MPLAB® XC8 PIC®汇编器用户指南

客户须知

本文档如同所有其他文档一样具有时效性。Microchip 将不断改进工具和文档以满足客户的需求，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（https://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“DSXXXXXXXXA_CN”，其中“XXXXXXXX”为文档编号，“A”为文档版本。

欲了解开发工具的最新信息，请参见 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。

目录

1. 前言

1.1 本指南使用的约定

本指南采用以下文档约定：

表 1-1. 文档约定

说明	表示	示例
Arial 字体：		
斜体字	参考书目	*MPLAB® IDE User's Guide*
	需强调的文字	...*仅有*的编译器...
首字母大写	窗口	Output 窗口
	对话框	Settings 对话框
	菜单选择	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	*File>Save*
粗体字	对话框按钮	单击 **OK**（确定）
	选项卡	单击 **Power** 选项卡
N'Rnnnn	verilog 格式的数字，其中 N 为总位数，R 为基数，n 为其中一位。	4'b0010，2'hF1
尖括号< >括起的文字	键盘上的按键	按下<Enter>，<F1>
Courier New 字体：		
常规 Courier New	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-`
	二进制位值	`0, 1`
	常量	`0xFF，‘A’`
斜体 Courier New	可变参数	*`file.o`*，其中 *`file`* 可以是任何有效的文件名
方括号[ ]	可选参数	`mcc18 [options] file [options]`
花括号和竖线：{ | }	选择互斥参数；“或”选择	`errorlevel {0|1}`
省略号...	代替重复文字	`var_name [, var_name...]`

	表示由用户提供的代码	`void main (void)`
		`{ ...`
		`}`

1.2 推荐读物

本用户指南介绍了 MPLAB XC8 PIC 汇编器的用法和功能。以下 Microchip 文档均已提供，并建议读者作为补充参考资料。

MPLAB® XC8 PIC®汇编器移植指南

本指南适用于拥有 MPASM 项目并希望将其移植到 MPLAB XC8 PIC 汇编器的客户。其中描述了与 MPASM 代码最接近的等效汇编器语法和伪指令。

MPLAB® XC8 PIC® Assembler Guide For Embedded Engineers

本入门指南介绍了 MPLAB XC8 PIC 汇编器示例项目和常用的编码序列。如果需要使用此汇编器开发新项目，请使用本指南。

适用于 PIC® MCU 的 MPLAB® XC8 C 编译器发行说明

有关此汇编器的更改和缺陷修复的最新信息，请阅读 MPLAB XC8 安装目录的 docs 子目录下的自述文件。

开发工具发行说明

有关使用其他开发工具的最新信息，请阅读与工具相关的自述文件，文件位于 MPLAB X IDE 安装目录的 docs 子目录下。

1.3 文档版本历史

版本 A（2020 年 3 月）

- 本文档的初始版本，基于 *MPLAB® XC8 C Compiler User's Guide*（DS50002737）的汇编器章节。

2. 汇编器概述

MPLAB XC8 PIC 汇编器是一款独立的交叉汇编器和链接器程序包，支持所有 8 位 PIC®单片机。

2.1 器件说明

本指南介绍了 MPLAB XC8 PIC 汇编器对采用低档、中档、增强型中档和 PIC18 内核的所有 8 位 Microchip PIC 器件的支持。以下描述说明了这些器件内核的区别：

低档内核使用 12 位宽的指令集，PIC10 以及部分 PIC12 和 PIC16 部件编号的器件采用低档内核。

增强型低档内核也使用 12 位指令集，但此指令集还包含其他指令。一些增强型低档芯片支持中断以及中断所使用的其他指令。采用增强型低档内核的器件的部件编号为 PIC12 和 PIC16。

中档内核使用 14 位宽的指令集，并包含比低档内核更多的指令。它还具有更大的数据存储器存储区和程序存储器页。PIC14 以及部分 PIC12 和 PIC16 部件编号的器件采用中档内核。

增强型中档内核也使用 14 位宽的指令集，但包含了更多的指令和功能。部分 PIC12 和 PIC16 部件编号的器件基于增强型中档内核。

PIC18 内核指令集为 16 位宽，具有更多的指令和扩展的寄存器集。PIC18 内核器件的部件编号以 PIC18 开头。

关于汇编器支持的器件的完整列表的信息，请参见 3.3.22 Print-devices。

2.2 兼容的开发工具

汇编器与许多其他 Microchip 工具配合使用，包括：

- MPLAB X IDE（www.microchip.com/mplab/mplab-x-ide）
- MPLAB X 软件模拟器
- 命令行 MDB 软件模拟器——请参见《Microchip 调试器（MDB）用户指南》（DS50002102E_CN）
- 所有 Microchip 调试工具和编程器（www.microchip.com/mplab/development-boards-and-tools）
- 支持 8 位器件的演示板和入门工具包

3. 汇编器驱动程序

MPLAB XC8 PIC 汇编器使用的命令行驱动程序的名称为 `pic-as`。

调用此驱动程序可执行汇编和链接步骤，开发环境（例如 MPLAB X IDE）编译汇编项目时也调用该驱动程序。

`pic-as` 驱动程序具有以下基本命令格式：

```
pic-as [options] files [libraries]
```

在本手册中，将假定汇编器应用程序处于控制台的搜索路径中，或者在执行应用程序时指定了完整路径。

习惯上在文件名之前声明 options（通过一个前导短划线“`-`”或双短划线“`--`”标识）；但是，这不是强制性的。

3.3 汇编器选项说明给出了 *`options`* 的格式，以及选项功能的相应说明。

`files` 可以是各种各样的汇编器源文件，以及预编译的中间文件。虽然这些文件的列出顺序并不重要，但它可能影响代码或数据的分配，并且可能影响一些输出文件的名称。

`libraries` 是用户定义的库文件的列表，编译器会在其中进行搜索。这些文件的顺序将决定对它们进行搜索的顺序。习惯上将 libraries 插入到源文件列表之后；但是，这不是强制性的。

3.1 单步汇编

使用 `pic-as` 驱动程序，只需一步即可完成一个或多个源文件的汇编。

以下命令将编译两个汇编源文件，并将这两个文件传递给相应的内部应用程序，然后链接生成的代码以形成最终的输出。

```
pic-as -mcpu=16F877A main.S mdef.s
```

驱动程序将编译所有源文件，无论它们自上次编译以来是否发生了更改。要实现增量编译，必须采用开发环境（如 MPLAB® X IDE）和 make 实用程序（见 3.2 多步汇编）。

除非另外指定，否则将生成 HEX 文件和 ELF 文件作为最终的输出。中间文件在编译完成之后会保留，但大多数其他临时文件会被删除，除非使用`-save-temps` 选项（见 3.3.26 Save-temps 选项），该选项会保留所有生成的文件。请注意，某些生成的文件可能位于不同于项目源文件的其他目录中（另请参见 3.3.21 O：指定输出文件）。

3.2 多步汇编

多步汇编方法可用于实现项目的增量编译。Make 实用程序会注意哪些源文件自上次编译以来发生了更改，然后仅重新编译这些文件，以此加快汇编速度。在 MPLAB X IDE 中，您可以选择增量编译（Build Project（编译项目）图标），或完全重新编译项目（Clean and Build Project（清除并编译项目）图标）。

Make 实用程序通常会多次调用汇编器：一次是为每个源文件生成一个中间文件，一次是执行链接步骤。

选项`-c` 用于创建中间文件。该选项在汇编器应用程序执行完毕后停止，得到的输出目标文件的扩展名将为`.o`。

随后将在编译的第二阶段将中间文件指定给驱动程序，从而将中间文件传递给链接器。

以下命令行的前两行为每个汇编源文件生成一个中间文件，然后在最后一条命令中将这些中间文件再次传递给驱动程序以进行链接。

```
pic-as -mcpu=16F877A -c main.s
pic-as -mcpu=16F877A -c io.s
pic-as -mcpu=16F877A main.o io.o
```

对于任何汇编器，执行编译的第二（链接）阶段时，构成项目的所有文件都必须存在。

此外，您可能希望生成中间文件来构造您自己的库文件。 关于库创建的更多信息，请参见 7.1 归档器/库管理器。

3.3 汇编器选项说明

使用传递给汇编器命令行驱动程序 pic-as 的选项可以对编译过程的大多数方面进行控制。

所有选项均区分大小写，并通过一个前导短划线或双短划线字符进行标识，例如-o 或--version。

使用--help 选项可获取关于命令行可接受选项的简要说明。

如果在 MPLAB X IDE 中进行编译，则它会根据项目的 **Project Properties**（项目属性）对话框中的选择向汇编器发出显式的选项。默认项目选项可能与在命令行上运行时汇编器使用的默认选项有所不同，因此应检查 IDE 属性以确保默认项目选项是可接受的。

下表汇总了所有可用的驱动程序选项及其详细说明。

表 3-1. PIC®汇编器驱动程序选项

选项	控制
-###	显示应用程序命令行，但不执行
-c	生成中间目标文件
-mcallgraph=*type*	映射文件中打印的调用图的类型
-mchecksum=*specs*	生成和放置校验和或哈希值
-mcpu=*device*	目标器件，将为其编译代码
-D	定义预处理器符号
-mdfp=*path*	要使用的器件系列包
-dM	列出所有已定义的宏
-m[no-]download	最终 HEX 文件的调整方式
-E	生成仅预处理的文件
--fill=*options*	填充未使用的存储器
-gformat	生成的调试信息的类型
-H	列出包含的头文件
--help	仅显示帮助信息
-I	头文件搜索目录
-misa	选择 PIC18 指令集
-l	扫描的库
-L	库搜索目录
-fmax-errors	中止前的错误数
-mmaxichip	使用具有所选器件系列允许的最大存储器资源的假想器件
-o	指定输出文件名称
-mprint-devices	仅芯片信息
-mram=*ranges*	可供程序使用的数据存储器
-mreserve=*ranges*	应保留的存储器
-mrom=*ranges*	可供程序使用的程序存储器
-save-temps	编译后是否应保留中间文件
-mserial=*options*	在输出中插入序列号

.....（续）

选项	控制
-msummary=*types*	产生的存储器汇总信息
-U	取消定义预处理器符号
-v	详细编译输出
--version	显示汇编器版本信息
-w	禁止所有警告消息
-mwarn=*level*	警告输出阈值
-Wa,*option*	传递给汇编器的选项
-Wp,*option*	传递给预处理器的选项
-Wl,*option*	传递给链接器的选项
-x*language*	源文件解析语言
-Xassembler *option*	传递给汇编器的选项
-Xpreprocessor *option*	传递给预处理器的选项
-Xlinker *option*	传递给链接器的系统特定的选项

3.3.1 ###选项

-###选项类似于-v，但不执行命令。使用该选项，无需执行汇编器即可查看汇编器的命令行。

3.3.2 C 选项：编译为中间文件

-c 选项用于为命令行上列出的每个源文件生成一个中间文件。

该选项通常用于通过 make 实用程序实现多步编译。

3.3.3 调用图选项

-mcallgraph=*type* 选项控制映射文件中打印的调用图的类型。下表列出了可用的类型。

表 3-2. 调用图类型

类型	产生的结果
none	无调用图
crit	仅调用图中的关键路径
std	标准的简短调用图（默认）
full	完整调用图

调用图是由链接器生成的，主要用于在已编译堆栈中为对象分配存储空间。某些程序定义的堆栈对象与其他堆栈对象不重叠，因此会增加程序使用的数据存储空间，这些程序被视为处于关键路径上。

3.3.4 校验和选项

-mchecksum=*specs* 选项将计算指定地址范围内的哈希值（例如校验和或 CRC），并将结果存储到十六进制文件中的指定目标地址。该选项的一般形式如下。

```
-mchecksum=start-end@destination[,specifications]
```

以下规范以逗号分隔列表的形式附加到该选项后面。

表 3-3. 校验和参数

参数	说明
`width=n`	对于非 Fletcher 算法，以字节为单位选择哈希结果的宽度；对于 SHA 算法，以位为单位选择哈希结果的宽度。宽度为负值时，将以小尾数法字节顺序存储结果；宽度为正值时，将以大尾数法字节顺序存储结果。允许选择一到四个字节的结果宽度，对于 SHA 算法，则为 256 位。
`offset=nnnn`	指定要与校验和相加的初始值或偏移量。
`algorithm=n`	选择 Hexmate 中实现的哈希算法之一。表 7-3 列出了可供选择的算法。
`polynomial=nn`	选择在使用 CRC 算法时使用的多项式值
`code=nn`	指定结果中尾随在每个字节之后的十六进制代码。这样便可将结果的每个字节嵌入指令中。例如，在中档器件上，`code=34` 会将结果嵌入到 `retlw` 指令中。
`revword=n`	从 *n* 字节宽的字中以相反的字节顺序读取数据。当前，该值可以为 0 或 2。值 0 用于禁止反向读取功能。

默认情况下，*start*、*end* 和 *destination* 属性是十六进制常量。定义输入范围的地址通常是算法宽度的倍数。如果不是这种情况，则任何缺失的输入字存储单元都将用零字节填充。

如果未指定伴随的`--fill` 选项（3.3.11 填充选项），则指定地址范围内的未用存储单元将使用 0xFFF（对于低档器件）、0x3FFF（对于中档器件）或 0xFFFF（对于 PIC18 器件）自动填充。这是为了从计算中去除所有未知值，并确保结果的准确性。

例如：

```
-mchecksum=800-fff@20,width=1,algorithm=2
```

将对 0x800 至 0xfff 地址范围计算 1 个字节的校验和，并将它存储在地址 0x20 处。将使用 16 位加法算法。表 3-9 列出了可用的算法，7.2.2 哈希函数对这些算法进行了详细说明。

表 3-4. 校验和算法选择

选择器	算法说明
-5	反射循环冗余校验（Cyclic Redundancy Check，CRC）
-4	初始值减去 32 位值
-3	初始值减去 24 位值
-2	初始值减去 16 位值
-1	初始值减去 8 位值
1	初始值加上 8 位值
2	初始值加上 16 位值
3	初始值加上 24 位值
4	初始值加上 32 位值
5	循环冗余校验（CRC）
7	Fletcher 校验和（8 位计算，2 字节结果宽度）
8	Fletcher 校验和（16 位计算，4 字节结果宽度）
10	SHA-2（当前仅支持 SHA256）

哈希计算由 Hexmate 应用程序执行。该驱动程序选项中的信息将在执行 Hexmate 应用程序时传递给它。

3.3.5 Cpu 选项

在编译时必须使用-mcpu=*device* 选项指定目标器件。它是惟一的必需选项。

例如，-mcpu=18f6722 将选择 PIC18F6722 器件。要查看可以使用该选项的支持器件的列表，请使用-mprint-devices 选项（3.3.22 Print-devices）。

3.3.6 D：定义宏

-D*macro*=*text* 选项用于定义预处理器宏。要对宏进行后续处理，必须对源文件进行预处理，具体方法是使用.S 文件扩展名或使用-xassembler-wth-cpp 选项。

该选项和宏名称之间可存在空格。

如果选项中未指定=*text*，则该选项定义一个称为 *macro* 的预处理器宏，该宏将被视为已由检查此类定义的任何预处理器伪指令（例如#ifdef 伪指令）定义，并且将在该宏将被替换的上下文中使用时扩展为值 1。例如，当使用选项-DMY_MACRO（或-D MY_MACRO）并提供以下代码时：

```
#ifdef MY_MACRO
movlw MY_MACRO;
#endif
```

将汇编 movlw 指令，并将值 1 赋值给 W 寄存器。

当在该选项中指定替换项 *text* 时，宏将在随后扩展为通过该选项指定的替换项。 因此，如果使用选项-DMY_MACRO=0x55 编译以上示例代码，则指令将汇编为：movlw 0x55

在任何-U 选项之前处理命令行中的所有-D 实例。

3.3.7 dM：预处理器调试转储选项

-dM 选项可使预处理器生成的宏定义在预处理结束时生效。该选项应与-E 选项一起使用，如果要将输出定向到文件，应同时使用-o 选项。

3.3.8 Dfp 选项

-mdfp=*path* 选项指示应从某个器件系列包（Device Family Pack，DFP）的内容中获取对目标器件（由-mcpu 选项指示）的器件支持，其中 *path* 是该 DFP 的 xc8 子目录的路径。

如果未使用该选项，则 pic-as 驱动程序将尽可能使用汇编器发行版中提供的器件特定文件。

MPLAB X IDE 自动使用该选项通知汇编器要使用的器件特定信息。如果已为汇编器获取了其他 DFP，则在命令行上使用该选项。

DFP 可能包含器件特定的头文件、配置位数据和库等，从而不必更新汇编器即可使用新器件上的功能。DFP 始终不包含可执行文件，也不提供针对任何现有工具或标准库函数的缺陷修复或改进。

使用该选项时，预处理器将首先在<*DFP*>/xc8/pic/include/proc 和<*DFP*>/xc8/pic/include 目录中搜索包含文件，然后搜索标准搜索目录。

3.3.9 下载选项

-mdownload 选项可调整 Intel HEX 文件，以供自举程序使用。-mdownload-hex 选项与之等效。

使用该选项时，会将 Intel HEX 文件中的数据记录填充为 16 字节的长度，使其按 16 字节边界对齐。

默认操作是不修改 HEX 文件，具体可使用选项-mno-download （-mno-download-hex）显式指定。

3.3.10 E：仅预处理

-E 选项用于生成预处理的汇编源文件（也称为模块或翻译单元）。

使用该选项时，编译序列将在预处理阶段结束后终止，并留下基本名称与相应源文件相同且扩展名为.i 的文件。

您可以检查预处理的源文件，以确保预处理器宏已扩展为您希望的内容。该选项还可用于创建不需要任何单独头文件的汇编源文件。需要将文件发送给同事，或者需要获取技术支持时，这非常有用，因为可以不必发送所有头文件（可能驻留在多个目录中）。

3.3.11 填充选项

`--fill=`*`options`* 选项用于以多种方式向未使用的存储器中填充指定的值。

该选项的功能与 Hexmate 的`-fill` 选项相同。关于更多详细信息和可以用于该选项的高级控制，请参见 7.2.1.11 Fill。

3.3.12 G：生成调试信息选项

`-g`*`format`* 选项指示汇编器生成附加信息，硬件工具可以使用这些信息来调试程序。

下表列出了支持的格式。

表 3-5. 支持的调试文件格式

格式	调试文件格式
`-gcoff`	COFF
`-gdwarf-3`	ELF/Dwarf 发行版 3
`-ginhx32`	具有扩展线性地址记录的 Intel HEX，允许使用超过 64 KB 的地址
`-ginhx032`	INHX32，并将高位地址初始化为零

默认情况下，汇编器生成 Dwarf 发行版 3 文件。

3.3.13 H：打印头文件选项

除了一些常规操作之外，`-H` 选项还可将使用的每个头文件的名称打印到控制台。

3.3.14 帮助

`--help` 选项显示有关 `pic-as` 汇编器选项的信息，随后驱动程序将终止。

3.3.15 I：指定包含文件搜索路径选项

`-I`*`dir`* 选项将目录 *`dir`* 添加到供搜索头文件的目录列表的开头。该选项和目录名称之间可能存在空格。

该选项可以指定绝对路径或相对路径，如果要搜索多个其他目录，则可以多次使用该选项，这种情况下将从左到右进行扫描。对该选项指定的目录完成扫描后，将搜索标准系统目录。

在 Windows 操作系统下，使用目录反斜杠字符可能会无意间形成转义序列。如果要指定的包含文件路径以目录分隔符结尾并加引号，请使用`-I "E:\\"`形式（而非`-I "E:\"`），以避免形成转义序列`\"`。请注意，MPLAB X IDE 将引用您在项目属性中指定的任何包含文件路径，并且搜索路径是相对于输出目录（而不是项目目录）而言。

3.3.16 Isa 选项

`-misa=`*`set`* 选项用于为 PIC18 目标器件指定指令集。默认选择为标准 PIC18 指令集，具体可通过使用`-misa=std` 显式指定。或者，也可以使用`-misa=xinst` 来选择 PIC18 扩展指令集。当使用的器件不支持所请求的指令集时，将忽略该选项并发出警告。

请注意，该选项将允许汇编器检查汇编程序是否与所选指令集相符，但是它不指示器件使用所选指令集进行操作。使用 `XINST` 配置位在您的器件中使能该功能。

3.3.17 L：指定库文件选项

`-l`*`library`* 选项在链接时查找指定的文件（扩展名为`.a`），并在此库归档文件中扫描未解析的符号。

使用`-l` 选项（例如`-lmylib`）和在命令行上指定文件名（例如 `mylib.a`）之间的惟一区别是，汇编器将在`-L` 选项所指定的多个目录中搜索使用`-l` 指定的库。

3.3.18 L：指定库搜索路径选项

`-L`*`dir`* 选项用于为已使用`-l` 选项指定的库文件指定一个供搜索的其他目录。汇编器将自动搜索标准库位置，因此仅在链接自己的库时才需要使用该选项。

3.3.19 最大错误数

`-fmax-errors=n` 选项用于设置每个汇编器应用程序以及驱动程序在终止执行之前显示的最大错误数。

默认情况下，每个应用程序在汇编器终止前最多显示 20 条错误消息。例如，选项`-fmax-errors=10` 可以确保应用程序在仅显示 10 条错误后终止。

3.3.20 Maxichip 选项

`-mmaxichip` 选项用于告知汇编器为一个假想器件进行编译，该假想器件的物理内核和外设与所选器件相同，但具有所选器件系列允许的最大存储器资源。当您的程序所需的程序和数据存储空间大小超出所选目标器件上的存储空间时，如果您希望知道需编程到目标器件中的代码或数据需要减少多少，则可以使用该选项。

如果所选器件的程序存储器、数据存储器或 EEPROM 用尽，汇编器通常将终止。使用该选项时，PIC18 和中档器件的程序存储器将最大化，即从地址 0 到外部存储器的底部或 PC 寄存器允许的最大地址（以地址较低者为准）。低档器件的程序存储器也将最大化，即从地址 0 到配置字的最低地址。

数据存储器中存储区的数量扩展为由 BSR 寄存器（对于 PIC18 器件）、STATUS 寄存器中的 RP 位（对于中档器件）或 FSR 寄存器中的存储区选择位（对于低档器件）定义的最大可选存储区数量。每个附加存储区中的 RAM 大小等于物理实现的存储区中最大连续存储区域的大小。

如果器件上存在 EEPROM，则 EEPROM 将视情况最大化为 EEADR 或 NVMADR 寄存器中的位数所指示的大小。

如果需要，请检查映射文件（见 6.3 映射文件）以查看在器件上使用该选项时可用存储器的大小和布局。

注： 使用`-mmaxichip` 选项时，并非为实际器件进行编译。生成的代码可能无法在软件模拟器或所选器件中加载或执行。该选项不允许将额外的代码置于器件中。

3.3.21 O：指定输出文件

`-o` 选项指定输出文件的基本名称和目录。

例如，选项`-o main.elf` 会将生成的输出置于名为 `main.elf` 的文件中。可随文件名指定现有目录的名称，例如`-o build/main.elf`，这样输出文件便会出现在该目录中。

