MPLAB® XC8 PIC®汇编器嵌入式工程师用户指南

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“DSXXXXXXXXN_CN”，其中“XXXXXXXX”为文档编号，“N”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®汇编器用户指南
本用户指南介绍了 MPLAB XC8 PIC 汇编器的用法和功能。

MPLAB® XC8 PIC®汇编器移植指南
本指南适用于拥有 MPASM™汇编器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 C Compiler Release Notes for PIC® MCU
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，请阅读相应工具的自述文件，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 5 月）

- 本文档的初始版本。

版本 B（2021 年 1 月）

- 复制了上一版本中的中断示例代码并进行了更新。
- 更改了器件并在中档器件示例中增加了配置位。
- 更正了存储器分区示例中使用的操作符。
- 增加了在 MPLAB X IDE 中进行编译的说明和项目属性设置的屏幕截图。
- 对代码和文本进行了少量更正与改进。

2. 简介

本指南介绍并说明了可使用适用于中档和 PIC18 器件系列的 MPLAB® XC8 PIC 汇编器（PIC 汇编器）编译的示例汇编程序。

本文档中介绍的示例汇集了各种编程概念、汇编器伪指令和操作符以及命令行选项，更多相关详细信息可参见《MPLAB® XC8 PIC®汇编器用户指南》。

如果要将程序从 Microchip MPASM™汇编器移植到 PIC 汇编器，结合使用本指南与《MPLAB® XC8 PIC®汇编器用户指南》会大有帮助。

3. PIC18 器件的基本示例

本章以如下完整汇编程序为例介绍 PIC 汇编器。该示例针对 PIC18F47K42 器件编写，但其大部分内容对于其他器件同样适用。该代码执行的任务很普通，即反复读取 PORTA 上的值并将读取到的最高值记录下来。后续各节将分别从各个方面来介绍该程序。

PIC18 基本示例

```
/* Find the highest PORTA value read, storing this into
   the object max */

PROCESSOR 18F47K42

#include <xc.inc>

CONFIG FEXTOSC = OFF        // External Oscillator Selection->Oscillator not enabled
CONFIG RSTOSC = HFINTOSC_1MHZ    // Reset Oscillator Selection->HFINTOSC, HFFRQ=4MHz, CDIV=4:1
CONFIG CLKOUTEN = OFF       // Clock out Enable bit->CLKOUT function is disabled
CONFIG PR1WAY = ON          // PRLOCKED One-Way Set Enable->PRLOCK cleared/set only once
CONFIG CSWEN = ON           // Clock Switch Enable bit->Writing to NOSC and NDIV is allowed
CONFIG FCMEN = ON           // Fail-Safe Clock Monitor Enable->Clock Monitor enabled

CONFIG MCLRE=EXTMCLR        // If LVP=0, MCLR pin is MCLR; If LVP=1, RE3 pin function is MCLR
CONFIG PWRTS=PWRT_OFF       // PWRT is disabled
CONFIG MVECEN=OFF           // Vector table isn't used to prioritize interrupts
CONFIG IVT1WAY=ON           // IVTLOCK bit can be cleared and set only once
CONFIG LPBOREN=OFF          // ULPBOR disabled
CONFIG BOREN=SBORDIS        // Brown-out Reset enabled, SBOREN bit is ignored
CONFIG BORV=VBOR_2P45       // Brown-out Reset Voltage (VBOR) set to 2.45V
CONFIG ZCD=OFF              // ZCD disabled, enable by setting the ZCDSEN bit of ZCDCON
CONFIG PPS1WAY=ON           // PPSLOCK cleared/set only once; PPS locked after clear/set cycle
CONFIG STVREN=ON            // Stack full/underflow will cause Reset
CONFIG DEBUG=OFF            // Background debugger disabled
CONFIG XINST=OFF            // Extended Instruction Set and Indexed Addressing Mode disabled

CONFIG WDTCPS=WDTCPS_31     // Divider ratio 1:65536; software control of WDTPS
CONFIG WDTE=OFF             // WDT Disabled; SWDTEN is ignored
CONFIG WDTCWS=WDTCWS_7      // window open 100%; software control; keyed access not required
CONFIG WDTCCS=SC            // Software Control

CONFIG BBSIZE=BBSIZE_512    // Boot Block size is 512 words
CONFIG BBEN=OFF             // Boot block disabled
CONFIG SAFEN=OFF            // SAF disabled
CONFIG WRTAPP=OFF           // Application Block not write protected
CONFIG WRTB=OFF             // Configuration registers (300000-30000Bh) not write-protected
CONFIG WRTC=OFF             // Boot Block (000000-0007FFh) not write-protected
CONFIG WRTD=OFF             // Data EEPROM not write-protected
CONFIG WRTSAF=OFF           // SAF not Write Protected
CONFIG LVP=ON               // Low voltage programming enabled, MCLR pin, MCLRE ignored

CONFIG CP=OFF               // PFM and Data EEPROM code protection disabled

GLOBAL max                  ;make this global so it is watchable when debugging

;objects in common (Access bank) memory
PSECT udata_acs
max:
    DS         1            ;reserve 1 byte for max
tmp:
    DS         1            ;1 byte for tmp

;this must be linked to the reset vector
PSECT resetVec,class=CODE,reloc=2
resetVec:
    goto       main

/* find the highest PORTA value read, storing this into
   the object max */
PSECT code
main:
    ;set up the oscillator
    movlw      0x62
```

```
    movlb      57                  ;(banking discussed in a later example)
    movwf      BANKMASK(OSCCON1),b
    movlw      2
    movwf      BANKMASK(OSCFRQ),b
    movlb      58
    clrf       BANKMASK(ANSELA),b  ;select digital input for port A

    clrf       max,c               ;starting point
loop:
    movff      PORTA,tmp           ;read the port
    movf       tmp,w,c             ;is this value larger than max?
    subwf      max,w,c
    bc         loop                ;no - read again
    movff      tmp,max             ;yes - record this new high value
    goto       loop                ;read again

    END        resetVec
```

