



MPASM™ 汇编器
MPLINK™ 目标链接器
MPLIB™ 目标库管理器
用户指南

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。

提供本文档的中文版本仅为了便于理解。Microchip Technology Inc. 及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc. 的英文原本文档。

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利, 它们可能由更新之信息所替代。确保应用符合技术规范, 是您自身应负的责任。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保, 包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。Microchip 对因这些信息及使用这些信息而引起的后果不承担任何责任。未经 Microchip 书面批准, 不得将 Microchip 的产品用作生命维持系统中的关键组件。在 Microchip 知识产权保护下, 不得暗或以其他方式转让任何许可证。

商标

Microchip 的名称和徽标组合、Microchip 徽标、Accuron、dsPIC、KEELOQ、microID、MPLAB、PIC、PICmicro、PICSTART、PRO MATE、PowerSmart、rfPIC 和 SmartShunt 均为 Microchip Technology Inc. 在美国和其他国家或地区的注册商标。

AmpLab、FilterLab、Migratable Memory、MXDEV、MXLAB、PICMASTER、SEEVAL、SmartSensor 和 The Embedded Control Solutions Company 均为 Microchip Technology Inc. 在美国的注册商标。

Analog-for-the-Digital Age、Application Maestro、dsPICDEM、dsPICDEM.net、dsPICworks、ECAN、ECONOMONITOR、FanSense、FlexROM、fuzzylAB、In-Circuit Serial Programming、ICSP、ICEPIC、Linear Active Thermistor、MPASM、MPLIB、MPLINK、MPSIM、PICKit、PICDEM、PICDEM.net、PICLAB、PICtail、PowerCal、PowerInfo、PowerMate、PowerTool、rfLAB、rfPICDEM、Select Mode、Smart Serial、SmartTel、Total Endurance 和 WiperLock 均为 Microchip Technology Inc. 在美国和其他国家或地区的商标。

SQTP 是 Microchip Technology Inc. 在美国的服务标记。

在此提及的所有其他商标均为各持有公司所有。

© 2005, Microchip Technology Inc. 版权所有。

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

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 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于 2003 年 10 月通过了 ISO/TS-16949:2002 质量体系认证。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

前言	1
PICmicro 语言工具和 MPLAB IDE	9
第 1 部分 ——MPASM 汇编器	
第 1 章 MPASM 汇编器概述	
1.1 简介	21
1.2 MPASM 汇编器的定义	21
1.3 MPASM 汇编器为您提供的帮助	21
1.4 汇编器移植路径	22
1.5 汇编器兼容性问题	22
1.6 汇编器操作	22
1.7 汇编器输入 / 输出文件	24
第 2 章 汇编器界面	
2.1 简介	31
2.2 MPLAB IDE 界面	31
2.3 Windows 界面	32
2.4 命令 shell 界面	33
2.5 命令行接口	35
2.6 疑难解答	37
第 3 章 表达式语法和运算法则	
3.1 简介	39
3.2 文本字符串	39
3.3 保留字和段名	41
3.4 数字常数和基数	41
3.5 算术运算符和优先级	42
第 4 章 伪指令	
4.1 简介	45
4.2 伪指令的类型	45
4.3 access_ovr ——在快速操作 RAM 中开始目标文件覆盖段 (PIC18 MCU)	48
4.4 __badram ——标识未用的 RAM	48
4.5 __badrom ——标识未用的 ROM	49
4.6 bankisel ——生成间接存储区选择代码 (PIC12/16 MCU)	50
4.7 banksel ——生成存储区选择代码	52
4.8 cblock ——定义常数块	54
4.9 code ——开始目标文件代码段	56

4.10 code_pack ——开始一个在目标文件中被压缩的代码段 (PIC18 MCU)	57
4.11 __config ——设置处理器的配置位	58
4.12 config ——设置处理器的配置位 (PIC18 MCU)	59
4.13 constant ——声明符号常数	60
4.14 da ——在程序存储器中存储字符串 (PIC12/16 MCU)	61
4.15 data ——创建数字和文本数据	62
4.16 db ——声明一个字节的数	65
4.17 de ——声明 EEPROM 数据字节	67
4.18 #define ——定义文本替换标号	68
4.19 dt ——定义表 (PIC12/16 MCU)	70
4.20 dw ——声明一个字的数据	70
4.21 else ——开始 if 条件的备用汇编块	71
4.22 end ——结束程序块	71
4.23 endc ——结束自动常数块	72
4.24 endif ——结束条件汇编块	72
4.25 endm ——结束宏定义	73
4.26 endw ——结束 while 循环	73
4.27 equ ——定义一个汇编器常数	74
4.28 error ——发出一条错误消息	74
4.29 errorlevel ——设置消息级别	76
4.30 exitm ——退出宏	78
4.31 expand ——扩展宏列表	80
4.32 extern ——声明一个外部定义的标号	80
4.33 fill ——指定程序存储器填充值	82
4.34 global ——导出标号	84
4.35 idata ——开始目标文件已初始化的数据段	85
4.36 idata_acs ——在快速操作 RAM 中开始目标文件已初始化的数据段 (PIC18 MCU)	86
4.37 __idlocs ——设置处理器 ID 单元	87
4.38 if ——开始条件汇编代码块	88
4.39 ifdef ——如果已经定义了符号则执行	90
4.40 ifndef ——如果未定义符号则执行	91
4.41 #include ——包含额外的源文件	92
4.42 list ——列表选项	93
4.43 local ——声明局部宏变量	94
4.44 macro ——声明宏定义	96
4.45 __maxram ——定义最大 RAM 单元	97
4.46 __maxrom ——定义最大 ROM 单元	98
4.47 messg ——创建用户定义的消息	98
4.48 noexpand ——关闭宏扩展	100
4.49 nolist ——关闭列表输出	100
4.50 org ——设置程序起始处	100
4.51 page ——在列表文件中插入换页符	103
4.52 pagesel ——生成页面选择代码 (PIC10/12/16 MCU)	103

4.53 pageselw ——使用 WREG 命令生成页选择代码 (PIC10/12/16 MCU)	105
4.54 processor ——设置处理器类型	106
4.55 radix ——指定默认基数	106
4.56 res ——保留存储器	107
4.57 set ——定义汇编器变量	109
4.58 space ——插入空白列表行	110
4.59 subtitle ——指定程序副标题	110
4.60 title ——指定程序标题	111
4.61 udata ——开始目标文件中未初始化的数据段	111
4.62 udata_acs ——开始目标文件未初始化快速操作的数据段 (PIC18 MCU)	113
4.63 udata_ovr ——开始目标文件中覆盖的未初始化的数据段	114
4.64 udata_shr ——开始目标文件中共享的未初始化的数据段 (PIC12/16 MCU)	116
4.65 #undefine ——删除替换标号	117
4.66 variable ——声明符号变量	118
4.67 while ——当条件为 TRUE 时执行循环	119
第 5 章 汇编器示例、技巧和窍门	
5.1 简介	123
5.2 显示端口计数示例	124
5.3 端口 B 交替和延时程序的示例	125
5.4 使用变量和常数进行计算的示例	132
5.5 32 位延时程序的示例	134
5.6 在固件中仿真 SPI™ 的示例	136
5.7 十六进制字节到 ASCII 字节转换的示例	138
5.8 获取示例的其他渠道	139
5.9 技巧和窍门	139
第 6 章 可重定位目标	
6.1 简介	143
6.2 头文件	143
6.3 程序存储器	144
6.4 Low、High 和 Upper 操作数	144
6.5 RAM 分配	147
6.6 配置位和 ID 单元	148
6.7 访问来自其他模块的标号	148
6.8 分页和分区问题	149
6.9 生成目标模块	150
6.10 代码示例	150

第 7 章 宏语言

7.1 简介	153
7.2 宏语法	153
7.3 已定义的宏指令	154
7.4 宏定义	154
7.5 宏调用	154
7.6 宏代码示例	155

第 8 章 错误、警告、消息和限制

8.1 简介	157
8.2 汇编器错误	157
8.3 汇编器警告	163
8.4 汇编器消息	166
8.5 汇编器限制	168

第 2 部分 —— MPLINK 目标链接器

第 9 章 MPLINK 链接器概述

9.1 简介	171
9.2 MPLINK 链接器的定义	171
9.3 MPLINK 链接器工作原理	171
9.4 MPLINK 链接器的功能	172
9.5 所支持的链接器平台	172
9.6 链接器工作原理	172
9.7 链接器输入 / 输出文件	173

第 10 章 链接器接口

10.1 简介	179
10.2 MPLAB IDE 接口	179
10.3 命令行接口	179
10.4 命令行示例	180

第 11 章 链接描述文件

11.1 简介	181
11.2 标准链接描述文件	181
11.3 链接描述文件命令行信息	182
11.4 链接描述文件忠告	183
11.5 存储器区域定义	183
11.6 逻辑代码段定义	185
11.7 堆栈定义	186

第 12 章 链接器处理过程

12.1 简介	187
12.2 链接器处理过程概述	187
12.3 链接器分配算法	188
12.4 分配示例	189
12.5 已初始化数据	190
12.6 保留的段名	190

第 13 章 应用程序示例

13.1 简介	191
13.2 如何编译应用程序示例	191
13.3 应用程序示例 1——修改链接描述文件	193
13.4 应用程序示例 2——存放代码和设置配置位	195
13.5 应用程序示例 3——使用引导加载程序	198
13.6 应用程序示例 4——配置外部存储器	208

第 14 章 错误、警告和常见问题

14.1 简介	213
14.2 链接器解析错误	213
14.3 链接器错误	215
14.4 链接器警告	220
14.5 库文件错误	220
14.6 COFF 文件错误	221
14.7 COFF 到 COD 转换错误	222
14.8 COFF 到 COD 转换警告	222
14.9 COD 文件错误	222
14.10 Hex 文件错误	222
14.11 常见问题	223

第 3 部分 —— MPLIB 目标库管理器

第 15 章 MPLIB 库管理器概述

15.1 简介	227
15.2 MPLIB 库管理器介绍	227
15.3 MPLIB 库管理器的工作原理	227
15.4 MPLIB 库管理器为您提供的帮助	228
15.5 库管理器操作	228
15.6 库管理器输入 / 输出文件	229

第 16 章 库管理器界面

16.1 简介	231
16.2 MPLAB IDE 界面	231
16.3 命令行选项	232
16.4 命令行示例和提示	232

第 17 章 错误

17.1 简介	233
17.2 库管理器解析错误	233
17.3 库文件错误	233
17.4 COFF 文件错误	233

第 4 部分 —— 附录

附录 A 指令集

A.1 简介	237
A.2 12 或 14 位宽的指令集中的关键字	237
A.3 12 位宽指令集	239
A.4 14 位宽指令集	240
A.5 12 位或 14 位宽的伪指令	242
A.6 PIC18 器件指令集中的关键字	243
A.7 PIC18 器件指令集	244
A.8 PIC18 器件扩展指令集	248

附录 B 有用的表格

B.1 简介	249
B.2 ASCII 字符集	249
B.3 十六进制转换到十进制	250

术语表	251
-----------	------------

索引	265
----------	------------

全球销售及服务中心	270
-----------------	------------

前言

用户须知

所有文档均会更新，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 “DSXXXXA”，其中 “XXXX” 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

本章包含使用汇编器 / 链接器 / 库管理器用户指南前需要了解的有用的一般信息。内容包括：

- 文档编排
- 本指南使用的约定
- 推荐读物
- **Microchip** 网站
- 开发系统客户变更通知服务
- 客户支持

文档编排

本文档介绍了如何使用 **MPASM[™]** 汇编器、**MPLINK[™]** 目标链接器和 **MPLIB[™]** 目标库管理器开发 **PICmicro[®]** 单片机（MCU）应用。所有这些工具都可在 **MPLAB[®]** 集成开发环境（Integrated Development Environment，IDE）中使用。参见 **MPLAB IDE** 文档了解 **MPLAB IDE** 基本功能的详细信息。

PICmicro 语言工具概述——概述了如何在 **MPLAB IDE** 中使用本手册中的所有工具。这是大多数开发者使用这些工具的方式。

MPASM 汇编器

- **第 1 章：MPASM 汇编器概述**——介绍了 MPASM 汇编器的定义、功能以及它与其他工具配合使用的方式。
- **第 2 章：汇编器界面**——回顾了 MPLAB IDE 中使用 MPASM 汇编器的方法，并介绍了在命令 Shell 界面或 Windows Shell 界面中使用汇编器的方法。
- **第 3 章：表达式语法和运算法则**——提供对在 MPASM 汇编器源文件中使用复杂表达式的指导。
- **第 4 章：伪指令**——按字母顺序罗列每条 MPASM 汇编器伪指令，并对其进行详细说明，带有示例。
- **第 5 章：汇编器示例、技巧和窍门**——提供在应用程序中综合使用 MPASM 汇编器伪指令的方法并用实例说明。
- **第 6 章：可重定位目标**——介绍 MPASM 汇编器与 MPLINK 链接器配合使用的方法。
- **第 7 章：宏语言**——介绍使用 MPASM 汇编器内嵌宏处理器的方法。
- **第 8 章：错误、警告、消息和限制**——包含一组 MPASM 汇编器生成的错误、警告和消息的描述列表及工具限制。

MPLINK 目标链接器

- **第 9 章：MPLINK 链接器概述**——介绍了 MPLINK 目标链接器的定义、功能以及它与其他工具配合使用的方式。
- **第 10 章：链接器接口**——回顾了 MPLAB IDE 中使用 MPLINK 链接器的方法并介绍了用命令行使用链接器的方法。
- **第 11 章：链接描述文件**——讨论了生成和使用描述文件控制链接器工作的方法。
- **第 12 章：链接器处理过程**——介绍链接器处理文件的方式。
- **第 13 章：应用程序示例**——提供了如何使用链接器创建应用程序的示例。
 - **应用程序示例 1**——解释了如何找到和使用模板文件，以及如何修改链接描述文件。
 - **应用程序示例 2**——解释了如何将程序代码保存在不同的存储区，如何将数据表保存到 ROM 及如何用 C 语言设置配置位。
 - **应用程序示例 3**——解释了如何为引导安装程序分区以及如何编译将载入外部 RAM 并执行的代码。
 - **应用程序示例 4**——解释了如何新建链接器描述存储区段，如何通过 `#pragma code` 伪指令声明外部存储区以及如何使用 C 指针访问外部存储区。
- **第 14 章：错误、警告和常见问题**——包含一组 MPLINK 链接器生成的错误和消息的描述列表及常见问题和工具限制。

MPLIB 目标库管理器

- **第 15 章：MPLIB 库管理器概述**——介绍了 MPLIB 目标库管理器的定义、功能以及它与其他工具配合使用的方式。
- **第 16 章：库管理器界面**——回顾了在 MPLAB IDE 中使用 MPLIB 库管理器的方法并介绍了用命令行使用库管理器的方法。
- **第 17 章：错误**——包含一组关于 MPLIB 库管理器生成的错误、警告和消息的描述列表。

附录

- **附录 A：指令集**——罗列了 PICmicro MCU 器件的指令集。
- **附录 B：有用的表格**——为代码开发提供了一些有用的表格。
 - **ASCII 字符集**——罗列了 ASCII 字符集。
 - **十六进制到十进制转换**——显示如何将十六进制数转换成十进制数。

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	涵义	示例
Arial 字体:		
斜体字	参考书目	<i>MPLAB® IDE User's Guide</i>
	需强调的文字	<i>... 仅有的编译器 ...</i>
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	窗口或对话框中的字段名	“Save project before build”
带右尖括号有下划线的斜体文字	菜单路径	<i>File>Save</i>
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	autoexec.bat
	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	<i>file.o</i> , 其中 <i>file</i> 可以是任一有效文件名
方括号 []	可选参数	mpasm [options] <i>file</i> [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

本文档介绍如何使用汇编器 / 链接器 / 库管理器用户指南。下面列出了其他有用的文档。以下 Microchip 文档均已提供，建议作为补充参考资料使用。

自述文件——**readme.asm** 和 **readme.lkr**

有关工具的最新信息和已知问题，请参见 MPASM 汇编器自述文件 (**readme.asm**) 或 MPLINK 目标链接器 / MPLIB 目标库管理器自述文件 (**readme.lkr**)。可以在 MPLAB IDE 安装目录 “Readme” 文件夹下找到这些 ASCII 文本文件。

在线帮助

在线帮助提供了全面的 MPASM 汇编器、MPLINK 链接器 / MPLIB 目标库管理器帮助文件。

MPASM and MPLINK PICmicro Quick Reference Card (DS30400)

快速参考卡 (QRC) 包含 MPASM 汇编器伪指令语言综述、支持的 MPASM 汇编器基数类型、MPLINK 目标链接器命令行选项、MPLIB 目标库管理器使用格式和示例、PIC18 器件的特殊功能寄存器、ASCII 字符集和 PICmicro MCU 指令集综述。

C 编译器用户指南和库

MPLINK 链接器和 MPLIB 库管理器还可与针对 PIC18 器件的 Microchip C 语言编译器 MPLAB C18 配合使用。参见以下文档了解更多关于 MPLAB C18 的信息：

- MPLAB C18 C 编译器入门 (DS51295E_CN)
- MPLAB C18 C 编译器用户指南 (DS51288C_CN)
- MPLAB C18 C 编译器函数库 (DS51297C_CN)
- PIC18 Configuration Settings Addendum (DS51537)

MPLAB IDE 文档

关于 MPLAB IDE 集成开发环境的信息可参见以下文档：

- MPLAB IDE Quick Chart (DS51410) ——快速查找图表。
- MPLAB IDE 用户指南 (DS51519A_CN) ——全面的用户手册。
- MPLAB IDE 快速入门 (DS51281C_CN) ——用户指南的第 1 章和第 2 章。
- 在线帮助文件——MPLAB IDE 的最新信息。

PICmicro MCU 数据手册和应用笔记

数据手册包含器件工作原理以及电气规范等信息。应用笔记演示如何使用不同的 PICmicro MCU。可在 Microchip 网站上找到相应器件的上述两种文档。

MICROCHIP 网站

Microchip 网站 (www.microchip.com) 为客户提供在线支持。客户可通过该网站方便地获取文件和信息。只要使用常用的因特网浏览器即可访问，网站提供以下信息：

- **产品支持**——数据手册和勘误表、应用笔记和样本程序、设计资源、用户指南以及硬件支持文档、最新的软件发布以及存档软件。
- **一般技术支持**——常见问题（Frequently Asked Questions, FAQ）、技术支持请求、在线讨论组、Microchip 顾问计划成员名单。
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、Microchip 销售办事处、代理商以及工厂代表列表。

开发系统客户变更通知服务

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“客户变更通知”服务并按照注册说明完成注册。

开发系统产品分类如下：

- **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**——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 IDE 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 和 PRO MATE® II 器件编程器以及 PICSTAR® Plus 和 PICKit™ 1 开发编程器。

客户支持

Microchip 产品的用户可以通过以下渠道获取帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（Field Application Engineer， FAE）
- 技术支持
- 开发系统信息热线

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:

PICmicro 语言工具和 MPLAB IDE

简介

MPASM 汇编器、MPLINK 目标链接器和 MPLIB 目标库管理器通常在 MPLAB IDE 中一起使用以提供对 PICmicro MCU 器件的应用程序代码的 GUI 开发。在此将讨论如何通过 MPLAB IDE 对这些 PICmicro 语言工具进行操作。

本章包括以下主题：

- MPLAB IDE 和工具安装
- MPLAB IDE 的设置
- MPLAB IDE 项目
- 项目建立
- 项目示例

MPLAB IDE 和工具安装

要通过 MPLAB IDE 使用 PICmicro 语言工具，必须首先安装 MPLAB IDE。该免费软件的最新版本可在我们的网站（<http://www.microchip.com>）下载或向任何销售办事处（见封底）索取。在安装 MPLAB IDE 时，同时也会安装 MPASM 汇编器、MPLINK 目标链接器和 MPLIB 目标库管理器。

在默认情况下，语言工具将被安装在以下目录：

C:\Program Files\Microchip\MPASM Suite

各个工具的以下可执行文件都在此目录中：

- MPASM 汇编器——mpasmwin.exe
- MPLINK 目标链接器——mplink.exe
- MPLIB 目标库管理器——mplib.exe

所有器件的包含（Include，头）文件也都在此目录中。欲知更多有关这些文件的信息，请参见 MPASM 汇编器文档。

所有的器件链接描述文件都在 LKR 子目录中。欲知更多有关这些文件的信息，请参见 MPLINK 目标链接器文档。

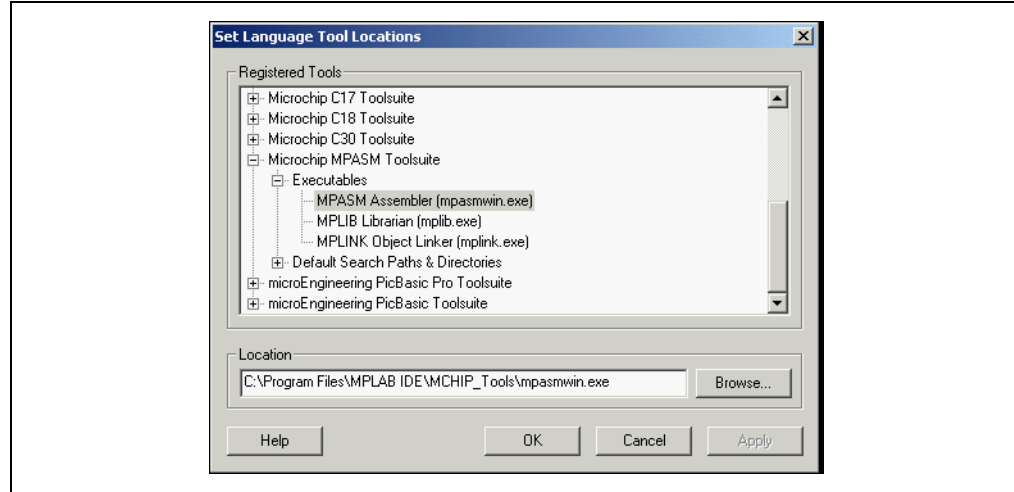
子目录中还包括代码示例和模板文件可供使用。还提供了用于绝对代码（Code）和可重定位代码（Object）开发的模板文件。

MPLAB IDE 的设置

一旦在 PC 上安装了 MPLAB IDE，请检查以下设置以确保已在 MPLAB IDE 下正确识别了语言工具。

1. 在 MPLAB IDE 菜单栏中，选择 **Project>Set Language Tool Locations** 打开对话框以设置 / 检查语言工具的可执行文件的位置。

图 1： 设置语言工具的位置

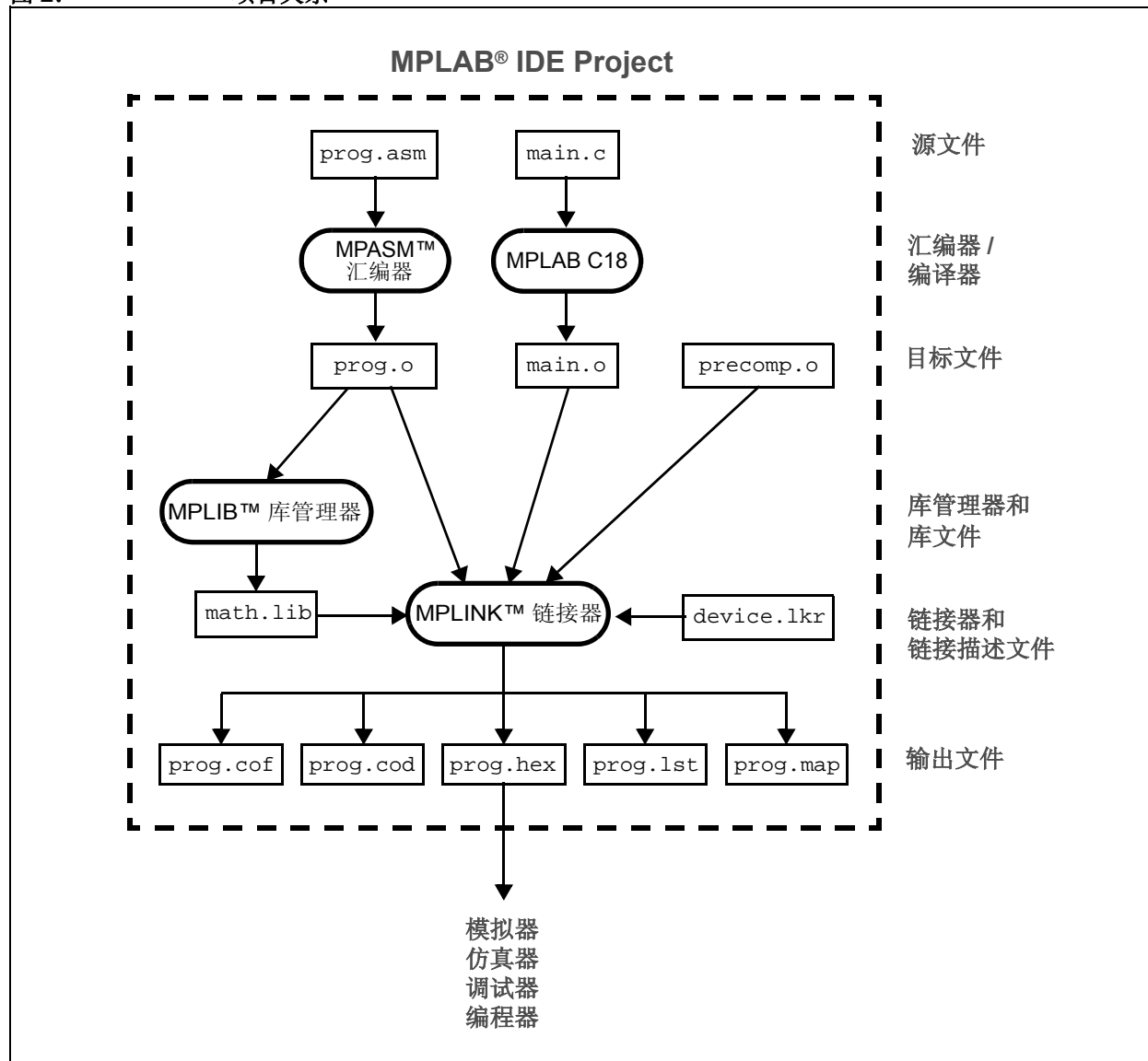


2. 在对话框中，从 “Registered Tools” 中选择 “Microchip MPASM Toolsuite”。单击 “+” 展开。
3. 选择 “Executables”。单击 “+” 展开。
4. 选择 “MPASM Assembler (mpasmwin.exe)”。在 “Location” 下，应该显示可执行文件的路径。如果没有显示路径，则输入路径或浏览到此文件的位置。其默认位置列在 “MPLAB IDE 和工具安装” 中。
5. 选择 “MPLIB Object Librarian (mplib.exe)”。在 “Location” 下，应该显示可执行文件的路径。如果没有显示路径，则输入路径或浏览到此文件的位置。其默认位置列在 “MPLAB IDE 和工具安装” 中。
6. 选择 “MPLINK Object Linker (mplink.exe)”。在 “Location” 下，应该显示可执行文件的路径。如果没有显示路径，则输入路径或浏览到此文件的位置。其默认位置列在 “MPLAB IDE 和工具安装” 中。
7. 单击 **OK**。

MPLAB IDE 项目

MPLAB IDE 中的项目是编译一个应用程序所需的一组文件以及它们与各种编译工具的关联。以下是通用的 MPLAB IDE 项目。

图 2: 项目关系



在此 MPLAB IDE 项目中，汇编源文件 (`prog.asm`) 和关联的汇编器 (MPASM 汇编器) 一起显示出来。MPLAB IDE 将使用此信息生成目标文件 `prog.o`，此文件将输入到 MPLINK 目标链接器中。欲知更多有关汇编器的信息，请参见 MPASM 汇编器文档。

C 语言源文件 `main.c` 也与关联的 MPLAB C18 C 编译器同时显示。MPLAB IDE 将使用此信息生成目标文件 (`main.o`)，此文件将输入到 MPLINK 目标链接器中。欲知有关编译器的更多信息，请参见推荐读物所列出的 MPLAB C18 C 编译器文档。

此外，项目中还可能包括预编译的目标文件 (`precomp.o`)，而无需任何关联工具。例如，MPLAB C18 要求包含预编译的标准代码模块 `c018i.o`。欲知更多有关可用的 Microchip 预编译目标文件的信息，请参见 MPLAB C18 C 编译器文档。

编译器中附带有某些库文件 (`math.lib`)。其他库文件可以使用库管理器工具 (MPLIB 目标库管理器) 编译。欲知更多有关库管理器的信息，请参见 MPLIB 库管理器文档。欲知更多有关可用的 Microchip 库的信息，请参见 MPLAB C18 C 编译器文档。

通过链接器 (MPLINK 目标链接器)，用目标文件和库文件以及链接描述文件 (`device.lkr`) 生成项目输出文件。欲知更多有关链接描述文件和使用链接器的信息，请参见 MPLINK 链接器文档。

MPLINK 链接器生成的主要输出文件是 **hex 文件** (`prog.hex`)，模拟器、仿真器、调试器和编程器都使用这些文件。欲知更多有关链接器输出文件的信息，请参见 MPLINK 链接器文档。

欲知更多有关项目和相关的工作区信息，请参见 MPLAB IDE 文档。

项目建立

第一次建立 MPLAB IDE 项目时，建议使用内置的 Project Wizard (*Project>Project Wizard*)。在此向导中，可以选择使用 MPASM 汇编器的语言工具套件，例如 Microchip MPASM 工具套件。欲知更多有关向导和 MPLAB IDE 项目的信息，请参见 MPLAB IDE 文档。

一旦建立了项目，接下来就要在 MPLAB IDE 中设置工具的属性。

- 1. 在 MPLAB IDE 菜单栏中，选择 *Project>Build Options>Project* 打开对话框以设置 / 检查项目编译选项。

注： MPASM 汇编器不能识别在 MPLAB IDE 中指定的包含路径信息。

- 2. 单击 **Tool** 选项卡修改工具设置。
 - Build Options 对话框， MPASM Assembler 选项卡
 - Build Options 对话框， MPLINK Linker 选项卡
 - Build Options 对话框， MPASM/C17/C18 Suite 选项卡

Build Options 对话框， MPASM Assembler 选项卡

选择一个类别，然后设置汇编器选项。其他选项请参见第 2 章 “汇编器界面”。

通用类别

产生命令行	
禁止区分大小写	汇编器不会区分字母的大小写。 注：禁止区分大小写选项将使所有标号变为大写。
扩展模式	使能 PIC18F 扩展指令支持。
默认进制	设置默认的进制，可以是十六进制、十进制或八进制。
宏定义	添加宏伪指令定义。
恢复默认设置	恢复选项卡的默认设置。
使用备用设置	
文本框	在命令行（非 GUI）格式中输入选项。

输出类别

产生命令行	
诊断级别	选择仅显示错误；错误和警告；或错误、警告和消息。Output 窗口中将显示这些内容。
生成交叉引用文件	创建交叉引用文件。交叉引用文件包含在汇编代码中使用的所有符号的列表。
Hex 文件格式 (用于单个文件的汇编)	在汇编单个文件时，将使用汇编器来生成 hex 文件。在此选择格式。 在汇编多个文件时，汇编器将生成目标文件，这些目标文件必须与链接器相链接才能生成 hex 文件。在这种情况下，为链接器选择 hex 文件格式。
恢复默认设置	恢复选项卡的默认设置。
使用备用设置	
文本框	在命令行（非 GUI）格式中输入选项。

Build Options 对话框，MPLINK Linker 选项卡

选择一个类别，然后设置链接器选项。其他选项请参见第 10 章“链接器接口”。

所有选项类别

产生命令行	
Hex 文件格式	选择链接器 hex 文件格式或禁止 hex 文件的输出。
生成映射文件	创建映射文件。映射文件在最终输出中提供有关源代码符号的绝对位置的信息。它也提供存储器使用的信息，指明已用的 / 未用的存储器。
禁止 COD 文件的生成	不生成 COD 文件。 注： COD 文件名（包括路径）最多只能有 62 个字符。COFF 文件则没有此限制。 注： 这也将导致不生成链接器列表文件。这不会影响汇编器列表文件的生成。
输出文件根目录	输入保存输出文件的根目录。
恢复默认设置	恢复选项卡的默认设置。
使用备用设置	
文本框	在命令行（非 GUI）格式中输入选项。

Build Options 对话框，MPASM/C17/C18 Suite 选项卡

确定是使用链接器将项目中的文件编译为正常输出（hex 文件等）还是使用 MPLIB 库管理器将它们编译为库（lib 文件）。

项目示例

在此示例中，您将用多个汇编文件创建 MPLAB IDE 项目。因此，您将需要使用 MPASM 汇编器和 MPLINK 链接器来创建最终输出的可执行（.hex）文件。

- 运行项目向导
- 设置编译选项
- 编译项目
- 编译错误
- 输出文件
- 进一步开发
- 源代码列表

运行项目向导

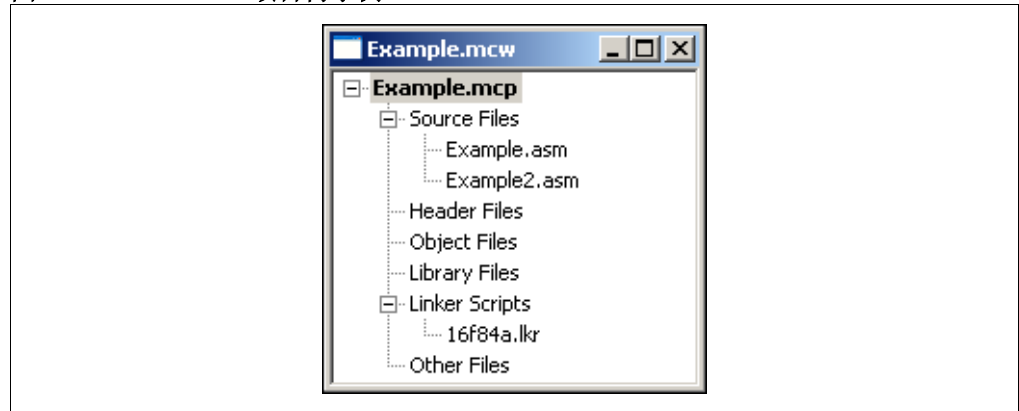
在 MPLAB IDE 中，选择 **Project>Project Wizard** 启动向导。在欢迎屏幕上单击 **Next>**。

1. 在 Device 中选择 PIC16F84A。单击 **Next>** 继续。
2. 如果还没有设置语言工具的话，设置语言工具。请参见“MPLAB IDE 的设置”。单击 **Next>** 继续。
3. 输入“Example”作为项目的名称。然后浏览并选择项目的位置。单击 **Next>** 继续。

4. 给项目添加文件。
 - a) 在对话框左边列出文件的框中找到以下目录：
C:\Program Files\Microchip\MPASM Suite\EXAMPLE
选择 Example.asm 和 Example2.asm。单击 **Add>>** 将这些文件添加到项目中。
 - b) 在对话框左边列出文件的框中找到以下目录：
C:\Program Files\Microchip\MPASM Suite\LKR
选择 16f84a.lkr。单击 **Add>>** 将此文件添加到项目中。
 - c) 勾选每个文件旁边的复选框将每个文件复制到项目目录中（这将保存原始文件）。单击 **Next>** 继续。
5. 复查信息摘要。如果发生了错误，使用 **<Back** 返回并更正错误的项。单击 **Finish** 完成项目创建和设置。

一旦 Project Wizard 完成，Project 窗口就会包含项目树。工作区名称为 Example.mcw，项目名称为 Example.mcp，并且所有项目文件都被列在其各自的文件类型下面。欲知更多有关项目和工作区的信息，请参见 MPLAB IDE 文档。

图 3： 项目树示例



设置编译选项

选择 **Project>Build Options>Project** 打开 Build Options 对话框。

1. 单击 **MPASM Assembler** 选项卡。对于 “Categories: General”，选择将 “Default Radix” 设置为 “Hexadecimal”。对于 “Categories: Output”，选择 “Diagnostics level” 为包含所有错误、警告和消息。然后勾选 “Generate cross-reference file” 复选框。
2. 单击 **MPLINK Linker** 选项卡。对于 “Categories: (All Options)”，选择将 “Hex File Format” 设置为 “INHX32”。然后勾选 “Generate map file” 复选框。
3. 单击 **MPASM/C17/C18 Suite** 选项卡。对于 “Categories: (All Options)”，选定 “Build normal target (invoke MPLINK)”。
4. 在对话框底部单击 **OK** 接受编译选项并关闭对话框。
5. 选择 **Project>Save Project** 保存 Example 项目的当前配置。

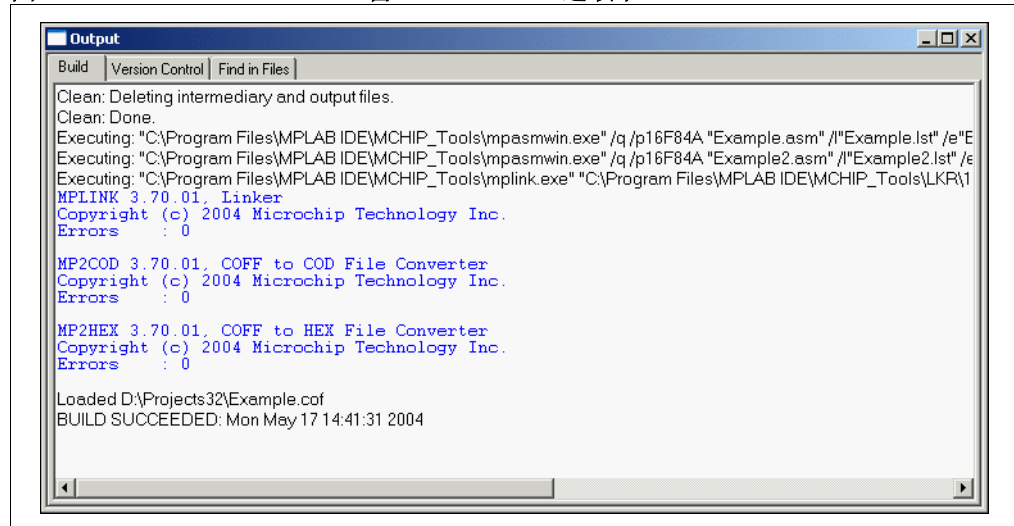
编译项目

选择 **Project>Build All** 编译项目。

注： 也可以在项目树中对准项目名称 “Example.mcp” 单击鼠标右键，并在弹出菜单中选择 “Build All”。

编译将要完成时会出现 Output 窗口并显示编译结果。

图 4： OUTPUT 窗口——BUILD 选项卡



编译错误

如果编译没有成功完成，请检查以下各项：

1. 复查此示例中前面的步骤。确定已经正确设置了语言工具并且所有项目文件和编译选项都正确。
2. 如果修改了样本源代码，请在 Output 窗口的 Build 选项卡中检查源代码中是否有语法错误。如果有的话，在错误上双击进入包含该错误的源代码行。改正错误，然后尝试再次编译。

输出文件

在 MPLAB IDE 中打开文件查看项目输出文件。

1. 选择 **File>Open**。在 Open 对话框中找到项目目录。
2. 在 “Files of type” 下选择 “All files” (*.*) 查看所有项目文件。
3. 选择 “Example.xrf”。单击 **Open** 在 MPLAB IDE 编辑器窗口中查看 Example.asm 的汇编器交叉引用文件。欲知有关此文件的更多信息，请参见第 1.7.6 节 “交叉引用文件 (.xrf)”。
4. 重复步骤 1 和 2。选择 “Example.map”。单击 **Open** 在 MPLAB IDE 编辑器窗口中查看链接器映射文件。欲知有关此文件的更多信息，请参见第 9.7.8 节 “映射文件 (.map)”。
5. 重复步骤 1 和 2。选择 “Example.lst”。单击 **Open** 在 MPLAB IDE 编辑器窗口中查看链接器列表文件。当 MPASM 汇编器和 MPLINK 链接器一起使用时，列表文件由链接器生成。欲知有关此文件的更多信息，请参见第 9.7.7 节 “列表文件 (.lst)”。
6. 重复步骤 1 和 2。注意只有一个 hex 文件 “Example.hex”。这是各种调试工具所使用的基本输出文件。进行调试时并不查看此文件，而是使用 **View>Program Memory** 或 **View>Disassembly Listing**。

二次开发

通常，应用程序代码在编译时总是会有错误。因此，需要调试工具来帮助您开发代码。有几种可与 **MPLAB IDE** 配合使用的调试工具可帮助您进行调试，它们使用前面讨论的输出文件。您可选择 **Microchip Technology** 或第三方开发商制造的模拟器、在线仿真器或在线调试器。请参见这些工具的文档，了解它们如何帮助您开发代码。

一旦开发了代码，您就会希望将代码编程到器件中。同样有几种可与 **MPLAB IDE** 配合使用的编程器可帮助您将代码编程到器件中。请参见这些工具的文档，了解它们如何帮助您开发代码。

欲知更多有关使用 **MPLAB IDE** 的信息，请咨询此应用程序的在线帮助或者从我们的网站下载可打印的文档。

注:

第 1 部分——MPASM 汇编器

第 1 章	MPASM 汇编器概述	21
第 2 章	汇编器界面	31
第 3 章	表达式语法和运算法则	39
第 4 章	伪指令	45
第 5 章	汇编器示例、技巧和窍门	123
第 6 章	可重定位目标	143
第 7 章	宏语言	153
第 8 章	错误、警告、消息和限制	157

注:

第 1 章 MPASM 汇编器概述

1.1 简介

本章概述了 MPASM 汇编器及其功能。

本章涉及的主题：

- MPASM 汇编器的定义
- MPASM 汇编器为您提供的帮助
- 汇编器移植路径
- 汇编器兼容性问题
- 汇编器操作
- 汇编器输入 / 输出文件

1.2 MPASM 汇编器的定义

MPASM 汇编器（汇编器）是命令行或基于 Windows 的 PC 应用程序，它为 Microchip 的 PICmicro 单片机（MCU）系列提供了一个开发汇编语言代码的平台。

该汇编器有两种可执行版本：

- Windows 版本（mpasmwin.exe）。此版本可在 MPLAB IDE 中、在独立的 Windows 应用程序中或命令行上使用。MPLAB IDE 或 MPLAB C18 C 编译器的常规版本和演示版本有此汇编器版本。这是推荐的版本。
- 命令行版本（mpasm.exe）。在命令行上使用此版本，从命令 shell 界面运行或直接从命令行运行。MPLAB C18 C 编译器的常规版本和演示版本可以使用此版本。

MPASM 汇编器支持所有的 PICmicro MCU 器件，同时也支持来自 Microchip Technology Inc. 的存储器 and KeeLoq® 安全数据产品（MPLAB IDE v5.70.40 以后的版本不支持有些存储器和 KeeLoq 器件）。

1.3 MPASM 汇编器为您提供的帮助

MPASM 汇编器为所有 Microchip 的 PICmicro MCU 提供了开发汇编代码的通用解决方案。其重要特性有：

- 与 MPLAB IDE 兼容
- 命令行接口
- Windows/ 命令 shell 界面接口
- 丰富的伪指令语言
- 灵活的宏语言

1.4 汇编器移植路径

由于 MPASM 汇编器是所有 MCU 器件的通用汇编器，所以为 PIC16F877A 开发的应用程序代码可被转换为供 PIC18F452 使用的程序。这可能需要更改两种器件之间不同的指令助记符（假定使用的寄存器和外设相似）。其余的伪指令和宏语言相同。

1.5 汇编器兼容性问题

MPASM 汇编器与 MPLAB IDE 集成开发环境兼容（mpasmwin.exe 版本），并与当前批量生产的所有 Microchip PICmicro MCU 开发系统兼容。

MPASM 汇编器支持用一种有规则并且一致的方法来指定基数（参见第 3.4 节“**数字常数和基数**”）。建议您使用本文档中描述的基数和其他伪指令方法来进行开发，即使出于兼容性原因该汇编器也可能支持某些较老的语法。

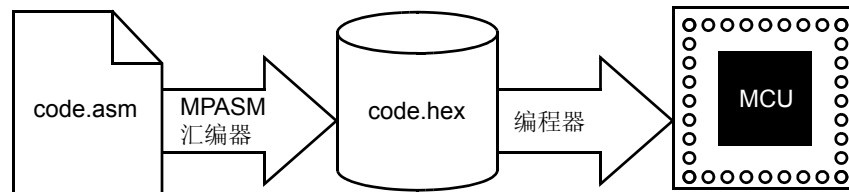
1.6 汇编器操作

MPASM 汇编器可以有两种使用方式：

- 生成可直接由单片机执行的**绝对代码**。
- 生成可与其他独立汇编或编译的模块链接的**可重定位代码**。

1.6.1 生成绝对代码

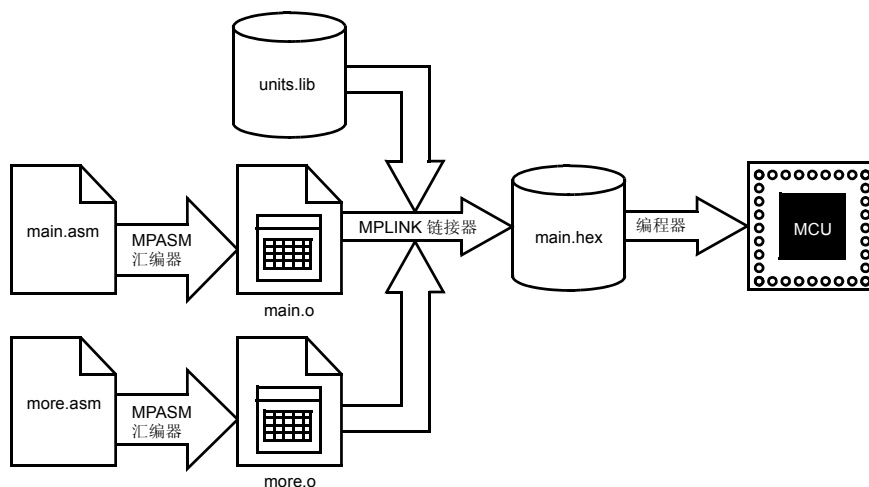
MPASM 汇编器的默认输出是绝对代码。此过程如下所示。



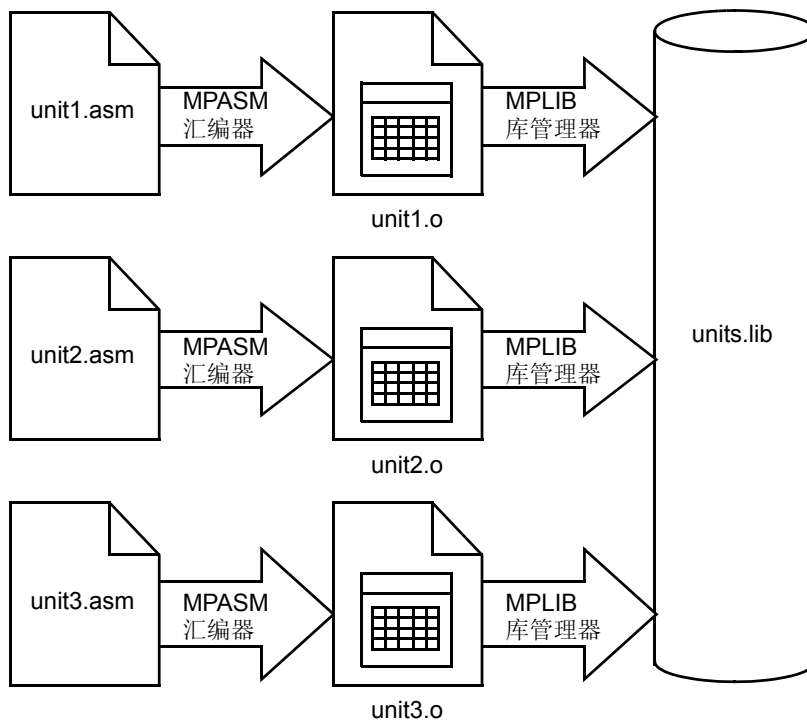
当以此方式汇编源文件时，在该源文件中使用的所有变量和程序必须在该源文件内定义，或者在已经被该源文件明确包含的文件中定义。如果汇编处理未发生错误，将生成一个 **hex** 文件，该文件包含目标器件的可执行机器码。之后，利用调试器可使用此文件进行代码测试，或利用器件编程器使用此代码进行单片机编程。

1.6.2 生成可重定位代码

MPASM 汇编器还可以生成可重定位目标模块，使用 Microchip 的 MPLINK 链接器，可将这些目标模块与其他模块相链接以组成最终可执行代码。此方法对于创建可重用的模块很有用。



使用 Microchip 的 MPLIB 库管理器可以将相关的模块放在一起并存储在一个库中。可以在链接时指定所需的库，并且只有需要的程序才会被包含在最终的可执行文件中。



如需了解有关绝对目标和可重定向目标汇编之间差异的信息，请参见第 6 章“可重定位目标”。

1.7 汇编器输入 / 输出文件

以下是供汇编器和关联实用程序函数使用的默认文件扩展名。

表 1-1: 输入文件

源代码 (.asm)	输入汇编器的默认源文件扩展名。
包含文件 (.inc)	包含 (头) 文件

表 1-2: 输出文件

列表文件 (.lst)	由汇编器生成的列表文件的默认输出扩展名。
错误文件 (.err)	汇编器的错误文件的输出扩展名。
Hex 文件格式 (.hex, .hxl 和 .hxx)	汇编器的 hex 文件的输出扩展名。
交叉引用文件 (.xrf)	汇编器的交叉引用文件的输出扩展名。
符号和调试文件 (.cod)	符号和调试文件的输出扩展名。 对于绝对代码, 此文件将由汇编器生成。 对于可重定位代码, 此文件和 .coff 文件将由 MPLINK 链接器生成。如需了解更多详情, 请参见 MPLINK 链接器文档。
目标文件 (.o)	汇编器的目标文件的输出扩展名。

1.7.1 源代码 (.asm)

汇编语言是一种可以用来为应用程序开发源代码的编程语言。可以使用任何 ASCII 文本文件编辑器来创建源代码文件。

注: MPLAB IDE 包含了几个免费的示例源代码文件。

源代码应该遵循下列基本指导方针。

源文件的每一行至多可以包含四种类型的信息:

- 标号
- 助记符、伪指令和宏
- 操作数
- 注释

这些信息的顺序和位置很重要。为了便于调试, 建议标号从第一列开始, 助记符从第二列或以后的列开始。操作数跟在助记符后面。注释可以跟在操作数、助记符或标号后, 并且可以从任何一列开始。最大列宽为 255 个字符。

必须用空白或冒号将标号和助记符号分开, 必须用空白将助记符和操作数分开。必须用逗号将多个操作数分开。

空白是指一个或多个空格或制表符。空白用来将源代码行分段。使用空白的目的是使代码便于他人阅读。除非在字符常数内部, 否则任何空白的意义与一个空格相同。

例 1-1: MPASM 汇编器绝对源代码（给出了多个操作数）

标号 ↓	助记符 伪指令 宏 ↓	操作数 ↓	注释 ↓
	list	p=18f452	
	#include	p18f452.inc	
Dest	equ	0x0B	;Define constant
	org	0x0000	;Reset vector
	goto	Start	
	org	0x0020	;Begin program
Start			
	movlw	0x0A	
	movwf	Dest	
	bcf	Dest, 3	;This line uses 2 operands
	goto	Start	
	end		

1.7.1.1 标号

标号用来表示一行、一组代码或一个常数值。跳转指令需要它（例 1-1）。

标号应该从第一列开始。后面可以跟冒号（:）、空格、制表符或行尾。标号必须以一个字母字符或一个下划线（_）开头，可以包含字母数字字符、下划线和问号。

标号不能：

- 以两个前导下划线开头，例如：__config。
- 以一个前导下划线和一个数字开头，例如：_2NDLOOP。
- 是汇编器的保留字（参见第 3.3 节“保留字和段名”）。

标号的最大长度为 32 字符。默认情况下，它们是区分大小写的，但是通过命令行选项（/c）可以忽略大小写。如果在定义标号时使用冒号，冒号将被视作标号运算符而非标号自身的一部分。

1.7.1.2 助记符、伪指令和宏

助记符告诉汇编器对哪些机器指令进行汇编。例如，加（add）、跳转（goto）或移动（movwf）。与您自己创建的标号不同，助记符由汇编语言提供。助记符不区分大小写。

伪指令是出现在源代码中的汇编器命令，但是通常不被直接编译为操作码。它们用于控制汇编器的输入、输出和数据分配。伪指令不区分大小写。

宏是用户定义的一组指令和伪指令，每当调用宏时，这些指令和伪指令将嵌入汇编器源代码同一行执行。

汇编器指令助记符、伪指令和宏调用应该从第二列或以后的列开始。如果同一行有一个标号，必须用冒号或一个或多个空格或制表符将指令与该标号分开。

1.7.1.3 操作数

操作数给出有关指令的信息，包括应该使用的数据和该指令的存储单元。

必须用一个或多个空格或制表符将操作数与助记符分开。必须用逗号将多个操作数分开。

1.7.1.4 注释

注释是解释一行或数行代码的操作的文本。

MPASM 汇编器将分号后的任何文本视作注释。分号后直至行尾的所有字符均被忽略。包含分号的字符串常数是允许的，且不会与注释混淆。

1.7.2 包含文件 (.inc)

汇编器包含（头）文件是指任何包含有效汇编代码的文件。通常，该文件包含特定器件的寄存器和位分配。该文件可以被“包含”在代码中，从而可以被很多程序重复使用。

例如，如果要将 PIC18F452 器件的标准头文件添加到汇编代码中，则使用：

```
#include p18f452.inc
```

标准头文件位于：

```
C:\Program Files\Microchip\MPASM Suite
```

1.7.3 列表文件 (.lst)

MPASM 汇编器列表文件提供了到目标代码的源代码映射。它还提供了一个包括符号值、存储器使用情况以及产生的错误、警告和消息数量的列表。可以在 MPLAB IDE 中查看此文件，方法是：

1. 选择 **File>Open** 启动 Open 对话框。
2. 从“Files of type”下拉列表中选择“List files (*.lst)”
3. 查找所需的列表文件
4. 单击列表文件名
5. 单击 **Open**。

MPASM 汇编器和 MPLINK 链接器都能生成列表文件。欲知有关 MPLINK 链接器列表文件的信息，请参见第 9.7.7 节“列表文件 (.lst)”。

要防止生成汇编器列表文件，请使用 /l- 选项或 MPLINK 链接器（链接器列表文件将覆盖汇编器列表文件）。使用 /t 选项设置列表文件中制表符的大小。

例 1-2: MPASM 汇编器绝对列表文件

产品名称和版本、汇编日期和时间，以及页号出现在每页的顶部。

第一列包含了存储器中将存放该代码的基地址。第二列显示了用 set、equ、variable、constant 或 cblock 伪指令创建的任何符号的 32 位值。第三列保留用以存放机器指令。这是将被 PICmicro MCU 执行的代码。第四列列出了与这行关联的源文件行号。这一行的其余部分保留用以存放生成机器代码的源代码行。

错误、警告和消息嵌入在源码行之间，并且与下一源码行相关。同样地，在列表末尾也有一个摘要。

符号表列出了在该程序中定义的所有符号。

存储器使用情况映射图用图形表示存储器的使用情况。“X”表示已用的单元，“-”表示对象还未使用的存储区。该图还显示了程序存储器的使用情况。如果生成目标文件，就不会输出存储器映射图。

注： 由于页宽限制，将一些注释缩短，用“..”表示。另外，还删除了一些符号表，以“:”表示。欲知完整的符号清单，请查看标准头文件 p18f452.inc。

MPASM 03.70 Released

SOURCE.ASM 4-5-2004 15:40:00

PAGE 1

LOC	OBJECT CODE	LINE	SOURCE TEXT
	VALUE		
		00001	list p=18f452
		00002	#include p18f452.inc
		00001	LIST
		00002	; P18F452.INC Standard Header File, Version 1.4..
		00845	LIST
0000000B		00003	Dest equ 0x0B
		00004	
000000		00005	org 0x0000
000000 EF10 F000		00006	goto Start
000020		00007	org 0x0020
000020 0E0A		00008	Start movlw 0x0A
000022 6E0B		00009	movwf Dest
000024 960B		00010	bcf Dest, 3 ;This line uses 2 op..
000026 EF10 F000		00011	goto Start
		00012	end

MPASM 03.70 Released

SOURCE.ASM 4-5-2004 15:40:00

PAGE 2

SYMBOL TABLE	
LABEL	VALUE
A	00000000
ACCESS	00000000
:	:
_XT_OSC_1H	000000F9
__18F452	00000001

MPASM 03.70 Released

SOURCE.ASM 4-5-2004 15:40:00

PAGE 12

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

0000 : XXXX----- XXXXXXXXXXXX-----

All other memory blocks unused.