不能使用该选项来更改输出文件的类型（格式）。

只有使用该选项指定的文件基本名称才有意义。不能使用该选项来更改输出文件的扩展名。

3.3.22 Print-devices

`-mprint-devices` 选项将显示汇编器支持的器件的列表。

列出的名称是可以与`-mcpu` 选项一起使用的器件。该选项将仅显示汇编器发布时官方支持的器件。可通过器件系列包（DFP）支持的其他器件不会显示在该列表中。

打印器件列表之后，汇编器将会终止。

3.3.23 Ram 选项

`-mram=`*`ranges`* 选项用于调整为目标器件指定的数据存储器。不使用该选项时，器件上实现的所有片上 RAM 均可用，因此只有存在特殊的存储器要求时，才需要使用该选项。指定不处于目标器件中的额外存储器可以成功编译，但可能会导致代码运行时失败。

例如，要在片上已存在的存储器之外指定额外的存储器范围，可以使用：

```
-mram=default,+100-1ff
```

这会将 100h 至 1ffh 的范围添加到片上存储器中。要仅使用外部范围并忽略所有片上存储器，可以使用：

```
-mram=0-ff
```

该选项还可用于保留已在相关 chipinfo 文件中定义为片上存储器的存储器范围。要实现这一点，需要提供使用减号字符`-`作为前缀的范围，例如：

```
-mram=default,-100-103
```

将使用所有已定义的片上存储器，但不使用范围 100h 至 103h 中的地址来分配 RAM 对象。

该选项将调整链接器类使用的存储器范围（见 6.1.1 A：定义链接器类）。 对于 psect 中包含的任何对象，如果不使用受该选项影响的类，可能链接到该选项指定的有效存储器之外。

3.3.24 保留选项

`-mreserve=`*`ranges`* 选项用于保留程序通常使用的存储器。该选项的一般形式为：

```
-mreserve=space@start:end
```

其中，*space* 可以是 `ram` 或 `rom`，分别表示数据和程序存储空间；*start* 和 *end* 是地址，表示要排除的范围。例如，`-mreserve=ram@0x100:0x101` 将保留数据存储器中从地址 100h 开始的两个字节。

该选项执行的任务与`-mram` 和`-mrom` 选项类似，但是不能用于向程序可用的存储器中添加额外的存储空间。

3.3.25 Rom 选项

`-mrom=`*`ranges`* 选项用于更改为目标器件指定的默认程序存储器。不使用该选项时，器件上实现的所有片上程序存储器均可用，因此只有存在特殊的存储器要求时，才需要使用该选项。指定不处于目标器件中的额外存储器可以成功编译，但可能会导致代码运行时失败。

例如，要在片上存储器之外指定额外的存储器范围，可以使用：

```
-mrom=default,+100-2ff
```

这会将 100h 至 2ffh 的范围添加到片上存储器中。要仅使用外部范围并忽略所有片上存储器，可以使用：

```
-mrom=100-2ff
```

该选项还可用于保留已在芯片配置文件中定义为片上存储器的存储器范围。要实现这一点，需要提供使用减号字符`-`作为前缀的范围，例如：

```
-mrom=default,-100-1ff
```

将使用所有已定义的片上存储器，但不使用范围 100h 至 1ffh 中的地址来分配 ROM 对象。

该选项将调整链接器类使用的存储器范围（见 6.1.1 A：定义链接器类）。 对于 psect 中包含的任何代码或对象，如果不使用受该选项影响的类，可能链接到该选项指定的有效存储器之外。

请注意，一些 psect 必须链接到某个阈值地址之上，最明显的就是一些存放 `const` 限定数据的 psect。使用该选项去除高存储器地址范围可能会使得无法放置这些 psect。

3.3.26 Save-temps 选项

`-save-temps` 选项指示汇编器在编译完成后保留临时文件。

中间文件将放置在当前目录中，并具有一个基于相应源文件的名称。因此，使用`-save-temps` 编译 `foo.s` 将产生 `foo.o` 目标文件。

`-save-temps=cwd` 选项与`-save-temps` 等效。

该选项的`-save-temps=obj` 形式类似于`-save-temps`，但是如果指定了`-o` 选项，则这些临时文件将与输出目标文件置于同一目录中。如果未指定`-o` 选项，则`-save-temps=obj` 开关的行为类似于 `-save-temps`。

3.3.27 序列号选项

`-mserial=`*`options`* 选项用于将一个十六进制代码存储在程序存储器中的特定地址处。该选项的典型任务是将某个序列号存储到程序存储器中。

要存储的数据的字节宽度由选项中十六进制代码参数的字节宽度决定。例如，要在程序存储器地址 1000h 存储 1 字节值 0，可以使用`-mserial=00@1000`。要将同一个值存储为一个 4 字节的量，可以使用`-mserial=00000000@1000`。

该选项的功能与 Hexmate 的-serial 选项相同。关于更多详细信息和可以用于该选项的高级控制，请参见 7.2.1.20 序列。

3.3.28 摘要选项

-msummary=*type* 选项用于选择编译完成后显示的摘要中包含的信息类型。默认情况下，或者选择了 mem 类型时，将会显示关于所有存储空间总使用量的存储器摘要。

通过使能 psect 类型，可以显示 psect 摘要。 它会显示各个 psect（由链接器对它们进行分组之后），以及它们涵盖的存储器范围。表 4-20 列出了可用的摘要类型。打印的默认输出与 mem 设置相对应。

此外，也可以使用该选项显示生成的十六进制文件的 SHA 哈希值。这些值可用于快速确定十六进制文件中的内容与先前版本相比是否发生了任何更改。

表 3-6. 汇总类型

类型	显示
psect	将显示 psect 名称及其链接地址的摘要。
mem	将显示已用存储器的简明摘要（默认）。
class	将显示每个存储空间中所有类的摘要。
十六进制	将显示形成最终输出文件的地址和 HEX 文件的摘要。
file	摘要信息将在屏幕上显示，并保存到文件中。
sha1	十六进制文件的 SHA1 哈希值。
sha256	十六进制文件的 SHA256 哈希值。
xml	摘要信息将显示在屏幕上，主存储空间的使用信息将保存在 XML 文件中。
xmlfull	摘要信息将显示在屏幕上，所有存储空间的使用信息将保存在 XML 文件中。

通过指定，可以使 XML 文件中包含有关所选器件上的存储空间的信息，包括空间的名称、可寻址单元、大小、已用空间大小和可用空间大小。

3.3.29 U：取消定义宏

-U*macro* 选项用于取消定义宏 macro。

该选项将取消定义使用-D 定义的任何宏。所有-U 选项在所有-D 选项之后评估。

3.3.30 V：详细编译

-v 选项用于指定详细编译。

使用该选项时，将在执行内部汇编器应用程序时显示其名称和路径，然后显示传递给每个应用程序的命令行参数。

使用该选项，可以确认您的驱动程序选项是否已按预期进行处理，或者查看哪个内部应用程序正在发出警告或错误。

3.3.31 版本

--version 选项将打印汇编器版本信息，然后退出。

3.3.32 W：禁止所有警告选项

-w 选项用于禁止所有警告消息，因此应谨慎使用。

3.3.33 Wa：将 Option 传递给汇编器选项

-Wa,*option* 选项用于将其 option 参数直接传递给汇编器。如果 option 包含逗号，则表示以逗号分隔的多个选项。例如，-Wa,-a 将请求汇编器生成一个汇编列表文件。

3.3.34 警告选项

-mwarn=*level* 选项用于设置警告级别阈值。允许的警告级别范围为-9 至 9。警告级别决定汇编器对于可疑类型转换和构造的严格程度。每条警告都具有指定的警告级别；警告级别越高，警告消息就越重要。如果警告消息的警告级别超出所设置的阈值，将会打印警告。默认的警告级别阈值为 0，即允许所有正常的警告消息。

请小心使用该选项，因为一些警告消息指示代码很可能会在执行期间发生失败，或损害可移植性。

3.3.35 Wp：将 Option 传递给预处理器选项

-Wp,*option* 选项将 *option* 传递给预处理器，在此它将被解析为预处理器选项。如果 option 包含逗号，则表示以逗号分隔的多个选项。

3.3.36 WL：将 Option 传递给链接器选项

-Wl,*option* 选项将 *option* 传递给链接器，在此它将被解析为链接器选项。如果 *option* 包含逗号，则表示以逗号分隔的多个选项。

3.3.37 X：指定源语言选项

-x*language* 选项用于指定随后的源文件是以指定的语言编写的，与这些文件使用的扩展名无关。

下表列出了 PIC 汇编器允许的语言。

表 3-7. 语言选项

语言	说明
assembler	汇编源代码
assembler-with-cpp	必须预处理的汇编源代码

例如，以下命令：

```
pic-as -mcpu=18f4520 -xassembler-with-cpp init.s
```

将告知汇编器对汇编源文件运行预处理器，即使 init.s 文件名不使用扩展名.S 亦如此。

3.3.38 Xassembler 选项

-Xassembler *option* 选项将 *option* 传递给汇编器，在此它将被解析为汇编器选项。该选项可用于提供汇编器不知道如何识别或-Wa 选项无法解析的系统特定的汇编器选项。

3.3.39 Xpreprocessor 选项

-Xpreprocessor *option* 选项将 *option* 传递给预处理器，在此它将被解析为预处理器选项。该选项可用于提供汇编器不知道如何识别的系统特定的预处理器选项。

3.3.40 Xlinker 选项

-Xlinker *option* 选项将 *option* 传递给链接器，在此它将被解析为链接器选项。该选项可用于提供汇编器不知道如何识别的系统特定的链接器选项。

4. MPLAB XC8 汇编语言

本章将介绍宏汇编器所接受的源语言的信息。

所有操作码助记符和操作数语法均特定于目标器件，应查阅器件数据手册。本章还会介绍附加助记符、与指令集的差异和汇编器伪指令。

4.1 汇编指令差异

MPLAB XC8 汇编器可使用相对于 Microchip 数据手册所指定的汇编语言稍作修改的汇编语言形式。这种形式通常更容易读取，但也可以使用数据手册中指定的形式。以下信息将详细介绍允许的指令格式差异，以及除器件指令集外还可使用的伪指令。

4.1.1 目标和访问操作数

要指定面向字节的文件寄存器指令的目标，可以使用表 4-1 中列出的任一样式的操作数。wreg 目标指示指令结果将被写入 W 寄存器，而文件寄存器目标指示指令结果将被写入指令的文件寄存器操作数所指定的寄存器。该操作数在器件数据手册中通常用,d 表示。

表 4-1. 目标操作数样式

样式	Wreg 目标	文件寄存器目标
XC8	,w	,f
MPASM	,0	,1

例如（本例忽略存储区选择和地址掩码）：

```
addwf   foo,w        ;add wreg to foo, leaving the result in wreg
addwf   foo,f        ;add wreg to foo, updating the content of foo
addwf   foo,0        ;add wreg to foo, leaving the result in wreg
addwf   foo,1        ;add wreg to foo, updating the content of foo
```

强烈建议始终根据需要使用这些指令来指定目标操作数。如果省略目标操作数，则假定目标为文件寄存器。

要指定 PIC18 器件的 RAM 访问位，可以使用表 4-2 中列出的任一样式的操作数。分区访问指示指令中指定的文件寄存器地址只是当前所选存储区内的偏移量。 非分区访问指示文件寄存器地址是快速操作存储区或公共存储区内的偏移量。

表 4-2. RAM 访问操作数样式

样式	分区访问	非分区访问
XC8	,b	,c 或,a
MPASM	,1	,0

该操作数在器件数据手册中通常用,a 表示。

或者，也可以通过在指令操作数前面加上字符“c:”来指示该地址处于快速操作存储区中。例如：

```
addwf   bar,f,c         ;add wreg to bar in common memory
addwf   bar,f,a         ;add wreg to bar in common memory
addwf   bar,1,0         ;add wreg to bar in common memory
addwf   bar,f,b         ;add wreg to bar in banked memory
addwf   bar,1,1         ;add wreg to bar in banked memory
btfsc   c:bar,3            ;test bit three in the common memory symbol bar
```

建议始终指定 RAM 访问操作数或公共存储区前缀。如果不指定，则指令地址为绝对地址，并且该地址位于快速操作存储区的上半部分（这表明该地址不得掩码）内，指令将使用快速操作存储区 RAM。在所有其他情况下，指令将访问分区存储器。

如果使用 XC8 样式，则 PIC18 指令的目标操作数和 RAM 访问操作数可以任何顺序列出。例如，以下两条指令完全相同：

```
addwf   foo,f,c
addwf   foo,c,f
```

对于每条指令而言，使用的操作数样式应始终保持一致，最好在整个程序中都保持一致。例如，指令 `addwf bar,1,c` 是非法的。

例如，以下指令显示了先将 W 寄存器内容传送到某个绝对地址，然后再传送到由某个标识符表示的地址。 本例中使用了存储区选择和掩码。下面显示了这些指令的 PIC18 操作码（假定赋给 `foo` 的地址为 0x516，赋给 `bar` 的地址为 0x55）。

```
6EE5  movwf 0FE5h             ;write to access bank location 0xFE5
6E55  movwf bar,c             ;write to access bank location 0x55
0105  BANKSEL(foo)            ;set up BSR to access foo
6F16  movwf BANKMASK(foo),b   ;write to foo (banked)
6F16  movwf BANKMASK(foo)     ;defaults to banked access
```

请注意，前两条指令操作码的 RAM 访问位（操作码的 bit 8）会清零，但在最后两条指令中该位将会置 1。

4.1.2 存储区和页选择

`BANKSEL()` 伪指令可以用于生成用以选择指定操作数所在的存储区的指令。操作数应为处于数据存储器中的某个对象的符号或地址。

根据目标器件，生成的代码可能包含一条或多条将相应寄存器中的位置 1/清零的位指令，或使用 `movlb` 指令（对于增强型中档或 PIC18 器件）。 由于该伪指令在中档或低档器件上会扩展为多条指令，所以在这些器件上，它不应紧随 `btfsx` 指令。

例如：

```
movlw 20
BANKSEL(_foobar)    ;select bank for next file instruction
movwf BANKMASK(_foobar)  ;write data and mask address
```

与之类似，`PAGESEL()` 伪指令可以用于生成用以选择地址操作数所在的页的代码。对于当前页，可以使用位置计数器 `$` 作为操作数。

根据目标器件，生成的代码可能包含一条或多条将相应寄存器中的位置 1/清零的指令，或使用 `movlp` 指令（对于增强型中档 PIC 器件）。由于该伪指令会扩展为多条指令，所以它不应紧随 `btfsx` 指令。

例如：

```
fcall _getInput
PAGESEL $        ;select this page
```

针对 PIC18 目标器件进行编译时，可以接受该伪指令，但它没有任何作用，不会生成任何代码。支持它纯粹是为了方便跨 8 位器件进行移植。

4.1.3 地址掩码

与大多数指令一起使用的文件寄存器地址均应进行掩码，以便从该地址中去除存储区信息。如果不进行掩码，可能会导致链接器修正（fixup）错误。

所有 MPLAB XC8 汇编标识符都代表一个完整的地址。该地址包括其代表的对象的存储区信息。在采用文件寄存器操作数的 8 位 PIC 指令集中，几乎所有指令均期望该操作数的值为当前所选存储区内的偏移量。由于不同器件系列具有不同的存储区大小，因此每个器件系列中该偏移量的宽度也有所不同。

`BANKMASK()` 宏可以与标识符或地址操作数一起使用。该宏使用合适的掩码通过逻辑“与”运算去除地址中的存储区信息。包含<xc.inc>后，该宏即可使用。使用该宏可提高汇编代码在 Microchip 不同器件之间的可移植性，因为它会调整掩码以适应目标器件的存储区大小。4.1.2 存储区和页选择给出了该宏的示例。

请勿将该宏与任何期望其操作数为完整地址的指令（例如 PIC18 的 `movff` 指令）一起使用。

4.1.4 Movfw 伪指令

MPASM 实现的 movfw 伪指令在 MPLAB XC8 汇编器中未实现。您将需要使用执行相同功能的标准 PIC 指令。请注意 MPASM 指令：

```
movfw foobar
```

直接映射到标准 PIC 指令：

```
movf foobar,w
```

4.1.5 Movff/movffl 指令

movff 指令是物理器件指令，但是对于具有扩展数据存储器的 PIC18 器件，它也用作 movffl 指令的占位符。

对于这些器件，在为 movff 指令生成输出时，汇编器会检查存放操作数符号的 psect。如果包含源操作数的 psect 和包含目标操作数的 psect 都指定了 lowdata psect 标志，则指令将被编码为双字 movff 指令。如果操作数是绝对地址，并且该地址位于存储器的低 4 KB 中，则对于短型指令而言也是可以接受的。在所有其他情况下，指令均被编码为三字 movffl 指令。

请注意，汇编列表文件将始终显示 movff 助记符，与其编码方式无关。检查操作码字数以确定哪条指令已被编码。

4.1.6 中断返回模式

retfie PIC18 指令后可以跟随“f”（无逗号），以指示在执行时应获取影子寄存器内容并复制到其相应的主寄存器。如果没有该修饰符，则不会从影子寄存器中更新其主寄存器。该语法与低档和中档器件无关。

以下同时给出两种形式的示例，以及它们生成的操作码。

```
0011  retfie f    ;shadow registers copied
0010  retfie      ;return without copy
```

如果需要，可以交替使用器件数据手册中指示的操作数 0 和 1。

4.1.7 长跳转和调用

汇编器可以识别几个会扩展为常规 PIC MCU 汇编指令的助记符。这些助记符为 fcall 和 ljmp。

在低档和中档器件上，这些指令分别扩展为常规的 call 和 goto 指令，并确保在目标地址位于程序存储器的另一个页中时生成必需的指令来设置 PCLATH（对于中档器件）或 STATUS（对于低档器件）中的位。页选择指令可能紧靠在 call 或 goto 之前，或作为先前 fcall/ljmp 助记符的一部分生成并紧随其后。

在 PIC18 器件上，提供这些助记符纯粹是为了与程序存储器较小的 8 位器件保持兼容，并且总是扩展为常规的 PIC18 call 和 goto 指令。

应尽可能使用这些特殊的助记符，因为它们可以使汇编代码独立于要执行的程序的最终位置。

以下中档 PIC 示例显示了汇编列表文件中的 fcall 指令。可以看到 fcall 指令已扩展为 5 条指令。在该示例中，存在两条置 1/清零 PCLATH 寄存器中位的位指令。在进行调用之后，为了重新选择在 fcall 之前选择的页，也会置 1/清零此寄存器中的位。

```
13  0079  3021                    movlw   33
14  007A  120A  158A  2000       fcall   _phantom
          120A  118A
15  007F  3400                    retlw   0
```

由于 fcall 和 ljmp 指令可能扩展为多条指令，所以在它们之前不应使用可以跳过的指令，例如 btfsc 指令。

fcall 和 ljmp 指令假定包含它们的 psect 小于一页。如果当前 psect 大于一页，请勿使用这些指令将控制转移到该 psect 中的某个标号。默认的链接器选项将不允许代码 psect 大于一页。

在 PIC18 器件上，常规 call 指令后可以跟随“,f”，用以指示应将 W、STATUS 和 BSR 寄存器压入其相应的影子寄存器。它代替器件数据手册中所述的“,1”语法。

4.1.8 相对转移

PIC18 器件实现了条件相对转移指令，例如 bz 和 bnz。与 goto 指令相比，这些指令的跳转范围相对有限。

与 MPLAB XC8 C 编译器不同，PIC 汇编器永远不会通过转换相对转移序列来扩大其范围。如果您需要一条可以到达常规指令范围以外的目标的相对转移指令，请使用通过 goto 指令以相反条件实现的相对转移。例如，不写：

```
  bz next
```

而是写诸如下面的内容：

```
  bnz tmp
  goto next
tmp:
```

4.2 语句格式

表 4-3 中列出了有效的语句格式。

label 字段是可选的，如果存在，则应包含一个标识符。标号可以自己单独一行出现，也可以在前面具有一个助记符，如第二种格式所示。

第三种格式仅对于某些汇编器伪指令是合法的，例如 MACRO、SET 和 EQU。*name* 字段是必需的，其中应包含一个标识符。

如果汇编源文件先由预处理器进行处理，那么它可能还包含构成有效预处理器伪指令的行。关于这些伪指令的格式的更多信息，请参见 5.1 预处理器伪指令。

对于语句的任何部分应出现在哪个列或行的哪个部分，不存在任何限制。

表 4-3. 汇编器语句格式

格式编号	字段 1	字段 2	字段 3	字段 4
格式 1	*label*:			
格式 2	*label*:	*mnemonic*	*operands*	*; comment*
格式 3	name	*pseudo-op*	*operands*	*; comment*
格式 4	*; comment only*			
格式 5	空行			

4.3 字符

所使用的字符集为标准 7 位 ASCII。字母大小写对于标识符是有意义的，但对于助记符和保留字则无意义。制表符等同于空格。

4.3.1 分隔符

所有数字和标识符都必须使用空格、非字母数字字符或行结束符进行分隔。

4.3.2 特殊字符

有几个字符在某些上下文中是特殊的。在宏主体内，字符&用于标记连接。要在宏主体内使用按位&操作符，需要改为通过使用&&对它进行转义，或使用该操作符的 and 形式。在宏参数列表中，尖括号<和>用于括起宏参数。

4.4 注释

汇编注释使用不属于字符串或字符常量一部分的分号开头。

如果汇编文件先由预处理器进行处理，则文件还可能包含使用标准/* ...*/和//语法的 C 或 C++样式的注释。

4.4.1 特殊注释字符串

MPLAB XC8 C 编译器生成的汇编代码使用特殊的注释字符串，例如;volatile 和;wreg free。由于 MPLAB XC8 PIC 汇编器不使用优化，因此这些字符串无效，不需要与汇编源代码一起使用。

4.5 常量

4.5.1 数字常量

汇编器会以有符号 32 位精度执行所有算术运算。

所有数字的默认基数为 10。可以通过尾随一个基数说明符（如表 5-2 所列）来指定其他基数。

表 4-4. 数字和基数

基数	格式
二进制	数字 0 和 1，后面跟随 B
八进制	数字 0 至 7，后面跟随 O、Q、o 或 q
十进制	数字 0 至 9，后面跟随 D 或 d，或不跟随任何字符
十六进制	数字 0 至 9、A 至 F，前面加上 0x 或后面跟随 H 或 h