3.1 注释

汇编源代码中可以使用多种样式的注释。

汇编器注释由分号;与其后面的注释文本组成，既可以单独成行，如下所示：

```
;objects in common (Access bank) memory
```

也可以放在指令或伪指令的后面，如下所示：

```
    movff    PORTA,tmp     ;read the port
```

如果对汇编源文件进行了预处理，则可在汇编源代码中使用 C 样式的注释。要对汇编源文件运行预处理器，应使用.S 扩展名（大写）为该文件命名，或在编译时使用-xassembler-with-cpp 选项。

单行 C 样式注释以双斜杠//开头，后跟注释文本，如下所示：

```
CONFIG PWRTS=PWRT_OFF    // PWRT is disabled
```

多行 C 样式注释以斜杠星号/*开头，以星号斜杠*/结尾。这是惟一一种允许注释文本跨越多行的注释样式，如下所示：

```
/* find the highest PORTA value read, storing this into
   the object max */
```

3.2 配置位

必须始终将器件的配置位设置为适当的值。如果配置位无效，程序可能会无法正确执行，甚至根本不会执行。用户应查阅器件数据手册，以确保了解每项配置设置的功能以及应使用的适当值。

可借助 MPLAB X IDE 来创建配置位设置所需的代码，但生成的代码必须都复制到项目的源文件中。

在本章的示例中，使用 PIC 汇编器的 CONFIG 伪指令指定了配置位，例如：

```
CONFIG WDTE=OFF    // WDT Disabled; SWDTEN is ignored
```

如示例代码中所示，通常会为这类伪指令提供设置-值对，将每个配置位设置为所需状态，但需要时也可以使用一条伪指令对整个配置字进行编程。此外，CONFIG 伪指令在相关的情况下也可用于设置器件的 IDLOC 位。

如果未使用 MPLAB X IDE，则可根据 HTML 指南确定与每个器件相关的配置位设置和值。对于 PIC18 器件，打开 pic18_chipinfo.html 指南；对于所有其他器件，打开 pic_chipinfo.html 指南。这些指南位于包含 PIC 汇编器的 MPLAB XC8 C 编译器发行版中的 docs 目录下。单击目标器件的链接会打开一个页面，其中显示适合所需伪指令的设置和值。请注意，这些 HTML 指南中演示#pragma config 伪指令的用例仅与 C 程序相关，但其中的设置-值对与 CONFIG 汇编器伪指令相关。

3.3 包含文件

与 C 源代码一样，汇编代码也可以包含其他文件的内容。

将文件包含到源代码中的最佳方法是使用#include 伪指令。由于该伪指令属于预处理器伪指令，因此必须对源文件运行预处理器才能执行该伪指令。为此，应使用（大写）.S 扩展名为源文件命名，或者使用-xassembler-with-cpp 选项。

预处理器将从汇编器发行版中的标准目录下搜索尖括号< >内指定的包含文件。 对于双引号" "内指定的文件名，先从当前工作目录中搜索，再从标准目录中搜索。此外，还可以使用-I 选项指定其他搜索路径。

如本章示例中的以下行所示：

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

汇编源文件通常应始终包含 PIC 汇编器提供的<xc.inc>包含文件。该文件允许您的源代码访问特殊功能寄存器以及器件的其他功能。请勿包含器件特定的包含文件，也不要自定义这些文件，因为这两类文件的内容在未来的汇编器版本中可能会发生变化。

3.4 常用伪指令

有几个汇编器伪指令通常在每个模块中都会使用。本章的示例代码中也使用了这些伪指令，本节将详细进行介绍。

Processor 伪指令

PROCESSOR 伪指令并不强制使用，但如果模块中的汇编源代码仅适用于一个器件，则应使用该伪指令。

-mcpu 选项始终指定要为哪个器件编译代码。如果使用了 PROCESSOR 伪指令，但该伪指令中指定的器件与选项中指定的器件不匹配，则会触发错误。以下代码行：

```
PROCESSOR 18F47K42
```

表示指定的器件为 PIC18F47K42。有了这行代码，便不能为任何其他器件编译代码。

如果要为多个器件编译代码，则应省略 PROCESSOR 伪指令。此外，也可以使用预处理器的条件包含功能和 PIC 汇编器预定义的预处理器宏，使代码段特定于目标器件。例如：

```
#ifdef _18F47K40
  movwf LATE
#endif
```

将仅针对 PIC18F47K40 器件汇编 movwf 指令。_18F47K40 预处理器宏是由汇编器根据-mcpu 选项选择的器件自动定义的。有关预定义宏的完整列表，请参见《MPLAB® XC8 PIC®汇编器用户指南》。

End 伪指令

END 伪指令并不强制使用，只是用于表示相应模块中的源代码结束。END 伪指令后面不能有其他源代码行，甚至连空白行也不能存在。

如果在程序中使用一条或多条 END 伪指令，则必须有且仅有一条伪指令指定程序的入口点标签，以防止生成汇编器警告。具体如示例程序的最后一行所示：

```
END resetVec
```

3.5 预定义 Psect

所有代码、数据对象或任何要链接到器件存储器的内容均必须置于 psect 中。

Psect（全称为 program sections，程序段）是一种对程序中彼此相关的部分进行分组保存的容器，这些相关部分的源代码在源文件中的位置可能并不相邻，甚至可能分布在多个模块中。Psect 是链接器定位到存储器中的最小实体。

要将代码或对象置于 psect 中，应在代码或对象的定义之前使用 PSECT 伪指令和要使用的 psect 的名称。第一次在模块中使用一个 psect 时，必须定义与该 psect 相关的 psect 标志。这类标志控制 psect 将位于哪个存储空间中，并进一步细化 psect 的链接方式。