Program Memory Bytes Used: 14

Program Memory Bytes Free: 32754

Errors : 0
Warnings : 0 reported, 0 suppressed
Messages : 0 reported, 0 suppressed

1.7.4 错误文件 (.err)

默认情况下，MPASM 汇编器会生成错误文件。在调试代码时，此文件很有用。MPLAB IDE 将在 Output 窗口中显示错误信息。错误文件中的消息格式是：

类型[编号] 文件行描述

例如：

Error[113] C:\PROG.ASM 7 : Symbol not previously defined (start)

错误文件可以包含任何数量的 MPASM 汇编器错误、警告和消息。欲知更多有关这些的信息，请参见第 8 章“错误、警告、消息和限制”。

要阻止生成错误文件，请使用 /e- 选项。

1.7.5 Hex 文件格式 (.hex, .hxl 和 .hxx)

MPASM 汇编器和 MPLINK 链接器能够生成不同格式的 ASCII 文本 hex 文件。

格式名称	格式类型	文件扩展名	用途
Intel Hex 格式	INHX8M	.hex	8 位内核的器件编程器
Intel Split Hex 格式	INHX8S	.hxl 和 .hxx	奇 / 偶编程器
Intel Hex 32 格式	INHX32	.hex	16 位内核的器件编程器

此文件格式用于将 PICmicro MCU 系列代码传送到 Microchip 编程器或第三方 PICmicro MCU 编程器。

1.7.5.1 INTEL HEX 格式

此格式将低字节、高字节组合生成 8 位 hex 文件。由于在这种格式下每个地址只能包含 8 位，所以所有的地址都是两个 8 位地址的组合。

每个数据记录以 8 个字符的前缀开头，以 2 个字符的校验和结束。每个记录的格式如下：

:BBAAAATTHHHH...HHHCC

其中：

- BB 两位十六进制字节计数，表示将出现在该行上的数据字节数。
- AAAA 四位十六进制地址，表示数据记录的起始地址。
- TT 两位记录类型，除了文件末尾记录（为“01”）外，总是“00”。
- HH 两位十六进制数据字节，以低字节 / 高字节组合形式给出。
- CC 两位十六进制校验和，它是该记录中前面所有字节之和的 2 的补码（Two's Complement）。

例 1-3: INHX8M

```
file_name.hex
:100000000000000000000000000000000000F0
:040010000000000000EC
:100032000000280040006800A800E800C80028016D
:100042006801A9018901EA01280208026A02BF02C5
:10005200E002E80228036803BF03E803C8030804B8
:1000620008040804030443050306E807E807FF0839
:06007200FF08FF08190A57
:00000001FF
```

1.7.5.2 INTEL SPLIT HEX 格式

拆分的 8 位文件格式生成两个输出文件: .hxl 和 .hxx。除了数据字的低字节存储在 .hxl 文件中, 数据字的高字节存储在 .hxx 文件中以及地址被一分为二之外, 这种格式与 8 位格式是相同。这种格式用于将 16 位字编程到成对的 8 位 EPROM 中, 一个文件保存低字节, 一个文件保存高字节。

例 1-4: INHX8S

```
file_name.hxl
:0A000000000000000000000000000000F6
:1000190000284068A8E8C82868A989EA28086ABFAA
:10002900E0E82868BFE8C8080808034303E8E8FFD0
:03003900FFFF19AD
:00000001FF
file_name.hxx
:0A000000000000000000000000000000F6
:100019000000000000000000101010101020202CA
:100029000202030303030304040404050607070883
:0300390008080AAA
:00000001FF
```

1.7.5.3 INTEL HEX 32 格式

除了扩展的线性地址记录也被输出用来建立数据地址的高 16 位之外, 扩展的 32 位地址 hex 格式与 hex 8 格式相似。这种格式主要用于 16 位内核器件, 因为它们的可寻址程序存储器空间超过了 64 KB。

每个数据记录以一个 8 个字符的前缀开头, 以一个 2 字符的校验和结束。每个记录的格式如下:

```
:BBAAATTHHHH...HHHCC
```

其中:

BB	两位十六进制字节计数, 表示将出现在该行上的数据字节数。
AAAA	四位十六进制地址, 表示数据记录的起始地址。
TT	两位记录类型: 00 —— 数据记录 01 —— 文件末尾记录 02 —— 段地址记录 04 —— 线性地址记录
HH	两位十六进制数据字节, 以低字节 / 高字节组合形式给出。
CC	两位十六进制校验和, 它是该记录中前面所有字节之和的 2 的补码。

1.7.6 交叉引用文件 (.xrf)

交叉引用文件包含在汇编代码中使用的所有符号的列表。该文件具有如下格式：

- 这些符号按照名称排序，在 “Label” 列中列出。
- “Type” 列定义符号的类型。在该文件末尾提供了一个 “Label Type” 列表。
- “File Name” 列列出了使用该符号的文件的名称。
- “Source File Reference” 列列出了在 “File Name” 列中对应的文件中定义 / 引用该符号的行号。星号表示定义。

要阻止生成交叉引用文件，请使用 /x- 选项。

1.7.7 符号和调试文件 (.cod)

MPLAB IDE 使用 COD 文件来调试绝对汇编代码。

对于生成绝对代码，MPASM 汇编器生成一个 .cod 文件用于调试。

对于生成可重定位代码，MPASM 汇编器和 MPLINK 链接器一同使用，并且链接器生成 .cod 和 .coff 文件用于调试。

注： 带路径的 COD 文件名不得超过 62 个字符。COFF 文件名无此限制。

使用链接器时如果要阻止生成 COD 文件，进行以下操作之一：

- 在命令行上使用 /w 选项。
- 在 MPLAB IDE 中，在 Build Options 对话框 ([*Project>Build Options>Project*](#)) 的 **MPLINK Linker** 选项卡上选择 “Suppress COD file generation”。

1.7.8 目标文件 (.o)

汇编器从源代码创建一个可重定位的目标文件。此目标文件尚未解析地址，并且在用作可执行文件前必须进行链接。

要生成一个在编程到器件之后执行的文件，请参见第 1.7.5 节 “Hex 文件格式 (.hex, .hxl 和 .hxx)”。

要阻止生成目标文件，请使用 /o- 选项。

第 2 章 汇编器界面

2.1 简介

根据汇编器版本的不同，MPASM 汇编器可能有几界面可供使用。本章将讨论这些界面。

在安装 MPLAB IDE 的同时也将安装视窗版的 MPASM 汇编器 (mpasmwin.exe)。此外，视窗版和命令行版 (mpasm.exe) 汇编器可随常规版或试用版 MPLAB C18 编译器一起提供。

本章涉及以下主题：

- MPLAB IDE 界面
- Windows 界面
- 命令 shell 界面
- 命令行接口
- 疑难解答

2.2 MPLAB IDE 界面

MPASM 通常在 MPLAB IDE 项目中与 MPLINK 链接器一起使用以生成可重定位的代码。参见“**PICmicro 语言工具和 MPLAB IDE**”了解更多信息。

汇编器也可在 MPLAB IDE 中通过使用快速编译功能生成绝对代码（不使用 MPLINK 链接器或 MPLAB IDE 项目）。操作步骤如下：

1. 从 MPLAB IDE 菜单栏中选择 **Project>Set Language Tool Locations**，打开对话框设置或检查语言工具可执行文件的位置。
2. 在对话框的“Registered Tools”中选择“Microchip MPASM Toolsuite”。单击“+”展开。
3. 选择“Executables”。单击“+”展开。
4. 选择 MPASM Assembler (mpasmwin.exe)。在 Location 下应显示 mpasmwin.exe 文件的路径。如果没有显示路径，则输入文件路径或浏览此文件的位置。在默认情况下，它的位置是：
C:\Program Files\Microchip\MPASM Suite\mpasmwin.exe
5. 单击 **OK**
6. 从 MPLAB IDE 菜单栏中选择 **Project>Quickbuild**，使用 MPASM 汇编器汇编指定的 asm 文件。

2.3 WINDOWS 界面

MPASM 汇编器的 Windows 版提供了用于设置汇编器选项的图形界面。在 Windows 资源管理器中执行 mpasmwin.exe 或输入命令提示符启动汇编器。

图 2-1: MPASM™ 汇编器 WINDOWS SHELL 界面



通过输入文件名或使用 **Browse** 按钮选择源文件。按照下面的描述设置各个选项。（默认选项从源文件中读出。）然后单击 **Assemble** 汇编源文件。

注： 当通过 MPLAB IDE 启动 Windows 版 MPASM 汇编器时，不会显示该选项屏幕。使用 MPLAB IDE 的 Build Options 对话框中的 MPASM Assembler 选项卡（*Project>Build Options>Project*）来设置选项。

选项	说明
Radix	改写任何源文件的基数设置。 参见第 4.42 节 “list——列表选项”，第 4.55 节 “radix——指定默认基数” 和第 3.4 节 “数字常数和基数”
Warning Level	改写任何源文件的消息级别设置。 参见第 4.47 节 “messg——创建用户定义的消息”
Hex Output	改写任何源文件 hex 文件的格式设置。 参见第 1.7.5 节 “Hex 文件格式 (.hex, .hxl 和 .hxx)”
Generated Files	使能 / 禁止各种输出文件。 参见第 1.7 节 “汇编器输入 / 输出文件”
Case Sensitivity	使能 / 禁止区分大小写。如果使能，汇编器会区分字母的大小写。
Tab Size	设置列表文件的制表符的大小。 参见第 1.7.3 节 “列表文件 (.lst)”
Macro Expansion	改写任何源文件的宏扩展设置。 参见第 4.31 节 “expand——扩展宏列表”

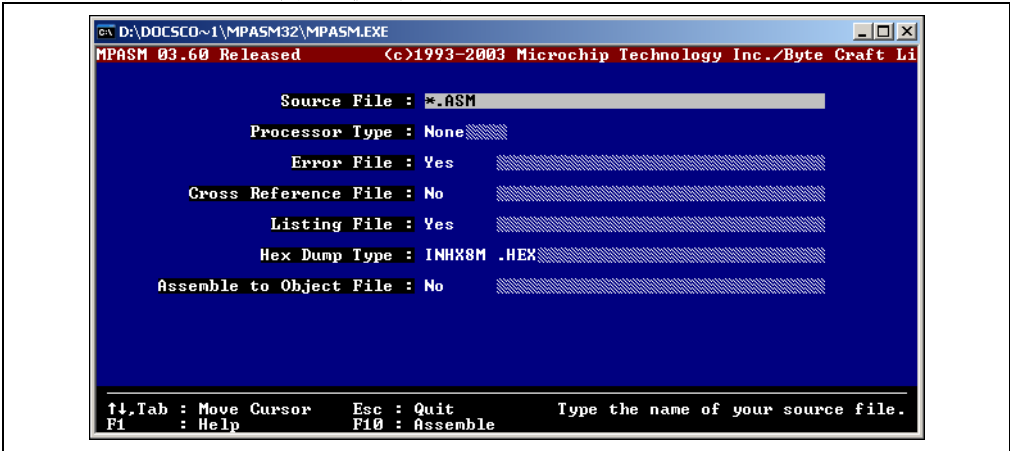
选项	说明
Processor	改写任何源文件的处理器设置。
Extended Mode	使能 PIC18F 扩展指令支持。
Extra Options	任何其他命令行选项。 参见第 2.5 节 “命令行接口”
Save Settings on Exit	将这些设置保存到 mplab.ini。当下一次运行 mpasmwin.exe 时会使用这些设置。

2.4 命令 SHELL 界面

MPASM 汇编器命令 shell 界面是以文本图形模式显示的屏幕。执行 Windows 资源管理器中的 mpasm.exe 启动汇编器。

在该屏幕中您可输入要汇编的源文件的名称和其他选项。

图 2-2: 文本图形模式显示



在 “Source File” 后输入源文件名。文件名可包括 DOS 路径和通配符。如果使用通配符（* 或 ?），就会列出所有符合条件的文件以供选择。要在该字段内自动输入 *.ASM，按 <Tab> 键即可。

参见第 1.7.1 节 “源代码（.asm）” 了解更多关于该文件类型的信息。

按照下面的描述设置各个选项。

选项	说明
Processor Type	如果在源文件中未指定处理器，使用该字段选择处理器。使用方向键选定该字段，然后通过按 <Enter> 键在所支持的处理器之间切换。
Error File	默认情况下创建一个错误文件（sourcename.err）。参见第 1.7.4 节 “错误文件（.err）” 了解更多关于该文件类型的信息。要关闭错误文件，使用方向键移动到该字段，然后按 <Enter> 键把它改为 “No”。通过按 <Tab> 键移动到阴影区域然后输入新文件名，就能更改错误文件名。在错误文件名中不允许使用通配符。

选项	说明
Cross Reference File	默认情况下不生成交叉引用文件（ <i>sourcename.xrf</i> ）。参见第 1.7.6 节“交叉引用文件（.xrf）”了解更多关于该文件类型的信息。 要创建交叉引用文件，使用方向键移动到该字段，然后按 <Enter> 把它改为“ Yes ”。通过按 <Tab> 键移动到阴影区域然后输入新文件名，就能改变交叉引用文件的名称。在交叉引用文件名中不允许使用通配符。
Listing File	默认情况下创建一个列表文件（ <i>sourcename.lst</i> ）。参见第 1.7.3 节“列表文件（.lst）”了解更多关于该文件类型的信息。 要关闭列表文件，使用方向键移动到该字段，然后按 <Enter> 键把它改为“ No ”。通过按 <Tab> 键移动到阴影区域然后输入新文件名，就能改变列表文件的名称。在列表文件名中不允许使用通配符。
Hex Dump Type	设置该值以生成所需的 hex 文件格式。参见第 1.7.5 节“Hex 文件格式（.hex, .hxl 和 .hxx）”了解更多关于该格式的信息。 用方向键将光标移动到该字段，然后按 <Enter> 键在所提供的选项间滚动可以改变该值。要改变 hex 文件名，按 <Tab> 键移动到阴影区域然后输入新文件名。
Assemble to Object File	使能该选项将生成可重定位的并能输入到链接器的目标代码，并能阻止 hex 文件的生成。参见第 1.7.8 节“目标文件（.o）”了解更多关于该文件类型的信息。 要打开目标文件，使用方向键移动到该字段，然后按 <Enter> 键把它改为“ Yes ”。通过按 <Tab> 键移动到阴影区域然后输入新文件名，就能改变目标文件名。在目标文件名中不允许使用通配符。

2.5 命令行接口

通过如下的命令行接口（命令提示）可启动 MPASM 汇编器：

```
mpasmwin [/option1.../optionN] filename
```

或

```
mpasm [/option1.../optionN] filename
```

其中

/option——表示命令行选项之一

filename——表示要汇编的文件

例如，如果 test.asm 存在于当前的目录中，可以用下面的命令进行汇编：

```
mpasmwin /e /l test.asm
```

如果省略源文件名，将会调用相应的 shell 界面，即

- mpasmwin——显示 Windows 界面，包括一个帮助按钮
- mpasm——显示一个交互式文本界面（同 mpasm /?）

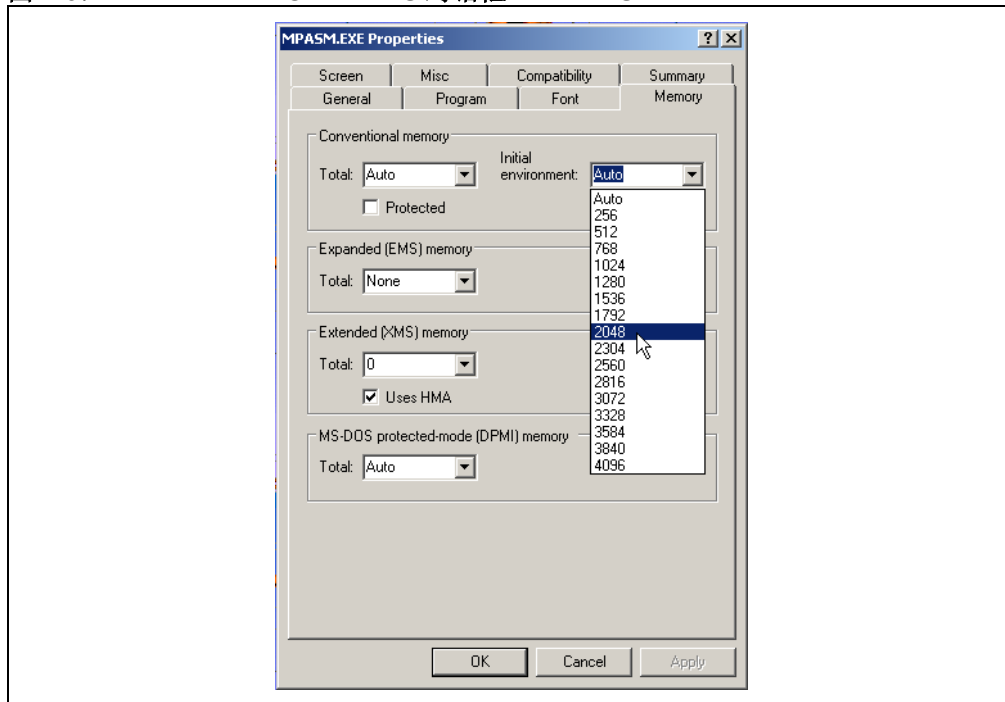
选项	默认	说明
/?	N/A	显示汇编器帮助屏幕（仅 mpasm.exe）。
/ahex-format	INHX32*	从汇编器直接生成绝对 .cod 和 .hex 输出，其中 <i>hex-format</i> 为 {INHX8M INHX8S INHX32} 之一。 参见第 1.7.5 节“Hex 文件格式 (.hex, .hxl 和 .hxx)”了解更多信息。
/c	启用	使能 / 禁止区分大小写。如果使能，汇编器会区分字母的大小写。
/dlabel[=value]	N/A	定义文本串替换，即将 <i>value</i> 指派给 <i>label</i> 。
/e[+ - path]	启用	使能 / 禁止 / 设置错误文件的路径。 /e 使能 /e+ 使能 /e- 禁止 /e path 使能 / 指定路径 参见第 1.7.4 节“错误文件 (.err)”了解更多信息。
/h	N/A	显示汇编器帮助屏幕（仅 mpasm.exe）。
/l[+ - path]	启用	使能 / 禁止 / 设置列表文件的路径。 /l 使能 /l+ 使能 /l- 禁止 /l path 使能 / 指定路径 参见第 1.7.3 节“列表文件 (.lst)”了解更多信息。
/m[+ -]	启用	使能 / 禁止宏扩展。 参见第 4.31 节“expand——扩展宏列表”了解更多信息。

选项	默认	说明
/o[+ - path]	禁用	使能 / 禁止 / 设置目标文件的路径。 /o 使能 /o+ 使能 /o- 禁止 /o path 使能 / 指定路径 参见第 1.7.8 节“目标文件 (.o)”了解更多信息。
/pprocessor_type	无	设置处理器类型，其中 processor_type 指 PICmicro® MCU 器件（如 PIC18F452）。
/q[+ -]	禁用	使能 / 禁止安静模式（禁止屏幕输出）。
/rradix	Hex	定义默认的基数，其中 radix 为 {HEX DEC OCT} 之一。 参见第 4.42 节“list——列表选项”或第 4.55 节“radix——指定默认基数”了解更多信息。
/t	8	设置列表文件中制表符的大小。 参见第 1.7.3 节“列表文件 (.lst)”了解更多信息。
/wvalue	0	设置消息级别，其中 value 为 {0 1 2} 之一。 0 所有消息 1 错误和警告 2 仅错误 参见第 4.47 节“messg——创建用户定义的消息”了解更多信息。
/x[+ - path]	禁用	使能 / 禁止 / 设置交叉引用文件的路径。 /x 使能 /x+ 使能 /x- 禁止 /x path 使能 / 指定路径 参见第 1.7.6 节“交叉引用文件 (.xrf)”了解更多信息。
/y[+ -]	禁止	使能 / 禁止扩展指令集。 /y 使能 /y+ 使能 /y- 禁止 只能使能用于支持扩展指令集的处理器和 PIC18CXXX 通用处理器。/y- 会覆盖 LIST PE=type 伪指令（参见第 4.42 节“list——列表选项”）。
* 默认值取决于选定的处理器。		

2.6 疑难解答

如果在使用 `mpasm.exe` 时得到一条消息，说环境空间不够，请使用 Microsoft Windows Internet Explorer 在 MPLAB IDE 安装目录下选择 `mpasm.exe`，然后单击右键打开 Properties 对话框。

图 2-3: PROPERTIES 对话框——MPASM.EXE



增加初始环境的大小。通常将环境空间设置为 2048 已经足够了，不过如果有许多应用程序，它们要设置变量并添加到 `AUTOEXEC.BAT` 文件的路径语句中，那么就可能需要更大的空间。

注:

第 3 章 表达式语法和运算法则

3.1 简介

本章介绍 MPASM 汇编器使用的各种表达式格式、语法和运算法则。

本章涉及以下主题：

- 文本字符串
- 保留字和段名
- 数字常数和基数
- 算术运算符和优先级

3.2 文本字符串

“字符串”是用双引号引出的任何有效的 ASCII 字符（为 0 到 127 之间的十进制数）序列。可能包括引号或空字符。

在字符串中添加特殊字符的方法是在这些特殊字符前用反斜杠 ‘\’ 进行转义。应用于字符串的转义序列同样也适用于字符。

字符串可以任意长，只要不超过 255 列的源代码行即可。如果找到相应的后引号，则表明字符串结束。如果在行末前仍未找到相应的后引号，字符串将在行尾处结束。虽然不可以将字符串直接延伸到第二行，但通常可以在下一行再次使用 dw 伪指令来达到这一目的。

dw 伪指令将把整个字符串存储到连续的字单元中。如果字符串字符（字节）数为奇数，dw 和 data 伪指令将在字符串尾填充一个字节的 0（00）。

如果字符串用作立即操作数，它必须为一个字符长，否则就会出错。

3.2.1 转义字符

汇编器接受 ANSI C 转义序列以表示某些特殊的控制字符：

表 3-1: ANSI C 转义序列

转义 字符	说明	十六进制 值
\a	报警字符	07
\b	后退字符	08
\f	换页符	0C
\n	换行符	0A
\r	回车符	0D
\t	水平制表符	09
\v	垂直制表符	0B
\\	反斜杠	5C
\?	问号字符	3F
\'	单引号（撇号）	27
\"	双引号	22
\000	八进制数（0，八进制数字，八进制数字）	
\xHH	十六进制数	

3.2.2 代码示例

参见下面由包含字符串的不同语句生成的目标代码示例。

```
7465 7374 696E      dw  "testing output string one\n"
6720 6F75 7470
7574 2073 7472
696E 6720 6F6E
650A
                                #define  str  "testing output string two"
B061                                movlw  "a"
7465 7374 696E      data  "testing first output string"
6720 6669 7273
7420 6F75 7470
7574 2073 7472
696E 6700
```


3.3 保留字和段名

不能用下面的词作为标号、常数或变量名：

- 伪指令（参见第 4 章“伪指令”）。
- 指令（参见附录 A“指令集”）。

此外，汇编器还保留了以下段名：

表 3-2: 保留的段名

段名	用途
<code>.access_ovr</code>	<code>access_ovr</code> 伪指令的默认段名。
<code>.code</code>	<code>code</code> 伪指令的默认段名。
<code>.idata</code> <code>.idata_acs</code>	分别为 <code>idata</code> 和 <code>idata_acs</code> 伪指令的默认段名。
<code>.udata</code> <code>.udata_acs</code> <code>.udata_ovr</code> <code>.udata_shr</code>	分别为 <code>udata</code> 、 <code>udata_acs</code> 、 <code>udata_ovr</code> 和 <code>udata_shr</code> 的默认段名。

3.4 数字常数和基数

MPASM 汇编器支持以下常数基数格式：十六进制、十进制、八进制、二进制和 ASCII。默认的基数是十六进制；当没有用基本描述符明确指定基数时，默认的基数将决定要给目标文件中的常数赋什么样的值。

注： 数字常数的基数可以与 `radix` 或 `list r=` 伪指令指定的默认基数不同。同样，允许使用的基数也仅限于十六进制、十进制和八进制。

常数前可以添加一个加号或减号。如果没有符号，数值将被假定为正数。

注： 常数表达式中的中间值作为 32 位无符号整数处理。每当试图将一个常数放入一个过小的字段时，就会发出截短警告。

下面的表格给出了各种基数的规范：

表 3-3: 基数规范——MPASM™ 汇编器 /MPLIN™ 链接器

注	类型	语法	示例
1	二进制	B' 二进制数字 '	B'00111001'
2	八进制	O' 八进制数字 '	O'777'
3	十进制	D' 十进制数字 ' . 数字	D'100' .100
4	十六进制	H' 十六进制数字 ' 十六进制数字	H'9f' 0x9f
5	ASCII	A' 字符 ' ' 字符 '	A'C' 'C'

- 注 1: 二进制整数为 'b' 或 'B' 后跟一个或多个用单引号括起的二进制数字 '01'。
- 2: 八进制整数为 'o' 或 'O' 后跟一个或多个用单引号括起的八进制数字 '01234567'。
- 3: 十进制整数为 'd' 或 'D' 后跟一个或多个用单引号括起的十进制数字 '0123456789'。或者，十进制整数为 '.' 后跟一个或多个十进制数字 '0123456789'。
- 4: 十六进制整数为 'h' 或 'H' 后跟一个或多个用单引号括起的十六进制数字 '0123456789abcdefABCDEF'。或者，十六进制整为 '0x' 后跟一个或多个十六进制数字 '0123456789abcdefABCDEF'。
- 5: ASCII 字符为 'a' 或 'A' 后跟一个用单引号括起的字符（见第 B.2 节 “ASCII 字符集”）。或者，ASCII 字符是一个单引号括起的字符。

3.5 算术运算符和优先级

算术运算符可与下表中规定的伪指令及其变量一起使用。

注: 这些运算符不能与程序变量一起使用。它们仅可与伪指令一起使用。

表中运算符的次序与其优先级相对应，其中第一个运算符优先级最高，最后一个运算符优先级最低。优先级指的是运算符在代码语句中的执行顺序。

表 3-4: 算术运算符（按优先级排列）

运算符		示例
\$	当前 / 返回程序计数器	goto \$ + 3
(左括号	1 + (d * 4)
)	右括号	(Length + 1) * 256
!	非（逻辑取反）	if ! (a == b)
-	取反（2 的补码）	-1 * Length
~	取补	flags = ~flags
low ¹	返回地址低字节	movlw low CTR_Table
high ¹	返回地址次高字节	movlw high CTR_Table
upper ¹	返回地址最高字节	movlw upper CTR_Table
*	乘	a = b * c
/	除	a = b / c
%	取余数	entry_len = tot_len % 16
+	加	tot_len = entry_len * 8 + 1
-	减	entry_len = (tot - 1) / 8
<<	左移	flags = flags << 1
>>	右移	flags = flags >> 1
>=	大于或等于	if entry_idx >= num_entries
>	大于	if entry_idx > num_entries
<	小于	if entry_idx < num_entries
<=	小于或等于	if entry_idx <= num_entries
==	等于	if entry_idx == num_entries
!=	不等于	if entry_idx != num_entries
&	位与	flags = flags & ERROR_BIT
^	位异或	flags = flags ^ ERROR_BIT
	位同或	flags = flags ERROR_BIT
&&	逻辑与	if (len == 512) && (b == c)
	逻辑或	if (len == 512) (b == c)
=	赋值	entry_index = 0
+=	加赋值	entry_index += 1
-=	减赋值	entry_index -= 1
*=	乘赋值	entry_index *= entry_length
/=	除赋值	entry_total /= entry_length
%=	取余数赋值	entry_index %= 8
<<=	左移赋值	flags <<= 3
>>=	右移赋值	flags >>= 3
&=	与赋值	flags &= ERROR_FLAG
=	同或赋值	flags = ERROR_FLAG
^=	异或赋值	flags ^= ERROR_FLAG
++	加一 (2)	i ++
--	减一 (2)	i--

- 注**
- 1: 应用于段的 low、high 和 upper 操作数的优先级相同。参见第 6.4 节“Low、High 和 Upper 操作数”了解更多信息。
 - 2: 这些运算符只能在一行中独立使用；不能嵌套在其他的表达式求值中。

注:

第 4 章 伪指令

4.1 简介

伪指令是出现在源代码中的汇编器命令，但是通常不被直接编译为操作码。它们用于控制汇编器的输入、输出和数据分配。

很多汇编器伪指令有备用的名称和格式。这些备用的名称和格式用于向后兼容 Microchip 的早期汇编器，以及符合个人的编程习惯。如果希望得到可移植的代码，建议按照此处提供的规范编写程序。

注： 虽然 MPLINK 目标链接器经常与 MPASM 汇编器配合使用，但是 MPLINK 链接描述文件并不支持 MPASM 汇编器伪指令。如需了解更多有关控制列表和 hex 文件输出的链接器选项的信息，请参见 MPLINK 目标链接器文档。

各个伪指令信息包括语法、说明、使用和相关的伪指令，以及简单或扩展（在有些情况下）的使用示例。在大多数情况下，可以通过添加 end 语句汇编和运行简单的示例。扩展示例可以经过汇编按“现状”运行以演示使用伪指令的应用程序。

可以按照字母顺序（在后面的章节中）或按类型排序（第 4.2 节“伪指令的类型”）找到各个伪指令。

注： 伪指令不区分大小写，例如，cblock 可以作为 CBLOCK、cblock 和 Cblock 等执行

4.2 伪指令的类型

汇编器提供了六种基本类型的伪指令。

1. 控制伪指令
2. 条件汇编伪指令
3. 数据伪指令
4. 列表伪指令
5. 宏伪指令
6. 目标文件伪指令

4.2.1 控制伪指令

控制伪指令控制如何汇编代码。

• #define——定义文本替换标号	p. 68
• #include——包含额外的源文件	p. 92
• #undef——删除替换标号	p. 117
• constant——声明符号常数	p. 60
• end——结束程序块	p. 71
• equ——定义一个汇编器常数	p. 74
• org——设置程序起始处	p. 100
• processor——设置处理器类型	p. 106
• radix——指定默认基数	p. 106
• set——定义汇编器变量	p. 109
• variable——声明符号变量	p. 118

4.2.2 条件汇编伪指令

条件汇编伪指令允许汇编符合条件的代码段。与 C 语言中相应的指令不同，这些不是运行时指令。它们定义汇编哪些代码，而不是如何执行代码。

• else——开始 if 条件的备用汇编块	p. 71
• endif——结束条件汇编块	p. 72
• endw——结束 while 循环	p. 73
• if——开始条件汇编代码块	p. 88
• ifdef——如果已经定义了符号则执行	p. 90
• ifndef——如果未定义符号则执行	p. 91
• while——当条件为 TRUE 时执行循环	p. 119

4.2.3 数据伪指令

数据伪指令控制对存储器进行分配，并提供了用符号（即有意义的名称）引用数据项的方法。

• __badram——标识未用的 RAM	p. 48
• __badrom——标识未用的 ROM	p. 49
• __config——设置处理器的配置位	p. 58
• config——设置处理器的配置位（PIC18 MCU）	p. 59
• __idlocs——设置处理器 ID 单元	p. 87
• __maxram——定义最大 RAM 单元	p. 97
• __maxrom——定义最大 ROM 单元	p. 98
• cblock——定义常数块	p. 54
• da——在程序存储器中存储字符串（PIC12/16 MCU）	p. 61
• data——创建数字和文本数据	p. 62
• db——声明一个字节的的数据	p. 65
• de——声明 EEPROM 数据字节	p. 67
• dt——定义表（PIC12/16 MCU）	p. 70
• dw——声明一个字的数据	p. 70
• endc——结束自动常数块	p. 72
• fill——指定程序存储器填充值	p. 82
• res——保留存储器	p. 107

4.2.4 列表伪指令

列表伪指令控制 MPASM 汇编器列表文件的格式。这些伪指令允许对标题、分页及其他列表控制进行规范。有些列表伪指令还控制如何汇编代码。

• error——发出一条错误消息	p. 74
• errorlevel——设置消息级别	p. 76
• list——列表选项	p. 93
• messg——创建用户定义的消息	p. 98
• nolist——关闭列表输出	p. 100
• page——在列表文件中插入换页符	p. 103
• space——插入空白列表行	p. 110
• subtitle——指定程序副标题	p. 110
• title——指定程序标题	p. 111

4.2.5 宏伪指令

宏伪指令在宏体定义内部控制执行和数据分配。

• endm——结束宏定义	p. 73
• exitm——退出宏	p. 78
• expand——扩展宏列表	p. 80
• local——声明局部宏变量	p. 94
• macro——声明宏定义	p. 96
• noexpand——关闭宏扩展	p. 100

4.2.6 目标文件伪指令

只有在创建目标文件时使用目标文件伪指令。

• access_ovr——在快速操作 RAM 中开始目标文件覆盖段 (PIC18 MCU)	p. 48
• bankisel——生成间接存储区选择代码 (PIC12/16 MCU)	p. 50
• banksel——生成存储区选择代码	p. 52
• code——开始目标文件代码段	p. 56
• code_pack——开始一个在目标文件中被压缩的代码段 (PIC18 MCU) .	p. 57
• extern——声明一个外部定义的标号	p. 80
• global——导出标号	p. 84
• idata——开始目标文件已初始化的数据段	p. 85
• idata_acs——在快速操作 RAM 中开始目标文件已初始化的数据段 (PIC18 MCU)	p. 86
• pagesel——生成页面选择代码 (PIC10/12/16 MCU)	p. 103
• pageselw——使用 WREG 命令生成页选择代码 (PIC10/12/16 MCU) .	p. 105
• udata——开始目标文件中未初始化的数据段	p. 111
• udata_acs——开始目标文件未初始化快速操作的数据段 (PIC18 MCU)	p. 113
• udata_ovr——开始目标文件中覆盖的未初始化的数据段	p. 114
• udata_shr——开始目标文件中共享的未初始化的数据段 (PIC12/16 MCU)	p. 116

4.3 `access_ovr`——在快速操作 RAM 中开始目标文件覆盖段 (PIC18 MCU)

4.3.1 语法

```
[label] access_ovr [RAM_address]
```

4.3.2 说明

此伪指令声明在快速操作 RAM 中开始一段覆盖数据。如果未指定 `label`，此数据段被命名为 `.access_ovr`。起始地址被初始化为指定的地址，如果没有指定地址，则在链接时分配起始地址。此段声明的空间被所有其他同名的 `access_ovr` 段覆盖。用户不能将任何代码放置在此段。

4.3.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

`access_ovr` 与 `udata_acs` 和 `udata_ovr` 类似，不同之处仅在于 `access_ovr` 在 PIC18 快速操作 RAM 中声明了一段未初始化的数据段，该数据段可以被其他同名的覆盖快速操作段覆盖。覆盖快速操作段使您能复用快速操作存储区的数据空间。

4.3.4 相关伪指令

```
extern global udata udata_ovr udata_acs
```

4.3.5 简单示例

```
;The 2 indentially-named sections are overlayed in PIC18 Access RAM.  
;In this example, u16a is overlaid with memory locations used  
;by ua8 and u8b. u16b is overlaid with memory locations used  
;by u8c and u8d.
```

```
myaoscn      access_ovr  
u8a:         res 1  
u8b:         res 1  
u8c:         res 1  
u8d:         res 1
```

```
myaoscn      access_ovr  
u16a:        res 2  
u16b:        res 2
```

4.4 `__badram`——标识未用的 RAM

注： `badram` 前面有两根下划线。

4.4.1 语法

```
__badram expr[-expr][, expr[-expr]]
```


4.4.2 说明

__maxram 和 __badram 伪指令一起标记对未用寄存器的访问。__badram 定义了具有无效 RAM 地址的单元。此伪指令设计用以与 __maxram 伪指令一起使用。

__maxram 伪指令必须出现在任一条 __badram 伪指令之前。每个 *expr* 必须小于或等于由 __maxram 指定的值。一旦使用了 __maxram 伪指令，就启用了严格的 RAM 地址检查，RAM 地址检查使用由 __badram 指定的 RAM 映射。要指定一系列无效单元，请使用语法 *minloc* - *maxloc*。

4.4.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

__badram 不常使用，因为 RAM 和 ROM 的细节问题由包含文件 (*.inc) 或链接描述文件 (*.lkr) 处理。

4.4.4 相关伪指令

__maxram

4.4.5 简单示例

```
#include p16c622.inc
__maxram 0x0BF
__badram 0x07-0x09, 0x0D-0xE
__badram 0x87-0x89, 0x8D, 0x8F-0x9E
movwf 0x07 ; Generates invalid RAM warning
movwf 0x87 ; Generates invalid RAM warning
           ; and truncation message
```

4.5 __badrom——标识未用的 ROM

注： badrom 的前面有两根下划线。

4.5.1 语法

__badrom *expr*[-*expr*][, *expr*[-*expr*]]

4.5.2 说明

__maxrom 和 __badrom 伪指令一起标记对未用寄存器的访问。__badrom 定义了具有无效 ROM 地址的单元。此伪指令设计用以与 __maxrom 伪指令一起使用。

__maxrom 伪指令必须出现在 __badrom 伪指令之前。每个 *expr* 必须小于或等于由 __maxrom 指定的值。一旦使用 __maxrom 伪指令，就启用了严格的 ROM 地址检查，ROM 地址检查使用由 __badrom 指定的 ROM 映射。要指定一系列无效单元，请使用语法 *minloc* - *maxloc*。

在下列特定情况下将发出警告。

- GOTO 或 CALL 指令的目标被汇编器计算为一个常数，并落在无效 ROM 区域内
- LGOTO 或 LCALL 伪操作码的目标被汇编器计算为一个常数，并落在无效 ROM 区域内
- 生成一个 .hex 文件，且指令的一部分落在无效 ROM 区域内

4.5.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

__badrom 不常使用，因为 RAM 和 ROM 的细节问题由包含文件 (*.inc) 或链接描述文件 (*.lkr) 处理。

4.5.4 相关伪指令

__maxrom

4.5.5 简单示例

```
#include p12c508.inc
__maxrom 0x1FF
__badrom 0x2 - 0x4, 0xA
org 0x5
    goto 0x2    ; generates a warning
    call 0x3    ; generates a warning
org 0xA
    movlw 5     ; generates a warning
```

4.6 bankisel——生成间接存储区选择代码（PIC12/16 MCU）

4.6.1 语法

bankisel label

4.6.2 说明

此伪指令指示汇编器或链接器来生成相应的存储区选择代码，以对由 label 指定的寄存器地址进行间接访问。应只指定一个 label。不能对 label 执行任何操作。必须已经在前面定义了此标号。

链接器将生成相应的存储区选择代码。对于指令宽度为 14 位的（大多数 PIC12/PIC16）器件，将生成针对 STATUS 寄存器中 IRP 位的相应置位 / 清零指令。如果可以在不使用这些指令的情况下指定间接地址，将不会生成任何代码。

4.6.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

指令宽度为 14 位的 PICmicro 器件可使用此伪指令。指令宽度为 12 位的器件和 PIC18 器件不可使用此伪指令。

4.6.4 相关伪指令

banksel pagesel

4.6.5 简单示例

```
movlw Var1
movwf FSR      ;Load the address of Var1 info FSR
bankisel Var1  ;Select the correct bank for Var1
:
movwf INDF     ;Indirectly write to Var1
```

4.6.6 应用程序示例——bankisel

此程序演示了 bankisel 伪指令。此伪指令生成相应的代码来置位 / 清零 STATUS 寄存器的 IRP 位以进行间接访问。

```
#include p16f877a.inc      ;Include standard header file
                           ;for the selected device.

group1  udata  0x20        ;group1 data stored at locations
                           ;starting at 0x20 (IRP bit 0).
      group1_var1  res  1    ;group1_var1 located at 0x20.
      group1_var2  res  1    ;group1_var2 located at 0x21.

group2  udata  0x120       ;group2 data stored at locations
                           ;starting at 0x120 (IRP bit 1).
      group2_var1  res  1    ;group2_var1 located at 0x120.
      group2_var2  res  1    ;group2_var2 located at 0x121.

RST      CODE      0x0     ;The code section named RST
                           ;is placed at program memory
                           ;location 0x0. The next two
                           ;instructions are placed in
                           ;code section RST.
      pagesel  start      ;Jumps to the location labelled
      goto    start      ;'start'.

PGM      CODE              ;This is the begining of the
                           ;code section named PGM. It is
                           ;a relocatable code section
                           ;since no absolute address is
                           ;given along with directive CODE.

start
      movlw  0x20          ;This part of the code addresses
      movwf  FSR           ;variables group1_var1 &
      bankisel group1_var1 ;group1_var1 indirectly.
      clrf  INDF
      incf  FSR,F
      clrf  INDF

      movwf  FSR
      bankisel group2_var1
      clrf  INDF
      incf  FSR,F
      clrf  INDF

      goto  $              ;Go to current line (loop here)
      end
```

4.6.7 应用程序示例 2 —— bankisel

```
#include p16f877a.inc      ;Include standard header file
                           ;for the selected device.

bankisel EEADR              ;This register is at location 100h
                           ;in banks 2 or 3 so the IRP bit
                           ;must be set. bankisel will set it
                           ;but only where it is used.

movlw    EEADR,W            ;Put the address of the register to
                           ;be accessed indirectly into W.

movwf    FSR                ;Copy address from W to FSR to set
                           ;up pointer to EEADR.

clrf     INDF               ;Clear EEADR through indirect
                           ;accessing of EEADR through FSR/INDF.
                           ;It would have cleared PIR2 (00Dh)
                           ;if bankisel had not been used to
                           ;set the IRP bit.

goto     $                  ;Prevents fall off end of code.
end                          ;All code must have an end statement.
```

4.7 banksel——生成存储区选择代码

4.7.1 语法

```
banksel label
```

4.7.2 说明

此伪指令指示汇编器和链接器生成存储区选择代码，以将存储区设置为包含指定 *label* 的存储区。应只指定一个 *label*。不能对 *label* 执行任何操作。必须已经在前面定义了此标号。

链接器将生成相应的存储区选择代码。对于指令宽度为 12 位的（PIC10F 和一些 PIC12/PIC16）器件，将生成针对 FSR 的相应的位置位 / 清零指令。对于指令宽度为 14 位的（大多数 PIC12/PIC16）器件，将生成针对 STATUS 寄存器的位置位 / 清零指令。对于 PIC18 器件，将生成 movlb。如果器件仅包含一个 RAM 存储区，将不会生成任何指令。

4.7.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

所有的 PICmicro 器件都可以使用此伪指令。快速操作 RAM 中的变量不需要此伪指令（PIC18 器件）。

4.7.4 相关伪指令

```
bankisel pagesel
```

4.7.5 简单示例

```
banksel Var1 ;Select the correct bank for Var1
movwf Var1   ;Write to Var1
```

4.7.6 应用程序示例——banksel

此程序演示了 banksel 伪指令。此伪指令生成相应的代码来置位 / 清零 STATUS 寄存器的 RP0 和 RP1 位。

```
#include p16f877a.inc      ;Include standard header file
                           ;for the selected device.

group1  udata  0x20        ;group1 data stored at locations
                           ;starting at 0x20 (bank 0).
      group1_var1  res  1    ;group1_var1 located at 0x20.
      group1_var2  res  1    ;group1_var2 located at 0x21.

group2  udata  0xA0        ;group2 data stored at locations
                           ;starting at 0xA0 (bank 1)
      group2_var1  res  1
      group2_var2  res  1

RST      CODE      0x0      ;The code section named RST
                           ;is placed at program memory
                           ;location 0x0. The next two
                           ;instructions are placed in
                           ;code section RST.
      pagesel  start      ;Jumps to the location labelled
      goto     start      ;'start'.

PGM      CODE            ;This is the begining of the
                           ;code section named PGM. It is
                           ;a relocatable code section
                           ;since no absolute address is
                           ;given along with directive CODE.

start
      banksel  group1_var1 ;This directive generates code
                           ;to set/clear bank select bits
                           ;RP0 & RP1 of STATUS register
                           ;depending upon the address of
                           ;group1_var1.

      clrf  group1_var1
      clrf  group1_var2

      banksel  group2_var1 ;This directive generates code
                           ;to set/clear bank select bits
                           ;RP0 & RP1 of STATUS register
                           ;depending upon the address of
                           ;group2_var1.

      clrf  group2_var1
      clrf  group2_var2

      goto  $              ;Go to current line (loop here)
      end
```

4.7.7 应用程序示例 2 —— banksel

```
#include p16f877a.inc      ;Include standard header file
                           ;for the selected device.

banksel TRISB              ;Since this register is in bank 1,
                           ;not default bank 0, banksel is
                           ;used to ensure bank bits are correct.

clrf    TRISB              ;Clear TRISB. Sets PORTB to outputs.
banksel PORTB              ;banksel used to return to bank 0,
                           ;where PORTB is located.

movf    PORTB, 0x55        ;Set PORTB value.
goto    $
end                                ;All programs must have an end.
```

4.8 cblock——定义常数块

4.8.1 语法

```
cblock [expr]
      label[:increment][,label[:increment]]
endc
```

4.8.2 说明

定义命名的连续符号列表。此伪指令的作用是将地址偏移量分配给很多标号。当遇到 `endc` 伪指令时，此名称列表结束。

`expr` 表明块中第一个名称的起始值。如果没有表达式，第一个名称将接收到一个值，这个值比前一个 `cblock` 中的最后一个名称的值大 1。如果源文件中的第一个 `cblock` 没有 `expr`，被赋给的值将从零开始。

如果指定了 `increment`，那么下一个 `label` 将被赋给一个比前一个 `label` 的值大 `increment` 的值。

在一行可以给出多个名称，各名称用逗号分开。

`cblock` 用于在程序和数据存储器中定义常数以生成绝对代码。

4.8.3 使用

此伪指令在以下类型的代码中使用：绝对代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

使用此伪指令代替或辅助 `equ` 伪指令。当创建非可重定位（绝对）代码时，经常使用 `cblock` 来定义变量地址单元名称。不要使用 `cblock` 或 `equ` 来为可重定位代码定义变量单元名称。

4.8.4 相关伪指令

```
endc equ
```

4.8.5 简单示例

```

cblock 0x20      ; name_1 will be assigned 20
  name_1, name_2 ; name_2, 21 and so on
  name_3, name_4 ; name_4 is assigned 23.
endc
cblock 0x30
  TwoByteVar: 0, TwoByteHigh, TwoByteLow ;TwoByteVar =0x30
                                           ;TwoByteHigh=0x30
                                           ;TwoByteLow  =0x31

  Queue: QUEUE_SIZE
  QueueHead, QueueTail
  Double1:2, Double2:2
endc

```

4.8.6 应用程序示例——cblock/endc

此示例说明了如何使用 CBLOCK 和 ENDC 伪指令在数据存储器空间中定义常数或变量。这两个伪指令也可以用于程序存储器空间。

此程序计算矩形的周长。矩形的长度和宽度将存储在缓冲器中，通过 length（22H）和 width（23H）可对这两个缓冲器寻址。计算得到的周长将被存储在一个双精度的缓冲器中，可以通过 perimeter（即，20H 和 21H）对该缓冲器寻址。

```

#include p16f877a.inc ;Include standard header file
                        ;for the selected device.
CBLOCK 0x20            ;Define a block of variables
                        ;starting at 20H in data memory.
  perimeter:2          ;The label perimeter is 2 bytes
                        ;wide. Address 20H and 21H is
                        ;assigned to the label perimeter.
  length               ;Address 22H is assigned to the
                        ;label length.
  width                ;Address 23H is assigned to the
                        ;label width.
ENDC                  ;This directive must be supplied
                        ;to terminate the CBLOCK list.
clrf    perimeter+1    ;Clear perimeter high byte
                        ;at address 21H.
movf    length,w       ;Move the data present in the
                        ;register addressed by 'length'
                        ;to 'w'
addwf   width,w        ;Add data in 'w' with data in the
                        ;register addressed by 'width'.
                        ;STATUS register carry bit C
                        ;may be affected.
movwf   perimeter      ;Move 'w' to the perimeter low
                        ;byte at address 20H. Carry bit
                        ;is unaffected.
rlf     perimeter+1    ;Increment register 21H if carry
                        ;was generated. Also clear carry
                        ;if bit was set.
rlf     perimeter      ;Multiply register 20H by 2.
                        ;Carry bit may be affected.
rlf     perimeter+1    ;Again, increment register 21H
                        ;if carry was generated.
goto    $              ;Go to current line (loop here)
end

```

4.9 code——开始目标文件代码段

4.9.1 语法

```
[label] code [ROM_address]
```

4.9.2 说明

此伪指令声明开始一段程序代码。如果未指定 *label*，该段被命名为 *.code*。起始地址被初始化为指定的地址，如果没有指定地址，则在链接时分配起始地址。

注： 源文件中的两个代码段不能同名。

4.9.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

没有“end code”伪指令。当定义了另一个代码段或数据段或者到达文件的末尾时，一个段的代码自动结束。

4.9.4 相关伪指令

```
extern code_pack global idata udata udata_acs udata_ovr udata_shr
```

4.9.5 简单示例

```
RESET code 0x01FF  
goto START
```

4.9.6 应用程序示例——code

此程序演示 code 伪指令，该指令声明开始一个程序代码段。

```
#include p16f877a.inc      ;Include standard header file  
                           ;for the selected device.  
  
RST      CODE      0x0      ;The code section named RST  
                           ;is placed at program memory  
                           ;location 0x0. The next two  
                           ;instructions are placed in  
                           ;code section RST.  
        pagesel  start      ;Jumps to the location labelled  
        goto     start      ;'start'.
```



```

PGM      CODE                ;This is the begining of the
                                ;code section named PGM. It is
                                ;a relocatable code section
                                ;since no absolute address is
                                ;given along with directive CODE.

start
  clrw
  goto $                      ;Go to current line (loop here)

      CODE                    ;This is a relocatable code
nop                                ;section since no address is
                                ;specified. The section name will
                                ;be, by default, .code.

end

```

4.10 code_pack——开始一个在目标文件中被压缩的代码段（PIC18 MCU）

4.10.1 语法

```
[label] code_pack [ROM_address]
```

4.10.2 说明

此伪指令声明开始一段程序代码或 ROM 数据，其中一个全零的填充字节不会附加到奇数个字节上。如果未指定 *label*，该段被命名为 *.code*。起始地址被初始化为 *ROM_address*，如果没有指定地址，则在链接时分配起始地址。如果指定了 *ROM_address*，它必须是字对齐的。如果需要填充数据，请使用 *db*。

注： 源文件中的两个段不能同名。

4.10.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令常用于将数据存储到 PIC18 器件的程序存储器（与 *db* 一起使用）或 EEPROM 数据存储器（与 *de* 一起使用）中。

4.10.4 相关伪指令

```
extern code global idata udata udata_acs udata_ovr udata_shr
```

4.10.5 简单示例

```

00001 LIST P=18Cxx
                                00002
                                00003 packed code_pack 0x1F0
0001F0 01 02 03                00004 DB 1, 2, 3
0001F3 04 05                    00005 DB 4, 5
                                00006
                                00007 padded code
000000 0201 0003                00008 DB 1, 2, 3
000004 0504                    00009 DB 4, 5
                                00010
                                00011 END

```

4.11 __config——设置处理器的配置位

注： `config` 的前面有两根下划线。

4.11.1 语法

首选语法：

```
__config expr  
__config addr, expr (仅 PIC18)
```

注： PIC18FXXJ 不支持此伪指令。使用 `config` 伪指令（无下划线）。

支持的语法：

```
__fuses expr
```

4.11.2 说明

设置处理器的配置位。如果使用 MPLAB IDE，在使用此伪指令之前，必须通过命令行、`list` 伪指令或 `processor` 伪指令或 [Configure>Select Device](#) 声明处理器。有关配置位的说明，请参见各种 PICmicro 单片机的数据手册。

PIC10/12/16 MCU

将处理器的配置位设置为由 *expr* 说明的值。

PIC18 MCU

对于由 *addr* 指定的有效配置字节的地址，将配置位设置为由 *expr* 说明的值。

注： 配置位必须以升序排列。

虽然此伪指令可以用来为 PIC18 MCU 器件设置配置位，但还是推荐您使用 `config` 伪指令（该指令前没有下划线）。对于 PIC18FXXJ 器件，必须使用 `config` 伪指令。

注： 不要在同一个代码文件中混用 `__config` 和 `config` 伪指令。

4.11.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令被放在源代码中，因此当代码在被汇编为 `hex` 文件时，这些配置值已被预设为应用程序中所需的值。当把您的文件交给第三方编程厂家时这很有用，因为这有助于确保对器件编程时其配置正确。

将配置位赋值放在代码的开头。在标准包含（*.inc）文件中使用配置选项（名称）。这些名称可以用 `&` 按位进行“与”运算以声明多个配置位。

4.11.4 相关伪指令

```
config __idlocs list processor
```

4.11.5 简单示例

例 1: PIC16 器件

```
#include p16f877a.inc    ;include file with config bit definitions
__config _HS_OSC & _WDT_OFF & _LVP_OFF    ;Set oscillator to HS,
                                           ;watchdog time off,
                                           ;low-voltage prog. off
```

例 2: PIC17X 器件

```
#include p17c42.inc    ;include file with config bit definitions
__config 0xFFFF        ;default configuration bits
```

例 3: PIC18 器件

```
#include p18c452.inc    ;Include standard header file
                        ;for the selected device.

;code protect disabled.
__CONFIG    _CONFIG0, _CP_OFF_0

;Oscillator switch disabled, RC oscillator with OSC2
;as I/O pin.
__CONFIG    _CONFIG1, _OSCS_OFF_1 & _RCIO_OSC_1

;Brown-OutReset enabled, BOR Voltage is 2.5v
__CONFIG    _CONFIG2, _BOR_ON_2 & _BORV_25_2

;Watch Dog Timer enable, Watch Dog Timer PostScaler
;count - 1:128
__CONFIG    _CONFIG3, _WDT_ON_3 & _WDTPS_128_3

;CCP2 pin Mux enabled
__CONFIG    _CONFIG5, _CCP2MX_ON_5

;Stack over/underflow Reset enabled
__CONFIG    _CONFIG6, _STVR_ON_6
```

4.12 config——设置处理器的配置位（PIC18 MCU）

4.12.1 语法

```
config setting=value [, setting=value]
```

4.12.2 说明

定义一个配置位设置定义的列表。此列表将 *setting* 代表的 PIC18 处理器的配置位设置为 *value* 说明的值。有关配置位的说明，请参见具体 PIC18 单片机的数据手册。可用的设置和值可在标准处理器包含 (*.inc) 文件和 *PIC18 Configuration Settings Addendum* (DS51537) 中找到。

在一行上可以定义多个设置，各设置间用逗号分开。也可以在不同的行上定义一个配置字节的设置。

在使用此伪指令之前，必须通过命令行、list 伪指令、processor 伪指令或 MPLAB IDE 中的 Configure>Select Device 来声明 PIC18 MCU。

另一条可以用来为 PIC18 MCU 器件设置配置位的伪指令是 `__config` 伪指令，但不推荐在新的代码中使用此伪指令。

注： 不要在同一代码文件中混用 <code>__config</code> 和 <code>config</code> 伪指令。

4.12.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令被放在源代码中，从而使代码在被编译 / 汇编为 hex 文件时，这些配置值已被预设为应用程序中的所需值。当把您的文件交给第三方编程厂家时这很有用，因为这可以确保对器件编程时器件的配置正确。

将配置位赋值放在代码的开头。使用标准包含 (`*.inc`) 文件或附录中所列的配置选项 (`setting=value` 对)。在源代码中可以多次使用 `config` 伪指令，但是如果同一位被多次赋值，就会产生错误，即：

```
CONFIG CP0=OFF, WDT=ON
CONFIG CP0=ON ; (由于 CP0 被两次赋值，所以会产生错误)
```

4.12.4 相关伪指令

`__config __idlocs list processor`

4.12.5 简单示例

```
#include p18f452.inc           ;Include standard header file
                                ;for the selected device.

;code protect disabled
CONFIG      CP0=OFF

;Oscillator switch enabled, RC oscillator with OSC2 as I/O pin.
CONFIG      OSCS=ON, OSC=LP

;Brown-OutReset enabled, BOR Voltage is 2.5v
CONFIG      BOR=ON, BORV=25

;Watch Dog Timer enable, Watch Dog Timer PostScaler count - 1:128
CONFIG      WDT=ON, WDTPS=128

;CCP2 pin Mux enabled
CONFIG      CCP2MUX=ON

;Stack over/underflow Reset enabled
CONFIG      STVR=ON
```

4.13 `constant`——声明符号常数

4.13.1 语法

```
constant label=expr [... ,label=expr]
```

4.13.2 说明

创建在 MPASM 汇编器表达式中使用的符号。一旦常数被初始化之后，就不可以被重置了，而且赋值时表达式必须完全可解析。这是声明为常数和那些声明为变量或者由 `set` 伪指令创建的符号之间的主要区别。除此之外，常数和变量可以在绝对代码表达式中互换使用。

4.13.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

虽然更常使用 `equ` 或 `cblock` 来创建常数，但是 `constant` 伪指令也能创建常数。

4.13.4 相关伪指令

`set variable equ cblock`

4.13.5 示例

参见 `variable` 下的示例。

4.14 `da`——在程序存储器中存储字符串（PIC12/16 MCU）

4.14.1 语法

```
[label] da expr [, expr2, ..., exprn]
```

4.14.2 说明

`da`——数据 ASCII。

生成压缩的 14 位数字来表示两个 7 位 ASCII 字符。

4.14.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

PIC16 MCU 可以使用此伪指令将字符串存储在存储器中。

4.14.4 简单示例

- `da "abcdef"`
会将 30E2 31E4 32E6 放入程序存储器
- `da "12345678" ,0`
会将 18B2 19B4 1AB6 1BB8 0000 放入程序存储器
- `da 0xFFFF`
会将 0x3FFF 放入程序存储器

4.14.5 应用程序示例——`da`

此示例演示了使用伪指令 `da` 将字符串存入 14 位架构器件的程序存储器中。此伪指令生成一个压缩的 14 位数字来表示两个 7 位 ASCII 字符。

```
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.

ORG    0x0000          ;The following code will be
                        ;programmed in reset address 0.
```

```
goto start          ;Jump to an address labelled
                    ;'start'.

start               ;Write your main program here.

goto $              ;Go to current line (loop here)

ORG 0x1000          ;Store the string starting from
                    ;1000H.

Ch_stng da "PICmicro"

伪指令 da 生成四个 14 位数字: 2849、21ED、34E3 和 396F, 分别代表 PI、Cm、ic
和 ro 所对应的 ASCII 码。更多信息, 请参见以下内容。

Sngl_ch da "A"      ;7-bit ASCII equivalents of 'A'
                    ;and a NULL charater will be packed
                    ;in a 14-bit number.

da 0xff55           ;Places 3f55 in program memory.
                    ;No packing.

end
```

确定 14 位数字

对于如下语句:

```
Ch_stng da "PICmicro"
```

伪指令 `da` 生成四个 14 位数字: 2849、21ED、34E3 和 396F, 分别代表 PI、Cm、ic 和 ro 所对应的 ASCII 码。

要了解如何确定 14 位数字, 请查看 P 和 I 的 ASCII 值, 它们分别为 50h (01010000) 和 49h (01001001)。每个值都可以用 7 位二进制码表示为 (0) 1010000 和 (0) 1001001。压缩的 14 位数字是 101000 01001001, 它被存储为 (00) 101000 01001001 或 2849。

4.15 data——创建数字和文本数据

4.15.1 语法

```
[label] data expr[,expr,...,expr]
[label] data "text_string"[,"text_string",...]
```

4.15.2 说明

用数据初始化一个或多个程序存储器字。数据的形式可以是: 常数、可重定位或外部标号, 或是上述任何一种的表达式。该数据还可以包括 ASCII 字符串 `text_string`, 单字符需要加上单引号, 字符串需要加上双引号。单字符项被放入一个字的低字节, 而字符串是将其包含的每两个字符装在一个字中。如果一个字符串给出了奇数个字符, 则需要在最后一个字节填充零。对于除 PIC18 以外的所有器件系列, 首字符在字的最高字节中。对于 PIC18 器件系列, 首字符在字的最低有效字节中。