十六进制数字必须具有一个前导数字（如 0ffffh），以将它们与标识符区分开。大写或小写形式的十六进制数字均可接受。

请注意，二进制常量后面必须跟随大写 B，因为小写 b 用于临时（数字）标号反向引用。

在表达式中，将以常规的格式接受实数，并将它解释为 IEEE 32 位格式。

4.5.2 字符常量和字符串

字符常量是包含在一对单引号'中的单个字符。

多字符常量或字符串是包含在一对匹配引号内的字符（不包括回车符或换行符）的序列。单引号'或双引号"均可使用，但左引号和右引号必须相同。

4.6 标识符

汇编标识符是用户定义的代表存储单元或数字的符号。符号可以包含任意数量的以下字符：字母、数字和特殊字符（美元符号$、问号?和下划线_）。

标识符的第一个字符不能为数字，也不能为$字符。字母区分大小写，例如，Fred 与 fred 是不同的符号。以下给出了标识符的一些示例：

```
An_identifier
an_identifier
an_identifier1
?$_12345
```

标识符不能为汇编器伪指令、关键字或 psect 标志。

4.6.1 标识符的意义

对于其他汇编器的用户，如果试图对标识符实现任何形式的数据类型定义，应注意该汇编器不对任何符号附加任何意义，也不对符号的用法作出任何限制或期望。

psect（程序段）的名称和普通符号占据独立的、重叠的名称空间，但除此之外，汇编器不关心符号是用于表示字节、字还是“跑车”。不需要也未提供特殊的语法来定义位或任何其他数据类型的地址，如果在多个上下文中使用了某个符号，汇编器也不会发出任何警告。指令和寻址模式语法将提供汇编器生成正确代码所需的所有信息。

4.6.2 汇编器生成的标识符

如果在某个宏块中使用了 LOCAL 伪指令，则在该宏的每个扩展中，汇编器将生成一个惟一符号来替换每个指定的标识符。这些惟一符号将具有??*nnnn* 的形式，其中的 *nnnn* 是一个 4 位数字。用户应避免定义具有相同形式的符号。

4.6.3 位置计数器

活动程序段内的当前位置可通过符号$访问。该符号会扩展为当前正在执行的指令的地址（这不同于执行该指令时程序计数器（Program Counter，PC）寄存器所包含的地址）。因而：

```
goto $   ;endless loop
```

表示将跳转到自身并构成无限循环。通过使用该符号和偏移量，可以指定相对跳转目标。

与$相加的所有地址偏移量都具有目标器件的本机寻址能力。因此，对于低档和中档器件，偏移量是相对于当前位置的指令数量，因为这些器件具有可字寻址的程序存储器。对于 PIC18 指令（使用可字节寻址的程序存储器），该符号的偏移量代表相对于当前位置的字节数。 由于 PIC18 指令必须按字对齐，相对于位置计数器的偏移量应为 2 的倍数。所有偏移量均向下舍入到最接近的 2 的倍数。

例如：

```
goto       $+2   ;skip...
movlw      8     ;to here for PIC18 devices, or
movwf      _foo  ;to here for baseline and mid-range devices
```

在低档或中档器件上将跳过 movlw 指令。在 PIC18 器件上，goto $+2 将跳转到随后的指令处；即，行为类似于 nop 指令。

4.6.4 寄存器符号

汇编模块中的代码可以通过包含预定义的汇编头文件来访问特殊功能寄存器。可以通过向汇编源文件添加行

```
#include <xc.inc>
```

来包含相应的文件，并在源文件名中使用扩展名.S 来确保对其进行预处理。该头文件包含相应的命令，以确保在源文件中包含特定于目标器件的头文件。

这些头文件包含所有字节或多字节大小寄存器的 EQU 声明，以及字节寄存器内指定位的#define 宏。

4.6.5 符号标号

标号是一个赋值等于当前 psect 内当前地址的符号别名。标号直到链接时才会赋值。

标号定义由任意有效汇编标识符组成，该标识符必须后跟一个冒号:。定义可以单独在一行上，也可以位于指令或汇编器伪指令的左侧。以下给出了两个穿插在汇编代码中的合法标号的示例。

```
frank:
  movlw   1
  goto        fin
simon44: clrf _input
```

此处，标号 frank 最终将被赋给 movlw 指令的地址，simon44 将被赋给 clrf 指令的地址。无论它们如何定义，汇编器生成的汇编列表文件将总是单独一行显示它们。

在汇编代码中可以使用（并最好使用）标号，而不在指令中使用绝对地址。这样一来，它们可以用作跳转型指令的目标位置，或用于将某个地址装入寄存器。

类似于变量，标号也具有作用域。默认情况下，可以在定义它们的模块中的任意位置使用它们。它们可以由位于其定义之前的代码使用。要使标号可在其他模块中访问，请使用 GLOBAL 伪指令（有关更多信息，请参见 4.9.26 Global 伪指令）。

4.7 表达式

指令和伪指令的操作数由表达式组成。表达式可以包含数字、标识符、字符串和操作符。

操作符可以为一元（一个操作数，如 not）或二元（两个操作数，如+）。表 5-3 列出了表达式中允许的操作符。

控制表达式语法的常规规则适用。

所列出的操作符全都可以在常量和可重定位表达式中自由地组合使用。链接器允许对复杂表达式进行重定位，因而涉及到可重定位标识符的表达式的结果可能直到链接时才会求解得到。

表 4-5. 汇编操作符

操作符	用途	示例
*	乘法	movlw 4*33,w
+	加法	bra $+1
-	减法	DB 5-2
/	除法	movlw 100/4
=或 eq	等于	IF inp eq 66
>或 gt	有符号大于	IF inp > 40
>=或 ge	有符号大于或等于	IF inp ge 66
<或 lt	有符号小于	IF inp < 40
<=或 le	有符号小于或等于	IF inp le 66
<>或 ne	有符号不等于	IF inp <> 40
low	操作数的低字节	movlw low(inp)
high	操作数的高字节	movlw high(1008h)
highword	操作数的高 16 位	DW highword(inp)
mod	求模	movlw 77mod4
&或 and	按位与	clrf inp&0ffh
^	按位异或	movf inp^80,w
|	按位或	movf inp|1,w
not	按位取反	movlw not 055h,w
<<或 shl	左移	DB inp>>8
>>或 shr	右移	movlw inp shr 2,w
rol	循环左移	DB inp rol 1
ror	循环右移	DB inp ror 1
float24	24 位形式的实数操作数	DW float24(3.3)
nul	测试宏参数是否为空值	

4.8 程序段

程序段或 psect 只是一段代码或数据。它们是一种将程序的各个部分组合在一起的方法（通过 psect 的名称），即使源代码在源文件中可能在物理上并不相邻或甚至分布在几个模块中也是如此。

psect 通过名称标识，并具有多个属性。PSECT 汇编器伪指令用于定义 psect。它需要一个名称参数，该参数后面可以是一个用逗号分隔的标志列表，这些标志用于定义其属性。6. 链接器介绍了可用于控制 psect 在存储器中的位置的链接器选项。这些选项可在驱动程序中使用-Wl 驱动程序选项进行访问（见 3.3.36 WL：将 Option 传递给链接器选项），无需显式运行链接器。

有关汇编器可以提供的所有 psect 的列表，请参见“汇编器提供的 Psect 和链接器类”部分。

除非定义为 abs（绝对）类型，否则 psect 将是可重定位的。

未使用 PSECT 伪指令显式放置到一个 psect 中的代码或数据将成为默认（未指定）psect 的一部分。由于您无法控制该 psect 的链接位置，因此建议始终将 PSECT 伪指令放置在代码和对象之前。

编写汇编代码时，包含<xc.inc>后，就可以使用提供的 psect。这些 psect 的名称和功能与 8 位 MPASM 汇编器使用的程序段相似。

如果自行创建 psect，请尝试将它们与现有的链接器类关联（见 5.3 默认链接器类和 6.1.2 C：将链接器类与 Psect 相关联），否则需要指定链接器选项才能正确分配它们。

请注意，psect 的长度和位置很重要。如果全部可执行代码所在的 psect 均未跨越任何器件页边界，则编写代码会更容易。同理，如果数据 psect 未跨越存储区边界，则编写代码也会更容易。psect 的位置（它们链接的位置）必须与访问 psect 内容的汇编代码匹配。

4.9 汇编器伪指令

汇编器伪指令或伪操作的使用方式类似于指令助记符。PAGESEL 和 BANKSEL 属于例外，这两条伪指令不会生成指令。DB、DW 和 DDW 伪指令会将数据字节放入当前 psect。下表列出了这些伪指令。

表 4-6. 汇编器伪指令

伪指令	用途
ALIGN	使输出按指定边界对齐。
ASMOPT	控制后续代码是否由汇编器优化。
BANKSEL	生成用以选择操作数存储区的代码。
CALLSTACK	指示剩余的调用堆栈深度。
[NO]COND	控制是否在列表文件中包含条件代码的操作。
CONFIG	指定配置位。
DB	定义常量字节。
DW	定义常量字。
DS	保留存储。
DABS	定义绝对存储。
DLABS	定义线性存储器绝对存储。
DDW	定义双倍宽度的常量字。
ELSE	切换条件汇编。
ELSIF	切换条件汇编。
ENDIF	结束条件汇编。
END	结束汇编。
ENDM	结束宏定义。
EQU	定义符号值。
ERROR	生成用户定义的错误。
[NO]EXPAND	控制是否在列表文件中扩展汇编器宏的操作。
EXTRN	与其他模块中定义的全局符号链接。

.....（续）

伪指令	用途
FNADDR	表示程序的地址已被占用。
FNARG	指示程序参数中的调用。
FNBREAK	断开调用图中的链接。
FNCALL	指示调用层级。
FNCONF	指示调用堆栈设置。
FNINDIR	指示程序进行的间接调用。
FNSIZE	指示程序自动对象和参数对象的大小。
FNROOT	指示调用树的根。
GLOBAL	使符号可供其他模块访问或允许引用其他模块中定义的其他全局符号。
IF	条件汇编。
INCLUDE	以文本形式包含指定文件的内容。
IRP	使用列表重复代码块。
IRPC	使用字符列表重复代码块。
[NO]LIST	为列表文件定义选项。
LOCAL	定义局部标签。
MACRO	宏定义。
MESSG	生成用户定义的建议性消息。
ORG	设置当前 psect 内的位置计数器。
PAGELEN	指定列表文件页的长度。
PAGESEL	生成用以设置该页 PCLATH 位的置 1/清零指令。
PAGEWIDTH	指定列表文件页的宽度。
PROCESSOR	定义对该文件进行汇编所针对的特定芯片。
PSECT	声明或恢复程序段。
RADIX	指定数字常量的基数。
REPT	重复代码块 n 次。
SET	定义或重新定义符号值。
SIGNAT	定义函数签名。
SUBTITLE	指定列表文件的程序的副标题。
TITLE	指定列表文件的程序的标题。

4.9.1 Align 伪指令

ALIGN 伪指令会将其后的所有内容（数据存储或代码等）对齐到当前 psect 内的指定偏移量边界。边界在伪指令之后以字节数的形式指定。

例如，要将输出对齐到 psect 内的 2 字节（偶）地址处，可以使用以下代码。

```
ALIGN 2
```

请注意，如果 psect 以偶地址作为起始，之后的内容将仅以偶数绝对地址作为起始；即，对齐是在当前 psect 内进行的。关于 psect 对齐，请参见 4.9.40.16 Reloc 标志。

ALIGN 伪指令还可以用于确保 psect 的长度为某个特定数字的倍数。例如，如果将以上 ALIGN 伪指令放在某个 psect 的末尾，该 psect 的长度将总是为偶数个字节长。

4.9.2 Asmopt 伪指令

ASMOPT *action* 伪指令在处理汇编代码时有选择地控制汇编器优化器。表 4-7 列出了允许的操作。

表 4-7. Asmopt 操作

操作	用途
off	针对后续代码禁止汇编器优化器。
on	针对后续代码使能汇编器优化器。
pop	检索汇编器优化设置的状态。
push	保存汇编器优化设置的状态。

在 ASMOPT off 伪指令之后，不会修改任何代码。在 ASMOPT on 伪指令之后，汇编器将执行允许的优化。

ASMOPT push 和 ASMOPT pop 伪指令允许将汇编器优化器的状态保存到状态堆栈中，以便之后再进行恢复。当您需要确保针对一小段代码禁止优化器时，它们非常有用，但是您不知道先前是否已禁止优化器。

例如：

```
ASMOPT PUSH  ;store the state of the assembler optimizers
ASMOPT OFF   ;optimizations must be off for this sequence
movlw 0x55
movwf EECON2
movlw 0xAA
movwf EECON2
ASMOPT POP   ;restore state of the optimizers
```

请注意，MPLAB XC8 PIC 汇编器不执行任何优化，这些控制将被忽略。

4.9.3 Banksel 伪指令

BANKSEL 伪指令可用于生成用以选择操作数所在的数据存储区的代码。操作数应为处于数据存储器中的某个对象的符号或地址（见 4.1.2 存储区和页选择）。

4.9.4 Callstack 伪指令

CALLSTACK *depth* 伪指令向汇编器指示在程序中的特定点仍可用的调用堆栈层级数。

汇编器优化器使用该指令来确定是否可以进行诸如过程抽象之类的转换。

请注意，MPLAB XC8 PIC 汇编器不执行任何优化，该控制将被忽略。

4.9.5 Cond 伪指令

COND 伪指令在汇编列表文件中包含条件代码。互补的 NOCOND 伪指令不会在列表文件中包含条件代码。

4.9.6 Config 伪指令

config 伪指令允许在汇编源文件中指定配置位或熔丝。

该伪指令具有以下形式：

```
CONFIG setting = value
CONFIG register = literal_value
```

此处，*setting* 是配置设置描述符，例如 WDT 和 *value* 可以是所需状态的文本描述（例如 OFF）或数值。数值的限制与其他数字常量操作数相同。

register 字段是配置或 ID 存储单元寄存器的名称。

可用 *setting*、*register* 和 *value* 字段记录在与器件相关的 chipinfo 文件（即 pic_chipinfo.html 和 pic18_chipinfo.html）中。

一条 CONFIG 伪指令可用于设置每个配置设置；或者，可以通过同一伪指令指定多个用逗号分隔的配置设置。该伪指令可以根据需要多次使用，以确保完全配置器件。

以下示例给出了配置寄存器作为整体进行编程和使用该寄存器中包含的各个设置单独编程的代码。

```
; PIC18F67K22
; VREG Sleep Enable bit : Enabled
; LF-INTOSC Low-power Enable bit : LF-INTOSC in High-power mode during Sleep
; SOSC Power Selection and mode Configuration bits : High Power SOSC circuit selected
; Extended Instruction Set : Enabled
config RETEN = ON, INTOSCSEL = HIGH, SOSCSEL = HIGH, XINST = ON
; Alternatively
config CONFIG1L = 0x5D
; IDLOC @ 0x200000
config IDLOC0 = 0x15
```

4.9.7 Db 伪指令

DB 伪指令用于以字节形式初始化存储单元。其参数是以逗号分隔的表达式列表，其中每个表达式将汇编为一个字节，并装入连续的存储单元中。

示例：

```
alabel: DB  'X',1,2,3,4,
```

如果程序存储器中地址单元的长度为 2 个字节（对于低档和中档器件就是如此，请参见 4.9.40.4 Delta 标志），DB 伪操作将初始化一个字，其高字节设置为零。以上示例将定义填充为以下字的字节。

```
0058 0001 0002 0003 0004
```

但在 PIC18 器件上（PSECT 伪指令的 delta 标志应为 1），将不会发生填充，并且在 HEX 文件中将出现以下数据。

```
58 01 02 03 04
```

4.9.8 Dw 伪指令

DW *value_list* 伪指令的工作方式类似于 DB，只是它会将表达式汇编为 16 位字。示例：

```
DW -1, 3664h, 'A'
```

4.9.9 Ds 伪指令

DS *units* 伪指令保留指定的空间大小，但不对其进行初始化。惟一的参数是要保留的地址单元数。地址单元由与存放该伪指令的 psect 一起使用的标志确定。

该伪指令通常用于在数据存储器中为基于 RAM 的对象保留字节（所在 psect 的 space 标志设置为 1）。如果该伪指令所在的 psect 是 bit psect（psect 的 bit 标志已置 1），则该伪指令保留请求的位数。如果在链接到程序存储器的 psect 中使用，它将会移动位置计数器，但不会在 HEX 文件输出中放入任何内容。请注意，在中档和低档器件上，由于程序存储器中地址单元的长度为 2 个字节（见 4.9.40.4 Delta 标志），因此在这种情况下 DS 伪操作实际上会保留字。

通常通过使用标号来定义对象，然后通过 DS 伪指令在标号位置处保留存储单元。

示例：

```
PSECT myVars,space=1,class=BANK2
alabel:
  DS 23    ;reserve 23 bytes of memory
PSECT myBits,space=1,bit,class=COMRAM
```

```
xlabel:
  DS 2+3   ;reserve 5 bits of memory
```

4.9.10 Ddw 伪指令

DDW 伪指令的工作方式类似于 DW，只是它会将表达式汇编为双倍宽度（32 位）的字。示例：

```
DDW 'd', 12345678h, 0
```

4.9.11 Dabs 伪指令

DABS 伪指令用于在指定地址处保留一个或多个字节的存储空间。该伪指令的一般形式为：

```
DABS memorySpace, address, bytes [,symbol]
```

其中，*memorySpace* 是一个编号，代表将保留的存储空间，*address* 是保留存储空间的地址，*bytes* 是要保留的字节数。symbol 是可选的，表示指定地址处对象的名称。

在该伪指令中使用符号将有助于调试。该符号可自动进行全局访问，等于该伪指令中指定的地址。例如，以下伪指令：

```
DABS 1,0x100,4,foo
```

与以下伪指令相同：

```
GLOBAL foo
foo EQU 0x100
DABS 1,0x100,4
```

该伪指令与 DS 伪指令的不同之处在于它可用于在任意位置保留存储空间，而不仅仅在当前 psect 内。事实上，这些伪指令可以放置在汇编代码中的任意位置，而不必一定放在当前选定 psect 中。

存储空间编号与使用 space 标志选项为 psect 指定的编号相同（见 4.9.40.18 Space 标志）。

链接器会从目标文件中读取此与 DABS 相关的信息，并确保不将保留的地址用于其他存储器分配。

4.9.12 Dlabs 伪指令

DLABS 伪指令用于在支持线性寻址的器件上的指定线性地址处保留一个或多个字节的存储器。该伪指令的一般形式为：

```
DLABS memorySpace, address, bytes [,symbol]
```

memorySpace 参数是一个表示线性存储空间的编号。该编号将与分区数据空间编号相同。*address* 是保留存储空间的地址，可将其指定为线性地址或分区地址。*bytes* 是要保留的字节数。symbol 是可选的，表示指定地址处对象的名称。

在该伪指令中使用符号将有助于调试。该符号可自动进行全局访问，等于该伪指令中指定的线性地址。例如，以下伪指令：

```
DLABS 1,0x120,128,foo
```

与以下伪指令相同：

```
GLOBAL foo
foo EQU 0x2080
DABS 1,0x120,80
DABS 1,0x1A0,48
DABS 1,0x20A0,0
```

请注意，对象跨两个存储区，线性地址 0x2080（而非分区地址 0x100）已被设为等于符号。

该伪指令与 DS 伪指令的不同之处在于它可用于在任意位置保留存储空间，而不仅仅在当前 psect 内。事实上，这些伪指令可以放置在汇编代码中的任意位置，而不必一定放在当前选定 psect 中。

存储空间编号与使用 space 标志选项为 psect 指定的编号相同（见 4.9.40.18 Space 标志）。

链接器会从目标文件中读取此与 DLABS 相关的信息，并确保不将保留的地址用于其他存储器分配。

4.9.13 End 伪指令

END *label* 伪指令是可选的，但如果存在，则应位于程序的最末尾。它将终止汇编，其后不能跟随任何内容，即使是空行。

如果将某个表达式作为参数提供，则将使用该表达式来定义程序的入口点。它存储在汇编器所生成的目标文件的起始记录中。它是否有任何用途将取决于链接器。

例如：

```
END  start_label  ;defines the entry point
```

或

```
END               ;do not define entry point
```

4.9.14 Equ 伪指令

EQU 伪操作定义一个符号，并使其值等于一个表达式。例如

```
thomas EQU 123h
```

标识符 thomas 将赋值为 123h。 只有符号先前未定义时，EQU 才是合法的。有关重新定义值的信息，请参见 4.9.43 Set 伪指令。

该伪指令执行与预处理器的#define 伪指令类似的功能（见 5.1 预处理器伪指令）。

4.9.15 Error 伪指令

error 伪指令生成用户定义的编译时错误消息，该消息将停止汇编器。如果执行该伪指令，则要打印的消息被指定为字符串。通常，将该伪指令放置在条件语句中，以检测无效情况。

例如：

```
IF MODE
  call process
ELSE
  ERROR "no mode defined"
ENDIF
```

4.9.16 Expand 伪指令

EXPAND 伪指令显示由汇编器列表文件中的宏扩展生成的代码。互补的 NOEXPAND 伪指令隐藏由汇编器列表文件中的宏扩展生成的代码。

4.9.17 Extrn 伪指令

EXTRN *identifier* 伪操作与 GLOBAL（见 4.9.26 Global 伪指令）类似，但只能用于与其他模块中定义的全局符号进行链接。如果将 EXTRN 与同一模块中定义的符号一起使用，则会触发错误。

4.9.18 Fnaddr 伪指令

FNADDR *routine* 伪指令告知链接器某程序已获取其地址，因此可以间接调用该程序。将对象分配给编译堆栈时，链接器将使用该信息。

4.9.19 Fnarg 伪指令

FNARG *routine1*,*routine2* 伪指令告知链接器对 *routine1* 程序的参数进行求值涉及调用 *routine2*，因此基于编译堆栈的两个程序的参数存储器不应重叠。

例如

```
FNARG init,start   ;start is called to obtain an argument for init
```

4.9.20 Fnbreak 伪指令

FNBREAK *routine1,routine2* 伪指令用于断开链接器生成的调用图信息中的链接。

这意味着，在检查出现在多个调用图中的程序时，不应考虑对以 routine2 为根的树之外的树中的 routine1 进行任何调用。routine1 的编译堆栈对象的存储器将仅在以 routine2 为根的程序的图中分配。

4.9.21 Fncall 伪指令

FNCALL *caller,callee* 伪指令告知链接器某程序已被另一个程序调用。链接器使用该信息来防止这两个程序定义的任何基于堆栈的对象在编译堆栈中重叠。

例如

```
FNCALL main,init   ;main calls init
```

4.9.22 Fnconf 伪指令

FNCONF *psect,autos,args* 伪指令用于为链接器提供调用图的配置信息。

第一个参数是应在其中放置编译堆栈的 psect 的名称。后两个参数是用于自动类型对象和参数类型对象的前缀。这两个前缀与用于定义对象的函数的名称一起使用。

例如：

```
FNCONF rbss,?a,?
```

告知链接器应将编译堆栈放置在名为 rbss 的 psect 中，并且任何自动变量块均以字符串?a 开头，函数参数块以?开头。 因此，使用 FNARG 伪指令定义编译堆栈对象的程序 foo 将在名为 rbss 的 psect 中创建分别以标识符?afoo 和?foo 开头的两个块。