为了方便移植和开发代码，PIC 汇编器预定义了几个带有相应标志的 psect，在将<xc.inc>文件包含到源代码中后即可使用。有关这些 psect 的完整列表，请参见《MPLAB® XC8 PIC®汇编器用户指南》。

在本章的示例中，使用预定义的 code psect 保存程序的大部分代码，具体在以下行中指定：

```
PSECT code
main:
    ;set up the oscillator
    movlw      0x62
    ...
```

这样便可将 main 标签及随后的说明置于 code psect 中。此外，还使用 udata_acs psect 将 max 和 tmp 标签以及与它们关联的保留存储器置于 PIC18 快速操作存储区中，如以下行所示：

```
PSECT udata_acs
max:
    DS         1           ;reserve 1 byte for max
tmp:
    DS         1           ;1 byte for tmp
```

配置数据也需位于 psect 内，以便其会出现在 HEX 文件中的适当地址处；但是，CONFIG 伪指令将自动确保其输出位于将链接到所需位置的 psect 中，因此不应在代码中的 CONFIG 伪指令之前使用 PSECT 指令。用户可以看到配置数据在映射文件中的链接位置，3.7 编译示例中将对此进行说明。

使用预定义 psect 的一个优点是它们已经与合适的链接器类相关联，因此在编译项目时将自动链接到适当的存储区，而无需指定任何链接器选项。

3.6 PIC18 器件的用户定义 psect

将代码或对象链接到特殊位置时，需要自定义惟一的 psect（而不是使用 PIC 汇编器的预定义 psect）。之后，可以将这些 psect 链接到所需的地址，以免影响其余代码或数据的存放。

在本章的示例中，程序的入口点由一小段代码组成，这段代码将在复位后立即执行。因此，必须在复位向量地址 0 处链接这段代码。

要将代码或对象置于 psect 中，应使用 PSECT 伪指令和要使用的 psect 的名称。第一次在模块中使用一个 psect 时，必须定义与该 psect 相关的 psect 标志。这类标志控制 psect 将位于哪个存储空间中，并进一步细化 psect 的链接方式。后续对同一 psect 使用 PSECT 伪指令时，可以省略这些标志。不同模块中的同名 psect 的标志必须一致。

在示例中，用于保存复位代码的 psect 定义如下，但请注意，该 psect 使用的标志仅与 PIC18 器件相关。

```
PSECT resetVec,class=CODE,reloc=2
resetVec:
    goto main
```

该伪指令定义并选择的 psect 名称为 resetVec。随后的标签和 goto 指令将属于该 psect 的一部分。请注意，psect 名称区分大小写。

psect 通常通过 class psect 标志与链接器类关联。在上面的示例中，resetVec psect 已与名为 CODE 的链接器类（由汇编器预定义）相关联。如果需要，一个类可定义一个或多个可链接 psect 的存储范围。由于本示例中的 resetVec psect 必须显式链接到绝对地址（复位向量），因此实际上并不需要类关联，但通常都会指定 psect 的类，哪怕只是为了更方便地在映射文件中找到 psect 列表。此外，仍可以使用链接器选项将与类关联的 psect 置于任意位置，这种情况下将忽略类的存储范围。有关预定义类的列表，请参见《MPLAB® XC8 PIC®汇编器用户指南》。

resetVec psect 中的 reloc 标志可确保 psect 的开头与等于标志值整数倍的地址对齐。对于所有保存可执行代码的 PIC18 psect，必须将该标志设置为 2，因为在 PIC18 器件上指令必须按字对齐。对于保存其他器件指令的 psect，必须将该标志设置为 1（这同时也是未指定 reloc 标志时的默认值）。设置该标志后，如果用于放置 psect 的链接器选项（如 3.7 编译示例所示）请求一个不兼容的地址，则链接器将报错。对于 psect 已分配存储空间（链接器类范围内的任何位置）的其他情况，链接器将选择一个与指定值对齐的地址。

任何用于保存指令的预定义 psect（例如 code psect）的预定义 psect 均为所使用的器件指定相关的 reloc 值。

3.7 编译示例

执行一次 pic-as 汇编器驱动程序即可编译一个或多个汇编源文件。

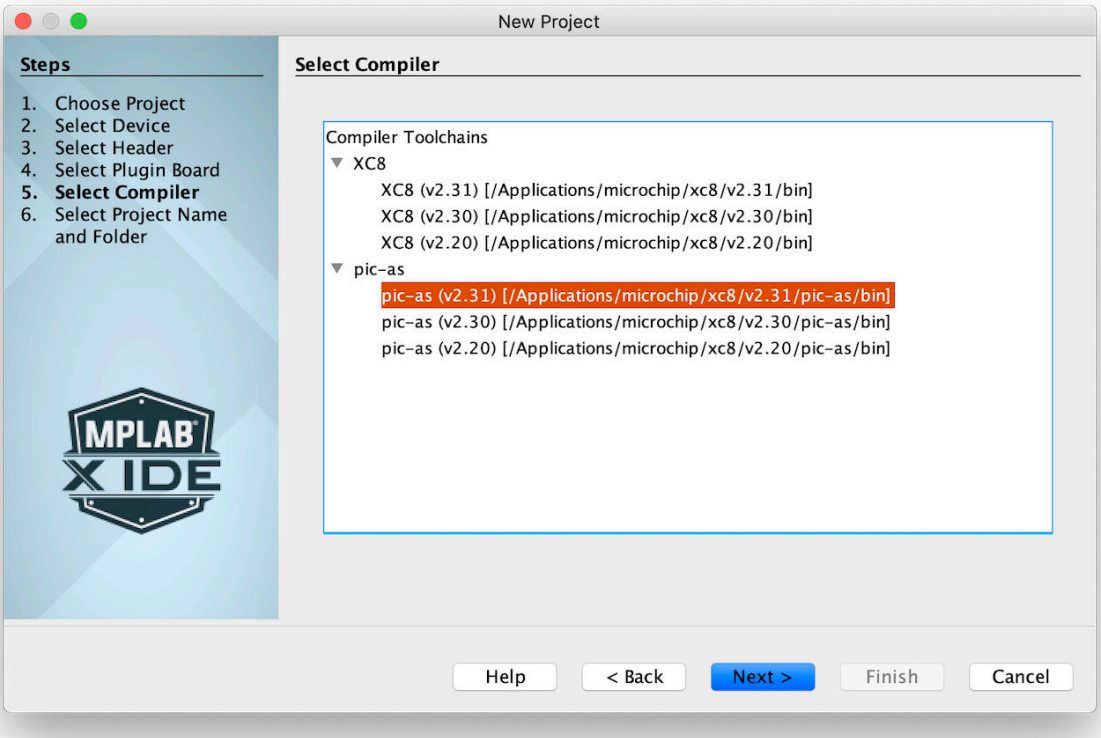
如果 3. PIC18 器件的基本示例所示的整个源代码已保存在纯文本文件（例如，名称为 readPort.S）中，则可以使用下面的命令对其进行编译。请注意，文件名使用大写扩展名 .S，以便先对文件进行预处理，再进行任何其他处理。

```
pic-as -mcpu=18f47k42 readPort.S
```

