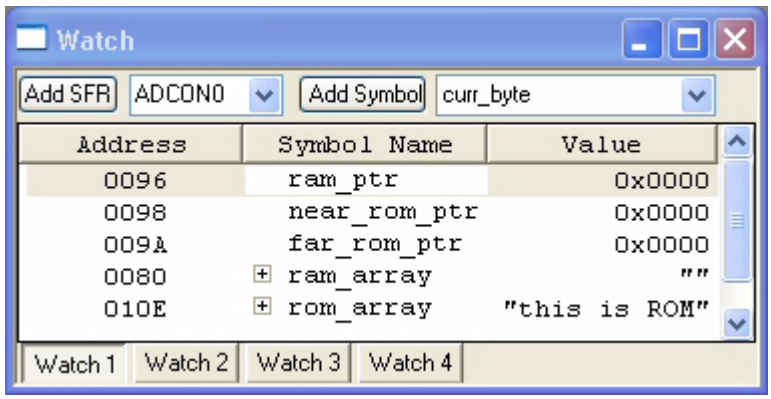
图 5-38： 指针：项目



选择 *Project>Build All* 编译项目。先不要运行项目。

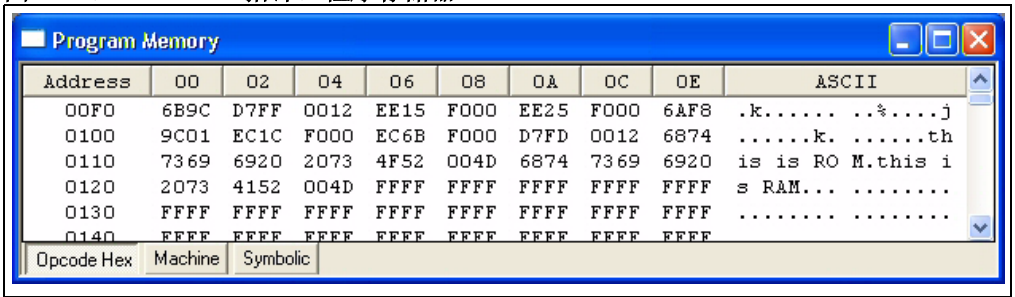
选择 *View>Watch* 打开一个空的 Watch 窗口，然后选中源代码窗口中的 3 个指针和 2 个数组并将它们拖放到 Watch 窗口中，如图 5-39 所示。

图 5-39： 指针：程序运行前的 WATCH 窗口



在执行此演示程序之前，不妨看看程序存储器。注意 Watch 窗口显示名为 “rom_array” 的数组位于程序存储器地址 0x010E 处。选择 *View>Program Memory* （视图 > 程序存储器）打开 Program Memory 窗口，向下滚动以查看 0x010E 附近的地址 （见图 5-40）。

图 5-40： 指针：程序存储器



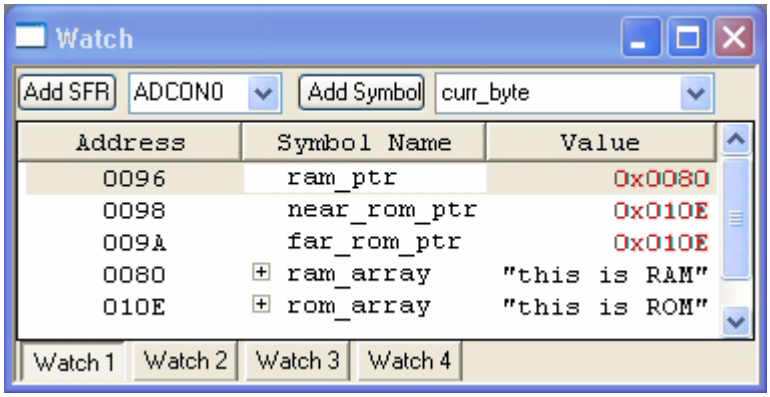
注：	确保选择了窗口底部的 **Opcode Hex** （十六进制操作码）选项卡。

在位于地址 0x010E 处的名为 “rom_array” 的 ROM 数组中设置的文本明确地将 “this is ROM” 文本存储到程序存储器中。

注： 在图 5-40 中，RAM 数组中的文本 “this is RAM” 也紧随其后显示了出来。这是为什么呢？这是一个已初始化数据的示例。RAM 数组是在源代码中定义的，但当 PIC18 器件最初上电时，并没有设置 RAM 的内容，其中存储的是随机值。为了在程序运行时初始化 RAM，会执行 MPLAB C18 初始化代码 （c018i.o 作为预编译的库包含在链接描述文件中），从而将程序存储器中的这一文本传送到 RAM。

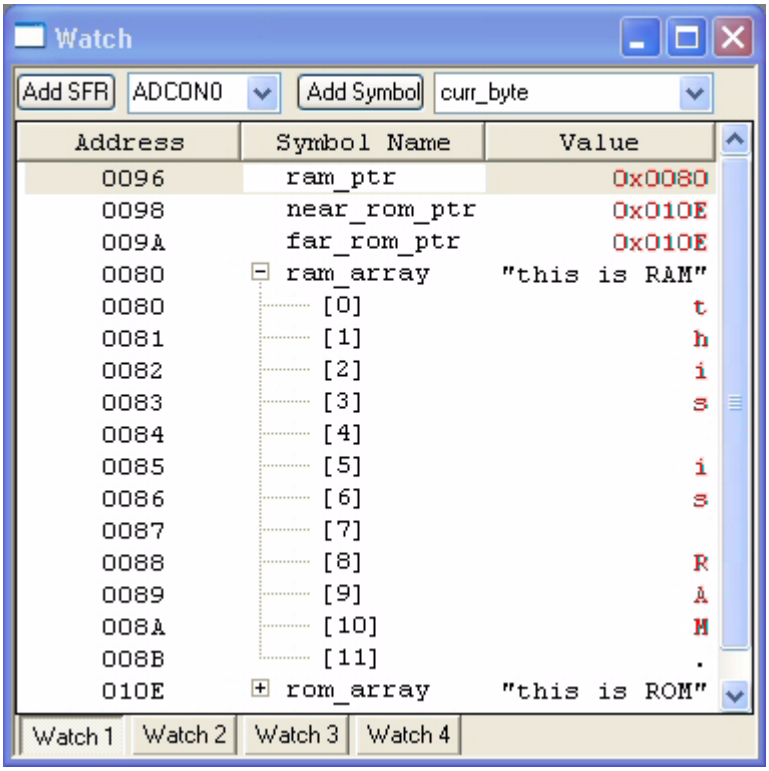
单击 **Run** 图标执行程序，然后单击 **Halt** 图标。Watch 窗口 （见图 5-41）现在将显示 3 个指针的值，并且文件寄存器 （RAM）区域包含 “this is RAM” 字符串。

图 5-41： 指针：程序运行后的 WATCH 窗口



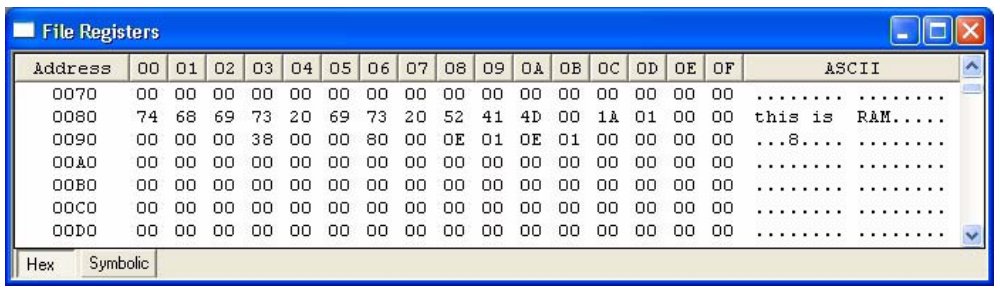
可展开数组，以显示其中每个元素的地址 （见图 5-42）：

图 5-42： 指针：数组展开后的 WATCH 窗口



选择 *View>File Registers* （视图 > 文件寄存器）并向下滚动到地址 0x0080 以查看 RAM 数组的内容 （见图 5-43）。

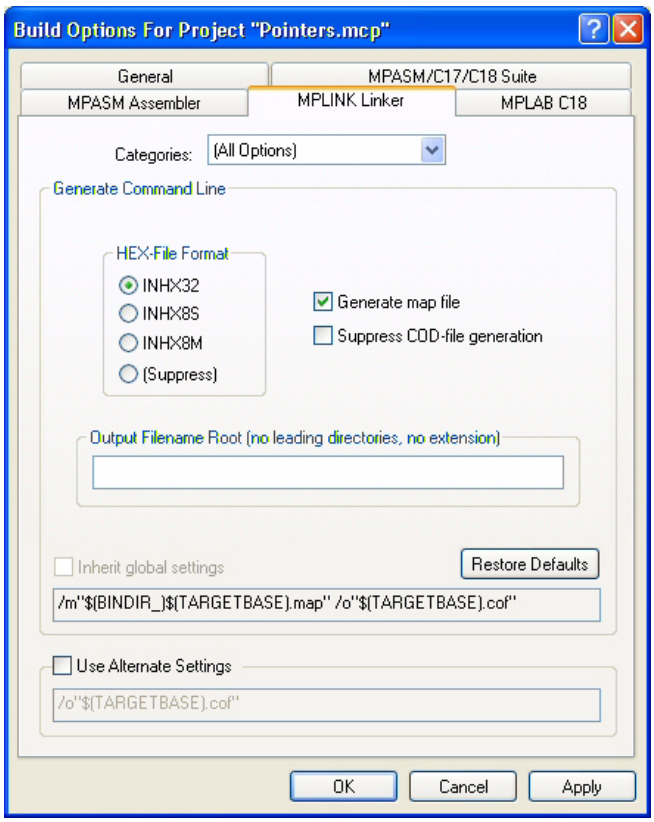
图 5-43： 指针：程序运行后的文件寄存器



5.4.5 映射文件

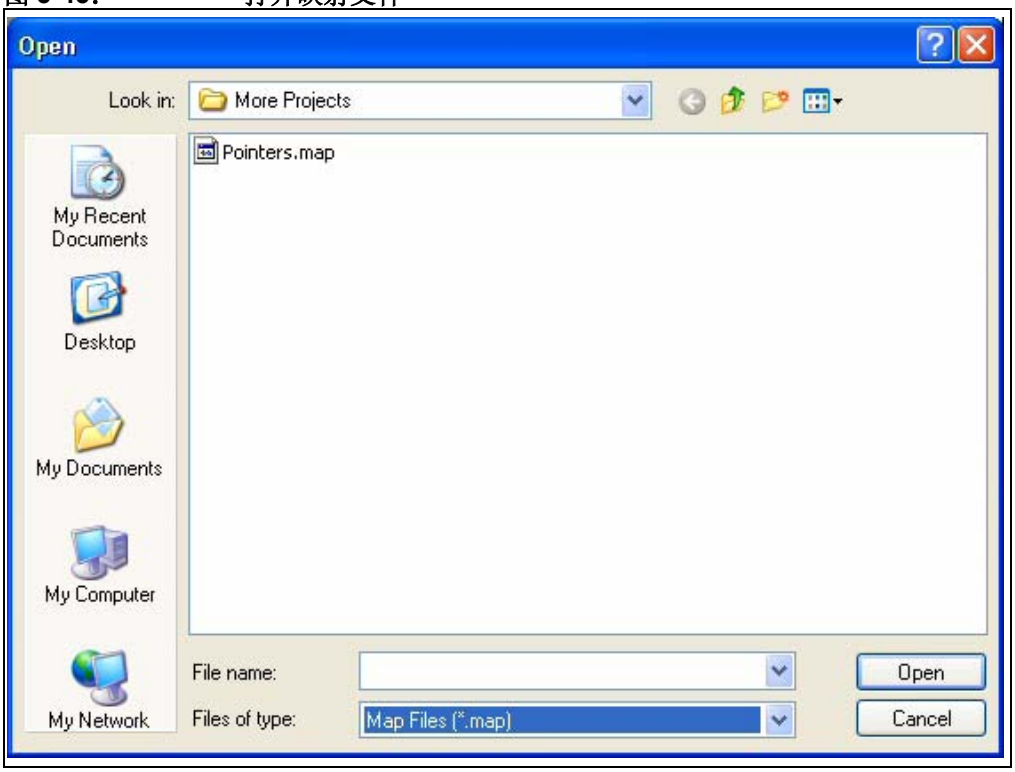
可由链接器生成映射文件，以提供一个文挡来定义由链接器确定的变量和代码的地址。要生成映射文件，请选择 *Project>Build Project>Project* 并选择 **MPLINK Linker**（MPLINK 链接器）选项卡 （见图 5-44）。选中标有 **Generate Map File** （生成映射文件）的复选框。