4.9.23 Fnindir 伪指令

FNINDIR *routine,signature* 伪指令告知链接器指定程序已对具有指示签名值的程序执行间接调用。关于如何设置签名值的信息，请参见 4.9.44 Signat 伪指令。链接器将假定程序可能在调用具有匹配签名值且已获取其地址的任何其他程序。有关如何指示某函数已获取其地址的信息，请参见 4.9.18 Fnaddr 伪指令。

例如，如果一个名为 fred 的程序对签名值为 8249 的函数执行间接调用，请使用以下伪指令：

```
 FNINDIR _fred,8249
```

4.9.24 Fnroot 伪指令

FNROOT *routine* 伪指令告知链接器 *routine* 为调用图的根。该程序调用的程序是使用 FNCALL 伪指令编译的，请参见 4.9.21 Fncall 伪指令。每个调用图都会为其中的程序所定义的基于堆栈的对象分配惟一的存储空间。

4.9.25 Fnsize 伪指令

FNSIZE *routine,autos,args* 伪指令告知链接器与 *routine* 关联的自动变量和参数区域的大小。在编译调用图以及将地址分配给变量和参数区域时，链接器将使用这些值。

例如，伪指令：

```
FNSIZE fred, 10, 5
```

指示程序 fred 需要 10 字节的自动变量和 5 字节的参数。有关如何访问这些存储区域的信息，请参见 4.9.22 Fnconf 伪指令。

4.9.26 Global 伪指令

GLOBAL *identifier_list* 伪指令用于声明以逗号分隔的符号的列表。如果这些符号是在当前模块中定义的，则它们会被设为公共符号。如果这些符号不是在当前模块中定义的，则它们将可用于引用在外部模块中定义的公共符号。因而，要在两个模块中使用同一符号，必须至少使用 GLOBAL 伪指令两次：一次是在定义该符号的模块中，用以将该符号设为公共符号；另一次是在使用该符号的模块中，用以链接外部定义。

例如：

```
GLOBAL  lab1,lab2,lab3
```

4.9.27 If、Elsif、Else 和 Endif 伪指令

这些伪指令用于实现条件汇编。

IF 和 ELSIF 的参数应为绝对表达式。如果它不为零，则对从其后一直到下一个匹配 ELSE、ELSIF 或 ENDIF 的代码进行汇编。如果表达式为零，则不会输出下一个匹配的 ELSE 或 ENDIF 之前的代码。在 ELSE 处，将反向解释条件编译的含义，而 ENDIF 将终止条件汇编块。条件汇编块可以进行嵌套。

这些伪指令并未实现与 C 语句 if 一样的运行时条件语句；它们仅在编译代码时进行求值。此外，对于汇编代码，无论条件为 true 还是 false，都会始终进行扫描和解析，但对于指令所对应的机器码，仅当条件匹配时才会将其输出。这意味着，无论条件表达式的状态如何，都将处理汇编器伪指令（例如 EQU），因此不应在 IF 结构内部使用这些伪指令。

例如：

```
IF ABC
    goto aardvark
ELSIF DEF
    goto denver
ELSE
    goto grapes
ENDIF
ENDIF
```

在该示例中，如果 ABC 非零，则汇编第一条 goto 指令，但不汇编第二条或第三条。如果 ABC 为零，DEF 非零，则汇编第二条 goto 指令，但不汇编第一条和第三条。如果 ABC 和 DEF 均为零，则汇编第三条 goto 指令。

4.9.28 Include 伪指令

INCLUDE "*filename*"伪指令会导致指定文件以文本形式替换该伪指令。例如：

```
INCLUDE "options.inc"
```

汇编器驱动程序不会向汇编器传递任何搜索路径，所以如果包含文件不位于当前工作目录中，必须通过文件名指定文件的完整路径。

扩展名为.S 的汇编源文件已经过预处理，因此可使用预处理器伪指令，例如#include（它是 INCLUDE 伪指令的替代伪指令）。

4.9.29 Irp 和 Irpc 伪指令

IRP 和 IRPC 伪指令的工作方式类似于 REPT；但是，它们不是将块重复固定的次数，而是对于参数列表的每个成员重复一次。

对于 IRP，其列表是一个常规的宏参数列表。对于 IRPC，它是一个参数中的每个字符。在每次重复时，都会将一个形参替换为实参。

例如：

```
IRP number,4865h,6C6Ch,6F00h
    DW number
ENDM
```

将扩展为：

```
DW 4865h
DW 6C6Ch
DW 6F00h
```

请注意，您可以使用与常规宏相同的方式使用局部标号和尖括号。

IRPC 伪指令与之类似，只是它每次从一个非空格字符组成的字符串中替换一个字符。

例如：

```
IRPC char,ABC
    DB 'char'
ENDM
```

将扩展为：

```
DB 'A'
DB 'B'
DB 'C'
```

4.9.30 List 伪指令

LIST 伪指令控制是否生成列表输出。

如果先前使用 NOLIST 伪指令关闭了列表，LIST 伪指令将再次将其开启。

此外，LIST 控制还可以包含用于控制汇编和列表的选项。表 5-9 列出了这些选项。

表 4-8. List 伪指令选项

列表选项	默认值	说明
c=*nnn*	80	设置页（即列）的宽度。
n=*nnn*	59	设置页的长度。
t=ON|OFF	OFF	截断列表输出行。默认值将折行。
p=*device*	n/a	设置器件类型。
x=ON|OFF	OFF	开启或关闭宏扩展。

4.9.31 Local 伪指令

LOCAL *label* 伪指令用于为给定宏的每个扩展定义惟一标号。对于在 LOCAL 伪指令之后列出的所有符号，在对宏进行扩展时，都会被替换为汇编器生成的惟一符号。例如：

```
down MACRO count
  LOCAL more
  more: decfsz count
  goto more
ENDM
```

在进行扩展时，将包含汇编器生成的惟一标号来代替 more。例如：

```
down foobar
```

将扩展为：

```
??0001 decfsz foobar
goto ??0001
```

如果第二次进行调用，标号 more 将扩展为??0002，并避免多次定义符号的错误。

4.9.32 Macro 和 Endm 伪指令

MACRO … ENDM 伪指令用于定义汇编宏，并可以选择使用参数。关于值与标识符的简单关联，请参见 4.9.14 Equ 伪指令；关于预处理器的#define 宏伪指令（它也可以使用参数），请参见 5.1 预处理器伪指令。

MACRO 伪指令前面应为宏名，并（可选）在后面跟随形参的逗号分隔列表。在使用宏时，应以与机器操作码相同的形式使用宏名，后面跟随用于替换形参的参数列表。

例如：

```
;macro: movlf - Move a literal value into a nominated file register
;args:  arg1 - the literal value to load
;       arg2 - the NAME of the source variable
movlf   MACRO   arg1,arg2
  movlw arg1
  movwf arg2 mod 080h
ENDM
```

在使用时，该宏将扩展为宏主体内的 2 条指令，并将形参替换为实参。因而：

```
movlf 2,tempvar
```

将扩展为：

```
movlw 2
movwf tempvar mod 080h
```

&字符可用于将宏参数与其他文本连接，但在实际扩展中会将其去除。例如：

```
loadPort MACRO port, value
  movlw value
  movwf PORT&port
ENDM
```

如果在调用时 port 为 A，则将装入 PORTA，依此类推。宏中&标记的特殊含义意味着不能在汇编宏中使用按位 AND 运算符（也可表示为&）；而应使用该运算符的 and 形式。

通过使用两个分号;;作为注释的开头，可以在宏扩展中禁止注释（从而节省宏存储空间）。

在调用某个宏时，参数列表必须用逗号分隔。如果希望在参数中包含逗号（或其他分隔符，如空格），可以使用尖括号<和>将其括起。

如果参数前面具有一个百分号%，则该参数将作为表达式进行求值，并以十进制数字而不是字符串的形式传递。如果宏主体内的参数求值会产生不同的结果，这将非常有用。

在宏中可以使用 nul 操作符来测试宏参数，例如：

```
IF nul     arg3  ;argument was not supplied.
...
ELSE             ;argument was supplied
...
ENDIF
```

关于在宏中使用惟一局部标号的信息，请参见 4.9.31 Local 伪指令。

默认情况下，汇编列表文件将以未扩展格式显示宏；即，以调用宏的形式。通过使用 EXPAND 汇编器伪指令，可以在列表文件中显示宏的扩展形式（见 4.9.16 Expand 伪指令）。

4.9.33 Messg 伪指令

messg 伪指令生成用户定义的编译时建议性消息。执行该伪指令不会阻止汇编器进行编译。如果执行该伪指令，则要打印的消息被指定为字符串。通常，将该伪指令放置在条件语句中，以检测无效情况。

例如：

```
IF MODE
  call process
```

```
ELSE
  MESSG "no mode defined"
ENDIF
```

4.9.34 Org 伪指令

ORG 伪指令用于更改当前 psect 内的位置计数器的值。这意味着使用 ORG 设置的地址将相对于 psect 的基址，它直到链接时才会确定。

注： 经常被滥用的 ORG 伪指令并不会将位置计数器移动到您指定的绝对地址处。只有放置该伪指令的 psect 是绝对且重叠时，才会将位置计数器移动到指定地址处。要将对象放置在特定地址处，可以将它们放置在它们自己的 psect 中，并使用链接器-P 选项将该 psect 链接到所需的地址处（见 6.1.17 P：定位 psect）。在程序中通常不需要使用 ORG 伪指令。

ORG 的参数必须为绝对值或引用当前 psect 的值。在两种情况下，当前的位置计数器均设置为由参数确定的值。无法将位置计数器向后移动。例如：

```
ORG 100h
```

会将位置计数器移动到当前 psect 起始位置加 100h 处。实际位置只有到链接时才会知道。

要使用 ORG 伪指令将位置计数器设置为某个绝对值，必须从重叠的绝对 psect 内使用该伪指令。例如：

```
PSECT absdata,abs,ovrld
ORG 50h
;this is guaranteed to reside at address 50h
```

4.9.35 Page 伪指令

PAGE 伪指令会导致在列表输出中开始新页。在源代码中遇到 Control-L（换页）字符时，也会导致产生新页。

4.9.36 Pagelen 伪指令

PAGELEN *nnn* 伪指令将汇编列表的长度设置为指定行的数量。

4.9.37 Pagesel 伪指令

PAGESEL 伪指令可用于生成用以选择地址操作数所在页的代码。（见 4.1.2 存储区和页选择）。

4.9.38 Pagewidth 伪指令

PAGEWIDTH *nnn* 伪指令将汇编列表的宽度设置为指定字符的数量。

4.9.39 Processor 伪指令

如果汇编器源代码仅适用于一个器件，则应在模块中使用 PROCESSOR 伪指令。编译时必须始终使用-mcpu 选项来指定代码编译所针对的目标器件。如果伪指令和选项中指定的器件不匹配，则会触发错误。

例如：

```
PROCESSOR 18F4520
```

4.9.40 Psect 伪指令

PSECT 伪指令用于声明或恢复某个程序段。

该伪指令可接受一个名称和一个（可选）标志的逗号分隔列表作为参数。允许的标志指明了 psect 的属性。表 5-5 列出了这些标志。

psect 名称处于独立于普通汇编符号的名称空间中，所以 psect 可以使用与普通汇编标识符相同的标识符。但是，psect 名称不能为汇编器伪指令、关键字或 psect 标志。

声明某个 psect 之后，稍后可以使用另一条 PSECT 伪指令恢复它；但此时不需要重复指定标志，它们将从先前的声明中传递。如果遇到同一 psect 的两条 PSECT 伪指令具有相互冲突的标志，则将产生错误，但可重新指定 reloc、size 和 limit 标志，而不产生错误。

表 4-9. Psect 标志

标志	含义
abs	psect 是绝对的。
bit	psect 存放位对象。
class=*name*	指定 psect 的类名。
delta=*size*	寻址单元的长度。
global	psect 是全局的（默认）。
inline	可以在调用时内联 psect 内容（函数）。
keep	内联后不会删除 psect。
limit=*address*	psect 的地址上限（仅 PIC18）。
local	psect 是惟一的，不会与其他同名的 psect 链接。
lowdata	psect 将完全位于 0x1000 地址之下。
merge=*allow*	允许或阻止合并该 psect。
noexec	出于调试目的，该 psect 不包含可执行代码。
note	psect 不包含任何应出现在程序映像中的数据。
optim=*optimizations*	指定该 psect 允许的优化。
ovrld	psect 将与其他模块中的相同 psect 重叠。
pure	psect 将是只读的。
reloc=*boundary*	psect 在指定边界处开始。
size=*max*	psect 的最大长度。
space=*area*	表示 psect 将驻留的区域。
split=*allow*	允许或阻止拆分该 psect。
with=*psect*	将 psect 放入与指定 psect 相同的页。

以下给出了一些使用 PSECT 伪指令的示例：

```
; swap output to the psect called fred
PSECT fred
; swap to the psect bill, which has a maximum size of 100 bytes and which is global
PSECT bill,size=100h,global
; swap to joh, which is an absolute and overlaid psect that is part of the CODE linker class,
; and whose content has a 2-byte word at each address
PSECT joh,abs,ovrld,class=CODE,delta=2
```

4.9.40.1 Abs 标志

abs psect 标志用于将当前 psect 定义为绝对的；即，它从地址 0 开始。这并不意味着该模块中的对象在 psect 中的地址将从 0 开始，因为其他模块中的对象也可能存放在同一 psect 中（另请参见 4.9.40.14 Ovrld 标志）。

带有 abs 标志的 psect 是不可重定位的，如果发出尝试将这种 psect 放在任意位置的链接器选项，将产生错误。

4.9.40.2 Bit 标志

bit psect 标志指定 psect 存放 1 位长的对象。这种 psect 的 scale（比例）值将为 8，指示每个存储字节具有 8 个可寻址单元，与该 psect 关联的所有地址均为位地址，而不是字节地址。psect 的非单位 scale 值在映射文件中指示（见 6.3 映射文件）。

4.9.40.3 Class 标志

`class` psect 标志指定该 psect 对应的链接器类名。类是一个可在其中放置 psect 的地址范围。

类名可用于在链接时定位局部 psect，因为它们并不总是可以通过`-P` 链接器选项按它们自己的名称进行引用（如果具有多个同名的局部 psect，就会如此）。

需要仅将 psect 定位到某个地址范围中的任意位置，而不是某个特定地址时，类名也很有用。将某个类与某个已定义的 psect 关联通常意味着不需要提供定制的链接器选项即可将其放置在存储器中。

关于如何定义链接器类的信息，请参见 6.1.1 A：定义链接器类。

4.9.40.4 Delta 标志

`delta` psect 标志用于定义可寻址单元的长度。即，与每个地址相关联的数据字节数。

对于 PIC 中档和低档器件，程序存储空间是可字寻址的，因而该空间中的 psect 使用的 delta（增量）必须为 2。 即，程序存储器中的每个地址在 HEX 文件中都需要 2 个数据字节来定义其内容。因此，HEX 文件中的地址将与程序存储器中的地址不匹配。

这些器件上的数据存储空间是可字节寻址的，因而该空间中的 psect 使用的 delta 必须为 1。这是默认的 delta 值。

PIC18 器件上的所有存储空间都是可字节寻址的，所以应对于这些器件上的所有 psect 使用 delta 值 1（默认值）。

使用相互冲突的 delta 值重新定义 psect 可能导致汇编器发出阶段错误。

4.9.40.5 Global 标志

`global` psect 标志指示链接器应将该 psect 与其他模块中的同名全局 psect 连接在一起。

除非使用了 `local` 标志，否则 psect 在默认情况下被视为全局类型。

4.9.40.6 Inline 标志

该标志已弃用。可考虑改用 `optim` psect 标志。

代码生成器使用 `inline` psect 标志来告知汇编器可以内联 psect 的内容。如果执行该操作，则将复制 `inline` psect 的内容并用于替换对 psect 中定义的函数的调用。

4.9.40.7 Keep 标志

该标志已弃用。可考虑改用 `optim` psect 标志。

`keep` psect 标志可确保在汇编器优化器进行任何内联后不会删除 psect。内联后可删除内联候选 psect（见 4.9.40.6 Inline 标志）。

4.9.40.8 Limit 标志

`limit` psect 标志用于指定 psect 可延伸到的最高地址的限制。如果将它定位到存储器中时超出该限制，将产生错误。当前仅在为 PIC18 器件编译时可用。

4.9.40.9 Local 标志

使用 `local` psect 标志定义的 psect 在链接时不会与来自其他模块的其他 `local` psect 进行合并，即使存在其他同名的 psect 也是如此。如果在一个模块中存在两个 `local` psect，它们将引用相同的 psect。`local` psect 不能与任何 `global` psect 同名，即使它是处于另一个模块中也是如此。

无法使用`-P` 链接器选项将与链接器类（见 4.9.40.3 Class 标志）不关联的局部 psect 链接到某个地址，因为可能有多个同名 psect。通常，这些 psect 会定义一个类标志，并放置在该类范围内的任何位置。

4.9.40.10 Merge 标志

该标志已弃用。可考虑改用 `optim` psect 标志。

`merge` psect 标志用于控制 psect 的合并方式。该标志可以赋值为 0 或 1，或不指定。当赋值为 0 时，汇编优化器在优化过程中将永远不会合并 psect。如果赋值为 1，则 psect 可被合并：如果其他 psect 属性允许该操作，并且优化器确定这样做存在优势。如果未指定该标志，则不会发生合并。

通常情况下，仅对基于代码的 psect（`text` psect）执行合并。

4.9.40.11 Noexec 标志

noexec psect 标志用于指示 psect 不包含可执行代码。该信息仅与调试有关。

4.9.40.12 Note 标志

note psect 标志由特殊 psect 使用，其内容用于汇编器或调试器工具，不会复制到最终程序输出中。指定 note 标志后，将禁止其他几个 psect 标志，如果将这些标志与同一 psect 一起使用，将产生警告。

4.9.40.13 Optim 标志

optim psect 标志用于指示可对 psect 内容执行的优化，前提是允许并使能了此类优化。

该标志对于使用 MPLAB XC8 PIC 汇编器编译的汇编代码不起作用，因而不执行优化。

优化由冒号分隔的名称列表指示，如表 5-6 所示。空列表表示无法对 psect 进行优化。

表 4-10. Optim 标志名称

名称	优化
inline	允许内联 psect 内容。
jump	执行基于跳转的优化。
merge	允许将 psect 的内容与其他类似 psect 的内容合并（仅 PIC10/12/16 器件）。
pa	执行过程抽象。
peep	执行窥孔优化。
remove	如果完全内联，则允许将 psect 完全删除。
split	允许将超出大小限制范围的 psect 拆分为多个较小的 psect（仅 PIC10/12/16 器件）。
empty	不对该 psect 执行任何优化。

例如，psect 定义：

```
PSECT myText,class=CODE,reloc=2,optim=inline:jump:split
```

可对 myText psect 内容执行内联、拆分和跳转型优化，前提是已使能这些优化。定义：

```
PSECT myText,class=CODE,reloc=2,optim=
```

会禁止与该 psect 相关的所有优化，而与优化器设置无关。

optim psect 标志可取代多个单独的 psect 标志来使用：merge、split、inline 和 keep。

4.9.40.14 Ovrld 标志

ovrld psect 标志告知链接器该 psect 的内容应在链接时与其他模块的内容重叠。通常，不同模块的同名 psect 会连接在一起。同一模块中存放在重叠 psect 中的对象始终连接在一起。

配合使用该标志与 abs 标志（见 4.9.40.1 Abs 标志）可定义真正的绝对 psect；即，在该 psect 中定义的所有符号都是绝对符号。

4.9.40.15 Pure 标志

pure psect 标志告知链接器该 psect 不在运行时进行修改。例如，将其放置在 ROM 中。该标志的作用有限，因为它取决于链接器和实施它的目标系统。

4.9.40.16 Reloc 标志

reloc psect 标志用于指定将 psect 对齐到特定边界。指定边界必须为 2 的幂，例如 2、8 或 0x40。例如，标志 reloc=100h 将指定该 psect 的起始地址必须为 0x100 的倍数（例如，0x100、0x400 或 0x500）。

PIC18 指令必须按字对齐，因而对于包含可执行代码的所有 PIC18 psect，使用的 reloc 值必须为 2。所有其他段和所有其他器件的所有段通常可以使用默认 reloc 值 1。

4.9.40.17 Size 标志

`size` psect 标志用于为 psect 指定最大长度，例如 `size=100h`。从所有模块中组合 psect 之后，链接器将检查这一点。

4.9.40.18 Space 标志

`space` psect 标志用于区分具有重叠地址但存在区别的存储区域。定位到程序存储器和数据存储器的 psect 具有不同的 space 值；举例来说，它可以指示程序空间地址 0 是不同于数据存储器地址 0 的存储单元。

表 5-7 列出了与不同 space 标志编号关联的存储空间。

表 4-11. Space 标志编号

Space 标志编号	存储空间
0	程序存储器和 PIC18 器件的 EEPROM
1	数据存储器
2	保留
3	中档器件上的 EEPROM
4	配置位
5	IDLOC
6	注

具有分区数据空间的器件不使用不同的 space 值来标识每个存储区。对于该空间中的对象，将使用包含存储区编号的完整地址，因而可以惟一地标识每个存储单元。例如，存储区大小为 0x80 字节的器件将使用地址 0 至 0x7F 来表示存储区 0 中的对象，使用地址 0x80 至 0xFF 来表示存储区 1 中的对象，依此类推。

4.9.40.19 Split 标志

该标志已弃用。可考虑改用 `optim` psect 标志。

`split` psect 标志可以赋值为 0 或 1，或不指定。当赋值为 0 时，汇编优化器在优化过程中将永远不会拆分 psect。如果赋值为 1，则可拆分 psect：如果其他 psect 属性允许进行拆分，并且 psect 太大，无法装入可用的存储器。如果未指定该标志，则将基于该 psect 是否可以合并来决定是否拆分该 psect，请参见 4.9.40.10 Merge 标志。

4.9.40.20 With 标志

`with` psect 标志用于将某个 psect 放置在与另一个 psect 相同的页中。例如，标志 `with=text` 将指定该 psect 应放置在与 `text` psect 相同的页中。

术语 `withtotal` 指的是与其他 psect 放置在一起的每个 psect 的长度总和。

4.9.41 Radix 伪指令

`RADIX` *`radix`* 伪指令为在汇编器源文件中指定的数字常量控制基数。表 4-12 列出了允许的基数。

表 4-12. 基数操作数

基数	含义
`dec`	十进制常量
`hex`	十六进制常量
`oct`	八进制常量

4.9.42 Rept 伪指令

`REPT` 伪指令用于临时定义一个未命名的宏，然后根据其参数对它进行多次扩展。

例如：

```
REPT 3
    addwf fred,w
ENDM
```

将扩展为：

```
addwf fred,w
addwf fred,w
addwf fred,w
```

（见 4.9.29 Irp 和 Irpc 伪指令）。

4.9.43 Set 伪指令

SET 伪指令等效于 EQU（4.9.14 Equ 伪指令），只是它允许重新定义符号，而不会产生错误。例如：

```
thomas SET 0h
```

该伪指令执行与预处理器的#define 伪指令类似的功能（见 5.1 预处理器伪指令）。

4.9.44 Signat 伪指令

SIGNAT 伪指令用于将一个 16 位签名值与某个标号相关联。在链接时，链接器会检查为特定标号定义的所有签名是否相同，如果它们不同，将产生一个错误。使用 SIGNAT 伪指令来强制在链接时检查函数原型和调用约定。