该命令将生成 readPort.hex 和 readPort.elf（也会生成其他文件），可供 IDE 和调试器工具用于执行和调试代码。此命令假定 pic-as 位于控制台的搜索路径中。如果不是这种情况，则应在运行应用程序时指定 pic-as 的完整路径。

如果希望在 MPLAB X IDE 中编译，则应以通常方式创建一个项目。当系统要求选择编译器时，选择所需版本的 pic-as 工具链。PIC 汇编器随 MPLAB XC8 C 编译器一起提供，但应确保选择汇编器工具，如下图所示。

图 3-1. 为 MPLAB X IDE 项目选择 PIC 汇编器



尽管上面的命令行（或默认 IDE 项目）编译后不会产生错误，但在代码运行之前，仍需要指定一些附加选项。要了解原因，使用-Wl 驱动程序选项再次编译代码以请求映射文件，如下所示。

```
pic-as -mcpu=18f47k42 -Wl,-Map=readPort.map readPort.S
```

打开 readPort.map 并查找代码中定义的特殊 resetVec psect。您将看到类似下面的内容：

```
TOTAL           Name                 Link     Load   Length     Space
        CLASS   CODE
                resetVec            1FFDE    1FFDE        4         0
                code                1FFE2    1FFE2       1E         0
```

resetVec psect 将出现在多个位置，但是可以在与之关联的 CODE 类下方轻松找到它。请注意，它链接到 1FFDE，而不是我们需要的地址 0，因为链接器没有收到显式的放置指令，所以只是将其链接到 CODE 类所定义的存储器中的空闲位置。但请注意，它确实遵循了 psect 的 reloc=2 标志并将其链接到等于 2 的整数倍的地址。

可以在映射文件的顶部看到 CODE（和其他）链接器类的定义（-A 链接器选项），这是链接器选项的一部分。对于该器件，CODE 类的定义如下：

```
-ACODE=00h-01FFFFh
```

这表明该类表示从地址 0 至 0x1FFFF 的一大块存储空间。

再次编译源文件，这次使用-Wl 驱动程序选项将-p 选项传递给链接器。在下面的命令行中，该选项将 resetVec psect 显式置于地址 0 处。

```
pic-as -mcpu=18f47k42 -Wl,-Map=readPort.map -Wl,-presetVec=0h readPort.S
```

现在，映射文件的内容将类似于下面，表明复位代码位于正确的位置。

```
TOTAL           Name                 Link     Load   Length     Space
        CLASS   CODE
                resetVec                0        0        4         0
                code                 1FFE2    1FFE2       1E         0
```

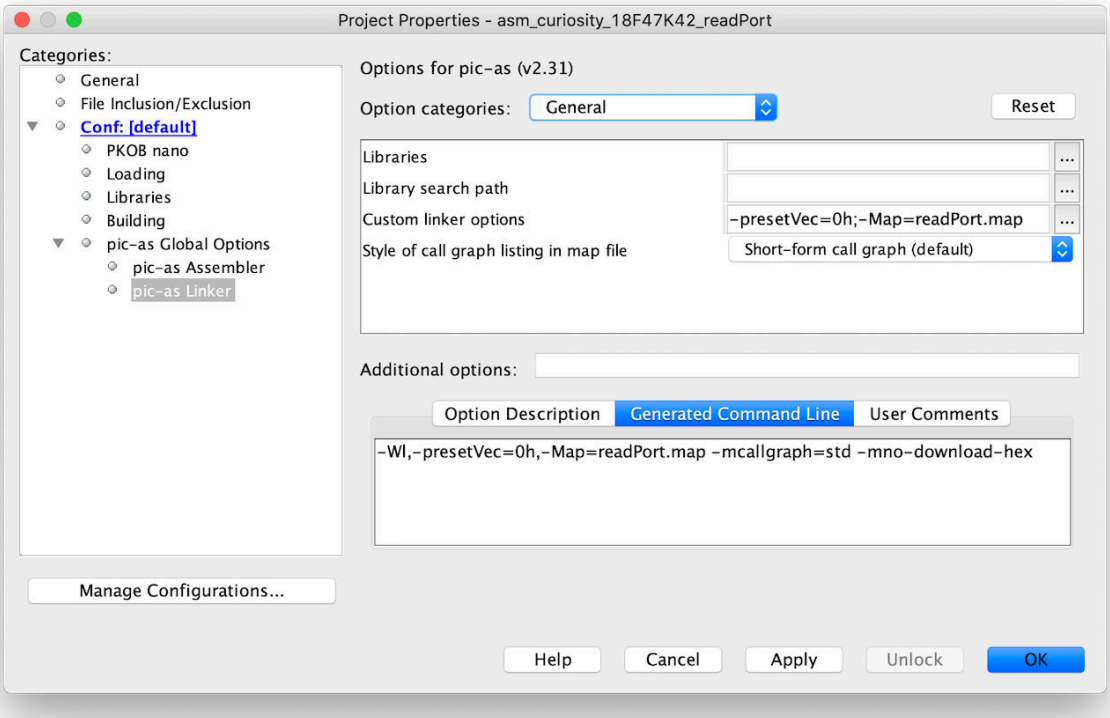
可以尝试使用上述选项将 resetVec psect“故意”链接到不等于 2 的整数倍的地址（例如地址 0x1），以确认链接器对 reloc psect 标志的处理。