图 5-44： 生成映射文件



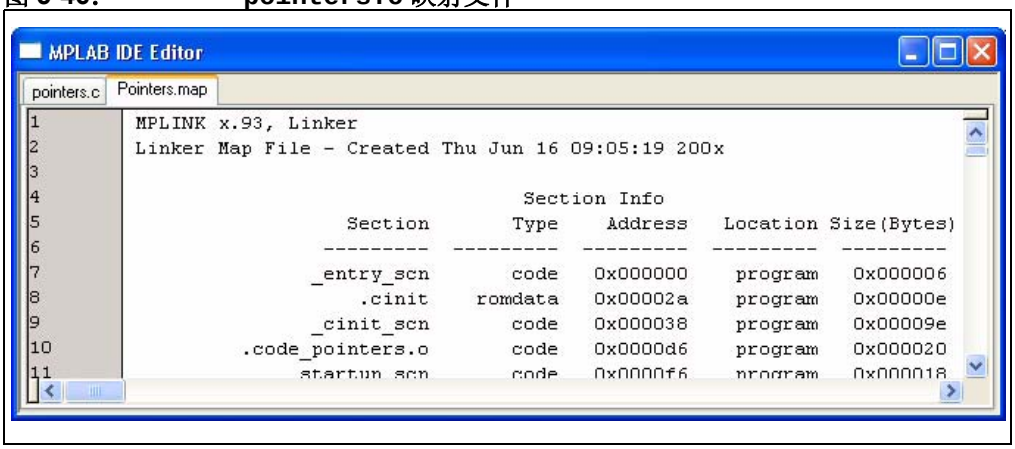
用 *Project>Build All* 重新编译项目，然后使用 *File>Open*，使用标有 **Files of Type**（文件类型）的底部下拉菜单将文件分类，使只显示 **Map Files (*.map)**（映射文件）。请参见图 5-45。应该已生成了一个名为 `pointers.map` 的映射文件。

图 5-45：　　打开映射文件



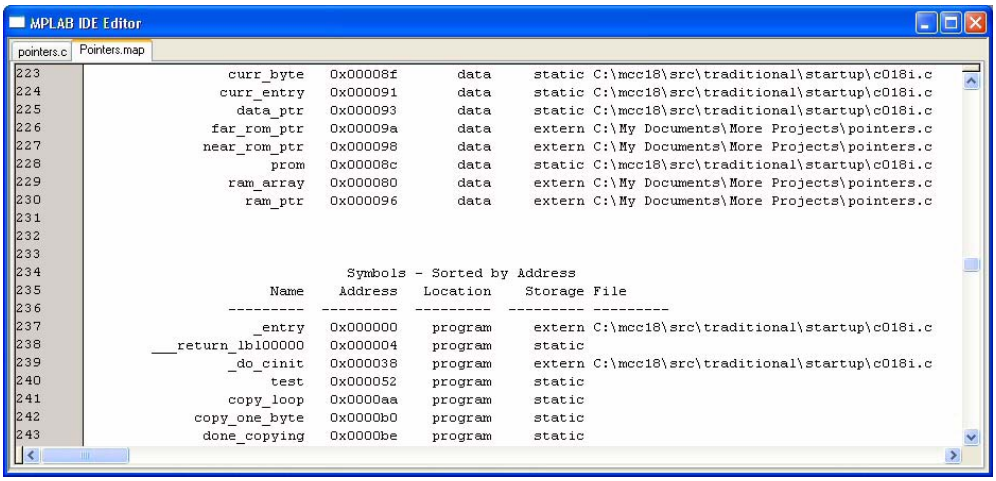
选中这个文件并单击 **Open**（打开）在 MPLAB 编辑器中查看文件。这个文件相当长，其顶部应该如图 5-46 所示：

图 5-46：　　`pointers.c` 映射文件



下向滚动文件，将会看到在 pointers.c 程序中定义的变量 （见图 5-47）。映射文件显示了这些变量在存储器中的地址以及定义它们的源文件。

图 5-47： pointers.c 中定义的变量



第 6 章 架构

6.1 简介

C 编译器的每个实现都必须支持目标处理器的特定功能。在 MPLAB C18 中，PIC18XXXX 的独特特性要求必须对存储器结构、中断、特殊功能寄存器以及单片机内核中与标准 C 语言无关的其他方面的细节加以考虑。本章提供了对 PIC18XXXX 某些方面的概述，数据手册中包含对这些方面的详尽叙述。

- PIC18XXXX 架构
- MPLAB C18 启动代码
- `#pragma` 伪指令
- 段
- SFR 和软件 / 硬件定时器
- 中断
- 数学函数库和 I/O 函数库

6.2 PIC18XXXX 架构

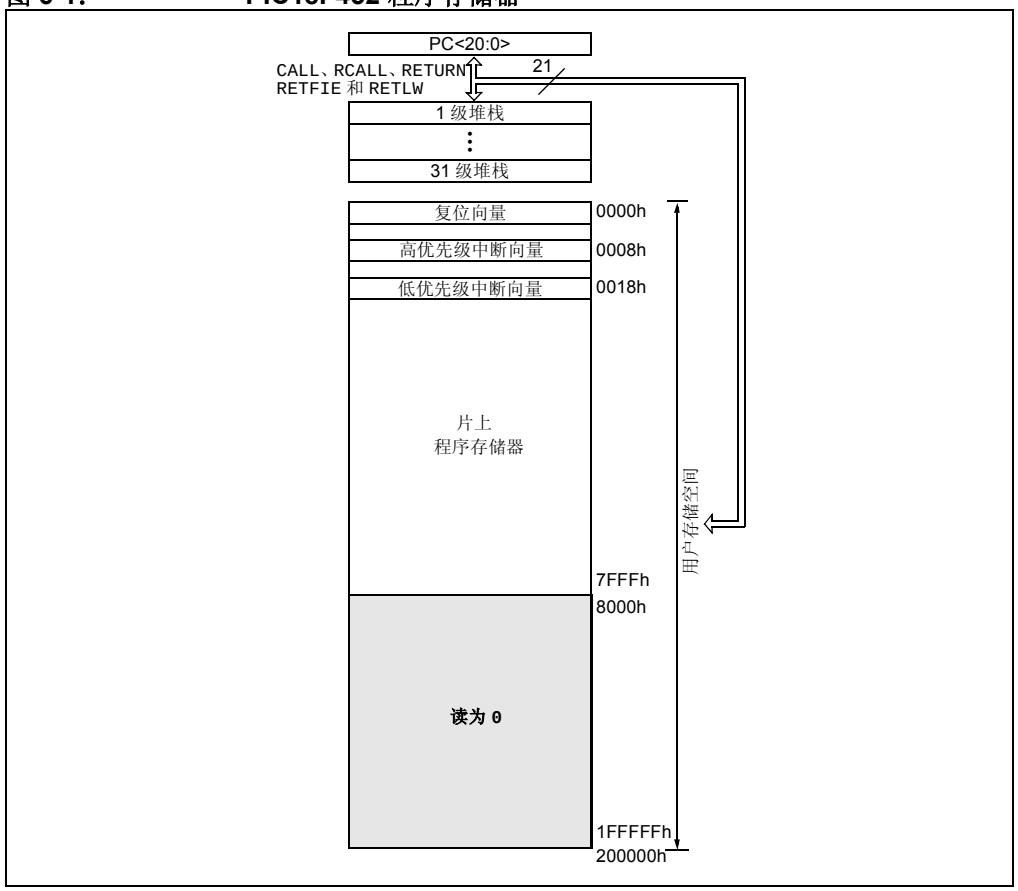
PIC18XXXX MCU 是 “哈佛架构”的单片机，即程序存储空间和数据存储空间是相互分离的。返回堆栈有其自身专用的存储区，并且如果特定器件具有片上数据 EEPROM 存储器的话，还会有一个非易失性存储空间。

6.2.1 程序存储器

使用 21 位的程序计数器来寻址程序存储空间，因此可允许 2 MB 的程序存储空间 （见图 6-1）。通常，PIC18XXXX MCU 拥有 16 KB 至 128 KB 范围的片上程序存储器。有些器件还允许扩展外部存储器。

复位时，程序计数器被设置为 0，取出第一条指令。中断向量位于地址为 0x000008 和 0x000018 的存储单元中，因此通常会在地址为 0 的存储单元中放置一条 GOTO 指令以使程序跳过中断向量。

图 6-1： PIC18F452 程序存储器



程序存储器包含 16 位字。大多数指令是 16 位的，但有些是双字的 32 位指令。不能按奇数字节边界访问指令。

PIC18F 器件带有闪存程序存储器，PIC18C 器件带有一次性可编程（One-Time-Programmable，OTP）存储器（或在某些情况下，为紫外线（UV）可擦除窗口器件）。通常，只有将固件烧写到器件中时才会写 OTP 存储器。而可以通过运行程序来擦除和重写闪存存储器。只需对器件进行少许连接，就可以对 OTP 和闪存器件编程了，从而能在器件被焊接到目标电路板上后对它们进行编程。

以下是 PIC18 架构的一些重要特性以及 MPLAB C18 与程序存储器有关的功能：

MPLAB C18 实现

请参见 《MPLAB® C18 C 编译器用户指南》了解更多关于这些特性的信息。

- 通常通过段属性 `code` 将指令存储到程序存储器中。
- 可通过段属性 `romdata` 和 `rom` 关键字将数据存储到程序存储器中。
- 可对 MPLAB C18 进行配置，为两种存储模型 （小存储模型和大存储模型）生成代码。在使用小存储模型时，指向程序存储器的指针使用 16 位。大存储模型时使用 24 位指针。