4.15.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

在生成可链接的目标文件时，也可用此伪指令来声明初始化的数据值。如需了解更多信息，请参见 `idata` 伪指令。

`db` 和其他数据伪指令比 `data` 更常使用。

4.15.4 相关伪指令

`db de dt dw idata`

4.15.5 简单示例

```
data reloc_label+10    ; constants
data 1,2,ext_label     ; constants, externals
data "testing 1,2,3"   ; text string
data 'N'               ; single character
data start_of_program ; relocatable label
```

4.15.6 PIC16 应用程序示例——`data`

此示例演示了使用伪指令 `data` 将一个或多个字存入程序存储器。

```
#include p16f877a.inc    ;Include standard header file
                        ;for the selected device.

ORG    0x0000           ;The following code will be
                        ;programmed in reset address 0.

goto   start           ;Jump to an address labelled
                        ;'start'.

start                    ;Write your main program here.

goto   $               ;Go to current line (loop here)

ORG    0x1000           ;Store the string starting from
                        ;1000H.

Ch_stng    data        'M','C','U' ;3 program memory locations
                        ;will be filled with ASCII
                        ;equivalent of 'M','C' and
                        ;'U'.
```

伪指令 `data` 生成三个 14 位数字：004Dh、0043h 和 0055h。4Dh、43h 和 55h 分别是 ‘M’、‘C’ 和 ‘U’ 所对应的 ASCII 码分别。

```
tbl_dta    data        0xffff,0xaa55 ;Places 3fffh and 2a55h in
                        ;two consecutive program
                        ;memory locations. As program
                        ;memory is 14-bit wide,
                        ;the last nibble can store
                        ;a maximum value 3.

end
```

4.15.7 PIC18 应用程序示例——data

此示例演示了使用伪指令 `data` 将一个或多个字存入程序存储器。

```
#include p18f452.inc      ;Include standard header file
                           ;for the selected device.

ORG    0x0000             ;The following code will be
                           ;programmed in reset address 0.
goto   start              ;Jump to an address labelled
                           ;'start'.

start                          ;Write your main program here.

goto   $                  ;Go to current line (loop here)

ORG    0x1000             ;Store the string starting from
                           ;1000H. In PIC18 devices, the
                           ;first character is in least
                           ;significant byte.

Ch_stng    data    'M','C','U' ;3 program memory locations
                           ;will be filled with ASCII
                           ;equivalent of 'M','C' and
                           ;'U'.
```

伪指令 `data` 生成三个 16 位数字：004Dh、0043h 和 0055h。4Dh、43h 和 55h 分别是 ‘M’、‘C’ 和 ‘U’ 所对应的 ASCII 码。为了更好地利用存储器，请参见第 4.10 节 “**code_pack**——开始一个在目标文件中被压缩的代码段（PIC18 MCU）”。

```
Ch_stg1    data    "MCU"      ;2 program memory locations
                           ;will be filled with two
                           ;words (16-bit numbers),
                           ;each representing ASCII
                           ;equivalent of two
                           ;characters. The last
                           ;character will be taken as
                           ;NULL in case odd number of
                           ;characters are specified.
```

伪指令 `data` 生成两个字：434Dh 和 0055h。434Dh 代表 ‘C’ 和 ‘M’。

```
tbl1_dta   data    0xffff,0xaa55 ;Places ffff and aa55 in
                           ;two consecutive program
                           ;memory locations.

end
```


4.16 db——声明一个字节的数据

4.16.1 语法

```
[label] db expr[,expr,...,expr]
```

4.16.2 说明

db——数据字节。

用 8 位值保留程序存储器字。多个表达式继续连续地填充字节，直到表达式结束为止。如果有奇数个表达式，最后一个字节将是零，除非在 PIC18 code_pack 段中。

4.16.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

在生成可链接的目标文件时，此伪指令也可以用来声明初始化的数据值。如需了解更多信息，请参见 idata 伪指令。

对于 PIC18 器件，将 code_pack 和 db 配合使用，因为不希望用零填充字节。如需了解更多信息，请参见 code_pack 的说明。

4.16.4 相关伪指令

```
data de dt dw idata code_pack
```

4.16.5 简单示例

例 1: PIC16 器件

```
db 0x0f, 't', 0x0f, 'e', 0x0f, 's', 0x0f, 't', '\n'
```

ASCII: 0x0F74 0x0F65 0x0F73 0x0F74 0x0a00

例 2: PIC18 器件

```
db 't', 'e', 's', 't', '\n'
```

ASCII: 0x6574 0x7473 0x000a

4.16.6 PIC16 应用程序示例——db

此示例演示了使用伪指令 db 将一个或多个字节或字符存入程序存储器。

```
#include p16f877a.inc    ;Include standard header file
                          ;for the selected device.

ORG    0x0000            ;The following code will be
                          ;programmed in reset address 0.
goto   start             ;Jump to an address labelled
                          ;'start'.

start                                ;Write your main program here.

goto   $                    ;Go to current line (loop here)

ORG    0x1000              ;Store the string starting from
                          ;1000H.

Ch_stng    db    0,'M',0,'C',0,'U'
```

Ch_strng 包含三个 14 位数字: 004Dh、0043h 和 0055h。它们分别是 ‘M’、‘C’ 和 ‘U’ 所对应的 ASCII 码。

```
tbl_dta    db    0,0xff        ;Places 00ff in program memory
                                   ;location.
```

```
end
```

4.16.7 PIC18 应用程序示例——db

此示例演示了使用伪指令 db 将一个或多个字节或字符存入程序存储器。

```
#include p18f452.inc    ;Include standard header file
                           ;for the selected device.

ORG    0x0000           ;The following code will be
                           ;programmed in reset address 0.
goto   start            ;Jump to an address labelled
                           ;'start'.

start                    ;Write your main program here.

goto   $                ;Go to current line (loop here)

ORG    0x1000           ;Store the string starting from
                           ;1000H. In PIC18 devices, the
                           ;first character is in least
                           ;significant byte.

Ch_stng    db    'M','C','U'
```

Ch_strng 包含三个 16 位数字: 004Dh、0043h 和 0055h。它们分别是 ‘M’、‘C’ 和 ‘U’ 所对应的 ASCII 码。如需了解有关在 PIC18 架构上将数据存储一个程序字的两个字节中的信息, 请参见第 4.10 节 “code_pack——开始一个在目标文件中被压缩的代码段 (PIC18 MCU)”。

```
tbl_dta    db    0,0xff        ;Places ff00 in program memory
                                   ;location.
```

```
end
```

4.17 de——声明 EEPROM 数据字节

4.17.1 语法

```
[label] de expr [, expr, ..., expr]
```

4.17.2 说明

de——数据 EEPROM。

此伪指令可用于任何处理器的任何单元。

对于 PIC18 器件，保留存储器字的字节是压缩的。如果指定了奇数个字节，将会添加一个零字节，除非在 code_pack 段中。如需了解更多信息，请参见 code_pack 的说明。

对于所有其他的 PICmicro 器件，用 8 位数据保留存储器字。每个 expr 必须计算为一个 8 位值。程序字的高字节是全零。字符串中的每一个字符存储在不同的字中。

4.17.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

设计此伪指令主要是为了在带有 EE 数据闪存的 PICmicro 器件的 EE 数据存储区域中初始化数据。

对于 PIC18 器件，确保将数据存储器的起始地址指定为 0xF00000。对于其他 PICmicro 器件，确保将数据存储器的起始地址指定为 0x2100。应始终参照器件编程规范检查地址是否正确。

4.17.4 相关伪指令

```
data db dt dw code_pack
```

4.17.5 简单示例

在 PIC16 器件上初始化 EEPROM 数据：

```
org 0x2100
de "My Program, v1.0", 0
```

4.17.6 PIC16 应用程序示例——`de`

```
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.

org 0x2100              ;The absolute address 2100h is
                        ;mapped to the 0000 location of
                        ;EE data memory.

;You can create a data or character table starting from any
;address in EE data memory.

ch_tbl2 de "PICmicro" ;8 EE data memory locations
                        ;(starting from 0) will be filled
                        ;with 8 ASCII characters.

end
```

4.17.7 PIC18 应用程序示例——`de`

```
#include p18f452.inc ;Include standard header file
                        ;for the selected device.

org 0xF00000           ;The absolute address F00000h is
                        ;mapped to the 0000 location of
                        ;EE data memory for PIC18 devices.

;You can create a data or character table starting from any
;address in EE data memory.

ch_tbl2 de "PICmicro" ;8 EE data memory locations
                        ;(starting from 0) will be filled
                        ;with 8 ASCII characters.

end
```

4.18 `#define`——定义文本替换标号

4.18.1 语法

```
#define name [string]
```

4.18.2 说明

此伪指令定义一个文本替换字符串。在汇编代码中的任何位置遇到 *name* 时，它将会被 *string* 替换。

使用不带 *string* 的伪指令将引起 *name* 的定义在内部被标记，并可能检测是否可使用 `ifdef` 伪指令。

此伪指令仿效了 ANSI “C” 标准中的 `#define`。使用此方法定义的符号不能使用 MPLAB IDE 查看。

4.18.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

#define 用于定义程序中常数的值。

注： 处理器特定的包含文件中有预定义的 SFR 名称。推荐您使用此文件，而不要自行定义变量。如需了解如何在程序中包含一个文件，请参见 #include。

此伪指令还可以和 ifdef 和 ifndef 伪指令一起使用，这两个伪指令将在符号表中寻找是否存在某个项。

4.18.4 相关伪指令

#undef #include ifdef ifndef

4.18.5 简单示例

```
#define length 20
#define control 0x19,7
#define position(X,Y,Z) (Y-(2 * Z +X))
:
:
test_label dw position(1, length, 512)
bsf control ; set bit 7 in f19
```

4.18.6 应用程序示例—— #define/#undef

此示例演示了 #define 和 #undef 伪指令的用法。使用 #undef 伪指令可以将先前用 #define 伪指令定义的符号名从符号表中删除。可以再次定义同一个符号。

```
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.

area set 0 ;The label 'area' is assigned
           ;the value 0.
#define lngth 50H ;Label 'lngth' is assigned
                ;the value 50H.
#define wdth 25H ;Label 'wdth' is assigned
                ;the value 25H
area set lngth*wdth ;Reassignment of label 'area'.
                   ;So 'area' will be reassigned a
                   ;value equal to 50H*25H.

#undef lngth ;Undefine label 'lngth'.
#undef wdth ;Undefine label 'wdth'
#define lngth 0 ;Define label 'lngth' to '0'.

end
```

通过使用上面的伪指令，lngth 将被重新赋值为 ‘0’，而 wdth 将从列表 (.lst) 文件中的符号列表中被删除。必须先取消标号 lngth 的定义，才能将它定义为 ‘0’。

4.19 dt——定义表 (PIC12/16 MCU)

4.19.1 语法

```
[label] dt expr [, expr, ..., expr]
```

4.19.2 说明

dt——数据表。

生成一系列 RETLW 指令，每个 *expr* 一条指令。每个 *expr* 必须是一个 8 位值。字符串中的每一个字符都存储在自己的 RETLW 指令中。

4.19.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

当为 PIC12/16 器件系列生成数据表时使用此伪指令。如果您使用 PIC18 器件，推荐您使用表读 / 表写 (TBLRD/TBLWT) 功能。如需了解更多信息，请参见器件数据手册。

4.19.4 相关伪指令

```
data db de dw
```

4.19.5 简单示例

```
dt "A Message", 0  
dt FirstValue, SecondValue, EndOfValues
```

4.20 dw——声明一个字的数据

4.20.1 语法

```
[label] dw expr[,expr,...,expr]
```

4.20.2 说明

dw——数据字。

为数据保留程序存储器字，将该空间初始化为特定值。对于 PIC18 器件而言，dw 的作用与 db 相同。值被存入连续的存储器单元中，且单元计数器增一。表达式可以是立即数字符串，并如同 db 数据伪指令中说明的那样存储。

4.20.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

在生成可链接的目标文件时，也可用此伪指令来声明初始化的数据值。如需了解更多信息，请参见 *idata* 伪指令。

虽然 db 更常用，但是您仍然可以使用 dw 将数据存储在闪存 PIC16FXXX 器件中，因为这些器件中有很多器件可以在运行时读取一个程序存储器字的全部 14 位。如需获取示例和更多信息，请参见 PIC16F877A 数据手册。

4.20.4 相关伪指令

```
data db idata
```

4.20.5 简单示例

```
dw 39, "diagnostic 39", 0x123
dw diagbase-1
```

4.21 else——开始 if 条件的备用汇编块

4.21.1 语法

首选语法:

```
else
```

支持的语法:

```
#else
.else
```

4.21.2 说明

与 if 伪指令一起使用，以在 if 求值为 FALSE 时提供汇编代码的备用路径。可以在常规程序块或宏内部使用 else。

4.21.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令不是指令。它用于执行代码的条件汇编。

4.21.4 相关伪指令

```
endif if
```

4.21.5 简单示例

```
if rate < 50
    incf speed, F
else
    decf speed, F
endif
```

4.21.6 应用程序示例——if/else/endif

参见 if 下的示例。

4.22 end——结束程序块

4.22.1 语法

```
end
```

4.22.2 说明

表示程序的结束。

4.22.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

在任何汇编程序中需要至少一条 end 伪指令来表示编译的结束。在单个汇编文件程序中，必须且只能使用一条 end 伪指令。

小心不要包含含有 `end` 的文件，因为这些文件会使汇编过早地停止。

4.22.4 相关伪指令

`org`

4.22.5 简单示例

```
#include p18f452.inc
:    ; executable code
:    ;
end    ; end of instructions
```

4.23 `endc`——结束自动常数块

4.23.1 语法

`endc`

4.23.2 说明

`endc` 终止 `cblock` 列表。必须提供此伪指令以终止列表。

4.23.3 使用

此伪指令在以下类型的代码中使用：绝对代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

每使用一条 `cblock` 伪指令，都必须有一条对应的 `endc` 伪指令。

4.23.4 相关伪指令

`cblock`

4.23.5 示例

参见 `cblock` 下的示例。

4.24 `endif`——结束条件汇编块

4.24.1 语法

首选语法：

`endif`

支持的语法：

```
#endif
.endif
.fi
```

4.24.2 说明

此伪指令标志着条件汇编块的结束。`endif` 可以在常规程序块或宏内部使用。

4.24.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

每使用一条 `if` 伪指令，都必须有一条对应的 `endif`。

`if` 和 `endif` 不是指令，仅用于代码汇编。

4.24.4 相关伪指令

`else if`

4.24.5 示例

参见 `if` 下的示例。

4.25 `endm`——结束宏定义

4.25.1 语法

`endm`

4.25.2 说明

终止由 `macro` 开始的宏定义。

4.25.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

每使用一条 `macro` 伪指令，都必须有一条对应的 `endm`。

4.25.4 相关伪指令

`macro` `exitm`

4.25.5 简单示例

```
make_table macro arg1, arg2
    dw arg1, 0 ; null terminate table name
    res arg2   ; reserve storage
endm
```

4.25.6 应用程序示例——`macro/endm`

参见 `macro` 下的示例。

4.26 `endw`——结束 `while` 循环

4.26.1 语法

首选语法：

`endw`

支持的语法：

`.endw`

4.26.2 说明

`endw` 终止 `while` 循环。只要由 `while` 伪指令指定的条件保持为 `TRUE`，`while` 伪指令和 `endw` 伪指令之间的源代码将在汇编源代码流中被重复地扩展。此伪指令可以在常规模程序块或宏内部使用。

4.26.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

每使用一条 `while` 伪指令，都必须有一条对应的 `endw`。

`while` 和 `endw` 不是指令，仅用于代码汇编。

4.26.4 相关伪指令

`while`

4.26.5 示例

参见 `while` 下的示例。

4.27 `equ`——定义一个汇编器常数

4.27.1 语法

`label equ expr`

4.27.2 说明

`expr` 的值被赋值给 `label`。

4.27.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

在单个汇编文件程序中，`equ` 通常用于将变量名称分配给 RAM 中的地址单元。在编译一个已链接的项目时，不要使用此方法分配变量；在数据段伪指令（`idata` 和 `udata`）内部使用 `res` 伪指令。

4.27.4 相关伪指令

`set cblock res idata udata udata_acs udata_ovr udata_shr`

4.27.5 简单示例

`four equ 4 ; assigned the numeric value of 4 to label four`

4.27.6 应用程序示例 —— `set/equ`

参见 `set` 下的示例。

4.28 `error`——发出一条错误消息

4.28.1 语法

`error "text_string"`

4.28.2 说明

`text_string` 的显示格式与 MPASM 汇编器的任何错误消息相同。`text_string` 字符可以为 1 到 80 个。

4.28.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

您可以使用此伪指令为您自己或其他编译您的代码的人生成错误消息。您可以根据需要生成任何错误消息，只要消息的长度不超过 80 个字符。

4.28.4 相关伪指令

```
messg if
```

4.28.5 简单示例

```
error_checking macro arg1
    if arg1 >= 55 ; if arg is out of range
        error "error_checking-01 arg out of range"
    endif
endm
```

4.28.6 应用程序示例——error

此程序演示了 error 汇编器伪指令，它将错误消息设置为显示在列表文件和错误文件中。

```
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.

variable baudrate      ;variable used to define
                        ;required baud rate

baudrate set D'5600'    ;Enter the required value of
                        ;baud rate here.

if (baudrate!=D'1200')&&(baudrate!=D'2400')&&
(baudrate!=D'4800')&&(baudrate!=D'9600')&&
(baudrate!=D'19200')
    error "Selected baud rate is not supported"
endif
```

如果所选择的波特率是除 1200、2400、4800、9600 或 19200 Hz 之外的其他波特率，上面的 if-endif 代码将输出 error。

```
RST      CODE      0x0      ;The code section named RST
                        ;is placed at program memory
                        ;location 0x0. The next two
                        ;instructions are placed in
                        ;code section RST.

        pagesel  start      ;Jumps to the location labelled
        goto     start      ;'start'.

PGM      CODE

                        ;This is the begining of the
                        ;code section named PGM. It is
                        ;a relocatable code section
                        ;since no absolute address is
                        ;given along with directive CODE.

start
        goto     $          ;Go to current line (loop here)
        end
```

4.29 errorlevel——设置消息级别

4.29.1 语法

```
errorlevel {0|1|2|+msgnum|-msgnum} [, ...]
```

4.29.2 说明

设置在列表文件和错误文件中显示的消息的类型。

设置	作用
0	显示消息、警告和错误
1	显示警告和错误
2	显示错误
-msgnum	禁止显示消息 msgnum
+msgnum	启用显示消息 msgnum

4.29.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

不能禁止错误。使用设置 2 可以禁止警告。使用设置 1 或 2 可以禁止消息。此外，还可以个别地禁止消息。然而，设置 0、1 或 2 会覆盖单个消息的禁止或启用。

禁止警告和消息时要小心，因为这可能使调试代码变得更加困难。

此伪指令最常见的用途是禁止“MESSAGE 302 - Operand Not in bank 0, check to ensure bank bits are correct”（消息 302——操作数不在存储区 0 中，检查以确认存储区是正确的）。参见简单示例以了解如何做到这一点。

4.29.4 相关伪指令

```
list error
```

4.29.5 简单示例

```
errorlevel -302 ; Turn off banking message
                ; known tested (good) code
:
errorlevel +302 ; Enable banking message
                ; untested code
:
end
```

4.29.6 应用程序示例——errorlevel

此程序演示了 errorlevel 汇编器伪指令，它用来设置显示在列表文件和错误文件中的消息类型。

```
#include p16f877a.inc    ;Include standard header file
                        ;for the selected device.

errorlevel 0             ;Display/print messages,
                        ;warnings and errors.

messg "CAUTION: This program has errors" ;display on build
```

对于错误级别 0，此消息将显示。

```
errorlevel 1          ;Display/print only warnings
                      ;and errors.

messg "CAUTION: This program has errors" ;display message
```

对于错误级别 1 或 2，此消息将不会显示。

```
group1  udata  0x20
group1_var1  res  1      ;Label of this directive is not
                        ;at column 1. This will generate
                        ;a warning number 207.
```

对于错误级别 0 或 1，将显示警告 207。

```
errorlevel  -207      ;This disables warning whose
                      ;number is 207.

group1_var2  res 1     ;label of this directive is also
                      ;not at column 1, but no warning
                      ;is displayed/printed.

errorlevel  +207      ;This enables warning whose
                      ;number is 207
```

```
group2  udata

errorlevel 2          ;Display/print only errors

group2_var1 res 1     ;label of this directive is not
                      ;at column 1. This will generate
                      ;a warning number 207.
```

对于错误级别 2，将不会显示警告 207。

```
errorlevel 1          ;Display/print warnings
                      ;and errors.

group2_var2 res 1     ;label of this directive is not
                      ;at column 1. This will generate
                      ;a warning number 207.

RST      CODE  0x0     ;The code section named RST
                      ;is placed at program memory
                      ;location 0x0. The next two
                      ;instructions are placed in
                      ;code section RST.

pagesel  start
goto     start         ;Jumps to the location labelled
                      ;'start'.

INTRT    CODE  0x4     ;The code section named INTRT is
                      ;placed at 0x4. The next two
                      ;instructions are placed in
                      ;code section INTRT

pagesel  service_int
goto     service_int   ;Label 'service_int' is not
                      ;defined. Hence this generates
                      ;error[113].
```

无论何种错误级别，总是显示错误 113。

```
PGM CODE ;This is the begining of the code
          ;section named 'PGM'. It is a
          ;relocatable code section since
          ;no absolute address is given along
          ;with directive CODE.

start
  movwf group1_var1
  goto $ ;Go to current line (loop here)
end
```

4.30 exitm——退出宏

4.30.1 语法

exitm

4.30.2 说明

在汇编时，强制立即从宏扩展返回。作用与遇到 `endm` 伪指令时相同。

4.30.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

使用此伪指令以提前结束宏，通常针对某种特定情况。这与 C 语言命令 `break` 相似。

4.30.4 相关伪指令

`endm macro`

4.30.5 简单示例

```
test macro filereg
  if filereg == 1 ; check for valid file
    exitm
  else
    error "bad file assignment"
  endif
endm
```

4.30.6 应用程序示例——exitm

此程序演示汇编器伪指令 `exitm`，该伪指令导致从宏立即退出。在示例中使用此伪指令在满足某些条件时从宏退出。

```
#include p16f877a.inc ;Include standard header file
                      ;for the selected device.

result equ 0x20 ;Assign value 20H to label
               ;result.

RST CODE 0x0 ;The code section named RST
             ;is placed at program memory
             ;location 0x0. The next two
             ;instructions are placed in
             ;code section RST.
```

```

    pagesel    start        ;Jumps to the location labelled
    goto       start        ;'start'.

add    MACRO    num1,num2    ;'add' is a macro. The values of
                             ;'num1' and 'num2' must be passed
                             ;to this macro.

    if    num1>0xff          ;If num1>255 decimal,
        exitm                ;force immediate return from
                             ;macro during assembly.

    else
        if    num2>0xff      ;If num2>255 decimal,
            exitm            ;force immediate return from
                             ;macro during assembly.

        else

    movlw    num1            ;Load W register with a literal
                             ;value assigned to the label
                             ;'num1'.

    movwf    result          ;Load W register to an address
                             ;location assigned to the label
                             ;'result'.

    movlw    num2            ;Load W register with a literal
                             ;value assigned to the label
                             ;'num2'.

    addwf    result          ;Add W register with the memory
                             ;location addressed by 'result'
                             ;and load the result back to
                             ;'result'.

    endif
endif
endm                        ;End of 'add' MACRO

    org      0010            ;My main program starts at 10H.

start                                         ;The label 'start' is assigned an
                                         ;address 10H.

    add     .100,.256         ;Call 'add' MACRO with decimal
                             ;numbers 100 and 256 assigned to
                             ;'num1' and 'num2' labels,
                             ;respectively. EXTIM directive in
                             ;macro will force return.
                             ;Remember '.' means decimal, not
                             ;floating point.

end

```

4.31 `expand`——扩展宏列表

4.31.1 语法

`expand`

4.31.2 说明

扩展列表文件中的所有宏。此伪指令大致与 MPASM 汇编器命令行选项 `/m` 相同，但是可由随后的 `noexpand` 禁止。

4.31.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令在分析内含很多宏的小块代码时可能有用。

4.31.4 相关伪指令

`macro noexpand`

4.32 `extern`——声明一个外部定义的标号

4.32.1 语法

`extern label [, label...]`

4.32.2 说明

此伪指令声明可在当前模块中使用但在另一个模块中被定义为全局标号的符号名称。

必须先包含 `extern` 语句，才能使用该 `label`。在这行上必须至少指定一个标号。如果在当前模块中定义了该 `label`，则 MPASM 汇编器将产生重复标号错误。

4.32.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

只要项目中有一个以上的文件，就可以使用此伪指令。

当某文件使用一个标号（通常是变量）时，则该文件将使用 `extern`。`global` 将在另一个文件中使用从而使该标号可被其他文件看到。您必须按照规定使用这两条伪指令，否则该标号在其他文件中将是不可见的。

4.32.4 相关伪指令

`global idata udata udata_acs udata_ovr udata_shr`

4.32.5 简单示例

```
extern Function
:
call Function
```


4.32.6 应用程序示例——extern/global

程序main.asm以及sub.asm，演示了global和extern伪指令，这两个指令使您能在模块中使用在其他模块中定义的符号。这样就允许将项目分成多个文件（此示例中为两个文件）以进行代码重用。

```

;*****
;main.asm
;*****
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.

UDATA
delay_value res 1

GLOBAL delay_value ;The variable 'delay_value',
                    ;declared GLOBAL in this
                    ;module, is included in an
                    ;EXTERN directive in the module
                    ;sub.asm.

EXTERN delay ;The variable 'delay', declared
             ;EXTERN in this module, is
             ;declared GLOBAL in the module
             ;sub.asm.

RST CODE 0x0 ;The code section named RST
             ;is placed at program memory
             ;location 0x0. The next two
             ;instructions are placed in
             ;code section RST.

    pagesel start ;Jumps to the location labelled
    goto start ;'start'.

PGM CODE ;This is the begining of the
          ;code section named PGM. It is
          ;a relocatable code section
          ;since no absolute address is
          ;given along with directive CODE.

start
    movlw D'10'
    movwf delay_value
    xorlw 0x80
    call delay

    goto start
end

;*****
;sub.asm
;*****
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.

GLOBAL delay ;The variable 'delay' declared
              ;GLOBAL in this module is
              ;included in an EXTERN directive
              ;in the module main.asm.

```

```
        EXTERN  delay_value      ;The variable 'delay_value'
                                   ;declared EXTERN in this module
                                   ;is declared GLOBAL in the
                                   ;module main.asm.

PGM  CODE

delay
    decfsz  delay_value,1
    goto   delay
    return

end
```

4.33 fill——指定程序存储器填充值

4.33.1 语法

```
[label] fill expr,count
```

4.33.2 说明

生成程序字或字节（PIC18 器件）*expr* 的 *count* 发生次数，以及 *expr*。如果用圆括号将 *expr* 括起来，*expr* 可以是一条汇编器指令。

4.33.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令经常用于将已知的数据强制放入未用的程序存储器。这有助于确保当代码在运行时跳转到未用区域时，发生故障保护状态。例如，PIC16 器件上的看门狗定时器（WDT）不太使用此伪指令。未用的程序存储器将被 *goto* 或跳转指令填充以阻止执行代码中的 *clrwdt* 指令，该指令将导致器件复位。如需了解更多有关 WDT 的信息，请参见器件数据手册。

4.33.4 相关伪指令

```
data dw org
```

4.33.5 简单示例

例 1: PIC10/12/16 MCU

```
fill 0x1009, 5 ; fill with a constant
fill (GOTO RESET_VECTOR), NEXT_BLOCK-$
```

例 2: PIC18 器件

```
#include pl8f252.inc

    org  0x12
failsafe goto $

    org  0x100
fill (goto failsafe), (0x8000-$)/2 ;Divide by 2 for
                                   ;2-word instructions
end
```

4.33.6 PIC16 应用程序示例——fill

fill 伪指令用常数或汇编指令指定连续的程序存储器单元。

```
#include p16f877a.inc    ;Include standard header file
                          ;for the selected device.

RST      CODE      0x0000 ;The code section named RST
                          ;is placed at program memory
                          ;location 0x0. The next two
                          ;instructions are placed in
                          ;code section RST.

      pagesel  start
      goto    start      ;Jumps to the location labelled
                          ;'start'.

      fill    0, INTRPT-$ ;Fill with 0 up to address 3 -
                          ;INTRPT addr. minus current addr.

INTRPT   CODE      0x0004 ;The code section named INTRPT
                          ;is placed at program memory
                          ;location 0x4. The next two
                          ;instructions are placed in
                          ;code section INTRPT.

      pagesel  ISR
      goto    ISR        ;Jumps to the location labelled
                          ;ISR.

      fill    (goto start), start-$ ;Fill upto address 0Fh with
                          ;instruction <goto start>.

      CODE      0x0010
start                                         ;Write your main program here.

      fill    (nop), 5      ;Fill 5 locations with NOPs.
      goto    $            ;Go to current line (loop here)

ISR                                           ;Write your interrupt service
      retfie                ;routine here.
      end
```

4.33.7 PIC18 应用程序示例——fill

fill 伪指令用常数或汇编指令指定连续的程序存储器单元。对于 PIC18 器件而言，只允许将偶数指定为将被填充的数据单元数。

```
#include p18f452.inc    ;Include standard header file
                          ;for the selected device.

RST      CODE      0x0000 ;The code section named RST
                          ;is placed at program memory
                          ;location 0x0. The instruction
                          ;'goto start' is placed in
                          ;code section RST.

      goto    start      ;Jumps to the location labelled
                          ;'start'.

      fill    0, HI_INT-$ ;Fills 0 in 2 program memory
                          ;locations: 0004 and 0006 -
                          ;HI_INT addr. minus current addr.

HI_INT   CODE      0x0008
      goto    INTR_H
```

```
fill    (goto start),6    ;Fills 6 locations (each location
                           ;is 2 bytes wide) with 3 numbers
                           ;of 2 word wide instructions
                           ;<goto start>

LO_INT  CODE    0x0018
goto    INTR_L
fill    10a9, start-$     ;Fills address 1Ch and 1Eh with
                           ;10a9h

CODE    0x0020
start   ;Write your main program here

fill    (nop), 4          ;Fills 2 locations (4 bytes) with
                           ;NOP
goto    $                 ;Go to current line (loop here)

INTR_H  ;Write your high interrupt ISR here
retfie
INTR_L  ;Write your low interrupt ISR here
retfie
end
```

4.34 global——导出标号

4.34.1 语法

```
global label [, label...]
```

4.34.2 说明

此伪指令声明当前模块中定义的并且对于其他模块可用的符号名。在这行上必须至少指定一个标号。

4.34.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

当项目使用多个文件时，需要生成可链接目标代码。当发生这种情况时，可以使用 `global` 和 `extern` 伪指令。

`global` 用于让标号对于其他文件可见。`extern` 必须在使用标号的文件中使用以便让标号在该文件中可见。

4.34.4 相关伪指令

```
extern idata udata udata_acs udata_ovr udata_shr
```

4.34.5 简单示例

```
global Var1, Var2
global AddThree

        udata
Var1    res 1
Var2    res 1
        code
AddThree
        addlw 3
        return
```

4.34.6 应用程序示例——extern/global

参见 extern 的示例。

4.35 idata——开始目标文件已初始化的数据段

4.35.1 语法

```
[label] idata [RAM_address]
```

4.35.2 说明

此伪指令声明开始一段已初始化的数据。如果未指定 `label`，此段被命名为 `.idata`。起始地址被初始化为指定的地址，如果没有指定地址，则在链接时分配起始地址。用户不能将代码放置在此段。

链接器将为在 `idata` 段中指定的每个字节生成查找表的项。然后必须链接或包含相应的初始化代码。在 MPLINK 链接器样本应用程序示例中可以找到可以使用而且在需要时可修改的初始化代码示例。

注： 12 位指令宽的（PIC10 和某些 PIC12/PIC16）器件不能使用此伪指令。

可以使用 `res`、`db` 和 `dw` 伪指令为变量保留空间。`res` 会产生初始值零。`db` 会初始化 RAM 的连续字节。`dw` 会按照从低字节到高字节的顺序一次初始化一个字，初始化 RAM 的连续字节。

4.35.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

使用此伪指令初始化变量，或者使用 `udata` 伪指令，然后用代码中的值初始化变量。建议您总是初始化变量。依赖 RAM 初始化会导致问题，特别是在使用仿真器的时候，因为仿真器和实际部件之间的行为差异会导致问题发生。

4.35.4 相关伪指令

```
extern global udata udata_acs udata_ovr udata_shr
```

4.35.5 简单示例

```
idata
LimitL dw 0
LimitH dw D'300'
Gain dw D'5'
Flags db 0
String db 'Hi there!'
```

4.35.6 应用程序示例——idata

此伪指令为变量保留 RAM 单元并指导链接器产生查找表，此查找表可被用于初始化在此段中指定的变量。可从映射文件（`.map`）获取查找表的起始地址。如果在 `idata` 段中不指定一个值，则变量将初始化为 0。

```
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.
```

```
group1  IDATA  0x20      ;Initialized data at location
                        ;20h.
    group1_var1  res  1    ;group1_var1 located at 0x20,
                        ;initialized with 0.
    group1_var2  res  1    ;group1_var2 located at 0x21,
                        ;initialized with 0.

group2  IDATA              ;Declaration of group2 data.The
                        ;addresses for variables under
                        ;this data section are allocated
                        ;automatically by the linker.

    group2_var1  db  1,2,3,4 ;4 bytes in RAM are reserved.
    group2_var2  dw  0x1234  ;1 word in RAM is reserved.

RST      CODE      0x0      ;The code section named RST
                        ;is placed at program memory
                        ;location 0x0. The next two
                        ;instructions are placed in
                        ;code section RST.
    pagesel  start
    goto     start          ;Jumps to the location labelled
                        ;'start'.

PGM      CODE              ;This is the begining of the
                        ;code section named PGM. It is
                        ;a relocatable code section
                        ;since no absolute address is
                        ;given along with directive CODE.

start
    goto $                  ;Go to current line (loop here)
end
```

4.36 `idata_acs`——在快速操作 RAM 中开始目标文件已初始化的数据段（PIC18 MCU）

4.36.1 语法

```
[label] idata_acs [RAM_address]
```

4.36.2 说明

此伪指令声明在快速操作 RAM 中开始一段已初始化的数据。如果未指定 `label`，此数据段被命名为 `.idata_acs`。起始地址被初始化为指定的地址，如果没有指定地址，则在链接时分配起始地址。用户不能将代码放置在此段。

链接器将为在 `idata` 段中指定的每个字节生成查找表的项。然后必须链接或包含相应的初始化代码。在 MPLINK 链接器样本应用程序示例中可以找到可使用的而且在需要时可修改的初始化代码示例。

可以使用 `res`、`db` 和 `dw` 伪指令为变量保留空间。`res` 会产生初始值零。`db` 会初始化 RAM 的连续字节。`dw` 会按照从低字节到高字节的顺序一次初始化一个字，初始化 RAM 的连续字节。

4.36.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

使用此伪指令初始化变量，或者使用 `udata` 伪指令，然后用代码中的值初始化变量。建议总是初始化变量。依赖 RAM 初始化会导致问题，特别是在使用仿真器的时候，由于仿真器和实际部件之间的行为差异会导致问题发生。

4.36.4 相关伪指令

```
extern global udata udata_acs udata_ovr udata_shr
```

4.36.5 简单示例

```
idata_acs
LimitL dw 0
LimitH dw D'300'
Gain dw D'5'
Flags db 0
String db 'Hi there!'
```

4.37 `__idlocs`——设置处理器 ID 单元

注： `idlocs` 的前面有两根下划线。

4.37.1 语法

```
__idlocs expr  
__idlocs addr, expr（仅用于 PIC18 器件）
```

4.37.2 说明

对于 PIC12 和 PIC16 器件，`__idlocs` 将四个 ID 单元设置为 `expr` 的十六进制值。例如，如果 `expr` 等于 1AF，则第一个（低地址）ID 单元为 0、第二个为 1、第三个为 10 而第四个为 15。

对于 PIC18 器件而言，`__idlocs` 在单元 `addr` 将双字节器件 ID 设置为 `expr` 的十六进制值。

在使用此伪指令之前，必须通过命令行、`list` 伪指令或 `processor` 伪指令声明处理器。

4.37.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令并不经常使用，但是它为序列化器件提供了一种简便方法。`__idlocs` 可以被编程器读取。PIC18 器件可以在运行时读取此值，但是 PIC12/16 器件不能。

4.37.4 相关伪指令

```
__config config list processor
```

4.37.5 简单示例

例 1：PIC16 器件

```
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.
__idlocs 0x1234        ;Sets device ID to 1234.
```

例 2: PIC18 器件

注: 根据编程规范, `__idlocs` 的最高有效半字节始终为 0x0。

```
#include p18f452.inc    ;Include standard header file
                        ;for the selected device.

__idlocs    _IDLOC0, 0x1 ;IDLOC register 0 will be
                        ;programmed to 1.
__idlocs    _IDLOC1, 0x2 ;IDLOC register 1 will be
                        ;programmed to 2.
__idlocs    _IDLOC2, 0x3 ;IDLOC register 2 will be
                        ;programmed to 3.
__idlocs    _IDLOC3, 0x4 ;IDLOC register 3 will be
                        ;programmed to 4.
__idlocs    _IDLOC4, 0x5 ;IDLOC register 4 will be
                        ;programmed to 5.
__idlocs    _IDLOC5, 0x6 ;IDLOC register 5 will be
                        ;programmed to 6.
__idlocs    _IDLOC6, 0x7 ;IDLOC register 6 will be
                        ;programmed to 7.
__idlocs    _IDLOC7, 0x8 ;IDLOC register 7 will be
                        ;programmed to 8.
```

4.38 if——开始条件汇编代码块

4.38.1 语法

首选语法:

```
if expr
```

支持的语法:

```
#if expr
```

```
.if expr
```

4.38.2 说明

开始执行条件汇编的代码块: 如果 *expr* 的值为 **TRUE**, 将汇编紧跟在 `if` 后的代码。否则, 将跳过后续的代码直到遇到 `else` 或 `endif` 伪指令。

值为 0 的表达式视为逻辑 **FALSE**。值为任何其他值的表达式视为逻辑 **TRUE**。`if` 和 `while` 伪指令根据表达式的逻辑值来运行。相关表达式的值为 **TRUE** 将保证返回一个非 0 的值, 值为 **FALSE** 的表达式则返回 0。

`if` 最多可嵌套 16 级。

4.38.3 使用

此伪指令在以下类型的代码中使用: 绝对代码或可重定位代码。欲知有关代码类型的信息, 请参见第 1.6 节“汇编器操作”。

此伪指令不是指令, 用于控制代码的汇编方式而不是代码运行时的行为。使用此伪指令进行条件汇编或者进行条件检查, 如检查是否产生错误消息。

4.38.4 相关伪指令

```
else endif
```


4.38.5 简单示例

```
if version == 100; check current version
    movlw 0x0a
    movwf io_1
else
    movlw 0x01a
    movwf io_2
endif
```

4.38.6 应用程序示例——if/else/endif

此程序演示 if、else 和 endif 汇编伪指令的实用程序。

```
#include p16f877a.inc    ;Include standard header file
                        ;for the selected device.

variable cfab           ;variable used to define
                        ;required configuration of
                        ;PORTA & PORTB

cfab set .1             ;Set config to decimal .1

RST      CODE      0x0    ;The code section named RST
                        ;is placed at program memory
                        ;location 0x0. The next two
                        ;instructions are placed in
                        ;code section RST.

        pagesel start    ;Jumps to the location labelled
        goto      start    ;'start'.

PGM      CODE           ;This is the begining of the
                        ;code section named PGM. It is
                        ;a relocatable code section
                        ;since no absolute address is
                        ;given along with directive CODE.

start
    banksel TRISA
    if cfab==0x0        ;If config==0x0 is true,
        clrw            ;assemble the mnemonics up to
        movwf TRISA     ;the directive 'else'. Set up PORTA
        movlw 0xff      ;as output.
        movwf TRISB

    else
        clrw            ;If config==0x0 is false,
        movwf TRISB     ;assemble the mnemonics up to
        movlw 0xff      ;the directive 'endif'. Set up PORTB
        movwf TRISA     ;as output.
    endif

    goto $              ;Go to current line (loop here)
end
```

4.39 ifdef——如果已经定义了符号则执行

4.39.1 语法

首选语法:

```
ifdef label
```

支持的语法:

```
#ifdef label
```

4.39.2 说明

如果前面已经定义了 *label* (通常通过执行 `#define` 伪指令或在 MPASM 汇编器命令行上设置值), 则获取条件路径。汇编将继续进行直到遇到匹配的 `else` 或 `endif` 伪指令为止。

4.39.3 使用

此伪指令在以下类型的代码中使用: 绝对代码或可重定位代码。欲知有关代码类型的信息, 请参见第 1.6 节 “汇编器操作”。

此伪指令不是指令, 用于控制代码的汇编方式而不是代码运行时的行为。在调试过程中使用此伪指令删除或添加代码, 而无需对大块的代码进行注释。

4.39.4 相关伪指令

```
#define #undef else endif ifndef
```

4.39.5 简单示例

```
#define testing 1      ; set testing "on"
:
ifdef testing
    <execute test code> ; this path would be executed.
endif
```

4.39.6 应用程序示例——ifdef

```
#include p16f877a.inc
#define AlternateASM ;Comment out with ; if extra
                        ;features not desired.

#ifdef AlternateASM
MyPort equ PORTC        ;Use Port C if AlternateASM defined.
MyTris equ TRISC         ;TRISC must be used to set data
                        ;direction for PORTC.

#else
MyPort equ PORTB        ;Use Port B if AlternateASM not defined.
MyTris equ TRISB        ;TRISB must be used to set data
                        ;direction for PORTB.

#endif

banksel MyTris
clrf    MyTris           ;Set port to all outputs.
banksel MyPort           ;Return to bank used for port.
movlw   55h              ;Move arbitrary value to W reg.
movwf   MyPort           ;Load port selected with 55h.
end
```

4.39.7 应用程序示例 2——`ifdef`

此程序使用控制伪指令 `#define` 以及 `ifdef`、`else` 和 `endif` 伪指令有选择地汇编代码，这些代码将用于仿真器或实际器件。控制伪指令 `#define` 用于创建“标志”以表明如何汇编代码（代码用于仿真器或实际器件）。

```
#include p18f452.inc
#define EMULATED      ;Comment out with ; if actual part
.
.
INIT
#ifdef EMULATED      ;If emulator used, add lines of
movlw 0xb0           ;initialization code to work around
movwf 0xf9c          ;table read limitation.
#endif
.
.
```

4.40 `ifndef`——如果未定义符号则执行

4.40.1 语法

首选语法：

```
ifndef label
```

支持的语法：

```
#ifndef label
```

4.40.2 说明

如果前面没有定义 `label` 或已经通过执行 `#undef` 伪指令解除定义，则伪指令后的代码将被汇编。汇编将被使能或禁止直到遇到下一条匹配的 `else` 或 `endif` 伪指令。

4.40.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令不是指令，用于控制代码的汇编方式而不是代码运行时的行为。在调试过程中使用此伪指令删除或添加代码，无需对大块的代码进行注释。

4.40.4 相关伪指令

```
#define #undef else endif ifdef
```

4.40.5 简单示例

```
#define testing1      ; set testing on
:
#undef testing1       ; set testing off
ifndef testing ; if not in testing mode
:                ; execute this path
endif
end               ; end of source
```

4.40.6 应用程序示例——`ifndef`

```
#include p16f877a.inc
#define UsePORTB      ;Comment out with ; to use PORTC

#ifndef UsePORTB
MyPort equ PORTC      ;Use Port C if UsePORTB not defined.
MyTris equ TRISC       ;TRISC must be used to set data
                        ;direction for PORTC.

#else
MyPort equ PORTB      ;Use Port B if UsePORTB defined.
MyTris equ TRISB       ;TRISB must be used to set data
                        ;direction for PORTB.

#endif

banksel MyTris
clrf    MyTris         ;Set port to all outputs.
banksel MyPort         ;Return to bank used for port.
movlw   55h            ;Move arbitrary value to W reg.
movwf   MyPort         ;Load port selected with 55h.
end
```

4.41 `#include`——包含额外的源文件

4.41.1 语法

首选语法:

```
#include include_file
#include "include_file"
#include <include_file>
```

支持的语法:

```
include include_file
include "include_file"
include <include_file>
```

4.41.2 说明

指定文件作为源代码读取。效果就相当于将包含文件的整个文本插入 `include` 语句所在单元的文件。在文件末尾，将从原始源文件继续汇编源代码。最多允许 **5** 层嵌套。最多允许 **255** 个包含文件。

`include_file` 中包含的任何空间都必须用引号或尖括号括起来。如果指定了完全合格的路径，就只会搜索到该路径。否则，搜索的顺序是：

- 当前工作目录
- 源文件目录
- MPASM 汇编器可执行文件目录

4.41.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

应该使用一次 `include` 伪指令来包含选定处理器的标准头文件。此文件包含为特定处理器定义的寄存器、位和其他名称，这样就不需要在代码中定义这些内容了。

4.41.4 相关伪指令

```
#define #undef
```

4.41.5 简单示例

```
#include p18f452.inc ;standard include file
#include "c:\Program Files\mydefs.inc" ;user defines
```

4.42 list——列表选项

4.42.1 语法

```
list [list_option, ..., list_option]
```

4.42.2 说明

在一行中只出现 `list` 时，此伪指令的作用是，如果列表输出先前是关闭的话，打开列表输出。否则，可提供以下列表选项之一来控制汇编进程或列表文件的格式。

4.42.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

下面对可以与 `list` 伪指令一起使用的选项进行了说明。

选项	默认	说明
<code>b=nnn</code>	8	设置制表符空间。
<code>c=nnn</code>	132	设置列宽。
<code>f=format</code>	INHX8M	设置 hex 文件输出格式。 <i>format</i> 可以是 INHX32、INHX8M 或 INHX8S 注：Hex 文件格式在 MPLAB IDE 中设置（Build Options 对话框）。
<code>free</code>	FIXED	使用自由格式的分析算法。提供向后兼容性。
<code>fixed</code>	FIXED	使用固定格式的分析算法。
<code>mm={ON OFF}</code>	On	在列表文件中显示存储器映射。
<code>n=nnn</code>	60	设置每页的行数。
<code>p=type</code>	无	设置处理器类型，例如 PIC16F54。也可参见 processor。 注：处理器类型在 MPLAB IDE 中设置（Configure>Device）。
<code>pe=type</code>	无	设置处理器类型并使能扩展的指令集，例如 LIST pe=PIC18F4620 只对于支持扩展的指令集的处理器和通用处理器 PIC18XXX 有效。可通过命令行选项 /y-（禁止扩展的指令集）来改写。 注：处理器类型在 MPLAB IDE 中设置（Configure>Device）。
<code>r=radix</code>	hex	设置基数：hex、dec 或 oct。参见 radix。
<code>st={ON OFF}</code>	On	在列表文件中显示符号表。
<code>t={ON OFF}</code>	Off	截断列表中的行（否则换行）。
<code>w={0 1 2}</code>	0	设置消息级别。也可参见 errorlevel。

选项	默认	说明
x={ON OFF}	On	打开或关闭宏扩展。

注： 在默认情况下，所有列表选项的值为十进制数。

4.42.4 相关伪指令

errorlevel expand noexpand nolist processor radix

4.42.5 简单示例

将处理器类型设置为 PIC18F452、hex 文件输出格式设置为 INHX32 并将基数设置为十进制。

list p=18f452, f=INHX32, r=DEC

4.43 local——声明局部宏变量

4.43.1 语法

首选语法：

local label[,label...]

支持的语法：

.local label[,label...]

4.43.2 说明

声明将指定的数据元素看作宏的局部元素。label 可以与在宏定义之外声明的其他标号相同，两个标号之间不会有冲突。

如果递归调用宏，每个调用都会有自己的局部复制。

4.43.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

如果多次使用宏并且宏中有标号，如果不使用此伪指令，就会出现“Duplicate Label”（标号重复）错误。

4.43.4 相关伪指令

endm macro

4.43.5 简单示例

```
<main code segment>
:
:
len    equ 10           ; global version
size   equ 20           ; note that a local variable
                        ; may now be created and modified
test   macro size
        local len, label ; local len and label
len     set size         ; modify local len
label   res len          ; reserve buffer
len     set len-20
endm                                ; end macro
```

4.43.6 应用程序示例——local

此代码演示 local 伪指令实用程序，此伪指令声明指定的数据元素被看作宏的局部元素。

```
#include p16f877a.inc    ;Include standard header file
                        ;for the selected device.

incr equ 2              ;Assembler variable incr is set
                        ;equal to 2.

add_incr macro          ;Declaration of macro 'add_incr'.
    local incr          ;Local assembler variable 'incr'.
```

主代码中使用了相同的名称 incr，在主代码中它的值为 2。

```
incr set 3              ;Local 'incr' is set to 3, in
                        ;contrast to 'incr' value
                        ;of 2 in main code.

    clrw                ;w register is set to zero
    addlw incr          ;w register is added to incr and
                        ;result placed back
endm                    ;in w register.

RST      CODE          0x0    ;The code section named RST
                        ;is placed at program memory
                        ;location 0x0. The next two
                        ;instructions are placed in
                        ;code section RST.

    pagesel start
    goto    start        ;Jumps to the location labelled
                        ;'start'.

PGM      CODE          ;This is the begining of the
                        ;code section named PGM. It is
                        ;a relocatable code section
                        ;since no absolute address is
                        ;given along with directive CODE.

start
    clrw                ;W register set to zero.

    addlw incr          ;W register is added with the
                        ;value of incr which is now equal
                        ;to 2.

    add_incr            ;W register is added with the
                        ;value of incr which is now equal
                        ;to 3 (value set locally in the
                        ;macro add_incr).

    clrw                ;W register is set to zero again.

    addlw incr          ;incr is added to W register and
                        ;result placed in W register.
                        ;incr value is again 2, not
                        ;affected by the value set in the
                        ;macro.

    goto $              ;Go to current line (loop here)
end
```

4.44 macro——声明宏定义

4.44.1 语法

```
label macro [arg, ..., arg]
```

4.44.2 说明

宏是一系列指令，可以使用一个宏调用将这一系列指令插入到汇编源代码中。必须首先定义宏，然后才能在后续的源代码中引用它。

从源代码行中读取的参数将被存储在一个链接的列表中，然后对它们进行计数。参数的最大数量为一行源代码除标号和宏项目之后所能容纳的数量。因此，源代码行的长度最大值应为 200 个字节。

一个宏可以调用另一个宏或者递归调用自己。宏调用的最大嵌套数量为 16 层。

欲知更多信息，请参见第 7 章“宏语言”。

4.44.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

4.44.4 相关伪指令

```
ENDM EXITM LOCAL
```

4.44.5 简单示例

```
;Define macro Read
Read macro device, buffer, count
    movlw device
    movwf ram_20
    movlw buffer ; buffer address
    movwf ram_21
    movlw count ; byte count
    call sys_21 ; subroutine call
endm
:
;Use macro Read
Read 0x0, 0x55, 0x05
```

4.44.6 应用程序示例——macro/endm

此代码演示用于定义宏的 macro 伪指令的实用程序。

```
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.

result equ 0x20        ;Assign value 20H to label
                        ;result.

ORG 0x0000             ;The following code will be placed
                        ;in reset address 0.

goto start             ;Jump to an address whose label is
                        ;'start'.

add MACRO num1,num2    ;'add' is a macro. The values of
                        ;'num1' and 'num2' must be passed
                        ;to this macro.
```



```

movlw  num1           ;Load W register with a literal
                       ;value assigned to the label
                       ;'num1'.

movwf  result         ;Load W register to an address
                       ;location assigned to the label
                       ;'result'.

movlw  num2           ;Load W register with a literal
                       ;value assigned to the label
                       ;'num2'.

addwf  result         ;Add W register with the memory
                       ;location addressed by 'result'
                       ;and load the result back to
                       ;'result'.

endm                  ;end of 'add' MACRO

ORG    0x0010         ;Main program starts at 10H.

start                  ;The label 'start' is assigned an
                       ;address 10H.

add    .100,.90       ;Call 'add' MACRO with decimal
                       ;numbers 100 and 90 assigned to
                       ;'num1' and 'num2' labels,
                       ;respectively. 100 and 90 will be
                       ;added and the result will be in
                       ;'result'.

end

```

4.45 `__maxram`——定义最大 RAM 单元

注： `maxram` 的前面有两根下划线。

4.45.1 语法

`__maxram expr`

4.45.2 说明

`__maxram` 和 `__badram` 伪指令一起标记对未用寄存器的访问。`__maxram` 定义有效 RAM 地址的绝对最大值并将有效 RAM 地址映射初始化为等于或小于 `expr` 的所有有效地址。`expr` 必须大于或等于页 0 的最大 RAM 地址并且小于 1000H。此伪指令设计与 `__badram` 伪指令一起使用。一旦使用了 `__maxram` 伪指令，就启用了严格的 RAM 地址检查，此 RAM 地址检查使用由 `__badram` 指定的 RAM 映射。

`__maxram` 可在源文件中多次使用。每次使用都会重定义有效 RAM 地址的最大值并重置 RAM 映射到所有单元。

4.45.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令在用户代码中不常使用，因为 RAM 和 ROM 的细节问题由包含文件 (*.inc) 或链接描述文件 (*.lkr) 处理。

4.45.4 相关伪指令

`__badram`

4.45.5 简单示例

参见 `__badram` 的示例。

4.46 `__maxrom`——定义最大 ROM 单元

注: <code>maxrom</code> 的前面有两根下划线。

4.46.1 语法

`__maxrom expr`

4.46.2 说明

`__maxrom` 和 `__badrom` 伪指令一起标记对未用寄存器的访问。`__maxrom` 定义有效 ROM 地址的绝对最大值并将有效 ROM 地址映射初始化为等于和小于 `expr` 的所有有效地址。`expr` 必须大于或等于目标器件的最大 ROM 地址。此伪指令设计与 `__badrom` 伪指令一起使用。一旦使用了 `__maxrom` 伪指令，就启用了严格的 ROM 地址检查，ROM 地址检查使用由 `__badrom` 指定的 ROM 映射。

`__maxrom` 可在源文件中多次使用。每次使用都会重定义有效 ROM 地址的最大值并重置 ROM 映射到所有单元。

4.46.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令在用户代码中不常使用，因为 RAM 和 ROM 的细节问题由包含文件 (*.inc) 或链接描述文件 (*.lkr) 处理。

4.46.4 相关伪指令

`__badrom`

4.46.5 简单示例

参见 `__badrom` 的示例。

4.47 `messg`——创建用户定义的消息

4.47.1 语法

`messg "message_text"`

4.47.2 说明

导致在列表文件中显示报告消息。消息文本最多可以有 80 个字符。发出 `messg` 伪指令不会设置任何错误返回代码。

4.47.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

可以使用此伪指令生成任何需要的消息。它在条件汇编中很有用处，用以在汇编程序中验证编译了哪些代码。

4.47.4 相关伪指令

error

4.47.5 简单示例

```
mssg_macro macro
    mssg "mssg_macro-001 invoked without argument"
endm
```

4.47.6 应用程序示例——mssg

此程序演示了 mssg 汇编器伪指令，它将消息设置为显示在列表文件和错误文件中。

```
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.

variable baudrate      ;variable used to define
                        ;required baud rate

baudrate set D'5600'    ;Enter the required value of
                        ;baud rate here.

if (baudrate!=D'1200')&&(baudrate!=D'2400')&&
    (baudrate!=D'4800')&&(baudrate!=D'9600')&&
    (baudrate!=D'19200')
    error "Selected baud rate is not supported"
    mssg "only baud rates 1200,2400,4800,9600 & 19200 Hz "&&
        "are supported"
endif
```

如果所选择的波特率是除 1200、2400、4800、9600 或 19200 Hz 之外的其他波特率，if-endif 代码将输出 error 和 mssg。

```
RST      CODE      0x0      ;The code section named RST
                        ;is placed at program memory
                        ;location 0x0. The next two
                        ;instructions are placed in
                        ;code section RST.

        pagesel  start
        goto     start      ;Jumps to the location labelled
                        ;'start'.

PGM      CODE

                        ;This is the begining of the
                        ;code section named PGM. It is
                        ;a relocatable code section
                        ;since no absolute address is
                        ;given along with directive CODE.

start
        goto $              ;Go to current line (loop here)
end
```

4.48 noexpand——关闭宏扩展

4.48.1 语法

noexpand

4.48.2 说明和用法

在列表文件中关闭宏扩展。

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

4.48.3 相关伪指令

expand

4.49 nolist——关闭列表输出

4.49.1 语法

nolist

4.49.2 说明和用法

关闭列表文件输出。

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

4.49.3 相关伪指令

list

4.50 org——设置程序起始处

4.50.1 语法

[*label*] org *expr*

4.50.2 说明

将后续代码的程序起始处设置在由 *expr* 定义的地址。如果指定了 *label*，它将给出 *expr* 的值。如果未指定 org 伪指令，将从地址 0 开始生成代码。

对于 PIC18 器件，仅允许偶数 *expr* 值。

当生成目标文件时，org 伪指令被解释为用内部生成的名称引入一个绝对 CODE 段。如下例：

```
L1: org 0x200
```

被解释为：

```
.scnname CODE 0x200  
L1:
```

其中 .scnname 由汇编器生成，它将与此文件中前面生成的名称均不同。

4.50.3 使用

此伪指令在以下类型的代码中使用：绝对代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

`org` 通常在单个文件的汇编程序中使用，用于当代码需要被放置在一个特殊单元中时。通常使用的值为 `0x0`（复位）、`0x4`（PIC16 器件中断向量）、`0x8`（PIC18 器件高优先级中断向量）和 `0x18`（PIC18 器件低优先级中断向量）。

4.50.4 相关伪指令

```
fill res end
```

4.50.5 简单示例

```
int_1 org 0x20
    ; Vector 20 code goes here
int_2 org int_1+0x10
    ; Vector 30 code goes here
```