另请注意，在映射文件中，CONFIG 伪指令指定的数据置于名为 config 的 psect 中，该 psect 链接到该器件的 hex 文件中的正确地址。

```
TOTAL           Name                 Link     Load   Length     Space
...
        CLASS   CONFIG
                config                       300000   300000         A         4
```

如果要在 IDE 中进行编译，应在 Project Properties（项目属性）对话框的 **pic-as Linker > general**（pic-as 链接器 > 常规）类别中的 **Customer linker options**（客户链接器选项）字段中添加其他链接器选项（上面使用驱动程序选项-Wl,*linkerOption* 指定的链接器选项），如下图所示。向该字段添加选项时，请勿包括开头的-Wl,，因为它是由 IDE 添加的。

图 3-2. 在 PIC 汇编器项目中指定将由链接器处理的其他选项



4. 中档器件的基本示例

以下完整汇编程序示例针对 PIC16F18446 器件编写，但其大部分内容对于其他器件同样适用。该程序执行的任务很普通，即反复读取 PORTC 上的值并将读取到的最高值记录下来。该示例是 3. PIC18 器件的基本示例中提供的信息的延续，后续各节将分别从各个方面来介绍该代码。

PIC16 基本示例

```
/*
 * Find the highest PORTC value read, storing this into the object max
 */

PROCESSOR 16F18446

#include <xc.inc>

CONFIG FEXTOSC = OFF        // External Oscillator mode selection bits->Oscillator not enabled
CONFIG RSTOSC = HFINT1      // Power-up default value for COSC bits->HFINTOSC (1MHz)
CONFIG CLKOUTEN = OFF       // Clock Out Enable->CLKOUT disabled; i/o or osc function on OSC2
CONFIG CSWEN = ON           // Clock Switch Enable bit->Writing to NOSC and NDIV is allowed
CONFIG FCMEN = ON           // Fail-Safe Clock Monitor Enable bit->FSCM timer enabled

CONFIG MCLRE = ON           // Master Clear Enable bit->MCLR pin is Master Clear function
CONFIG PWRTS = OFF          // Power-up Timer Enable bit->PWRT disabled
CONFIG LPBOREN = OFF        // Low-Power BOR enable bit->ULPBOR disabled
CONFIG BOREN = ON           // Brown-out reset enable->Brown-out Reset Enabled, SBOREN ignored
CONFIG BORV = LO            // Brown-out Reset Voltage Selection->VBOR set to 2.45V
CONFIG ZCDDIS = OFF         // Zero-cross detect disable->Zero-cross circuit disabled at POR
CONFIG PPS1WAY = ON         // Peripheral Pin Select control->PPSLOCK cleared/set only once
CONFIG STVREN = ON          // Stack Over/Underflow Reset Enable->Over/Underflow causes reset

CONFIG WDTCPS = WDTCPS_31   // WDT Period Select->Divider 1:65536; software control of WDTPS
CONFIG WDTE = OFF           // WDT operating mode->WDT Disabled, SWDTEN is ignored
CONFIG WDTCWS = WDTCWS_7    // WDT Window Select->window always open (100%); software control
CONFIG WDTCCS = SC          // WDT input clock selector->Software Control

CONFIG BBSIZE = BB512       // Boot Block Size Selection bits->512 words boot block size
CONFIG BBEN = OFF           // Boot Block Enable bit->Boot Block disabled
CONFIG SAFEN = OFF          // SAF Enable bit->SAF disabled
CONFIG WRTAPP = OFF         // Application Block Write Protection->Not write protected
CONFIG WRTB = OFF           // Boot Block Write Protection bit->Block not write protected
CONFIG WRTC = OFF           // Configuration Register Write Protection->Not write protected
CONFIG WRTD = OFF           // Data EEPROM write protection->Data EEPROM NOT write protected
CONFIG WRTSAF = OFF         // Storage Area Flash Write Protection->SAF not write protected
CONFIG LVP = ON             // Low Voltage Programming Enable->LV Programming enabled

CONFIG CP = OFF             // UserNVM Program memory code protection->protection disabled

skipnc  MACRO
    btfsc      CARRY
ENDM

;objects in bank 0 memory
PSECT udata_bank0
max:
    DS         1                ;reserve 1 byte for max
tmp:
    DS         1                ;reserve 1 byte for tmp

PSECT resetVec,class=CODE,delta=2
resetVec:
    PAGESEL    main             ;jump to the main routine
    goto       main

/* find the highest PORTA value read, storing this into
   the object max */
PSECT code
main:
    ;set up the oscillator
    movlw      0x62
    movlb      17
    movwf      OSCCON1
```

```
    movlw       2
    movwf       OSCFRQ
    PAGESEL     loop                ;ensure subsequent jumps are correct
    BANKSEL     max                 ;starting point
    clrf        BANKMASK(max)
    BANKSEL     ANSELC
    clrf        BANKMASK(ANSELC)    ;select digital input for port C
loop:
    BANKSEL     PORTC               ;read and store port value
    movf        BANKMASK(PORTC),w
    BANKSEL     tmp
    movwf       BANKMASK(tmp)
    subwf       max^(tmp&0ff80h),w  ;is this value larger than max?
    skipnc
    goto        loop                ;no - read again
    movf        BANKMASK(tmp),w     ;yes - record this new high value
    movwf       BANKMASK(max)
    goto        loop                ;read again

    END         resetVec
```