例如：

```
SIGNAT _fred,8192
```

会将签名值 8192 与符号_fred 关联。如果任何目标文件中存在_fred 的不同签名值，链接器将报告错误。

通常，该伪指令与从 C 调用的汇编语言程序一起使用。确定 MPLAB XC8 C 编译器使用的签名值的最简单方法是编写与汇编程序具有相同原型的 C 程序，并检查由代码生成器确定的相应函数的签名伪指令参数（如汇编列表文件所示）。

4.9.45 Space 伪指令

SPACE *nnn* 伪指令在汇编列表输出中插入 *nnn* 空行。

4.9.46 Subtitle 伪指令

SUBTITLE "*string*"伪指令用于定义一个出现在每个汇编列表页面顶部，但处于标题下的子标题。应将该子标题括在单引号或双引号中。

4.9.47 Title 伪指令

TITLE "*string*"伪指令关键字用于定义出现在每个汇编列表页顶部的标题。应将该标题括在单引号或双引号中。

5. 汇编器功能

PIC 汇编器允许访问 C 预处理器以及可以在程序中使用的许多预定义 psect 和标识符。

5.1 预处理器伪指令

MPLAB XC8 除了标准伪指令之外，还可接受几条专用的预处理器伪指令。这些伪指令均已在下表中列出。

表 5-1. 预处理器伪指令

伪指令	含义	示例
#	预处理器空伪指令，不执行任何操作。	#

#define	定义预处理器宏。	`#define SIZE (5)`
		`#define FLAG`
		`#define add(a,b) ((a)+(b))`

#elif	#else #if 的缩写。	请参见#ifdef
#else	根据条件包含源代码行。	请参见#if
#endif	终止条件包含源代码。	请参见#if
#error	产生一条错误消息。	#error Size too big

#if	如果常量表达式为 true，则包含源代码行。	`#if SIZE < 10`
		`  c = process(10)`
		`#else`
		`  skip();`
		`#endif`

#ifdef	如果预处理器符号已定义，则包含源代码行。	`#ifdef FLAG`
		`  do_loop();`
		`#elif SIZE == 5`
		`  skip_loop();`
		`#endif`

#ifndef	如果预处理器符号未定义，则包含源代码行。	`#ifndef FLAG`
		`  jump();`
		`#endif`

#include	在源代码中包含文本文件。	`#include <stdio.h>` `#include "project.h"`
#line	指定列表的行号和文件名	#line 3 final
#*nn* filename	（其中 *nn* 为编号，*filename* 为源文件名）其后的内容源自指定的文件和行号。	#20 init.c
#undef	取消定义预处理器符号。	#undef FLAG
#warning	产生一条警告消息。	#warning Length not set

带参数的宏扩展可以使用#字符将参数转换为一个字符串，以及使用##序列来连接参数。如果要连接两个表达式，可以考虑使用两个宏，以处理其中任一表达式自身需要替换的情况，例如

```
#define __paste1(a,b)   a##b
#define __paste(a,b)    __paste1(a,b)
```

使您可以使用 paste 宏来连接两个自身可能需要进一步扩展的表达式。请记住，对某个宏标识符进行扩展之后，如果它在连接之后出现，将不会再次对它进行扩展。

5.2 汇编器提供的 Psect

PIC 汇编器提供 psect 定义（如下表所示），可根据需要用于存放代码和数据。在源文件中包含<xc.inc>后，即可使用这些 psect，其名称与在 MPASM 汇编器中创建相似段的 MPASM 伪指令类似。例如，要将指令置于 code psect 中，请使用以下代码。

```
#include <xc.inc>
PSECT code
;place instructions here
```

表中列出了与各个 psect 相关联的链接器类以及这些 psect 定义所适用的目标器件系列。此外，所示的链接器类也已由 PIC 汇编器预定义，因此无需定义。

如果需要，可以改用您自己定义的 psect 和链接器类。如果 psect 必须位于特定的位置，则必须使用惟一的 psect，以便可以将其独立地链接到其他 psect。请参见 4.9.40 Psect 伪指令和 6.1.17 P：定位 psect。

表 5-2. 汇编器提供的 Psect 和链接器类

Psect 名称	链接器类	目标器件系列	用途
code	CODE	全部	存放可执行代码
eedata	EEDATA	全部	将数据存放在 EEPROM 中
data	STRCODE	低档和中档	将数据存放在程序存储器中
data	CONST	PIC18	将数据存放在程序存储器中
udata	RAM	全部	将可分配的对象存放在 GPR 中的任意位置
udata_acs	COMRAM	PIC18	将可分配的对象存放在快速操作存储区 GPR 中
udata_bank*n*	BANKN	全部	将可分配的对象存放在特定的数据存储区中
udata_shr	COMMON	低档和中档	将可分配的对象存放在公共存储区中

5.3 默认链接器类

链接器使用类来表示 psect 可以链接到的存储器范围。

类由链接器选项定义（见 6.1.1 A：定义链接器类）。汇编器驱动程序根据所选的目标器件将默认的一组此类选项传递给链接器。

Psect 通常从与它们关联的类中分配可用存储空间。关联使用 PSECT 伪指令的 class 标志来实现（见 4.9.40.3 Class 标志）。或者，也可以使用链接器选项将 psect 显式地放置在与某个类相关联的存储器中（见 6.1.17 P：定位 psect）。

类可能代表单个存储器范围或多个范围。即使两个范围是连续的，一个范围结束、另一个范围开始的地址也会构成边界，放置在类中的 psect 永远不能跨越这种边界。您可以创建这样的类：涵盖相同的地址，但被拆分为不同的范围，具有不同的边界。这样便可适应这样的 psect：其内容对其自身或其所访问数据在存储器中的位置做出假设。从一个类中分配的存储器会被保留，防止被指定了相同存储器地址的其他类使用。

对于链接器，类名或它定义的存储器没有任何意义。

如果使用-mreserve 选项（见 3.3.24 保留选项）或者使用-mram 和-mrom 选项移除存储器范围（见 3.3.23 Ram 选项和 3.3.25 Rom 选项），可以从这些类中移除存储器。

除了从类中保留存储器之外，切勿更改或移除类指定的地址边界。

5.3.1 程序存储器类

包含<xc.inc>后，就会定义以下链接器类，这些类代表程序存储空间。并非每款器件都存在所有类。

CODE 包含映射到目标器件上的程序存储器页的范围，用于包含可执行代码的 psect。
在低档器件上，它只能由通过跳转表访问的代码使用。

ENTRY 与低档器件 psect 相关，此类 psect 包含通过 `call` 指令访问的可执行代码。调用只能针对这些器件上的页的上半部分。该类以这种方式定义：它的长度为整个页，但它包含的 psect 将定位为使它从页的上半部分开始。
该类还用于中档器件中，将包含许多 0x100 字长并在 0x100 边界对齐的范围。

STRING 包含多个 0x100 字长并在 0x100 边界对齐的范围。因此，它对于内容不能跨越 0x100 字边界的 psect 很有用。

STRCODE 定义涵盖整个程序存储器的单个存储器范围。它对于内容可以出现在任何页中并可以跨越页边界的 psect 很有用。

CONST 包含多个 0x100 字长并在 0x100 边界对齐的范围。因此，它对于内容不能跨越 0x100 字边界的 psect 很有用。

5.3.2 数据存储器类

包含<xc.inc>后，就会定义以下链接器类，这些类代表数据存储空间。并非每款器件都存在所有类。

RAM 包含涵盖目标器件所有通用 RAM 存储器、但不包括任何公共（非分区）存储区的范围。
因此，它对于必须放置在通用 RAM 存储区中的 psect 很有用。

BIGRAM 包含单个存储器范围，该范围涵盖增强型中档器件的线性数据存储器或 PIC18 器件的整个可用存储空间。它适用于其内容通过线性寻址方式访问或自身不需要包含在单个数据存储区中的任何 psect。

ABS1 包含涵盖目标器件所有通用 RAM 存储器，包括任何公共（非分区）存储区在内的范围。
因此，它对于必须放置在通用 RAM 中、但可以放置在任意存储区或公共存储区中的 psect 很有用。

BANK*x* （其中的 *x* 是存储区编号）——每个均包含单个范围，该范围涵盖该存储区中的通用 RAM，但不包括任何公共（非分区）存储区。

COMMON 对于所有中档器件，包含单个存储器范围，该范围涵盖公共（非分区）RAM（如存在）。

COMRAM 对于所有 PIC18 器件，包含单个存储器范围，该范围涵盖公共（非分区）RAM（如存在）。

SFR*x* （其中的 *x* 是编号）——每个均包含单个范围，该范围涵盖存储区 x 中的 SFR 存储器。
编程人员通常不使用这些类，因为它们不代表通用 RAM。

5.3.3 其他类

包含<xc.inc>后，就会定义以下链接器类，这些类代表特殊用途的存储器。并非每款器件都存在所有类。

CONFIG 包含单个范围，该范围涵盖为配置位数据保留的存储器。
编程人员通常不使用该类，因为它不代表通用 RAM。

IDLOC 包含单个范围，该范围涵盖为 hex 文件中的 ID 单元数据保留的存储器。编程人员通常不使用该类，因为它不代表通用 RAM。

EEDATA 包含单个范围，该范围涵盖目标器件的 EEPROM 存储器（如存在）。该类用于包含要编程到 EEPROM 中的数据的 psect。

5.4 链接器定义的符号

链接器会定义特殊的符号，可以用于确定一些段链接到存储器中的位置。如果需要，可以在代码中使用这些符号。

段的链接地址可以通过名称为__L*name*（两个前导下划线）的全局符号的值获得，其中的 *name* 是段的名称。例如，__LbssBANK0 是 bssBANK0 段的下边界。段的最高地址（即链接地址加长度）使用符号__H*name* 表示。如果段具有不同的装入和链接地址，则装入起始地址使用符号__B*name* 表示。

不为未使用-P 链接器选项置于存储器中的段（见 6.1.17 P：定位 psect）分配这种符号类型，请注意段名称会因器件而异。

汇编代码可以通过全局声明这些符号来使用它们（注意名称中的两个前导下划线字符），例如：

```
GLOBAL __Lidata
```

5.5 汇编列表文件

汇编列表文件是可阅读的列表，其中显示了最终输出中的操作码及其所在地址。

如果使用-Wa,-a 选项指示，汇编器会生成汇编列表文件。每个汇编源文件都会生成一个汇编列表文件。

5.5.1 列表文件格式

汇编列表文件将内容分配到各列中，其一般形式为：

***line* [*address*] [*data*]**

列表文件的每个 ***line*** 都有一个行号。这些数字仅与列表文件本身有关，而与生成列表的汇编源文件中的行号无关。

address（如果存在）是任何输出在器件中所处的地址。它可以是程序或数据空间地址，由该行所在的 psect 确定。查找相关行上方的第一条 psect 伪指令。

data（如果存在）表示与指定地址关联的内容。如果这是指令操作码，则显示该指令的助记符。代表数据存储器中标号或内容的行通常不显示数据字段，因为没有与这些行相对应的汇编器输出。数据也可以是以分号;开头的注释或汇编器伪指令。

以下 PIC18 示例给出了一个典型的列表文件。请注意 movlw 指令，其操作码为 0E50（位于程序存储器中的地址 746E 处），位于列表文件的第 51135 行；DS 伪指令用于保留地址 100 处的数据存储空间。

```
51131                           PSECT brText,class=CODE,space=0,reloc=2
51132                           ; Clear objects in BANK1
51133                           GLOBAL bank1Data
51134  00746A  EE01  F000           lfsr   0,bank1Data
51135  00746E  0E50                 movlw        80
51136  007470                       clear:
51137  007470  6AEE                 clrf postinc0,c
51138  007472  06E8                 decf wreg
51139  007474  E1FD                 bnz  clear
51140  007476  0012                 return
51141  007452                       PSECT        bank1,class=BANK1,space=1,noexec
51142                           bank1Data:
51143  000100                   input:
51144  000100                      DS      2
```

如果链接阶段已成功结束，链接器将会更新列表文件，使它包含绝对地址和符号值。因而，可以使用汇编列表文件来确定指令的位置和确切的操作码。 地址或操作码旁边的汇编列表中的撇号标记“'”表示链接器未更新列表文件，这很可能是由于编译错误或在链接阶段之前停止编译的汇编器选项所致。例如，在以下列表中：

```
85  000A' 027F                   subwf   127,w
86  000B' 1D03                   skipz
87  000C' 2800'                  goto    u15
```

这些标记表示地址只是其所在 psect 中的地址偏移量，并且操作码尚未修正。操作码中任何未修正的地址字段都将显示为值 0。

5.5.2 Psect 信息

汇编列表文件可以用于确定数据对象或代码段所放入的 psect 的名称。

对于全局符号，您可以查看映射文件中的符号表，其中列出了每个符号的 psect 名称。对于模块的局部符号，请在列表文件中查找符号的定义。对于标号，则是符号的名称后跟一个冒号“:”。查找该代码上方的第一条 PSECT 汇编器伪指令。与该伪指令关联的名称是放置代码的 psect 的名称（见 4.9.40 Psect 伪指令）。

5.5.3 符号表

每个汇编列表文件的底部是一个符号表。它与映射文件中的符号表不同（见 6.3.2.6 符号表），主要体现在以下两个方面：

- 仅列出与生成该汇编列表文件的汇编模块（而不是整个程序）关联的符号。
- 列出与该模块关联的局部和全局符号。

列出每个符号及已为其分配的地址。

6. 链接器

本章介绍链接器的工作原理以及用法。

链接器的应用程序名称为 hlink。大多数情况下都不需要直接调用链接器，因为汇编器驱动程序 pic-as 会自动使用所有必需的参数执行链接器。直接使用链接器并不简单，只有那些非常熟悉链接过程要求的用户才应尝试这么做。如果 psect 未正确链接，可能会导致代码失败。

6.1 工作原理

链接器命令采用以下形式：

```
hlink [options] files
```

options 是零个或多个不区分大小写的链接器选项，其中每个选项都会以某种方式修改链接器的行为。*files* 是一个或多个目标文件，以及零个或多个库文件（.a 扩展名）。

表 6-1 列出了链接器可识别的选项，随后的段落对它们进行了介绍。

表 6-1. 链接器命令行选项

选项	作用
-8	使用 8086 样式的段：偏移地址形式。
-A*class=low-high ,...*	指定类的地址范围。
-C*psect=class*	指定全局 psect 的类名。
-C*baseaddr*	生成以 baseaddr 为基址的二进制输出文件。
-D*class=delta*	指定类的 delta（增量）值。
-D*symfile*	生成旧式的符号文件。
-E*errfile*	将错误消息写入 errfile。
-F	生成仅具有符号记录的.o 文件。
-G *spec*	指定段选择器的计算。
-H *symfile*	生成符号文件。
-H+ *symfile*	生成增强型符号文件。
-I	忽略未定义的符号。
-J *num*	设置在中止之前的最大错误数。
-K	阻止重叠函数参数和自动变量区域。
-L	保留.o 文件中的重定位项。
-LM	保留.o 文件中的段重定位项。
-N	按地址顺序对映射文件中的符号表排序。
-Nc	按类地址顺序对映射文件中的符号表排序。
-Ns	按空间地址顺序对映射文件中的符号表排序。
-M*mapfile*	在指定文件中生成链接映射。
-O*outfile*	指定输出文件的名称。
-P*spec*	指定 psect 地址和排序。

.....（续）	
选项	作用
-Q*processor*	指定器件类型（仅出于美化原因）。
-S	禁止符号文件中的符号列表。
-S*class=limit*[,*bound*]	指定属于某个类的 psect 的地址限制和起始边界。
-U*symbol*	以未定义形式在表中预先输入符号。
-V*avmap*	使用文件 avmap 来生成 Avocet 格式的符号文件。
-W*warnlev*	设置警告级别（-9 至 9）。
-W*width*	设置映射文件宽度（>=10）。
-X	从符号文件中移除所有局部符号。
-Z	从符号文件中移除不重要的局部符号。
--DISL=*list*	指定禁止的消息。
--EDF=*path*	指定消息文件位置。
--EMAX=*number*	指定最大错误数。
--NORLF	不重定位列表文件。
--VER	打印版本号并停止。

如果标准输入是一个文件，则假定该文件包含命令行参数。可以通过在前一行的末尾加上反斜杠\来进行换行。通过这种方式，可以发出几乎无限长的 hlink 命令。例如，名为 x.lnk 并包含以下文本的链接命令文件：

```
-Z -Ox.o -Mx.map \
-Ptext=0,data=0/,bss,nvram=bss/.\
x.o y.o z.o
```

可以通过以下方法之一传递给链接器：

```
hlink @x.lnk
hlink < x.lnk
```

有几个链接器选项要求指定存储器地址或长度。 这些选项的语法都是类似的。默认情况下，数字会被解释为十进制值。要强制将它解释为十六进制数字，应添加一个 H 或 h 后缀。例如，765FH 将被视为十六进制数字。

6.1.1 A：定义链接器类

-A*class=low-high* 选项用于为一个或多个地址范围分配一个链接器类，以便 psect 可以放置在该类中的任何位置。范围不需要是连续的。例如：

```
-ACODE=1020h-7FFEh,8000h-BFFEh
```

指定名为 CODE 的类代表所示的两个不同地址范围。

通过使用-P 选项并将类名用作地址，可以将 psect 放置在这两个范围内的任何位置（见 6.1.17 P：定位 psect），例如：

```
-PmyText=CODE
```

另外，属于 CODE 类的任何 psect，一经定义（见 4.9.40.3 Class 标志），将自动链接到该范围，除非它们被另一个选项显式地定位。

如果存在一些相同的连续地址范围，则可以使用 x 字符后跟重复计数的方式来指定它们。例如：

```
-ACODE=0-0FFFFhx16
```

指定存在 16 个连续范围，每个均从地址 0 开始，长度为 64 KB。虽然这些范围是连续的，但没有任何 psect 会跨越 64k 边界，因此该选项产生的 psect 放置将与指定选项

```
-ACODE=0-0FFFFFh
```

时不同，该选项不包含 64k 倍数边界。

-A 选项不会指定与地址关联的存储空间。将某个 psect 分配给某个类之后，该 psect 的 space 值会被分配给该类（见 4.9.40.18 Space 标志）。

6.1.2 C：将链接器类与 Psect 相关联

-C*psect=class* 选项用于将某个 psect 与某个特定类相关联。通常，在命令行上不需要该选项，因为 psect 类在目标文件中指定（见 4.9.40.3 Class 标志）。

6.1.3 D：定义类的 Delta 值

-D*class=delta* 选项用于定义属于指定类成员的 psect 的 delta 值。*delta* 值应为一个数字，表示 psect 内对象的每个可寻址单元的字节数。大多数 psect 都不需要该选项，因为它们已定义了 delta 值（见 4.9.40.4 Delta 标志）。

6.1.4 D：定义旧式的符号文件

使用-D*symfile* 选项可生成旧式的符号文件。旧式的符号文件是一个 ASCII 文件，其中每行都具有符号的链接地址，之后跟随符号名称。

6.1.5 E：指定错误文件

-E*errfile* 选项会使链接器将所有错误消息都写入指定的文件，而不是屏幕，后者是默认的标准错误目标。

6.1.6 F：生成仅包含符号的目标文件

通常，链接器会生成同时包含程序代码和数据字节以及符号信息的目标文件。有时需要生成仅包含符号的目标文件，用于在再次运行链接器时提供符号值。-F 选项禁止输出文件中包含数据和代码字节，使其仅包含符号记录。

在需要将一个项目的一部分（即，独立编译）与其他项目共用时，可以使用该选项，自举程序和应用程序都可能会出现这种情况。一个项目的文件使用该链接器选项进行编译来生成仅包含符号的目标文件；然后，将该目标文件与其他项目的文件进行链接。

6.1.7 G：使用替代段选择器

使用分段或需进行存储区切换的 psect 链接程序时，链接器可以通过两种方式为每个段分配段地址或选择器。段定义为一组连续的 psect，序列中的每个 psect 的链接和装入地址都与组中的上一个 psect 相连。段的段地址或选择器是在链接器处理段类型重定位时获得的值。

默认情况下，段选择器通过将段的基本装入地址除以段的重定位量而得到；重定位量基于在汇编器级别为 psect 指定的 reloc=标志值（见 4.9.40.16 Reloc 标志）。-G*spec* 选项可用作一种计算段选择器的替代方法。-G 的参数为以下形式的字符串：

```
A/10h-4h
```

其中，A 表示段的装入地址，/表示除法运算。它表示“获取 psect 的装入地址，除以十六进制数字 10，然后减去 4。”这种形式可以修改为以下形式：以 N 代替 A，以*代替/（表示乘法运算），并加上而不是减去一个常量。标记 N 会被替换为由链接器分配的段编号。例如：

```
N*8+4
```

表示“获取段编号，乘以 8 并加上 4”。其结果为段选择器。该特定示例将按序列 4, 12, 20...为已定义的段编号分配段选择器。

映射文件中会显示每个 psect 的选择器（见第 6.4.2.2 节“按模块列出的 psect 信息”）。

6.1.8 H：生成符号文件

`-Hsymfile` 选项将指示链接器生成符号文件。可选参数 *symfile* 指定用于接收数据的文件的名称。默认文件名为 `l.sym`。

6.1.9 H+：生成增强型符号文件

`-H+symfile` 选项将指示链接器生成增强型符号文件。该文件除提供标准符号文件的内容外，还提供与每个符号关联的类名，以及一个关于段的部分，其中列出每个类名及其所占用的存储器范围。如果要使用调试器运行代码，则建议使用该格式。可选参数 *symfile* 指定用于接收符号文件的文件。默认文件名为 `l.sym`。

6.1.10 I：忽略未定义的符号

通常，无法解析对未定义符号的引用属于致命错误。使用`-I` 选项会导致将未定义符号改为视为警告。

6.1.11 J：指定最大错误计数

链接器会在错误达到一定数量之后停止处理目标文件（而不是产生警告）。默认数量为 10，但是`-Jerrcount` 选项可以用于更改它。

6.1.12 L：允许装入重定位

在链接器生成输出文件时，它通常不会保存任何重定位信息，因为文件现在是绝对文件。在某些情况下，会在装入时对程序进行进一步的“重定位”。`-L` 选项将对于输入中的每个重定位记录在输出文件中生成一个空的重定位记录。