4.50.6 PIC16 应用程序示例——`org`

此示例演示了 `org` 伪指令的用法。代码从由 **`org address`** 指定的地址开始生成。此伪指令也可以指定数据表的起始处。当 PICmicro 器件带有 EE 数据闪存时，可以将数据表放在程序存储器区域或 EE 数据存储区域。

```
#include p16f877a.inc    ;Include standard header file
                        ;for the selected device.

org    0x0000           ;The following code will be
                        ;placed in reset address 0.
goto   Main             ;Jump to an address whose label
                        ;is 'Main'.

org    0x0004           ;The following code will be
                        ;placed in reset address 4.
goto   int_routine      ;Jump to an address whose label
                        ;is 'int_routine'.

org    0x0010           ;The following code section will
                        ;placed starting from address 10H.
Main
    ;
    ;
    ;
goto   Main             ;Loop back to 'Main'.

org    0x0100           ;The following code section will
                        ;be placed starting from address
                        ;100H.

int_routine
    ;
    ;
    ;
    ;Write your interrupt service
    ;routine here.
    ;Return from interrupt.
retfie

org    0x1000           ;You can create a data or
                        ;character table starting from
                        ;any address in program memory.
                        ;In this case the address is
                        ;1000h.

ch_tbl1 da  "PICwithFLASH" ;6 program memory locations
                        ;(starting from 1000h) will
```

```
                                ;be filled with six 14-bit
                                ;packed numbers, each
                                ;representing two 7-bit ASCII
                                ;characters.

org    0x2100                    ;The absolute address 2100h is
                                ;mapped to the 0000 location of
                                ;EE data memory in PIC16Fxxx.
                                ;You can create a data or
                                ;character table starting from
                                ;any address in EE data memory.

ch_tbl2 de    "PICwithFLASH"    ;12 EE data memory locations
                                ;(starting from 0) will be
                                ;filled with 12 ASCII
                                ;characters.

end
```

4.50.7 PIC18 应用程序示例——org

此示例演示了 `org` 伪指令的用法。代码从由 `org address` 指定的地址开始生成。此伪指令也可以指定数据表的起始处。当 PICmicro 器件带有 EE 数据闪存时，可以将数据表放在程序存储器区域或 EE 数据存储区域。

```
#include p18f452.inc    ;Include standard header file
                        ;for the selected device.

org    0x0000            ;The following code will be
                        ;programmed in reset address 0.
goto   Main              ;Jump to an address whose label is
                        ;'Main'.

org    0x0008            ;The following code will be
                        ;programmed in high priority
                        ;interrupt address 8.
goto   int_hi            ;Jump to an address whose label is
                        ;'int_hi'.

org    0x0018            ;The following code will be
                        ;programmed in low priority
                        ;interrupt address 18h.
goto   int_lo            ;Jump to an address whose label is
                        ;'int_lo'.

org    0x0020            ;The following code section will
                        ;be programmed starting from
                        ;address 20H.
Main
;                        ;Write your main program here.
;
;
goto   Main              ;Loop back to 'Main'

org    0x0100            ;The following code section will
                        ;be programmed starting from
                        ;address 100H.

int_hi
;
```

```

;                                ;Write your high priority
;                                ;interrupt service routine here.
retfie                          ;Return from interrupt.

org    0x0200                   ;The following code section will
                                ;be programmed starting from
                                ;address 200H.

int_lo
;
;                                ;Write your low priority
;                                ;interrupt service routine here.
retfie                          ;Return from interrupt.

org    0x1000                   ;You can create a data or
                                ;character table starting from any
                                ;address in program memory. In
                                ;this case the address is 1000h.

ch_tbl1 db  "PICwithFLASH"

end

```

4.51 page——在列表文件中插入换页符

4.51.1 语法

```
page
```

4.51.2 说明和用法

在列表文件中插入换页符。

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

4.51.3 相关伪指令

```
list subtitle title
```

4.52 pagesel——生成页面选择代码（PIC10/12/16 MCU）

4.52.1 语法

```
pagesel label
```

4.52.2 说明

该指令指示链接器生成页面选择代码，为包含指定 *label* 的页设置页位。应该只指定一个 *label*。不能对 *label* 执行操作。必须已经在前面定义了 *label*。

链接器将生成相应的页选择代码。对于指令宽度为 12 位的（PIC10F 和一些 PIC12/PIC16）器件，将生成针对 STATUS 寄存器的相应的位置位 / 清零指令。对于指令宽度为 14 位的（大多数 PIC12/PIC16）器件，将使用 BSF 和 BCF 命令来调整 PCLATH 寄存器的第三位和第四位。如果器件仅包含一页程序存储器，将不会生成代码。

对于 PIC18 器件，此命令没有任何用处，因为器件不使用分页。

4.52.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令省却了您手工更改页位代码的操作。而且，由于它自动生成代码，代码会更便于移植。

如果您正在使用可重定位代码并且器件的程序存储器大于 2K（或 12 位指令宽器件的程序存储器大于 0.5K），建议使用此伪指令，特别是在代码必须在两个或多个代码段之间跳转的时候。

如果您希望显示出 RETLW 表的起始地址或者计算 GOTO 的跳转表，就必须使用 pageselw 伪指令。

4.52.4 相关伪指令

bankisel banksel

4.52.5 简单示例

```
pagesel GotoDest
goto    GotoDest
:
pagesel CallDest
call    CallDest
```

4.52.6 应用程序示例——pagesel

此程序演示 pagesel 伪指令，它生成相应的代码来置位 / 清零 PCLATH 位。这样就可以更简单地使用分页的存储器，如 PIC16 器件中的存储器。

```
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.

RST      CODE      0x0      ;The code section named RST
                        ;is placed at program memory
                        ;location 0x0. The next two
                        ;instructions are placed in
                        ;code section RST.

      pagesel start
      goto    start      ;Jumps to the location labelled
                        ;'start'.

PGM0     CODE      0x500    ;The code section named PGM1 is
                        ;placed at 0x500.

start
      pagesel page1_pgm    ;address bits 12 & 11 of
                        ;page1_pgm are copied to PCLATH
                        ;4 & 3 respectively.

      goto    page1_pgm

PGM1     CODE      0x900    ;The code section named PGM1 is
                        ;placed at 0x900. Label
                        ;page1_pgm is located in this
page1_pgm ;code section.
      goto $      ;Go to current line (loop here)
      end
```


4.53 pageselw——使用 WREG 命令生成页选择代码（PIC10/12/16 MCU）

4.53.1 语法

```
pageselw label
```

4.53.2 说明

此指令指示链接器生成页面选择代码，为包含指定标号的页设置页位。应该只指定一个 *label*。不能对 *label* 执行操作。必须已经在前面定义了 *label*。

链接器将生成相应的页选择代码。对于指令宽度为 12 位的（PIC10F 和一些 PIC12/PIC16）器件，将生成针对 STATUS 寄存器的相应的位置位 / 清零指令。对于 14 位指令宽度（大多数 PIC12/PIC16）的器件，将生成 MOVLW 和 MOVWF 指令来修改 PCLATH。如果器件仅包含一页程序存储器，将不会生成代码。

对于 PIC18 器件，当器件不使用页时，此命令没有任何用处。

4.53.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令省却了您手动更改页位代码的操作。而且，由于它自动生成代码，代码会更便于移植。

如果您正在使用可重定位代码并且器件的程序存储器大于 2K（或 12 位指令宽器件的程序存储器大于 0.5K），建议使用此伪指令，特别是在代码必须在两个或多个代码段之间跳转的时候。

如果希望表示 RETLW 表的起始地址或计算 GOTO 的跳转表，就必须使用此伪指令而不是 pagesel。当 PC 增加了 8 位偏移量时，只会将相应的值载入 PC 的高 5 位。仍然必须考虑 256 字边界，如应用笔记 AN586，“*Macros for Page and Bank Switching*”中的讨论。

4.53.4 相关伪指令

```
bankisel banksel
```

4.53.5 简单示例

```
pageselw CommandTableStart ;Get the byte read and use it to
movlw CommandTableStart    ;index into our jump table.If
addwf Comm.RxTxByte,w      ;we crossed a 256-byte boundary,
btfsc STATUS,C             ;then increment PCLATH. Then load the
incf PCLATH,f              ;program counter with computed goto.
movwf PCL

CommandTableStart
goto GetVersion            ;0x00 - Get Version
goto GetRTSample           ;0x01 - Get Real Time sample
goto Configure             ;0x02 - stub
goto Go                    ;0x03 - stub
goto ReadBuffer            ;0x04 - Read Buffer, just sends Vout
goto AreYouThroughYet     ;0x05
goto CommDone              ;0x06
goto CommDone              ;0x07
```

4.54 processor——设置处理器类型

4.54.1 语法

`processor processor_type`

4.54.2 说明

将处理器类型设置为 `processor_type`。

4.54.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令不常使用，因为在 MPLAB IDE 中可以设置处理器（[Configure>Device](#)）。如果必须在代码中设置处理器类型，请使用 `processor` 或 `list` 伪指令选项 `p=` 来设置处理器类型。

4.54.4 相关伪指令

`list`

4.54.5 简单示例

```
processor 16f877a    ;Sets processor to PIC16F877A
```

4.55 radix——指定默认基数

4.55.1 语法

`radix default_radix`

4.55.2 说明

设置数据表达式的默认基数。默认基数是 16。有效的基数值为：

- `hex`——16（基 16）
- `dec`——10（基 10）
- `oct`——8（基 8）

您也可以使用 `list` 伪指令来指定基数。要指定一个常数的基数，请参见第 3.4 节“数字常数和基数”。

4.55.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

很多程序都使用默认基数 16，所以不需要设置基数。但是，如果需要在程序中更改基数，就应该非常小心，因为 `radix` 声明后的所有数字值都将使用该基数值。欲知更多有关更改基数的含义的信息，请参见基数示例。

使用 `radix` 伪指令或 `list` 伪指令选项 `r=` 来更改代码中的基数。

4.55.4 相关伪指令

`list`

4.55.5 简单示例

```
radix dec
```

4.55.6 应用程序示例 —— radix

此示例演示了用于数据表达的 radix 伪指令的用法。如果不声明，默认的基数是 16。

```
list r=dec           ;Set the radix as decimal.
#include p16f877a.inc ;Include standard header file
                    ;for the selected device.

movlw 50H           ;50 is in hex
movlw 0x50          ;Another way of declaring 50 hex
movlw 500           ;50 is in octal
movlw 50            ;50 is not declared as hex or
                    ;octal or decimal. So by default
                    ;it is in decimal as default radix
                    ;is declared as decimal.

radix oct           ;Use 'radix' to declare default
                    ;radix as octal.

movlw 50H           ;50 is in hex.
movlw 0x50          ;Another way of declaring 50 hex.
movlw .50           ;50 is in decimal.
movlw 50            ;50 is not declared as hex or
                    ;octal or decimal. So by default
                    ;it is in octal as default radix
                    ;is declared as octal.

radix hex           ;Now default radix is in hex.

movlw .50           ;50 is declared in decimal.
movlw 500           ;50 is declared in octal
movlw 50            ;50 is not declared as hex or
                    ;octal or decimal. So by default
                    ;it is in hex as default radix
                    ;is declared as hex.

end
```

4.56 res——保留存储器

4.56.1 语法

```
[label] res mem_units
```

4.56.2 说明

使存储器单元指针从当前的单元向前移在 *mem_units* 中指定的值。在可重定位代码中（使用 MPLINK 链接器），*res* 可被用于保留数据存储空间。在非可重定位代码中，*label* 被假设为一个程序存储器地址。

PIC12/16 MCU 用字来定义地址单元，PIC18 MCU 用字节来定义地址单元。

4.56.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

res 最常用的用途是在可重定位代码中存储数据。

4.56.4 相关伪指令

```
fill org equ cblock
```

4.56.5 简单示例

```
buffer res 64 ; reserve 64 address locations of storage
```

4.56.6 应用程序示例 —— **res**

此示例演示了 **res** 伪指令在开发可重定位代码方面的优势。该程序计算矩形的周长。矩形的长度和宽度将存储在缓冲器中，通过 **length** 和 **width** 可对这两个缓冲器寻址。计算得到的周长将被存储在一个双精度的缓冲器中，通过 **perimeter** 寻址。

```
#include p18f452.inc    ;Include standard header file
                        ;for the selected device.

UDATA                  ;This directive allows the
                        ;following data to be placed only
                        ;in the data area.

perimeter res 2         ;Two locations of memory are
                        ;reserved for the label
                        ;'perimeter'. Addresses of the
                        ;memory locations will be
                        ;allocated by MPLINK.

length    res 1         ;One location of memory is
                        ;reserved for the label 'length'.
                        ;Address of the memory location
                        ;will be allocated by MPLINK.

width     res 1         ;One location of memory is
                        ;reserved for the label 'width'.
                        ;Address of the memory location
                        ;will be allocated by MPLINK.

Start CODE 0x0000       ;Following code will be placed in
                        ;address 0.
```

此处伪指令 **code** 与 **org** 有相同的效果。但是 **org** 在 **MPASM** 汇编器中使用来生成绝对代码，而 **code** 则在 **MPLINK** 链接器中使用以生成目标文件。**code** 的不同之处还在于通常不指定地址；链接器在程序闪存和数据 **RAM** 存储器中处理空间分配。

```
goto PER_CAL           ;Jump to label PER_CAL

CODE                   ;CODE directive here dictates that
                        ;the following lines of code will
                        ;be placed in program memory, but
                        ;the starting address will be
                        ;decided by MPLINK linker.

PER_CAL
    clrf perimeter      ;Clear the buffers addressed by
    clrf perimeter+1    ;'perimeter'.
    movf    length,w    ;Move the data present in the
                        ;register addressed by 'length'
                        ;to 'w'.
    addwf   width,w     ;Add data in 'w' with data in the
                        ;register addressed by 'width'
```

```

movwf    perimeter        ;Move 'w' to the register
                        ;addressed by 'perimeter'.
incfsz   perimeter+1      ;Increment 'perimeter+1' if carry
                        ;is generated.
bcf       STATUS,C        ;Clear carry bit in STATUS
                        ;register.

rlf       perimeter+1
rlf       perimeter
incfsz    perimeter+1

```

前面的三行代码将把中间结果乘以（通过左移一位）2。

```

goto $          ;Go to current line (loop here)
end

```

4.57 set——定义汇编器变量

4.57.1 语法

首选语法：

```
label set expr
```

支持的语法：

```
label .set expr
```

4.57.2 说明

将由 *expr* 指定的有效 MPASM 汇编器表达式的值赋给 *label*。set 伪指令的功能与 equ 伪指令相似，不同之处在于设置的值后来会被其他 set 伪指令更改。

4.57.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

由于 set 伪指令的值会被后来的 set 伪指令更改，所以 set 在一个循环中定义变量时特别有用（如 while 循环）。

4.57.4 相关伪指令

```
equ variable while
```

4.57.5 简单示例

```

area    set 0
width   set 0x12
length  set 0x14
area    set length * width
length  set length + 1

```

4.57.6 应用程序示例 —— set/equ

此示例演示 set 伪指令的用法，用于创建仅可在 MPASM 汇编器表达式中使用的符号。用此伪指令创建的符号不在单片机上占用任何物理存储单元。

```

#include p16f877a.inc    ;Include standard header file
                        ;for the selected device.

perimeter set 0          ;The label 'perimeter' is
                        ;assigned value 0.

```

```
area set 0 ;The label 'area' is assigned
           ;value 0.
```

由 `set` 伪指令赋值的值以后可被重新赋值。

```
length equ 50H ;The label 'length' is assigned
           ;the value 50H.
width equ 25H ;The label 'width' is assigned
           ;the value 25H.
```

由 `equ` 伪指令进行的赋值以后不能被重新赋值。

```
perimeter set 2*(length+width) ;Both 'perimeter' and
area set length*width ;'area' values are
                        ;reassigned.
end
```

4.58 `space`——插入空白列表行

4.58.1 语法

首选语法:

```
space expr
```

支持的语法:

```
spac expr
```

4.58.2 说明和用法

在列表文件中插入 `expr` 数量的空白行。

此伪指令在以下类型的代码中使用: 绝对代码或可重定位代码。欲知有关代码类型的信息, 请参见第 1.6 节“汇编器操作”。

4.58.3 相关伪指令

```
list
```

4.58.4 简单示例

```
space 3 ;Inserts three blank lines
```

4.59 `subtitle`——指定程序副标题

4.59.1 语法

首选语法:

```
subtitle "sub_text"
```

支持的语法:

```
stitle "sub_text"
```

```
subtitl "sub_text"
```

4.59.2 说明和用法

sub_text 是用双引号引起来的 ASCII 字符串，长度为 60 个字符或以下。此伪指令创建第二个程序头行作为列表输出中的副标题。

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

4.59.3 相关伪指令

```
list title
```

4.59.4 简单示例

```
subtitle "diagnostic section"
```

4.60 title——指定程序标题

4.60.1 语法

```
title "title_text"
```

4.60.2 说明和用法

title_text 是加上双引号的可显示 ASCII 字符串。它的长度必须小于等于 60 个字符。此伪指令在列表文件中创建在每页的最顶行使用的文本。

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

4.60.3 相关伪指令

```
list subtitle
```

4.60.4 简单示例

```
title "operational code, rev 5.0"
```

4.61 udata——开始目标文件中未初始化的数据段

4.61.1 语法

```
[label] udata [RAM_address]
```

4.61.2 说明

此伪指令声明开始一段未初始化的数据。如果未指定标号，此段被命名为 `.udata`。起始地址被初始化为指定的地址，如果没有指定地址，则在链接时分配起始地址。此段不会生成代码。应该使用 `res` 伪指令来为数据保留空间。

注： 相同源文件的两段不允许同名。

4.61.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

对于可重定位代码，此伪指令用于创建数据（RAM）段。对于绝对代码不使用此伪指令。使用伪指令 `equ` 或 `cblock`。

4.61.4 相关伪指令

```
extern global idata udata_acs udata_ovr udata_shr
```

4.61.5 简单示例

```
        udata
Var1    res 1
Double res 2
```

4.61.6 应用程序示例 —— `udata`

此程序演示了 `udata` 伪指令，它声明开始一段未初始化的数据。`udata` 不设置（初始化）此变量的起始值，所以您必须在代码中设置此变量的起始值。

```
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.

group1 udata 0x20      ;group1 data stored at locations
                        ;starting at 0x20.
        group1_var1 res 1 ;group1_var1 located at 0x20.
        group1_var2 res 1 ;group1_var2 located at 0x21.

group2 udata           ;Declaration of group2 data.The
                        ;addresses for variables under
        group2_var1 res 1 ;this data section are allocated
        group2_var2 res 1 ;automatically by the linker.

RST      CODE      0x0      ;The code section named RST
                        ;is placed at program memory
                        ;location 0x0. The next two
                        ;instructions are placed in
                        ;code section RST.
        pagesel start      ;Jumps to the location labelled
        goto      start      ;'start'.

PGM      CODE           ;This is the begining of the
                        ;code section named PGM. It is
                        ;a relocatable code section
                        ;since no absolute address is
                        ;given along with directive CODE.

start
        banksel group1_var1
        clrf group1_var1
        clrf group1_var2

        banksel group2_var1
        clrf group2_var1
        clrf group2_var2

        goto $              ;Go to current line (loop here)
end
```


4.62 udata_acs——开始目标文件未初始化快速操作的数据段（PIC18 MCU）

4.62.1 语法

```
[label] udata_acs [RAM_address]
```

4.62.2 说明

此伪指令声明开始一段未初始化快速操作的数据。如果未指定 *label*，此数据段被命名为 `.udata_acs`。起始地址被初始化为指定的地址，如果没有指定地址，则在链接时分配起始地址。此伪指令用于声明在 PIC18 器件的快速操作 RAM 中分配的变量。此段不会生成代码。应该使用 `res` 伪指令来为数据保留空间。

注： 相同源文件中的两段不允许同名。

4.62.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令与 `udata` 类似，不同的是它只能用于 PIC18 器件并且只能将变量放入快速操作 RAM。PIC18 器件有一个 RAM 区域被称为快速操作 RAM。不管存储区选择寄存器（BSR）指向哪个存储区，都可以使用快速操作存储器中的变量。这对于经常使用的变量以及全局变量非常有用。

4.62.4 相关伪指令

```
extern global idata udata udata_ovr udata_shr
```

4.62.5 简单示例

```
        udata_acs
Var1    res 1
Double res 2
```

4.62.6 应用程序示例 —— udata_acs

此程序演示了 `udata_acs` 伪指令。此伪指令声明开始一段未初始化的数据。

```
#include p18f452.inc    ;Include standard header file
                        ;for the selected device.

group1 udata_acs 0x20 ;group1 data stored at access
                        ;RAM locations starting at 0x20.
    group1_var1 res 1 ;group1_var1 located at 0x20.
    group1_var2 res 1 ;group1_var2 located at 0x21.

group2 udata_acs      ;Declaration of group2 data.The
                        ;addresses for data under this
                        ;section are allocated
                        ;automatically by the linker.
    group2_var1 res 1 ;All addresses be will allocated
    group2_var2 res 1 ;in access RAM space only.

RST      CODE      0x0 ;The code section named RST
                        ;is placed at program memory
```

```
                                ;location 0x0. The instruction
                                ;'goto start' is placed in
                                ;code section RST.
goto    start                  ;Jumps to the location labelled
                                ;'start'.

PGM    CODE                    ;This is the beginning of the code
                                ;section named PGM. It is a
                                ;relocatable code section
                                ;since no absolute address is
                                ;given along with directive CODE.

start

    clrf    group1_var1,A      ;group1_var1 initialized to zero
    clrf    group1_var2,A      ;group1_var2 initialized to zero

    clrf    group2_var1,A      ;group2_var1 initialized to zero
    clrf    group2_var2,A      ;group2_var2 initialized to zero

    goto $                      ;Go to current line (loop here)
end
```

在上面的代码中，“A”指快速操作 RAM。A 这一指定可以由代码明确声明，但由于汇编器只要可能，会自动将代码放置在快速操作存储器中，所以不需要这样做。

4.63 udata_ovr——开始目标文件中覆盖的未初始化的数据段

4.63.1 语法

```
[label] udata_ovr [RAM_address]
```

4.63.2 说明

此伪指令声明开始一段被覆盖的未初始化的数据。如果未指定 *label*，此数据段被命名为 `.udata_ovr`。起始地址被初始化为指定的地址，如果没有指定地址，则在链接时分配起始地址。此段声明的空间被所有其他同名的 `udata_ovr` 段覆盖。由于它允许在同一个存储器单元中声明多个变量，因此这是一种声明临时变量的理想方法。此段不会生成代码。应该使用 `res` 伪指令来为数据保留空间。

注： 相同源文件的两段不允许同名。

4.63.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令与 `udata` 相类似，不同的是它允许通过用一个数据区域来“覆盖”另一个数据区域以实现数据空间的重用。它用于临时变量，因为每个数据段都可以改写（因而做到共用）相同的 RAM 地址单元。

4.63.4 相关伪指令

```
extern global idata udata udata_acs udata_shr
```

4.63.5 简单示例

```

Temps      udata_ovr
Temp1 res 1
Temp2 res 1
Temp3 res 1
Temps      udata_ovr
LongTemp1 res 2 ; this will be a variable at the
                ; same location as Temp1 and Temp2
LongTemp2 res 2 ; this will be a variable at the
                ; same location as Temp3

```

4.63.6 应用程序示例 —— udata_ovr

此程序演示了 udata_ovr。此伪指令声明开始一段被覆盖的未初始化的数据。

```

#include p16f877a.inc ;Include standard header file
                    ;for the selected device.

same_var  udata_ovr  0x20    ;Declares an overlaid
                             ;uninitialized data section
                             ;named 'same_var' starting at
                             ;location 0x20.

    var1  res  1

same_var  udata_ovr  0x20    ;Declares an overlaid
                             ;uninitialized data section
                             ;with the same name as the one
                             ;declared above. Thus variables
                             ;var1 and var2 are allocated
                             ;at the same address.

    var2  res  1

RST      CODE      0x0      ;The code section named RST
                             ;is placed at program memory
                             ;location 0x0. The next two
                             ;instructions are placed in
                             ;code section RST.

    pagesel start
    goto   start            ;Jumps to the location labelled
                             ;'start'.

PGM      CODE

                             ;This is the begining of the
                             ;code section named PGM. It is
                             ;a relocatable code section
                             ;since no absolute address is
                             ;given along with directive CODE.

start
    banksel var1            ;Any operation on var1 affects
    movlw  0xFF             ;var2 also since both variables
    movwf  var1             ;are overlaid.

    comf   var2

    goto $                  ;Go to current line (loop here)
end

```

4.64 udata_shr——开始目标文件中共享的未初始化的数据段（PIC12/16 MCU）

4.64.1 语法

```
[label] udata_shr [RAM_address]
```

4.64.2 说明

此伪指令声明开始一段共享的未初始化的数据。如果未指定 *label*，此数据段被命名为 *.udata_shr*。起始地址被初始化为指定的地址，如果没有指定地址，则在链接时分配起始地址。此伪指令用于声明在 **RAM** 中被分配的变量，此 **RAM** 是在所有 **RAM** 存储区中共享的（即未分区的 **RAM**）。此段不会生成代码。应该使用 *res* 伪指令来为数据保留空间。

注： 相同源文件的两段不允许同名。

4.64.3 使用

此伪指令在以下类型的代码中使用：可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令与 *udata* 类似，所不同的是它仅在具备可从多个存储区访问的存储器的器件上使用。*udata_shr* 段在链接描述文件中与 **SHAREBANK** 单元共同使用，而 *udata* 段在链接描述文件中和 **DATABANK** 单元共同使用。如需特定示例，请参见 PIC16F873A 的数据手册。

4.64.4 相关伪指令

```
extern global idata udata udata_acs udata_ovr
```

4.64.5 简单示例

```
Temps udata_shr
Temp1 res 1
Temp2 res 1
Temp3 res 1
```

4.64.6 应用程序示例 —— udata_shr

此程序演示了 *udata_shr* 伪指令。此伪指令声明开始一段共享的未初始化的数据。此伪指令用于声明在 **RAM** 中被分配的变量，此 **RAM** 是在所有 **RAM** 存储区中共享的（即未分区的 **RAM**）。

```
#include p16f877a.inc ;Include standard header file
                        ;for the selected device.

shared_data udata_shr ;Declares the beginning of a data
                        ;section named 'shared data',
                        ;which is shared by all banks.
                        ;'var' is the location which can
                        ;be accessed irrespective of
                        ;banksel bits.
                        var res 1

bank0_var udata 0x20 ;Declares beginning of a data
var0 res 1 ;section named 'bank0_var',
            ;which is in bank0. var0 is
            ;allocated the address 0x20.
```

```

bank1_var    udata    0xa0    ;Declares beginning of a data
                var1    res    1    ;section named 'bank1_var',
                                ;which is in bank1. var1 is
                                ;allocated the address 0xa0

bank2_var    udata    0x120   ;Declares beginning of a data
                var2    res    1    ;section named 'bank2_var',
                                ;which is in bank2. var2 is
                                ;allocated the address 0x120

bank3_var    udata    0x1a0   ;Declares beginning of a data
                var3    res    1    ;section named 'bank3_var',
                                ;which is in bank3. var3 is
                                ;allocated the address 0x1a0

RST          CODE      0x0     ;The code section named RST
                                ;is placed at program memory
                                ;location 0x0. The next two
                                ;instructions are placed in
                                ;code section RST.

    pagesel   start
    goto      start           ;Jumps to the location labelled
                                ;'start'.

PGM          CODE              ;This is the begining of the
                                ;code section named PGM. It is
                                ;a relocatable code section
                                ;since no absolute address is
                                ;given along with directive CODE.

start
    bankssel  var0              ;Select bank0.
    movlw    0x00
    movwf    var                ;var is accessible from bank0.

    bankssel  var1              ;Select bank1.
    movlw    0x01
    movwf    var                ;var is accessible from bank1
                                ;also.

    bankssel  var2              ;Select bank2.
    movlw    0x02
    movwf    var                ;var is accessible from bank2
                                ;also.

    bankssel  var3              ;Select bank3.
    movlw    0x03
    movwf    var                ;var is accessible from bank3
                                ;also.

    goto $                      ;Go to current line (loop here)
end

```

4.65 #undef——删除替换标号

4.65.1 语法

```
#undef label
```

4.65.2 说明

`label` 是前面用 `#define` 伪指令定义的标识符。所命名的符号将从符号表中删除。

4.65.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令经常和 `ifdef` 和 `ifndef` 伪指令一起使用，这两个伪指令在符号表中寻找是否存在某个项。

4.65.4 相关伪指令

`#define #include ifdef ifndef`

4.65.5 简单示例

```
#define length 20
:
#undef length
```

4.65.6 应用程序示例 —— `#define/#undef`

参见 `#define` 的示例。

4.66 `variable`——声明符号变量

4.66.1 语法

```
variable label[=expr][,label[=expr]...]
```

4.66.2 说明

创建在 MPASM 汇编器表达式中使用的符号。变量和常数在表达式中可以互换使用。

此变量伪指令创建的符号其功能与 `set` 伪指令创建的符号相似。区别就在于变量伪指令在声明符号的时候不需要将符号初始化。

`variable` 的值在一个操作数内不能更新。必须在分开的行上进行变量的赋值、递增和递减。

4.66.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令多用于条件汇编代码。

注： `variable` 不用于声明运行时的变量，而是用于声明由汇编器使用的变量。要创建运行时的变量，请参见 `res`、`equ` 或 `cblock`。

4.66.4 相关伪指令

`constant set`

4.66.5 简单示例

```
variable RecLength=64          ; Set Default
                                ; RecLength
constant BufLength=512        ; Init BufLength
```

```

        .                ; RecLength may
        .                ; be reset later
        .                ; in RecLength=128
        .                ;
constant MaxMem=RecLength+BufLength ;CalcMaxMem

```

4.66.6 应用程序示例 —— variable/constant

此示例演示 `variable` 伪指令的用法，用该伪指令创建仅可在 MPASM 汇编器表达式中使用的符号。用此伪指令创建的符号不能在单片机上占用任何物理存储单元。

```

#include p16f877a.inc ;Include standard header file
                    ;for the selected device.

variable perimeter=0 ;The symbol 'perimeter' is
                    ;initialized to 0
variable area        ;If a symbol is declared as
                    ;variable, then initialization
                    ;is optional, i.e. it may or may
                    ;not be initialized.

constant lngth=50H   ;The symbol 'lngth' is
                    ;initialized to 50H.
constant wdth=25H    ;The symbol 'wdth' is
                    ;initialized to 25H.
                    ;A constant symbol always needs
                    ;to be initialized.
perimeter=2*(lngth+wdth);The value of a CONSTANT cannot
                    ;be reassigned after having been
                    ;initialized once. So 'lngth' and
                    ;'wdth' cannot be reassigned. But
                    ;'perimeter' has been declared
                    ;as variable, and so can be
                    ;reassigned.

area=lngth*wdth

end

```

4.67 while——当条件为 TRUE 时执行循环

4.67.1 语法

首选语法:

```

while expr
:
endw

```

支持的语法:

```

.while expr
:
.endw

```

4.67.2 说明

只要 *expr* 为 TRUE，就汇编 `while` 和 `endw` 之间的行。值为 0 的表达式被看作逻辑 FALSE。值为其他值的表达式被看作逻辑 TRUE。相应的 TRUE 表达式可以保证返回一个非 0 的值，而 FALSE 表达式则返回 0。

while 循环可最多可包含 100 行并最多重复 256 次。while 循环最多可嵌套 8 层。

4.67.3 使用

此伪指令在以下类型的代码中使用：绝对代码或可重定位代码。欲知有关代码类型的信息，请参见第 1.6 节“汇编器操作”。

此伪指令不是指令，用于控制代码的汇编方式而不是代码运行时的行为。对条件汇编使用此伪指令。

4.67.4 相关伪指令

endw if

4.67.5 简单示例

while 不在运行时执行，而是根据条件产生汇编代码。可通过查看列表文件 (*.lst) 或反汇编窗口来查看此示例的结果。

```
test_mac macro count
    variable i
i = 0
    while i < count
        movlw i
i += 1
    endwhile
endm
start
    test_mac 5
end
```

4.67.6 应用程序示例 —— while/endw

此示例演示了伪指令 while 在某条件为 TRUE 时执行循环的用法。此伪指令与 endw 伪指令一起使用。

```
#include p16f877a.inc    ;Include standard header file
                        ;for the selected device.

variable i              ;Define the symbol 'i' as a
                        ;variable.

mydata udata 0x20       ;Allocate RAM for labels
    reg_hi res 1        ;reg_hi and reg_lo.
    reg_lo res 1

RST      CODE      0x0    ;The code section named RST
                        ;is placed at program memory
                        ;location 0x0. The next two
                        ;instructions are placed in
                        ;code section RST.

    pagesel start      ;Jumps to the location labelled
    goto start         ;'start'.

shift_right macro by_n  ;Beginning of a macro, which
                        ;shifts register data n times.
                        ;Code length generated after
                        ;assembly, varies depending upon
                        ;the value of parameter 'by_n'.
i=0
    while i< by_n      ;Initialize variable i.
                        ;Following 3 lines of assembly
                        ;code are repeated as long as
```



```
;i< by_n.
```

while 循环中最多允许 100 行。

```
    bcf     STATUS,C           ;Clear carry bit.
    rrf     reg_hi,F           ;reg_hi and reg_lo contains
    rrf     reg_lo,F           ;16-bit data which is rotated
                                ;right through carry.
i+=1                             ;Increment loop counter i.
```

i 的值不能递增超过十进制数 255。

```
    endw                             ;End while loop. The loop will
                                ;break here after i=by_n.
    endm                             ;End of 'shift_right' macro.

PGM      CODE                      ;This is the begining of the
                                ;code section named PGM. It is
                                ;a relocatable code section
                                ;since no absolute address is
                                ;given along with directive CODE.

start
    movlw   0x88                  ;Initialize reg_hi and
    movwf   reg_hi                ;reg_lo for observation.
    movlw   0x44
    movwf   reg_lo

    shift_right 3                 ;Shift right 3 times the 16-bit
                                ;data in reg_hi and reg_lo. This
                                ;is an example. A value 8 will
                                ;shift data 8 times.

    goto $                        ;Go to current line (loop here)
end
```

注:

第 5 章 汇编器示例、技巧和窍门

5.1 简介

通过示例说明多条 MPASM 汇编器伪指令的用法。

伪指令是出现在源代码中的汇编器命令，但不是操作码。它们用于控制汇编器的输入、输出和数据分配。

很多汇编器伪指令都有备用的名称和格式。这些备用的名称和格式用于向后兼容来自 Microchip 的早期汇编器，以及符合个人的编程习惯。如果需要可移植的代码，建议按照包含在此文档内的规范编写程序。

如需获得在本章示例中讨论的所有伪指令的列表，请参见第 4 章“伪指令”。

<p>注： 虽然 MPLINK 目标链接器经常与 MPASM 汇编器一起使用，但是 MPLINK 链接描述文件不支持 MPASM 汇编器伪指令。如需了解更多有关控制列表和 hex 文件输出的链接器选项的信息，请参见 MPLINK 目标链接器文档。</p>
--

本章涉及以下主题：

- 显示端口计数示例
- 端口 B 交替和延时程序的示例
- 使用变量和常数进行计算的示例
- 32 位延时程序的示例
- 在固件中仿真 SPI™ 的示例
- 十六进制字节到 ASCII 字节转换的示例
- 获取示例的其他渠道
- 技巧和窍门

5.2 显示端口计数示例

此示例中突出说明的伪指令有：

- #include
- end

5.2.1 程序功能描述

这个简单的程序不断对 PORTA 和 PORTB 进行递增计数。此计数可在软件中显示在 SFR 中或 MPLAB IDE 的 watch 窗口中，或在硬件中显示在连接的 LED 或示波器上。使用延时程序可以延缓计数（参见其他示例）。

一旦计数增加到 0xFF，它将计满返回到 0x00 并再次开始递增计数。

该应用程序以绝对代码编写，即，您只能使用汇编器（而不是汇编器和链接器）来生成可执行文件。

使用 #include 包含选定处理器的标准头文件。然后端口输出数据锁存器被清零。必须将端口 A 设置为数字 I/O，因为上电时，有几个引脚是模拟引脚。数据方向寄存器（TRISx）被清零以将端口引脚设置为输出。进入一个名为 Loop 的循环，在该循环中每个端口的值无限递增直到该程序停止为止。最终，该程序以 end 结束。

5.2.2 注释代码列表

```
;Toggles Port pins with count on PIC18F8720
;PortA pins on POR:
; RA5, RA3:0 = analog inputs
; RA6, RA4 = digital inputs
;PortB pins on POR:
; RB7:0 = digital inputs

#include p18f8720.inc ;Include file needed to reference
                      ;data sheet names.

clrfs PORTA           ;Clear output data latches on Ports
clrfs PORTB

movlw 0x0F            ;Configure Port A for digital I/O
movwf ADCON1

clrfs TRISA           ;Set data direction of Ports as outputs
clrfs TRISB

Loop
incf PORTA,F          ;Read PORTA, add 1 and save back.
incf PORTB,F          ;Read PORTB, add 1 and save back.
goto Loop             ;Do this repeatedly - count.
end                   ;All programs must have an end directive.
```

5.3 端口 B 交替和延时程序的示例

此示例中突出说明的伪指令有：

- `udata, res`
- `equ`
- `code`
- `banksel, pagesel`

此示例中涉及的事项有：

- 程序功能说明
- 注释代码列表
- 头文件
- 寄存器和位分配
- 程序存储器代码段和页面选择
- 存储区选择
- 中断

5.3.1 程序功能说明

此程序不断地使端口 B 引脚上的输出在 1 和 0 之间交替。使用中断的两个延时程序可以为交替输出定时。如果端口 B 连接了 LED，它们将会闪烁（1 = 点亮，0 = 熄灭）。

PICmicro MCU 的类型在 MPLAB IDE 中设置，所以无需在代码中设置。然而，如果您希望在代码中指定 MCU 和基数，您可以通过使用 `processor` 和 `radix` 伪指令或 `list` 命令达到目的，即，`list p=16f877a, r=hex`。

该应用程序用可重定位代码编写，即，您必须使用汇编器和链接器来生成可执行文件。欲知有关如何使用汇编器文件和链接描述文件建立项目的信息，请参见 **PICmicro 语言工具和 MPLAB IDE**。

使用 `#include` 包含选定处理器的标准头文件。使用 `udata`、`res` 和 `equ` 伪指令分配寄存器。使用 `code` 语句创建代码段。需要时，数据存储器的存储区选择和程序存储器的页面选择可以通过使用 `banksel` 和 `pagesel` 伪指令实现。最终，程序以 `end` 结束。

5.3.2 注释代码列表

```

;*****
;* MPASM Assembler Control Directives *
;* Example Program 1                    *
;* Alternate output on Port B between *
;* 1's and 0's                          *
;*****

#include p16f877a.inc      ;Include header file
    
```

MPLAB IDE 包含了很多头文件（*.inc）可供所支持的器件使用。这些文件可在安装目录中找到。欲知更多有关头文件的信息，请参见第 5.3.3 节“头文件”。

```
    udata                                ;Declare storage of RAM variables
DTEMP res 1                             ;Reserve 1 address location
DFLAG res 1                             ;Reserve 1 address location

DFL0 equ 0x00                           ;Set flag bit - 0 bit of DFLAG
```

将 DTEMP 设置为临时寄存器，该寄存器位于由链接器决定的 RAM 单元中。将 DFLAG 设置为标志寄存器，该寄存器位于 DTEMP 寄存器所在单元的下一个单元。将 DFL0 设置为一个值以代表 DFLAG 寄存器中的一位，在此例中为第 0 位。如需了解更多信息，请参见附加注释部分。

```
rst      code      0x00      ;Reset Vector
    pagesel Start          ;Ensure correct page selected
    goto      Start          ;Jump to Start code
```

将复位向量置于程序存储器单元 0x00 中。当程序复位时，程序将跳转到 Start。

```
intrpt   code      0x04      ;Interrupt Vector
    goto   ServInt          ;Jump to service interrupt
```

将中断向量代码置于程序存储器单元 0x04 中，因为此器件会自动跳转到此地址处理中断。当中断发生时，程序将跳转到 ServInt 程序。

```
isr      code      0x08      ;Interrupt Service Routine
ServInt

    banksel OPTION_REG        ;Select Option Reg Bank (1)
    bsf     OPTION_REG, T0CS   ;Stop Timer0

    banksel INTCON            ;Select INTCON Bank (0)
    bcf     INTCON, T0IF       ;Clear overflow flag
    bcf     DFLAG, DFL0        ;Clear flag bit

    retfie                    ;Return from interrupt
```

对于 PIC16F877A，没有足够的存储器空间添加 pagesel ServInt 语句以确保正确的页面选择。因此，需要特别地将 ISR 代码放在页 0。如需了解有关 ISR 代码的更多信息，请参见第 5.3.7 节“中断”。

```
;*****
;* Main Program                                     *
;*****

    code                                ;Start Program
```

开始程序代码。由于未指定地址，程序存储器单元将由链接器决定。如需了解更多信息，请参见第 5.3.5 节“程序存储器代码段和页面选择”。

```
Start
    clrf    PORTB                ;Clear PortB

    banksel TRISB                ;Select TRISB Bank (1)
    clrf    TRISB                ;Set all PortB pins as outputs

    banksel INTCON               ;Select INTCON Bank (0)
    bsf     INTCON, GIE          ;Enable Global Int's
    bsf     INTCON, T0IE         ;Enable Timer0 Int
```

首先，使用数据方向（TRISB）寄存器将端口 B 引脚全部设置为输出。然后设置 Timer 0 和中断供以后使用。

```
Loop
    movlw   0xFF
    movwf   PORTB                ;Set PortB
    call    Delay1               ;Wait

    clrf    PORTB                ;Clear PortB
    pagesel Delay2               ;Select Delay2 Page
    call    Delay2               ;Wait

    pagesel Loop                  ;Select Loop Page
    goto    Loop                 ;Repeat
```

将端口 B 的所有引脚设置为高电平并等待 Delay 1。然后将端口 B 的所有引脚设置为低电平并等待 Delay 2。重复该过程直到程序停止。这将出现“交替翻转（flashing）”端口 B 的电平的效果。

```
;*****
;* Delay 1 Routine - Timer0 delay loop    *
;*****
```

Delay1

```
    movlw   0xF0                ;Set Timer0 value
    movwf   TMR0                ;0x00-longest delay
                                ;0xFF-shortest delay

    clrf    DFLAG
    bsf     DFLAG, DFL0         ;Set flag bit

    banksel OPTION_REG          ;Select Option Reg Bank (1)
    bcf     OPTION_REG, T0CS    ;Start Timer0

    banksel DFLAG               ;Select DFLAG Bank (0)

TLoop                                ;Wait for overflow: 0xFF->0x00
    btfsc   DFLAG, DFL0         ;After interrupt, DFL0 = 0
    goto    TLoop

    return
```

使用 Timer 0 来产生 Delay 1。首先，给定时器一个初始值。然后，使能该定时器并等待它从 0xFF 溢出到 0x00。这样将产生中断，而中断将结束延时。如需了解更多信息，请参见第 5.3.7 节“中断”。

```
;*****  
;* Delay 2 Routine - Decrement delay loop *  
;*****  
  
decldly code      0x1000    ;Page 2
```

将 Delay2 程序放在程序存储器页 2 上的单元 0x1000 中。（如需了解更多有关 code 的信息，请参见第 5.3.5 节“程序存储器代码段和页面选择”。）将此代码放在页 0 以外的页上以演示程序如何跨页工作。

```
Delay2  
  
    movlw    0xFF          ;Set DTEMP value  
    movwf    DTEMP         ;0x00-shortest delay  
                           ;0xFF-longest delay  
  
DLoop    ;Use a simple countdown to  
    decfsz   DTEMP, F      ;create delay.  
    goto     DLoop         ;End loop when DTEMP=0  
  
    return
```

将寄存器 DTEMP 从初始值递减计数到 0x00 所需的时间作为 Delay 2。此方法不需要定时器或中断。

```
end
```

程序结束，即告诉汇编器没有更多的代码需要汇编。

5.3.3 头文件

使用 #include 伪指令将头文件包含在程序流中。

```
#include p16f877a.inc      ;Include header file
```

可以使用尖括号或引号将头文件的名称括起来，也可以不使用任何符号。可以指定被包含文件的完整路径，或者让汇编器去搜索。如需了解更多有关搜索顺序的信息，请参见第 4.41 节“#include——包含额外的源文件”中对 #include 伪指令的讨论。

头文件对于指定经常使用的常数（例如寄存器和引脚名称）极其有用。这些信息只需输入一次，然后该文件可被包含在使用带有这些寄存器和引脚的处理器的任何代码中。

5.3.4 寄存器和位分配

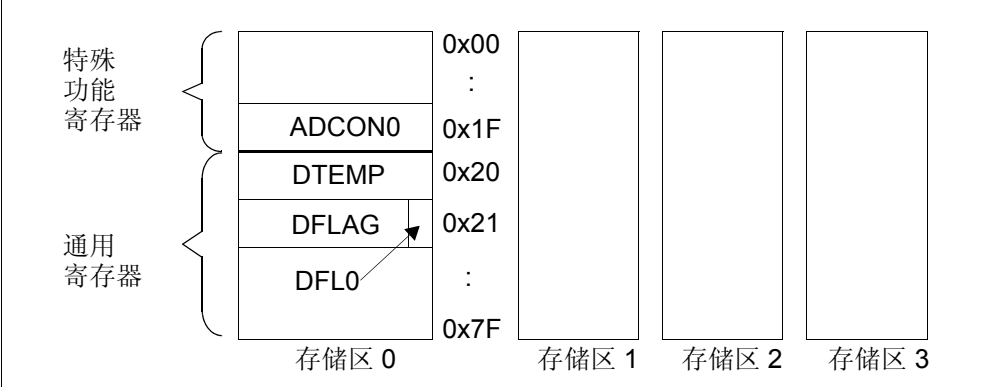
通过使用 `udata`、`res` 和 `equ` 伪指令可以指定您自己的寄存器和位，如下面几行所示。

```
    udata                                ;Declare storage of RAM variables
DTEMP res 1                             ;Reserve 1 address location
DFLAG res 1                             ;Reserve 1 address location

DFL0 equ 0x00                           ;Set flag bit - 0 bit of DFLAG
```

链接器在 **RAM** 中为 `DTEMP` 和 `DFLAG` 各分配了一个地址单元。为了便于说明，假定链接器选择的单元是通用寄存器（General Purpose Register，GPR）`0x20` 和 `0x21`。`DFL0` 被分配了 `0x00` 值，并且将在 `DFLAG` 寄存器中被用作引脚 0 的名称。

图 5-1: PIC16F877A 寄存器文件映射图



在可重定位代码中使用伪指令 `udata` 和 `res` 而不是 `equ` 来定义多个寄存器。如需了解更多有关这些伪指令的信息，参见：

- 第 4.61 节 “`udata`——开始目标文件中未初始化的数据段”
- 第 4.56 节 “`res`——保留存储器”
- 第 4.27 节 “`equ`——定义一个汇编器常数”

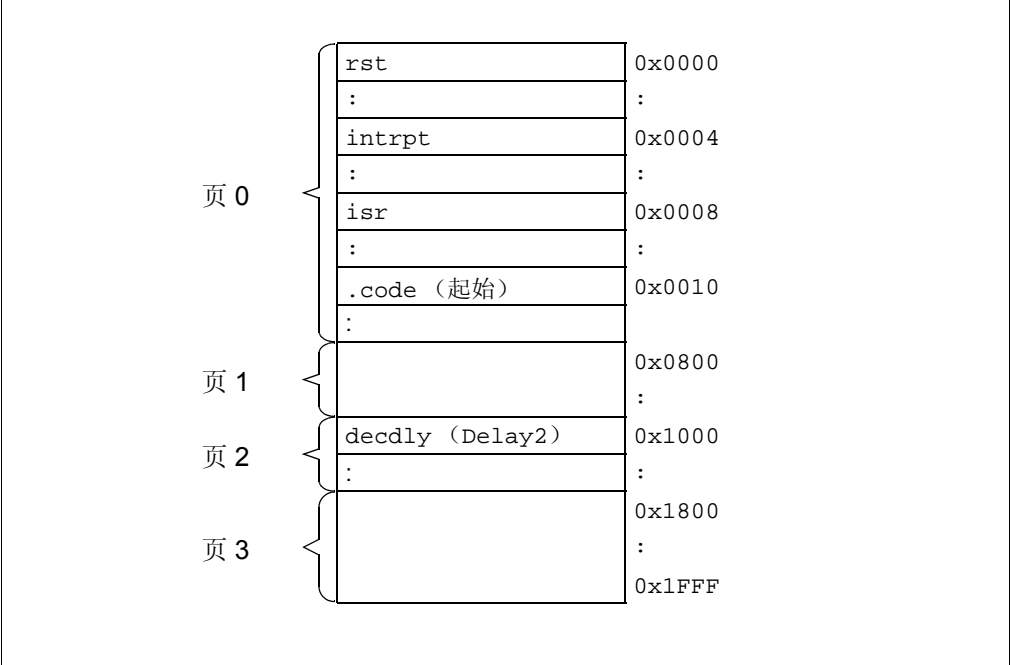
5.3.5 程序存储器代码段和页面选择

`code` 伪指令用来指定可重定位代码段。`org` 伪指令用来指定绝对代码段。如需了解更多有关可重定位代码和绝对代码之间差异的信息，参见第 6 章 “可重定位目标”。如需了解更多有关这些伪指令的信息，参见：

- 第 4.9 节 “`code`——开始目标文件代码段”
- 第 4.50 节 “`org`——设置程序起始处”

如果未使用 `code` 伪指令，将从地址 0 开始生成代码。本例中，`code` 用于指定 `0x00`（复位地址）、`0x04`（中断地址）、`0x08`（中断服务程序）和 `0x1000`（Delay 2 地址）处的代码。它并未明确地设置程序的起始地址，而是让链接器能正确地放置代码。由于链接器首先放置编址的代码，然后根据大小尝试放置可重定位代码，可能的程序存储器使用情况如下所示。

图 5-2: PIC16F877A 程序存储器映射图



由于主代码（.code 段）的实际位置未知，必须使用 pagesel 伪指令以确保程序正确地跳转到其他段。

```

rst      code      0x00      ;Reset Vector
pagesel Start
goto    Start
:
      code              ;Start Program
:
pagesel Delay2          ;Select Delay2 Page
call    Delay2          ;Wait
:
pagesel Loop            ;Select Loop Page
goto    Loop            ;Repeat
:
    
```

如需了解更多有关这条伪指令的信息，参见第 4.52 节 “pagesel——生成页面选择代码（PIC10/12/16 MCU）”。

5.3.6 存储区选择

在此示例中，必须配置端口 B，以使切换到数据存储器的存储区 1 的操作去访问 TRISB 寄存器。这种切换到存储区 1，随后返回到存储区 0 的操作可通过使用 banksel 伪指令轻易实现。

```

banksel TRISB          ;Select TRISB Bank (1)
clrf    TRISB          ;Set PortB as output

banksel INTCON          ;Select INTCON Bank (0)
bsf     INTCON, GIE     ;Enable Global Int's
bsf     INTCON, TOIE    ;Enable Timer0 Int
    
```

另外两个程序也使用 banksel 来访问选项寄存器（OPTION_REG）。如需了解更多有关这条伪指令的信息，参见第 4.7 节 “banksel——生成存储区选择代码”。

5.3.7 中断

此程序中的 **Delay 1** 程序使用 **Timer 0** 溢出中断作为定时机制。一旦发生中断，程序将跳转到中断向量。代码位于此处以跳转到找到中断处理代码的单元。

```
intrpt    code    0x04            ;Interrupt Vector
          goto     ServInt         ;Jump to service interrupt
```

中断处理代码，又称为中断服务程序（ISR），是由编程器生成的，用来处理外设中断和程序的特定要求。在此例中，**Timer 0** 停止且它的标志位被清零，从而使定时器可以再次运行。然后，程序定义的标志位被清零。最后，**retfie** 将程序带回中断发生时将要执行的指令。

```
isr        code    0x08            ;Interrupt Service Routine
ServInt

          banksel  OPTION_REG        ;Select Option Reg Bank (1)
          bsf      OPTION_REG, T0CS  ;Stop Timer0

          banksel  INTCON            ;Select INTCON Bank (0)
          bcf      INTCON, T0IF      ;Clear overflow flag
          bcf      DFLAG, DFL0       ;Clear flag bit

          retfie                    ;Return from interrupt
```

当程序代码再次开始执行时，被清零的标志位 **DFL0** 使延时循环 **TLOOP** 结束，从而结束 **Delay 1** 程序。

5.4 使用变量和常数进行计算的示例

此示例中突出说明的伪指令有：

- #define, #undef
- set
- constant, variable

此示例中论及的事项有：

- 程序功能说明
- 注释代码列表
- 使用 Watch 窗口

5.4.1 程序功能说明

此程序使用定义的常数和变量执行几种计算。

该应用程序用可重定位代码编写，即，您必须使用汇编器和链接器来生成可执行文件。

使用 #include 包含选定处理器的标准头文件。使用 code 语句创建代码段。

5.4.2 注释代码列表

```
*****  
;* MPASM Assembler Control Directives *  
;* Example Program 2 *  
;* Perform calculations *  
*****  
  
#include pl6f877a.inc ;Include header file  
  
#define Tdistance1 50 ;Define the symbol  
;Tdistance1  
#define Tdistance2 25 ;Define the symbol  
;Tdistance2  
#undef Tdistance2 ;Remove Tdistance2 from  
;the symbol table
```

#define 伪指令用于定义两个替代字符串：Tdistance1 用来替代 50，而 Tdistance2 用来替代 25。然后使用 #undef 将 Tdistance2 从符号表中删除，即，Tdistance2 不能再用来替代 25。

```
udata 0x20 ;Set up distance_reg  
distance_reg res 1 ;at GPR 0x20
```

udata 和 res 伪指令用来将 distance_reg 分配给寄存器 0x20。如需了解更多有关这些伪指令的信息，参见示例 1。

```
rst code 0x00 ;Reset Vector  
pagesel Start  
goto Start  
  
code ;Start Program  
Start  
clrf distance_reg ;Clear register  
  
movlw Tdistance1 ;Move value of Tdistance1
```

```
movwf distance_reg      ;into distance_reg

constant distance1=10    ;Declare distance1
                        ;a constant symbol
```

声明一个常数符号 `distance1`，其值为 **10**。一旦声明了常数，其值将不能被更改。

```
variable distance2      ;Declare distance2
                        ;a variable symbol
```

声明变量符号 `distance2`。当声明变量时，`variable` 伪指令无需初始化符号。

```
distance3 set 10         ;Define a value for
                        ;the symbol distance3
```

将符号 `distance3` 定义为 **10**。

```
distance2=15             ;Give distance2 an
                        ;initial value
distance2=distance1+distance2 ;Add distance1
                        ;to distance2
```

变量的赋值、加减语句必须放在独立的行上。

```
distance3 set 15         ;Change value of distance3
distance2=distance2+distance3 ;Add distance3
                        ;to distance2

movlw distance2          ;Move value of distance2
movwf distance_reg       ;into distance_reg

goto Start               ;Loop back to Start
end
```

5.4.3 使用 Watch 窗口

一旦程序开始，`Tdistance1` 的值将被放入 `distance_reg`。这可以在 MPLAB IDE 中的 **watch** 窗口中观察到，其中 `distance_reg` 的值将变为 **50**。在 **watch** 窗口符号列表中不会出现符号 `Tdistance1`，因为在 MPLAB IDE 中不能查看使用 `#define` 伪指令定义的符号，因为它们不是 RAM 变量。

该示例程序的最后几行将 `distance2` 的最终值写入 `distance_reg`。如果您打开 **watch** 窗口查看装入了值 **50** 的 `distance_reg`，将看到该值变为 **3A**。记住基数是 **16**，所以使用十六进制加法来确定 `distance2` 的值。

5.5 32 位延时程序的示例

此示例中突出说明的伪指令有：

- macro, endm
- banksel

5.5.1 程序功能说明

在很多应用程序中需要延时程序。此示例中，每个延时的增量为 20 μ s，该程序可以产生的总延时范围是 40 μ s 到 23.8 小时。（这里假定使用 4 MHz 的时钟。）

5.5.2 注释代码列表

```
;Each loop takes 20 clocks, or 20 us per loop,
;at 4MHz or 1MIPS clock.
;Turn off in config bits WDT for long simulations

#include p16F877A.inc

        udata 0x20
Dly0    res    1    ;Stores 4 bytes of data for the delay count
Dly1    res    1    ;Dly0 is the least significant byte
Dly2    res    1    ;while Dly3 is the most significant byte
Dly3    res    1

Dly32   MACRO DLY
        goto    $+1    ;delay 2 cycles
        goto    $+1    ;delay total of 4 cycles

;Take the delay value argument from the macro, precalculate
;the required 4 RAM values and load the The RAM values Dly3
;through Dly0.
        BANKSEL Dly3
        movlw   (DLY-1) & H'FF'
        movwf   Dly0
        movlw   (DLY-1) >>D'08' & H'FF'
        movwf   Dly1
        movlw   (DLY-1) >>D'16' & H'FF'

;Bytes are shifted and anded by the assembler to make user
;calculations easier.
        movwf   Dly2
        movlw   (DLY-1) >>D'24' & H'FF'

;Call DoDly32 to run the delay loop.
        movwf   Dly3
        call    DoDly32
        ENDM          ;End of Macro definition

RST     CODE    0x00          ;Reset Vector
        pagesel TestCode
        goto    TestCode

        CODE          ;Code starts here
TestCode
        Dly32    D'50000'    ;Max 4 billion+ (runs Dly32 Macro,
                             ;1 sec in this case).
        nop              ;ZERO STOPWATCH, put breakpoint here.
```

```
        goto    TestCode    ;Go back to top of program and
                               ;run the delay again.

;Subroutine, called by the Macro Dly32 (20 Tcy per loop)
DoDly32
    movlw    H'FF'        ;Start with -1 in W

    addwf    Dly0,F        ;LSB decrement
    btfsc    STATUS,C      ;was the carry flag set?
    clrw     ;If so, 0 is put in W

    addwf    Dly1,F        ;Else, we continue.
    btfsc    STATUS,C
    clrw     ;0 in W

    addwf    Dly2,F
    btfsc    STATUS,C
    clrw     ;0 in W

    addwf    Dly3,F
    btfsc    STATUS,C
    clrw     ;0 in W

    iorwf    Dly0,W        ;Inclusive-OR all variables
    iorwf    Dly1,W        ;together to see if we have reached
    iorwf    Dly2,W        ;0 on all of them.
    iorwf    Dly3,W

    btfss    STATUS,Z      ;Test if result of Inclusive-OR's is 0
    goto     DoDly32      ;It was NOT zero, so continue counting
    retlw    0            ;It WAS zero, so exit this subroutine.

END
```