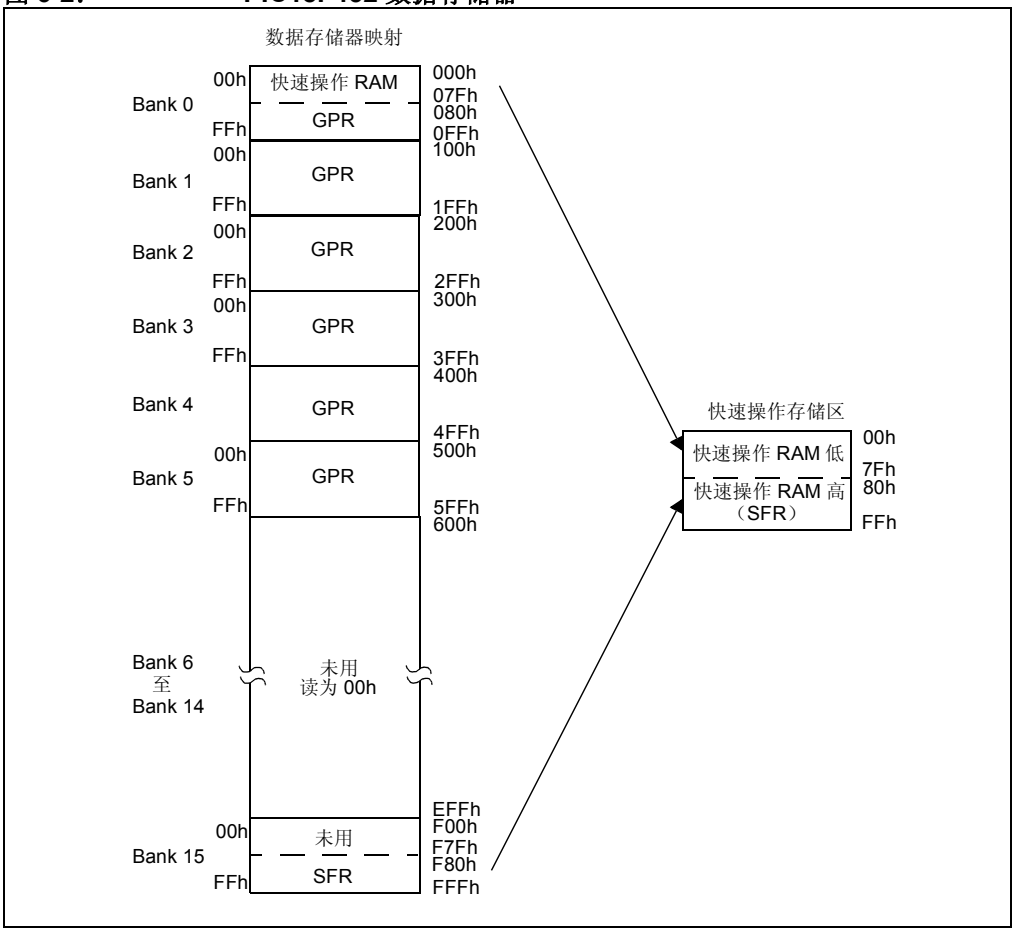
PIC18 架构

- `GOTO` 指令和 `CALL` 指令都是双字 （32 位）指令，可以跳转到程序存储器中的任何位置。
- 如果执行双字指令的第二个字 （使用 `GOTO` 指令转移或跳转到指令的中间），它将始终被执行为一条 `NOP` 指令。
- 所有指令都按偶数字节边界对齐。
- 在某些 PIC18XXXX 器件中，可以将整个程序存储器或程序存储器的部分空间代码保护。代码会正确执行但不能被读出或复制。
- 可使用表读指令读取程序存储器，并使用表写指令以特定的代码顺序写程序存储器。

6.2.2 数据存储器

数据存储器在 PIC18XXXX 系列中被称为 “文件寄存器”。它由一个多达 4096 个字节的 8 位 RAM 组成。在上电时，数据存储器中的值是随机的。数据由 256 字节的存储区为单位构成。在用存储区选择寄存器 （Bank Select Register， BSR）（寄存器地址的高 4 位）选定了某个存储区后， PIC18 指令才能只使用寄存器地址的低 8 位来读写文件寄存器。12 位指针允许在不指定存储区的情况下间接访问整个 RAM 空间。此外，可以直接访问 Bank 0 和 Bank 15 中的特殊存储区，而无需考虑分区和 BSR 寄存器的内容。这些特殊数据存储区被称为快速操作 RAM。大部分特殊功能寄存器位于快速操作 RAM 的高地址区域中 （见图 6-2）。以上说明只适用于非扩展模式。

图 6-2： PIC18F452 数据存储器



未初始化的数据存储器变量、数组和结构通常用段属性 `udata` 存储在数据存储器中。

可在 MPLAB C18 中定义已初始化数据，以确保在编译器执行初始化时变量的值正确。这意味着启动时，存储在程序存储器中的值会被传送到数据存储器中。根据应用程序需要多大初始化存储区，使用已初始化数据 （而不是在运行时设置数据值）可能会对程序存储器的使用效率产生不良影响。

因为文件寄存器是 8 位的，所以在使用变量时，应该考虑是将其定义为 `int` 还是 `char`。当预计变量的值不会超过255时，将其定义为`unsigned char`会产生更小更快的代码。

6.2.3 特殊功能寄存器

特殊功能寄存器 （SFR）包括 CPU 内核寄存器 （如堆栈指针、 STATUS 寄存器和程序计数器）和用于单片机外设模块的寄存器 （见图 6-3）。大多数 SFR 位于 Bank 15 中，并可以被直接访问而不需要使用 BSR，除非器件具有更多外设寄存器，超过了 Bank 15 的 128 字节存储区。那样的话，必须使用 BSR 来读写那些特殊功能寄存器。

注： 如果有超过 128 字节的特殊功能寄存器，某些 PIC18 器件会减少 Bank 0 中快速操作 RAM 的数量，并增加 Bank 15 中的快速操作 RAM 的数量。

图 6-3： PIC18F452 特殊功能寄存器

地址	名称	地址	名称	地址	名称	地址	名称
FFFh	TOSU	FDFh	INDF2(3)	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2(3)	FBEh	CCPR1L	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2(3)	FBDh	CCP1CON	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2(3)	FBCh	CCPR2H	F9Ch	—
FFBh	PCLATU	FDBh	PLUSW2(3)	FBBh	CCPR2L	F9Bh	—
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	—
FF9h	PCL	FD9h	FSR2L	FB9h	—	F99h	—
FF8h	TBLPTRU	FD8h	STATUS	FB8h	—	F98h	—
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	—	F97h	—
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	—	F96h	TRISE(2)
FF5h	TABLAT	FD5h	T0CON	FB5h	—	F95h	TRISD(2)
FF4h	PRODH	FD4h	—	FB4h	—	F94h	TRISC
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB
FF2h	INTCON	FD2h	LVDCON	FB2h	TMR3L	F92h	TRISA
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	—
FF0h	INTCON3	FD0h	RCON	FB0h	—	F90h	—
FEFh	INDF0(3)	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—
FEEh	POSTINC0(3)	FCEh	TMR1L	FAEh	RCREG	F8Eh	—
FEDh	POSTDEC0(3)	FCDh	T1CON	FADh	TXREG	F8Dh	LATE(2)
FECh	PREINC0(3)	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD(2)
FEBh	PLUSW0(3)	FCBh	PR2	FABh	RCSTA	F8Bh	LATC
FEAh	FSR0H	FCAh	T2CON	FAAh	—	F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	—
FE7h	INDF1(3)	FC7h	SSPSTAT	FA7h	EECON2	F87h	—
FE6h	POSTINC1(3)	FC6h	SSPCON1	FA6h	EECON1	F86h	—
FE5h	POSTDEC1(3)	FC5h	SSPCON2	FA5h	—	F85h	—
FE4h	PREINC1(3)	FC4h	ADRESH	FA4h	—	F84h	PORTE(2)
FE3h	PLUSW1(3)	FC3h	ADRESL	FA3h	—	F83h	PORTD(2)
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h	—	FA0h	PIE2	F80h	PORTA

注 **1：** 未用的寄存器，读为 0。
2： 该寄存器在 PIC18F2X2 器件上不存在。
3： 这不是一个物理寄存器。

注： 在使用 MPLAB C18 时，分区通常是透明的，但可使用 #pragma varlocate 伪指令告知编译器变量存储的位置，从而产生更高效的代码。

6.2.4 返回地址堆栈

CALL 和 RETURN 指令将程序计数器的值压入和弹出返回地址堆栈。返回堆栈是存储器中一块独立的区域，允许 31 级子程序嵌套。

注： CALL/RETURN 堆栈与由 MPLAB C18 维护的软件堆栈不同。软件堆栈用于存储自动参数和局部变量，并按照链接描述文件中的定义将它们分配到文件寄存器存储区中。

6.2.5 EEPROM 数据存储器

数据 EEPROM 是独立的非易失性存储器。它可用来在芯片断电时存储数据。它可通过 4 个特殊功能寄存器进行访问，对其进行写操作需要特殊的写顺序。在许多 PIC18XXXX 器件中，该存储区也可被保护，使得数据不能被读出或复制。请参见《MPLAB® C18 C 编译器用户指南》获取读写数据 EEPROM 存储器的代码示例。

6.2.6 配置存储区

配置位控制 PIC18XXXX 器件的各种模式，包括振荡器类型、看门狗定时器、代码保护和其他特性。该存储区超过了程序存储器的 21 位地址范围，但可使用表读和表写指令进行访问。大多数配置位已在编程器件时（使用 MPLAB PM3、 PICSTART Plus、MPLAB ICD 2 或其他编程硬件）被设置为了所需的状态。最初编程时，可通过 MPLAB C18 `#pragma config` 伪指令将这些位置 1，通常应用程序不需要访问存储器的这一区域。

6.2.7 扩展模式

某些 PIC18XXXX 器件有一种为提高可重入代码效率而设计的备用工作模式。当这些器件被编程为使用扩展模式时，寻址快速操作 RAM 的方式会受到影响，一些指令的执行会有所不同，并且可使用某些新的指令和寻址模式。此外，使用扩展模式的应用程序还需要随 MPLAB C18 安装、名称以 `_e` 结尾的特殊链接描述文件。

请参见相关的 PIC18XXXX 数据手册了解该模式的有关信息，尤其是在应用程序中使用了汇编语言代码的情况。

6.3 MPLAB C18 启动代码

预编译代码块必须链接到每个 MPLAB C18 程序中，来初始化寄存器并为编译器设置数据堆栈。该代码在应用程序启动时执行，然后跳转到应用程序中的 `main()`。有不同的启动代码组可供选择，这取决于是否要求在启动时将变量初始化为 0 以及扩展模式是否使能。请参见 《MPLAB® C18 C 编译器用户指南》了解有关启动代码的信息。

6.4 #pragma 伪指令

按照目标处理器架构的要求，ANSI C 标准为每个 C 实现提供了定义独特语法结构的方法。这是通过使用 `#pragma` 伪指令完成的。MPLAB C18 编译器中最常见的 `#pragma` 伪指令可标识 PIC18XXXX 中使用的存储器段。例如，

```
#pragma code
```