6.1.13 LM：允许段装入重分配

类似于`-L` 选项，`-LM` 选项会在输出文件中保存重定位记录，但仅保存段重定位记录。

6.1.14 M：生成映射文件

`-Mmapfile` 选项会导致链接器在指定文件或标准输出（如果省略了文件名）中生成链接映射。6.3 映射文件说明了映射文件的格式。

6.1.15 N：指定符号表排序

默认情况下，映射文件中的符号表按名称排序。`-N` 选项将使它基于符号的值以数字顺序排序。`-Ns` 和`-Nc` 选项的作用类似，但符号将按其 space 值或类进行组合。

6.1.16 O：指定输出文件名

该选项用于指定目标文件的输出文件名。

6.1.17 P：定位 psect

基于通过`-Pspec` 选项传递给链接器的信息将 psect 链接在一起，并为其分配地址。`-P` 选项的参数由逗号分隔的序列组成，格式为：

```
-Ppsect=linkaddr+min/loadaddr+min,psect=linkaddr/loadaddr,...
```

所有值都可以省略，这种情况下将根据前面的值应用默认值。psect 的链接地址是用于在运行时访问它的地址。装入地址是 psect 在输出文件（HEX 或二进制文件等）中的起始地址，但是 8 位 PIC 器件很少使用它。 指定地址可以为数字地址或者其他 psect、类或特殊标记的名称。

该选项的最常见基本形式的示例如下：

```
-Ptext10=02000h
```

将 psect `text10` 的起始地址放置（链接）到地址 0x2000 处：

```
-PmyData=AUXRAM
```

将 psect myData 放置在链接器类 AUXRAM（需要使用-A 选项进行定义，请参见 6.1.1 A：定义链接器类）指定的地址范围内的任何位置，

```
-PstartCode=0200h,endCode
```

将 endCode 紧接在起始地址为 0x200 的 startCode 的末尾放置。

该选项很少需要使用其他形式，但下文会进行介绍。

如果不允许链接或装入地址降至最小值以下，则后缀+*min* 表示最小地址。

如果链接地址为负数，则 psect 将以相反的顺序进行链接，即 psect 的顶端出现在指定地址减 1 处。跟随在负地址后的 psect 将放置在存储器中第一个 psect 之前。

如果完全省略装入地址，则它默认设为链接地址。如果提供的斜杠/字符后面没有地址，则装入地址将与前一个 psect 的装入地址连接在一起。例如，在处理如下选项之后：

```
-Ptext=0,data=0/,bss
```

text psect 的链接地址和装入地址为 0；data 的链接地址为 0，装入地址位于 text 装入地址之后。bss psect 的链接地址和装入地址都与 data 相连。

指定为点字符“.”的装入地址告知链接器将装入地址设置为与链接地址相同。

映射文件中将显示 psect 的最终链接和装入地址（见 6.3.2.2 按模块列出的 psect 信息）。

6.1.18 Q：指定器件

-Q*processor* 选项用于指定器件类型。它纯粹用于放入映射文件中的信息。该选项的参数是一个描述器件的字符串。没有任何行为会因器件类型而发生改变。

6.1.19 S：省略符号文件中的符号信息

-S 选项可阻止将符号信息包含到链接器生成的符号文件中。但仍然会包含段信息。

6.1.20 S：对类应用地址上限

属于某个类的 psect 可能具有与之关联的地址上限。-S*classlimit*[,*bound*]=选项的以下示例将属于 CODE 类的 psect 的最高地址限制为小于 400h。

```
-SCODE=400h
```

请注意，要设置某个 psect 的上限，必须在汇编代码中使用 psect limit 标志设置（见 4.9.40.8 Limit 标志）。

如果使用了 *bound*（边界）参数，则属于该类的 psect 的地址将从边界地址的倍数处开始。以下示例将属于 FARCODE 类的 psect 放置在 1000h 倍数地址处，但地址上限为 6000h。

```
-SFARCODE=6000h,1000h
```

6.1.21 U：添加未定义的符号

-U*symbol* 选项用于将指定符号作为未定义符号输入到链接器的符号表中。这对于以下情形很有用：完全从库中进行链接，或从库中链接某个模块、并且顺序安排为在默认情况下链接后一个模块。

6.1.22 V：生成 Avocet 符号文件

要生成 Avocet 格式的符号文件，需要使用-V*avmap* 选项向链接器提供一个映射文件，以使它能够将 psect 名称映射为 Avocet 存储器标识符。*avmap* 文件通常由编译器提供，或由编译器驱动程序根据需要自动创建。

6.1.23 W：指定警告级别/映射宽度

-W*num* 选项可以用于设置警告级别（范围为-9 至 9）或映射文件的宽度（对于 *num* >= 10 时的值）。

-W9 将禁止所有警告消息。-W0 是默认值。将警告级别设置为-9（-W-9）将提供最全面的警告消息。

6.1.24 X：省略符号文件中的局部符号

使用-X 选项可以从符号文件中禁止局部符号。全局符号将总是出现在符号文件中。

6.1.25 Z：省略符号文件中不重要的符号

一些局部符号是编译器生成的，调试过程并不关心它们。-Z 选项将从符号文件中禁止具有单个字母字符后跟一个数字字符串形式的所有局部符号。可以用于开始一个不重要符号的字母集合当前为“klfLSu”。-Z 选项将剔除以其中字母之一开始并后跟数字字符串的所有局部符号。

6.1.26 Disl

--disl=*messages* 选项主要由命令行驱动程序 pic-as 用于禁止特定消息编号。它可接受要在编译过程中禁止的消息编号的逗号分隔列表。

6.1.27 Edf

--edf=*file* 选项主要由命令行驱动程序 pic-as 用于指定消息描述文件的路径。默认文件位于汇编器安装目录的 dat 目录中。

6.1.28 Emax

--emax=*number* 选项主要由命令行驱动程序 pic-as 用于指定在汇编器终止之前可以遇到的最大错误数量。默认数量为 10 个错误。

如果使用命令行驱动程序 pic-as 和-fmax-errors 驱动程序选项进行编译，则会应用该选项。

6.1.29 Norlf

使用--norlf 选项可以阻止链接器对汇编器所生成的汇编列表文件应用修正。该选项通常由命令行驱动程序 pic-as 在执行预链接阶段时使用，但在执行最终的链接步骤时会省略它，以使列表文件显示最终的绝对地址。

如果要尝试解决修正错误，则应禁止该选项，从而修正汇编列表文件，并可以为该文件计算绝对地址。如果汇编器驱动程序检测到存在等于 1 的预处理器宏__DEBUG，则在编译时将禁止该选项。在 MPLAB X IDE 中选择 Debug（调试）编译时，该宏会置 1，所以如果遇到此类错误，请总是选择该选项。

6.1.30 Ver

--ver 选项会打印说明链接器版本和内部版本的信息。即使命令行上存在其他选项和文件，链接器也会在处理完该选项之后终止。

6.2 Psect 和重定位

链接器可以读取可重定位目标文件（.o 扩展名）和目标文件库（.a 扩展名）。库文件是打包到单个单元中的目标文件的集合，解压后的处理方式与单个目标文件的处理方式相同。

每个目标文件都包含一定数量的记录。每个记录都具有一个类型，指示它存放何种信息。一些记录类型存放关于目标器件及其配置的一般信息，其他记录类型可能存放数据和其他信息（如程序调试信息）。

目标文件中的许多信息都与 psect（程序段）有关。psect 是一种汇编域构造，实质上是某种内容（指令或数据）组成的块。程序中的所有对象都位于 psect 中。关于 psect 的介绍指南，请参见 4.8 程序段。有一种特定的记录类型用于存放 psect 中的数据。每个目标文件的大部分内容都由包含可执行代码和一些对象的 psect 记录组成。

链接器执行以下任务。

- 将引用的所有可重定位目标文件的内容合并为一个。
- 将目标文件中包含的 psect 重定位到可用器件存储器中。
- 修正 psect 的内容中的符号引用。

重定位包括将 psect 分配到目标器件的存储器。

目标器件的存储器规范通过链接器选项的方式传递给链接器。这些选项由命令行驱动程序 pic-as 生成。不存在任何链接描述文件，也无法在任何源文件中指定这些选项。默认链接器选项很少需要调整。但是，如果需要，可以谨慎地使用驱动程序选项-Wl 来更改它们（见 3.3.36 WL：将 Option 传递给链接器选项）。

将 psect 放置在其最终存储单元之后，就可以将 psect 中的符号引用替换为绝对值。该过程称为修正。

链接器的输出是单个目标文件。该目标文件是绝对的，因为重定位已完成，并且所有代码和对象都分配了地址。

6.3 映射文件

映射文件包含有关 psect 的存储器分配和赋给这些 psect 内符号的地址的信息。

6.3.1 映射文件生成

如果通过 MPLAB X IDE 执行编译，则默认情况下会生成一个映射文件。如果从命令行上使用驱动程序，则需要使用`-Wl,-Map` 选项来请求生成映射文件（见 3.3.36 WL：将 Option 传递给链接器选项）。通常为映射文件分配扩展名`.map`。

映射文件由链接器应用程序生成。如果编译过程在执行链接器之前停止，则不会生成任何映射文件。即使链接器生成错误，也会生成一个映射文件，并且这个部分完成的文件可以帮助您查出这些错误的原因。但是，如果由于太多错误或致命错误而导致链接器未运行完毕，则将不会创建映射文件。可以使用`-fmax-errors` 驱动程序选项来增加链接器退出之前允许的错误数量。

6.3.2 内容

映射文件中的各个部分如下所示（按出现顺序）。

- 汇编器名称和版本号。
- 用于调用链接器的命令行的副本。
- 链接的第一个文件中的目标代码的版本号。
- 机器类型。
- 按 psect 的父目标文件排序的 psect 摘要。
- 按 psect 的类排序的 psect 摘要。
- 段摘要。
- 未用地址范围摘要。
- 符号表。
- 每个函数的信息摘要。
- 每个模块的信息摘要。

以下几节显示了示例映射文件的某些部分，以及说明性文本。

6.3.2.1 一般信息

映射文件顶部是与链接器执行有关的一般信息。

在分析程序时，请总是确认映射文件最顶部显示的汇编器版本号，以确保正在使用的汇编器是您希望使用的编译器。

使用`-mcpu` 选项（见 3.3.5 Cpu 选项）或在 IDE 中选择的器件应出现在 **Machine type**（机器类型）条目之后。

object code version（目标代码版本）与汇编器生成的可重定位目标文件所用的文件格式有关。除非独立地更新了汇编器或链接器，否则不需要关注它。

一个典型映射文件的开头可能类似以下精简示例。

```
Linker command line:
--edf=/Applications/Microchip/XC8/2.20/dat/en_msgs.txt -cs -h+main.sym -z \
-Q16F946 -ol.o -Mmain.map -ver=XC8 -ACONST=00h-0FFhx32 \
-ACODE=00h-07FFhx4 -ASTRCODE=00h-01FFFh -AENTRY=00h-0FFhx32 \
-ASTRING=00h-0FFhx32 -ACOMMON=070h-07Fh -ABANK0=020h-06Fh \
-ABANK1=0A0h-0EFh -ABANK2=0120h-016Fh -ABANK3=01A0h-01EFh \
-ARAM=020h-06Fh,0A0h-0EFh,0120h-016Fh,01A0h-01EFh \
-AABS1=020h-07Fh,0A0h-0EFh,0120h-016Fh,01A0h-01EFh -ASFR0=00h-01Fh \
-ASFR1=080h-09Fh -ASFR2=0100h-011Fh -ASFR3=0180h-019Fh \
-preset_vec=00h,intentry,init,end_init -ppowerup=CODE -pfunctab=CODE \
-ACONFIG=02007h-02007h -pconfig=CONFIG -DCONFIG=2 -AIDLOC=02000h-02003h \
-pidloc=IDLOC -DIDLOC=2 -AEEDATA=00h-0FFh/02100h -peeprom_data=EEDATA \
-DEEDATA=2 -DCODE=2 -DSTRCODE=2 -DSTRING=2 -DCONST=2 -DENTRY=2 -k \
```

```
startup.o main.o

Object code version is 3.10

Machine type is 16F946
```

Linker command line:（链接器命令行：）后的信息会显示对于最后一次编译传递给链接器的所有命令行选项和文件。请记住，这些是链接器选项，不是命令行驱动程序选项。

链接器选项一定是很复杂的。幸运的是，很少需要调整它们，只需使用其默认设置。它们由命令行驱动程序 `pic-as` 基于所选的目标器件和指定的驱动程序选项构造。通常，可以通过查看映射文件中的链接器选项来确认驱动程序选项是否有效。例如，如果指示驱动程序保留某个存储区，将会看到所使用的链接器选项中发生变化。

如果必须更改默认的链接器选项，可以使用驱动程序`-Wl` 选项间接通过驱动程序来实现（见 3.3.36 WL：将 Option 传递给链接器选项）。如果使用该选项，请总是确认映射文件中出现正确的变化。

6.3.2.2 按模块列出的 psect 信息

映射文件中的下一个部分会列出产生输出的那些模块，以及关于这些模块定义的 psect 的信息。

该部分以包含以下题头的行开始：

```
Name   Link  Load  Length  Selector  Space  Scale
```

该行下最左侧是目标文件（`.o` 扩展名）的列表。其中将显示从源文件模块生成的目标文件和从目标库文件（`.a` 扩展名）中提取的目标文件。对于后一种情况，库文件的名称会打印在目标文件列表之前。

目标文件旁边是从该目标文件链接到程序中的 psect（在 **Name**（名称）列下）。各列显示有关该 psect 的有用信息，如下所示。

链接器处理两种地址：链接地址和装入地址。一般来说，psect 的**链接**地址是用于在运行时访问它的地址。

Load（装入）地址（通常与链接地址相同）是 psect 在输出文件（HEX 或二进制文件等）内的起始地址。如果某个 psect 用来存放位对象，则装入地址用于存放链接地址（位地址）转换为的字节地址。

psect 的 **Length**（长度）将以 psect 所使用的单元显示。

Selector（选择器）较不常用，针对 PIC 器件进行编译时不用关心它。

Space（空间）字段非常重要，因为它指示 psect 所放入的存储空间。对于具有独立存储空间的哈佛架构机器（如 PIC 器件），该字段必须与地址一起使用，用以指定确切的存储位置。space 为 0 指示程序存储器，space 为 1 指示数据存储器（见 4.9.40.18 Space 标志）。

psect 的 **Scale**（比例）指示每个字节的地址单元数量。如果 scale 为 1，它将保留为空；对于存放位对象的 psect，它将显示为 8。存放位对象的 psect 的装入地址将显示链接地址转换为的字节地址（见 4.9.40.2 Bit 标志）。

例如，映射文件中显示了以下内容。

```
Name     Link  Load  Length Selector  Space  Scale
ext.o    text    3A     3A      22       30      0
         bss     4B     4B      10       4B      1
         rbit    50      A       2        0      1      8
```

这表明链接器处理了一个名为 `ext.o` 的文件。

该目标文件包含 `text` psect，以及名为 `bss` 和 `rbit` 的 psect。

psect `text` 链接到地址 3A，`bss` 链接到地址 4B。乍看之下，考虑到 text 为 22 个字长，这似乎会发生问题。但是，由于 space 标志指示它们处于不同的存储区（`text` 的 space 标志为 0，`bss` 的 space 标志为 1），因此它们甚至不会占用相同的存储空间。

psect `rbit` 包含位对象，这可以通过查看 scale 值（值为 8）确认。同样，乍看之下，似乎可能发生 `rbit` 被链接到 `bss` 之上的问题。它们的 space 标志是相同的，但由于 `rbit` 包含位对象，其链接地址将为位地址。`rbit` psect 的装入地址字段显示链接地址转换为的字节地址：50h/8 => Ah。

6.3.2.3 按类列出的 psect 信息

映射文件中的下一个部分会显示相同的 psect 信息，但将按 psect 的类进行分组。

该部分以包含以下题头的行开始：

```
TOTAL   Name  Link  Load  Length
```

在这下面是类名，后面跟随属于该类的 psect（见 4.9.40.3 Class 标志）。 这些 psect 与上一个部分中按模块列出的那些 psect 是相同的；该部分中不包含任何新信息，只是不同的呈现方式。

6.3.2.4 段列表

在映射文件中，类列表后面是段列表。用户通常可以忽略映射文件的这一部分。

段是同一存储空间中的连续 psect 的概念性分组，链接器使用其来协助进行 psect 放置。不存在段汇编器伪指令，也无法以任何方式来控制段。

该部分以包含以下题头的行开始：

```
SEGMENTS   Name  Load  Length  Top  Selector  Space  Class
```

段的名称将来自连续分组中链接地址最低的 psect。这可能会导致与同名 psect 产生混淆。不要阅读映射文件这一部分中的 psect 信息。

映射文件的这一部分同样可以忽略。

6.3.2.5 未用地址范围

存储器摘要的最后部分将显示尚*未*分配、仍可使用的存储器。

该部分跟随以下题头：

```
UNUSED ADDRESS RANGES
```

其后为类的列表，以及每个类中仍然可用的存储器。如果某个类中具有多个存储器范围，则每个范围将单独打印一行。其中不会显示类中的任何分页边界，但 Largest block（最大的块）列将显示在考虑到存储器范围中的所有分页之后的最大连续可用空间。如果要确定为什么 psect 无法放入存储器的原因（例如，无法找到空间类型的错误），则需要研究一下这个重要的信息。

请注意，存储器可能与多个类关联，因而可用空间总量并不只是将所有未用范围相加。

6.3.2.6 符号表

映射文件中的下一个部分会按字母顺序列出程序定义的全局符号。该部分具有以下题头：

```
Symbol Table
```

该表中列出的符号包括：

- 全局汇编标号
- 全局 EQU/SET 汇编器伪指令标号
- 链接器定义的符号

汇编符号通过 GLOBAL 汇编器伪指令设为全局符号，请参见 4.9.26 Global 伪指令了解更多信息。

链接器定义的符号的行为类似于 EQU 伪指令。但是，它们是由链接器在链接过程中定义的，它们的所有定义都不会出现在任何源文件或中间文件中（见 5.4 链接器定义的符号）。

对于每个符号都会显示定义时将其放入的 psect，以及已为其分配的值（通常为一个地址）。不会包含指示符号是代表代码还是数据，或者指示它所驻留的存储空间的信息。

如果某个符号的 psect 显示为（abs），这意味着该符号不直接与 psect 相关联。绝对 C 变量或在汇编代码中使用 EQU 伪指令定义的所有符号就是如此。

请注意，在每个汇编列表文件中也会显示一个符号表。这些符号表与映射文件中显示的符号表的区别在于，它们还会列出局部符号，并仅显示在相应模块中定义的符号。

6.3.2.7 函数信息

符号表后是与程序中每个函数有关的信息。该信息与汇编列表文件中显示的函数信息相同。不过，所有函数的信息都集中在一个位置。

6.3.2.8 模块信息

映射文件的最后一部分将显示每个模块的代码使用摘要。对于程序中的每个模块都将显示类似于下面所示的信息。

```
Module      Function          Class         Link    Load    Size
main.c
            init              CODE          07D8    0000    1
            main              CODE          07E5    0000    13
            getInput          CODE          07D9    0000    4
main.c estimated size: 18
```

其中列出了模块名称（上例中为 `main.c`）。特殊模块名称 shared 用于分配到程序存储器以及非特定于任何特定模块的代码的数据对象。

接下来，列出每个模块定义的用户自定义函数和库函数以及该 psect 所在的类、该 psect 的链接地址、装入地址和大小（对于 PIC18 器件以字节为单位显示，对于其他 8 位器件以字为单位显示）。

函数列表之后是该模块使用的程序存储器的估算大小。

7. 实用程序

本章介绍一些与汇编器一起捆绑的实用应用程序。

本章讨论的是较为常用的应用程序，但是通常不需要直接执行它们。这些应用程序的某些功能由基于命令行参数或 MPLAB X IDE 项目属性选择的命令行驱动程序间接调用。

7.1 归档器/库管理器

归档器/库管理器程序具有将几个中间文件组合为单个文件（称为库归档文件）的功能。库归档文件相比于其中包含的各个文件更易于管理，并且占用的磁盘空间可能会更小。

归档器可以构建汇编器所需的所有库归档文件类型，并且可以检测现有归档文件的格式。

7.1.1 使用归档器/库管理器

归档器程序名为 `xc8-ar`，用于创建和编辑库归档文件。它具有以下基本命令格式：

```
xc8-ar [options] file.a [file1.p1 file2.o...]
```

其中，*`file`*`.a` 表示正在创建或编辑的库归档文件。

`options` 是零个或多个用于控制程序的选项，如下表所示。

表 7-1. 归档器命令行选项

选项	作用
`-d`	删除模块
`-m`	重新排序模块
`-p`	列出模块
`-r`	替换模块
`-t`	列出模块与符号
`-x`	提取模块
`--target`	指定目标器件

在替换或提取模块时，必须指定要替换或提取的模块的名称。如果未提供任何名称，将替换或提取归档中的所有模块。

通过请求归档器替换归档中的模块来创建归档文件或将文件添加到现有归档。由于模块并不存在，所以会将该模块追加到归档的末尾。可以将目标模块和 p 代码模块添加到同一归档中。归档器将按模块在命令行上的顺序使用模块来创建库归档。在更新归档时，将会保留模块的顺序。添加到归档的任何模块都将追加到末尾。

模块在归档中的顺序对于链接器是有意义的。如果归档中包含的某个模块引用同一个归档中另一个模块中定义的符号，定义该符号的模块应位于引用符号的模块之后。

使用`-d` 选项时，将从归档中删除指定的模块。在这种情况下，不提供任何模块名称会产生错误。

`-p` 选项将列出归档文件中的模块。

`-m` 选项可接受模块名称的列表，并会对归档文件中的匹配模块重新排序，使它们的顺序与命令行上列出的模块顺序相同。未列出的模块将保留其现有顺序，并将出现在重新排序的模块之后。

`avr-ar` 归档器不适用于仅使用 LTO 数据编译（即，使用`-fno-fat-lto-objects` 选项编译）的目标文件。对于此类目标文件，请改用 `avr-gcc-ar` 归档器。

7.1.1.1 示例

以下给出了一些库管理器使用的示例。以下命令：

```
xc8-ar -r myPicLib.a ctime.p1 init.p1
```

会创建一个名为 `myPicLib.a` 的库，其中包含模块 `ctime.p1` 和 `init.p1`。

以下命令会从库 `lcd.a` 中删除目标模块 `a.p1`：

```
xc8-ar -d lcd.a a.p1
```

7.2 Hexmate

Hexmate 实用程序是用来操作 Intel HEX 文件的程序。 Hexmate 是一个链接阶段后的实用程序，它由汇编器驱动程序自动调用，并提供了以下功能：

- 计算并存储可变长度的哈希值。
- 使用已知的数据序列填充未用的存储单元。
- 将多个 Intel HEX 文件合并为一个输出文件。
- 将 INHX32 文件转换为其他 INHX 格式（如 INHX8M）。
- 检测 HEX 文件中的特定或部分操作码序列。
- 查找/替换特定或部分操作码序列。
- 提供 HEX 文件中使用的地址的映射。
- 更改或修正 HEX 文件中的数据记录的长度。
- 验证 Intel HEX 文件中的校验和。