4.1 汇编器宏

本章的示例程序中定义并使用了汇编器宏。

汇编器宏会定义一个标识符来表示代码序列，因此执行的功能与预处理器的`#define` 伪指令类似。当要表示跨越多行的大量代码时，汇编器宏定义可能会比预处理器宏更易读。

在本章的示例中，宏定义为仅表示一条指令，如下所示。

```
skipnc  MACRO
    btfsc     CARRY
ENDM
```

上述代码创建的宏会根据 STATUS 寄存器中的进位位跳过以下指令。SFR 的位访问将在后面的 7.2 定义和使用位中介绍。

该宏在本示例中仅使用一次，具体如下所示：

```
    subwf       max^(tmp&0ff80h),w
    skipnc
    goto        loop
```

汇编器宏也可以有参数，并且有一些字符在宏定义中具有特殊含义。有关完整说明，请参见《MPLAB® XC8 PIC®汇编器用户指南》。

4.2 中档和低档器件的用户定义 Psect

本章的示例代码将与复位关联的入口点代码置于用户定义的 psect 中，具体操作与 3. PIC18 器件的基本示例所示的等效 PIC18 示例代码中的操作相同。但是，该 psect 的定义与 PIC18 代码使用的定义略有不同，具体如下所示：

```
PSECT resetVec,class=CODE,delta=2
```

请注意，对于保存中档或低档器件代码的 psect，并不需要像 PIC18 示例中那样使用 `reloc=2` 标志。由于未指定该标志，因此 `reloc` 值默认为 1，这表示该 psect 将不按字对齐。中档或低档器件上的指令可以位于任何地址，因此不需要按字对齐。

与 `resetVec` 配合使用的新 psect 标志是 `delta` 标志。当 `delta` 值为 2 时，表示该 psect 所在的存储空间中的每个地址上有 2 个字节。这与中档和低档器件上实现的程序存储器（宽度分别为 14 位或 12 位）一致，需要 2 个完整字节来进行编程。 另一方面，PIC18 器件在每个地址处定义 1 个字节的存储空间，因此，与保存这些器件代码的 psect 相关联的 `delta` 值应设置为 1（这同时也是未使用 `delta` 标志时的默认值）。

中档或低档器件上所有保存可执行代码的程序存储器 psect 均必须将 `delta` 标志设置为 2。目标为数据存储器的 psect 或与其他器件配合使用的 psect 应省略 `delta` 标志或将标志值显式设置为 1。

4.3 使用数据存储区

本章示例中的中档器件代码说明了如何处理数据存储器分区。请查阅具体器件的数据手册，以确保了解所选器件上的存储器分区处理方式。

等效的 PIC18 示例代码（如 3. PIC18 器件的基本示例所示）通过使用 movff 指令在很大程度上避免了分区数据存储器的复杂性，该指令可以访问整个数据存储器，而无需设置存储区选择寄存器。（movffl 指令还可以访问数据 RAM 容量更大的 PIC18 器件（例如，18F47K42）上的整个数据存储器。）此外，max 和 tmp 对象置于目标为快速操作存储区的 psect 中，该 psect 也可以使用文件寄存器指令进行访问，而无需使用存储区选择寄存器。

中档和低档 PIC 器件未实现 movff 指令，因此所有数据传送均通过分区指令来执行。同样，所有其他访问文件寄存器的指令（例如 clrf 和 addwf）也都是分区指令。此外，低档和中档器件的公共存储器可以独立于当前选择的存储区进行访问，但其空间非常有限，因此低档和中档器件上的大多数程序将需要考虑对象使用的存储区。

为了说明应如何使用分区指令，本中档示例中的代码通过将 tmp 和 max 对象置于汇编器定义的 psect udata_bank0 中，使这些对象链接到分区存储器而不是公共存储器中。这意味着，访问这些对象的代码必须处理存储区选择和地址掩码。

如果程序中的所有对象均位于同一存储区中，则可能只需要一个存储区选择序列。同样，对于存储区 0 中的对象，可以省略地址掩码。但是，最好始终使用地址掩码，并特别注意存储区选择，尤其是在修改对象放置时。

示例程序主要部分的开头如下。

```
    BANKSEL     max                 ;starting point
    clrf        BANKMASK(max)
    BANKSEL     ANSELC
    clrf        BANKMASK(ANSELC)    ;select digital input for port C
loop:
    BANKSEL     PORTC               ;read and store port value
    movf        BANKMASK(PORTC),w
    BANKSEL     tmp
    movwf       BANKMASK(tmp)
    subwf       max^(tmp&0ff80h),w  ;is this value larger than max?
  ...
```

BANKSEL max 伪指令用于选择对象 max 的存储区，以供后续指令之后对其进行访问。尽管可以手动编写为所用器件设置存储区选择位所需的一条或多条指令，但 BANKSEL 伪指令更易于移植和解析。在 PIC18 和增强型中档器件以外的器件上，该伪指令有时可以扩展为多条指令，因此永远不要紧接在 btfsc 等可能执行跳过的指令之后使用它。

clrf 指令用于清零对象 max。为此，BANKMASK() 预处理器宏与指令操作数配合使用，以移除包含在 max 的地址中的存储区位。该伪指令将地址与适当的掩码进行“逻辑与”运算。如果未移除该存储区信息，则链接器可能会发出修正溢出错误，表示链接器确定的操作数值对于指令操作码中的相关字段而言过大。（例如，中档器件文件寄存器指令的操作码中有一个 7 位宽的字段，用于指定在要访问的地址的当前所选存储区内的偏移。）