告知 MPLAB 18 将该伪指令后的 C 语言代码编译到程序存储器的 “`code`”段中。对于每个 PIC18XXXX 器件 ，在每个 PIC18XXXX 器件的相应链接描述文件中定义 `code` 段，指定执行指令的程序存储区。可以如给出的示例那样插入该伪指令，也可以在该伪指令后直接跟目标处理器 `code` 区中的某个地址，从而可以完全控制代码在存储器中的位置。这通常无关紧要，但在某些应用程序中，如引导加载程序，对应用程序中某些代码块的执行位置进行严格控制就非常重要。

将代码从另一种编译器移植到 MPLAB C18 时，原编译器的 #pragma 伪指令的操作必须被识别并转换成 MPLAB C18 的类似伪指令。无法被 MPLAB C18 识别的 #pragma 伪指令将被忽略，从而允许将代码从一种架构移植到另一种架构，并且不会出现编译错误。理解原编译器中 #pragma 伪指令的功能以及新的目标架构是工程师的职责所在，以在不同的单片机之间有效移植代码。

在为变量分配存储空间时，另一种最常见的 #pragma 伪指令是

```
#pragma udata
```

在此声明之后定义的未初始化变量将使用通用寄存器进行存储。

这与为变量和指令位于相同存储空间内的器件编写 C 程序不同。在 PIC18XXXX 上，程序存储器和文件寄存器存储器有很大不同，因此，必须明确地区分数据存储器和程序存储器中的存储区。

MPLAB C18 中的 #pragma 伪指令如表 6-1 所示：

表 6-1： MPLAB® C18 #pragma 伪指令

伪指令	用途
code	程序存储器指令。将所有后续指令编译到目标 PIC18XXXX 的程序存储器段中。
romdata	存储在程序存储器中的数据。将后续静态数据编译到目标 PIC18XXXX 的程序存储器段中。
udata	未初始化的数据。使用 PIC18XXXX 的文件寄存器 （数据）空间存储将要在后面的源代码中使用的未初始化的静态变量。这些存储单元的值未被初始化。更多信息，请参见 《MPLAB® C18 C 编译器用户指南》中的 “启动代码”章节。
idata	已初始化数据。使用 PIC18XXXX 的文件寄存器 （数据）空间存储将要在后面的源代码中使用的未初始化的变量。 但是，与 udata 不同，这些单元将被设置为在源代码中定义的值。注意，这意味着这些值将被放在程序存储器中的某个位置，然后在应用程序开始执行前被编译器初始化代码传送到文件寄存器中。
config	定义 PIC18XXXX 配置位的状态。这些状态会生成到由链接器输出的 .HEX 文件中，并将与应用固件一同被编程到器件中。
interrupt	将指定 C 函数中的代码编译为高优先级中断服务程序。请参见 《MPLAB® C18 C 编译器用户指南》中的 “中断服务程序”章节。
interruptlow	将指定 C 函数中的代码编译为低优先级中断服务程序。请参见 《MPLAB® C18 C 编译器用户指南》中的 “中断服务程序”章节。
varlocate	指定变量的存放位置，使得编译器不会生成在访问变量时设置存储区的额外指令。请参见 《MPLAB® C18 C 编译器用户指南》中的 “#pragma varlocate”章节。

关于这些 #pragma 伪指令及其他伪指令的完整信息，请参见 《MPLAB® C18 C 编译器用户指南》。

6.5 段

如上所述，段是 PIC18XXXX 存储器中的各种区域，包括程序存储器、文件寄存器（数据）存储器、 EEDATA 非易失性存储器和数据堆栈存储区以及其他一些存储区。

通常，程序存储器和数据存储器需要段。随着设计日益复杂，还可能需要其他类型的段。

段是在链接描述文件中定义的。以下是 PIC18F452 的链接描述文件：

例 6-1： PIC18F452 的链接描述文件示例

```
// Sample linker script for the PIC18F452 processor

LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f452.lib

CODEPAGE   NAME=vectors    START=0x0            END=0x29           PROTECTED
CODEPAGE   NAME=page       START=0x2A           END=0x7FFF
CODEPAGE   NAME=idlocs     START=0x200000       END=0x200007       PROTECTED
CODEPAGE   NAME=config     START=0x300000       END=0x30000D       PROTECTED
CODEPAGE   NAME=devid      START=0x3FFFFE       END=0x3FFFFF       PROTECTED
CODEPAGE   NAME=eedata     START=0xF00000       END=0xF000FF       PROTECTED

ACCESSBANK NAME=accessram  START=0x0            END=0x7F
DATABANK   NAME=gpr0       START=0x80           END=0xFF
DATABANK   NAME=gpr1       START=0x100          END=0x1FF
DATABANK   NAME=gpr2       START=0x200          END=0x2FF
DATABANK   NAME=gpr3       START=0x300          END=0x3FF
DATABANK   NAME=gpr4       START=0x400          END=0x4FF
DATABANK   NAME=gpr5       START=0x500          END=0x5FF
ACCESSBANK NAME=accesssfr  START=0xF80          END=0xFFF          PROTECTED

SECTION    NAME=CONFIG     ROM=config

STACK SIZE=0x100 RAM=gpr5
```

该链接描述文件定义名为 page 的主程序存储区，地址从 0x002A 延伸到 0x7FFF。当遇到 #pragma code 伪指令时，编译器将生成存放到该区域的机器码指令。

为 PIC18F452 的 6 个文件寄存器存储区 （gpr = 通用寄存器存储区）定义了数据存储器段。由于在 PIC18XXXX 上存储器是分区的，这 6 个区域被各自定义为独立的段。在遇到 #pragma udata 和 #pragma idata 时，编译器将在这些文件寄存器存储区中保留区域，来存储随后定义的变量。

accessram 和 accesssfr 段用于定义数据存储器中的快速操作 RAM 区域。

注意，某些区域被标记为 “PROTECTED”。这意味着链接器不会将代码或数据放入那些区域，除非特别指定。要将代码或数据放入受保护的区域，请使用如下所示的 #pragma 伪指令：

```
#pragma code page
```

这将导致其后的指令被编译到 page 段中，通常是在链接描述文件中定义的主程序存储区。

6.6 SFR 和软件 / 硬件定时器

PIC18XXXX 特殊功能寄存器 （SFR）是单片机文件寄存器区域中的特殊寄存器。包括控制单片机内核的寄存器，如堆栈指针、 STATUS 寄存器和程序计数器，以及控制各种外设的寄存器。外设包括输入和输出引脚、定时器、USART 和读写器件 EEDATA 区域的寄存器等。MPLAB C18 可以通过名称访问这些寄存器，并且可像读写应用程序中定义的变量那样对它们进行读写。但是使用时仍要小心，因为一些特殊功能寄存器的特性与变量有所不同。有些特殊功能寄存器只有某些位可用，有些是只读的，还有些可能会在被访问时影响其他寄存器或器件的操作。

6.6.1 I/O 寄存器

通过读写与器件上的端口引脚相关的寄存器来完成 PIC18XXXX 引脚的输入和输出。请查看数据手册了解器件的可用端口。每个端口都有 3 个与之相关的特殊功能寄存器。其中一个称为 TRIS 寄存器，它定义端口引脚的方向：输入或输出。第二个称为 PORT 寄存器，用来读写端口引脚的值；第三个称为 LAT 寄存器，它是一个锁存器，可允许读写端口上的值，而实际上并不读端口上引脚的当前状态。这在 PIC18XXXX 架构中很重要，因为需要考虑读 - 修改 - 写操作。存储 I/O 端口寄存器的内容不能像存储变量一样处理——两者的操作有很大差异。欲知更多信息请参见数据手册。

有些引脚是复用的，在将它们用作数字 I/O 之前可能需要通过其他特殊功能寄存器进行配置。特别是很多 PIC18XXXX 器件上的 PORTA 也能用作 A/D 转换器的模拟输入。

要将 PORTB 配置为 4 个输入引脚和 4 个输出引脚，在 MPLAB C 18 中可编写下列代码：

```
TRISB = 0xF0 /* configure PORTB as 4 input pins, bits 4-7
             and 4 output pins, bits 0-3 */
PORTB = 0x0C /* set pins 0 and 1 low, pins 2 and 3 high */
```

6.6.2 硬件定时器

PIC18XXXX 定时器也是通过特殊功能寄存器配置并访问的。大多数 PIC18XXXX 器件至少有 3 个定时器。例如，PIC18F452 中的 Timer0 通过 T0CON 寄存器进行配置，并且可使用两个 8 位寄存器 TMR0L 和 TMR0H 读取和写入其计数器 / 定时器值。INTCON 寄存器中的某些位可用来将 Timer0 设置为中断源，并且还控制 Timer0 在用作计数器时是依靠振荡器的输出还是外部信号计数。

6.6.3 软件定时器

正如在任何 C 程序中那样，可以用软件创建延时和定时循环。设计考虑将影响到如何采用软件定时器和硬件定时器。典型的软件延时循环包括创建一个计数变量，并将其递减直至为零。软件定时器的缺点在于，如果发生中断，软件定时器的延时将被延长，并可能变得无法预测。此外，在处理软件延时循环时，程序除了能对中断作出响应外，什么也不能做。

6.7 中断

中断是 PIC18XXXX 内核的一个功能。定时器、I/O 引脚、USART、A/D 和其他外设均能导致中断。当发生中断时，应用程序中的代码被暂停，而执行中断服务程序中的代码。 当中断服务程序结束时，会执行一条 “从中断返回”指令，程序将从停止的地方恢复执行。

在 PIC18XXXX 中有低优先级中断和高优先级中断两种。决定使用哪种中断是设计应用时需要考虑的事项。在 MPLAB C18 中可使用这两种中断类型，但设计人员必须注意特定中断操作的细节以保护一些关键内部寄存器的内容。最重要的是要仔细考虑变量的使用和库 （尤其是在中断服务程序中使用的变量和库）。

当中断发生时，低优先级中断只保存 PC （程序计数器寄存器）。对于高优先级中断，PIC18XXXX 内核将自动保存 PC、 WREG、 BSR 和 STATUS 寄存器。请参见《MPLAB® C18 C 编译器用户指南》中的 “ISR 现场保护”章节，了解关于在中断过程中保存应用程序变量的信息。

6.8 数学函数库和 I/O 函数库

MPLAB C18 有用于外设控制、外设软件实现、通用数据处理和数学函数的库。请参见《MPLAB® C18 C 编译器函数库》（DS51297F_CN）了解对这些库的完整描述。

为这些库提供了源代码，因此可根据应用的需求修改这些库，从而对它们进行定制并重建。

对外设库进行使用通常要求理解器件数据手册中所描述的外设操作。使用 C 函数库可以简化外设的初始化和使用。

MPLAB C18 数学函数库包括浮点运算、三角运算和其他运算。在 8 位嵌入式控制器上使用浮点和复数数学函数时，应该仔细权衡，评估这些运算对于特定设计是否是高效率的选择。通常，表、插值表或使用其他方法的近似算法会为任务提供足够的精度。32 位浮点运算执行起来通常需要几百个周期，可能会占用非常大的程序存储空间。