Hexmate 的典型应用可能包括：

- 在编译时将自举程序或调试模块合并到主应用程序中。
- 计算某个程序存储器范围的校验和或 CRC 值，并将它的值存储在程序存储器或 EEPROM 中。
- 用一条指令填充未用的存储单元以将程序计数器发送到某个已知地址（如程序计数器丢失）。
- 在某个固定地址处存储序列号。
- 在某个固定地址处存储字符串（如时间戳）。
- 在某个特定存储器地址处存储初始值（如初始化 EEPROM）。
- 检测是否使用了有缺陷/受限的指令。
- 调整 HEX 文件，使之满足特定自举程序的要求。

7.2.1 Hexmate 命令行选项

Hexmate 由命令行驱动程序 `pic-as` 自动调用，用于将命令行上指定的所有 HEX 文件与源文件生成的输出进行合并。某些 hexmate 函数可直接从驱动程序请求，而无需显式地运行 Hexmate，但若要实现完全控制，可以在命令行上运行 hexmate 并使用此处详细介绍的选项。

如果要直接运行 Hexmate，其用法为：

```
hexmate [specs,]file1.hex [...[specs,]fileN.hex] [options]
```

其中，*`file1.hex`* 至 *`fileN.hex`* 构成要使用 Hexmate 合并的输入 Intel HEX 文件的列表。

如果仅指定一个 HEX 文件，则不会发生合并，但会通过其他选项指定其他功能。下表列出了 Hexmate 可接受的命令行选项。

表 7-2. Hexmate 命令行选项

选项	作用
`--edf`	指定消息描述文件。

.....（续）

选项	作用
--emax	设置在终止之前允许的最大错误数。
--msgdisable	禁止具有指定编号的消息。
--sla	设置类型 5 记录的起始线性地址。
--ver	显示版本和内部版本信息，然后退出。
-addressing	将所有 hexmate 选项中的地址字段设置为使用字寻址或其他寻址模式。
-break	拆分连续数据，从而使新记录在所设置地址处开始。
-ck	计算并存储值。
-fill	使用已知值编程未用存储单元。
-find	进行搜索，并在检测到特定代码序列时发出通知。
-find...,delete	如果检测到该代码序列则删除它（慎用）。
-find...,replace	使用新的代码序列替换代码序列。
-format	指定最大数据记录长度或选择 INHX 形式。
-help	显示所有选项或显示特定选项的帮助消息。
-logfile	将 hexmate 输出分析和各种结果保存到文件中。
-mask	将某个存储器范围与位掩码进行逻辑与运算。
-o*file*	指定输出文件的名称。
-serial	在某个固定地址处存储序列号或代码序列。
-size	报告生成的 HEX 映像中包含的数据字节数。
-string	在某个固定地址处存储 ASCII 字符串。
-strpack	使用字符串压缩在某个固定地址处存储 ASCII 字符串。
-w	调整警告灵敏度。
+	用作任意选项的前缀，以覆盖其地址范围中的其他数据（如需要）。

如果使用驱动程序 pic-as（或 IDE）编译项目，则默认情况下会生成一个日志文件，它将使用项目名称和扩展名.hxl。

现在将详细地介绍 Hexmate 的输入参数。介绍每个选项时都会顺带介绍该选项相关值的格式或使用的基数。请注意，除非在-addressing 选项中另外指定，否则这些选项中指定的任何地址字段都需要以字节地址的形式输入。

7.2.1.1 规范和文件名

Hexmate 可以处理使用 INHX32 或 INHX8M 格式的 Intel HEX 文件。可以对每个 HEX 文件应用其他规范，以指定关于应如何处理该文件的限制或条件。

如果使用了任何规范，则它们必须位于文件名之前。之后通过逗号将规范列表与文件名进行分隔。

可以使用规范 r*Start*-*End* 来应用范围限制，其中 *Start* 和 *End* 均假定为十六进制值。范围限制将导致仅使用该范围内的地址数据。例如：

```
r100-1FF,myfile.hex
```

将使用 myfile.hex 作为输入，但仅处理该文件中地址处于范围 100h-1FFh（含 100h 和 1FFh）内的数据。

可以通过规范 *sOffset* 来应用地址偏移。如果使用地址偏移，则在生成输出时从该 HEX 文件中读取的数据将移动（按指定的偏移量）到一个新的地址。偏移量可以为正值或负值。例如：

```
r100-1FFs2000,myfile.HEX
```

会将数据块从 100h-1FFh 移动到新的地址范围 2100h-21FFh。

移动可执行代码段时，需要小心。只有程序代码是位置无关的代码时，才能对它进行移动。

7.2.1.2 覆盖前缀

当参数或输入文件之前带有+操作符时，从该数据源获取的数据会被强制添加到输出文件中，并覆盖该地址范围内的其他现有数据。例如：

```
input.HEX +-string@1000="My string"
```

会将-STRING 选项指定的数据放置在地址 1000 处；但：

```
+input.HEX -string@1000="My string"
```

会将 hex 文件中地址 1000 处包含的数据复制到最终输出中。

通常，如果两个数据源尝试在同一位置存储不同数据，Hexmate 会发出错误。使用+操作符可通知 Hexmate，如果多个数据源尝试将数据存储到同一地址，使用+前缀指定的数据源将具有优先权。

7.2.1.3 Edf

--edf=*file* 指定显示警告或错误消息时要使用的消息描述文件。该参数应是消息文件的完整路径。Hexmate 包含该文件的内部副本，因此不需要使用该选项，但是您可能希望指定一个包含更新内容的文件。

消息文件位于安装目录中的 pic/dat 目录下（例如，英语文件名为 en_msgs.txt）。

7.2.1.4 Emax

--emax=*num* 选项设置执行终止之前 Hexmate 将显示的最大错误数，例如--emax=25。默认情况下，最多显示 20 条错误消息。

7.2.1.5 Msgdisable

--msgdisable=*number* 选项用于在 Hexmate 执行过程中禁止错误、警告或建议性消息。

将向该选项传递要禁止的消息编号的逗号分隔列表。除非后跟:off 参数，否则该列表中的所有错误消息编号都将被忽略。如果将消息列表指定为 0，则会禁止所有警告。

7.2.1.6 Sla

--sla=*address* 选项用于为 HEX 输出文件中的类型 5 记录指定线性起始地址，例如--sla=0x10000。

7.2.1.7 Ver

--ver 选项会要求 Hexmate 打印版本和内部版本的信息，然后退出。

7.2.1.8 寻址

该选项用于更改 Hexmate 命令行选项中任何地址的寻址单元。

默认情况下，Hexmate 选项中的所有地址参数都预期值将以 Intel HEX 文件中指定的字节地址的形式输入。在一些器件架构中，本机寻址格式可能不是字节寻址。在这些情况下，该选项可用于将面向器件的地址与 Hexmate 的命令行选项一起使用。

该选项接受一个参数，该参数配置每个地址单元包含的字节数。对于低档、中档和 24 位 PIC 器件，如果需要，可以使用 2 字节的寻址单元。对于所有其他器件，通常将使用 1 字节的默认寻址单元。

7.2.1.9 Break

-break 选项接受一个逗号分隔的地址列表。如果在 HEX 文件中遇到其中任何地址，当前数据记录将结束，新的数据记录将从指定地址处重新开始。这可用于使用新的数据记录来强制区分程序空间中的不同功能区域。一些 HEX 文件阅读器依赖于它。

7.2.1.10 Ck

`-ck` 选项用于计算哈希值。该选项的用法如下：

```
-ck=start-end@dest [+offset][wWidth][tCode][gAlgorithm][pPolynomial][rwidth]
```

其中：

- *start* 和 *end* 指定用于计算哈希值的地址范围。如果这些地址不是算法宽度的倍数，则会将零值填充到缺失的相关输入字单元中。
- *dest* 是将存储哈希结果的地址。该值不能处于用于计算的地址范围内。
- *offset* 是在计算中使用的可选初始值。
- *Width* 是可选的，它指定结果的宽度。对于大多数算法，计算结果可以为 1 到 4 字节的字节宽度，但对于 SHA 算法，则为位宽度。如果请求一个正宽度，则结果将以大尾数法字节顺序存储。负宽度将导致结果以小尾数法字节顺序存储。如果宽度未指定，则结果将为 2 字节宽，并以小尾数法字节顺序存储。如果您选择了任何 Fletcher 算法，则不使用该 width 参数。
- *Code* 是结果中尾随在每个字节之后的十六进制代码。这样便可将哈希结果的每个字节嵌入指令中。例如，在中档器件上，`code=34` 会将结果嵌入到 `retlw` 指令中。
- *Algorithm* 是一个整数，用以选择要用于计算结果的 Hexmate 哈希算法。表 7-3 列出了可选的算法。如果未指定，则使用的默认算法为 8 位校验和加法（算法 1）。
- *Polynomial* 是一个十六进制值，它是选择 CRC 算法时要使用的多项式。
- *r* 是十进制字宽。如果它不为零，则在计算哈希值时，将以相反的顺序读取每个字中的字节。目前，宽度必须为 0 或 2。零宽度将禁止反向字节功能，就像不存在 `r` 子选项一样。使用 Hexmate 与由 PIC 硬件 CRC 模块生成的 CRC 进行匹配时，应使用该子选项（PIC 硬件 CRC 模块使用扫描器模块将数据流传输至 Hexmate）。

除算法选择器和结果宽度假定为十进制值外，所有数字参数均假定为十六进制值。

使用校验和选项的典型示例为：

```
-ck=0-1FFF@2FFE+2100w2g2
```

这将计算范围 0-0x1FFF 的校验和，并将校验和结果写入地址 0x2FFE。校验和值将偏移 0x2100。结果将为 2 字节宽。

表 7-3. Hexmate 哈希算法选择

选择器	算法说明
-5	反射循环冗余校验（Cyclic Redundancy Check，CRC）。
-4	初始值减去 32 位值。
-3	初始值减去 24 位值。
-2	初始值减去 16 位值。
-1	初始值减去 8 位值。
1	初始值加上 8 位值。
2	初始值加上 16 位值。
3	初始值加上 24 位值。
4	初始值加上 32 位值。
5	循环冗余校验（CRC）。
7	Fletcher 校验和（8 位计算，2 字节结果宽度）。
8	Fletcher 校验和（16 位计算，4 字节结果宽度）。
10	SHA-2（当前仅支持 SHA256）

有关用于计算校验和的算法的更多详细信息，请参见 7.2.2 哈希函数。

7.2.1.11 Fill

-fill 选项用于以已知值填充未用的存储单元。该选项的用法如下：

```
-fill=[const_width:]fill_expr@address[:end_address]
```

其中：

- const_width 的形式为 *wn*，指示 fill_expr 中每个常量的宽度（*n* 字节）。如果未指定 const_width，则默认值为 2 字节。例如，-fill=w1:1 将使用值 0x01 填充每个未使用的字节。
- fill_expr 可以使用以下语法（其中，const 和 increment 均为 n 字节常量）：
  - *const* 会使用重复的常量填充存储器；即，-fill=0xBEEF 将变为 0xBEEF、0xBEEF、0xBEEF 和 0xBEEF。
  - *const*+=*increment* 会使用递增的常量填充存储器；即，-fill=0xBEEF+=1 将变为 0xBEEF、0xBEF0、0xBEF1 和 0xBEF2。
  - *const*-=*increment* 会使用递减的常量填充存储器；即， -fill=0xBEEF-=0x10 将变为 0xBEEF、0xBEDF、0xBECF 和 0xBEBF。
  - *const*,*const*,...,*const* 将使用重复的常量的列表填充存储器；即，-fill=0xDEAD,0xBEEF 将变为 0xDEAD、0xBEEF、0xDEAD 和 0xBEEF。
- fill_expr 之后的选项会产生以下行为：
  - @*address* 将使用 fill_expr 填充一个特定地址；即，-fill=0xBEEF@0x1000 会在地址 1000h 处放入 0xBEEF。如果填充值的宽度大于-addressing 指定的寻址值，则仅将填充值的一部分放入输出中。例如，如果寻址设置为 1，则上述选项会将 0xEF 放入地址 0x1000，并将发出警告。
  - @*address*:*end_address* 将使用 fill_expr 填充存储器范围；即，-fill=0xBEEF@0:0xFF 会在 0 和 255 之间的未用地址中放入 0xBEEF。如果地址范围（乘以-ADDRESSING 值）不是填充值宽度的倍数，则最终位置将仅使用填充值的一部分，并会发出警告。

填充值是按字对齐的，因此它们从数倍于填充宽度的地址处开始。如果填充值是指令操作码，则这种对齐方式将确保可以正确执行指令。同样，如果填充序列的总长度大于 1（即使指定的宽度为 1），则填充序列将与该总长度对齐。以下面的填充选项为例，它指定的填充序列长度为 2 字节，起始地址不是 2 的倍数：

```
-fill=w1:0x11,0x22@0x11001:0x1100c
```

将产生以下十六进制记录，其中起始地址已按照上述对齐方式用填充序列的第二个字节填充。

```
:0C1001002211221122112211221122112211B1
```

所有常量可以按正常 C 语法使用（无符号）二进制、八进制，十进制或十六进制表示，例如，1234 是十进制值，0xFF00 是十六进制值，FF00 是非法的。

7.2.1.12 查找

-find=*opcode* 选项用于检测并记录某个操作码或部分代码序列的出现。该选项的用法如下：

```
-find=Findcode [mMask]@Start-End [/Align][w][t"Title"]
```

其中：

- *Findcode* 是要搜索的十六进制代码序列。例如，要查找操作码为 0x01F1 的 clrf 指令，请使用 01F1 作为序列。在 HEX 文件中，它将显示为字节序列 F1 01，即 F1 位于十六进制地址 0 处，01 位于十六进制地址 1 处。
- *Mask* 是可选的。它指定应用于 *Findcode* 值的位掩码，以进行限制较少的搜索。它以小尾数法字节顺序输入。
- Start 和 *End* 限制要搜索的地址范围。
- *Align* 是可选的。它指定只有代码序列的起始地址位于该值的倍数地址时，才会匹配。
- w（如存在）将导致 Hexmate 在每次检测到代码序列时发出警告。
- *Title* 是可选的。它用于为该代码序列提供标题。定义标题将使日志报告和消息更具描述性、可读性更强。标题不会影响实际的搜索结果。

所有数字参数均假定为十六进制值。

以下给出了一些示例。

选项-find=1234@0-7FFF/2w 将检测 0h 和 7FFFh 之间，在 2 字节地址边界对齐的代码序列 1234h。w 指示将在每次发现该序列时发出一条警告。

在下一个示例-find=1234M0F00@0-7FFF/2wt"ADDXY"中，选项与上一个示例相同，但所匹配的代码序列使用 000Fh 作为掩码，所以 Hexmate 将搜索所有 123*x*h 操作码（其中，*x* 是任意数字）。如果使用字节掩码，则它的字节宽度必须与应用它的操作码相等。由 Hexmate 生成的所有消息或报告都会使用名称 ADDXY 来指代该操作码，因为它是为该搜索定义的标题。

如果 Hexmate 生成日志文件，它将包含所有搜索的结果。-find 接受长度为 1 至 8 字节的完整 HEX 数据字节。（可选）-find 可以与 replace 或 delete 配合使用（如下所述）。

7.2.1.13 查找并删除

如果使用了 delete 形式的-find 选项，则会删除所有匹配的序列。该功能应慎用，通常建议不要用于删除可执行代码。

7.2.1.14 查找并替换

如果使用了 replace 形式的-find 选项，则会使用新代码替换或部分替换所有匹配的序列。该子选项的用法如下：

```
-find...,replace=Code [mMask]
```

其中：

- *Code* 是用于替换匹配-find 条件的序列的十六进制代码序列。
- *Mask* 是可选的位掩码，指定 *Code* 内的哪些位将替换匹配的代码序列。举例来说，如果只需修改 16 位指令内的 4 位，这会很有用。其余 12 位可以被掩码掉而保留不变。

7.2.1.15 格式

-format 选项可用于指定 INHX 格式的特定形式或调整最大记录长度。该选项的用法如下：

```
-format=Type [,Length]
```

其中：

- *Type* 指定要生成的特定 INHX 格式。
- *Length* 是可选的，它设置每个数据记录的最大字节数。有效长度介于 1 和 16（十进制）之间，16 为默认值。

假设存在这种情形：某个自举程序尝试下载一个 INHX32 文件，但发生失败，因为它无法处理属于 INHX32 标准一部分的扩展地址记录。该自举程序只能编程地址在 0 至 64k 范围内的数据，HEX 文件中任何超出该范围的数据都可以安全地忽略。在这种情况下，通过以 INHX8M 格式生成 HEX 文件，该操作可能会成功。执行该操作的 Hexmate 选项为-FORMAT=INHX8M。

现在，假设同一自举程序还要求每个数据记录包含恰好 8 个字节的数据。这可以通过结合使用-format 与-fill 选项实现。适当使用-fill 可以确保要进行编程的地址范围内的数据中不存在空隙。这将满足最小数据长度要求。要将数据记录的最大长度设置为 8 字节，只需将前一个选项修改为-format=INHX8M,8。

表 7-4 列出了该选项支持的可能类型。请注意，INHX032 并不是实际的 INHX 格式。选择该类型会生成一个 INHX32 文件，但也会将高位地址信息初始化为零。 这是一些器件编程器的要求。

表 7-4. Inhx 类型

类型	说明
INHX8M	无法编程超出 64K 之外的地址。
INHX32	可以使用扩展线性地址记录编程超出 64K 之外的地址。
INHX032	INHX32，并将高位地址初始化为零。

7.2.1.16 帮助

使用-help 将列出所有 Hexmate 选项。输入另一个 Hexmate 选项作为-help 的参数时，将显示给定选项的详细帮助消息。例如：

```
-help=string
```

将显示-string Hexmate 选项的附加帮助。

7.2.1.17 Logfile

-logfile 选项会将 HEX 文件统计信息保存到指定文件中。例如：

```
-logfile=output.hxl
```

将分析 Hexmate 生成的 HEX 文件，并将报告保存到名为 output.hxl 的文件中。

7.2.1.18 掩码

-mask=*spec* 选项可将某个存储器范围与特定位掩码进行逻辑与运算。它用于确保程序字中的未实现位（如果有）保留为空。该选项的用法如下：

```
-mask=hexcode@start-end
```

其中，*hexcode* 是一个将与 *start* 至 *end* 地址范围内的数据进行与运算的值。所有值均假定为十六进制。多字节掩码值可以使用小尾数法字节顺序输入。

7.2.1.19 O：指定输出文件

使用-o*file* 选项时，将在该文件中创建生成的 Intel HEX 输出。例如：

```
-oprogram.hex
```

会将产生的输出保存到 program.hex。输出文件可以采用与其输入文件之一相同的名称，但这样做时，它将完全替换该输入文件。

7.2.1.20 序列

-serial=*specs* 选项将在某个固定地址处存储特定 HEX 值序列。该选项的用法如下：

```
-serial=Code[+/-Increment]@Address[+/-Interval][rRepetitions]
```

其中：

- *Code* 是要存储的十六进制序列。指定的第一个字节存储在最低地址。
- *Increment* 是可选的，并允许在每次重复（如要求）时根据该值更改 Code 的值。
- *Address* 是用于存储该代码的存储单元，或第一次重复使用的存储单元。
- *Interval* 是可选的，它指定每次重复该代码时的地址偏移。
- *Repetitions* 是可选的，它指定重复该代码的次数。

除 *Repetitions* 参数默认为十进制值外，所有数字参数均假定为十六进制值。

例如：

```
-serial=000001@EFFE
```

会将 HEX 代码 00001h 存储到地址 EFFEh。

另一个示例：

```
-serial=0000+2@1000+10r5
```

将从值 0000 开始在地址 1000h 处存储 5 个代码。后续的代码将出现在+10h 地址间隔处，代码值将按增量+2h 变化。

7.2.1.21 大小

使用-size 选项将在标准输出中报告所产生的 HEX 映像中的数据字节数。如果请求使用日志文件，长度还会被记录在日志文件中。

7.2.1.22 字符串

-string 选项将在某个固定地址处嵌入 ASCII 字符串。该选项的用法如下：

```
-string@Address[tCode]="Text"
```

其中：

- *Address* 假定为十六进制值，代表将存储字符串的地址。
- *Code* 是可选的，用于在字符串每个字节之后尾随某个字节序列。这可以用于将字符串字节编码到指令内。
- *Text* 是要转换为 ASCII 并嵌入的字符串。

例如：

```
-string@1000="My favorite string"
```

将在地址 1000h 处存储字符串 My favorite string（包括 null 终止符）的 ASCII 数据。

同样：

```
-string@1000t34="My favorite string"
```

将存储相同的字符串，字符串中的每个字节都尾随 HEX 代码 34h。

7.2.1.23 Strpack

-strpack=*spec* 选项执行与-string 相同的功能，但存在两个重要的区别。首先，它仅存储每个字符的低 7 位。然后，7 位字符对将连接在一起，并以 14 位字而不是独立字节的形式进行存储。这被称为字符串压缩。这通常仅对于程序空间以 14 位字进行寻址的器件（PIC 器件）有用。第二个区别是-string 的 t 说明符不适用于-strpack 选项。

7.2.2 哈希函数

哈希值是一个长度固定的较小值，从任意长度的数据块中的所有值计算得出并用于表示这些值。如果复制该数据块，则会为新数据块重新计算哈希值并与原哈希值进行比较。如果两个哈希值一致，即可充分确定复制有效性。哈希算法有很多种。算法越复杂，验证就越可靠，但在嵌入式环境中使用时可能会占用过多资源。

Hexmate 可用于计算由 MPLAB XC8 PIC 汇编器编译的 HEX 文件中包含的程序映像的哈希值。该哈希值可以嵌入到该 HEX 文件中，并与程序映像一起烧写到目标器件中。在运行时，目标器件可对当前存储在其存储器中的程序映像运行类似的哈希算法。如果计算得出的哈希值与存储的哈希值相同，嵌入式程序即可认定要执行的程序映像是有效的。

Hexmate 实现了多种校验和与循环冗余校验算法来计算哈希值。如果您使用 pic-as 驱动程序执行项目编译，则驱动程序的-mchecksum 选项将指示驱动程序调用 Hexmate 并传递适当的 Hexmate 选项。MPLAB X IDE 中也提供该选项。如果要显式地驱动 Hexmate，请参见 7.2.1.10 Ck 中用于选择算法的选项。在下面的算法讨论中，假定您正在使用汇编器驱动程序来请求校验和或 CRC。