5.6 在固件中仿真 SPI™ 的示例

此示例中突出说明的伪指令有：

- list
- #define
- udata, res
- global

5.6.1 程序功能描述

此程序用于在固件中仿真 SPI 功能。

该应用程序用可重定位代码编写，即，您必须使用汇编器和链接器来生成可执行文件。

list 伪指令用于定义处理器以及设置列表文件的格式。使用 #include 包含选定处理器的标准头文件。使用 #define 声明 SPI 变量。使用 udata 和 res 伪指令分配程序寄存器。使用 code 语句创建代码段。使用 global 访问外部代码。

5.6.2 注释代码列表

```
;*****
; Emulates SPI in firmware
; Place byte in Buffer, call SPI_Out - sends MSB first
;*****

LIST          P=18F4520      ;define processor
#include      <P18F4520.INC> ;include file

list          c=132, n=0      ;132 col, no paging

;*****

#define Clk LATB,0 ; SPI clock output
#define Dat LATB,1 ; SPI data output
#define Bus LATB,2 ; busy indicator

;*****
;Variable definitions
        udata
Buffer      res 1 ; SPI transmit data
Counter     res 1 ; SPI transmit bit counter
DelayCtr    res 1

;*****
        code
SPI_Out
        clrf    Counter          ; init bit counter
        bsf     Counter,7

        bcf     Clk              ; clear clock
        bcf     Dat              ; clear data out
        bsf     Bus              ; indicate busy
```



```
Lup      movf    Counter,W           ; get mask
          andwf   Buffer,W           ; test selected bit

          btfss   STATUS,Z           ; was result zero?
          bsf     Dat                ; set data

          bsf     Clk                ; set clock
          bcf     Clk                ; clear clock

          bcf     Dat                ; clear data

          rrrncf   Counter,F         ; test next bit

          btfss   Counter,7          ; done with byte?
          bra     Lup                ; no

          bcf     Bus                ; indicate not busy

          return
```

```
;*****
```

```
    global  SPI_Out, Buffer
end
```

5.7 十六进制字节到 ASCII 字节转换的示例

此示例中突出说明的伪指令有：

- `udata, res`
- `global`

5.7.1 程序功能说明

此程序将一个十六进制字节转换为两个 ASCII 字节。

该应用程序用可重定位代码编写，即，您必须使用汇编器和链接器来生成可执行文件。

使用 `udata` 和 `res` 伪指令分配程序寄存器。使用 `code` 语句创建代码段。使用 `global` 访问外部代码。

5.7.2 注释代码列表

```
;*****
; get a hex byte in W, convert to 2 ASCII bytes in ASCIIH:ASCIIL
; req 2 stack levels
;
;*****
Variables    udata
HexTemp      res 1
ASCIIH       res 1
ASCIIL       res 1

;*****
                        code
Hex2ASC
    movf    HexTemp,W
    andlw   0x0F           ; get low nibble
    call    DecHex
    movwf   ASCIIL

    swapf   HexTemp,F
    movf    HexTemp,W
    andlw   0x0F           ; get high nibble
    call    DecHex
    movwf   ASCIIH

    return

;*****
DecHex
    sublw   0x09           ; 9-WREG
    btfss   STATUS,C       ; is nibble Dec?
    goto    HexC           ; no, convert hex

Dec
    movf    HexTemp,W       ; convert DEC nibble to ASCII
    andlw   0x0F
    addlw   A'0'
    return

HexC
    movf    HexTemp,W       ; convert HEX nibble to ASCII
    andlw   0x0F
    addlw   A'A'-0x0A
    return
```

```
;*****
global    Hex2ASC, ASCIIH, ASCIIIL

END
```

5.8 获取示例的其他渠道

单条伪指令的简短使用示例列在每条伪指令的主题下。参见第 4 章“伪指令”。

从以下这些渠道可以获得多条伪指令的使用示例：

- **readme.asm**——串行 EEPROM 支持
- 应用笔记，技术摘要
 - 网站——<http://www.microchip.com>
- 代码示例和模板
 - MPLAB IDE 安装目录
 - 网站——<http://www.microchip.com>

5.9 技巧和窍门

为了降低成本，设计者需要充分利用 MCU 中的可用程序存储器空间。程序存储器通常在 MCU 成本中占据较大的比重。对代码进行优化有助于避免多花钱购买超出实际需求的存储器。这里给出了一些有助于缩短代码长度的方法。如需了解更多信息，请参见 *Tips'n Tricks* (DS40040)。

- 技巧 1：延时技巧
- 技巧 2：优化目标
- 技巧 3：条件置位 / 清零位
- 技巧 4：交换文件寄存器和 W 的内容
- 技巧 5：使用进位标志位移位

5.9.1 技巧 1：延时技巧

- 使用 *GOTO Next Instruction* 代替两条 NOP 指令。
- 使用 *CALL Rtrn* 代替四条 NOP 指令（其中 *Rtrn* 是从现有子程序退出的标号）。

```
;*****
NOP
NOP                ;2 instructions, 2 cycles
;*****
GOTO $+1           ;1 instruction, 2 cycles
;*****
Call Rtrn          ;1 instruction, 4 cycles
:
Rtrn RETURN
;*****
```

MCU 常常通过数据总线、LED、按钮和锁存器等与“外部世界”连接。由于 MCU 以固定的频率运行，它经常需要延时程序来实现以下目标：符合其他器件的设置 / 保持时间、握手的暂停时间或降低共用总线的数据速率。

较长的延时很适合 DECFSZ 和 INCFSZ 指令，在这两条指令中变量递增或递减直到达到零，这时将执行条件跳转。对于周期数不多的较短的延时，这里给出了一些缩短代码长度的方法。

对于双周期延时，通常使用两条 NOP 指令，这将使用两个程序存储器单元。使用 GOTO \$+1 可以得到样的结果。\$ 代表 MPASM 汇编器中的当前程序计数器值。当遇到此指令时，MCU 将跳转到下一个存储器单元。这与使用两条 NOP 指令的结果相同，但是因为 GOTO 指令使用两个指令周期来执行，所以产生了一个双周期延时。这样仅使用一个程序存储器单元就产生了双周期延时。

要产生四周期延时，可在代码中为 RETURN 指令添加一个标号。在此示例中，标号 Rtrn 被添加到已经存在于代码中某处的子程序中的 RETURN 指令中。当执行 CALL Rtrn 时，MCU 延时两个指令周期来执行 CALL，外加两个指令周期来执行 RETURN。这样不使用四条 NOP 指令产生一个四周期延时，而添加一条 CALL 指令可实现同样的结果。

5.9.2 技巧 2：优化目标

- 目标位决定是用 W 还是 F 来保存结果
- 看一下数据移动和重构

示例：A + B → A

MOVF	A,W	MOVF	B,W
ADDWF	B,W	ADDWF	A,F
MOVWF	A		

3 条指令

2 条指令

在指令中小心地使用目标位可以节省程序存储器空间。此处，寄存器 A 和寄存器 B 的值相加，结果放入寄存器 A。在逻辑和算术运算中可以使用目标选项。在第一个示例中，ADDWF 指令的结果被放在工作寄存器中。MOVWF 指令用于将该结果从工作寄存器移动到寄存器 A。在第二个示例中，ADDWF 指令使用目标位将该结果放入 A 寄存器，节省了一条指令。

5.9.3 技巧 3：条件置位 / 清零位

- 将数据的一个位从 REGA 移动到 REGB
- 预处理 REGB 位
- 测试 REGA 位，需要时修改 REGB

BTFSS	REGA,2	BCF	REGB,5
BCF	REGB,5	BTFSC	REGA,2
BTFSC	REGA,2	BSF	REGB,5
BSF	REGB,5		

4 条指令

3 条指令

将一个位从 REGA 寄存器移动到 REGB 的一个技巧是进行位测试。在第一个示例中，使用 BTFSS 指令对 REGA 中的位进行测试。如果该位为零，则执行 BCF 指令并清零 REGB 位；如果该位为 1，则跳过该指令。第二个位测试确定该位是否为 1，如果是，则执行 BSF 置位 REGB 位，否则跳过该指令。此过程需要四条指令。

更有效的技巧是假定 REGA 中的位为零，清零 REGB 位然后测试 REGA 位是否为零。如果 REGA 位为零，则假定正确，跳过 BSF 指令，否则将 REGB 位置 1。第二个示例中的序列使用三条指令，因为有一个位测试不需要。

很重要的一点是当 REGB 是一个输出高电平的端口时，第二个示例将产生一个双周期的毛刺。这是因为无论 REGA 中的位值如何，BCF 和 BTFSC 指令都将执行。

5.9.4 技巧 4：交换文件寄存器和 W 的内容

下面的宏在不使用第二个寄存器的情况下交换 W 和 REG 的内容。

```
SWAPWF  MACRO  REG
        XORWF  REG,F
        XORWF  REG,W
        XORWF  REG,F
ENDM
```

需要：0 个临时寄存器，3 条指令，3 个 Tcy

交换某个寄存器和工作寄存器的内容的高效方法是使用三条 XORWF 指令。它需要三条指令，不需要临时寄存器。此处为一个示例：

W	REG	Instruction
10101100	01011100	XORWF REG,F
10101100	11110000	XORWF REG,W
01011100	11110000	XORWF REG,F
01011100	10101100	Result

5.9.5 技巧 5：使用进位标志位移位

通过进位移动一个字节，无需使用循环计数的 RAM 变量：

- 易于在串行接口发送程序中应用。
- 进位标志位清零（最后一个周期除外），重复该循环直到全零标志位置 1 表示结束为止。

```
list p=12f629
#include p12f629.inc

buffer equ 0x20

bsf  STATUS,C      ;Set 'end of loop' flag
rlf  buffer,F      ;Place first bit into C
bcf  GPIO,Dout     ;Precondition output
btfsc STATUS,C     ;Check data - 0 or 1?
bsf  GPIO,Dout
bcf  STATUS,C      ;Clear data in C
rlf  buffer,F      ;Place next bit into C
movf buffer,F      ;Force Z bit
btfss STATUS,Z     ;Exit?
goto Send_Loop
```

注:

第 6 章 可重定位目标

6.1 简介

将 MPASM 汇编器和 MPLINK 目标链接器配合使用能够生成和链接预编译的目标模块。编写将要被汇编成目标模块的源代码与编写用于直接生成可执行（hex）文件的代码略有不同。针对绝对地址汇编设计的 MPASM 汇编器程序需做一些小改动以正确编译可重定位的目标模块。

本章涉及的主题：

- 头文件
- 程序存储器
- Low、High 和 Upper 操作数
- RAM 分配
- 配置位和 ID 单元
- 访问来自其他模块的标号
- 分页和分区问题
- 生成目标模块
- 代码示例

6.2 头文件

在生成目标模块时应该使用 Microchip 提供的标准头文件（例如 p18f8720.inc）。这些头文件定义目标处理器的特殊功能寄存器。

例 6-1: 包含头文件

```
#include p18f8720.inc  
:
```

更多信息请参见第 4.41 节 “#include——包含额外的源文件”。

6.3 程序存储器

程序存储器代码必须被组织在逻辑代码段中。要做到这一点，在代码的开头必须添加一个 code 段声明（参见第 4.9 节“code——开始目标文件代码段”）以使该代码可重新定位。

绝对代码	等效可重定位代码
Start clrw option	code ;Address determined ;by the linker. Start clrw option
Progl org 0x0100 movlw 0x0A movwf var1	Progl code 0x0100 ;Start at 0x0100 movlw 0x0A movwf var1

如果在源文件中定义了多个 code 代码段，每个代码段必须具有惟一的名称。如果未指定名称，则默认命名为 .code。

在一个源文件中每个程序存储器段必须是连续的。同一源文件中的一个段可能无法被分成几块。

代码的物理地址可通过在 code 伪指令中提供可选地址参数固定。在以下情形中必须执行上述操作：

- 指定复位和中断向量
- 确保代码段不会与页边界重叠

例 6-2: 可重定位代码

```
Reset code 0x01FF ;Fixed address
      goto Start
Main code ;Address determined by the linker
      clrw
      option
```

6.4 LOW、HIGH 和 UPPER 操作数

Low、high 和 upper 操作数用于返回多字节标号值中的某一字节。如果使用 low，那么将仅使用表达式的第 0 位到第 7 位。如果使用 high，那么将仅使用表达式的第 8 位到第 15 位。如果使用 upper，那么将仅使用表达式的第 16 位到第 21 位。

操作数	定义
low	返回地址的低字节
high	返回地址的次高字节
upper	返回地址的最高字节
scnsz_low	返回段大小的低字节。
scnsz_high	返回段大小的次高字节。
scnsz_upper	返回段大小的最高字节。
scnend_low	返回段尾地址的低字节。
scnend_high	返回段尾地址的次高字节。
scnend_upper	返回段尾地址的最高字节。
scnstart_low	返回段首地址的低字节。
scnstart_high	返回段首地址的次高字节。
scnstart_upper	返回段首地址的最高字节。

操作数的优先级信息可在第 3.5 节“算术运算符和优先级”中查询。

对于可重定位符号的操作数有一些限制。例如，low、high 和 upper 操作数必须为以下格式：

[low|high|upper]（可重定位符号 + 常数偏移量）

其中：

- 可重定位符号为定义程序或数据存储器地址的任意标号
- 常数偏移量为在汇编时可以被解析为介于 -32768 到 32767 之间的值的表达式

可重定位符号或常数偏移量均可以省略。

如果以下两个符号被在同一代码或数据段中定义，则以下格式的操作数：

可重定位符号 - 可重定位符号

将相减得到一个常数值。

除了段操作数之外，还有段伪指令。

伪指令	定义
scnend_lfsr	scnend_lfsr n,s, 其中 n 为 0、1 或 2（与 LFSR 指令相同），并且 s 为要用作段名称的字符串。该指令用段的尾地址装载 LFSR。
scnstart_lfsr	scnstart_lfsr n,s, 其中 n 为 0、1 或 2（与 LFSR 指令相同），并且 s 为要用作段名称的字符串。该指令用段的首地址装载 LFSR。

只有当生成了目标文件以后这些操作数才有意义。它们不能在生成目标代码时使用。

例 6-3: 使用常规操作数

常规操作数 low、high 和 upper 可被用于访问表中的数据。下列代码实例来自随 PICDEM™ 2 Plus 演示板提供的 p18demo.asm 文件。以下是从该 asm 文件中节选的代码，显示了如何将 “Microchip” 从表中读出并显示在演示板 LCD 上。

```
#include p18f452.inc
:
PROG1 CODE

stan_table                                ;table for standard code
;      "XXXXXXXXXXXXXXXXXX"
;
data   " Voltmeter      "      ;0
data   "   Buzzer      "      ;16
data   " Temperature    "      ;32
data   "   Clock       "      ;48
data   "RA4=Next RB0=Now"      ;64
data   "  Microchip    "      ;80
data   " PICDEM 2 PLUS  "      ;96
data   "RA4=Set RB0=Menu"      ;112
data   "RA4= --> RB0= ++"      ;128
data   "   RB0 = Exit   "      ;144
data   "Volts =         "      ;160
data   "Prd.=128 DC=128 "      ;176
:
```

;***** STANDARD CODE MENU SELECTION *****

```
    movlw    .80                      ;send "Microchip" to LCD
    movwf    ptr_pos
    call     stan_char_1
    :
;----Standard code, Place characters on line-1----
stan_char_1
    call     LCDLine_1                ;move cursor to line 1
    movlw    .16                      ;1-full line of LCD
    movwf    ptr_count
    movlw    UPPER stan_table        ;use operands to load
    movwf    TBLPTRU                 ;table pointer values
    movlw    HIGH stan_table
    movwf    TBLPTRH
    movlw    LOW stan_table
    movwf    TBLPTRL
    movf     ptr_pos,W
    addwf    TBLPTRL,F
    clrf     WREG
    addwfc   TBLPTRH,F
    addwfc   TBLPTRU,F

stan_next_char_1
    tblrd    *+
    movff    TABLAT,temp_wr
    call     d_write                  ;send character to LCD

    decfsz   ptr_count,F              ;move pointer to next char
    bra      stan_next_char_1

    movlw    "\n"                    ;move data into TXREG
    movwf    TXREG                   ;next line
    btfss    TXSTA,TRMT              ;wait for data TX
    goto     $-2
    movlw    "\r"                    ;move data into TXREG
    movwf    TXREG                   ;carriage return
    btfss    TXSTA,TRMT              ;wait for data TX
    goto     $-2

    return
    :
```

6.5 RAM 分配

必须在数据段内分配 RAM 空间。有以下五种数据段类型：

注： 使用快数操作、覆盖或共享数据的能力随器件的不同而不同。如需了解更多信息，请查询具体器件的数据手册。

- **udata**——未初始化数据。这是数据段最常见的类型。在此类段中保留的单元未被初始化，并且只能通过在该段内定义的标号访问或间接访问。参见第 4.61 节“**udata**——开始目标文件中未初始化的数据段”。
- **udata_acs**——未初始化快速操作数据。此类数据段用于存放将被放入 PIC18 器件的快速操作 RAM 中的变量。快速操作 RAM 用作特定指令的快速数据访问。参见第 4.62 节“**udata_acs**——开始目标文件未初始化快速操作的数据段（PIC18 MCU）”。
- **udata_ovr**——未初始化覆盖数据。此类数据段用于存放某些变量，这些变量可在与同一个模块或其他链接模块中的其他变量相同的地址处声明。通常此类数据段用于存放临时变量。参见第 4.63 节“**udata_ovr**——开始目标文件中覆盖的未初始化的数据段”。
- **udata_shr**——未初始化共享数据。此类数据段用于存放那些将要被放入 PIC12/16 器件中未分区 RAM 或在所有区块之间共享的 RAM 中的数据。参见第 4.64 节“**udata_shr**——开始目标文件中共享的未初始化的数据段（PIC12/16 MCU）”。
- **idata**——初始化数据。链接器将生成一个查找表，用于将此类段中的变量初始化为指定的值。当链接 MPLAB C17 或 C18 代码时，将在执行启动代码的过程中初始化这些单元。只能通过在此类段中定义的标号访问或间接访问在该段中保留的单元。参见第 4.35 节“**idata**——开始目标文件已初始化的数据段”。

下面给出了如何创建数据声明的示例。

例 6-4: RAM 分配

绝对代码

使用 `cblock` 来定义变量寄存器单元（参见第 4.8 节“**cblock**——定义常数块”）。变量值需要在代码中指定。

```
cblock 0x20
    HistoryVector          ;Must be initialized to 0
    InputGain, OutputGain ;Control loop gains
    Temp1, Temp2, Temp3    ;Used for internal calculations
endc
```

等效可重定位代码

使用数据声明来定义寄存器单元和初始化。

```
idata
    HistoryVector db 0      ;Initialized to 0
udata
    InputGain, OutputGain ;Control loop gains
    OutputGain res 1
udata_ovr
    Temp1 res 1             ;Used for internal calculations
    Temp2 res 1
    Temp3 res 1
```

必要时，可以通过提供可选地址参数在存储器中固定该段中的单元。如果指定了多个段类型，那么每个段必须有惟一的名称。如果未提供段名，那么段将采用默认的名称：`.idata`、`.udata`、`.udata_acs`、`.udata_shr` 和 `.udata_ovr`。

当在 `idata` 段中定义初始化数据时，可以使用 `db`、`dw` 和 `data` 伪指令。`db` 定义数据存储器中的连续字节，而 `dw` 和 `data` 则以低字节到高字节的顺序定义数据存储器中的连续字。下面给出了如何初始化数据的示例。

例 6-5: 重定位代码清单

```
00001 IDATA
0000 01 02 03 00002 Bytes DB 1,2,3
0003 34 12 78 56 00003 Words DW 0x1234,0x5678
0007 41 42 43 00 00004 String DB "ABC", 0
```

6.6 配置位和 ID 单元

仍可使用以下伪指令在可重定位目标中定义配置位和 ID 地址单元：

- 第 4.11 节 “`__config`——设置处理器的配置位”
- 第 4.12 节 “`config`——设置处理器的配置位（PIC18 MCU）”
- 第 4.37 节 “`__idlocs`——设置处理器 ID 单元”

只有一个已链接的模块可指定这些伪指令。应该在声明任一代码段之前使用这些伪指令。使用了这些伪指令之后，当前的段将变为未定义的段。

6.7 访问来自其他模块的标号

在某一模块中定义而要在其他模块中使用的标号必须使用 `global` 伪指令导出（参见第 4.34 节 “`global`——导出标号”）。使用这些标号的模块必须使用 `extern` 伪指令（参见第 4.32 节 “`extern`——声明一个外部定义的标号”）来声明这些标号是存在的。下面给出了使用 `global` 和 `extern` 伪指令的示例。

例 6-6: 可重定位代码，定义模块

```
udata
    InputGain res 1
    OutputGain res 1
global InputGain, OutputGain
code
Filter
    global Filter
    : ; Filter code
```

例 6-7: 可重定位代码，引用模块

```
extern InputGain, OutputGain, Filter
udata
    Reading res 1

code
:
movlw GAIN1
movwf InputGain
movlw GAIN2
movwf OutputGain
```

```
movf Reading,W
call Filter
```

6.8 分页和分区问题

在许多情形下，RAM 分配将跨多个区而可执行代码将跨多个页。在这些情形下，有必要执行适当的区设置和页设置来正确地访问标号。但是，由于这些变量和地址标号的绝对地址在汇编时可能尚未知，因此并不总能将适当的代码放入源文件。针对这种情况，增加了 `banksel`（第 4.7 节“`banksel`——生成存储区选择代码”）和 `pagesel`（第 4.52 节“`pagesel`——生成页面选择代码（PIC10/12/16 MCU）”）两条伪指令。这两条伪指令指示链接器生成指定标号的正确存储区选择或页选择代码。下面给出了应如何转换代码的示例。

例 6-8: BANKSEL 和 PAGESEL

硬编码的存储区选择和页选择

使用间接寻址（FSR）和 STATUS 寄存器分别进行存储区选择和页选择。

```
#include p12f509.inc
Var1 equ 0x10          ;Declare variables
Var2 equ 0x30
...
movlw InitialValue
bcf   FSR, 5           ;Data memory Var1 bank (0)
movwf Var1
bsf   FSR, 5           ;Data memory Var2 bank (1)
movwf Var2
bsf   STATUS, PA0      ;Program memory page 1
call  Subroutine
...
Subroutine clrw         ;On Page 1
...
retlw 0
```

等效可重定位代码

使用 `banksel` 和 `pagesel` 伪指令分别存储区选择和页选择。

```
#include p12f509.inc
extern Var1, Var2      ;Declare variables

code
movlw InitialValue
banksel Var1           ;Select data memory Var1 bank
movwf Var1
banksel Var2           ;Select data memory Var2 bank
movwf Var2
pagesel Subroutine     ;Select program memory page
call  Subroutine
...
Subroutine clrw        ;Page unknown at assembly time
...
retlw 0
```

6.9 生成目标模块

一旦代码转换完毕，就将在 MPLAB IDE 中自动生成目标模块，或通过使用命令行或 shell 界面请求目标文件来生成目标模块。当在 Windows 操作系统中使用 MPASM 汇编器时，选中名为“Object File”的复选框。当使用 DOS 命令行接口时，指定 /o 选项并把“Assemble to Object File”切换为“Yes”。输出文件将具有扩展名 .o。

6.10 代码示例

由于 8 位乘 8 位的乘法是一个有用且通用的程序，将它分拆成一个独立的目标文件并在需要时链接将会很方便。绝对代码文件可被分为两个可重定位的代码文件：表示一个应用程序的调用文件和一个可编入库中的通用文件。

以下代码是从应用笔记 AN617 中摘录的。欲获得此应用笔记的可下载 PDF 文件，请访问 Microchip 网站。

例 6-9: 绝对代码

```
; Input:  fixed point arguments in AARGB0 and BARGB0
; Output: product AARGxBARG in AARGB0:AARGB1
; Other comments truncated. See AN617.
;*****
#include    p16f877a.inc ;Use any PIC16 device you like

LOOPCOUNT EQU    0x20    ;7 loops needed to complete routine
AARGB0      EQU    0x21    ;MSB of result out,
AARGB1      EQU    0x22    ;operand A in (8 bits)
BARGB0      EQU    0x23    ;LSB of result out,
                        ;operand B in (8 bits)

TestCode
    clrf     AARGB1        ;Clear partial product before testing
    movlw   D'11'
    movwf   AARGB0
    movlw   D'30'
    movwf   BARGB0
    call    UMUL0808L      ;After loading AARGB0 and BARGB0,
                        ;call routine
    goto    $              ;Result now in AARGB0:AARGB1,
                        ;where (B0 is MSB)

END

UMUL0808L
    movlw   0x08
    movwf   LOOPCOUNT
    movf    AARGB0,W
LOOPUM0808A
    rrf     BARGB0, F
    btfsc   STATUS,C
    goto    LUM0808NAP
    decfsz  LOOPCOUNT, F
    goto    LOOPUM0808A
    clrf    AARGB0
    retlw   0x00
LUM0808NAP
    bcf     STATUS,C
    goto    LUM0808NA
```

```

LOOPUM0808
    rrf      BARGB0, F
    btfsc   STATUS,C
    addwf   AARGB0, F
LUM0808NA
    rrf      AARGB0, F
    rrf      AARGB1, F
    decfsz  LOOPCOUNT, F
    goto    LOOPUM0808
    retlw   0

END

```

例 6-10: 可重定位代码，调用文件

```

; Input:  fixed point arguments in AARGB0 and BARGB0
; Output: product AARGxBARG in AARGB0:AARGB1
; Other comments truncated. See AN617.
;*****
#include    p16f877a.inc ;Use any PIC16 device you like

EXTERN    UMUL0808L, AARGB0, AARGB1, BARGB0

Reset     CODE    0x0
    pagesel   TestCode
    goto      TestCode

CODE
TestCode
    banksel   AARGB1
    clrf      AARGB1          ;Clear partial product before testing
    movlw     D'11'           ;Load in 2 test values
    movwf     AARGB0
    movlw     D'30'
    movwf     BARGB0
    pagesel   UMUL0808L
    call      UMUL0808L      ;After loading AARGB0 and BARGB0,
                                ;call routine
    goto      $              ;Result now in AARGB0:AARGB1,
                                ;where (AARGB0 is MSB)

END

```

例 6-11: 可重定位代码，库程序

```

; Input:  fixed point arguments in AARGB0 and BARGB0
; Output: product AARGxBARG in AARGB0:AARGB1
; Other comments truncated. See AN617.
;*****
#include    p16f877a.inc ;Use any PIC16 device you like

GLOBAL     UMUL0808L, AARGB0, AARGB1, BARGB0

UDATA
LOOPCOUNT RES    1      ;7 loops needed to complete routine
AARGB0      RES    1      ;MSB of result out,
AARGB1      RES    1      ;operand A in (8 bits)
BARGB0      RES    1      ;LSB of result out,
                                ;operand B in (8 bits)

```

```
CODE
UMUL0808L
    movlw    0x08
    movwf    LOOPCOUNT
    movf     AARGB0,W
LOOPUM0808A
    rrf      BARGB0, F
    btfsc    STATUS,C
    goto     LUM0808NAP
    decfsz   LOOPCOUNT, F
    goto     LOOPUM0808A
    clrf     AARGB0
    retlw    0x00
LUM0808NAP
    bcf      STATUS,C
    goto     LUM0808NA
LOOPUM0808
    rrf      BARGB0, F
    btfsc    STATUS,C
    addwf    AARGB0, F
LUM0808NA
    rrf      AARGB0, F
    rrf      AARGB1, F
    decfsz   LOOPCOUNT, F
    goto     LOOPUM0808
    retlw    0

END
```

第 7 章 宏语言

7.1 简介

宏是用户定义的指令集和伪指令集，每当调用宏时，这些指令和伪指令将嵌入到汇编器源代码中。

宏由汇编指令和伪指令序列组成。可将它们写成能够接受参数，使其使用起来非常灵活。它们的优点包括：

- 较高的抽象性，可读性和可靠性均得到提高。
- 为经常执行的函数提供一致的解决方案。
- 易于修改。
- 可测试性得到改善。

应用程序可能包含创建复杂的表、常用的代码和复杂的运算。

本章涉及以下主题：

- 宏语法
- 已定义的宏指令
- 宏定义
- 宏调用
- 宏代码示例

7.2 宏语法

MPASM 汇编器宏根据下面的语法定义：

```
label macro [arg1,arg2 ..., argn]  
:  
:  
endm
```

其中 *label* 是一个有效的汇编器标号，它将成为宏的名称；而 *arg* 是向宏提供的任意数量的可选参数（其数量必须符合源代码行的长度）。调用宏时，在宏内无论何处出现参数名，赋值给这些参数的值将被替换。

宏可包含 MPASM 汇编器伪指令、PICmicro MCU 汇编指令或 MPASM 汇编器宏伪指令（如 *local*）。汇编器会持续处理宏直到遇到 *exitm* 或 *endm* 伪指令为止。

注： 使用宏前必须先定义宏，即不允许向前引用宏。

7.3 已定义的宏指令

有些伪指令是定义宏所独有的。它们只能用在宏环境中。

- 第 4.44 节 “macro——声明宏定义”
- 第 4.30 节 “exitm——退出宏”
- 第 4.25 节 “endm——结束宏定义”
- 第 4.31 节 “expand——扩展宏列表”
- 第 4.48 节 “noexpand——关闭宏扩展”
- 第 4.43 节 “local——声明局部宏变量”

编写宏时，可以使用上述任一条伪指令，以及任何其他汇编器支持的伪指令。

注： 不再支持先前特定于宏伪指令的 “dot” 格式语法。

7.4 宏定义

宏中可出现字符串替换和表达式求值。

命令	说明
<i>arg</i>	替换所提供的参数文本，作为调用宏的一部分。
<i>#v(expr)</i>	返回 <i>expr</i> 的整数值。通常用于创建具有常用前缀或后缀的惟一变量名。不能用于条件汇编伪指令（如 <i>ifdef</i> 和 <i>while</i> ）。

可以在宏内的任何地方使用参数，但参数不能作为常规表达式的一部分。

exitm 伪指令提供了另一种终止宏扩展的方法。在宏扩展时，该伪指令会停止当前的宏扩展并忽略针对宏的 *exitm* 与 *endm* 之间的所有代码。如果宏是嵌套的，*exitm* 会使代码生成返回到前一级宏扩展。

7.5 宏调用

一旦定义了宏，可以使用宏调用在源代码模块的任何位置调用宏，宏调用格式如下：

macro_name [*arg*, ..., *arg*]

其中 *macro_name* 是先前定义的宏的名称，并且参数将按要求提供。

宏调用本身不会占用任何存储单元。但宏扩展会从当前的存储单元开始。可以用逗号预留一个参数位置。在这种情况下，参数将为一个空字符串。用空白或分号可终止参数列表。

例 7-1: 宏代码生成

调用下列宏:

```
define_table macro
    local a = 0
    while a < 3
        entry#v(a) dw 0
        a += 1
    endwhile
endmacro
```

时, 会生成:

```
entry0 dw 0
entry1 dw 0
entry2 dw 0
entry3 dw 0
```

7.6 宏代码示例

以下是宏示例:

- 立即数到 RAM 数据的转换
- 常数比较

7.6.1 立即数到 RAM 数据的转换

该代码将任一 32 位立即数转换成 4 个独立的 RAM 数据值。在本例中, 立即数 0x12345678 按 0x12、0x34、0x56 和 0x78 放入规定的 8 位寄存器。任何立即数都可用该宏命令按照这种方式“拆开”。

```
#include p16F877A.inc

    udata 0x20
Out0    res    1    ; LSB
Out1    res    1    ; :
Out2    res    1    ; :
Out3    res    1    ; MSB

Unpack32    MACRO Var, Address ;Var = 32 bit literal to be unpacked
    BANKSEL Address            ;Address specifies the LSB start
    movlw    Address            ;Use FSR and INDF for indirect
    movwf    FSR                ;access to desired address

    movlw    Var & H'FF'        ;Mask to get LSB
    movwf    INDF               ;Put in first location
    movlw    Var >>D'08' & H'FF' ;Mask to get next byte of literal
    incf    FSR,F               ;Point to next byte
    movwf    INDF               ;Write data to next byte
    movlw    Var >>D'16' & H'FF' ;Mask to get next byte of literal
    incf    FSR,F               ;Point to next byte
    movwf    INDF               ;Write data to next byte
    movlw    Var >>D'24' & H'FF' ;Mask to get last byte of literal
    incf    FSR,F               ;Point to last byte
    movwf    INDF               ;Write data to last byte
    ENDM                        ;End of the Macro Definition

    ORG      0
Start
    Unpack32 0x12345678,Out0    ;TEST CODE for Unpack32 MACRO
    goto    $                  ;Put Unpack Macro here
                                ;Do nothing (loop forever)
END
```

7.6.2 常数比较

下面的宏可以作为另一个示例：

```
#include "pic16f877a.inc"
;
; compare file to constant and jump if file
; >= constant.
;
cfl_jge macro file, con, jump_to
    movlw con & 0xff
    subwf file, w
    btfsc status, carry
    goto jump_to
endm
```

可通过以下语句调用：

```
cfl_jge switch_val, max_switch, switch_on
```

将生成以下代码：

```
movlw max_switch & 0xff
subwf switch_val, w
btfsc status, carry
goto switch_on
```

第 8 章 错误、警告、消息和限制

8.1 简介

本章列出并详细说明了 MPASM 汇编器产生的错误消息、警告消息和一般消息。这些消息总是直接出现在列表文件中，在发生错误的行的上面一行。同时也列出了汇编器工具的限制。

如果没有指定 MPASM 汇编器选项，消息就被存储在错误文件（.err）中。如果使用了 /e- 选项（关闭错误文件），则消息会出现在屏幕上。如果 /q（安静模式）选项和 /e- 一起使用，则消息既不会出现在屏幕上也不会出现在错误文件中。但消息仍然会出现在列表文件中。

本章涉及的主题：

- 汇编器错误
- 汇编器警告
- 汇编器消息
- 汇编器限制

8.2 汇编器错误

下面按数字顺序列出了 MPASM 汇编器错误：

101 ERROR:

用户错误，用 error 伪指令调用。

102 Out of memory

宏、#define 或内部处理的存储器空间不足。

103 Symbol table full

没有更多的存储空间可供符号表使用。

104 Temp file creation error

不能创建临时文件。检查可用的磁盘空间。

105 Cannot open file

不能打开文件。如果是一个源文件，则文件可能不存在。如果是一个输出文件，可能旧版本被写保护了。

要检查写保护，请在 Windows 中右击由 MPLAB IDE 命名的文件。选择“属性”看是否选中了“只读”复选框。如果选中，MPLAB IDE 不能修改此文件并且会产生此错误消息。通常在把项目保存到 CD-R 或类似的一次性写入介质作为备份，然后将数据复制到计算机时会出现此错误消息。在把文件复制到 CD 时所有文件均标记为只读（它们在 CD-R 上不可以被更改），而在将文件复制到硬盘驱动器时，属性也随着它们转移，因此它们在硬盘驱动器上都是只读文件。避免这一错误的最佳方法是将所有文件都存档成一个文件，比如 *.ZIP，然后再从 CD 上恢复它们。存档文件会保持原始文件的属性。

106 String substitution too complex

进行过的字符串替换太复杂。检查 #define 的嵌套。

107 Illegal digit

数字中有非法数字。二进制数的有效数字为 0—1，八进制数的有效数字为 0—7、十进制数的有效数字为 0—9，十六进制数的有效数字为 0—9、a—f 以及 A—F。

108 Illegal character

标号中有非法字符。标号中的有效字符为字母（a—f 和 A—F）、数字（0—9）、下划线（_）和问号（?）。标号的开头不能为数字。

109 Unmatched (

前半个圆括号没有匹配的后半个圆括号。例如：

```
DATA (1+2.
```

110 Unmatched)

后半个圆括号没有匹配的前半个圆括号。例如：

```
DATA 1+2).
```

111 Missing symbol

equ 或 set 伪指令没有赋值符号。

112 Missing operator

表达式中遗漏了算术运算符。如 DATA 1 2。

113 Symbol not previously defined

引用的符号尚未定义。检查代码中使用的所有符号的拼写和声明的位置。只有地址可以用作向前引用。常数和变量在使用之前必须声明。

当在项目中使用 #include 文件时有时会发生这种情况。如果来自包含文件的文本被插入到 #include 语句的位置并且在该点之前使用了标号，就可能会发生此错误。而打字错误、拼写错误或者标号中的大小写更改都可能导致此错误发生。MyLabel 和 Mylabel 是不同的，除非关闭了区分大小写（它在默认情况下是打开的）。此外，goto MyLabel 不会查找到代码 Mylabel 或 Mylable。首先检查这几类错误。一个通则是将包含文件放在每个文件的顶部。如果这看起来很混乱，可以包含其他包含文件中的文件。

114 Divide by zero

在表达式运算过程中遇到除以零。

115 Duplicate label

标号在多个单元被声明为一个常数（如使用 equ 或 cblock 伪指令）。

116 Address label duplicated or different in second pass

在两个单元使用了同一个标号。或者，标号只使用了一次但是第二次运行到该标号时得到了一个不同的单元。当用户尝试写页位设置宏（此宏会根据目标来产生不同数量的指令）时经常会发生这种情况。

117 Address wrapped around 0

对于 PIC12/16 器件来说，单元计数器只能计数到 0xFFFF。在这之后，它回到 0。错误 117 之后将显示错误 118。

118 Overwriting previous address contents

此地址的代码是前面产生的。

119 Code too fragmented

此代码被分成了太多段。此错误非常罕见，而且只会在引用地址大于 32K 的源代码中发生（包括配置位）。

120 Call or jump not allowed at this address

在此地址处不能进行调用和跳过。例如，PIC16C5x 系列上的 CALL 目标必须在页的下半部份。

121 Illegal label

在某些伪指令行上不允许标号。只要把标号放在伪指令上方的自己的行中。而且，high、low、page 和 bank 都不允许作为标号。

122 Illegal opcode

记号（Token）是无效的操作码。

123 Illegal directive

所选的处理器不允许伪指令，例如，有 ID 单元的器件上的 __idlocs 伪指令。

124 Illegal argument

非法的伪指令参数；如 list foobar。

125 Illegal condition

不良条件汇编。如不匹配的 endif。

126 Argument out of range

操作码或伪指令参数超出了有效范围；如 TRIS 10。

127 Too many arguments

为一个宏调用指定了太多参数。

128 Missing argument(s)

宏调用或者操作码没有足够的参数。

129 Expected

需要一种特定类型的参数。会提供所需参数的列表。

130 Processor type previously defined

选择了其他系列的处理器。

131 Processor type is undefined

在定义处理器之前代码已经生成了。注意在定义处理器之前，操作码集是未知的。

132 Unknown processor

选定的处理器不是有效处理器。

133 Hex file format INHX32 required

指定了大于 32K 的地址。

134 Illegal hex file format

list 伪指令中指定了非法的 hex 文件格式。

135 Macro name missing

定义的宏没有名称。

136 Duplicate macro name

宏名称重复了。

137 Macros nested too deep

超过了最大的宏嵌套级别。

138 Include files nested too deep

超过了最大的包含文件嵌套级别。

139 Maximum of 100 lines inside WHILE-ENDW

while-endw 最多只能包含 100 行。

140 WHILE must terminate within 256 iterations

while-endw 循环必须在 256 次迭代后终止。这是为了阻止无限循环汇编。

141 WHILEs nested too deep

超过了最大的 while-endw 嵌套级别。

142 IFs nested too deep

超过了最大的 if 嵌套级别。

143 Illegal nesting

必须将宏、if 和 while 完全嵌套，不能重叠。如果 while 循环中有 if，则必须在 endw 之前有一个 endif。

144 Unmatched ENDC

发现 endc 但没有 cblock。

145 Unmatched ENDM

发现 endm 但没有 macro 定义。

146 Unmatched EXITM

发现 `endm` 但没有 `macro` 定义。

147 Directive/operation only allowed when generating an object file

所示指令 / 操作数只有在生成了可链接的目标文件时才有意义。它在生成绝对代码时不能使用。

148 Expanded source line exceeded 200 characters

在 `#define` 和宏参数替换之后源代码行的最大长度为 200 个字符。注意 `#define` 替换不包括注释，但是宏参数替换包括注释。

149 Directive only allowed when generating an object file

某些伪指令，如 `global` 和 `extern`，只有在生成可链接目标文件时才有意义。它们在生成绝对代码时不能使用。

150 Labels must be defined in a code or data section when making an object file

当生成可链接目标文件时，必须在数据或代码段内定义所有数据和代码地址标号。可以在数据和代码段之外定义由 `equ` 和 `set` 伪指令定义的符号。

151 Operand contains unresolvable labels or is too complex

在生成目标文件时，操作数的格式必须为 `[高 | 低]` (`[可重定位地址标号]+[偏移量]`)。

152 Executable code and data must be defined in an appropriate section

在生成可链接目标文件时，必须将所有可执行代码和数据声明都放在相应的段内。

153 Page or Bank bits cannot be evaluated for the operand

`pagesel`、`banksel` 或 `bankisel` 伪指令的操作数必须是可重定位地址标号或常数。

154 Each object file section must be contiguous

除 `udata_ovr` 段以外的目标文件段不能在一个源代码文件内被断开或继续。要解决这个问题，可以为每个段取一个自用的名称，或者移动代码和数据声明，使每个段成为连续的。如果不同类型的两个段同名，也会产生这种错误。

155 All overlaid sections of the same name must have the same starting address

如果声明了多个名字相同的 `udata_ovr` 段，它们的起始地址必须全部相同。

156 Operand must be an address label

当生成目标文件时，只有代码或数据段中的地址标号可被声明为全局标号。不能导出 `set` 或 `equ` 伪指令声明的变量。

157 ORG at odd address

对于 PIC18 器件，不能将 `org` 放在奇数地址，只能放在偶数地址。具体请查询器件数据手册。

158 Cannot use RES directive with odd number of bytes

对于 PIC18 器件，不能使用 `res` 指定奇数个字节，只能指定偶数个字节。具体请查询器件数据手册。

159 Cannot use FILL directive with odd number of bytes

对于 PIC18 器件，不能使用 `fill` 在数据中填充奇数个字节，只能填充偶数个字节。具体请查询器件数据手册。

160 CODE_PACK directive not available for this part; substituting CODE

`code_pack` 伪指令只能与可字节寻址的 ROM 一起使用。

161 Non-negative value required for this context.

某些上下文要求非负的值。

162 Expected a section name

某些运算符和伪运算符将段的名称作为操作数。段名称的词法格式就是其标识符的词法格式，加上可选的前缀“.”。

163 __CONFIG directives must be contiguous

不要将其他代码放在两个 `__config` 伪指令声明之间。

164 __IDLOC directives must be contiguous

不要将其他代码放在两个 `__idloc` 伪指令声明之间。

165 extended mode not available for this device

PIC18 器件不支持扩展模式。

166 left bracket missing from offset operand

偏移量的左括号遗漏，即应为 `[0x55`。

167 right bracket missing from offset operand

偏移量的右括号遗漏，即应为 `0x55]`。

168 square brackets required around offset operand

偏移量需要加上方括号，如 `[0x55]`

169 access bit cannot be specified with indexed mode

在使用变址模式时，不能指定快速操作位。

170 expression within brackets must be constant

括号内指定的表达式非常数值。

171 address specified is not in access ram range of [0x60, 0xFF]

在使用快速操作 RAM 时，必须在指定的快速访问存储区范围之内进行寻址。

172 PCL, TOSL, TOSH, or TOSU cannot be destination of MOVFF or MOVSF

不能用 movff 或 movsf 命令写入这些寄存器。

173 source file path exceeds 62 characters

MPASM 汇编器限制源文件路径名称（即路径加文件名的长度）最多为 62 个字符。欲知更多信息，请参见第 8.5 节“汇编器限制”。

174 __CONFIG directives must be listed in ascending order

以升序列出 config 伪指令的配置寄存器，即，

```
__CONFIG    _CONFIG0, _CP_OFF_0
__CONFIG    _CONFIG1, _OSCS_OFF_1 & _RCIO_OSC_1
__CONFIG    _CONFIG2, _BOR_ON_2 & _BORV_25_2
:
```

175 __IDLOCS directives must be listed in ascending order

以升序列出 __idlocs 伪指令的 ID 寄存器，即，

```
__idlocs    _IDLOC0, 0x1
__idlocs    _IDLOC1, 0x2
__idlocs    _IDLOC2, 0x3
:
```

176 CONFIG Directive Error:

在 config 伪指令语法中发现错误。

177 __CONFIG directives cannot be used with CONFIG directives

在代码中赋值配置位时请勿混用 __config 伪指令和 config 伪指令。

178 __CONFIG Directive Error:

在 __config 伪指令语法中发现错误。

UNKNOWN ERROR

发生了内部应用程序错误。（### 是最后定义的错误加 1 的值。）

如果不能解决此错误，请联络 Microchip 现场应用工程师（FAE）或 Microchip 技术支持。

8.3 汇编器警告

下面按数字顺序列出了 MPASM 汇编器警告：

201 Symbol not previously defined.

被 #undefine 的符号前面并未定义过。

202 Argument out of range. Least significant bits used.

参数与所分配的空间大小不一致。例如，立即数必须为 8 位。

203 Found opcode in column 1.

在为标号预留的第一列内发现操作码。

204 Found opcode in column 1.

在为标号保留的第一列内发现伪操作码。

205 Found directive in column 1.

在为标号保留的第一列内发现伪指令。

206 Found call to macro in column 1.

在为标号保留的第一列内发现对宏的调用。

207 Found label after column 1.

在第一列后发现标号，这通常由于操作码拼错而造成。

208 Label truncated at 32 characters.

最大标号长度为 32 个字符。

209 Missing quote.

文本字符串或字符中遗漏了引号。如 `DATA 'a`。

210 Extra ","

在行末发现多余的逗号。

211 Extraneous arguments on the line.

在行上发现多余的参数。

212 Expected (ENDIF)

需要 `endif` 语句，即，使用了 `if` 语句但没有 `endif` 语句。

213 The EXTERN directive should only be used when making a .o file.

只有创建了目标文件时，`extern` 伪指令才有意义。此警告已被 Error 149 代替了。

214 Unmatched (

发现了不完整的括号。如果括号不用于表示等式的运算顺序，则使用此警告。

215 Processor superseded by command line.

处理器在命令行和源文件中指定了。在命令行中指定的处理器优先。

如果在 MPLAB IDE 中使用汇编，请在 [Configure>Select Device](#) 中将器件设置为匹配源文件。

216 Radix superseded by command line.

基数在命令行和源文件中指定了。在命令行中指定的基数优先。

217 Hex file format specified on command line.

hex 文件格式在命令行和源文件中指定了。在命令行中指定的 hex 文件格式优先。

218 Expected DEC, OCT, HEX. 使用 HEX。

指定了错误的基数。

219 Invalid RAM location specified.

如果使用了 `__maxram` 和 `__badram` 伪指令，此警告标志出使用了由这两条伪指令已声明为无效的 RAM 单元。注意所提供的头文件包含每个处理器的 `__maxram` 和 `__badram`。

220 Address exceeds maximum range for this processor.

指定的 ROM 单元超过了处理器存储空间大小的范围。

221 Invalid message number.

指定的显示或隐藏的消息号是无效的消息号。

222 Error messages cannot be disabled.

不能用 `errorlevel` 命令禁止错误消息。

223 Redefining processor

`list` 或 `processor` 伪指令选择了已选择过的处理器。

224 Use of this instruction is not recommended.

此指令已作废，建议现在不要使用。但是为了保留旧代码，仍然支持此指令。

225 Invalid label in operand

操作数是无效的地址。例如，如果用户尝试执行一条 `CALL` 指令来调用 `MACRO` 名称。

226 Destination address must be word aligned

目标地址与程序存储器字的起始部分未对齐。请使用偶数字节来为此器件指定地址。

227 Substituting RETLW 0 for RETURN pseudo-op

使用 `retlw 0` 而不是 `return` 来恢复程序执行。

228 Invalid ROM location specified

指定的数据存储单元对于指定的操作无效或者不存在。

229 extended mode is not in effect -- overridden by command line

命令行选项禁止了扩展模式操作。

230 `__CONFIG` has been deprecated for PIC18 devices. Use directive `CONFIG`.

对于 PIC18 MCU 器件，尽管您仍可使用 `__config` 伪指令，但我们强烈建议您使用 `config` 伪指令（前面无下划线）。对于 PIC18FXXJ MCU，您必须使用 `config` 伪指令。

UNKNOWN WARNING

发生了内部应用程序错误。(### 是最后定义的警告加 1 的值。)

但是，这还不至于严重到不能汇编代码，因为这只是警告，并不是错误。

8.4 汇编器消息

下面按数字顺序列出了 MPASM 汇编器消息：

301 MESSAGE:

用户可定义的消息，用 `messg` 伪指令调用（见第 4.47 节“`messg`——创建用户定义的消息”）。

302 Register in operand not in bank 0. Ensure that bank bits are correct.

这是一条常见的提示消息，告诉您被快速操作的变量不在存储区 0 中。添加此消息是为了提醒您检查代码，特别是除存储区 0 之外的其他存储区中的代码。复查 `banksel` 段（第 4.7 节“`banksel`——生成存储区选择代码”）和 `bankisel`（第 4.6 节“`bankisel`——生成间接存储区选择代码（PIC12/16 MCU）”）上的代码，确保您的代码不论何时从一个存储区更改到另一个存储区（包括存储区 0）都使用存储区位。

由于汇编器和链接器不能分辨代码将使用的路径，所以在使用非存储区 0 中的变量时总是会得到此消息。可以使用 `errorlevel` 命令来打开和关闭此消息或其他消息，但是需要当心，因为将此消息关闭可能无法察觉存储区选择的问题。欲知更多有关 `errorlevel` 的信息，请参见第 4.29 节“`errorlevel`——设置消息级别”。

用于页面选择的消息 306 与该消息类似。

303 Program word too large. Truncated to core size.

对于所选器件的内核（程序存储器）字长来说程序字（指令宽度）太长。因此，字被截断为合适的长度。

例如，14 位的指令必须被截断为 12 位才能被 PIC16F54 使用。

304 ID Locations value too large. Last four hex digits used.

ID 单元仅允许四个十六进制数。

305 Using default destination of 1 (file).

如果没有指定目标位则使用默认设置。通常导致此消息出现的代码是在寄存器名称后面缺少 `,W` 或 `,F`，但有时也可能因为输入了 `movf` 而不是 `movwf` 而产生的缺陷。

最好是改正导致此消息出现的所有代码。默认目标可能不是您要存储值的地方，而且还可能导致代码操作异常。

306 Crossing page boundary -- ensure page bits are set.

生成的代码跨越了页边界。这是一条提示消息，告诉您代码指向的标号位于除第 0 页之外的其他页上。这并不是错误或警告，只是提示您检查页面选择位。在这一点之前使用 `pagesel` 伪指令（第 4.52 节“`pagesel`——生成页面选择代码（PIC10/12/16 MCU）”），并记住返回页 0 时再次使用 `pagesel` 伪指令。

由于汇编器不能分辨代码将使用的路径，所以只要使用页 0 之外的任何页中的标号都会产生此消息。可以使用 `errorlevel` 命令来打开和关闭此消息或其他消息，但是需要当心，因为将此消息关闭可能无法察觉页面选择的问题。欲知更多有关 `errorlevel` 的信息，请参见第 4.29 节“`errorlevel`——设置消息级别”。

用于存储区选择的消息 302 与该消息类似。

307 Setting page bits.

用 `LCALL` 或 `LGOTO` 伪操作码设置页面选择位。

308 Warning level superseded by command line value.

警告级别在命令行和源文件中指定了。在命令行中指定的警告级别优先。

309 Macro expansion superseded by command line.

宏扩展在命令行和源文件中指定了。在命令行中指定的宏扩展优先。

310 Superseding current maximum RAM and RAM map.

`__maxram` 伪指令已在前面使用过。

311 Operand of HIGH operator was larger than H'FFFF'.

由 `high` 伪指令返回的地址的高字节大于 `0xFFFF`。

312 Page or Bank selection not needed for this device. 未产生代码。

如果器件仅包含一个 ROM 页或 RAM 存储区，则不需要选择页面或存储区，并且任何 `pagesel`、`banksel` 或 `bankisel` 伪指令都不会产生代码。

313 CBLOCK constants will start with a value of 0.

如果源文件中的首个 `cblock` 没有指定起始值，将会产生此消息。

314 LFSR instruction is not supported on some versions of the 18Cxx2 devices.

欲知更多信息请参见消息 315。

315 Please refer to Microchip document DS80058A for more details

Microchip 网站上有此文档的可下载 PDF 和 PIC18CXX2 芯片 / 数据手册勘误表。

316 W Register modified.

工作 (W) 寄存器已被修改。

317 W Register not modified. BSF/BCF STATUS instructions used instead.

工作 (W) 寄存器未修改。

318 Superseding current maximum ROM and ROM map.

操作将导致超出最大 ROM 空间。

UNKNOWN MESSAGE

发生了内部应用程序错误。（### 是最后定义的消息加 1 的值。）

但是，这还不至于严重到不能汇编代码，因为这只是消息，并不是错误。

8.5 汇编器限制

8.5.1 一般限制

- 在 MPASM 汇编器生成的调试（COD）文件中将文件和路径名称的长度限制为 **62** 个字符。在汇编文件名和 / 或路径名称很长的单个文件时，此限制会导致问题。

解决方法：

- 缩短文件名或将文件移到最靠近根目录的目录中（让路径名称更短），然后再尝试汇编文件。
- 为长的目录链创建映射驱动器。
- 将链接器与汇编器同时使用，而不要单独使用汇编器来生成输出文件。
MPLINK 链接器没有字符限制。
- 如果指定了完全合格的路径，就只会搜索到该路径。否则，搜索顺序是：（1）当前工作目录、（2）源文件目录和（3）MPASM 汇编器可执行文件目录。
- 源文件行（扩展的）限制为 **200** 个字符。
- 文件名被限制为 8.3 格式（仅 mpasm.exe）。

8.5.2 伪指令限制

- 不要在宏中使用 #includes。
- if 伪指令限制
 - 最大嵌套深度 = 16
- include 伪指令限制
 - 最大嵌套深度 = 5
 - 最大文件数 = 255
- macro 伪指令限制
 - 最大嵌套深度 = 16
- while 伪指令限制
 - 最大嵌套深度 = 8
 - 每个循环的最大行数 = 100
 - 最大迭代次数 = 256

第 2 部分——MPLINK 目标链接器

第 9 章	MPLINK 链接器概述	171
第 10 章	链接器接口	179
第 11 章	链接描述文件	181
第 12 章	链接器处理过程	187
第 13 章	应用程序示例	191
第 14 章	错误、警告和常见问题	213

注:

第 9 章 MPLINK 链接器概述

9.1 简介

本章概述了 MPLINK 目标链接器及其功能。

本章涉及以下主题：

- MPLINK 链接器的定义
- MPLINK 链接器工作原理
- MPLINK 链接器的功能
- 所支持的链接器平台
- 链接器工作原理
- 链接器输入 / 输出文件

9.2 MPLINK 链接器的定义

MPLINK 目标链接器（链接程序）将 MPASM 汇编器或 MPLAB C18 C 编译器生成的目标模块组合到一个可执行（hex）文件中。链接器也接受目标文件的库（如 MPLIB 目标库管理器生成的库）作为输入。链接过程由链接描述文件控制，该文件也将输入到 MPLINK 链接器。

欲了解更多信息，请参见第 1 章“MPASM 汇编器概述”。了解更多关于 MPLAB C18 的信息，请参见推荐读物中所列的 C 编译器文档。

9.3 MPLINK 链接器工作原理

MPLINK 链接器执行许多功能：

- 定位代码和数据。链接器获取代码或数据作为输入的重定位目标文件。它使用链接描述文件决定代码在程序存储器中的存储位置和变量在 RAM 中的存储位置。
- 解析地址。源文件中的外部引用在目标文件中生成重定位项。链接器在定位了代码和数据后，使用该重定位信息将外部引用更新为实际地址。
- 生成可执行文件。生成 .hex 文件，该文件可被编程到 PICmicro MCU 中或载入仿真器或模拟器来执行。
- 配置栈大小和位置。让 MPLAB C18 为动态栈保留 RAM 空间。
- 识别地址冲突。检查以确保不会把程序 / 数据指派给已分配的或保留的空间。
- 提供符号调试信息。输出一个文件，MPLAB IDE 在调试源代码时使用该文件跟踪地址标号、变量位置和行 / 文件信息。

9.4 MPLINK 链接器的功能

MPLINK 链接器可生成模块化的可重用代码。对链接过程的控制是通过一个链接描述文件和命令行选项完成的。链接器确保能够解析所有的符号引用并且代码和数据大小适用于所使用的 PICmicro MCU 器件。

MPLINK 链接器提供以下功能：

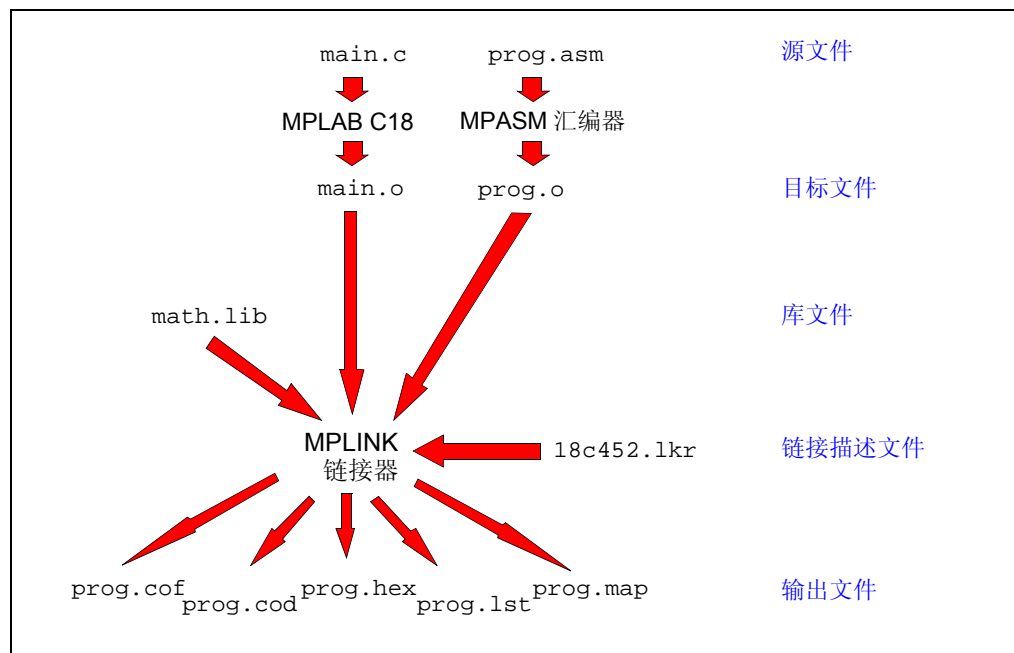
- 可重用的源代码。可以在一个可重用的小型模块中编译应用程序。
- 库。可以制作相关函数的库，这些函数可用于创建高效、兼容性好的应用程序。
- MPLAB C18。要使用 Microchip PIC18 器件编译器需要使用 MPLINK 链接器，而编译器能与预编译库和 MPASM 汇编器配合使用。
- 集中分配存储器。通过使用特定于应用程序的链接描述文件，预编译的目标和库可与新的源代码模块结合在一起，并在链接时高效地置入可用的存储器中。
- 加速开发过程。由于在代码每次改变时不需要重新编译已测试的模块和库，因此可缩短编译时间。