第 7 章 疑难解答

7.1 简介

本章涉及在初学 MPLAB C18 时可能遇到的常见错误消息。本章还回答了许多常见问题（Frequently Asked Question， FAQ）。

错误消息

- EM-1 链接器错误：“name exceeds file format maximum of 62 characters”（文件名超过了文件格式要求的最多 62 个字符的限制）
- EM-2 链接器错误：“could not find file ‘c018i.o’”（找不到文件 c018i.o）
- EM-3 编译器错误 “Error [1027] unable to locate ‘p18cxxx.h’”（错误 [1027] 不能定位 p18cxxx.h）
- EM-4 编译器错误：“Error [1105]symbol ‘symbol-name’ has not been defined.”（错误 [1105] 符号 “符号名”未定义。）
- EM-5 MPLAB IDE 错误：“Skipping link step. The project contains no linker script.”（跳过链接步骤。项目未包含链接描述文件。）
- EM-6 编译器错误：Syntax Error （语法错误）
- EM-7 链接器错误：“Could not find definition of symbol...”（找不到符号 ... 的定义）

常见问题 （FAQ）

- FAQ-1 是否安装了使用 MPLAB C18 所需的 MPLAB IDE 组件？
- FAQ-2 需要设置什么才能在 Output 窗口中显示 printf() 语句？
- FAQ-3 如何在某处声明一个全局结构 / 联合，这样就不需要在所有引用该变量的 .c 文件中添加 “extern”声明语句了？
- FAQ-4 为什么会出现 “Warning [2066] type qualifier mismatch in assignment”（警告 [2066] 指定的类型限定符不匹配）？
- FAQ-5 当我取字符串指针的内容时，结果不是字符串的第一个字符，为什么？
- FAQ-6 使用低优先级中断的代码示例在哪里？
- FAQ-7 可以使用 16 位变量访问 16 位定时器的特殊功能寄存器 （如 TMR1L 和 TMR1H）吗？
- FAQ-8 我如何修复数据存储器段的 “unable to fit section”（不能分配到段）错误？
- FAQ-9 我如何修复程序存储器段的 “unable to fit section”（不能分配到段）错误？
- FAQ-10 我如何在数据存储器中创建一个大对象 （> 256 字节）？
- FAQ-11 我如何将数据表存入程序存储器？
- FAQ-12 我如何将数据从程序存储器复制到数据存储器？
- FAQ-13 我如何在 C 程序中设置配置位？
- FAQ-14 有哪些有关在 C 程序中设置配置位的参考资料？
- FAQ-15 我如何使用 printf 输出字符串常量？
- FAQ-16 当我对两个字符执行算术运算，并将运算结果赋值给一个整型变量，但我没有得到期望的值。为什么？

7.2 错误消息

EM-1 链接器错误：“name exceeds file format maximum of 62 characters”（文件名超过了文件格式要求的最多 62 个字符的限制）

选择 *Project>Build Options…>Project* 的 **MPLINK** 选项卡，选中复选框 **Suppress Cod-file generation**（禁止生成 Cod 文件）。 COD 文件是较早的格式，现在已不再使用。

EM-2 链接器错误：“could not find file ‘c018i.o’”（找不到文件 c018i.o）

在 *Project>Build Options…>Project* 的 **General** 选项卡中输入正确的目录路径。将 **Library Path**（库路径）文本框设置为 “C:\mcc18\lib”。 c018i.o 是 MPLAB C18 的启动库。它设置堆栈，初始化变量，然后跳转到应用程序的 main() 函数。

EM-3 编译器错误 “Error [1027] unable to locate ‘p18cxxx.h’”（错误 [1027] 不能定位 p18cxxx.h）

在 *Project>Build Options&…>Project* 的 **General** 选项卡上输入正确的目录路径。将 **Include Path**（包含路径）文本框设置为 “C:\mcc18\h”。 p18cxxx.h 是一个通用头文件，包含特定于选定处理器的头文件。

EM-4 编译器错误：“Error [1105]symbol ‘*symbol-name*’ has not been defined.”（错误 [1105] 符号 “*符号名*” 未定义。）

如果 *符号名* 是一个特殊功能寄存器 （如 TRISB），请确认已包含了通用处理器头文件（#include <p18cxxx.h>）或特定于处理器的包含文件 （如 #include <p18f452.h>）。特殊功能寄存器在特定于处理器的头文件中声明。还要确认特殊功能寄存器全部为大写字母 （即 TRISB 而非 trisb），因为 C 语言是区分大小写的，特殊功能寄存器全用大写字母声明。

如果 *符号名* 不是特殊功能寄存器，请确认在使用前已定义符号，并且符号名拼写正确。

EM-5 MPLAB IDE 错误：“Skipping link step. The project contains no linker script.”（跳过链接步骤。项目未包含链接描述文件。）

请确认项目包含一个链接描述文件。链接描述文件在 MPLAB C18 安装目录的 lkr 子目录中。

EM-6 编译器错误：Syntax Error （语法错误）

这通常是源代码中的拼写错误。双击 Output 窗口中的错误行，就能转到 MPLAB 编辑器窗口，并且光标停留在该窗口中引起错误的行上。通常采用颜色区分代码的语法会显示错误。

EM-7 链接器错误：“Could not find definition of symbol...”（找不到符号 ... 的定义）

这可能是由于使用了错误的链接描述文件导致的。MPLAB C18 的链接描述文件包含其他库文件。请确认使用的是 MPLAB C18 安装目录 lkr 子目录中的链接描述文件。

项目编译通过，但在链接器尝试链接时，显示以下错误：

```
Error - could not find definition of symbol 'putsMYFILE' in file
'C:\My Projects\myfile.o'.
Errors : 1
```

可能是 C 文件同汇编文件同名，尽管扩展名不同。仔细查看 Output 窗口以判断是否链接器尝试生成两个文件名相同的 .o 文件。这样很有可能会忽略项目中同名的第一个文件。重命名文件，这样文件就不会同名了。

7.3 常见问题（FAQ）

FAQ-1 是否安装了使用 MPLAB C18 所需的 MPLAB IDE 组件？

下面是检查安装的组件的方法：

转到 Windows 的**开始**菜单，浏览到 **Microchip** 文件夹，选择 *MPLAB>Set Up MPLAB Tools*，验证是否安装了正确的组件。请参见图 1-1，了解 MPLAB C18 所需的最小 IDE 安装选择。

FAQ-2 需要设置什么才能在 Output 窗口中显示 `printf()` 语句？

在 MPLAB IDE 中，选择 *Debugger>Select Tool>MPLAB SIM*，启用软件模拟器，并访问 debugger 菜单。然后选择 *Debugger>Settings*，单击 **Uart1 I/O** 选项卡。请确认选中了 “**Enable Uart1 I/O**”，在 Output 框中选中 Window 选项 （见图 3-15）。

FAQ-3 如何在某处声明一个全局结构 / 联合，这样就不需要在所有引用该变量的 `.c` 文件中添加 “`extern`”声明语句了？

在头文件中创建一个 *typedef*：

```
typedef union {
  struct {
    unsigned char Outstanding_Comms_Req:1;
  };
  unsigned char All_Flags;
} RS485_t;
```

然后在一个 `.c` 文件中使用：

```
RS485_t RS485_Flags;
```

要定义联合，可以在其他 `.c` 文件中使用：

```
extern RS485_t RS485_Flags;
```

还可以将 `extern` 写入一个头文件中，然后将该头文件包含在 `.c` 文件中。

FAQ-4 为什么会出现 “Warning [2066] type qualifier mismatch in assignment”（警告 [2066] 指定的类型限定符不匹配）？

MPLAB C18 提供的库是使用大代码模型编译 （`-ml` 命令行选项）的。默认情况下，MPLAB IDE 和编译器将使用小代码模型编译应用程序。例如，随编译器提供的 `printf` 函数期望收到 “`const far rom char *`”，但没有为应用程序选择大代码模型时，应用程序实际发送 “`const near rom char *`” 到 `printf` 函数。正是 `far` 和 `near` 间的差别引起了“type qualifier mismatch in assignment”警告。要消除这些警告，应采取以下三种措施中的一种：

1. 使用小代码模型重新编译随 MPLAB C18 提供的库 （仅在所有应用程序均使用小代码模型时推荐）；
2. 在 IDE 中为特定应用程序启用大代码模型 （可能会增加代码尺寸）；或
3. 将常量字符强制转换为常量 `far rom` 字符串指针，如：

```
printf ((const far rom char *)"This is a test\n\r");
```

FAQ-5 当我取字符串指针的内容时，结果不是字符串的第一个字符，为什么？

```
const char *path = "file.txt";
while(*path)   // while end of string not found
{
    path++;
    length++;
}
```

MPLAB C18 将常量字符串存储在程序存储器中。但 path 是一个指向数据存储器的指针。当取 path 指针的内容时，访问的是数据存储器，而非程序存储器。解决方法是添加 rom 关键字，使指针指向 ROM 单元，而非 RAM。

```
const rom char *path = "file.txt";
```

FAQ-6 使用低优先级中断的代码示例在哪里？

请参见 《MPLAB® C18 C 编译器用户指南》中的 #pragma interruptlow 和 “示例”一章。

FAQ-7 可以使用 16 位变量访问 16 位定时器的特殊功能寄存器 （如 TMR1L 和 TMR1H）吗？

请不要将 TMR1H 和 TMR1L 组合成一个 16 位变量进行访问。两个特殊功能寄存器的读写顺序非常关键，因为只有读 / 写完 TMR1L 寄存器才表示读 / 写完了整个 16 位定时器。如果编译器正好在写 TMR1H 前写了 TMR1L，那么就不会将写入 TMR1H 的数据装入定时器的高字节。同样，编译器先读哪个字节也是不可控制的。

FAQ-8 我如何修复数据存储器段的 “unable to fit section”（不能分配到段）错误？

MPLAB C18 提供了两种不同的数据存储器段类型：

- udata——包含静态分配的未初始化用户变量
- idata——包含静态分配的已初始化用户变量

在 MPLAB C18 中每种段类型都有默认的段 （如 .udata_foobar.o）。

例如，给出以下位于 foobar.c 中的源代码：

```
unsigned char foo[255];
int bar;
void main (void)
{
   while (1)
       ;
}
```

该代码会导致下列错误：

```
Error - section  '.udata_foobar.o' can not fit the section.
Section  '.udata_foobar.o' length = 0x00000101.
```

有两种方法可解决这个错误：

1. 将 foobar.c 分成多个文件：

```
foo.c
   unsigned char foo[255];
   void main (void)
   {
     while (1)
      ;
   }

bar.c
   int bar;
```

2. 使用 #pragma udata 伪指令创建一个独立的段，包含 foo 和 bar 变量：

```
foobar.c
   #pragma udata foo
   unsigned char foo[255];
   #pragma udata bar
   int bar;
   void main (void)
   {
     while (1)
       ;
   }
```

FAQ-9 我如何修复程序存储器段的 “unable to fit section”（不能分配到段）错误？

MPLAB C18 提供了两种不同的程序存储器段类型：

- code——包含可执行指令
- romdata——包含变量和常量

默认情况下， MPLAB IDE 仅启用不影响调试的优化。要减少代码段所用的程序存储器空间，请启用所有的优化。要在 MPLAB IDE 中启用优化，请选择 *Project>Build options…>Project*，单击 **MPLAB C18** 选项卡，并设置 **Categories: Optimization** 启用所有优化。

FAQ-10 我如何在数据存储器中创建一个大对象 （> 256 字节）？

默认情况下， MPLAB C18 假设对象不超过存储区边界。要安全使用大于 256 字节的对象，需要执行下列步骤：

1. 必须使用 #pragma idata 或 #pragma udata 伪指令将对象分配到恰当的段中：

```
#pragma udata buffer_scn
static char buffer[0x180];
#pragma udata
```

2. 必须通过指针访问对象：

```
char * buf_ptr = &buffer[0];
...
// examples of use
buf_ptr[5] = 10;
if (buf_ptr[275] > 127)
```

...