在上文所示的代码片段的最后几条指令中，编程人员假定 max 和 tmp 在同一 psect 中定义，因此二者将位于同一数据存储区中。该示例代码选择 tmp 的存储区，使用 movwf 指令将 W 寄存器内容写入该对象，然后通过 subwf 指令访问 max。基于编程人员的假设，用于在 subwf 指令之前选择 max 存储区的 BANKSEL 伪指令是多余的，因此已将其省略以减小程序的大小。

尽管上述假设有效并且代码将正确执行，但如果 max 和 tmp 的定义发生更改导致两个对象最终位于不同的存储区中，则代码执行可能失败。此类失败将很难跟踪。幸运的是，还有其他方法可以从地址中移除存储区信息，这十分有利。

上述示例中的最后一行代码如下：

```
subwf   max^(tmp&0ff80h),w
```

其中包含一项检查，用以确保 tmp 和 max 的存储区相同。该操作数表达式将 max 的完整地址与 tmp 的地址中包含的存储区位（通过使用 AND 运算符&仅掩码掉 tmp 的存储区偏移来获取）进行异或运算（^运算符），而不是通过使用 BANKMASK() 宏通过“逻辑与”运算从 max 中获取存储区信息。如果 max 地址中的存储区位与 tmp 的地址中的存储区位相同，则二者在该表达式中的异或结果为零。如果二者不相同，则将在地址的高位中产生非零值，并触发链接器发出修正溢出错误。这比运行时代码默默失败更可取。

上面的表达式通过检查确保两个对象位于同一存储区中，但这种检查也可用于确保某个对象位于特定存储区中。例如，如果代码要求 max 必须在存储区 2 中，则在中档器件上使用地址表达式 (max ^ 100h) 可在不满足上述要求时触发修正错误。此处，值 0x100 由表示存储区 2 的位序列组成，存储区偏移的所有七位（最低有效位）清零。

如果决定不使用 BANKMASK() 宏，请参见《MPASM™汇编器至 MPLAB® XC8 PIC®汇编器移植指南》中有关文件寄存器地址掩码的部分，以了解地址格式和要使用的适当掩码的更多相关信息。

4.4 编译示例

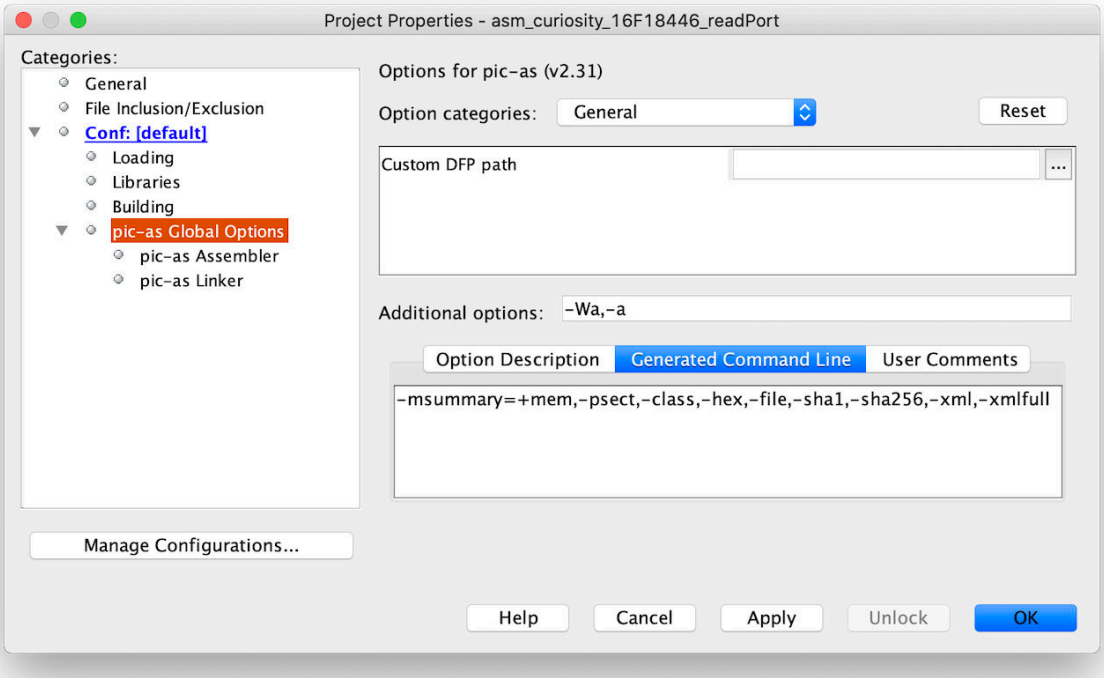
如果本章的整个示例源代码已保存在纯文本文件（例如，名称为 readPort.S）中，则可以使用下面的命令对其进行编译。请注意，文件名使用大写扩展名 .S，以便先对文件进行预处理，再进行任何其他处理。

```
pic-as -mcpu=16f18446 -Wl,-presetVec=0h -Wa,-a -Wl,-Map=readPort.map readPort.S
```

包含复位代码的 psect 已链接到地址 0。该命令还包括用于生成映射（readPort.map）的选项。选项 -Wa,-a 请求生成汇编列表文件，从而可以浏览生成的代码。该选项创建的列表文件的基本名称与命令行中的第一个源文件相同，而扩展名为 .lst。上面的命令将生成一个名为 readPort.lst 的列表文件。

如果要在 MPLAB X IDE 中编译，应在 **pic-as Global Options**（pic-as 全局选项）类别的 **Additional options**（其他选项）字段中为汇编器指定其他选项（例如 -Wa,-a），如下图所示。

图 4-1. 在 PIC 汇编器项目中指定将由汇编器处理的其他选项



5. 多个源文件、分页和线性存储器示例

在本 PIC16F18446 示例中，代码从 PORTC 读取多个值，并将这些值存储到使用线性存储器寻址访问的数组元素中。尽管只有增强型中档器件实现了线性存储器访问，但是该代码的其他方面与所有器件都相关。示例源代码已被特意拆分为两个文件，用以分别说明代码如何使用其他模块中定义的程序和对象，以及这对程序存储器页处理方式的影响。