9.5 所支持的链接器平台

MPLINK 链接器是一个 Windows 32 控制台应用程序，适用于 Windows 95/98 和 Windows NT/2000/XP 平台。

9.6 链接器工作原理

下面的框图说明了 MPLINK 链接器如何与其他 Microchip 工具配合工作。



MPLINK 链接器将由 MPASM 汇编器或 MPLAB C18 C 编译器生成的多个输入目标模块组合成单个输出可执行（.hex）文件。链接描述文件指导链接器如何组合这些模块。

当链接器与 MPASM 汇编器和 / 或 MPLAB C18 C 编译器配合使用完成对可重定位目标模块的汇编或编译后，执行链接程序。数据的实际地址和函数的位置在执行 MPLINK 链接器时进行分配。这意味着可以通过链接描述文件指导链接器将代码和数据保存在存储器中已命名的区域，如果未指定存储区域，则可保存在任何可用的存储器中。

链接描述文件还必须告知 MPLINK 链接器目标 PICmicro MCU 器件中可用的 ROM 和 RAM 存储区。然后，链接器可分析所有输入文件，将应用程序保存到 ROM 中，同时将数据变量保存到可用的 RAM 中。如果待保存的代码或变量过多，链接器就会给出一条错误信息。

MPLINK 链接器还可灵活地指定哪些数据存储区是可重用的，这样不同的程序就能够共享有限的 RAM 空间（这些程序从不相互调用，并且在执行间不要求数据被保留下来）。

当使用 C 编译器时，大多数 PICmicro MCU 外设和许多标准 C 函数都可以使用库。链接器只从包含的库中提取和链接当前应用程序所需的各个目标文件。这就意味着可以非常有效地运用较大的库。

MPLINK 链接器组合所有的输入文件以生成可执行输出文件，并确保能够解析所有地址。各种输入模块中的任何函数若试图访问未分配的数据，或试图调用未创建的程序，将导致链接器报错。

MPLINK 链接器也生成符号信息，用于在 MPLAB IDE 中调试应用程序（.cof，.lst 和 .map 文件）。

9.7 链接器输入 / 输出文件

MPLINK 链接器将多个目标文件组合成一个可执行 hex 文件。

输入文件

目标文件（.o）	由源文件生成的可重定位代码。
库文件（.lib）	为方便起见，将一组目标文件组合在一起的集合。
链接描述文件（.lkr）	特定处理器 / 项目的存储器分配描述。

输出文件

COFF 目标模块文件（.cof 和 .out）	MPLAB IDE v6.xx 及以后版本使用的调试文件。
符号和调试文件（.cod）	MPLAB IDE v5.xx 及以前版本使用的调试文件。
Hex 文件格式（.hex，.hxl 和 .hxx）	不带调试信息的十六进制文件。适用于编程。
列表文件（.lst）	原始源代码，与最终二进制代码一一对应。 注： 要求链接器能找到原始的源文件。
映射文件（.map）	显示链接后的存储器分配情况。指示已使用和未使用的存储区。

9.7.1 目标文件 (.o)

目标文件是由源文件生成的可重定位代码。MPLINK 链接器根据链接描述文件将目标文件和库文件组合到单个输出文件中。

可用 MPASM 汇编器从源文件创建目标文件，可用 MPLIB 库管理器从目标文件创建库文件。

9.7.2 库文件 (.lib)

使用库能方便地组合相关的目标模块。库文件可以由 MPLIB 库管理器从目标文件创建。欲了解更多关于库管理器的信息，参见第 15 章“MPLIB 库管理器概述”。

9.7.3 链接描述文件 (.lkr)

链接描述文件是 MPLINK 链接器的命令文件。欲了解更多关于链接描述文件的信息，参见第 11 章“链接描述文件”。

标准链接描述文件位于：

C:\Program Files\Microchip\MPASM Suite\LKR

在链接过程中，如果 MPLINK 链接器不能将引用解析为符号，它将搜索命令行指定的库或链接描述文件以尝试解析该引用。如果在库文件中找到了定义，则将会把包含定义的目标文件包含在链接过程中。

9.7.4 COFF 目标模块文件 (.cof 和 .out)

MPLINK 链接器生成一个 COFF 文件，该文件向 MPLAB IDE v6.xx 或以后版本提供调试信息。MP2COD.EXE 从 COFF 文件生成 COD 文件和列表文件，而 MP2HEX.EXE 生成 hex 文件。

9.7.5 符号和调试文件 (.cod)

MPASM 汇编器和 MPLINK 链接器都能生成 COD 文件，供 MPLAB IDE v5.xx 及早期版本使用。欲了解更多关于该格式的信息，参见第 1.7.7 节“符号和调试文件 (.cod)”。

对于 MPLINK 链接器，MP2COD.EXE 使用 COFF 文件生成 COD 和列表文件。要阻止 COD 和链接文件生成，使用 /w 选项。

9.7.6 Hex 文件格式 (.hex, .hxl 和 .hxx)

MPASM 汇编器和 MPLINK 链接器都能生成 hex 文件。欲了解更多关于该格式的信息，参见第 1.7.5 节“Hex 文件格式 (.hex, .hxl 和 .hxx)”。

对于 MPLINK 链接器，MP2HEX.EXE 使用 COFF 文件生成 hex 文件。要阻止 hex 文件生成，使用 /x 选项。

9.7.7 列表文件 (.lst)

MPLINK 链接器列表文件提供了源代码到目标代码的映射。它还提供了一个包括符号值、存储器使用情况信息以及产生的错误、警告和消息数的列表。可通过以下方式在 MPLAB IDE 中查看此文件：

1. 选择 **File>Open** 打开“打开”对话框
2. 从“文件类型”下拉列表中选择“List files (*.lst)”
3. 查找所需的列表文件
4. 单击列表文件名
5. 单击 **Open**

MPASM 汇编器和 MPLINK 链接器都能生成列表文件。欲了解更多关于 MPASM 汇编器的信息，参见第 1.7.3 节“列表文件 (.lst)”。

另一种查看列表文件的方式是使用 MPLAB IDE 的 Disassembly 窗口 ([View>Disassembly Listing](#)) 中的信息。

对于 MPLINK 链接器，MP2COD.EXE 使用 COFF 文件生成 COD 和列表文件。要阻止 COD 和链接器列表文件生成，使用 /w 选项。

例 9-1: MPLINK 链接器列表文件

在每一页的顶端会显示 COFF 至 COD 文件转换器版本号和列表文件生成的数据。

第一列包含了存储器中要存放该代码的基地址。第二列为机器指令而保留。这是将被 PICmicro MCU 执行的代码。第三列显示反汇编代码。第四列列出了相关的源代码行。第五列列出了与源代码行相关的文件。

注： 由于受到页面宽度限制，将一些备注缩短，用“..”代替。另外，相关文件名也用数字代替，即 (1) 和 (2)。参见在文件最后列出的实际文件的路径和名称。

MP2COD 3.80.03, COFF to COD File Converter
Copyright (c) 2004 Microchip Technology Inc.
Listing File Generated: Tue Nov 02 14:33:23 2004

Address	Value	Disassembly	Source	File
-----	-----	-----	-----	----
			#include p18f452.inc	(1)
			LIST	(2)
			; P18F452.INC Standard Header File,...	(2)
			LIST	(2)
			udata	(1)
			Dest res 1	(1)
			RST code 0x0	(1)
000000	ef16	GOTO 0x2c	goto Start	(1)
000002	f000			(1)
			PGM code	(1)
			Start movlw 0x0A	(1)
00002c	0e0a	MOVLW 0xa	movwf Dest	(1)
00002e	6f80	MOVWF 0x80,0x1	bcf Dest, 3	(1)
000030	9780	BCF 0x80,0x3,0x1	goto Start	(1)
000032	ef16	GOTO 0x2c		(1)
000034	f000		end	(1)

其中:

- (1) = D:\Projects32\PIC18F452\SourceReloc.asm
- (2) = C:\Program Files\Microchip\MPASM Suite\p18f452.inc

9.7.8 映射文件 (.map)

MPLINK 链接器生成的映射文件可在 MPLAB IDE 查看，选择 **File>Open** 并选择您在 MPLINK 链接器选项中指定的文件。映射文件提供有关最终输出文件中源代码符号绝对地址的信息。它还提供有关存储器使用的信息，指明已使用和未使用的存储空间。在每一次重编译后自动加载该窗口。

映射文件包含 4 个表。第一个表（段信息）显示每段的信息，包括段名、类型、起始地址、段是驻留于程序存储器还是数据存储器中，以及字节大小。

有 4 种类型的段：

- 代码
- 已初始化的数据（idata）
- 未初始化的数据（udata）
- 已初始化的 ROM 数据（romdata）

下表就是映射文件中某个段表的示例：

Section Info				
Section	Type	Address	Location	Size(Bytes)
Reset	code	0x000000	program	0x000002
.cinit	romdata	0x000021	program	0x000004
.code	code	0x000023	program	0x000026
.udata	udata	0x000020	data	0x000005

第二个表（程序存储器使用情况）列出了已使用的程序存储器地址并提供程序存储器总体利用率的统计数据。例如：

Program Memory Usage	
Start	End
0x000000	0x000005
0x00002a	0x00002b
0x0000bc	0x001174
0x001176	0x002895

10209 out of 32786 program addresses used, program memory utilization is 31%

映射文件中的第三个表（符号按名称排序）提供输出模块中符号信息。该表按符号名称排列，包含符号地址、符号是驻留在程序存储器中还是在数据存储器中、符号具有外部链接还是静态链接，以及文件的路径和名称。下表就是映射文件中以符号名称排序的符号表示例：

Symbols - Sorted by Name

Name	Address	Location	Storage	File
call_m	0x000026	program	static	C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
loop	0x00002e	program	static	C:\MPASM assemblerV2\MUL8X8.ASM
main	0x000024	program	static	C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
mpy	0x000028	program	extern	C:\MPASM assemblerV2\MUL8X8.ASM
start	0x000023	program	static	C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
H_byte	0x000022	data	extern	C:\MPASM assemblerV2\MUL8X8.ASM
L_byte	0x000023	data	extern	C:\MPASM assemblerV2\MUL8X8.ASM
count	0x000024	data	static	C:\MPASM assemblerV2\MUL8X8.ASM
mulcnd	0x000020	data	extern	C:\MPASM assemblerV2\MUL8X8.ASM
mulplr	0x000021	data	extern	C:\MPASM assemblerV2\MUL8X8.ASM

映射文件中的第四个表（符号——按地址排序）提供了第 2 个表提供的相同信息，但它以符号地址排列而不是以符号名称排列。

如果发生链接器错误，就不能生成完整的映射文件。然而，如果有 /m 选项，则将创建一个错误映射文件。错误映射文件只包含段信息而不提供符号信息。错误映射文件列出了所有在发生错误时已成功分配的段。该文件连同错误信息一起应提供足够的信息，以确定段不能被分配的原因。

注:

第 10 章 链接器接口

10.1 简介

本章介绍 MPLINK 目标链接器的用法。

当安装 MPLAB IDE 或 MPLAB C18 时，MPLINK 链接器 (mplink.exe) 也被安装。

本章涉及以下主题：

- MPLAB IDE 接口
- 命令行接口
- 命令行示例

10.2 MPLAB IDE 接口

MPLINK 链接器通常在 MPLAB IDE 项目中与 MPASM 汇编器配合使用以生成可重定位的代码。欲了解关于其用法的更多信息，参见“**PICmicro 语言工具和 MPLAB IDE**”。

链接器也可在 MPLAB IDE 中与 MPLAB C18 编译器配合使用。欲知有关 Microchip 编译器的更多信息，请参见推荐读物中列出的 MPLAB C18 C 编译器文档。

10.3 命令行接口

MPLINK 链接器可在 MPLAB IDE 中使用，或直接通过命令行调用。

当在 MPLAB IDE 中使用时，MPLINK 链接器的所有选项都可在 MPLINK Linker 选项卡（Project>Build Options 对话框）上找到。

当在批处理文件中使用 MPLINK 链接器或直接通过命令行使用时，用下面的语法调用链接器：

```
mplink cmdfiles objfiles [libfiles] [options]
```

cmdfile 是链接器命令文件的名称。所有的链接器命令文件的扩展名必须为 .lkr。

objfile 是汇编器或编译器生成的目标文件的名称。所有的目标文件的扩展名必须为 .o。

libfile 是库管理器创建的库文件的名称。所有的库文件的扩展名必须为 .lib。

option 是下面说明的链接器命令行选项之一。

选项	说明
<code>/a hexformat</code>	指定 hex 输出文件的格式。
<code>/h, /?</code>	显示帮助屏幕。
<code>/k pathlist</code>	添加链接描述文件所在目录的搜索路径。
<code>/l pathlist</code>	添加库文件所在目录的搜索路径。
<code>/m filename</code>	创建映射文件 <i>filename</i> 。
<code>/n length</code>	指定每个列表页的行数。
<code>/o filename</code>	指定输出文件 <i>filename</i> 。默认为 <code>a.out</code> 。
<code>/q</code>	安静模式。
<code>/w</code>	禁止 <code>mp2cod.exe</code> 。使用该选项将阻止生成 <code>.cod</code> 文件和 <code>.lst</code> 文件。
<code>/x</code>	禁止 <code>mp2hex.exe</code> 。使用该选项将阻止生成 <code>.hex</code> 文件。

对命令行参数的顺序没有要求，但改变参数的顺序会影响链接器的操作。值得一提的是，在命令行或命令文件中遇到其他的库 / 目标目录搜索路径时，此时这些搜索路径将被附加到当前的库 / 目标目录搜索路径之后。

按目录在库 / 目标目录搜索路径中的顺序搜索库和目标文件。所以，改变目录的顺序可能会改变被选择的文件。

`/o` 选项提供生成的输出 **COFF** 文件的名称用于 **MPLAB IDE** 调试。同时还生成用于编程的 **Intel** 格式的 **hex** 文件。该文件与输出 **COFF** 文件的文件名相同，但文件扩展名为 `.hex`。如果没有 `/o` 选项，那么默认的输出 **COFF** 文件的文件名为 `a.out`，相应的 **hex** 文件的文件名为 `a.hex`。

10.4 命令行示例

下面所示为 **MPLINK** 链接器命令行示例。

```
mplink 18f452.lkr main.o funct.o math.lib /m main.map /o main.out
```

该命令行指导 **MPLINK** 链接器使用 `18f452.lkr` 链接描述文件链接输入模块 `main.o`、`funct.o` 和预编译库文件 `math.lib`。它还指导链接器生成名为 `main.map` 的映射文件。`main.o` 和 `funct.o` 必须已经编译或汇编好。如果在链接过程中没有发生错误，则就会生成输出文件 `main.cof` 和 `main.hex`。

第 11 章 链接描述文件

11.1 简介

链接描述文件是链接器的命令文件。它们指定以下内容：

- 目标部件的程序和数据存储器区域
- 堆栈大小和位置（对于 MPLAB C18）
- 源代码中的逻辑代码段与程序和数据区域的映射

链接描述文件伪指令组成了用于控制链接行为的命令语言。链接描述文件伪指令分为四个基本类别。下列主题将讨论每个类别的伪指令以及一些有用的链接描述文件忠告。

注： “//” 表示链接描述文件的注释，即，“//” 和行末之间的任何文本均被忽略。

本章涉及的主题：

- 标准链接描述文件
- 链接描述文件命令行信息
- 链接描述文件忠告
- 存储器区域定义
- 逻辑代码段定义
- 堆栈定义

11.2 标准链接描述文件

每个器件都有标准链接描述文件，默认情况下，它位于以下目录：C:\Program Files\Microchip\MPASM Suite\LKR。

另外还提供用于 MPLAB C18 设置软件堆栈的特殊链接描述文件（见第 11.7 节“堆栈定义”）。默认情况下，这些文件位于以下目录：c:\mcc18\lkr。

链接描述文件如下：

用途	PIC10/12/16	PIC18
通用	<i>DevNum.lkr</i>	<i>DevNum.lkr</i> <i>DevNum_e.lkr</i>
MPLAB ICD 2	<i>DevNumi.lkr</i>	<i>DevNumi.lkr</i> <i>DevNumi_e.lkr</i>

- *DevNum* ——与器件相关的编号，例如，PIC18F452 器件为 18f452.lkr。
- *_e* ——扩展存储器已指定
- *i* ——为 ICD 资源保留的存储器已指定

当在项目包含标准链接描述文件时，建议将此文件复制到项目文件夹。这使您可在必要时针对该项目更改链接描述文件，而不会影响到原始文件。

11.3 链接描述文件命令行信息

MPLAB IDE 项目管理器可以直接设置此信息。如果您从命令行进行链接，可能只需要使用以下各项。

- LIBPATH
- LKRPATH
- FILES
- INCLUDE

11.3.1 LIBPATH

使用 `library/object` 搜索路径来搜索没有路径的库文件和目标文件。以下伪指令在 `library/object` 搜索路径上附加了额外的搜索目录：

```
LIBPATH libpath
```

其中 *libpath* 是用分号分隔的目录列表。

例 11-1: LIBPATH 示例

要将当前目录和目录 `C:\PROJECTS\INCLUDE` 附加到 `library/object` 搜索路径，应将下面的行添加到链接器命令文件上：

```
LIBPATH .;C:\PROJECTS\INCLUDE
```

11.3.2 LKRPATH

使用链接器命令文件搜索路径搜索用链接描述文件 `INCLUDE` 伪指令包含的链接器命令文件。以下伪指令在链接器命令文件搜索路径上附加了额外的搜索目录：

```
LKRPATH lkrpath
```

其中 *lkrpath* 是用分号分隔的目录列表。

例 11-2: LKRPATH 示例

要将当前目录的父目录和目录 `C:\PROJECTS\SCRIPTS` 添加到链接器命令文件搜索路径，应该将以下行加到链接器命令文件：

```
LKRPATH ..;C:\PROJECTS\SCRIPTS
```

11.3.3 FILES

以下伪指令指定链接的目标或库文件：

```
FILES objfile/libfile [objfile/libfile...]
```

其中 *objfile/libfile* 可以是目标或者库文件。

注： 可以在一个单独的 <code>FILES</code> 伪指令中指定多个目标或库文件。

例 11-3: FILES 示例

要指定目标模块 `main.o` 链接到库文件 `math.lib`，应该将以下行添加到链接器命令文件：

```
FILES main.o math.lib
```

11.3.4 INCLUDE

以下伪指令包含了额外的链接器命令文件：

```
INCLUDE cmdfile
```

其中 *cmdfile* 是要包含的链接器命令文件的名称。

例 11-4: INCLUDE 示例

要包含名为 *mylink.lkr* 的链接器命令文件，应该将以下行添加到链接器命令文件：

```
INCLUDE mylink.lkr
```

11.4 链接描述文件忠告

以下是一些有关链接描述文件的忠告：

- 在使用以 MPLINK 链接器包含的链接描述文件之前可能需要先将其修改一下。
- 要将 MPLAB C18 与 MPLINK 链接器一起使用，可能需要重新配置堆栈大小。
- 如果代码包含 *goto* 或 *call* 指令而又没有 *pagesel* 伪指令，可能需要使用链接器描述符分隔存储器页。
- 将 MPLINK 链接器和 MPLAB C18 C 编译器一起使用时，不能将数据存储区域结合在一起。MPLAB C18 要求所有代码段都位于一个存储区之内。有关创建大于一个存储区大小的变量的指导信息，请参见 MPLAB C18 文档。

11.5 存储器区域定义

链接描述文件说明 PICmicro MCU 的存储器架构。这使链接器可把代码放在可用的 ROM 空间，把变量放在可用的 RAM 空间。标记为 PROTECTED 的区域不会被用于一般程序或数据的分配。只有在为代码段指定了一个绝对地址时，或在链接描述文件中使用 SECTION 伪指令将该代码段分配给该区域时，代码或数据才被分配到这些区域。

11.5.1 定义 RAM 存储器区域

DATABANK、SHAREBANK 和 ACCESSBANK 伪指令用于内部 RAM 中的变量。这些伪指令的格式如下。

分组寄存器

```
DATABANK NAME=memName START=addr END=addr [PROTECTED]
```

未分组的寄存器

```
SHAREBANK NAME=memName START=addr END=addr [PROTECTED]
```

快速操作寄存器（仅用于 PIC18 器件）

```
ACCESSBANK NAME=memName START=addr END=addr [PROTECTED]
```

其中：

memName 是用于识别 RAM 中的区域的 ASCII 字符串。

addr 是用来指定地址的十进制（如 30）或十六进制（如 0xFF）数。

可选关键字 PROTECTED 表示一个存储器区域，该存储器区域只在源代码中特别标识时才可使用。链接器不会使用受保护的区域。

例 11-5: RAM 示例

根据 PIC16F877A 寄存器文件映射所显示的 RAM 存储器布局，链接描述文件中的 DATABANK 和 SHAREBANK 项的位置将如图后的示例所示。

PIC16F877A 寄存器文件映射

地址	存储区 0	存储区 1	存储区 2	存储区 3
00h	INDF0	INDF0	INDF0	INDF0
01h	TMR0	OPTION_REG	TMR0	OPTION_REG
02h	PCL	PCL	PCL	PCL
03h	STATUS	STATUS	STATUS	STATUS
04h	FSR	FSR	FSR	FSR
05h	PORTA	TRISA	—	—
:	:	:	:	:
0Fh	TMR1H	—	EEADRH	—
10h	T1CON	—	通用 RAM (分区的)	通用 RAM (分区的)
:	:	:		
1Fh	ADCON0	ADCON1		
20h	通用 RAM (分区的)	通用 RAM (分区的)		
:				
6Fh				
70h	通用 RAM （未分组的）			
:				
7Fh				

PIC16F877A 的 RAM 存储器声明——分区的存储器

```
//Special Function Registers in Banks 0-3
DATABANK  NAME=sfr0      START=0x0      END=0x1F      PROTECTED
DATABANK  NAME=sfr1      START=0x80     END=0x9F      PROTECTED
DATABANK  NAME=sfr2      START=0x100    END=0x10F     PROTECTED
DATABANK  NAME=sfr3      START=0x180    END=0x18F     PROTECTED
//General Purpose RAM in Banks 0-3
DATABANK  NAME=gpr0      START=0x20     END=0x6F
DATABANK  NAME=gpr1      START=0xA0     END=0xEF
DATABANK  NAME=gpr2      START=0x110    END=0x16F
DATABANK  NAME=gpr3      START=0x190    END=0x1EF
```

PIC16F877A 的 RAM 存储器声明——未分区的存储器

```
//General Purpose RAM - available in all banks
SHAREBANK NAME=gprnbnk  START=0x70     END=0x7F
SHAREBANK NAME=gprnbnk  START=0xF0     END=0xFF
SHAREBANK NAME=gprnbnk  START=0x170    END=0x17F
SHAREBANK NAME=gprnbnk  START=0x1F0    END=0x1FF
```

11.5.2 定义 ROM 存储器区域

CODEPAGE 伪指令用于程序代码、初始化的数据值、常数值和外部存储器。其格式如下：

```
CODEPAGE NAME=memName START=addr END=addr [PROTECTED] [FILL=fillvalue]
```

其中：

memName 是用于标识 CODEPAGE 的 ASCII 字符串。

addr 是用来指定地址的十进制或十六进制数。

fillValue 是用来填充存储器区块的所有未使用部分的值。如果这个值是十进制的，它将被假设为 16 位长。如果它是十六进制的（如 0x2346），它可以是可被一个字（16 位）整除的任何长度。

可选关键字 PROTECTED 表示存储器区域，只有特别要求使用该存储器区域的程序代码才可使用。

例 11-6: ROM 示例

下表所示为 PIC16F877A 单片机的程序存储器布局。

存储器	地址
复位向量	0000h - 0003h
中断向量	0004h
用户存储空间	0005h - 07FFh
用户存储空间	0800h - 0FFFh
用户存储空间	1000h - 17FFh
用户存储空间	1800h - 1FFFh
ID 地址单元	2000h - 2003h
保留	2004h - 2005h
器件 ID	2006h
配置存储空间	2007h
保留	2008h - 20FFh
EEPROM 数据	2100h - 21FFh

基于上图，CODEPAGE 声明为：

```
CODEPAGE    NAME=vectors    START=0x0000    END=0x0004    PROTECTED
CODEPAGE    NAME=page0      START=0x0005    END=0x07FF
CODEPAGE    NAME=page1      START=0x0800    END=0x0FFF
CODEPAGE    NAME=page2      START=0x1000    END=0x17FF
CODEPAGE    NAME=page3      START=0x1800    END=0x1FFF
CODEPAGE    NAME=.idlocs    START=0x2000    END=0x2003    PROTECTED
CODEPAGE    NAME=.config    START=0x2007    END=0x2007    PROTECTED
CODEPAGE    NAME=eedata     START=0x2100    END=0x21FF    PROTECTED
```

11.6 逻辑代码段定义

逻辑代码段用于指定应将定义的存储器区域的哪一个区域用于一段源代码。要使用逻辑代码段，用 SECTION 伪指令在链接描述文件中定义代码段，然后使用该语言内置机制在源文件中引用该名称（如 MPLAB C18 的 #pragma section）。

此代码段伪指令通过指定代码段的名称，以及包含此代码段的 ROM 中的程序存储器区块或 RAM 中的数据存储区块来定义代码段：

```
SECTION NAME=secName { ROM=memName | RAM=memName }
```

其中：

secName 是用于标识代码段的 ASCII 字符串。

memName 就是前面定义的 ACCESSBANK、SHAREBANK、DATABANK 或 CODEPAGE。

ROM 属性必须总是指前面用 CODEPAGE 伪指令定义的程序存储器。RAM 属性必须总是指前面用 ACCESSBANK、DATABANK 或 SHAREBANK 伪指令定义的数据存储器。

例 11-7: 逻辑代码段定义

要指定将名为 `filter_coeffs` 的代码段载入名为 `constants` 的程序存储器区域，应该将以下行添加到链接器命令文件：

```
SECTION NAME=filter_coeffs ROM=constants
```

例 11-8: 逻辑代码段的使用

要将 MPASM 源代码装入名为 `filter_coeffs` 的代码段，请在要使用的源代码之前使用以下行：

```
filter_coeffs CODE
```

11.7 堆栈定义

只有 MPLAB C18 要求设置软件堆栈。下面的语句指定堆栈大小以及堆栈将被分配到的可选 DATABANK：

```
STACK SIZE=allocSize [RAM=memName]
```

其中：

allocSize 是以字节表示的堆栈大小，而 *memName* 是前面使用 ACCESSBANK、DATABANK 或 SHAREBANK 语句声明的存储器的名称。

例 11-9: 堆栈示例

要在前面由 `gpr0` 定义的 RAM 区域中将堆栈大小设置为 `0x20`，可以将以下行添加到链接器命令文件中：

```
STACK SIZE=0x20 RAM=gpr0
```

第 12 章 链接器处理过程

12.1 简介

理解 MPLINK 链接器如何处理文件和信息对于编写和组织您的应用程序代码很有用。

本章涉及以下主题：

- 链接器处理过程概述
- 链接器分配算法
- 分配示例
- 已初始化数据
- 保留的段名

12.2 链接器处理过程概述

链接器将多个输入目标模块组合为一个可执行输出模块。输入目标模块可包含可重定位或绝对代码或数据段，这些段将被链接器分配到目标存储器中。在链接器命令文件中对目标存储器架构进行了说明。该链接器命令文件提供了灵活的机制，能够方便地指定目标存储器中的存储块或将代码段映射到指定的存储块中。如果链接器无法找到要将段分配进去的目标存储器的存储块，就将产生一个错误。链接器将名称相似的输入代码段组合到一个输出段中。**第 12.3 节“链接器分配算法”**中对链接器分配算法做了说明。

链接器一旦将来自所有输入模块的段分配到目标存储器以后，就开始对符号重定位进行处理。在每个输入段内定义的符号的地址均取决于符号所在段的首地址。链接器将根据符号最终被分配到的段的位置来调整符号的地址。

链接器完成对在每个输入段中定义的符号的重定位之后，即开始解析外部符号。链接器尝试将所有的外部符号引用与相应的符号定义相匹配。如果某一外部符号引用没有相应的符号定义，那么链接器将尝试在输入库文件中查找相应的符号定义。如果仍然找不到相应的符号定义，则将产生错误。

若外部符号解析成功，链接器将开始划分每个段的原数据。每个段均包含一系列重定位项，这些项将段的原数据中的地址与可重定位符号相关联。可重定位符号的地址被划分为原数据。**第 12.4 节“分配示例”**说明了重定位符号和划分段的过程。

链接器处理完所有可重定位项之后，就会产生可执行的输出模块。

12.3 链接器分配算法

链接器分配目标存储器中的存储区以最大限度地控制目标存储器中被称为“段”的代码或数据单元。链接器可处理以下四种类型的段：

1. 已指派的绝对段
2. 未指派的绝对段
3. 已指派的可重定位段
4. 未指派的可重定位段

绝对段为具有不能被链接器改变的固定的（绝对）地址的段。可重定位段为将根据链接器的分配算法被放置在存储器中的段。

已指派段为在链接器命令文件中已被指派给目标存储区的段。未指派段为未在上述文件中指派的段。

链接器首先分配绝对（已指派和未指派）段，然后分配已指派的可重定位段，最后分配未指派的可重定位段。链接器也处理堆栈的分配。

12.3.1 绝对分配

在链接器命令文件中可将绝对段分配给目标存储块。但由于绝对段的地址是固定的，因此链接器只能验证是否有存放绝对段的已指派目标存储区；目标存储区是否有足够的空间以及绝对段是否与覆盖其他段。如果没有指派目标存储块给绝对段，链接器将尝试为其寻找一个。如若仍然找不到，将产生错误。由于绝对段只能被分配到一个固定的地址，因此对已指派段和未指派段的分配顺序没有特殊要求。

12.3.2 可重定位分配

一旦所有绝对段都分配好了，链接器将开始分配可重定位的已指派段。对于可重定位的已指派段，链接器检查已指派的目标存储区，以验证其中是否有可用空间，若空间不足，将产生错误。对可重定位的已指派段的分配按照它们在链接器命令文件中指定的顺序执行。

当分配了所有的可重定位的已指派段之后，链接器将开始分配可重定位的未指派段。链接器首先分配最大的可重定位的未指派段，然后依次执行分配直至最小的可重定位的未指派段。每次分配时，链接器都会选择可容纳所要存放的段的具有最小可用空间的目标存储块。通过从最大的段开始分配并且选择最小的容纳空间，链接器增加了成功分配所有可重定位的未指派段的可能性。

12.3.3 堆栈分配

堆栈并非一个段，但是它随段一起分配。链接器命令文件不一定会将堆栈分配给特定的目标存储区。如果堆栈已被指派给了一个目标存储块，那么它将在可重定位的已指派段被分配之前被分配。如果堆栈未指派，那么它将在可重定位的已指派段被分配之前，可重定位的未指派段被分配之后被分配。

12.4 分配示例

以下例子说明了链接器分配段的过程。假设以下代码段存在于某一文件中：

```
/* File: ref.c */
char var1;           /* Line 1 */
void setVar1(void)   /* Line 2 */
{
    var1 = 0xFF;      /* Line 3 */
}
```

假设上述代码将编译为以下汇编指令：

注： 本示例刻意省略了由 MPLAB C18 生成的用于处理函数入口和退出的代码。

```
0x0000 MOVLW 0xFF
0x0001 MOVLW ?? ; Need to patch with var1's bank
0x0002 MOVWF ?? ; Need to patch with var1's offset
```

当编译器处理源代码的第一行时，将生成有关标识符 `var1` 的符号表项，包含以下信息：

```
Symbol[index] => name=var1, value=0, section=.data, class=extern
```

当编译器处理源代码的第 3 行时，由于标识符 `var1` 的最终地址在链接前是未知的，则将在代码段中生成两个有关该标识符的重定位项。这两条重定位项包含以下信息：

```
Reloc[index] => address=0x0001 symbol=var1 type=bank
Reloc[index] => address=0x0002 symbol=var1 type=offset
```

一旦链接器将每个段都放入目标存储区以后，最终地址就可知了。一旦所有标识符的最终地址都被指派了，链接器就必须使用可重定位项将所有的引用划分给这些标识符。在上例中，更新后的符号现在位于地址为 **0x125** 的单元中：

```
Symbol[index] => name=var1, value=0x125, section=.data, class=extern
```

如果上述代码段被重定位到以 **0x50** 开始的地址单元中，那么更新后的重定位项现在将位于 **0x51** 开始的地址单元中：

```
Reloc[index] => address=0x0051 symbol=var1 type=bank
Reloc[index] => address=0x0052 symbol=var1 type=offset
```

链接器将逐条划分可重定位项所对应的段。上述示例的最终汇编等效输出将为：

```
0x0050 MOVLW 0xFF
0x0051 MOVLW 0x1 ; Patched with var1's bank
0x0052 MOVWF 0x25 ; Patched with var1's offset
```

12.5 已初始化数据

MPLINK 链接器对具有已初始化数据的输入段进行特殊处理。已初始化数据段包含变量的初始值（初始状态）以及在该段内定义的常数。因为在已初始化数据段内的变量和常量驻留在 RAM 中，所以它们的数据必须存储在非易失性程序存储器（ROM）中。链接器将在程序存储器中为每个已初始化数据段创建一个段。在链接开始时，通过初始化代码（随 MPLAB C18 和 MPASM 汇编器一起提供）将数据传送到合适的 RAM 单元。

由链接器创建的初始化段的名称与已初始化的数据段的名称相同，只是在名称后附加了 `_i`。例如，如果输入目标模块包含名为 `.idata_main.o` 的已初始化数据段，那么链接器将在程序存储器中创建名为 `.idata_main.o_i` 的包含相应数据的段。

除了创建初始化段以外，链接器还将在程序存储器中创建名为 `.cinit` 的段。`.cinit` 段包含一个表，该表包含每个已初始化数据段的项。每项均分为三个组，分别指定初始化段在程序存储器中的首地址，已初始化数据段在数据存储器中的首地址以及已初始化数据段的字节数。引导代码访问该表并把数据从 ROM 复制到 RAM。

12.6 保留的段名

MPASM 汇编器和 MPLAB C18 C 编译器均为特定类型的段保留了名称。请参见与这些工具相关的文档以确保您不将这些保留的段名用作您自己的段的名称。如果存在段名冲突，链接器将无法生成应用程序。

第 13 章 应用程序示例

13.1 简介

您可以从下面四个应用程序示例中学习使用 MPLINK 链接器的基础知识。这些应用程序示例可作为您的应用程序的模板。

- 如何编译应用程序示例
- 应用程序示例 1——修改链接描述文件
 - 如何找到并使用模板文件
 - 如何修改链接描述文件
- 应用程序示例 2——存放代码和设置配置位
 - 如何在不同的存储区域存放程序代码
 - 如何将数据表存入 ROM 存储器
 - 如何用 C 语言设置配置位
- 应用程序示例 3——使用引导加载程序
 - 如何为引导加载程序划分存储空间
 - 如何编译那些将装入外部 RAM 并在外部执行的代码
- 应用程序示例 4——配置外部存储器
 - 如何创建新的链接描述文件存储段
 - 如何用 `#pragma code` 伪指令声明外部存储器
 - 如何用 C 指针访问外部存储器

13.2 如何编译应用程序示例

要编译应用程序示例，需要在个人 PC 上安装 MPASM 汇编器和 MPLINK 链接器，对于某些应用程序示例，还需要安装 MPLAB C18 C 编译器。汇编器和链接器可随 MPLAB DE 自动安装，也可从 Microchip 网站或 MPLAB C18 光盘单独获取。可以从 Microchip 网站上获取 MPLAB C18 C 编译器的免费演示（学生）版。MPLAB C18 C 编译器的完整版必须单独购买。

13.2.1 使用 MPLAB IDE

要使用 MPLAB IDE 编译应用程序：

1. 使用 Project 菜单下的 Project Wizard 创建项目。
 - 选择在应用程序示例中指定的器件。
 - 选择“Microchip MPASM Toolsuite”或“Microchip C18 Toolsuite”作为活动工具套件。确认可执行文件的路径正确。
 - 为项目命名并将其存放在项目文件夹中。
 - 向项目添加样本文件，如 `source1.c`、`source2.asm` 和 `script.lkr`。如果它们还不es 在项目文件夹中，选中每个文件旁的复选框，将它复制到文件夹中。

2. 一旦创建完项目，选择 **Project>Build Options>Project**，打开 Build Options for Project 对话框。
 - 对于 MPLAB C18 应用程序示例，单击 **General** 选项卡，在 “Library Path” 下输入 `c:\mcc18\lib`。
 - 单击 **MPLINK Linker** 选项卡，然后单击 “Generate map file” 旁的复选框选中它。
3. 选择 **Project>Build All**，编译应用程序。
4. 如果应用程序编译失败，检查在下一节中讨论的环境变量在工具安装时是否被正确设置。

13.2.2 使用命令行

要使用命令行编译应用程序：

1. 应按照规定对下面列出的环境变量进行设置。要设置这些变量，打开命令提示符，输入 **SET** 查看和设置变量。在 Windows 操作系统中，可通过 **开始 > 设置 > 控制面板 > 系统**，在 **高级** 选项卡中单击 **环境变量** 按钮。在此查看和编辑变量。
 - **PATH**——确保可以找到以下可执行文件。只有在使用 MPLAB C18 工具时才需要指定 MPLAB C18 的路径。

可执行文件	可执行文件的默认路径
mcc18.exe	c:\mcc18\bin
mpasmwin.exe	c:\mcc18\mpasm 或 c:\program files\microchip\mpasm suite
mplink.exe	c:\mcc18\bin 或 c:\program files\microchip\mpasm suite

- **MCC_INCLUDE**——如果使用 MPLAB C18，它应该指向 `c:\mcc18\h`（默认路径）。
2. 使用以下命令行通过 MPLAB C18 来编译源代码：
`mcc18 -p device source1.c`
其中，*device* 是选定应用程序示例所使用的器件，*source1.c* 是 C 代码源文件示例。对于多个文件，在每个文件间留一空格。
 3. 使用以下命令行通过 MPASM 来汇编源代码：
`mpasmwin -p device source2.asm`
其中，*device* 是选定应用程序示例所使用的器件，*source2.asm* 是汇编代码源文件示例。对于多个文件，在每个文件间留一空格。
 4. 要链接文件并创建应用程序，通过以下命令行使用 MPLINK 链接器：
`mplink script.lkr source1.o source2.o /l c:\mcc18\lib /m app.map`
其中，*script.lkr* 是链接描述文件，*source1.o* 是 C 代码目标文件，*source2.o* 是汇编代码目标文件，*app.map* 是映射文件。只有当使用 MPLAB C18 时才需要库路径 `c:\mcc18\lib`，以通过 MPLAB C18 C 编译器从 *source1.c* 生成 *source1.o*。

13.3 应用程序示例 1——修改链接描述文件

在安装 MPLAB IDE 时，为大多数 MPLAB IDE 支持的器件提供了源代码模板和链接描述文件模板。您可通过这些模板开始编写代码和了解链接描述文件。我们鼓励您根据需要自行修改源代码模板和链接器模板。事实上，一些链接描述文件必须进行编辑才能使用提供的源代码模板顺利进行编译。

在第一个例子中，将分析一个编译错误并修改链接描述文件来纠正此错误，以便能成功完成编译。

13.3.1 定位模板文件

如果 MPLAB IDE 安装在默认位置，源代码即可在下面的路径找到：

C:\Program Files\Microchip\MPASM Suite\Template

该路径下有以下两个子目录：

- Code——包含器件的绝对汇编代码示例
- Object——包含器件的可重定位汇编代码示例

PIC16F877A 的可重定位源代码模板 f877atempo.asm 可以在 Object 目录中找到。该模板在地址 0x0 处定义了用于存放复位向量的绝对代码段，在地址 0x04 处定义了用于存放中断向量的绝对代码段，并且还定义了用于存放 main 主函数的可重定位代码段。

如果 MPLAB IDE 安装在默认位置，链接描述文件可以在下面的路径找到：

C:\Program Files\Microchip\MPASM Suite\LKR

PIC16F877A 的链接描述文件 16f877a.lkr 可在该目录找到。该模板定义了名为 vectors 的程序代码段，该段起始于地址 0x0，结束于 0x04。同时也定义了其他段。

13.3.2 编译应用程序

如果要用这两个文件创建 MPLAB IDE 项目并试图编译该项目（见第 13.2 节“如何编译应用程序示例”），Output 窗口上的结果将如下所示：

注： 缩进的行表示一个换行（延续）的行。

```
Executing: "C:\Program Files\Microchip\MPASM Suite\MPASMWIN.EXE"
/q /p16F877A "f877atempo.asm" /l"f877atempo.lst"
/e"f877atempo.err" /o"f877atempo.o"
Executing: "C:\Program Files\Microchip\MPASM Suite\MPLINK.EXE"
"16f877a.lkr" "G:\docs\MPASM\User Guide
Code\linker_example1\f877atempo.o" /o"example1.cof"
MPLINK 3.90.01, Linker
Copyright (c) 2005 Microchip Technology Inc.
Error - section 'INT_VECTOR' can not fit the absolute section.
Section 'INT_VECTOR' start=0x00000004, length=0x00000018
Errors      : 1
```

BUILD FAILED: Wed Feb 02 17:12:49 2005

这些消息说明源代码已被汇编，但链接器报出一个错误。链接器错误消息说明不能将 INT_VECTOR 段存入链接器试图存放该代码的存储区。错误消息还说明 INT_VECTOR 段起始地址为 0x04，长度为 0x018。

13.3.3 查找错误

在源代码模板中，找到名为 INT_VECTOR 的段代码：

```

INT_VECTOR CODE 0x004 ; interrupt vector location
    
```

该代码语句定义了起始于 0x04 地址单元的绝对代码段， 0x04 地址单元在 PIC16F877A 中用于存放中断向量。

该语句下的源代码将一直属于 INT_VECTOR 段，直到遇到下一个 CODE 语句为止。数一下指令数就会发现 INT_VECTOR 段的长度恰好为 0x018 字节。这样就确定了这是引起链接器错误的代码。

在链接描述文件模板中，找到名为 vectors 的段。

```

CODEPAGE NAME=vectors START=0x0000 END=0x0004 PROTECTED
    
```

在该语句中，定义了起始地址为 0x0 和结束地址为 0x04 的段。用这些地址定义段的目的是为了复位向量地址单元（0x0）和复位地址单元（0x04）设置 PROTECTED 属性，即链接器将不会自动在复位段中存放代码。

正如您所看到的，代码段 INT_VECTOR 不能放入链接器中的 vectors 段并导致出错。

表 13-1: PIC16F877A 程序存储器映射表

程序存储器地址	链接描述文件段	源代码段
0x0000 : 0x0003	vectors——复位、中断向量	RESET_VECTOR——复位向量
0x0004		
0x0005 : 0x001C	page0——ROM 代码空间第 0 页	INT_VECTOR——中断向量
0x07FF		MAIN——主应用程序代码

13.3.4 修复错误

有几种方法可用于修改链接描述文件，从而可成功完成编译。

- 如果想将 INT_VECTOR 代码段放入保护存储区，将链接描述文件中 vectors 的定义修改为：


```
CODEPAGE NAME=vectors START=0x0000 END=0x001F PROTECTED
```

 将下一个段 page0 的起始地址修改为：


```
CODEPAGE NAME=page0 START=0x0020 END=0x07FF
```

2. 如果想将 INT_VECTOR 代码段放入无保护存储区，将链接描述文件中 vectors 的定义修改为：
`CODEPAGE NAME=vectors START=0x0000 END=0x0003 PROTECTED`
将下一个段 page0 的起始地址修改为：
`CODEPAGE NAME=page0 START=0x0004 END=0x07FF`
3. 如果不需要 vectors 段，则从链接描述文件中将其定义删除，并将链接描述文件中 page0 的定义修改为：
`CODEPAGE NAME=page0 START=0x0000 END=0x07FF`

13.4 应用程序示例 2——存放代码和设置配置位

本示例针对扩展单片机模式下的 PIC18F8720。

eeeprom2.asm 文件将中断处理代码放入 0x20000（外部存储器）。汇编代码伪指令 INTHAND CODE 将后续代码放入 INTHAND 段。链接描述文件（eeeprom.lkr）将 INTHAND 段映射到起始于 0x20000 的 CODE 段。

eeeprom1.c 文件在程序存储器中与中断处理程序相同的代码页上存放了 0x1000 大小的元素数据表。在 C 语言中使用 #pragma romdata 伪指令将数据表放入 DATTBL 段。链接描述文件将 DATTBL 段映射到起始于 0x20000 的 CODE 段。

此外，在 C 语言中使用 #pragma config 伪指令设置配置位。

由于没有明确地指派段的伪指令，C 文件中的主函数被默认放在 CODE 段中。

要了解更多信息，可以参见：

- PIC18F8720 Device Data Sheet（DS39609）
- MPLAB C18 C 编译器用户指南（DS51288C_CN）
- External Memory Interfacing Techniques for the PIC18F8XXX（AN869）

表 13-2: PIC18F8720 程序存储器映射表

程序存储器地址	链接描述文件段	源代码段
0x000000 0x000029	vectors——复位、中断向量	STARTUP
0x00002A 0x01FFFF	page——片内存储器	PROG——主应用程序代码
0x020000 0x1FFFFF	eeeprom——外部存储器 ~	INTHAND——中断处理程序 DATTBL——数据表
0x200000 0x200007	idlocs——ID 地址单元	
0x300000 0x30000D	config——配置位	CONFIG——配置设置
0x3FFFFE 0x3FFFFFF	devid——器件 ID	
0xF00000 0xF003FF	eedata——EE 数据	

13.4.1 C 源代码——eeprom1.c

```
/* eeprom1.c */

#include <p18f8720.h>

#define DATA_SIZE 0x1000

/* Data Table Setup */

#pragma romdata DATBL // Put following romdata into section DATBL
unsigned rom data[DATA_SIZE];
#pragma romdata // Set back to default romdata section

/* Configuration Bits Setup
The #pragma config directive specifies the processor-specific
configuration settings (i.e., configuration bits) to be used by
the application. For more on this directive, see the "MPLAB C18
C Compiler User's Guide" (DS51288). */

#pragma config OSC = ON, OSC = LP // Enable OSC switching and LP
#pragma config PWRT = ON // Enable POR
#pragma config BOR = ON, BORV = 42 // Enable BOR at 4.2v
#pragma config WDT = OFF // Disable WDT
#pragma config MODE = EM // Use Extended MCU mode

/* Main application code for default CODE section */

void main (void)
{
    while( 1 )
    {

    } // end while
} // end main
```

13.4.2 汇编源代码——eeprom2.asm

```
; eeprom2.asm

list p=18f8720

#include p18f8720.inc

INTHAND code

; place interrupt handling code in here

end
```

13.4.3 链接描述文件——eeprom.lkr

```
// $Id: 18f8720.lkr,v 1.1 2003/12/16 14:53:08 GrosbaJ Exp $
// File: 18f8720.lkr
// Sample linker script for the PIC18F8720 processor
// Modified for MPLINK Linker Sample Application 1

LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f8720.lib

CODEPAGE    NAME=vectors      START=0x0          END=0x29          PROTECTED
CODEPAGE    NAME=page         START=0x2A         END=0x1FFFF
CODEPAGE    NAME=eeprom       START=0x20000       END=0x1FFFFF      PROTECTED
CODEPAGE    NAME=idlocs       START=0x20000       END=0x20007       PROTECTED
CODEPAGE    NAME=config       START=0x30000       END=0x3000D       PROTECTED
CODEPAGE    NAME=devid        START=0x3FFFE       END=0x3FFFF       PROTECTED
CODEPAGE    NAME=eedata       START=0xF0000       END=0xF003FF      PROTECTED

ACCESSBANK  NAME=accessram    START=0x0          END=0x5F
DATABANK    NAME=gpr0         START=0x60         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
DATABANK    NAME=gpr6         START=0x600        END=0x6FF
DATABANK    NAME=gpr7         START=0x700        END=0x7FF
DATABANK    NAME=gpr8         START=0x800        END=0x8FF
DATABANK    NAME=gpr9         START=0x900        END=0x9FF
DATABANK    NAME=gpr10        START=0xA00        END=0xAFF
DATABANK    NAME=gpr11        START=0xB00        END=0xBFF
DATABANK    NAME=gpr12        START=0xC00        END=0xCFF
DATABANK    NAME=gpr13        START=0xD00        END=0xDFF
DATABANK    NAME=gpr14        START=0xE00        END=0xEFF
ACCESSBANK  NAME=accesssfr    START=0xF60        END=0xFFF          PROTECTED

SECTION     NAME=CONFIG       ROM=config

SECTION     NAME=STARTUP      ROM=vectors        // Reset and interrupt vectors
SECTION     NAME=PROG         ROM=page            // main application code space
SECTION     NAME=INTHAND      ROM=eeprom          // Interrupt handlers
SECTION     NAME=DATTLBL      ROM=eeprom          // Data tables

STACK SIZE=0x100 RAM=gpr14
```

13.4.4 编译应用程序

要编译应用程序，参见第 13.2 节“如何编译应用程序示例”。然后用 MPLAB IDE 继续开发：

1. 虽然代码中的配置位将单片机设置为外部模式，但必须告知 MPLAB IDE 您想用的外部存储器范围。选择 **Configure>External Memory**。在对话框中，单击“Use External Memory”，在“Address Range End”框中输入“0x1FFFF”作为结束地址。单击 **OK**。
2. 选择 **Project>Build All** 再次编译应用程序。

13.5 应用程序示例 3——使用引导加载程序

引导加载程序是一个特殊的程序，当编程进目标 PIC 单片机后，它负责下载可重定位应用程序代码并将它们编程进同一个目标 PIC 单片机中。可重定位应用程序或“用户”代码通常通过串行通信（如 RS232）传输到引导加载程序中。

13.5.1 MPLAB C18 的用法

下面用三个 MPLAB C18 示例来说明如何修改 MPLAB C18 链接描述文件，以及如何在 MPLAB C18 引导加载程序项目的源代码中使用 `#pragma code` 伪指令。

示例 1 显示了如何配置 MPLAB C18 链接描述文件，并建议在 MPLAB C18 引导加载程序中使用代码伪指令的方法。参见第 13.5.3 节“MPLAB C18 引导加载程序链接描述文件”和第 13.5.4 节“MPLAB C18 引导加载程序源代码”。

示例 2 显示了 MPLAB C18 链接描述文件的配置和建议在 MPLAB C18 应用程序中使用的代码伪指令，该应用程序是运行 MPLAB C18 引导加载程序的单片机的目标应用。参见第 13.5.5 节“MPLAB C18 应用程序链接描述文件”和第 13.5.6 节“MPLAB C18 应用程序源代码”。

示例 3 是一个使用 MPLAB C18 应用程序的混合语言示例，该应用程序是带有有限大小的引导块、运行 MPASM 引导加载程序的单片机（如 PIC18F8720）的目标应用。用 C 代码编写的引导加载程序通常比用汇编语言编写的需要更多的程序存储器空间，因此要求单片机引导区的范围较大，如 PIC18F8722。参见第 13.5.7 节“混合语言 MPLAB C18 应用程序链接描述文件”、第 13.5.8 节“混合语言 MPLAB C18 c018i.c 的修改”和第 13.5.9 节“混合语言 MPLAB C18 应用程序源代码”。

针对 MPLAB C18 编写的引导加载程序和应用程序代码必须使用 MPLAB C18 链接描述文件，来命令链接器将编译好的 C 源代码放入合适的程序存储段。通常，引导加载程序代码被编译和链接到目标单片机的程序存储器“boot”段中的目标地址。“应用程序”代码被编译和链接到程序存储器用户段中的目标地址。

13.5.2 MPLAB C18 存储器映射

演示的前两个 MPLAB C18 引导加载程序示例以 PIC18F8722 为例，并分别提供了 2K、4K 或 8K 字节的配置引导区。余下的程序存储空间可用于存放可重定位应用程序代码和数据表。这两个示例假设引导区大小被配置为 2 KB，并要求修改 MPLAB C18 链接描述文件以存放选定的引导区大小。

第 3 个示例是使用 PIC18F8720 的 MPASM 引导加载程序和 MPLAB C18 源代码的混合应用程序。参见第 13.6.7 节“MPASM 汇编器存储器映射”获取相应的存储器映射表。

表 13-3: PIC18F8722 程序存储器映射表

程序存储器地址	链接描述文件段	源代码段
0x000000 0x000029	vectors——复位、中断向量	Vectors、IntH、IntL
0x00002A 0x0007FF	boot——引导加载程序	Boot
0x000800 0x000829	rm_vectors——重映射向量	R_vectors、R_IntH、R_IntL
0x00082A 0x1FFFFFFF	user_code——用户代码	user_code——由引导加载程序更新的应用程序代码

13.5.3 MPLAB C18 引导加载程序链接描述文件

下面显示的部分 MPLAB C18 链接描述文件演示了编译 MPLAB C18 引导加载程序源代码文件时所需作的修改。链接器将用该设置把编译好的源代码链接到起始于 002Ah 的引导程序存储区。使用相应的 `#pragma code` 伪指令在引导加载程序源代码中指定向量的地址单元。

```
// $Id: 18f8722.lkr,v 1.2 2004/09/13 22:07:05 curtiss Exp $
// File: 18f8722.lkr
// Sample linker script for the PIC18F8722 processor
// Modified 2005/02/02 for MPLAB C18 boot loader examples

LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f8722.lib

CODEPAGE    NAME=vectors      START=0x0          END=0x29          PROTECTED
CODEPAGE    NAME=boot         START=0x2A         END=0x7FF
CODEPAGE    NAME=idlocs       START=0x200000     END=0x200007     PROTECTED
CODEPAGE    NAME=config       START=0x300000     END=0x30000D     PROTECTED
CODEPAGE    NAME=devid        START=0x3FFFFFFE   END=0x3FFFFFFF   PROTECTED
CODEPAGE    NAME=eedata       START=0xF00000     END=0xF003FF     PROTECTED
```

13.5.4 MPLAB C18 引导加载程序源代码

MPLAB C18 引导加载程序代码可由一个或多个聚集可重定位 C 源文件组成，这些文件在编译时被编译并链接在一起。在本示例中，源代码文件使用 `#pragma code` 伪指令指导链接器将中断向量放入 0008h 到 0018h 的存储单元。由于“main”函数要被在链接过程中加入的 MPLAB C18 启动代码调用，因此必须在代码中包含该函数。

```
#include <p18cxxx.h>
#define RM_RESET_VECTOR          0x000800 // define relocated vector addresses
#define RM_HIGH_INTERRUPT_VECTOR 0x000808
#define RM_LOW_INTERRUPT_VECTOR  0x000818

/** VECTOR MAPPING *****/
#pragma code _HIGH_INTERRUPT_VECTOR = 0x000008
void _high_ISR (void)
{
    _asm goto RM_HIGH_INTERRUPT_VECTOR _endasm
}

#pragma code _LOW_INTERRUPT_VECTOR = 0x000018
void _low_ISR (void)
{
    _asm goto RM_LOW_INTERRUPT_VECTOR _endasm
}
```

```
/** BOOT LOADER CODE *****/
#pragma code
void main(void)
{
    //Check Bootload Mode Entry Condition
    if(PORTBbits.RB4 == 1)        // If not pressed, User Mode
    {
        _asm goto RM_RESET_VECTOR _endasm
    }
    //Else continue with bootloader code here ...
}
#pragma code user = RM_RESET_VECTOR    // This address defined as 0x800 above
                                        // or can be defined in header file
/** END OF BOOT LOADER *****/
```

13.5.5 MPLAB C18 应用程序链接描述文件

下面显示的部分 MPLAB C18 链接描述文件演示了当编译 MPLAB C18 应用程序源代码文件时所需作的修改。链接器将使用该设置将编译好的源代码链接到地址指定为 082Ah 的 user_code 程序存储区，此存储区在被保护的引导加载程序段之上。

```
// $Id: 18f8722.lkr,v 1.2 2004/09/13 22:07:05 curtiss Exp $
// File: 18f8722.lkr
// Sample linker script for the PIC18F8722 processor
// Modified 2005/02/02 for MPLAB C18 application code examples
```

LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f8722.lib

CODEPAGE	NAME=vectors	START=0x0	END=0x29	PROTECTED
CODEPAGE	NAME=boot	START=0x2A	END=0x7FF	PROTECTED
CODEPAGE	NAME=rm_vectors	START=0x800	END=0x829	PROTECTED
CODEPAGE	NAME=user_code	START=0x82A	END=0x1FFFF	
CODEPAGE	NAME=idlocs	START=0x200000	END=0x200007	PROTECTED
CODEPAGE	NAME=config	START=0x300000	END=0x30000D	PROTECTED
CODEPAGE	NAME=devvid	START=0x3FFFFE	END=0x3FFFFFF	PROTECTED
CODEPAGE	NAME=eedata	START=0xF00000	END=0xF003FF	PROTECTED

13.5.6 MPLAB C18 应用程序源代码

MPLAB C18 应用程序代码可由一个或多个聚集可重定位 C 源文件组成，这些文件在编译时被编译并链接在一起。在下面显示的部分代码中，源代码文件使用 `#pragma code` 伪指令指导链接器将重定位的复位和中断向量放入相应的存储单元。由于 `main` 函数要被在链接过程中加入的 MPLAB C18 启动代码调用，因此代码中必须包含该函数。链接器自动包含 `c018i.c` 文件中提供的 MPLAB C18 初始化代码，并且必须通过下面给出的“内部”汇编指令 `goto` 才能被应用程序代码访问。

```
#include <p18cxxx.h>

/** VECTOR MAPPING *****/
extern void _startup (void);    // See c018i.c in your C18 compiler dir

#pragma code _RESET_INTERRUPT_VECTOR = 0x000800
void _reset (void)
{
    _asm goto _startup _endasm
}

#pragma code _HIGH_INTERRUPT_VECTOR = 0x000808
void _high_ISR (void)
{
    ;
}

#pragma code _LOW_INTERRUPT_VECTOR = 0x000818
void _low_ISR (void)
{
    ;
}

/** APPLICATION CODE*****/
#pragma code
void main(void)
{
    while(1)
    {
        ;                      // Main application code here
    }
}

/** END OF APPLICATION *****/
```