3. 必须在链接描述文件中创建一个跨越多个存储区的区域：

- 修改前的链接描述文件：

```
DATABANK NAME=gpr2 START=0x200 END=0x2FF
DATABANK NAME=gpr3 START=0x300 END=0x3FF
```

- 修改后的链接描述文件：

```
DATABANK NAME=big START=0x200 END=0x37F PROTECTED
DATABANK NAME=gpr3 START=0x380 END=0x3FF
```

4. 必须通过在链接描述文件中添加 SECTION 伪指令，将对象所在的段 （在第 1 步中创建）分配到新的区域 （在第 3 步中创建）：

```
SECTION NAME=buffer_scn RAM=big
```

FAQ-11 我如何将数据表存入程序存储器？

默认情况下，MPLAB C18 将用户变量存入数据存储器。rom 限定符用于指示将对象分配到程序存储器中：

```
rom int array_of_ints_in_rom[] =
{ 0, 1, 2, 3, 4, 5 };
rom int * q = &array_of_ints_in_rom[0];
```

在上面的示例中， array_of_ints_in_rom 表示程序存储器中的整型数组。q 是一个指针，可用于遍历数组中的元素。

FAQ-12 我如何将数据从程序存储器复制到数据存储器？

对于指针类型，使用下列某个标准库函数：

函数	说明
memcpypgm2ram	将 ROM 中的一段缓冲区复制到 RAM
memmovepgm2ram	将 ROM 中的一段缓冲区复制到 RAM
strcatpgm2ram	将 ROM 中的源字符串复制添加到 RAM 中目标字符串尾
strcpypgm2ram	将 RAM 中的字符串复制到 ROM
strncatpgm2ram	将 ROM 中源字符串中指定数量的字符添加到 RAM 中目标字符串尾
strncpypgm2ram	将 ROM 中源字符串中的字符复制到 RAM 中的目标字符串中

对于非指针类型，可以直接分配。

示例：

```
rom int rom_int = 0x1234;
ram int ram_int;
rom char * rom_ptr = “Hello, world!”;
ram char ram_buffer[14];
void main(void)
{
  ram_int = rom_int;
  strcpypgm2ram (ram_buffer, rom_ptr);
}
```

FAQ-13 我如何在 C 程序中设置配置位？

MPLAB C18 提供了 `#pragma config` 伪指令，可以在 C 程序中设置配置位。

使用示例：

```
/* Oscillator Selection: HS */
#pragma config OSC = HS
/* Enable watchdog timer and set postscaler to 1:128 */
#pragma config WDT = ON, WDTPS=128
```

FAQ-14 有哪些有关在 C 程序中设置配置位的参考资料？

- 《MPLAB® C18 C 编译器用户指南》包含对 `#pragma config` 伪指令的一般说明。
- *“PIC18 Configuration Settings Addendum”* 包含所有 PIC18 器件的可用配置设置和值。
- MPLAB C18 `--help-config` 命令行选项列出了特定器件具有的配置设置和标准输出值。

FAQ-15 我如何使用 printf 输出字符串常量？

因为字符串常量存储在程序存储器中，因而需要添加特定于 MPLAB C18 的转换运算符（%S）来处理程序存储器数组（rom char []）中字符的输出：

```
#include <stdio.h>
rom char * foo = “Hello, world!”;
void main (void)
{
  printf (“%S\n”, foo);
  printf (“%S\n”, “Hello, world!”);
}
```

当输出一个 far 程序存储器数组（far rom char []）时，请确认使用 H 大小指定符（即 %HS）：

```
#include <stdio.h>
far rom char * foo = “Hello, world!”;
void main (void)
{
  printf (“%HS\n”, foo);
}
```

FAQ-16 当我对两个字符执行算术运算，并将运算结果赋值给一个整型变量，但我没有得到期望的值。为什么？

给定以下示例：

```
unsigned char a, b;
unsigned int i;
a = b = 0x80;
i = a + b;
```

ANSI/ISO 期望 i 的值是 0x100，但 MPLAB C18 将 i 设置为 0x00。

默认情况下，MPLAB C18 以最大操作数的长度执行算术运算，即使两个操作数都小于 int 长度。要启用 ISO 要求的运算方式，即所有算术运算采用的精度均大于或等于 int，使用 -Oi 命令行选项。要在 MPLAB IDE 中启用该功能，请选择 *Project>Build options…>Project*，单击 **MPLAB C18** 选项卡，选中 **Enable integer promotions**。

术语表

ANSI

美国国家标准学会，是美国负责制订和批准的组织。

ASCII

美国信息交换标准码是使用 7 个二进制数字来表示每个字符的字符集编码。它包括大写和小写字母、数字、符号以及控制字符。

Build

编译并链接一个应用的所有源文件。

八进制 （Octal）

使用数字 0-7，以 8 为基数的计数体制。最右边的位表示 1 的倍数，右侧第二位表示 8 的倍数，右侧第三位表示 $8^2 = 64$ 的倍数，以此类推。

编译器 （Compiler）

将用高级语言编写的源文件翻译成机器码的程序。

C

具有表达式简练、现代控制流程和数据结构，以及运算符丰富等特点的通用编程语言。

COFF

公共目标文件格式。这种格式的目标文件包含机器码、调试及其他信息。

CPU

*参见*中央处理单元。

场景 （Scenario）

对于 MPLAB SIM 软件模拟器来说，是用于激励控制的一个特定设置。

操作码 （Opcode）

操作码。*参见*助记符。

程序存储器 （Program Memory）

器件中存储指令的存储器。亦指仿真器或软件模拟器中包含下载的目标应用固件的存储器。

程序计数器 （Program Counter）

包含正在执行的指令的地址的存储单元。

触发输出 （Trigger Output）

指可在任意地址或地址范围产生的仿真器输出信号，与跟踪和断点的设置无关。可设置任意个触发输出点。

次数计数器 （Pass Counter）

每次一个事件 （如执行特定地址处的一条指令）发生时都会递减 1 的计数器。当次数计数器的值为零时，事件满足。可将次数计数器分配给断点和跟踪逻辑，以及在 complex trigger （复杂触发）对话框中的任何连续事件。

存储类别 （Storage Class）

决定对象的生存时间。

存储模型 （Memory Models）

指定指向程序存储器的指针长度的描述。

存储限定符 （Storage Qualifier）

表明对象的特殊属性 （如 `volatile`）。

错误文件 （Error File）

包含由语言工具生成的错误消息和诊断的文件。

DSC

*参见*数字信号控制器。

单片机 （Microcontroller）

高度集成的芯片，它包括 CPU、 RAM、程序存储器、 I/O 端口和定时器。

单片机模式 （Microcontroller Mode）

PIC17 和 PIC18 系列单片机的一种程序存储器配置。在单片机模式下，仅允许内部执行。因此，在这种模式下仅可使用片内程序存储器。

导出 （Export）

以标准的格式将数据发送出 MPLAB IDE。

导入 （Import）

从外面的源 （如 hex 文件）将数据送入 MPLAB IDE。

递归 （Recursion）

已定义的函数或宏可调用自己的概念。当编写递归宏时要特别小心；当递归没有出口时容易陷入无限循环。

递归调用 （Recursive Call）

直接或间接调用自己的函数。

地址 （Address）

标识存储器中位置的值。

堆栈，软件 （Stack, Software）

用来存储返回地址、函数参数和局部变量的存储区。当用高级语言开发代码时，该存储区一般由编译器管理。

堆栈，硬件 （Stack, Hardware）

PICmicro 单片机中调用函数时存储返回地址的存储区。

段 （Section）

指定的代码或数据序列。

段属性 （Section Attribute）

段的特性 （如 access 段）。

断点，软件 （Breakpoint, Software）

一个地址，固件会在这个地址处暂停执行。通常由特殊的 break 指令获得。

断点，硬件 （Breakpoint, Hardware）

一种事件，执行这种事件会导致暂停。

EEPROM

电可擦除的可编程只读存储器。一种可电擦除的特殊 PROM。一次写或擦除一个字节。EEPROM 即使电源关闭时也能保留内容。

EPROM

可擦除的可编程只读存储器。通常通过紫外线照射来擦除的可编程只读存储器。

二进制 （Binary）

使用数字 0 和 1，以 2 为基数的计数体制。最右边的位表示 1 的倍数，右边第二位表示 2 的倍数，右边第三位表示 $2^2 = 4$ 的倍数，以此类推。

FNOP

强制空操作。强制 NOP 周期是双周期指令的第二个周期。由于 PICmicro 单片机的架构是流水线型，在执行当前指令的同时预取物理地址空间中的下一条指令，如果当前指令改变了程序计数器，那么这条预取的指令就被忽略了，导致一个强制 NOP 周期。

Free-Standing

一种 C 编译器实现，它接受任何不使用复杂类型的严格符合程序，而且在这种实现中，对 ISO 库条款中规定的属性的使用，仅限于标准头文件 <float.h>、<iso646.h>、<limits.h>、<stddef.h> 和 <stdint.h> 的内容。

仿真 （Emulation）

象执行存储在单片机中的固件一样执行装入仿真存储区中的软件的过程。

仿真存储器 （Emulation Memory）

仿真器内部的程序存储器。

仿真器 （Emulator）

执行仿真的硬件。

仿真器系统 （Emulator System）

MPLAB ICE 2000 和 4000 仿真器系统包括仿真器主机、处理器模块、器件适配器、电缆和 MPLAB IDE 软件。

仿真器主机 （Pod, Emulator）

包含仿真存储区、跟踪存储区、事件和周期定时器，以及跟踪 / 断点逻辑的仿真器盒子。

非扩展模式 （Non-Extended Mode）

在非扩展模式下，编译器不会使用扩展指令和立即数变址寻址模式；也称为 “传统” 模式。

非实时 （Non Real-Time）

指处理器执行到断点或单步执行指令，或 MPLAB IDE 运行在软件模拟器模式。

非易失性存储器 （Non-Volatile Storage）

电源关闭时保留其内容的存储器件。

符号 （Symbol）

描述组成程序的不同部分的一种通用机制。这些部分包括函数名、变量名、段名、文件名和结构 / 枚举 / 联合标记名等。 MPLAB IDE 中的符号主要指变量名、函数名和汇编标号。链接后符号的值就是其在存储器中的值。