当程序映像包含未使用的存储单元时，需要考虑一些因素。驱动程序的-mchecksum 选项自动请求 Hexmate 填充未使用的存储单元，以匹配未编程的器件存储器。如果显式地调用 Hexmate，则可能需要模拟该操作。

尽管 Hexmate 适用于所有器件，但并非所有器件都可以读取其程序存储器的整个宽度。中档 PIC 器件也使用 14 位宽的程序存储器，因此不能直接存储大于 1 字节的哈希值。对于这些器件，通常会将 code=*nn* 参数用于-mcodeoffset 选项，以将哈希值的每个字节封装在一条指令中。

以下各节提供了可用于在运行时计算哈希值的算法示例，但请注意，并非所有器件都能够直接使用这些示例。

7.2.3 加法算法

Hexmate 有几种简单的校验和算法可对程序映像中某个范围内的数据值求和。这些算法分别对应于算法子选项中的选择器值 1、2、3 和 4，并分别以 1、2、3 或 4 字节宽度读取程序映像中的数据。该总和将与通过相同选项提供给算法的初始值（偏移量）相加。该选项还指定最终校验和截断后的宽度，具体可以是 1、2、3 或 4 字节。 Hexmate 会自动将 HEX 文件中的校验和存储在校验和选项中指定的地址处。

如下所示的函数可根据不同的数据宽度（readType）与校验和宽度（resultType）组合进行自定义。

```
typedef unsigned char readType; // size of data values read and summed
typedef unsigned int  resultType; // size of checksum result
// add to offset n additions of values starting at address data,
// truncating and returning the result
// data: the address of the first value to sum
// n:    the number of sums to perform
// offset: the intial value to which the sum is added
resultType ck_add(const readType *data, unsigned n, resultType offset)
{
    resultType chksum;
    chksum = offset;
    while(n--) {
        chksum += *data;
        data++;
    }
    return chksum;
}
```

readType 和 resultType 类型定义应分别根据数据读取/求和宽度与校验和结果宽度进行调整。使用 MPLAB XC8 时，如果宽度为 1，请使用 char 类型；如果宽度为 4，请使用 long 类型，等等；也可使用<stdint.h>提供的具体宽度类型。如果从不使用偏移量，则可以删除该参数，并在循环前为 chksum 赋值 0。

下文以通过将 0x100 至 0x7fd 地址范围（起始偏移量为 0x20）内的 1 字节值相加来计算 2 字节校验和为例，说明该函数的使用方法。校验和将以小尾数法格式存储在 0x7fe 和 0x7ff 中。在编译项目时指定以下选项。 在 MPLAB X IDE 中，仅在 **XC8 Linker**（XC8 链接器）类别的 **Additional options**（附加选项）选项类别的 **Checksum**（校验和）字段中的第一个=右侧输入信息。

```
-mchecksum=100-7fd@7fe,offset=20,algorithm=1,width=-2
```

在您的项目中，添加以下代码片段，该代码片段调用 ck_add()并将运行时校验和与编译时 Hexmate 存储的校验和进行比较。

```
extern const readType ck_range[0x6fe/sizeof(readType)] __at(0x100);
extern const resultType hexmate __at(0x7fe);
resultType result;
result = ck_add(ck_range, sizeof(ck_range)/sizeof(readType), 0x20);
if(result != hexmate)
    ck_failure();  // take appropriate action
```

该代码使用占位符数组 ck_range 来表示用于计算校验和的存储器范围，并将变量 hexmate 映射到供 Hexmate 存储其校验和结果的存储单元。作为 extern 和绝对对象时，这些对象均不会占用额外的器件存储空间。这些对象的地址和宽度应根据传递给 Hexmate 的选项进行调整。

Hexmate 可以计算任何地址范围内的校验和；但是，测试函数 ck_add()假定求和范围的起始地址和结束地址是 readType 宽度的倍数。如果 readType 的宽度为 1，则不成问题。建议您在指定校验和计算范围时遵循该前提，否则您将需要修改测试代码以执行对起始和/或结束数据值的部分读取。这将大大增加代码的复杂性。

7.2.4 减法算法

Hexmate 有几种校验和算法可将程序映像中某个范围内的数据值相减。这些算法分别对应于算法子选项中的选择器值-1、-2、-3 和-4，并分别以-1、-2、-3 或-4 字节宽度读取程序映像中的数据。除此之外，这些算法与 7.2.3 加法算法 所述的加法算法完全相同。

如下所示的函数可根据不同的数据宽度（readType）与校验和宽度（resultType）组合进行自定义。

```
typedef unsigned char readType; // size of data values read and summed
typedef unsigned int  resultType; // size of checksum result
// add to offset n subtractions of values starting at address data,
// truncating and returning the result
// data: the address of the first value to subtract
// n:    the number of subtractions to perform
// offset: the intial value to which the subtraction is added
resultType ck_sub(const readType *data, unsigned n, resultType offset)
{
    resultType chksum;
```

```
    chksum = offset;
    while(n--) {
        chksum -= *data;
        data++;
    }
    return chksum;
}
```

下文以通过将 0x0 至 0x7fd 地址范围（起始偏移量为 0x0）内的 2 字节值相加来计算 4 字节校验和为例，说明该函数的使用方法。校验和将以小尾数法格式存储在 0x7fe 和 0x7ff 中。在编译项目时指定以下选项。在 MPLAB X IDE 中，仅在 **XC8 Linker** 类别的 **Additional options** 选项类别的 **Checksum** 字段中的第一个=右侧输入信息。

```
-mchecksum=0-7fd@7fe,offset=0,algorithm=-2,width=-4
```

在您的项目中，添加以下代码片段，该代码片段调用 `ck_sub()` 并将运行时校验和与编译时 Hexmate 存储的校验和进行比较。

```
extern const readType ck_range[0x7fe/sizeof(readType)] __at(0x0);
extern const resultType hexmate __at(0x7fe);
resultType result;
result = ck_sub(ck_range, sizeof(ck_range)/sizeof(readType), 0x0);
if(result != hexmate)
    ck_failure();  // take appropriate action
```

7.2.5 Fletcher 算法

Hexmate 有几种算法可实现 Fletcher 校验和计算。这些算法较为复杂，稳健性接近循环冗余校验，但计算量相对较少。这类算法有两种形式，分别对应于算法子选项中的选择器值 7（执行 1 字节计算并生成 2 字节结果）和 8（执行 2 字节计算并生成 4 字节结果）。 Hexmate 会自动将 HEX 文件中的校验和存储在校验和选项中指定的地址处。

如下所示的函数执行 1 字节的加法并生成 2 字节的结果。

```
unsigned int
fletcher8(const unsigned char * data, unsigned int n )
{
    unsigned int sum = 0xff, sumB = 0xff;
    unsigned char tlen;
    while (n) {
        tlen = n > 20 ? 20 : n;
        n -= tlen;
        do {
            sumB += sum += *data++;
        } while (--tlen);
        sum = (sum & 0xff) + (sum >> 8);
        sumB = (sumB & 0xff) + (sumB >> 8);
    }
    sum = (sum & 0xff) + (sum >> 8);
    sumB = (sumB & 0xff) + (sumB >> 8);
    return sumB << 8 | sum;
}
```

可按照与加法算法类似的方式调用该代码（见 7.2.3 加法算法）。

下面给出了生成 4 字节结果的 2 字节加法 Fletcher 算法的代码。

```
unsigned long
fletcher16(const unsigned int * data, unsigned n)
{
    unsigned long sum = 0xffff, sumB = 0xffff;
    unsigned tlen;
    while (n) {
        tlen = n > 359 ? 359 : n;
        n -= tlen;
        do {
            sumB += sum += *data++;
        } while (--tlen);
        sum = (sum & 0xffff) + (sum >> 16);
        sumB = (sumB & 0xffff) + (sumB >> 16);
    }
    sum = (sum & 0xffff) + (sum >> 16);
```

```
    sumB = (sumB & 0xffff) + (sumB >> 16);
    return sumB << 16 | sum;
}
```

7.2.6 CRC 算法

Hexmate 有几种算法可实现稳健的循环冗余校验（CRC）。具体有两种算法可供选择，分别对应于-mchecksum 的算法子选项中的选择器值 5（执行 CRC 计算）和-5（执行反射 CRC 计算）。反射算法首先处理数据的最低有效位。

可以在选项中指定要使用的多项式和初始值。 Hexmate 会自动将 CRC 结果存储在 HEX 文件中、校验和选项中指定的地址处。

一些器件在硬件中实现了 CRC 模块，可用于在运行时计算 CRC。这些模块可以使用扫描器模块来传输从程序存储器读取的数据流。为了确保 Hexmate 和 CRC/扫描器模块按照相同的顺序处理字节，必须使用子选项 revword=2 将保留字宽度指定为 2。这样便可确保按顺序读取 HEX 文件中的每个 2 字节字，但以相反的顺序处理这些字中的字节。

如下所示的函数可自定义为处理任意的结果宽度（resultType）。它使用 POLYNOMIAL 宏指定的多项式来计算 CRC 哈希值。

```
typedef unsigned int resultType;
#define POLYNOMIAL     0x1021
#define WIDTH   (8 * sizeof(resultType))
#define MSb     ((resultType)1 << (WIDTH - 1))

resultType
crc(const unsigned char * data, unsigned n, resultType remainder) {
    unsigned pos;
    unsigned char bitp;
    for (pos = 0; pos != n; pos++) {
        remainder ^= ((resultType)data[pos] << (WIDTH - 8));
        for (bitp = 8; bitp > 0; bitp--) {
            if (remainder & MSb) {
                remainder = (remainder << 1) ^ POLYNOMIAL;
            } else {
                remainder <<= 1;
            }
        }
    }
    return remainder;
}
```

resultType 类型定义应根据结果宽度进行调整。使用 MPLAB XC8 时，如果宽度为 1，请使用 char 类型；如果宽度为 4，请使用 long 类型，等等；也可使用<stdint.h>提供的具体宽度类型。

下文以在 0x0 至 0xFF 地址范围（起始初始值为 0xFFFF）内计算 2 字节 CRC 哈希值为例，说明该函数的使用方法。结果将以小尾数法格式存储在 0x100 和 0x101 中。在编译项目时指定以下选项。在 MPLAB X IDE 中，仅在 **XC8 Linker** 类别的 **Additional options** 选项类别的 **Checksum** 字段中的第一个=右侧输入信息。

```
-mchecksum=0-FF@100,offset=0xFFFF,algorithm=5,width=-2,polynomial=0x1021
```

在您的项目中，添加以下代码片段，该代码片段调用 crc()并将运行时哈希结果与编译时 Hexmate 存储的哈希结果进行比较。

```
extern const unsigned char ck_range[0x100] __at(0x0);
extern const resultType hexmate __at(0x100);
resultType   result;

result = crc(ck_range, sizeof(ck_range), 0xFFFF);
if(result != hexmate){
    // something's not right, take appropriate action
    ck_failure();
}
// data verifies okay, continue with the program
```

反射 CRC 结果可通过以下两种方式来计算：反射输入数据和最终结果，或者反射多项式。如下所示的函数可自定义为处理任意的结果宽度（resultType）。crc_reflected_IO()函数通过反射数据流位位置来计算反射 CRC 哈希

值，而 crc_reflected_poly()函数通过反射多项式（不调整数据流）来计算反射 CRC 哈希值。这两个函数中均通过 POLYNOMIAL 宏指定多项式，并且均使用 reflect()函数执行位反射。

```
typedef unsigned int resultType;
typedef unsigned char readType;
typedef unsigned int reflectWidth;
// This is the polynomial used by the CRC-16 algorithm we are using.
#define POLYNOMIAL     0x1021
#define WIDTH   (8 * sizeof(resultType))
#define MSb     ((resultType)1 << (WIDTH - 1))
#define LSb     (1)
#define REFLECT_DATA(X)       ((readType) reflect((X), 8))
#define REFLECT_REMAINDER(X)  (reflect((X), WIDTH))

reflectWidth
reflect(reflectWidth data, unsigned char nBits)
{
    reflectWidth reflection = 0;
    reflectWidth reflectMask = (reflectWidth)1 << nBits - 1;
    unsigned char bitp;
    for (bitp = 0; bitp != nBits; bitp++) {
        if (data & 0x01) {
            reflection |= reflectMask;
        }
        data >>= 1;
        reflectMask >>= 1;
    }
    return reflection;
}

resultType
crc_reflected_IO(const unsigned char * data, unsigned n, resultType remainder) {
    unsigned pos;
    unsigned char reflected;
    unsigned char bitp;
    for (pos = 0; pos != n; pos++) {
        reflected = REFLECT_DATA(data[pos]);
        remainder ^= ((resultType)reflected << (WIDTH - 8));
        for (bitp = 8; bitp > 0; bitp--) {
            if (remainder & MSb) {
                remainder = (remainder << 1) ^ POLYNOMIAL;
            } else {
                remainder <<= 1;
            }
        }
    }
    remainder = REFLECT_REMAINDER(remainder);
    return remainder;
}

resultType
crc_reflected_poly(const unsigned char * data, unsigned n, resultType remainder) {
    unsigned pos;
    unsigned char bitp;
    resultType rpoly;
    rpoly = reflect(POLYNOMIAL, WIDTH);
    for (pos = 0; pos != n; pos++) {
        remainder ^= data[pos];
        for (bitp = 8; bitp > 0; bitp--) {
            if (remainder & LSb) {
                remainder = (remainder >> 1) ^ rpoly;
            } else {
                remainder >>= 1;
            }
        }
    }
    return remainder;
}
```

下文以在 0x0 至 0xFF 地址范围（起始初始值为 0xFFFF）内计算 2 字节反射 CRC 结果为例，说明该函数的使用方法。结果将以小尾数法格式存储在 0x100 和 0x101 中。在编译项目时指定以下选项（请注意，本例中所选的算法为-5）。

```
-mchecksum=0-FF@100,offset=0xFFFF,algorithm=-5,width=-2,polynomial=0x1021
```

在您的项目中，调用 `crc_reflected_IO()`或 `crc_reflected_poly()`函数，如前文所示。

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。我们的网站提供以下内容：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题解答（FAQ）、技术支持请求、在线讨论组以及 Microchip 设计伙伴计划成员名单
- **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

产品变更通知服务

Microchip 的产品变更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，请访问 www.microchip.com/pcn，然后按照注册说明进行操作。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

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

客户应联系其代理商、代表或 ESE 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 www.microchip.com/support 获得网上技术支持。

Microchip 器件代码保护功能

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规范。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品非常安全。
- 目前，仍存在着用恶意、甚至是非法的方法来试图破坏代码保护功能的行为。我们确信，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。这种试图破坏代码保护功能的行为极可能侵犯 Microchip 的知识产权。
- 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。

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。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.的英文原版文档。

本出版物中提供的信息仅仅是为方便您使用 Microchip 产品或使用这些产品来进行设计。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“按原样”提供这些信息。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，包括但不限于针对非侵权性、适销性和特定用途的适用性的暗示担保，或针对其使用情况、质量或性能的担保。

在任何情况下，对于因这些信息或使用这些信息而产生的任何间接的、特殊的、惩罚性的、偶然的或间接的损失、损害或任何类型的开销，MICROCHIP 概不承担任何责任，即使 MICROCHIP 已被告知可能发生损害或损害可以预见。在法律允许的最大范围内，对于因这些信息或使用这些信息而产生的所有索赔，MICROCHIP 在任何情况下所承担的全部责任均不超出您为获得这些信息向 MICROCHIP 直接支付的金额（如有）。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切损害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任。除非另外声明，在 Microchip 知识产权保护下，不得暗中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、Adaptec、AnyRate、AVR、AVR 徽标、AVR Freaks、BesTime、BitCloud、chipKIT、chipKIT 徽标、CryptoMemory、CryptoRF、dsPIC、FlashFlex、flexPWR、HELDO、IGLOO、JukeBlox、KeeLoq、Kleer、LANCheck、LinkMD、maXStylus、maXTouch、MediaLB、megaAVR、Microsemi、Microsemi 徽标、MOST、MOST 徽标、MPLAB、OptoLyzer、PackeTime、PIC、picoPower、PICSTART、PIC32 徽标、PolarFire、Prochip Designer、QTouch、SAM-BA、SenGenuity、SpyNIC、SST、SST 徽标、SuperFlash、Symmetricom、SyncServer、Tachyon、TempTrackr、TimeSource、tinyAVR、UNI/O、Vectron 和 XMEGA 均为 Microchip Technology Incorporated 在美国和其他国家或地区的注册商标。

APT、ClockWorks、The Embedded Control Solutions Company、EtherSynch、FlashTec、Hyper Speed Control、HyperLight Load、IntelliMOS、Libero、motorBench、mTouch、Powermite 3、Precision Edge、ProASIC、ProASIC Plus、ProASIC Plus 徽标、Quiet-Wire、SmartFusion、SyncWorld、Temux、TimeCesium、TimeHub、TimePictra、TimeProvider、Vite、WinPath 和 ZL 均为 Microchip Technology Incorporated 在美国的注册商标。

Adjacent Key Suppression、AKS、Analog-for-the-Digital Age、Any Capacitor、AnyIn、AnyOut、BlueSky、BodyCom、CodeGuard、CryptoAuthentication、CryptoAutomotive、CryptoCompanion、CryptoController、dsPICDEM、dsPICDEM.net、Dynamic Average Matching、DAM、ECAN、EtherGREEN、In-Circuit Serial Programming、ICSP、INICnet、Inter-Chip Connectivity、JitterBlocker、KleerNet、KleerNet 徽标、memBrain、Mindi、MiWi、MPASM、MPF、MPLAB Certified 徽标、MPLIB、MPLINK、MultiTRAK、NetDetach、Omniscient Code Generation、PICDEM、PICDEM.net、PICkit、PICtail、PowerSmart、PureSilicon、QMatrix、REAL ICE、Ripple Blocker、SAM-ICE、Serial Quad I/O、SMART-I.S.、SQI、SuperSwitcher、SuperSwitcher II、Total Endurance、TSHARC、USBCheck、VariSense、ViewSpan、WiperLock、Wireless DNA 和 ZENA 均为 Microchip Technology Incorporated 在美国和其他国家或地区的商标。

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

Adaptec 徽标、Frequency on Demand、Silicon Storage Technology 和 Symmcom 为 Microchip Technology Inc.在其他国家或地区的注册商标。

GestIC 是 Microchip Technology Inc.的子公司 Microchip Technology Germany II GmbH & Co. KG 在除美国外的国家或地区的注册商标。

在此提及的所有其他商标均为各持有公司所有。

ISBN： 978-1-5224-6111-1

质量管理体系

有关 Microchip 的质量管理体系的信息，请访问 www.microchip.com/quality。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. Chandler, AZ 85224-6199 电话：480-792-7200 传真：480-792-7277 技术支持： www.microchip.com/support 网址： www.microchip.com **亚特兰大** 德卢斯，佐治亚州 电话：678-957-9614 传真：678-957-1455 **奥斯汀，德克萨斯州** 电话：512-257-3370 **波士顿** 韦斯特伯鲁，马萨诸塞州 电话：774-760-0087 传真：774-760-0088 **芝加哥** 艾塔斯卡，伊利诺伊州 电话：630-285-0071 传真：630-285-0075 **达拉斯** 阿迪森，德克萨斯州 电话：972-818-7423 传真：972-818-2924 **底特律** 诺维，密歇根州 电话：248-848-4000 **休斯顿，德克萨斯州** 电话：281-894-5983 **印第安纳波利斯** 诺布尔斯维尔，印第安纳州 电话：317-773-8323 传真：317-773-5453 电话：317-536-2380 **洛杉矶** 米慎维荷，加利福尼亚州 电话：949-462-9523 传真：949-462-9608 电话：951-273-7800 **罗利，北卡罗来纳州** 电话：919-844-7510 **纽约，纽约州** 电话：631-435-6000 **圣何塞，加利福尼亚州** 电话：408-735-9110 电话：408-436-4270 **加拿大 - 多伦多** 电话：905-695-1980 传真：905-695-2078	**澳大利亚 - 悉尼** 电话：61-2-9868-6733 **中国 - 北京** 电话：86-10-8569-7000 **中国 - 成都** 电话：86-28-8665-5511 **中国 - 重庆** 电话：86-23-8980-9588 **中国 - 东莞** 电话：86-769-8702-9880 **中国 - 广州** 电话：86-20-8755-8029 **中国 - 杭州** 电话：86-571-8792-8115 **中国 - 香港特别行政区** 电话：852-2943-5100 **中国 - 南京** 电话：86-25-8473-2460 **中国 - 青岛** 电话：86-532-8502-7355 **中国 - 上海** 电话：86-21-3326-8000 **中国 - 沈阳** 电话：86-24-2334-2829 **中国 - 深圳** 电话：86-755-8864-2200 **中国 - 苏州** 电话：86-186-6233-1526 **中国 - 武汉** 电话：86-27-5980-5300 **中国 - 西安** 电话：86-29-8833-7252 **中国 - 厦门** 电话：86-592-2388138 **中国 - 珠海** 电话：86-756-3210040	**印度 - 班加罗尔** 电话：91-80-3090-4444 **印度 - 新德里** 电话：91-11-4160-8631 **印度 - 浦那** 电话：91-20-4121-0141 **日本 - 大阪** 电话：81-6-6152-7160 **日本 - 东京** 电话：81-3-6880- 3770 **韩国 - 大邱** 电话：82-53-744-4301 **韩国 - 首尔** 电话：82-2-554-7200 **马来西亚 - 吉隆坡** 电话：60-3-7651-7906 **马来西亚 - 槟榔屿** 电话：60-4-227-8870 **菲律宾 - 马尼拉** 电话：63-2-634-9065 **新加坡** 电话：65-6334-8870 **台湾地区 - 新竹** 电话：886-3-577-8366 **台湾地区 - 高雄** 电话：886-7-213-7830 **台湾地区 - 台北** 电话：886-2-2508-8600 **泰国 - 曼谷** 电话：66-2-694-1351 **越南 - 胡志明市** 电话：84-28-5448-2100	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393 **丹麦 - 哥本哈根** 电话：45-4485-5910 传真：45-4485-2829 **芬兰 - 埃斯波** 电话：358-9-4520-820 **法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79 **德国 - 加兴** 电话：49-8931-9700 **德国 - 哈恩** 电话：49-2129-3766400 **德国 - 海尔布隆** 电话：49-7131-72400 **德国 - 卡尔斯鲁厄** 电话：49-721-625370 **德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44 **德国 - 罗森海姆** 电话：49-8031-354-560 **以色列 - 若那那市** 电话：972-9-744-7705 **意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781 **意大利 - 帕多瓦** 电话：39-049-7625286 **荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340 **挪威 - 特隆赫姆** 电话：47-72884388 **波兰 - 华沙** 电话：48-22-3325737 **罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50 **西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91 **瑞典 - 哥德堡** 电话：46-31-704-60-40 **瑞典 - 斯德哥尔摩** 电话：46-8-5090-4654 **英国 - 沃金厄姆** 电话：44-118-921-5800 传真：44-118-921-5820