13.5.7 混合语言 MPLAB C18 应用程序链接描述文件

下面显示的部分 MPLAB C18 链接描述文件演示了当编译混合的 MPASM 引导加载程序 /MPLAB C18 应用程序时所需作的修改。链接器将用该设置将汇编好的源代码链接到处于被保护的引导加载程序之上的用户程序存储区。在本链接描述文件示例中，MPLAB C18 启动文件 c018i.o 已被标为注释，从而阻止了链接器将此目标文件链接到项目中。

```
// $Id: 18f8720.lkr,v 1.2 2004/09/13 22:07:05 curtiss Exp $
// File: 18f8720.lkr
// Sample linker script for the PIC18F8720 processor
```

```
LIBPATH .
```

```
//FILES c018i.o <-- Note this line to be ignored by linker
FILES clib.lib
FILES p18f8720.lib
```

CODEPAGE	NAME=vectors	START=0x0	END=0x29	PROTECTED
CODEPAGE	NAME=boot	START=0x2A	END=0x1FF	PROTECTED
CODEPAGE	NAME=rm_vectors	START=0x200	END=0x229	PROTECTED
CODEPAGE	NAME=user_code	START=0x22A	END=0x1FFFFFF	
CODEPAGE	NAME=idlocs	START=0x200000	END=0x200007	PROTECTED
CODEPAGE	NAME=config	START=0x300000	END=0x30000D	PROTECTED
CODEPAGE	NAME=devuid	START=0x3FFFFFFE	END=0x3FFFFFFF	PROTECTED
CODEPAGE	NAME=eedata	START=0xF00000	END=0xF003FF	

13.5.8 混合语言 MPLAB C18 c018i.c 的修改

对于一个典型的 MPLAB C18 应用程序，c018i.c 启动代码通常将程序存储器地址单元 0000h 指定为段的入口，并在 MPLAB C18 链接描述文件中指定时，该启动代码还将被链接器链接到项目中。由于本示例中的 MPLAB C18 应用程序代码已被重定位到程序存储器 0200h 地址单元，因此有必要在 c018i.c 文件中对代码段 _entry_scn 的定义做如下修改并将 c018i.c 源文件添加到项目中，进行重编译和链接。

```
/* $Id: c018i.c,v 1.1 2003/12/09 22:53:19 GrosbaJ Exp $ */
/* Copyright (c)1999 Microchip Technology */
/* MPLAB C18 startup code, including initialized data */
/* Example modification to entry section for relocation to 0200h */
.
.
#pragma code _entry_scn=0x000200
void _entry (void)
{
    _asm goto _startup _endasm
}
.
.
```

13.5.9 混合语言 MPLAB C18 应用程序源代码

MPLAB C18 应用程序代码可由一个或多个可重定位 C 源文件组成，这些文件在编译时被编译并链接在一起。在下面显示的部分代码中，源代码文件使用 #pragma code 伪指令指导链接器将重定位的复位和中断向量放入相应的存储单元。由于 main 函数要被在链接过程中加入的 MPLAB C18 启动代码调用，因此必须在代码中包含该函数。

```

#include <pl8cxxx.h>

/** VECTOR MAPPING *****/

#pragma code _HIGH_INTERRUPT_VECTOR = 0x000208
void _high_ISR (void)
{
    ;                // ISR goes here
}

#pragma code _LOW_INTERRUPT_VECTOR = 0x000218
void _low_ISR (void)
{
    ;                // ISR goes here
}

/** APPLICATION CODE *****/
#pragma code
void main(void)
{
    while(1)
    {
        ;                // Main application code here
    }
}

/** END OF APPLICATION *****/

```

13.5.10 编译 MPLAB C18 应用程序

要编译 MPLAB C18 应用程序示例，请参见第 13.2 节“如何编译应用程序示例”。

13.5.11 MPASM 汇编器的用法

下面用三个 MPASM 示例来说明建议对链接描述文件进行的修改和在引导加载程序 and 应用程序项目中使用相应的源代码伪指令的方法。

示例 1 显示了 MPASM 引导加载程序。参见第 13.5.14 节“MPASM 汇编器引导加载程序源代码”。

示例 2 显示了一个多模块可重定位 MPASM 应用程序。参见第 13.5.15 节“MPASM 汇编器应用程序源代码”。

示例 3 将 MPASM 引导加载程序和多模块可重定位 MPASM 应用程序组合在一个程序存储器镜像中。参见第 13.5.16 节“MPASM 汇编器引导加载程序源代码和应用程序源代码”。

本示例所提供的修改的链接描述文件设计支持以上三种情形。参见第 13.5.13 节“MPASM 汇编器链接描述文件”。

13.5.12 MPASM 汇编器存储器映射

引导加载程序通常驻留在 PIC18F8720 程序存储器的“引导区”中，为存储区的前 512 个字节，从地址单元 0000h 到 01FFh。从 0200h 开始的其余程序存储空间用于存放可重定位的应用程序代码和数据查找表。其他 PIC18F 单片机提供更大的“引导区”，并且需要对链接描述文件进行与本例稍有不同修改。但此处所示的概念可以移植到其他 PIC 单片机上。下表显示了引导加载程序 and 应用程序代码在 PIC18F8720 存储器中的映射。

表 13-4: PIC18F8720 程序存储器映射表

程序存储器地址	链接描述文件段	源代码段
0x000000 0x000029	vector——复位、中断向量	Vectors、IntH 和 IntL
0x00002A 0x0001FF	boot_code——引导加载程序	Boot
0x000200 0x000229	r_vectors——重映射的向量	R_vectors、R_IntH 和 R_IntL
0x00022A 0x1FFFFFFF	user_code——用户代码	user_code——由引导加载程序更新的应用程序代码
0x1F0000 0x1FFFFFFF	const——数据表	

13.5.13 MPASM 汇编器链接描述文件

要保护引导区和向量存储区，链接描述文件使用修改后的 CODEPAGE 伪指令创建这些存储区，并使用 PROTECTED 修饰符阻止链接器指派任何没有明确指派给这些区域的可重定位代码。

下面的样本链接描述文件说明了链接器如何将可重定位的应用程序代码指派到未受保护的用户代码存储区。只有在源文件中用 CODE 伪指令指定将这些代码存放于被保护的存储区时，才可将代码指派给受保护的存储区。此链接描述文件设计用于存放本章所演示的所有三个引导加载程序。

boot.lkr——用于引导加载程序和应用程序代码示例项目的链接描述文件。

```
// $Id: 18f8720.lkr,v 1.8 2004/06/18 19:46:16 ConnerJ Exp $
// File: 18f8720.lkr
// Sample linker script for the PIC18F8720 processor
// Modified 2005/02/02 for MPASM boot loader examples

LIBPATH .

CODEPAGE NAME=vector START=0x0 END=0x29 PROTECTED
CODEPAGE NAME=boot_code START=0x2A END=0x1FF PROTECTED
CODEPAGE NAME=r_vectors START=0x200 END=0x229 PROTECTED
CODEPAGE NAME=user_code START=0x22A END=0x1FFFFFFF
CODEPAGE NAME=const START=0x1F0000 END=0x1FFFFFFF PROTECTED
CODEPAGE NAME=idlocs START=0x200000 END=0x200007 PROTECTED
CODEPAGE NAME=config START=0x300000 END=0x30000D PROTECTED
CODEPAGE NAME=devid START=0x3FFFFFFE END=0x3FFFFFFF PROTECTED
CODEPAGE NAME=eedata START=0xF00000 END=0xF003FF PROTECTED

ACCESSBANK NAME=accessram START=0x0 END=0x5F
DATABANK NAME=gpr0 START=0x60 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
DATABANK NAME=gpr6 START=0x600 END=0x6FF
DATABANK NAME=gpr7 START=0x700 END=0x7FF
DATABANK NAME=gpr8 START=0x800 END=0x8FF
DATABANK NAME=gpr9 START=0x900 END=0x9FF
```

DATABANK	NAME=gpr10	START=0xA00	END=0xAFF	
DATABANK	NAME=gpr11	START=0xB00	END=0xBFF	
DATABANK	NAME=gpr12	START=0xC00	END=0xCFF	
DATABANK	NAME=gpr13	START=0xD00	END=0xDFF	
DATABANK	NAME=gpr14	START=0xE00	END=0xEFF	
DATABANK	NAME=gpr15	START=0xF00	END=0xFF5F	
ACCESSBANK	NAME=accesssfr	START=0xF60	END=0xFFFF	PROTECTED

SECTION NAME=CONFIG ROM=config

13.5.14 MPASM 汇编器引导加载程序源代码

在本例中，引导加载程序是一个独立的源文件，在编译时不会与任何其他源代码链接。引导加载程序源代码中使用的 CODE 伪指令指导链接器将复位和中断向量放入 PIC 单片机中相应的程序存储单元，并将引导加载程序可执行代码的起始地址设在起始于 002Ah 的区域的上面。

程序存储器段名 Vectors、IntH 和 IntL 与 CODE 伪指令一起使用，指导链接器将每一条伪指令后汇编好的代码放在指定的程序存储单元中。此例中，引导加载程序不与任何应用程序代码相链接，所以重定位的复位和中断向量地址假定为 0208h、0218h 和 022Ah，并在代码中明确指定了存放位置。

18Fboot.asm—这是一个示例，说明了在只设计引导加载程序代码并将其编程进目标 PIC 单片机时如何配置引导加载程序的启动部分。

```
; *****
; 18Fboot.asm
; *****
LIST P=18F8722
#include P18cxxx.inc
; *****
Vectors    code    0x0000
VReset:    bra     Boot_Start

IntH        code    0x0008
VIntH:      bra     0x0208      ; Re-map Interrupt vector to app's code space

IntL        code    0x0018
VIntL:      bra     0x0218      ; Re-map Interrupt vector to app's code space

; *****
Boot        code    0x002A      ; Boot loader executable code starts here
Boot_Start:

; Logic to determine if bootloader executes or branch to user's code
; ...
        bra     0x022A      ; Branch to user's application code
; ...
; end of boot loader code section
; *****
END
```

13.5.15 MPASM 汇编器应用程序源代码

在本示例中，应用程序代码由几个可重定位源文件组成，这些文件在编译时被汇编和链接在一起。在 `main.asm` 中定义可重定位复位和中断向量地址单元，并用 `CODE` 伪指令将这些单元分配到特定的程序存储单元中。

main.asm——这是“main”源代码文件启动部分的示例，该源代码文件包含重定位的复位和中断，是进入应用程序的主要入口点。

```
; *****
; main.asm
; *****
    LIST P=18F8722
    #include P18cxxx.inc
; *****
R_vectors    code    0x200
RVReset:
    bra      main

R_IntH       code    0x208    ;Re-mapped HI-priority interrupt vector
RVIntH:
    ;High priority interrupt vector code here
    ;...
    retfie

R_IntL       code    0x218    ;Re-mapped LOW-priority interrupt vector
RVIntL:
    ;Low priority interrupt vector code here
    ;...
    retfie

user_code    code    0x22A
main:
; Entry into application code starts here
; ....
; end of main code section
; *****
END
```

13.5.16 MPASM 汇编器引导加载程序源代码和应用程序源代码

最后一个示例演示了将引导加载程序和应用程序代码组合到一个程序存储器镜像，并同时编程到目标单片机的可能性。由于引导加载程序将被汇编并与应用程序源代码文件相链接，链接器必须解析任何在应用程序代码中定义的对外部标号的引用。要完成此操作，`main.asm` 中使用的 `GLOBAL` 伪指令和引导加载程序源文件中使用的 `EXTERN` 伪指令允许链接器解析在 `main.asm` 中定义的而在 `18Fboot_r.asm` 中引用的重定位复位和中断向量。针对这个示例，使用与在前一个示例中相同的 `boot.lkr` 链接描述文件来将引导加载程序和应用程序文件链接到一起。

18Fboot_r.asm——该引导加载程序示例允许使用不在引导加载程序而是在应用程序源代码中定义的可重定位向量。

```
; *****
; 18Fboot_r.asm
; *****

LIST P=18F8722
#include P18cxxx.inc

; Declare labels used here but defined outside this module
extern RVReset, RVIntH, RVIntL

; *****
Vectors    code    0x0000
VReset:    bra     Boot_Start

IntH       code    0x0008
VIntH:     bra     RVIntH          ; Re-map Interrupt vector

IntL       code    0x0018
VIntL:     bra     RVIntL          ; Re-map Interrupt vector

; *****
Boot       code    0x002A          ; Define explicit Bootloader location
Boot_Start:

; Determine if bootloader should execute or branch to user's code
; ....
        bra     RVReset          ; Branch to user's application code
; Else Bootloader execution starts here
; ....

; *****
END
```

main_r.asm——这是一个主源代码文件示例，源代码文件使用 GLOBAL 伪指令将可重定位复位和中断向量标号提供给引导加载程序。

```
; *****
; main_r.asm
; *****

LIST P=18F8722
#include P18cxxx.inc

; Define labels here but used outside this module
global RVReset, RVIntH, RVIntL
; *****

R_vectors    code    0x200
RVReset:     ;Re-mapped RESET vector
        bra     main

R_IntH       code    0x208          ;Re-mapped HI-priority interrupt vector
RVIntH:      ;High priority interrupt vector code here
;...
        retfie

R_IntL       code    0x218          ;Re-mapped LOW-priority interrupt vector
RVIntL:      ;Low priority interrupt vector code here
;...
        retfie
```

```
user_code    code    0x22A
main:
; Entry into application code starts here
; ....
; end of main code section
; *****
END
```

13.5.17 编译 MPASM 汇编器应用程序

要编译 MPASM 汇编器应用程序示例，请参见第 13.2 节“如何编译应用程序示例”。

该应用程序示例的链接描述文件是对该器件的标准链接描述文件进行修改以后的文件，命名为 boot.lkr。

13.6 应用程序示例 4——配置外部存储器

大多数引脚数较多的 PIC 单片机均能通过外部存储总线（External Memory Bus，EMB）与外部 8 位或 16 位数据闪存或 SRAM 存储器连接。例如，PIC18F8722 具有 128 KB 的内部程序存储器（00000h—1FFFFh）。但是当配置为外部单片机模式时，可通过由 I/O 引脚创建的 EMB 外部寻址从 20000h 到 1FFFFFFh 的外部程序存储空间。

链接描述文件的使用能扩展到外部存储器映射的器件，如可编程 I/O 外设、实时时钟或任何带有多个可通过 8 位或 16 位数据总线访问的配置或控制寄存器的器件。

13.6.1 MPLAB C18 的用法

PIC18F8722 的 MPLAB C18 链接描述文件被修改以指导链接器通过添加如下所示的 CODEPAGE 定义可以使用一个新的存储区。使用 PROTECTED 修饰符阻止链接器向该区域指派任何可重定位的代码。xsram 的名称是随机给出的，它也可以是任何想用的名称。重要的是 START 和 END 地址，它们应与正在使用的外部存储器的物理存储器地址范围相匹配。

```
CODEPAGE    NAME=xsram    START=0x020000    END=0x01FFFFFF    PROTECTED
```

除了创建一个新的 CODEPAGE，还创建一个新的逻辑 SECTION 并把它们分配到在相关的 CODEPAGE 定义中指定的程序存储区。

```
SECTION     NAME=SRAM_BASE    ROM=xsram
```

在 MPLAB C18 应用程序源代码文件中，#pragma romdata 伪指令指导链接器将 SRAM 的起始地址分配到由 SRAM_BASE 逻辑段定义指定的存储区。该物理地址被 xsram 代码页伪指令定义在 20000h 处。由于 SRAM 占用的存储区是程序存储器，而非数据存储器，在声明 char 数组变量 sram[] 时需要有 rom 限定符。此外，该存储区超出了 16 位地址范围（64 KB），所以要求使用 far 限定符以使 C 指针能正确访问该区域。

```
#pragma romdata SRAM_BASE ;Assigns this romdata space at 0x020000
rom far char sram[];      ;Declare an array at starting address
```


13.6.2 MPLAB C18 存储器映射

下面的表格显示了 PIC18F8722 使用 1 MB 外部 SRAM 器件时的存储器映射。注意外部存储区的前 128 KB 与内部存储空间 的 128 KB 是重叠的，所以不能用外部存储总线访问它。若没有其他外部存储器地址译码方式，SRAM 的前 128 KB 空间是不可访问的，这样 SRAM 可寻址的起始地址为 20000h。

表 13-5: 程序存储器映射——PIC18F8722 和 1 MB SRAM

程序存储器地址	SRAM 地址	链接描述文件段	源代码段
0x000000 0x000029	0x000000 0x01FFFF	vectors——复位、中断向量	
0x00002A 0x01FFFF		page——片内存储器	
0x020000 0x0FFFFF	0x020000 0x0FFFFF	xsram——外部存储器	SRAM_BASE——rom 数据空间
0x100000 0x1FFFFF			

13.6.3 MPLAB C18 链接描述文件

下面显示的修改后的 PIC18F8722 MPLAB C18 链接描述文件演示了建议的对外部存储器应用程序的修改。

```
// $Id: 18f8722.lkr,v 1.2 2004/09/13 22:07:05 curtiss Exp $
// File: 18f8722.lkr
// Sample linker script for the PIC18F8722 processor
// This modified version saved as C18_xmem.lkr

LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f8722.lib

CODEPAGE NAME=vectors START=0x0 END=0x29 PROTECTED
CODEPAGE NAME=page START=0x2A END=0x1FFFF
CODEPAGE NAME=xsram START=0x020000 END=0x01FFFFF PROTECTED
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=0xF003FF PROTECTED

ACCESSBANK NAME=accessram START=0x0 END=0x5F
DATABANK NAME=gpr0 START=0x60 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
DATABANK NAME=gpr6 START=0x600 END=0x6FF
DATABANK NAME=gpr7 START=0x700 END=0x7FF
DATABANK NAME=gpr8 START=0x800 END=0x8FF
DATABANK NAME=gpr9 START=0x900 END=0x9FF
DATABANK NAME=gpr10 START=0xA00 END=0xAFF
DATABANK NAME=gpr11 START=0xB00 END=0xBFF
```

```
DATABANK    NAME=gpr12      START=0xC00      END=0xCFF
DATABANK    NAME=gpr13      START=0xD00      END=0xDFF
DATABANK    NAME=gpr14      START=0xE00      END=0xEFF
DATABANK    NAME=gpr15      START=0xF00      END=0xF5F
ACCESSBANK  NAME=accesssfr  START=0xF60      END=0xFFF      PROTECTED
```

```
SECTION     NAME=CONFIG     ROM=config
SECTION     NAME=SRAM_BASE   ROM=xsram
```

```
STACK SIZE=0x100 RAM=gpr14
```

13.6.4 MPLAB C18 源代码

这是一个简单的代码示例，显示了使用 `#pragma romdata` 声明外部存储器和使用 C 指针访问该存储器的方式。

```
#include <p18F8722.h>

#pragma romdata SRAM_BASE ; Assigns this romdata space at 0x02000
rom far char sram[];      ; Declare an array at starting address

#pragma code
void main(void)
{
    rom far char* dataPtr; ; Create a "far" pointer

    dataPtr = sram;        ; Assign this pointer to the memory array
    *dataPtr++ = 0xCC;     ; Write low byte of 16-bit word to SRAM
    *dataPtr = 0x55;       ; Write high byte of 16-bit word to SRAM
}
```

13.6.5 编译 MPLAB C18 应用程序

要编译 MPLAB C18 应用程序示例，请参见第 13.2 节“如何编译应用程序示例”。在这个项目中必须使用大容量存储器模型。

- 对于 MPLAB IDE，在第 2 步末单击 **MPLAB C18** 选项卡，在 Category 下拉列表中选择“Memory Model”。在“Code Mode”下单击“Large code mode (>64K)”。
- 对于命令行，编译时使用 `-ml` 选项。

该应用程序示例的链接描述文件是对该器件的标准链接描述文件修改后的文件。

13.6.6 MPASM 汇编器用法

在 MPASM 应用程序的源文件中，使用简单的 `#define` 或 `equ` 伪指令能很方便地定义 SRAM 起始地址，在表读或表写前用该地址设置表指针。

```
#define SRAM_BASE_ADDRS 0x20000 ;Base addrs for external
                                ;memory device
#define SRAM_END_ADDRS  0x1FFFFF ;End addrs (not required)
```

通过表读和表写访问外部程序存储器要求用相应的地址设置表指针寄存器，如下例所示。

```

movlw    upper (SRAM_BASE_ADDRS)
movwf    TBLPTRU
movlw    high (SRAM_BASE_ADDRS)
movwf    TBLPTRH
movlw    low (SRAM_BASE_ADDRS)
movwf    TBLPTRL

```

13.6.7 MPASM 汇编器存储器映射

下面的表格显示了当使用 1 MB 外部 SRAM 器件时的 PIC18F8722 的存储器映射。注意外部存储区的前 128 KB 与内部存储空间的 128 KB 是重叠的，所以不能用外部存储总线访问它。若没有其他外部存储器地址译码方式，SRAM 的前 128 KB 空间是不可访问的，这样 SRAM 中可寻址的起始地址为 20000h。

表 13-6: 程序存储器映射——PIC18F8722 和 1MB SRAM

程序存储器地址	SRAM 地址	链接描述文件段	源代码段
0x000000 0x000029	0x000000 0x01FFFF	vectors——复位、中断向量	vectors
0x00002A 0x01FFFF		page——片内存储器	prog——主程序
0x020000 0x0FFFFF	0x020000 0x0FFFFF	xsram——外部存储器	SRAM_BASE_ADDRS SRAM_END_ADDRS
0x100000 0x1FFFFF			

13.6.8 MPASM 汇编器链接描述文件

下面显示的修改后的 PIC18F8722 MPASM 链接描述文件演示了建议的对外部存储器应用程序进行的修改。

```

// $Id: 18f8722.lkr,v 1.1 2004/09/09 21:22:33 curtiss Exp $
// File: 18f8722.lkr
// Sample linker script for the PIC18F8722 processor

```

LIBPATH .

```

CODEPAGE    NAME=vectors      START=0x0          END=0x29          PROTECTED
CODEPAGE    NAME=page         START=0x2A         END=0x1FFFF
CODEPAGE    NAME=xsram        START=0x020000     END=0x1FFFFF     PROTECTED
CODEPAGE    NAME=idlocs       START=0x200000     END=0x200007     PROTECTED
CODEPAGE    NAME=config       START=0x300000     END=0x30000D     PROTECTED
CODEPAGE    NAME=devid        START=0x3FFFFE     END=0x3FFFFFF     PROTECTED
CODEPAGE    NAME=eedata       START=0xF00000     END=0xF003FF     PROTECTED

```

```

ACCESSBANK  NAME=accessram    START=0x0          END=0x5F
DATABANK    NAME=gpr0         START=0x60         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
DATABANK    NAME=gpr6         START=0x600        END=0x6FF
DATABANK    NAME=gpr7         START=0x700        END=0x7FF
DATABANK    NAME=gpr8         START=0x800        END=0x8FF
DATABANK    NAME=gpr9         START=0x900        END=0x9FF

```

DATABANK	NAME=gpr10	START=0xA00	END=0xAFF	
DATABANK	NAME=gpr11	START=0xB00	END=0xBFF	
DATABANK	NAME=gpr12	START=0xC00	END=0xCFF	
DATABANK	NAME=gpr13	START=0xD00	END=0xDFF	
DATABANK	NAME=gpr14	START=0xE00	END=0xEFF	
DATABANK	NAME=gpr15	START=0xF00	END=0xF5F	
ACCESSBANK	NAME=accesssfr	START=0xF60	END=0xFFF	PROTECTED
SECTION	NAME=CONFIG	ROM=config		
SECTION	NAME=VECTORS	ROM=vectors		
SECTION	NAME=PROG	ROM=page		
SECTION	NAME=SRAM	ROM=xsram		

13.6.9 MPASM 汇编器源代码

这是一个简单的代码示例，显示了将外部存储器 **SRAM** 的地址定义在 **20000h** 的过程以及如何使用表指针寄存器和写表指令将一个 **16** 位值写入两个连续的存储单元。

```
#include <p18F8722.inc>

#define SRAM_BASE_ADDRS 0x20000    ; Base addrs for external memory device
#define SRAM_END_ADDRS  0x1FFFFF   ; End addrs  (not required)

vectors code
    bra    main

prog    code
main:
; Example - how to write "0x55CC" to first word location in external SRAM memory

    movwf    upper (SRAM_BASE_ADDRS)
    movwf    TBLPTRU
    movlw    high (SRAM_BASE_ADDRS)
    movwf    TBLPTRH
    movlw    low (SRAM_BASE_ADDRS)
    movwf    TBLPTRL

    movlw    0xCC
    movwf    TBLLAT
    tblwt*+          ; Writes "0xCC" to byte location 0x020000;
                     ; Increments table pointer to next location

    movlw    0x55
    movwf    TBLLAT
    tblwt*          ; Write "0x55" to byte location 0x020001;
```

13.6.10 编译 MPASM 汇编器应用程序

要编译 MPASM 汇编器应用程序示例，请参见第 **13.2** 节 “如何编译应用程序示例”。该应用程序示例的链接描述文件是对该器件的标准链接描述文件修改后的文件。

第 14 章 错误、警告和常见问题

14.1 简介

错误消息和警告消息由 MPLINK 链接器产生。这些消息总是出现在列表文件中，位于发生错误的行上方。

本章还列出了链接器工具的常见问题和限制。

本章涉及以下主题：

- 链接器解析错误
- 链接器错误
- 链接器警告
- 库文件错误
- COFF 文件错误
- COFF 到 COD 转换错误
- COFF 到 COD 转换警告
- 常见问题

14.2 链接器解析错误

下面按字母顺序列出了 MPLINK 链接器解析错误：

Could not open 'cmdfile'.

无法打开链接器命令文件。检查文件是否存在，是否在当前的搜索路径中并且是否可读。

Illegal <filename> for FILES in 'cmdfile:line'.

目标文件或库文件必须分别以 .o 或 .lib 结束。

Illegal <filename> for INCLUDE in 'cmdfile:line'.

链接器命令文件名必须以 .lkr 结束。

Illegal <libpath> for LIBPATH in 'cmdfile:line'.

libpath 必须以分号分隔各个目录列表。将带有嵌入空间的目录名称以双引号括起。

Illegal <lkrpath> for LKRPATH in 'cmdfile:line'.

lkrpath 必须以分号分隔各个目录列表。将带有嵌入空间的目录名称以双引号括起。

Invalid attributes for memory in 'cmdfile:line'.

CODEPAGE、DATABANK 或 SHAREBANK 未指定 NAME、START 或 END 属性，或指定了一个无效的属性。

Invalid attributes for SECTION in 'cmdfile:line'.

SECTION 伪指令必须具有 NAME、RAM 或 ROM 属性。

Invalid attributes for STACK in 'cmdfile:line'.

STACK 伪指令未指定 SIZE 属性，或指定了一个无效的属性。

-k switch requires <pathlist>.

必须指定以分号分隔的路径。将带有嵌入空间的目录名称以双引号括起。例如：

```
-k ..;c:\mylkr;"c:\program files\microchip\mpasm suite\lkr"
```

-l switch requires <pathlist>.

必须指定以分号分隔的路径。将带有嵌入空间的目录名称以双引号括起。例如：

```
-l ..;c:\mylib;"c:\program files\microchip\mpasm suite"
```

-m switch requires <filename>.

必须指定映射文件的名称。例如：-m main.map。

Multiple inclusion of library file 'filename'.

在命令行上或使用链接器命令文件中的 FILES 伪指令多次包含了同一库文件。取消多次引用。

Multiple inclusion of linker command file 'cmdfile'.

链接器命令文件只能被包含一次。取消对引用的链接器命令文件的多次 INCLUDE 伪指令。

Multiple inclusion of object file 'filename'.

在命令行上或使用链接器命令文件中的 FILES 伪指令多次包含了同一目标文件。取消多次引用。

-n switch requires <length>.

必须指定列表文件每页的源代码行数。长度为 0 将禁止列表文件分页。

-o switch requires <filename>.

必须指定 COFF 输出文件的名称。例如：-o main.out

Unknown switch: 'cmdline token'.

使用了无法识别的命令行选项。参见使用用法文档，获取所支持的选项的列表。

Unrecognized input in 'cmdfile:line'.

链接器命令文件中所有的语句都必须以伪指令关键词或注释分隔符 // 开始。

14.3 链接器错误

下面按字母顺序列出了 MPLINK 链接器错误：

Absolute code section 'secName' must start at a word-aligned address.

程序代码段只能以字对齐的地址进行分配。如果指定的绝对代码段的地址不是以字对齐的，MPLINK 将给出一条错误信息。

Configuration settings have been specified for address 0x300001 in more than one object module. Found in 'foo.o' previously found in 'bar.o'.

在两个不同的 .c 文件（如 foo.c 和 bar.c）中使用了 MPLAB C18 的伪指令 #pragma config，并用同一个配置字节指定其设置时，将产生此错误。在一个单独的 .c 文件中设置给定字节的配置位。

Conflicting types for symbol 'symName' .

符号 *symName* 在不同的位置被定义为不同的类型。

Could not find definition of symbol 'symName' in file 'filename'.

使用的符号 *symName* 未在文件 *filename* 中定义。

Could not find file 'filename'.

指定的输入目标或库文件不存在或在链接器路径中找不到。

Could not open map file 'filename' for writing.

验证 *filename* 是否存在，并且其是否为只读文件。

Could not resolve section reference 'symName' in file 'filename'.

符号 *symName* 是一个外部引用。任何输入模块均未定义该符号。如果库模块中定义了符号，确保命令行或使用 FILES 伪指令的链接器命令文件包含该库模块。

Could not resolve symbol 'symName' in file 'filename'.

符号 *symName* 是一个外部引用。任何输入模块均未定义该符号。如果库模块中定义了符号，确保命令行或使用 FILES 伪指令的链接器命令文件包含该库模块。

Duplicate definition of memory 'memName'.

所有 CODEPAGE 和 DATABANK 伪指令均必须具备惟一的 NAME 属性。

Duplicate definitions of SECTION 'secName'.

每一个 SECTION 伪指令必须具备惟一的 NAME 属性。删除重复定义。

File 'filename', section 'secName', performs a call to \n symbol 'symName' which is not in the lower half of a page.

对于 12 位的器件，在 CALL 指令或任何修改 PCL 的指令中，程序计数器（Program Counter，PC）的第 8 位被清零。因此，所有子程序调用或计算跳转均被限制在任何程序存储器页（长度为 512 字）的头 256 个单元中。

Inconsistent length definitions of SHAREBANK 'memName'.

所有具有相同 NAME 属性的 SHAREBANK 定义长度必须相同。

Internal Coff output file is corrupt.

链接器无法写入 COFF 文件。

Memory 'memName' overlaps memory 'memName'.

CODEPAGE 块必须指定惟一的不重叠的存储区范围。类似地，DATABANK 和 SHAREBANK 块也不能重叠。

Mixing extended and non-extended mode modules not allowed.

链接器不能链接扩展模式模块和非扩展模式模块的混合体。扩展和非扩展存储器模式适用于 PIC18 器件。

当使用 MPASM 创建目标文件模块时，可通过在命令行中使用 /y 选项使能 / 禁止扩展存储器模式。在 MPLAB IDE 中，选择 Project>Build Options，在 **MPASM Assembly** 选项卡中选中 / 取消选择 “Extended Mode” 选项。

当使用 MPASM C18 创建目标文件模块时，可通过在命令行中使用 --extended 选项使能 / 禁止扩展存储器模式。在 MPLAB IDE 中，选择 Project>Build Options，在 **MPLAB C18** 选项卡中选中 / 取消选择 “Extended Mode” 选项。

当使用链接描述文件时，那些带有 _e 后缀的文件适用于扩展模式。

MPASM's __CONFIG directive (found in 'bar.o') cannot be used with either MPLAB C18's #pragma config directive or MPASM's CONFIG directive (found in 'foo.o').

当 MPASM 汇编器的 __CONFIG 伪指令在一个 .asm 文件（如 bar.asm）中被指定而 MPLAB C18 的 #pragma config 伪指令在一个 .c 文件（如 foo.c）中被指定时，链接器就会产生此错误消息。使用 MPASM 汇编器的 __CONFIG 伪指令或 MPLAB C18 的 #pragma config 伪指令之一设置配置位。

Multiple map files declared: 'filename1', 'filename2'.

多次指定了 -m <mapfile> 选项。

Multiple output files declared: 'filename1', 'filename2'.

多次指定了 -o <outfile> 选项。

Multiple STACK definitions.

在链接器命令文件或包含的链接器命令文件中多次出现 STACK 伪指令。删除重复的 STACK 伪指令。

No input object files specified.

没有给链接器指定任何输入目标或库文件。输入要链接的文件。

Overlapping definitions of SHAREBANK 'memName'.

SHAREBANK 伪指令指定的地址范围跟先前定义的范围重叠。不允许有重叠。

{PCL | TOSH | TOSU | TOSL} cannot be used as the destination of a MOVFF or MOVSF instruction.

当 MOVFF 指令的目标寄存器是 PCL、TOSH、TOSU 或 TOSL 时，其结果无法预料。MPLINK 不允许用这些地址中的任何一个替代 MOVFF 指令的目标寄存器。

Processor types do not agree across all input files.

每一个目标模块和库文件指定一种处理器类型或处理器系列。所有的输入模块必须与处理器类型或系列匹配。

Section {absolute|access|overlay|share} types for 'secName' do not match across input files.

名为 *secName* 的段出现在多个输入文件中。然而，在一些文件中，段被标为绝对、快速操作、重叠或共享，而在其他一些文件则不是这样。在源文件中更改段类型，然后重新编译目标模块。

Section 'secName' can not fit the absolute section. Section 'secName' start=0xHHHH, length=0xHHHH.

未在链接器命令文件中指派给存储器的段不能被分配。使用 `-m <mapfile>` 选项生成错误映射文件。错误映射文件会显示发生错误前分配的段。必须通过添加 CODEPAGE、SHAREBANK 或 DATABANK 伪指令或取消 PROTECTED 属性，或者减少输入段的数目以增加可用的存储空间。

Section 'romName' can not have a 'RAM' memory attribute specified in the linker command file.

在链接器命令文件中定义段时只能使用 ROM 属性。

Section 'secName' can not fit the section. Section 'secName' length='0xHHHH'.

未在链接器命令文件中指派给存储器的段不能被分配。使用 `-m <mapfile>` 选项生成错误映射文件。错误映射文件会显示发生错误前分配的段。必须通过添加 CODEPAGE、SHAREBANK 或 DATABANK 伪指令或取消 PROTECTED 属性，或者减少输入段的数目以增加可用的存储空间。

Section 'secName' contains code and can not have a 'RAM' memory attribute specified in the linker command file.

在链接器命令文件中定义段时只能使用 ROM 属性。

Section 'secName' contains initialized data and can not have a 'ROM' memory attribute specified in the linker command file.

在链接器命令文件中定义段时只能使用 RAM 属性。

Section 'secName' contains initialized rom data and can not have a 'RAM' memory attribute specified in the linker command file.

在链接器命令文件中定义段时只能使用 ROM 属性。

Section 'secName' contains uninitialized data and can not have a 'ROM' memory attribute specified in the linker command file.

在链接器命令文件中定义段时只能使用 RAM 属性。

Section 'secName' has a memory 'memName' which can not fit the absolute section. Section 'secName' start=0xHHHH, length=0xHHHH.

链接器命令文件中指派给段的存储区不是没有空间用于存储段，就是段与另一个段重叠。使用 -m <mapfile> 选项生成错误映射文件。错误映射文件会显示发生错误前分配的段。

Section 'secName' has a memory 'memName' which can not fit the section. Section 'secName' length='0xHHHH'.

链接器命令文件中指派给段的存储区不是没有空间用于存储段，就是段与另一个段重叠。使用 -m <mapfile> 选项生成错误映射文件。错误映射文件会显示发生错误前分配的段。

Section 'secName' has a memory 'memName' which is not defined in the linker command file.

在链接器命令文件中给未定义的存储区添加 CODEPAGE、DATABANK 或 SHAREBANK 伪指令。

Section 'secName' type is non-overlay and absolute but occurs in more than one input file.

名为 *secName* 的绝对段只能出现在单个输入文件中。名称相同的可重定位段可出现在多个输入文件中。要么删除源文件中的多个绝对段，要么使用可重定位段。

Starting addresses for absolute overlay section 'secName' do not match accross all input files.

名为 *secName* 的段出现在多个输入文件中。然而，各个文件的绝对重叠起始地址都不相同。在源文件中更改段地址，然后重新编译目标模块。

Symbol 'symName' has multiple definitions.

一个符号只能在一个输入模块中定义。

Symbol 'symName' is not word-aligned. It cannot be used as the target of a {branch | call or goto} instruction.

Branch、call 或 goto 指令的目标是地址为奇地址，但指令编码无法引用不是字对齐的地址。

symbol 'symName' out of range of relative branch instruction.

相对跳转指令将 *symName* 作为目标，但到 *symName* 的偏移量的二的补码编码与用作跳转指令目标的指令位的限制位数不配。

The `_CONFIG_DECL` macro can only be specified in one module. Found in 'foo.o' previously found in 'bar.o' .

当 MPLAB C18 的宏 `_CONFIG_DECL` 在两个不同的 `.c` 文件（如 `foo.c` 和 `bar.c`）中指定时链接器会产生此错误。在一个 `.c` 文件中使用宏 `_CONFIG_DECL` 设置配置位。

The `_CONFIG_DECL` macro (found in 'foo.o') cannot be used with MPASM's `__CONFIG` directive (found in 'bar.o')

当 MPLAB C18 的宏 `_CONFIG_DECL` 用于 `.c` 文件（如 `foo.c`），而 MPASM 汇编器的伪指令 `__CONFIG` 用于 `.asm` 文件（如 `bar.asm`）时，就会产生此错误。使用 MPLAB C18 的宏 `_CONFIG_DECL` 或 MPASM 汇编器的伪指令 `__CONFIG` 中的一个设置配置位。

The `_CONFIG_DECL` macro (found in 'foo.o') cannot be used with either MPLAB C18's `#pragma config` directive or MPASM's `CONFIG` directive (found in 'bar.o')

当 MPLAB C18 的宏 `_CONFIG_DECL` 用于 `.c` 文件（如 `foo.c`），而 MPLAB C18 的伪指令 `#pragma config` 用于另一个 `.c` 文件（如 `bar.c`）或是 MPASM 汇编器的伪指令 `__CONFIG` 用于 `.asm` 文件（如 `bar.asm`）时，会产生此错误。用伪指令 `_CONFIG_DECL`、`#pragma config` 或 `__CONFIG` 中的一个设置配置位。

TRIS argument is out of range '0xHHHH' not between '0xHHHH' and '0xHHHH'.

查看器件的数据手册，确定所使用的 TRIS 寄存器可接受的十六进制值。

Undefined CODEPAGE 'memName' for SECTION 'secName'.

带 ROM 属性的 SECTION 伪指令引用未定义的存储区。向链接器命令文件添加针对于该未定义的存储区的 CODEPAGE 伪指令。

Undefined DATABANK/SHAREBANK 'memName' for SECTION 'secName'.

带 RAM 属性的 SECTION 伪指令引用未定义的存储区。向链接器命令文件添加针对于该未定义的存储区的 DATABANK 或 SHAREBANK 伪指令。

Undefined DATABANK/SHAREBANK 'memName' for STACK.

没有指定任何输入目标文件。必须在命令行上或在链接器命令文件中使用 FILES 伪指令至少指定一个目标模块。

Unknown section type for 'secName'.

“secName”的段类型需要被定义。

Unknown section type for 'secName' in file 'filename'.

输入目标或库模块文件类型不对或文件已损坏。

Unsupported processor type in file 'filename'.

链接器目前不支持指定的处理器。参见自述文件获取所支持的器件列表。

Unsupported relocation type.

链接器目前不支持指定的重定位类型。

14.4 链接器警告

下面按字母顺序列出了 MPLINK 链接器警告：

Fill pattern for memory '*memName*' doesn't divide evenly into unused section locations. Last value was truncated.

如果填充模式是为 ROM 段指定的，但该段的空余空间却未按填充模式的大小平均划分，就会产生此模式不完整的警告。

'/a' command line option ignored with '/x'

/x 阻止生成 .hex 文件。所以，用 /a 指定 hex 输出文件格式不起作用。

'/n' command line option ignored with '/w'

/w 阻止生成 .cod 和 .lst 文件。所以，用 /n 指定每个列表页中的行数不起作用。

14.5 库文件错误

下面按字母顺序列出了 MPLINK 链接器库文件处理错误：

Could not build member '*memberName*' in library file '*filename*'.

文件不是一个有效的库文件或文件已损坏。

Could not open library file '*filename*' for reading.

验证 *filename* 是否存在，并且是否可读。

Could not open library file '*filename*' for writing.

验证 *filename* 是否存在，并且是否为只读。

Could not write archive magic string in library file '*filename*'.

文件可能已损坏。

Could not write member header for '*memberName*' in library file '*filename*'.

文件可能已损坏。

File '*filename*' is not a valid library file.

库文件必须以 .lib 结束。

Library file '*filename*' has a missing member object file.

文件不是一个有效的目标文件或文件可能已损坏。

'*memberName*' is not a member of library '*filename*'.

除非为库成员，否则 *memberName* 不能从库中提取或删除。

Symbol 'symName' has multiple external definitions.

一个符号在一个库文件中只能被定义一次。

14.6 COFF 文件错误

下面列出的所有 COFF 错误指出了文件内容的内部错误。如果产生以下任一错误，请联系 Microchip 技术支持。

- Coff file 'filename' could not read file header.
- Coff file 'filename' could not read line numbers.
- Coff file 'filename' could not read optional file header.
- Coff file 'filename' could not read raw data.
- Coff file 'filename' could not read relocation info.
- Coff file 'filename' could not read section header.
- Coff file 'filename' could not read string table.
- Coff file 'filename' could not read string table length.
- Coff file 'filename' could not read symbol table.
- Coff file 'filename' could not write file header.
- Coff file 'filename' could not write lineinfo.
- Coff file 'filename' could not write optional file header.
- Coff file 'filename' could not write raw data.
- Coff file 'filename' could not write reloc.
- Coff file 'filename' could not write section header.
- Coff file 'filename' could not write string.
- Coff file 'filename' could not write string table length.
- Coff file 'filename' could not write symbol.
- Coff file 'filename', cScnHdr.size() != cScnNum.size().
- Coff file 'filename' does not appear to be a valid COFF file.
- Coff file 'filename' has relocation entries but an empty symbol table.
- Coff file 'filename' missing optional file header.
- Coff file 'filename' section['xx'] has an invalid s_offset.
- Coff file 'filename', section 'secName' line['xx'] has an invalid l_fcndx.
- Coff file 'filename', section 'secName' line['xx'] has an invalid l_srcndx.
- Coff file 'filename', section 'secName' reloc['xx'] has an invalid r_symndx.
- Coff file 'filename' symbol['xx'] has an invalid n_offset.
- Coff file 'filename' symbol['xx'] has an invalid n_scnum.
- Coff file 'filename', symbol['xx'] has an invalid index.
- Could not find section name 'secName' in string table.
- Could not find symbol name 'symName' in string table.
- Could not open Coff file 'filename' for reading.
- Could not open Coff file 'filename' for writing.
- Could not read archive magic string in library file 'filename'.
- Unable to find aux_file name in string table.
- Unable to find section name in string table.
- Unable to find symbol name in string table.

14.7 COFF 到 COD 转换错误

Source file 'filename' name exceeds file format maximum of 63 characters.

COD 文件名包括路径最多只能有 63 个字符。

Coff file 'filename' must contain at least one 'code' or 'romdata' section.

为了把 COFF 文件转换成 COD 文件，COFF 文件必须有代码段或 rom 数据段。

Could not open list file 'filename' for writing.

验证 *filename* 是否存在，并且是否为只读。

14.8 COFF 到 COD 转换警告

Could not open source file 'filename'. This file will not be present in the list file.

无法打开引用的源文件。如果在具有不同目录结构的机器上编译输入目标 / 库模块，则可能发生这种情况。如果需要从源代码调试文件，在当前的机器上重新编译目标或库文件。

14.9 COD 文件错误

下面列出的 COD 文件错误指出了文件内容的内部错误。如果产生以下任一错误，请联系 Microchip 技术支持。

- Cod file 'filename' does not have a proper debug message table.
- Cod file 'filename' does not have a proper Index.
- Cod file 'filename' does not have a proper line info table.
- Cod file 'filename' does not have a proper local vars table.
- Cod file 'filename' does not have a proper long symbol table.
- Cod file 'filename' does not have a proper memory map table.
- Cod file 'filename' does not have a proper name table.
- Cod file 'filename' does not have a proper symbol table.
- Cod file 'filename' does not have a properly formed first directory.
- Cod file 'filename' does not have a properly formed linked directory.
- Could not open Cod file 'filename' for reading.
- Could not open Cod file 'filename' for writing.
- Could not write 'blockname' block in Cod file 'filename'.
- Could not write directory in Cod file 'filename'.

14.10 HEX 文件错误

Selected hex format does not support byte addresses above 64 kB; use INHX32 format!

代码寻址超过 64 KB 的程序存储器，但选定的 hex 格式不支持上述操作。改用 INHX32 格式。

Could not open hex file 'filename' for writing.

由于其他错误而使 hex 文件未创建，或已有的 hex 文件为写保护。

14.11 常见问题

尽管我用 **MPASM** 汇编器的伪指令设置了列表文件的属性，但在列表文件中并没有显示这些属性。

虽然 **MPASM** 汇编器经常与 **MPLINK** 目标链接器配合使用，但是 **MPLINK** 链接描述文件并不支持 **MPASM** 汇编器伪指令。参见第 10.3 节“命令行接口”了解对列表和 hex 文件输出的控制。

注:

第 3 部分——MPLIB 目标库管理器

第 15 章	MPLIB 库管理器概述	227
第 16 章	库管理器界面	231
第 14 章	错误、警告和常见问题	213

注:

第 15 章 MPLIB 库管理器概述

15.1 简介

本章概述了 MPLIB 目标库管理器及其功能。

本章涉及的主题：

- MPLIB 库管理器介绍
- MPLIB 库管理器的工作原理
- MPLIB 库管理器为您提供的帮助
- 库管理器操作
- 库管理器输入 / 输出文件

15.2 MPLIB 库管理器介绍

MPLIB 目标库管理器（库管理器）将由 MPASM 汇编器或 MPLAB C18 C 编译器生成的目标模块组合到单个库文件中。然后将该文件输入到 MPLINK 目标链接器。

15.3 MPLIB 库管理器的工作原理

库管理器管理库文件的创建和修改。一个库文件就是保存在单个文件中的目标模块的集合。以下列出了创建库文件的几种理由：

- 库更易于链接。因为库文件可以包含许多目标文件，所以在链接时就可以只使用库文件名，而不是很多独立的目标文件的名称。
- 库有助于缩减代码大小。因为链接器仅仅使用一个库中所需要的目标文件，所以不必将库中的所有目标文件都放入链接器的输出模块中。
- 库使项目更易于维护。如果在项目中包含一个库文件，那么添加或删除对该库的调用不会改变链接过程。
- 库有助于传达一组目标模块的用途。因为库能够将几个相关的目标模块组合在一起，库文件的用途通常比各个目标模块的用途更容易理解。例如名称为 `math.lib` 的库文件，其用途就比 `power.o`、`ceiling.o` 和 `floor.o` 的用途更明了。

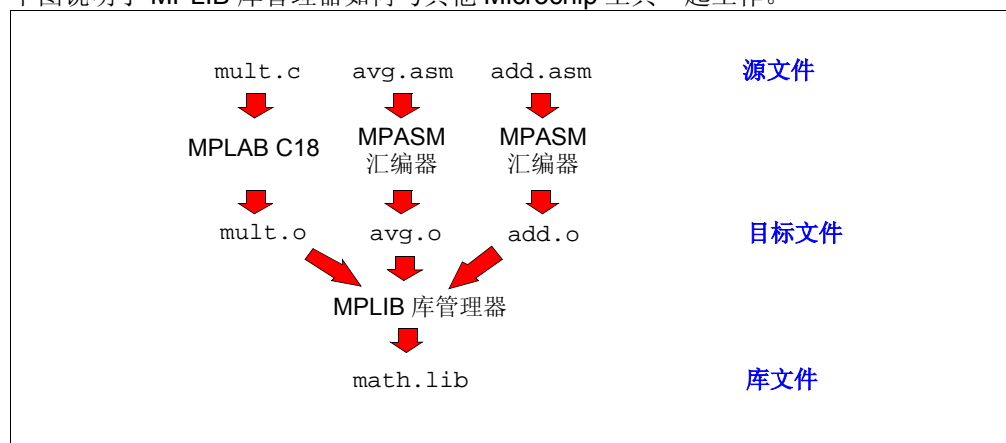
15.4 MPLIB 库管理器为您提供的帮助

MPLIB 库管理器可以在以下方面为您提供帮助：

- 由于库管理器可以包含单个的库而不是许多小文件，因此使得链接更加容易。
- 库管理器将相关的模块组合在一起，有利于代码的维护。
- 使用库管理器命令可以创建库，并能添加、列出、替换、删除或提取模块。

15.5 库管理器操作

下图说明了 MPLIB 库管理器如何与其他 Microchip 工具一起工作。



库管理器将由 MPASM 汇编器或 MPLAB C18 C 编译器生成的多个输入目标模块组合成单个输出库（.lib）文件。库文件与 MPLINK 链接器配合使用以生成可执行代码。

15.6 库管理器输入 / 输出文件

MPLIB 库管理器将多个目标文件组合为一个库（.lib）文件。

输入文件

目标文件（.o）	由源文件生成的可重定位代码。
----------	----------------

输出文件

库文件（.lib）	为方便起见将一组目标文件组合在一起。
-----------	--------------------

15.6.1 目标文件（.o）

目标文件是由源文件生成的可重定位代码。MPLIB 库管理器将几个目标文件组合成单个库文件。

15.6.2 库文件（.lib）

库文件可以使用 MPLIB 库管理器通过目标文件创建，也可以是已有的标准库。

注:

第 16 章 库管理器界面

16.1 简介

本章讨论 MPLIB 库管理器的使用方法。欲知库管理器如何与 MPASM 汇编器和 MPLINK 链接器配合使用，请参见这些工具的文档。

本章涉及以下主题：

- MPLAB IDE 界面
- 命令行选项
- 命令行示例和提示

16.2 MPLAB IDE 界面

可在 MPLAB IDE 中使用 MPLIB 库管理器来从项目目标文件而不是可执行（hex）文件创建库文件。

在 MPLAB IDE 中打开项目，选择 *Project>Build Options>Project*。在 Build Options 对话框中，单击 **MPASM/C17/C18 Suite** 选项卡。单击 “Build library target (invoke MPLIB)” 旁的单选按钮，然后单击 **OK**。现在当编译项目时就会编译库文件。

16.3 命令行选项

使用以下语法可启动 MPLIB 库管理器：

```
mplib [/q] /{ctdtx} LIBRARY [MEMBER...]
```

选项

选项	说明	详细信息
/c	创建库	创建一个新库，库中包含列出的成员。
/d	删除成员	删除库中的成员；如果没有指定成员，库就不会改变
/q	安静模式	不显示任何输出
/r	添加 / 替换成员	如果库中存在成员，那么它们则会被替换，否则会在库尾添加成员。
/t	列出成员	打印一张表格，显示库中成员的名称
/x	提取成员	如果库中存在成员，则它们将被提取。如果没有指定成员，则提取所有成员。

16.4 命令行示例和提示

使用示例

假设您要想从 `fft.o`，`fir.o` 和 `iir.o` 这三个目标模块创建一个名为 `dsp.lib` 的库。以下命令行可以产生所需的结果：

```
mplib /c dsp.lib fft.o fir.o iir.o
```

要显示名为 `dsp.lib` 的库文件中所包含的目标模块的名称，使用以下命令行：

```
mplib /t dsp.lib
```

提示

MPLIB 库管理器创建的库文件对任一符号可能只有一条外部定义。所以，如果两个目标模块定义了相同的外部符号，当把两个目标模块添加到相同的库文件时库管理器会报错。

要最大限度缩减链接库文件后生成的代码和数据空间，库成员目标模块应尽量小。创建只包含一个函数的目标模块可显著减少代码空间。

第 17 章 错误

17.1 简介

MPLIB 库管理器检测以下错误的来源并作出报告。

本章涉及以下主题：

- 库管理器解析错误
- 库文件错误
- COFF 文件错误

17.2 库管理器解析错误

下面按字母顺序列出了 MPLIB 库管理器解析错误。

Invalid Object Filename

所有的目标文件名必须以 “.o” 结束。

Invalid Switch

指定了一个不支持的选项。参见命令行选项获取所支持的选项列表。

Library Filename is Required

所有命令都需要一个库文件名。所有库文件名均必须以 “.lib” 结束。

17.3 库文件错误

参见第 14.5 节 “库文件错误” 获取库文件错误列表。

17.4 COFF 文件错误

参见第 14.6 节 “COFF 文件错误” 获取 COFF 文件错误列表。

注:

第 4 部分——附录

附录 A 指令集	237
附录 B 有用的表格	249

注:

附录 A 指令集

A.1 简介

PICmicro MCU 指令集用于使用 MPASM 汇编器、MPLINK 目标链接器和 MPLIB 目标库管理器开发应用程序。

在此列出的指令以指令宽度或器件编号分类。

指令宽度	支持的器件
12 位	PIC10F2XX、PIC12C5XX、PIC12CE5XX、PIC16X5X 和 PIC16C505
14 位	PIC12C67X、PIC12CE67X、PIC12F629/675 和 PIC16X
16 位	PIC18X

本章涉及以下主题：

- 12 或 14 位宽的指令集中的关键字
 - 12 位宽指令集
 - 14 位宽指令集
 - 12 位或 14 位宽的伪指令
- PIC18 器件指令集中的关键字
 - PIC18 器件指令集
 - PIC18 器件扩展指令集

A.2 12 或 14 位宽的指令集中的关键字

字段	说明
寄存器	
dest	目标寄存器，可以是 WREG 寄存器或指定寄存器的地址。参见 d。
f	寄存器地址（5 位、7 位或 8 位）。
p	外设寄存器地址（5 位）。
r	TRIS 端口。
x	无关位（0 或 1）。 汇编器将产生 x = 0 的代码。为了与所有 Microchip 软件工具兼容，建议使用这种格式。
立即数	
k	立即数字段、常数或标号。 k 4 位。 kk 8 位。 kkk 12 位。

汇编器 / 链接器 / 库管理器用户指南

字段	说明
位	
b	某 8 位文件寄存器内的位地址（0 到 7）。
d	目标寄存器选择位。 d = 0: 结果保存至 WREG 寄存器 d = 1: 结果保存至文件寄存器（默认）
i	表指针控制位。 i = 0: 不变。 i = 1: 指令执行后表指针加 1。
s	目标寄存器选择位 s = 0: 结果保存至文件寄存器 f 和 WREG s = 1: 结果保存至文件寄存器 f（默认）
t	表字节选择位。 t = 0: 对低字节执行操作。 t = 1: 对高字节执行操作。
' '	与十六进制值相对的位值。
命名的寄存器	
BSR	存储区选择寄存器。用于选择当前 RAM 存储区。
OPTION	OPTION 寄存器。
PCL	程序计数器的低字节。
PCH	程序计数器的高字节。
PCLATH	程序计数器的次高字节锁存器。
PCLATU	程序计数器的最高字节锁存器。
PRODH	乘积的高字节。
PRODL	乘积的低字节。
TBLATH	表锁存器（TBLAT）的高字节。
TBLATL	表锁存器（TBLAT）的低字节。
TBLPTR	16 位表指针（TBLPTRH:TBLPTRL）。指向程序存储单元。
WREG	工作寄存器（累加器）。
命名的位	
C、DC、Z、OV 和 N	ALU 状态位：进位标志位、辅助进位标志位、全零标志位、溢出标志位及负标志位。
TO	超时溢出位。
PD	掉电位。
GIE	全局中断允许位。
命名的器件功能部件	
PC	程序计数器。
TOS	栈顶。
WDT	看门狗定时器。
各种 描述符	
()	内容。
→, ↔	赋值。
< >	寄存器位域。

A.3 12 位宽指令集

Microchip 的低档 8 位单片机系列使用 12 位宽的指令集。除非另行说明。所有指令的执行时间均为一个指令周期。任何未使用的操作码被执行为一条 NOP 指令。

指令集被分为以下几类：面向字节的文件寄存器操作类指令，面向位的文件寄存器操作类指令以及内核立即数和控制操作类指令。此外，第 A.5 节“12 位或 14 位宽的伪指令”还给出了 12 位和 14 位器件均适用的指令。

指令操作码以十六进制格式显示并作了某种假设，这些操作码以关键字或脚注的形式列出。欲知有关每条指令的操作码位值、指令执行周期数和受影响的状态位的信息以及整条指令的详细信息，请参见相关器件的数据手册。

表 A-1: 12 位面向字节的文件寄存器操作

十六进制	助记符	说明	功能
1Ef*	ADDWF f, d	W 和 f 相加	WREG + f → 目标寄存器
16f*	ANDWF f, d	W 和 f 作逻辑与运算	WREG .AND. f → 目标寄存器
06f	CLRF f	将 f 清零	0 → f
040	CLRW	W 清零	0 → WREG
26f*	COMF f, d	f 取反	.NOT. f → 目标寄存器
0Ef*	DECF f, d	f 减 1	f - 1 → 目标寄存器
2Ef*	DECFSZ f, d	f 减 1，为 0 则跳过	f - 1 → 目标寄存器，为零则跳过
2Af*	INCF f, d	f 加 1	f + 1 → 目标寄存器
3Ef*	INCFSZ f, d	f 加 1，为 0 则跳过	f + 1 → 目标寄存器，为零则跳过
12f*	IORWF f, d	W 和 f 做逻辑或运算	WREG .OR. f → 目标寄存器
22f*	MOVF f, d	移动 f	f → 目标寄存器
02f	MOVWF f	将 W 的内容移动到 f	WREG → f
000	NOP	空操作	
36f*	RLF f, d	将 f 循环左移	
32f*	RRF f, d	将 f 循环右移	
0Af*	SUBWF f, d	f 减去 W	f - WREG → 目标寄存器
3Af*	SWAPF f, d	将 f 的高半字节与低半字节相交换	f(0:3) ↔ f(4:7) → 目标寄存器
1Af*	XORWF f, d	W 和 f 做逻辑异或运算	WREG .XOR. f → 目标寄存器