GPR

通用寄存器。器件数据存储器 （RAM）的一部分，作为一般用途。

概要 （Profile）

对于 MPLAB SIM 软件模拟器来说，是对寄存器执行的激励的汇总表。

高级语言 （High-Level Language）

编写程序的语言，它比汇编语言更不依赖于具体的处理器。

跟踪 （Trace）

记录程序执行的仿真器或软件模拟器功能。仿真器将程序执行记录到其跟踪缓冲区内，并可上载到 MPLAB IDE 的跟踪窗口中。

跟踪存储区 （Trace Memory）

跟踪存储区包含在仿真器内部。跟踪存储区有时称为跟踪缓冲区。

工作簿 （Workbook）

对于 MPLAB SIM 软件模拟器来说，是一种用于产生 SCL 激励的设置。

观察变量 （Watch Variable）

调试会话期间可在 Watch 窗口中监控的变量。

归档 （Archive）

可重定位目标模块的集合。由将多个源文件编译 / 汇编为目标文件，然后使用归档器将目标文件组合为一个库文件生成。可将库与目标模块和其他库链接，生成可执行代码。

归档器 （Archiver）

生成和操作库的工具。

国际标准化组织 （International Organization for Standardization）

制订许多行业和技术 （包括计算和通讯）方面的标准的一个组织。

过滤器 （Filter）

通过选择确定在跟踪显示或数据文件中包含 / 排除哪些数据。

宏 （Macro）

宏指令。以缩写形式表示指令序列的指令。

宏伪指令 （Macro Directive）

控制宏定义体中执行和数据分配的伪指令。

Hex 代码 （Hex Code）

以十六进制格式代码存储的可执行指令。 hex 代码包含在 hex 文件中。

Hex 文件 （Hex File）

包含适用于对器件编程的十六进制地址和值 （hex 代码）的 ASCII 文件。

环境 — IDE （Environment – IDE）

应用程序开发桌面的特定布局。

环境 — MPLAB PM3 （Environment – MPLAB PM3）

包含关于如何烧写器件的文件的文件夹。可将这个文件传输到 SD™/MMC 卡。

汇编器 （Assembler ）

把汇编源代码翻译成机器码的语言工具。

汇编语言 （Assembly）

以符号形式描述二进制机器码的符号语言。

ICD

在线调试器。 MPLAB ICD 2 是 Microchip 的在线调试器。

ICE

在线仿真器。 MPLAB ICE 2000 和 4000 是 Microchip 的在线仿真器。

IDE

集成开发环境。 MPLAB IDE 是 Microchip 的集成开发环境。

IEEE

电子和电气工程师协会。

IRQ

*参见*中断请求。

ISO

*参见*国际标准化组织。

ISR

*参见*中断服务程序。

激励 （Stimulus）

软件模拟器的输入 （即为模拟对外部信号的响应而生成的数据）。通常数据采用文本文件中一系列动作的形式。激励可以是异步的，同步的 （引脚），时钟激励和寄存器激励。

机器码 （Machine Code）

处理器实际读和解释的计算机程序的表示。二进制机器码的程序由一系列机器指令（可能还包含数据）组成。特定处理器的所有可能指令的集合称为 “指令集”。

机器语言 （Machine Language）

特定中央处理单元的指令集，不需翻译即可用于处理器。

基数 （Radix）

数字基，十六进制或十进制，用于指定一个地址。

交叉引用文件 （Cross Reference File）

引用符号表的一个文件及引用符号的文件列表。如果定义了符号，列出的第一个文件是定义的位置。其他文件包含对符号的引用。

校准存储区 （Calibration Memory）

用于保存PICmicro单片机片内RC振荡器或其他外设校准值的特殊功能寄存器或通用寄存器。

节点 （Node）

MPLAB IDE 项目的组件。

警告 （Warning）

提醒出现了可能导致器件、软件文件或设备物理损坏的通知。

静态 RAM 或 SRAM （Static RAM or SRAM）

静态随机访问存储器。目标板上可读 / 写且不需要经常刷新的程序存储器。

局部标号 （Local Label）

用 LOCAL 伪指令在一个宏内部定义的标号。这些标号特定于宏实例化的一个给定示例。也就是说，声明为 local 的符号和标号在遇到 ENDM 宏后不再可访问。

绝对段 （Absolute Section）

具有链接器不能改变的固定 （绝对）地址的段。

看门狗定时器 （Watchdog Timer）

PICmicro 单片机中在一段可选择长度的时间后复位处理器的定时器。使用配置位来使能、禁止和设置 WDT。

可重定位 （Relocatable）

其段没有被分配到固定存储地址的目标文件。

可重入函数 （Reentrant）

可以有多个同时运行的实例的函数。在下面两种情况下可能发生函数重入：直接或间接递归调用函数；或者在由函数转入的中断处理过程中又执行此函数。

控制伪指令 （Control Directive）

汇编语言代码中根据汇编时指定表达式的值包含或忽略代码的伪指令。

库 （Library）

*参见*归档。

库管理器 （Librarian）

*参见*归档器。

快速存取存储区 （Access Memory）

PIC18 单片机中的一些特殊寄存器，对这些寄存器的访问与存储区选择寄存器（BSR）的设置无关。

扩展单片机模式 （Extended Microcontroller Mode）

在扩展单片机模式中，既可使用片内程序存储器，也可使用外部存储器。如果程序存储器地址大于 PIC17C 或 PIC18C 器件的内部存储空间，执行自动切换到外部存储器。

扩展模式 （Extended Mode）

在扩展模式下，编译器将使用扩展指令 （即 ADDFSR、 ADDULNK、 CALLW、 MOVSF、 MOVSS、 PUSHL、 SUBFSR 和 SUBULNK）和立即数变址寻址模式。

链接描述文件 （Linker Script File）

链接器的命令文件。定义链接选项并描述目标平台上的可用存储器。

链接器 （Linker）

把目标文件和库文件组合起来生成可执行代码并解析一个模块对另外一个模块引用的语言工具。

列表伪指令 （Listing Directive）

控制汇编器列表文件格式的伪指令。它们允许指定标题、分页及其他列表控制。

列表文件 （Listing File）

列出为每条 C 源语句生成的机器码，源文件中遇到的汇编指令、汇编器伪指令或宏的 ASCII 文本文件。

逻辑探头 （Logic Probe）

Microchip 的某些仿真器最多可连接 14 个逻辑探头。逻辑探头提供外部跟踪输入、触发输出信号、 +5V 和公共接地端。

Make 项目 （Make Project）

重新编译应用程序的命令，仅编译自上次编译完成后更改了的源文件。

MCU

单片机。 microcontroller 的缩写形式；也写作 µC。

MPASM 汇编器 （MPASM Assembler）

PICmicro 单片机、 KEELOQ® 器件及 Microchip 存储器件的 Microchip 可重定位宏汇编器。

MPLAB ICD 2

Microchip 的在线调试器，与 MPLAB IDE 配合工作。ICD 支持内置调试电路的闪存器件。每个 ICD 的主要组件是模块。一个完整的系统包括模块、适配头、演示板、电缆和 MPLAB IDE 软件。

MPLAB ICE 2000/4000

Microchip 的在线仿真器，与 MPLAB IDE 配合工作。MPLAB ICE 2000 支持 PICmicro MCU。MPLAB ICE 4000 支持 PIC18F MCU 和 dsPIC30F DSC。每个 ICE 的主要组件是仿真器主机。一个完整的系统包括仿真器主机、处理器模块、电缆和 MPLAB IDE 软件。

MPLAB IDE

Microchip 的集成开发环境。

MPLAB PM3

Microchip 的器件编程器。用于对 PIC18 单片机和 dsPIC® 数字信号控制器编程。可与 MPLAB IDE 配合使用或独立使用。将取代 PRO MATE® II 而使之废弃。

MPLAB SIM

Microchip 的软件模拟器，与 MPLAB IDE 配合工作，支持 PICmicro MCU 和 dsPIC DSC 器件。

MPLIB 目标库管理器 （MPLIB Object Librarian ）

MPLIB 库管理器是用于将由 MPASM 汇编器（mpasm 或 mpasmwin v2.0）或 MPLAB C1X C 编译器生成的 COFF 目标模块组合成库文件的目标库管理器。

MPLINK 目标链接器 （MPLINK Object Linker）

MPLINK链接器是Microchip MPASM汇编器和Microchip MPLAB C17或C18 C编译器的目标链接器。也可将 MPLINK 链接器与 Microchip MPLIB 库管理器配合使用。MPLINK 链接器设计为在 MPLAB IDE 中使用，尽管它也可独立于 MPLAB IDE 使用。

MRU

最近使用的。指可从 MPLAB IDE 主下拉菜单选择的文件和窗口。

命令行接口 （Command Line Interface）

仅基于文本输入和输出，在程序和其用户之间进行通讯的一种方式。

模板 （Template）

为以后插入自己的文件中使用而创建的文本行。 MPLAB 编辑器将模板存储到模板文件中。

目标 （Target）

指用户硬件。

目标板 （Target Board）

构成目标应用的电路和可编程器件。

目标处理器 （Target Processor）

目标应用板上的单片机。

目标代码 （Object Code）

由汇编器或编译器生成的机器码。

目标文件 （Object File）

包含机器码，也可能包含调试信息的文件。它可以直接执行；或为可重定位的，需要与其他目标文件 （如库文件）链接来生成完全可执行的程序。

目标文件伪指令 （Object File Directives）

仅当生成目标文件时使用的伪指令。

目标应用程序 （Target Application）

目标板上的软件。

NOP

空操作。执行该指令时，除了程序计数器加 1 外没有任何其他影响。

内部链接 （Internal Linkage）

如果不能从定义函数或变量的模块外部访问它们，则这样的函数或变量具有内部链接。

匿名结构 （Anonymous Structure）

为 C 联合的一个成员的未命名结构。对匿名结构的成员的访问与访问包含该匿名结构的联合的成员的方法相同。例如，在下面的代码中， hi 和 lo 都是联合 caster 中匿名结构的成员：

```
union castaway
 int intval;
 struct {
  char lo; //accessible as caster.lo
  char hi; //accessible as caster.hi
 };
} caster;
```

OTP

可一次编程。非窗口封装的 EPROM 器件。由于 EPROM 需要紫外线照射来擦除其存储内容，因此只有窗口片是可擦除的。

PC

个人计算机或程序计数器。

PC 主机 （PC Host）

运行有一个支持的 Windows 操作系统的任何 IBM 或兼容个人计算机。

PICmicro MCU

PICmicro 单片机 （MCU）指 Microchip 的所有单片机系列。

PICSTART Plus

Microchip 器件的开发编程器。可对 8、14、28 和 40 引脚的 PICmicro 单片机进行编程。必须与 MPLAB IDE 软件配合使用。

Pragma

一种伪指令，它对于特定的编译器有意义。通常将 pragma 用于向编译器传送实现定义的信息。 MPLAB C30 使用属性来传送这种信息。

PRO MATE II

Microchip 的器件编程器。可对大多数 PICmicro 单片机、大多数存储器和 KEELOQ 器件进行编程。可与 MPLAB IDE 配合使用或单独使用。

PWM 信号 （PWM Signal）

脉冲宽度调制信号。某些 PICmicro MCU 包含 PWM 外设。

跑表 （Stopwatch）

测量执行周期的计数器。

片外存储器 （Off-Chip Memory）

指 PIC17 或 PIC18 器件的一种存储器选择，这种情况下存储器可位于目标板上，或所有程序存储器都由仿真器提供。

配置位 （Configuration Bit）

可编程来设置 PICmicro 单片机工作模式的专用位。配置位可或不可再编程。

器件编程器 （Device Programmer）

用于对电可编程半导体器件 （如单片机）进行编程的工具。

嵌套深度 （Nesting Depth）

宏可包含其他宏的最大深度。

RAM

随机访问存储器 （数据存储器）。可以以任意顺序访问这种存储器中的信息。

ROM

只读存储器 （程序存储器）。不能修改的存储器。

Run

将仿真器从暂停状态释放，允许仿真器实时运行应用代码、实时改变 I/O 状态或实时响应 I/O 的命令。

软件模拟器 （Simulator）

模拟器件操作的软件程序。

SFR

*参见*特殊功能寄存器。

Single Step

这一命令单步执行代码，一次执行一条指令。执行每条指令后， MPLAB IDE 更新寄存器窗口、通过查看变量及状态显示，可分析和调试指令。也可单步执行 C 编译器源代码，但不是每次执行一条指令，MPLAB IDE 将执行一行高级 C 语句生成的所有汇编指令。