* 假设 d 为默认位值。

表 A-2: 12 位面向位的文件寄存器操作

十六进制	助记符	说明	功能
4bf	BCF f, b	将 f 中的某位清零	0 → f(b)
5bf	BSF f, b	将 f 中的某位置 1	1 → f(b)
6bf	BTFSC f, b	检测位，为 0 则跳过	如果 (fb) = 0，则跳过
7bf	BTFSS f, b	检测位，为 1 则跳过	如果 (fb) = 1，则跳过

表 A-3: 12 位立即数和控制操作

十六进制	助记符	说明	功能
Ekk	ANDLW kk	立即数与 W 做逻辑与运算	kk .AND. WREG → WREG
9kk	CALL kk	调用子程序	PC + 1 → TOS, kk → PC
004	CLRWDT	将看门狗定时器清零	0 → WDT （如果分配的话还有预分频器）
Akk	GOTO kk	跳转到地址（k 为 9 位立即数）	kk → PC（9 位）
Dkk	IORLW kk	立即数与 W 做逻辑或运算	kk .OR. WREG → WREG
Ckk	MOVLW kk	将立即数移动到 W	kk → WREG
002	OPTION	装载 OPTION 寄存器	WREG → OPTION 寄存器
8kk	RETLW kk	返回时将立即数保存到 W	kk → WREG, TOS → PC
003	SLEEP	进入待机模式	0 → WDT, 振荡器停振
00r	TRIS r	将端口 r 设置为三态	WREG → I/O 控制寄存器 r
Fkk	XORLW kk	立即数和 W 做逻辑异或运算	kk .OR. WREG → WREG

A.4 14 位宽指令集

Microchip 的中档 8 位单片机系列使用 14 位宽的指令集。该指令集由 36 条指令组成，每条指令均占用一个 14 位宽的字。绝大多数指令均针对文件寄存器 f 和工作寄存器 WREG（累加器）进行操作。操作的结果可以被直接保存到文件寄存器或 WREG 寄存器，在某些指令中还被同时保存到上述两个寄存器中。有少数指令仅针对文件寄存器进行操作（如 BSF）。

该指令集被分为以下几类：面向字节的文件寄存器操作类指令，面向位的文件寄存器操作类指令以及内核立即数和控制操作类指令。此外，第 A.5 节“12 位或 14 位宽的伪指令”还给出了 12 位和 14 位器件均适用的指令。

指令操作码以十六进制格式显示并作了某种假设，这些操作码以关键字或脚注的形式列出。欲知有关每条指令的操作码位值、指令执行周期数和受影响的状态位的信息以及整条指令的详细信息，请参见相关器件的数据手册。

表 A-4: 14 位面向字节的文件寄存器操作

十六进制	助记符	说明	功能
07df	ADDWF f, d	W 和 f 相加	W + f → d
05df	ANDWF f, d	W 和 f 做逻辑与运算	W .AND. f → d
01'1'f	CLRF f	将 f 清零	0 → f
01xx	CLRW	将 W 清零	0 → W
09df	COMF f, d	f 取反	.NOT. f → d
03df	DECF f, d	f 减 1	f - 1 → d
0Bdf	DECFSZ f, d	f 减 1，为 0 则跳过	f - 1 → d，为零则跳过
0Adf	INCF f, d	f 加 1	f + 1 → d
0Fdf	INCFSZ f, d	f 加 1，为 0 则跳过	f + 1 → d，为零则跳过
04df	IORWF f, d	W 和 f 做逻辑或运算	W .OR. f → d
08df	MOVF f, d	移动 f	f → d
00'1'f	MOVWF f	将 W 的内容移动到 f	W → f
0000	NOP	空操作	
0Ddf	RLF f, d	将 f 循环左移	

表 A-4: 14 位面向字节的文件寄存器操作 (续)

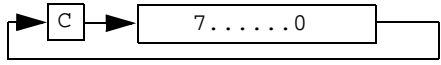
十六进制	助记符	说明	功能
0Cdf	RRF f, d	将 f 循环右移	
02df	SUBWF f, d	f 减去 W	$f - W \rightarrow d$
0Edf	SWAPF f, d	将 f 的高半字节与低半字节相交换	$f(0:3) \leftrightarrow f(4:7) \rightarrow d$
06df	XORWF f, d	W 和 f 做逻辑异或运算	$W .XOR. f \rightarrow d$

表 A-5: 14 位面向位的文件寄存器操作

十六进制	助记符	说明	功能
4bf	BCF f, b	将 f 中的某位清零	$0 \rightarrow f(b)$
5bf	BSF f, b	将 f 中的某位置 1	$1 \rightarrow f(b)$
6bf	BTFSC f, b	检测位, 为 0 则跳过	如果 $(fb) = 0$, 则跳过
7bf	BTFSS f, b	检测位, 为 1 则跳过	如果 $(fb) = 1$, 则跳过

表 A-6: 14 位立即数和控制操作

十六进制	助记符	说明	功能
3Ekk	ADDLW kk	立即数与 W 相加	$kk + WREG \rightarrow WREG$
39kk	ANDLW kk	立即数与 W 做逻辑与运算	$kk .AND. WREG \rightarrow WREG$
2'0'kkk	CALL kkk	调用子程序	$PC + 1 \rightarrow TOS, kk \rightarrow PC$
0064	CLRWDI	将看门狗定时器清零	$0 \rightarrow WDT$ (如果分配的话还有预分频器)
2'1'kkk	GOTO kkk	跳转到地址 (k 为 9 位立即数)	$kk \rightarrow PC$ (9 位)
38kk	IORLW kk	立即数与 W 做逻辑或运算	$kk .OR. WREG \rightarrow WREG$
30kk	MOVLW kk	将立即数移动到 W	$kk \rightarrow WREG$
0062	OPTION	装载 OPTION 寄存器	$WREG \rightarrow OPTION$ 寄存器
0009	RETFIE	中断返回	$TOS \rightarrow PC, 1 \rightarrow GIE$
34kk	RETLW kk	返回时将立即数保存到 W	$kk \rightarrow WREG, TOS \rightarrow PC$
0008	RETURN	从子程序返回	$TOS \rightarrow PC$
0063	SLEEP	进入待机模式	$0 \rightarrow WDT$, 振荡器停振
3Ckk	SUBLW kk	立即数减去 W	$kk - WREG \rightarrow WREG$
006r	TRIS r	将端口 r 设置为三态	$WREG \rightarrow I/O$ 控制寄存器
3Akk	XORLW kk	立即数和 W 做逻辑异或运算	$kk .OR. WREG \rightarrow WREG$

A.5 12 位或 14 位宽的伪指令

以下伪指令在 12 位和 14 位指令字器件中均适用。这些伪指令是标准 PICmicro 指令的另一种助记符，或是可生成一条或多条 PICmicro 指令的宏。建议不要在新设计中使用这些伪指令。在此列出主要供前后参考。

表 A-7: 12 位和 14 位特殊指令助记符

助记符	说明	等效操作	状态
ADDCF f, d	将文件寄存器与进位相加	BTFSC 3, 0 INCF f, d	Z
ADDDCF f, d	将文件寄存器与辅助进位相加	BTFSC 3, 1 INCF f, d	Z
B k	跳转	GOTO k	-
BC k	发生进位时跳转	BTFSC 3, 0 GOTO k	-
BDC k	发生辅助进位时跳转	BTFSC 3, 1 GOTO k	-
BNC k	未发生进位时跳转	BTFSS 3, 0 GOTO k	-
BNDC k	未发生辅助进位时跳转	BTFSS 3, 1 GOTO k	-
BNZ k	结果不为零时跳转	BTFSS 3, 2 GOTO k	-
BZ k	结果为零时跳转	BTFSC 3, 2 GOTO k	-
CLRC	清除进位标志	BCF 3, 0	-
CLRDC	清除辅助进位标志	BCF 3, 1	-
CLRZ	清除结果为零标志	BCF 3, 2	-
LCALL k	长调用	BCF/BSF 0x0A, 3 BCF/BSF 0x0A, 4 CALL k	
LGOTO k	长跳转	BCF/BSF 0x0A, 3 BCF/BSF 0x0A, 4 GOTO k	
MOVFW f	将文件寄存器的值移动到 W	MOVF f, 0	Z
NEGF f, d	将文件寄存器的值取补	COMF f, 1 INCF f, d	Z
SETC	将进位标志位置 1	BSF 3, 0	-
SETDC	将辅助进位标志位置 1	BSF 3, 1	-
SETZ	将结果为零标志位置 1	BSF 3, 2	-
SKPC	发生进位时跳过	BTFSS 3, 0	-
SKPDC	发生辅助进位时跳过	BTFSS 3, 1	-
SKPNC	未发生进位时跳过	BTFSC 3, 0	-
SKPNDC	未发生辅助进位时跳过	BTFSC 3, 1	-
SKPNZ	结果不为零时跳过	BTFSC 3, 2	-
SKPZ	结果为零时跳过	BTFSS 3, 2	-
SUBCF f, d	将文件寄存器的值减去进位	BTFSC 3, 0 DECF f, d	Z

表 A-7: 12 位和 14 位特殊指令助记符 (续)

助记符	说明	等效操作	状态
SUBDCF f, d	将文件寄存器的值减去辅助进位	BTFSC 3, 1 DECF f, d	Z
TSTF f	测试文件寄存器	MOVF f, 1	Z

A.6 PIC18 器件指令集中的关键字

字段	说明
寄存器	
dest	目标寄存器，可以是 WREG 寄存器或指定寄存器的地址。参见 d。
f	文件寄存器地址。 f 8 位 (0x00 到 0xFF)。 f' 12 位 (0x000 到 0xFFF)。这是源地址。 f" 12 位 (0x000 到 0xFFF)。这是目标地址。
r	FSR 编号，可以为 0、1 或 2。
x	无关位 (0 或 1)。 汇编器将产生 x = 0 的代码。为了与所有的 Microchip 软件工具兼容，建议使用这种格式。
z	间接寻址偏移量。 z' 对源文件寄存器进行间接寻址的 7 位偏移量。 z" 对目标文件寄存器进行间接寻址的 7 位偏移量。
立即数	
k	立即数字段、常数或标签。 k 4 位。 kk 8 位。 kkk 12 位。
向上或向下偏移	
n	相对转移指令的相对地址 (二的补码数)，或 Call/Branch 和 Return 指令的直接地址。
*	表读和表写指令的 TBLPTR 寄存器模式。
*+	只和表读 (TBLRD) 和表写 (TBLWT) 指令一起使用： 寄存器不变
*-	寄存器后加 1
++	寄存器后减 1
++	寄存器预加 1
位	
a	快速操作 RAM 位 a = 0: 快速操作 RAM 中的 RAM 单元 (BSR 寄存器被忽略) a = 1: RAM 存储区由 BSR 寄存器指定 (默认)
b	某 8 位文件寄存器内的位地址 (0 到 7)。
d	目标寄存器选择位 d = 0: 结果保存至 WREG 寄存器 d = 1: 结果保存至文件寄存器 f (默认)
s	快速调用 / 返回模式选择位。 s = 0: 不更新影子寄存器也不使用影子寄存器更新其他寄存器 (默认) s = 1: 将特定寄存器的值存入影子寄存器或把影子寄存器的值装入特定的寄存器 (快速模式)
' '	与十六进制值相对的位值。
命名的寄存器	
BSR	存储区选择寄存器。用于选择当前的 RAM 存储区。
FSR	指针寄存器。
PCL	程序计数器的低字节。

字段	说明
PCH	程序计数器的高字节。
PCLATH	程序计数器的次高字节锁存器。
PCLATU	程序计数器的最高字节锁存器。
PRODH	乘积的高字节。
PRODL	乘积的低字节。
STATUS	STATUS 寄存器
TABLAT	8 位表锁存器。
TBLPTR	21 位表指针（指向程序存储单元）。
WREG	工作寄存器（累加器）。
命名的位	
C、DC、Z、OV 和 N	ALU 状态位：进位标志位、辅助进位标志位、全零标志位、溢出标志位及负标志位。
TO	超时溢出位。
PD	掉电位。
PEIE	外设中断允许位
GIE 和 GIEL/H	全局中断允许位。
命名的器件功能部件	
MCLR	器件主复位。
PC	程序计数器。
TOS	栈顶。
WDT	看门狗定时器。
各种 描述符	
()	内容。
→	赋值。
< >	寄存器位域。

A.7 PIC18 器件指令集

Microchip 的新型高性能 8 位单片机系列使用 16 位宽的指令集。该指令集由 76 条指令组成，每条指令均占用一个 16 位宽的字（2 个字节）。绝大多数指令均针对文件寄存器 **f** 和工作寄存器 **WREG**（累加器）进行操作。操作的结果可以被直接保存到文件寄存器或 **WREG** 寄存器，在某些指令中还被同时保存到上述两个寄存器中。有少数指令仅针对文件寄存器进行操作（如 **BSF**）。

该指令集被分为以下几类：面向字节的文件寄存器操作类指令、面向位的文件寄存器操作类指令、控制操作类指令、立即数操作类指令和存储器操作类指令。此外，**第 A.8 节“PIC18 器件扩展指令集”**中还给出了扩展模式的指令。

指令操作码以十六进制格式显示并作了某种假设，这些操作码以关键字或脚注的形式列出。欲知有关每条指令的操作码位值、指令执行周期数和受影响的状态位的信息以及整条指令的详细信息，请参见相关器件的数据手册。

表 A-8: PIC18 面向字节的寄存器操作

十六进制	助记符	说明	功能
27f*	ADDWF <i>f, d, a</i>	WREG 与 <i>f</i> 相加	WREG+ <i>f</i> → 目标寄存器
23f*	ADDWFC <i>f, d, a</i>	WREG 与 <i>f</i> 带进位相加	WREG+ <i>f</i> +C → 目标寄存器
17f*	ANDWF <i>f, d, a</i>	WREG 与 <i>f</i> 做逻辑与运算	WREG .AND. <i>f</i> → 目标寄存器
6Bf*	CLRF <i>f, a</i>	将 <i>f</i> 清零	0 → <i>f</i>

表 A-8: PIC18 面向字节的寄存器操作 (续)

十六进制	助记符	说明	功能
1Ff*	COMF f,d,a	f 取反	.NOT.f → 目标寄存器
63f*	CPFSEQ f,a	将 f 与 WREG 相比较, 如果 f = WREG 则跳过	f ← WREG, 如果 f = WREG, PC+4 → PC 否则 PC + 2 → PC
65f*	CPFSGT f,a	将 f 与 WREG 相比较, 如果 f > WREG 则跳过	f ← WREG, 如果 f > WREG, PC+4 → PC 否则 PC + 2 → PC
61f*	CPFSLT f,a	将 f 与 WREG 相比较, 如果 f < WREG 则跳过	f ← WREG, 如果 f < WREG, PC+4 → PC 否则 PC + 2 → PC
07f*	DECF f,d,a	f 减 1	f ← 1 → dest
2Ff*	DECFSZ f,d,a	f 减 1, 为 0 时跳过	f ← 1 → dest, 如果 dest = 0, PC+4 → PC 否则 PC + 2 → PC
4Ff*	DCFSNZ f,d,a	f 减 1, 非 0 时跳过	f ← 1 → dest, 如果 dest ≠ 0, PC+4 → PC 否则 PC + 2 → PC
2Bf*	INCF f,d,a	f 加 1	f + 1 → dest
3Ff*	INCFSZ f,d,a	f 加 1, 为 0 时跳过	f+1 → dest, 如果 dest = 0, PC+4 → PC 否则 PC + 2 → PC
4Bf*	INFSNZ f,d,a	f 加 1, 非 0 时跳过	f+1 → dest, 如果 dest ≠ 0, PC+4 → PC 否则 PC + 2 → PC
13f*	IORWF f,d,a	WREG 与 f 做逻辑或运算	WREG .OR. f → 目标寄存器
53f*	MOVF f,d,a	移动 f	f → 目标寄存器
Cf Ff"	MOVFF f',f"	将源寄存器 f' 的内容移动到目标寄存器 fd" (第二个字)	f' → f"
6Ff*	MOVWF f,a	将 WREG 的内容移动到 f	WREG → f
03f*	MULWF f,a	WREG 与 f 相乘	WREG * f → PRODH:PRODL
6Df*	NEGF f,a	f 取补	-f → f
37f*	RLCF f,d,a	对 f 执行带进位的循环左移	
47f*	RLNCF f,d,a	将 f 循环左移 (不带进位)	
33f*	RRCF f,d,a	对 f 执行带进位的循环右移	
43f*	RRNCF f,d,a	将 f 循环右移 (不带进位)	
69f*	SETF f,a	将 f 置为全 1	0xFF → f
57f*	SUBFWB f,d,a	WREG 减去 f (带借位)	WREG - f - C → 目标寄存器
5Ff*	SUBWF f,d,a	从 f 减去 WREG	f - WREG → 目标寄存器
5Bf*	SUBWFB f,d,a	f 减去 WREG (带借位)	f - WREG - DC → 目标寄存器
3Bf*	SWAPF f,d,a	将 f 的高半字节和低半字节相交	f<3:0> → 目标寄存器 <7:4>, f<7:4> → dest<3:0>
67f*	TSTFSZ f,a	测试 f, 为 0 时跳过	如果 f = 0, PC+4 → PC, 否则 PC+2 → PC
1Bf*	XORWF f,d,a	WREG 与 f 做逻辑异或运算	WREG .XOR. f → 目标寄存器

* 假设 d 和 a 为默认位值。

汇编器 / 链接器 / 库管理器用户指南

表 A-9: PIC18 面向位的寄存器操作

十六进制	助记符	说明	功能
91f*	BCF f, b, a	将 f 中的某位清零	$0 \rightarrow f$
81f*	BSF f, b, a	将 f 中的某位置 1	$1 \rightarrow f$
B1f*	BTFSF f, b, a	检测 f 中的某位, 为 0 则跳过	如果 $f = 0$, $PC+4 \rightarrow PC$, 否则 $PC+2 \rightarrow PC$
A1f*	BTFS f, b, a	检测 f 中的某位, 为 1 则跳过	如果 $f = 1$, $PC+4 \rightarrow PC$, 否则 $PC+2 \rightarrow PC$
71f*	BTG f, b, a	将 f 中的某位取反	$f \rightarrow \overline{f}$

* 假设 b = 0 并且 a 为默认值。

表 A-10: PIC18 控制操作

十六进制	助记符	说明	功能
E2n	BC n	发生进位则跳转	如果 $C = 1$, $PC+2+2*n \rightarrow PC$, 否则 $PC+2 \rightarrow PC$
E6n	BN n	为负则跳转	如果 $N = 1$, $PC+2+2*n \rightarrow PC$, 否则 $PC+2 \rightarrow PC$
E3n	BNC n	未发生进位则跳转	如果 $C = 0$, $PC+2+2*n \rightarrow PC$, 否则 $PC+2 \rightarrow PC$
E7n	BNN n	不为负则跳转	如果 $N = 0$, $PC+2+2*n \rightarrow PC$, 否则 $PC+2 \rightarrow PC$
E5n	BN OV n	不溢出则跳转	如果 $OV = 0$, $PC+2+2*n \rightarrow PC$, 否则 $PC+2 \rightarrow PC$
E1n	BN Z n	不为零则跳转	如果 $Z = 0$, $PC+2+2*n \rightarrow PC$, 否则 $PC+2 \rightarrow PC$
E4n	BOV n	发生溢出则跳转	如果 $OV = 1$, $PC+2+2*n \rightarrow PC$, 否则 $PC+2 \rightarrow PC$
D'0'n	BRA n	无条件跳转	$PC+2+2*n \rightarrow PC$
E0n	BZ n	为零则跳转	如果 $Z = 1$, $PC+2+2*n \rightarrow PC$, 否则 $PC+2 \rightarrow PC$
ECkk* Fkkk	CALL n, s	调用子程序 第一个字 第二个字	$PC+4 \rightarrow TOS$, $n \rightarrow PC<20:1>$, 如果 $s = 1$, $WREG \rightarrow WREGs$, $STATUS \rightarrow STATUSs$, $BSR \rightarrow BSRs$
0004	CLRWDT	将看门狗定时器清零	$0 \rightarrow WDT$, $0 \rightarrow WDT$ 后分频器, $1 \rightarrow \overline{TO}$, $1 \rightarrow \overline{PD}$
0007	DAW	对 WREG 进行十进制调整	如果 $WREG<3:0> > 9$ 和 $DC = 1$, $WREG<3:0>+6 \rightarrow WREG<3:0>$, 否则 $WREG<3:0> \rightarrow WREG<3:0>$; 如果 $WREG<7:4> > 9$ 或 $DC = 1$, $WREG<7:4>+6 \rightarrow WREG<7:4>$, 否则 $WREG<7:4> \rightarrow WREG<7:4>$;
EFkk Fkkk	GOTO n	跳转到地址 第一个字 第二个字	$n \rightarrow PC<20:1>$
0000	NOP	空操作	空操作
Fxxx	NOP	空操作	空操作 (双字指令)
0006	POP	将返回堆栈栈顶 (TOS) 内容弹出	$TOS-1 \rightarrow TOS$
0005	PUSH	将数据压入返回堆栈的栈顶 (TOS)	$PC+2 \rightarrow TOS$
D'1'n	RCALL n	相对调用	$PC+2 \rightarrow TOS$, $PC+2+2*n \rightarrow PC$
00FF	RESET	器件的软件复位	与 \overline{MCLR} 复位相同
0010*	RETFIE s	中断返回 (并允许中断)	$TOS \rightarrow PC$, $1 \rightarrow GIE/GIEH$ 或 $PEIE/GIEL$, 如果 $s = 1$, $WREGs \rightarrow WREG$, $STATUSs \rightarrow STATUS$, $BSRs \rightarrow BSR$, $PCLATU/PCLATH$ 不变。
0012*	RETURN s	从子程序返回。	$TOS \rightarrow PC$, 如果 $s = 1$, $WREGs \rightarrow WREG$, $STATUSs \rightarrow STATUS$, $BSRs \rightarrow BSR$, $PCLATU$ 和 $PCLATH$ 不变
0003	SLEEP	进入休眠模式	$0 \rightarrow WDT$, $0 \rightarrow WDT$ 后分频器, $1 \rightarrow \overline{TO}$, $1 \rightarrow \overline{PD}$

* 假设 d 为默认位值。

表 A-11: PIC18 立即数操作

十六进制	助记符	说明	功能
0Fkk	ADDLW kk	将立即数与 WREG 相加	WREG+kk → WREG
0Bkk	ANDLW kk	立即数与 WREG 做逻辑与运算	WREG .AND. kk → WREG
09kk	IORLW kk	立即数与 WREG 做逻辑或运算	WREG .OR. kk → WREG
EErk F0kk	LFSR r, kk	将立即数 (12 位) (第二个字) 移动到 FSRr (第一个字)	kk → FSRr
010k	MOVLB k	将立即数移动到 BSR<3:0>	kk → BSR
0Ekk	MOVLW kk	将立即数移动到 WREG	kk → WREG
0Dkk	MULLW kk	立即数与 WREG 相乘	WREG * kk → PRODH:PRODL
0Ckk	RETLW kk	返回时将立即数保存到 WREG	kk → WREG
08kk	SUBLW kk	从立即数减去 WREG	kk-WREG → WREG
0Akk	XORLW kk	立即数与 WREG 做逻辑异或运算	WREG .XOR. kk → WREG

表 A-12: PIC18 存储器操作

十六进制	助记符	说明	功能
0008	TBLRD*	表读	程序存储区 (TBLPTR) → TABLAT
0009	TBLRD*+	表指针在表读后加 1	程序存储区 (TBLPTR) → TABLAT TBLPTR +1 → TBLPTR
000A	TBLRD*-	表指针在表读后减 1	程序存储区 (TBLPTR) → TABLAT TBLPTR -1 → TBLPTR
000B	TBLRD+*	表指针在表读前加 1	TBLPTR +1 → TBLPTR 程序存储区 (TBLPTR) → TABLAT
000C	TBLWT*	表写	TABLAT → 程序存储区 (TBLPTR)
000D	TBLWT*+	表指针在表写后加 1	TABLAT → 程序存储区 (TBLPTR) TBLPTR +1 → TBLPTR
000E	TBLWT*-	表指针在表写后减 1	TABLAT → 程序存储区 (TBLPTR) TBLPTR -1 → TBLPTR
000F	TBLWT+*	表指针在表写前加 1	TBLPTR +1 → TBLPTR TABLAT → 程序存储区 (TBLPTR)

A.8 PIC18 器件扩展指令集

某些 PIC18 器件具有用于 MPLAB C18 编译器的扩展操作模式。该模式会改变第 A.7 节“PIC18 器件指令集”中列出的一些指令的操作并且还会增加上述小节列出的指令。

一般情况下，您无需使用扩展指令集。但是，需要时，可以使用特殊器件配置位设置该扩展模式。关于扩展模式的更多信息，请参见《MPLAB C18 C 编译器用户指南》(DS51288C_CN) 和相应器件的数据手册。

指令操作码以十六进制格式显示并作了某种假设，这些操作码以关键字或脚注的形式列出。欲知有关每条指令的操作码位值、指令执行周期数和受影响的状态位的信息以及整条指令的详细信息，请参见相关器件的数据手册。

表 A-13: PIC18 扩展指令集

十六进制	助记符	说明	功能
E8fk	ADDFSR f, k	将立即数与 FSR 相加	FSR(f)+k → FSR(f)
E8Ck	ADDULNK k	将立即数与 FSR2 相加并返回	FSR2+k → FSR2, (TOS) → PC
0014	CALLW	使用 WREG 调用子程序	(PC + 2) → TOS, (W) → PCL, (PCLATH) → PCH, (PCLATU) → PCU
EB'0'z Ffff	MOVSF z', f"	将 z' (源) 第一个字， 移动到 f" (目标) 第二个字	((FSR2)+z') → f"
EB'1'z Fxzz	MOVSS z', z"	将 z' (源) 第一个字， 移动到 z" (目标) 第二个字	((FSR2)+z') → ((FSR2)+z")
EAKk	PUSHL k	将立即数保存至 FSR2， FSR2 减 1	k → (FSR2), FSR2-1 → FSR2
E9fk	SUBFSR f, k	从 FSR 减去立即数	FSR(f-k) → FSR(f)
E9Ck	SUBULNK k	从 FSR2 减去立即数并返回	FSR2-k → FSR2, (TOS) → PC

附录 B 有用的表格

B.1 简介

本附录包含一些供参考的有用表格。表格如下：

- ASCII 字符集
- 十六进制转换到十进制

B.2 ASCII 字符集

最低有效半字节	最高有效半字节								
	十六进制数	0	1	2	3	4	5	6	7
	0	NUL	DLE	Space	0	@	P	`	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	\$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	Bell	ETB	'	7	G	W	g	w
	8	BS	CAN	(8	H	X	h	x
	9	HT	EM)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[k	{
	C	FF	FS	,	<	L	\	l	
	D	CR	GS	ó	=	M]	m	}
E	SO	RS	.	>	N	^	n	~	
F	SI	US	/	?	O	_	o	DEL	

B.3 十六进制转换到十进制

本附录说明了十六进制转换到十进制的方法。对于每个十六进制数字，找到相应的十进制值；把数字加起来。

高字节				低字节			
HEX 1000	Dec	HEX 100	Dec	HEX 10	Dec	HEX 1	Dec
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

例如，十六进制数 **A38F** 按如下方式转换为 **41871**：

十六进制数的千位	十六进制数的百位	十六进制数的十位	十六进制数的个位	结果
40960	768	128	15	41871 十进制数

术语表

ANSI

美国国家标准学会（American National Standards Institute，ANSI）是美国负责制定和核准标准的组织。

ASCII

美国信息交换标准码是使用 7 位二进制数来表示每个字符的字符集编码。它包括大小写字母、数字、符号和控制符。

Build

为应用程序编译和链接所有源文件。

编译器（Compiler）

将用高级语言编写的程序翻译成机器码的程序。

C

一种通用的编程语言，具有精简的表达式、现代的控制流和数据结构，以及丰富的运算符集。

COFF

公共目标文件格式。此格式的目标文件包含机器码、调试信息和其他信息。

操作码（Opcodes）

操作码。参见助记符（Mnemonics）。

程序存储器（Program Memory）

器件中存储指令的存储区。也可能是仿真器或软件模拟器中包含已下载的目标应用固件的存储器。

程序计数器（Program Counter）

包含当前正在执行的指令的地址的存储单元。

触发输出（Trigger Output）

触发输出是指可在任何地址或地址范围产生的仿真器输出信号，它独立于跟踪和断点设置。可以设置任何数量的触发输出点。

次数计数器（Pass Counter）

每当事件（如在某个特定地址执行指令）发生时进行递减计数的计数器。当次数计数值达到零时，对该事件做出响应。可以将次数计数器指派给中断和跟踪逻辑，也可以将它指派给 complex trigger（复杂触发）对话框中的任何连续事件。

存档（Archive）

可重定位的目标模块集合。通过将多个源文件汇编为目标文件，然后使用归档器将这些目标文件组合为一个库文件来创建。库文件可以与目标模块和其他库文件链接以生成可执行代码。

DSC

参见数字信号控制器（Digital Signal Controller）。

单步执行（Single Step）

此命令以一次执行一条指令的方式执行代码。每条指令执行后，MPLAB IDE 会更新寄存器窗口、观察变量和状态显示，从而使您能分析和调试指令执行。也可以单步执行 C 编译器的源代码，但是 MPLAB IDE 并不是执行单条指令而是执行所有由这行高级 C 语句生成的汇编级指令。

单片机（Microcontroller）

一种高度集成的芯片，包含 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 系列单片机的一种可能的程序存储器配置。在单片机模式下，只允许执行内部存储器中的指令。因此，在单片机模式下只能使用片上程序存储器。

导出（Export）

以标准化的格式将数据从 MPLAB IDE 输出。

导入（Import）

将数据（IDE）从外部源（例如，Hex 文件）传送到 MPLAB IDE 中。

递归（Recursion）

递归为一个已定义的函数或宏可以调用自身的概念。写递归的宏时务必小心；当没有从递归退出的出口时，很容易陷入无限循环。

地址（Address）

标识存储器中位置的值。

EEPROM

电可擦除可编程只读存储器。一种可电擦除的特殊类型的 PROM。一次可写入或擦除一个字节的数据。即使关断电源，EEPROM 仍能保留自身内容。

EPROM

可擦除可编程只读存储器。通常可通过曝露在紫外线下而擦除的可编程只读存储器。

FNOP

强制性空操作。强制性 NOP 周期是双周期指令的第二个周期。由于 PICmicro 单片机采用流水线架构，所以当它执行当前指令时，会预取物理地址空间内的下一条指令。然而，如果当前指令更改了程序计数器，这条预取的指令将被忽略，引起一个强制性的 NOP 周期。

仿真（Emulation）

执行装入仿真存储器的软件的过程，就好象该软件是驻留在单片机上的固件一样。

仿真存储器（Emulation Memory）

包含在仿真器内的程序存储器。

仿真器（Emulator）

执行仿真的硬件。

仿真器系统（Emulator System）

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

仿真器主机（Pod, Emulator）

包含仿真存储器、跟踪存储器、事件和周期定时器以及跟踪 / 断点逻辑的外部仿真器机箱。

非实时 (Non Real-Time)

指处理器处于断点或正在单步执行指令或者 MPLAB IDE 正运行在软件模拟器模式下。

非易失性存储器 (Non-Volatile Storage)

关断电源后，能保存自身内容的存储设备。

符号 (Symbol)

符号是用于描述组成程序的各个部分的通用机制。这些部分包括函数名、变量名、段名、文件名、结构 / 枚举类 / 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，用于常规用途。

概要文件 (Profile)

对于 MPLAB SIM 软件模拟器来说，该文件是对寄存器执行的激励的摘要列表。

高级语言 (High Level Language)

用于编写比汇编语言更加脱离处理器机器码的程序的语言。

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器中，该缓冲器中的内容将被上载到 MPLAB IDE 的 trace (跟踪) 窗口。

跟踪存储器 (Trace Memory)

跟踪存储器包含在仿真器内。跟踪存储器有时也称为跟踪缓冲器。

工具栏 (Tool Bar)

一行或一列图标，可以在其中单击以执行 MPLAB IDE 功能。

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。

归档器 (Archiver)

创建和管理库的工具。

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

在很多业务和技术方面 (包括计算和通信) 设置标准的组织。

HEX 代码 (HEX Code)

以十六进制格式存储的可执行指令代码。HEX 代码包含在 HEX 文件中。

HEX 文件 (HEX File)

一种包含用于对器件进行编程的十六进制地址和值 (HEX 代码) 的 ASCII 文件。

宏 (Macro)

宏指令。以缩写形式表示一系列指令的指令。

宏伪指令 (Macro Directives)

在宏体定义内部控制执行和数据分配的伪指令。

环境——IDE (Environment – IDE)

用于开发应用程序的特定桌面布局。

环境——MPLAB PM3 (Environment – MPLAB PM3)

包含有关如何对器件编程的文件的文件夹。此文件夹可被传输到一个 SD/MMC 卡。

汇编器 (Assembler)

将汇编语言源代码翻译成机器码的语言工具。

汇编语言 (Assembly Language)

用符号形式描述二进制机器码的编程语言。

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。

IRQ

参见中断请求 (Interrupt Request)。

ISO

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

ISR

参见中断服务程序 (Interrupt Service Routine)。

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。通常这些数据以动作列表的形式放在文本文件中。激励可能是异步、同步（引脚）、时钟激励和寄存器激励。

机器码 (Machine Code)

处理器实际读取和解释的计算机程序表示。用二进制机器码编写的程序由一系列机器指令组成（可能还带有数据）。某个特定处理器的所有可用指令的集合称为它的“指令集”。

机器语言 (Machine Language)

用于特定中央处理单元的一组指令，旨在无需翻译即可由处理器使用。

基数 (Radix)

数基，十六进制或十进制，用于指定地址。

交叉引用文件 (Cross Reference File)

引用符号表的文件和引用符号的文件列表。如果符号已被定义，那么列出的第一个文件是该定义的位置。剩余的文件包含对该符号的引用。

校准存储区 (Calibration Memory)

用来保存 PICmicro 单片机的片上 RC 振荡器或其他器件外设的校准值的特殊功能寄存器。

节点 (Node)

MPLAB IDE 项目组件。

警告 (Warning)

向您提供的警告，以在将对器件、软件文件或设备造成物理损害的情形下发出警告。

静态 RAM 或 SRAM (Static RAM or SRAM)

静态随机访问存储器。可在目标板上读写的程序存储器，无需经常刷新。

局部标号 (Local Label)

是用 LOCAL 伪指令在宏内部定义的标号。这些标号特定于宏实例化的一个给定实例。换句话说，当遇到 ENDM 宏之后，将不再能访问声明为 local 的符号和标号。

绝对段 (Absolute Section)

具有固定（绝对）地址且不能被链接器更改的段。

看门狗定时器 (Watchdog Timer)

PICmicro 单片机上的一个定时器，它可在可选的时间长度之后复位处理器。可通过使用配置位使能、禁止或设置 WDT。

控制伪指令 (Control Directives)

汇编语言代码中的伪指令，它可以根据指定表达式的汇编时值包含或省略代码。

库 (Library)

参见存档 (Archive)。

库管理器 (Librarian)

参见归档器 (Archiver)。

快速操作存储区 (仅 PIC18) (Access Memory (PIC18 Only))

无论存储区选择寄存器 (bank select register, BSR) 的设置如何都允许访问的 PIC18 特殊寄存器。

扩展的单片机模式 (Extended Microcontroller Mode)

在扩展的单片机模式下，片上程序存储器和外部存储器都可用。如果程序存储器的地址超出了 PIC17 或 PIC18 器件的内部存储空间，程序执行将自动切换到外部存储器。

链接器 (Linker)

通过解析从一个模块到另一个模块的引用，将目标文件和库相结合以生成可执行代码的语言工具。

链接描述文件 (Linker Script Files)

链接描述文件是链接器的命令文件。它们定义链接器选项并描述目标平台上的可用存储器。

列表伪指令 (Listing Directives)

列表伪指令是那些控制汇编器列表文件格式的伪指令。它们实现了标题、分页和其他列表控制的规范。

列表文件 (Listing File)

列表文件是一种 ASCII 文本文件，它显示了为每个 C 源语句、汇编指令、汇编器伪指令或在源文件中遇到的宏生成的机器码。

逻辑探头 (Logic Probes)

至多 14 个逻辑探头可被连接到某些 Microchip 仿真器。逻辑探头提供外部跟踪输入、触发输出信号、+5V 电压以及一个公共接地端。

Make 项目（Make Project）

重新编译应用程序的命令，只重新编译那些自从上次编译完成后已更改的源文件。

MCU

单片机器件。Microcontroller 的缩写。另一种缩写是 uC。

MPASM 汇编器（MPASM Assembler）

Microchip PICmicro 单片机、KeeLoq 器件和 Microchip 存储器的可重定位的宏汇编器。

MPLAB ASM30

Microchip 提供的 dsPIC30F 数字信号控制器可使用的可重新定位的宏汇编器。

MPLAB C1X

指 Microchip 的 MPLAB C17 和 C18 C 编译器。

MPLAB C17 是 PIC17 器件的 C 编译器，而 C18 是 PIC18 器件的 C 编译器。

MPLAB C30

Microchip dsPIC30F 数字信号控制器的 C 编译器。

MPLAB ICD 2

与 MPLAB IDE 配合工作的 Microchip 在线调试器。ICD 支持带有内置调试电路的闪存器件。每个 ICD 的主要组件均为调试模块。完整的系统由模块、适配头、演示板、电缆和 MPLAB IDE 软件组成。

MPLAB ICE 2000/4000

与 MPLAB IDE 配合工作的 Microchip 在线仿真器。MPLAB ICE2000 支持 PICmicro MCU。MPLAB ICE 4000 支持 PIC18F MCU 和 dsPIC30F DSC。每个 ICE 的主要组件为仿真器主机。完整的系统由主机、处理器模块、线缆和 MPLAB IDE 软件构成。

MPLAB IDE

Microchip 的集成开发环境。

MPLAB LIB30

MPLAB LIB30 归档器/库管理器是用于管理和组织由 MPLAB ASM30 或 MPLAB C30 C 编译器生成的 COFF 目标模块的目标库管理器。

MPLAB LINK30

MPLAB LINK30 是 Microchip MPLAB ASM30 汇编器和 Microchip MPLAB C30 C 编译器的目标链接器。

MPLAB PM3

来自 Microchip 的器件编程器。为 PIC18 单片机和 dsPIC 数字信号控制器编程。可在 MPLAB IDE 中使用，也可单独使用。它将取代 PRO MATE II。

MPLAB SIM

在 MPLAB IDE 中使用的 Microchip 软件模拟器，支持 PICmicro MCU 和 dsPIC DSC 器件。

MPLIB 目标库管理器（MPLIB Object Librarian）

MPLIB 库管理器是管理和组织由 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 C18 C 编译器的目标链接器。MPLINK 链接器还可以与 Microchip MPLIB 库管理器一起使用。MPLINK 链接器是为在 MPLAB IDE 中使用而设计的，但是并不一定要在 MPLAB IDE 中使用它。

MRU

最近使用过的。指可从 MPLAB IDE 主下拉菜单中选择的文件和窗口。

命令行接口 (Command Line Interface)

程序及其用户之间的一种完全基于文本输入和输出的通信手段。

模板 (Template)

建立的文本行，用于在以后插入到文件中。MPLAB 编辑器将模板存储在模板文件中。

目标 (Target)

指用户硬件。

目标板 (Target Board)

电路和组成目标应用的可编程器件。

目标处理器 (Target Processor)

目标应用板上的单片机器件。

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（可能还有调试信息）的文件。它可被立即执行或者也可能需要与其他目标文件（如库文件）链接重定位，以生成完整的可执行程序。

目标文件伪指令 (Object File Directives)

仅当创建目标文件时使用的伪指令。

目标应用程序 (Target Application)

驻留在目标板上的软件。

NOP

空操作。除了让程序计数器加 1 外，没有其他作用的指令。

内部链接 (Internal Linkage)

如果某个函数或变量不可以从定义它的模块外被访问，则称该函数或变量有内部链接。

OTP

一次性可编程器件。使用非窗口封装的 EPROM 器件。由于 EPROM 需要紫外线才能擦除其存储器，因此只有窗口器件才可被擦除。

PC

个人计算机或程序计数器。

PC 主机 (PC Host)

任何运行受支持的 Windows 操作系统的 IBM 和兼容的 PC 机。

PICmicro MCU

PICmicro 单片机 (MCU) 是指所有的 Microchip 单片机系列。

PICSTART Plus

Microchip 的开发器件编程器。可对 8、14、28 和 40 引脚的 PICmicro 单片机进行编程。必须和 MPLAB IDE 软件一起使用。

PRO MATE II

Microchip 的器件编程器。对大多数 PICmicro 单片机以及大多数存储器和 KeeLoq 器件进行编程。可以和 MPLAB IDE 一起使用或独立使用。

PWM 信号 (PWM Signals)

脉宽调制信号。某些 PICmicro MCU 器件具有 PWM 外设。

跑表 (Stopwatch)

测量执行周期的计数器。

配置位 (Configuration Bits)

可通过编程来设置 PICmicro 单片机工作模式的专用位。配置位可能预编程也可能不预编程。

片外存储器 (Off-Chip Memory)

片外存储器的使用由 PIC17 或 PIC18 器件的存储器选择选项指定，存储器可驻留在目标板上，也可由仿真器提供所有的程序存储器。

器件编程器 (Device Programmer)

用来对可电编程的半导体器件（如单片机）进行编程的工具。

嵌套深度 (Nesting Depth)

一个宏可以包含其他宏的最大级数。

情形 (Scenario)

对于 MPLAB SIM 软件模拟器来说，是一种用于激励控制的特定设置。

RAM

随机访问存储器（数据存储器）。可以任何顺序访问其中信息的存储器。

ROM

只读存储器（程序存储器）。不能被修改的存储器。

Run (运行)

将仿真器从暂停释放的命令，允许它运行应用代码并实时更改或响应 I/O。

软件断点 (Breakpoint, Software)

固件执行将停止的地址。通常由特殊的中断指令实现。

软件堆栈 (Stack, Software)

供应用程序用来存储返回地址、函数参数和局部变量的存储区。当用高级语言开发代码时，通常由编译器管理此存储区。

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 (Special Function Registers)。

Shell

MPASM 汇编器 shell 是宏汇编器的提示输入接口。有两个 MPASM 汇编器 shell 一个针对 DOS 版本，一个针对 Windows 版本。

Skew

不同时间出现在处理器总线上与指令执行相关的信息。例如，在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；当实际执行该操作码时，总线上出现源数据的地址和值以及目标数据的地址。而目标数据的值在下一条指令执行时出现在总线上。跟踪缓冲器在一个操作完成时捕获总线上的这些信息。因此，一个跟踪缓冲器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之间捕捉的周期数称为 Skew。

Skid

当使用硬件断点来暂停处理器时，在处理器暂停之前可能再执行一条或多条额外的指令。在预期断点之后执行的额外指令数称为 skid。

Step Into（跳入）

此命令与单步运行相同。Step Into（与 Step Over 相反）随着 CALL 指令进入子程序。

Step Out（跳出）

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

Step Over（跳过）

Step Over 允许跳过子程序。此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）

一种 EEPROM，其数据以块形式而不是以字节形式写入或擦除。

上电复位仿真（Power-on-Reset Emulation）

在数据 RAM 区域中写入随机值以模拟最初给应用上电时 RAM 中未初始化的值的软件随机化过程。

上载（Upload）

上载功能将数据从一个工具（如仿真器或编程器）传输到主机 PC 或者从目标板传输到仿真器。

实时（Real-Time）

在仿真器或 MPLAB ICD 模式下，当从暂停状态恢复时，处理器将在实时模式下运行并且完全像正常芯片那样工作。在实时模式下，MPLAB ICE 的实时跟踪缓冲器被使能并不断地捕获所有选定的周期，同时所有的中断逻辑也被使能。在仿真器或 MPLAB ICD 中，处理器将实时执行直到由于遇到有效的断点而暂停，或者直到用户暂停仿真器。对于软件模拟器，实时只不过意味着单片机指令的执行速度与主机 CPU 模拟这些指令的速度相同。

事件（Event）

对总线周期的描述，可能包括地址、数据、次数计数、外部输入、周期类型（取指或读 / 写）以及时间标记。事件可用来描述触发、断点和中断。

数据存储器（Data Memory）

在 Microchip MCU 和 DSC 器件上，数据存储器（RAM）由通用寄存器（General Purpose Register, GPR）和特殊功能寄存器（Special Function Register, SFR）组成。有些器件还具有 EEPROM 数据存储器。

数据伪指令（Data Directives）

数据伪指令是那些控制汇编器对程序和数据存储器进行分配的指示性语句，它提供了用符号（即有意义的名称）引用数据项的方法。

数字信号控制器（Digital Signal Controller）

具有数字信号处理功能的单片机器件，即 Microchip dsPIC 器件。

特殊功能寄存器（Special Function Registers）

为数据存储器（RAM）的一部分，由专用于控制 I/O 处理功能、I/O 状态、定时器或者其他模式或外设的寄存器组成。

Watch 窗口（Watch Window）

Watch 窗口包含在每个断点更新的观察变量的列表。

WDT

参见看门狗定时器（Watchdog Timer）。

外部 RAM（External RAM）

片外读 / 写存储器。

外部标号（External Label）

有外部链接的标号。

外部符号（External Symbol）

有外部链接的标识符符号。它可能是一个引用，也可能是一个定义。

外部符号解析（External Symbol Resolution）

由链接器执行的过程，在该过程中将收集来自所有输入模块的外部符号定义以尝试解析所有的外部符号引用。任何没有对应定义的外部符号引用将会导致报告发生链接器错误。

外部链接（External Linkage）

如果某个函数或变量可以从定义它的模块外被引用，则称该函数或变量有外部链接。

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测线（TRIGIN）。

微处理器模式（Microprocessor Mode）

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

伪指令（Directives）

源代码中提供对语言工具的操作进行控制的语句。

未初始化的数据（Uninitialized Data）

未定义初始值的数据。如在 C 中的：

```
int myVar;
```

定义了一个将驻留在未初始化数据段中的变量。

文件寄存器（File Register）

片上数据存储器，包括通用寄存器（GPR）和特殊功能寄存器（SFR）。

系统窗口控件 (System Window Control)

系统窗口控件位于窗口和某些对话框的左上角。单击该控件通常会弹出一个含有 **Minimize** (最小化)、**Maximize** (最大化) 和 **Close** (关闭) 项的菜单。

下载 (Download)

下载是将数据从主机传送到另一个器件 (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, 或者为在复杂触发中发生在另一个操作前的事件。

项目 (Project)

为应用程序编译目标和可执行代码的一组源文件和指令。

消息 (Message)

警告您语言工具操作中存在潜在问题的文本。消息并不会停止语言工具的操作。

异步激励 (Asynchronous Stimulus)

为模拟到被模拟器件的外部输入而生成的数据。

应用 (Application)

可以由 PICmicro 单片机控制的一组软件和硬件。

硬件断点 (Breakpoint, Hardware)

执行后将导致程序暂停的事件。

硬件堆栈 (Stack, Hardware)

当调用函数时, 在 PICmicro 单片机中存储返回地址的存储单元。

源代码 (Source Code)

由编程人员编写的计算机程序的形式。源代码以某种正规的编程语言编写。可被翻译成机器码或由解释程序执行。

原始数据 (Raw Data)

与段相关的代码或数据的二进制表示。

原型系统 (Prototype System)

指用户的目标应用或目标板的术语。

源文件 (Source File)

包含源代码的 ASCII 文本文件。

运算符 (Operators)

加号 “+” 和减号 “-” 之类的符号, 它们在构成定义明确的表达式时使用。每个运算符都有已分配的优先级, 用来决定求值的顺序。

暂停 (Halt)

停止程序执行。执行暂停与在断点停止相同。

指令 (Instructions)

告诉中央处理单元执行某个特定操作的位序列，这些位可以包含在该操作中将使用的数据。

指令集 (Instruction Set)

某个特定处理器能够理解的机器语言指令的集合。

中断 (Interrupt)

中断为一个到 CPU 的信号，该信号可暂停执行正在运行的应用程序并将控制转移给中断服务程序 (Interrupt Service Routine, ISR) 从而能对该事件进行处理。

中断处理程序 (Interrupt Handler)

发生中断时，处理特殊代码的程序。

中断服务程序 (Interrupt Service Routine)

用户产生的代码，当发生中断时进入该程序。程序存储器中代码的位置通常取决于已发生的中断的类型。

中断请求 (Interrupt Request)

使处理器暂时挂起正常的指令执行并开始执行中断处理程序的事件。有些处理器具备几个中断请求事件，以允许产生不同优先级的中断。

助记符 (Mnemonics)

可以直接翻译成机器码的文本指令。也称为操作码。

状态栏 (Status Bar)

状态栏位于 MPLAB IDE 窗口的底部，指示诸如光标位置、开发模式、器件和活动工具栏之类的当前信息。

字母数字 (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)

字母字符是指那些在阿拉伯字母表中的字母字符 (a、b、……、z, A、B、……、Z)。

注:

注:

索引**符号**

#define	68
#include	92, 143
#undefine	117
\$	43
.asm	12
.c	12
.hex	12
.lib	12
.lkr	12, 174
.o	12
__badram	48
__badrom	49
__config	58, 148
__fuses	58
__idlocs	87, 148
__maxram	97
__maxrom	98

数字

8 × 8 乘法	155
----------------	-----

A

access_ovr	48
ACCESSBANK	183, 185
ASCII 字符集	249
AUTOEXEC.BAT	37
按区寻址	149

B

badram	48
badrom	49
bankisel	50
banksel	52, 149
包含额外的源文件	92
包含文件	26
保留的段名, 汇编器	41
保留字, 汇编器	41
变量	

定义	109
局部	94
声明	118

标准的链接描述文件	181
表, 定义	70
表达式符号	42

C

cblock	54
code_pack	57
CODEPAGE	184, 185
COFF 目标模块文件	174
config	59
constant	60
常见问题	
链接器	223
常数	

定义	74
块	54, 72
声明	60
常数比较	156
常数块	54, 72
程序存储器	144
处理器, 设置	93, 106
存储器	
Fill	82
保留	107
存储器区域	183
存储区选择	52, 130
存储区选择, 间接	50
错误	
COFF	221
COFF 到 COD 转换器	222
汇编器	157
库管理器解析	233
库文件	220
链接器	215
链接器解析	213
错误文件	28, 157

D

da	61
data	62
DATABANK	183, 185
data 伪指令	39
db	65
de	67
define	68
dt	70
dw	70
dw 伪指令	39
代码	144, 148
绝对	144, 147, 150
可重定位	144, 147, 149
可重定位, 库程序	151
可重定位代码, 定义模块	148
可重定位代码, 引用模块	148
代码段	56, 129
代码段, 压缩	57
定义 RAM 存储器区域	183
定义 ROM 存储器区域	184
定义了符号则执行	90
堆栈	186

E

else	71
end	71
endc	72
endif	72
endm	73
endw	73
equ	74
error	74
exitm	78

汇编器 / 链接器 / 库管理器用户指南

extern	148
F	
fi	72
FILES	182
fill	82
访问来自其他模块的标号	148
非, 逻辑	43
分配	
堆栈	188
绝对	188
可重定位	188
分页	149
符号常数	60
符号和调试文件	30, 174
G	
global	84
H	
Hex 文件	12, 28, 174
格式	34
high	43, 145
宏	
代码示例	155
结束	73
使用	154
退出	78
文本替换	154
无扩展	100
展开	80
宏伪指令	47
endm	73
exitm	78
local	94
macro	96
noexpand	100
定义的	154
宏语法	153
宏语言	153
环境变量	192
换页符	103
或, 逻辑	43
I	
idata	85, 147
idata_acs	86
idlocs	87
ID 地址单元	87
if	88
else	71
ifndef	91
INCLUDE	183
include	92
Internet 地址	6
J	
基数	41
基数, 设置	93, 106, 107
寄存器分配	129
加 1	43
简单示例	49, 50
减 1	43
警告	
COFF 到 COD 转换	222
汇编器	163
链接器	220

K	
可执行文件	12
可重定位	30
可重定位目标	143
客户变更通知服务	6
客户支持	7
空白	24
空白列表行	110
控制伪指令	46
#define	68
#include	92
#undefine	117
constant	60
end	71
equ	74
org	100
processor	106
radix	106
set	109
variable	118
库文件	12, 174, 229
快速操作段	
被覆盖	48
扩展单片机模式	195
L	
LIBPATH	182
LKRPATH	182
local	94
low	43, 145
链接描述符	183
链接描述文件	12, 174, 181
链接描述文件, 标准	181
链接器处理过程	187
列表伪指令	47
error	74
messg	98
page	103
列表文件	26, 47, 174
逻辑代码段	185
M	
macro	96
maxram	97
maxrom	98
MCC_INCLUDE	192
messg	98
Microchip 网站	6
mpasm.exe	21
mpasmwin.exe	21
MPASM 汇编器概述	21
MPLAB C18	181
MPLAB IDE 项目	11
MPLIB 库管理器概述	227
MPLIB 目标库管理器	12
MPLINK 链接器概述	171
MPLINK 目标链接器	12
描述文件, 链接	181
命令 Shell 界面	33
命令行接口	
汇编器	35
库管理器	232
链接器	179
目标模块, 生成	150
目标文件	30, 34, 174, 229
目标文件伪指令	47

access_ovr	48	errorlevel	76
bankisel	50	exitm	78
banksel	52	extern	80
code_pack	57	fill	82
global	84	global	84
idata	85	idata	85, 87
idata_acs	86	if	89
pageselw	105	ifdef	90
udata_shr	116	ifndef	91
N		list	94
noexpand	100	local	94
O		macro	96
o	150	messg	99
org	100	org	101
P		pagesel	104, 105
page	103	processor	106
pagesel	149	radix	106
pageselw	105	res	108
PATH	192	set	109
processor	106	space	48, 110
Processor, 设置	125	subtitle	111
PROTECTED	183	title	111
配置位	58, 59	udata	112
R		udata_acs	113
radix	106	udata_ovr	115
Radix, 设置	125	udata_shr	116
RAM 分配	147	variable	118
res	107	while	120
S		示例, 应用程序	
SECTION	183, 185	#define	69, 132, 136
set	109	#include	124
SHAREBANK	183, 185	#undefine	69, 132
STACK SIZE	186	bankisel	51, 52
生成绝对代码	22	banksel	53, 54, 125, 134
生成可重定位代码	23	cblock	55
十六进制转换到十进制	250	code	56, 125
示例, 简单		constant	119, 132
#define	69	da	61
#include	93	data	63, 64
#undefine	118	db	65, 66
__badram	49	de	68
__badrom	50	else	89
__config	59	end	124
__idlocs	59	endc	55
__maxram	47	endif	89
__maxrom	50	endm	96, 134
bankisel	50	endw	120
banksel	52	equ	109, 125
cblock	55	error	75
code	56	errorlevel	76
code_pack	57	exitm	78
config	60	extern	81
data	63	fill	83
db	65	global	81, 136, 138
de	67	idata	85
dt	70	if	89
dw	71	ifdef	90, 91
else	71	list	107, 136
end	72	local	95
endm	73	macro	96, 134
equ	74	messg	99
error	75	org	101, 102
		pagesel	104, 125
		radix	107
		res	108, 125, 136, 138
		set	109, 132

汇编器 / 链接器 / 库管理器用户指南

udata	112, 125, 136, 138
udata_acs	113
udata_ovr	115
udata_shr	116
variable	119, 132
while	120
输入 / 输出文件	
汇编器	24
库管理器	229
链接器	173
数据	
字	70
数据段	
被覆盖的未初始化的	114
共享的未初始化的	116
未初始化的	111
未初始化快速操作的	113
已初始化的	85
已初始化快速操作的	86
数据伪指令	46
__badram	48
__badrom	49
__config	58
__fuses	58
__idlocs	87
__maxram	97
__maxrom	98
cblock	54
config	59
da	61
data	62
db	65
de	67
dt	70
dw	70
endc	72
fill	82
res	107
搜索顺序, 包含文件	92
算术运算符	42
T	
条件汇编伪指令	46
else	71
endif	72
endw	73
fi	72
if	88
ifndef	91
头文件	92, 128, 143
推荐读物	5
U	
udata	147
udata_acs	147
udata_ovr	147
udata_shr	116, 147
undefine	117
upper	43, 145
V	
Variable	
Declare	118
variable	118
W	
Watch 窗口	133

Windows Shell 界面	32
WWW 地址	6
外部标号	80
伪指令, 汇编器	45
伪指令, 链接器	182
未定义符号则执行	91
未用 RAM	48
未用 ROM	49
位分配	129
文本替换标号	68
文本字符串	39
文档	
编排	1
约定	4
文件	
错误	157
列表	47
X	
限制	
汇编器	168
消息	98
汇编器	166
Y	
样本应用程序, 链接器	191
页面选择	103, 129
页选择 — WREG	105
疑难解答	157
已初始化数据	190
映射文件	176
与, 逻辑	43
预编译的目标文件	12
源代码	12
源代码文件, 汇编	24
Z	
指令操作数	144
指令集	237
12 位 / 14 位内核	242
12 位内核	239, 242
14 位内核	240
PIC18 器件	243
中断处理	
PIC16 示例	77, 83, 101, 126, 131
PIC18 示例	83, 102
转义序列	40
最大 RAM 单元	97
最大 ROM 单元	98

注:

全球销售及服务中心

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

圣何塞 San Jose
Mountain View, CA
Tel: 1-650-215-1444
Fax: 1-650-961-0286

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

亚太地区

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

中国 - 成都
Tel: 86-28-8676-6200
Fax: 86-28-8676-6599

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: 86-29-8833-7252
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: 91-80-2229-0061
Fax: 91-80-2229-0062

印度 India - New Delhi
Tel: 91-11-5160-8631
Fax: 91-11-5160-8632

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: 604-646-8870
Fax: 604-646-5086

菲律宾 Philippines - Manila
Tel: 632-634-9065
Fax: 632-634-9069

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 - Weis
Tel: 43-7242-2244-399
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: 34-91-352-30-52
Fax: 34-91-352-11-47

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