Skew

不同时间出现在处理器总线上与指令执行有关的信息。例如，执行前一条指令的过程中取指时，被执行的操作码出现在总线上；当实际执行操作码时，源数据地址及其值以及目标数据地址出现在总线上。当执行下一条指令时，目标数据值出现在总线上。跟踪缓冲区一次捕捉总线上的这些信息。因此，跟踪缓冲区的一条记录将包含三条指令的执行信息。执行一条指令时，从一条信息到另一条信息的捕捉周期数称为 skew。

Skid

当使用硬件断点来暂停处理器时，在处理器暂停前可能再执行一条或多条指令。断点后执行的指令条数称为 skid。

Step Into

这一命令与 Single Step 相同。Step Into（与 Step Over 相对）在 CALL 指令后，单步执行子程序。

Step Out

Step Out 允许跳出当前正在单步执行的子程序。此命令执行子程序中剩下的代码，然后在子程序的返回地址处停止执行。

Step Over

Step Over 允许单步执行时跳过子程序。这个命令执行子程序中的代码，然后在子程序的返回地址处停止执行。

当 step over 一条 CALL 指令时，下一个断点将设置在 CALL 指令后的下一条指令处。如果由于某种原因，子程序陷入无限循环或不正确返回，下一个断点将永远执行不到。选择 **Halt** 来重新获得对程序执行的控制。

闪存 （Flash）

按块 （而不是按字节）写或擦除数据的一种 EEPROM。

上电复位仿真 （Power-on-Reset Emulation）

在开始为应用上电时，将随机值写到数据 RAM 区中来模拟 RAM 中的未初始化值的软件随机过程。

上载 （Upload）

上载功能将数据从一个工具，如仿真器或编程器，传送到主机 PC，或将数据从目标板传送到仿真器。

事件 （Event）

对可能包含地址、数据、次数计数、外部输入、周期类型 （取指和读 / 写）及时间标记的总线周期的描述。事件用于描述触发、断点和中断。

十六进制 （Hexadecimal）

使用数字 0-9 以及字母 A-F （或 a-f），以 16 为基数的计数体制。字母 A-F 表示十进制数 10 到 15。最右边的位表示 1 的倍数，右边第二位表示 16 的倍数，第三位表示 16^2 = 256 的倍数，以此类推。

实时 （Real-Time）

当从仿真器或 MPLAB ICD 模式中的暂停状态释放时，处理器以实时模式运行且与芯片的正常操作相同。在实时模式下，使能 MPLAB ICE 的实时跟踪缓冲区，并持续捕捉所有选择的周期，使能所有 break 逻辑。在仿真器或 MPLAB ICD 模式下，处理器实时运行，直到有效断点导致暂停，或者直到用户暂停仿真器。在软件模拟器模式下，实时仅意味着单片机指令的执行速度与主机 CPU 可模拟的指令速度一样快。

数据存储器 （Data Memory）

在 Microchip MCU 和 DSC 器件中，数据存储器 （RAM）由通用寄存器 （GPR）和特殊功能寄存器 （SFR）组成。某些器件还有 EEPROM 数据存储器。

数据伪指令 （Data Directive）

指控制汇编器的程序和数据存储空间分配，并提供通过符号 （即有意义的名字）引用数据项的方法的伪指令。

数字信号控制器 （Digital Signal Controller）

具有数字信号处理能力的单片机 （如 Microchip dsPIC® 器件）。

特殊功能寄存器 （Special Function Register）

数据存储器 （RAM）的一部分，专用于控制 I/O 处理函数、I/O 状态、定时器或其他模式及外设的寄存器。

条件编译 （Conditional Compilation）

只有当预处理伪指令指定的某个常量表达式为真时才编译程序段的操作。

Watch 窗口 （Watch Window）

Watch 窗口包含一系列观察变量，这些变量在每次执行到断点时更新。

WDT

*参见*看门狗定时器。

外部 RAM （External RAM）

片外的读 / 写存储器。

外部符号 （External Symbol）

具有外部链接的标识符符号。这可能是一个引用或一个定义。

外部符号解析 （External Symbol Resolution）

链接器搜集所有输入模块的外部符号定义来解析所有外部符号引用的过程。没有相应定义的任何外部符号引用都会导致报告链接器错误。

外部输入线 （External Input Line）

用于根据外部信号设置事件的外部输入信号逻辑探针线 （TRIGIN）。

微处理器模式 （Microprocessor Mode）

PIC17 和 PIC18 系列单片机的一种程序存储器配置。在微处理器模式下，不使用片内的程序存储器。整个程序存储器映射到外部。

伪指令 （Directive）

源代码中控制语言工具操作的语句。

未初始化数据 （Uninitialized Data）

定义时未提供初始值的数据。在 C 中，

```
int myVar;
```

定义了将存放到未初始化数据段的一个变量。

文件寄存器 （File Register）

片内数据存储器，包括通用寄存器 （GPR）和特殊功能寄存器 （SFR）

系统窗口控制 （System Window Control）

系统窗口控制位于窗口或某些对话框的左上角。点击这一控制通常会弹出包含“Minimize （最小化）”、“Maximize （最大化）”和 “Close （关闭）”等项的菜单。

下载 （Download）

数据从主机发送到其他设备，如仿真器、编程器或目标板的过程。

限定符 （Qualifier）

次数计数器使用的地址或地址范围，或用作复杂触发中另一个操作之前的事件。

向量 （Vector）

指定事件 （如复位或中断）发生时，应用程序跳转到的存储器地址。

项目 （Project）

为应用构建目标代码和可执行代码的一组源文件及指令。

消息 （Message）

显示出来的文本，警告在语言工具的操作中可能存在的问题。消息不会停止操作。

小尾数法 （Little Endianess）

多字节数据的数据存储顺序机制。在这种机制中，最低有效字节存储到较低的地址。

样机系统 （Prototype System）

指用户目标应用或目标板的一个术语。

已分配段 （Assigned Section）

已在链接器命令文件中分配到目标存储区的段。

异步事件 （Asynchronous Events）

不同时发生的多个事件。通常用来指可能在处理器执行过程中的任意时刻发生的中断。

异步激励 （Asynchronous Stimulus）

使用软件模拟器时，为模拟被模拟器件的外部输入而生成的数据。

应用 （Application）

可由 PICmicro 单片机控制的一组软硬件。

源代码 （Source Code）

编程人员编写计算机程序的形式。采用某种正式的编程语言编写源代码，可翻译为机器码或被解释程序执行。

源文件 （Source File）

包含源代码的 ASCII 文本文件。

原始数据 （Raw Data）

与一个段有关的代码或数据的二进制表示。

运算符 （Operator）

构成表达式时使用的符号，如加法符号 “+”和减法符号 “-”。每个运算符都有用于确定求值顺序的指定优先级。

运行时模型 （Runtime Model）

描述目标架构资源的使用。

暂停 （Halt）

停止程序执行。执行 Halt 与在断点处停止相同。

帧指针 （Frame Pointer）

指向堆栈中位置的指针，它将堆栈中的函数参数和局部变量分隔开。提供了一种方便的方式来访问局部变量和当前函数的其他值。

指令 （Instruction）

告知中央处理单元执行特定操作，并包含操作中要使用的数据的位序列。

指令集 （Instruction Set）

特定处理器理解的机器语言指令的集合。

致命错误 （Fatal Error）

引起编译立即中止的错误。不产生其他消息。

中断 （Interrupt）

传递到 CPU 的信号，它使 CPU 暂停执行正在运行的应用程序，把控制权转交给中断服务程序 （ISR），以处理事件。

中断处理程序 （Interrupt Handler）

发生中断时处理特殊代码的子程序。

中断服务程序 （Interrupt Service Routine）

当产生中断时进入的用户生成代码。代码在程序存储器中的位置通常取决于所产生中断的类型。

中断请求 （Interrupt Request）

使处理器暂停正常的指令执行并开始执行中断处理程序的事件。某些处理器有几种中断请求事件，允许具有不同优先级的中断。

中断响应时间 （Latency）

从事件发生到得到响应的时间。

中央处理单元 （Central Processing Unit）

器件的一部分，负责取出要执行的正确指令，对指令进行译码，然后执行指令。需要时，它和算术逻辑单元 （ Arithmetic Logic Unit， ALU）配合工作，来完成指令的执行。它控制程序存储器地址总线、数据存储器地址总线和对堆栈的访问。

助记符 （Mnemonics）

可直接翻译为机器码的文本指令。也称为操作码。

状态条 （Status Bar）

状态条位于 MPLAB IDE 窗口的底部，表明光标位置、开发模式和器件，以及有效工具条等当前信息。

字节存储顺序 （endianness）

描述多字节对象的字节存储顺序。

字母数字字符 （Alphanumeric）

字母数字字符由字母字符和十进制数字 （0, 1, …, 9）组成。

字母字符 （Alphabetic Character）

字母字符指属于阿拉伯字母表 （a, b, …, z, A, B, …, Z）中字母的字符。

索引

注：

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://support.microchip.com
网址：www.microchip.com

亚特兰大 Atlanta
Alpharetta, GA
Tel: 1-770-640-0034
Fax: 1-770-640-0307

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Farmington Hills, MI
Tel: 1-248-538-2250
Fax: 1-248-538-2260

科科莫 Kokomo
Kokomo, IN
Tel: 1-765-864-8360
Fax: 1-765-864-8387

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608

圣克拉拉 Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

加拿大多伦多 Toronto
Mississauga, Ontario,
Canada
Tel: 1-905-673-0699
Fax: 1-905-673-6509

亚太地区

亚太总部 Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Habour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

中国 – 北京
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

中国 – 成都
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

中国 – 福州
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

中国 – 香港特别行政区
Tel: 852-2401-1200
Fax: 852-2401-3431

中国 – 青岛
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

中国 – 上海
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

中国 – 沈阳
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

中国 – 深圳
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

中国 – 顺德
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

中国 – 武汉
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

中国 – 西安
Tel: 86-29-8833-7250
Fax: 86-29-8833-7256

台湾地区 – 高雄
Tel: 886-7-536-4818
Fax: 886-7-536-4803

台湾地区 – 台北
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

台湾地区 – 新竹
Tel: 886-3-572-9526
Fax: 886-3-572-6459

亚太地区

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

印度 India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

印度 India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

印度 India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

日本 Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

韩国 Korea - Gumi
Tel: 82-54-473-4301
Fax: 82-54-473-4302

韩国 Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 或
82-2-558-5934

马来西亚 Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

菲律宾 Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

新加坡 Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

泰国 Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

欧洲

奥地利 Austria - Wels
Tel: 43-7242-2244-3910
Fax: 43-7242-2244-393

丹麦 Denmark-Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

法国 France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

德国 Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

荷兰 Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

英国 UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

08/29/06