PIC16 示例——file_1.S

```
/*
 * Take NUM_TO_READ samples of PORTC, storing this into an array accessed
 * using linear memory. NUM_TO_READ must be defined as a macro on the command-line.
 */

PROCESSOR 16F18446

#include <xc.inc>

CONFIG FEXTOSC = OFF        // External Oscillator mode selection bits->Oscillator not enabled
CONFIG RSTOSC = HFINT1      // Power-up default value for COSC bits->HFINTOSC (1MHz)
CONFIG CLKOUTEN = OFF       // Clock Out Enable->CLKOUT disabled; i/o or osc function on OSC2
CONFIG CSWEN = ON           // Clock Switch Enable bit->Writing to NOSC and NDIV is allowed
CONFIG FCMEN = ON           // Fail-Safe Clock Monitor Enable bit->FSCM timer enabled

CONFIG MCLRE = ON           // Master Clear Enable bit->MCLR pin is Master Clear function
CONFIG PWRTS = OFF          // Power-up Timer Enable bit->PWRT disabled
CONFIG LPBOREN = OFF        // Low-Power BOR enable bit->ULPBOR disabled
CONFIG BOREN = ON           // Brown-out reset enable->Brown-out Reset Enabled, SBOREN ignored
CONFIG BORV = LO            // Brown-out Reset Voltage Selection->VBOR set to 2.45V
CONFIG ZCDDIS = OFF         // Zero-cross detect disable->Zero-cross circuit disabled at POR
CONFIG PPS1WAY = ON         // Peripheral Pin Select control->PPSLOCK cleared/set only once
CONFIG STVREN = ON          // Stack Over/Underflow Reset Enable->Over/Underflow causes reset

CONFIG WDTCPS = WDTCPS_31   // WDT Period Select->Divider 1:65536; software control of WDTPS
CONFIG WDTE = OFF           // WDT operating mode->WDT Disabled, SWDTEN is ignored
CONFIG WDTCWS = WDTCWS_7    // WDT Window Select->window always open (100%); software control
CONFIG WDTCCS = SC          // WDT input clock selector->Software Control

CONFIG BBSIZE = BB512       // Boot Block Size Selection bits->512 words boot block size
CONFIG BBEN = OFF           // Boot Block Enable bit->Boot Block disabled
CONFIG SAFEN = OFF          // SAF Enable bit->SAF disabled
CONFIG WRTAPP = OFF         // Application Block Write Protection->Not write protected
CONFIG WRTB = OFF           // Boot Block Write Protection bit->Block not write protected
CONFIG WRTC = OFF           // Configuration Register Write Protection->Not write protected
CONFIG WRTD = OFF           // Data EEPROM write protection->Data EEPROM NOT write protected
CONFIG WRTSAF = OFF         // Storage Area Flash Write Protection->SAF not write protected
CONFIG LVP = ON             // Low Voltage Programming Enable->LV Programming enabled

CONFIG CP = OFF             // UserNVM Program memory code protection->protection disabled

PSECT code
;read PORTC, storing the result into WREG
readPort:
    BANKSEL   PORTC
    movf      BANKMASK(PORTC),w
    return

GLOBAL count                        ;make this globally accessible

PSECT udata_shr
count:
    DS        1                     ;1 byte in common memory

PSECT resetVec,class=CODE,delta=2
resetVec:
    PAGESEL   main
    goto      main

GLOBAL storeLevel                   ;link in with global symbol defined elsewhere

PSECT code
main:
    BANKSEL   ANSELC
```

```
    clrf      BANKMASK(ANSELC)
    clrf      count
loop:
    ;a call to a routine in the same psect
    call      readPort            ;value returned in WREG
    ;a call to a routine in a different module
    PAGESEL   storeLevel
    call      storeLevel          ;expects argument in WREG
    PAGESEL   $
    ;wait for a few cycles
    movlw     0xFF
delay:
    decfsz    WREG,f
    goto      delay
    ;increment the array index, count, and stop iterating
    ;when the final element is reached
    movlw     NUM_TO_READ
    incf      count,f
    xorwf     count,w
    btfss     ZERO
    goto      loop

    goto      $                   ;loop forever

    END       resetVec
```

PIC16 示例——file_2.S

```
PROCESSOR 16F18446

#include <xc.inc>

GLOBAL storeLevel                 ;make this globally accessible
GLOBAL count                      ;link in with global symbol defined elsewhere

PSECT udata_shr
tmp:
    DS     1

;define NUM_TO_READ bytes of linear memory, at banked address 0x120
DLABS  1,0x120,NUM_TO_READ,levels

PSECT code
;store byte passed via WREG into the count-th element of the
;linear memory array, levels
storeLevel:
    movwf     tmp                 ;store the parameter
    movf      count,w             ;add the count index to...
    addlw     low(levels)         ;the base address of the aray...
    movwf     FSR1L               ;storing the result in FSR1
    movlw     high(levels)
    clrf      FSR1H
    addwfc    FSR1H
    movf      tmp,w               ;retrieve the parameter
    movwf     INDF1               ;access levels in linear memory
    return

    END
```

5.1 多个源文件和共享访问

汇编程序可以根据需要拆分为多个源文件。这样会使源代码更易于管理和浏览，但前提是必须执行一系列步骤在模块之间共享对象和程序，并且需要在代码中考虑其他模块中的程序最终可能处于什么位置。

若要访问另一个源文件中定义的对象或程序，需要遵循两个步骤。首先，定义对象或程序时，必须告知链接器允许全局访问符号（标签），具体使用 GLOBAL 伪指令来完成。接下来，必须在每个需要访问该符号的其他文件中声明相同的符号，以通知链接器该符号已链接到另一个模块中的定义，具体可以使用 GLOBAL 或 EXTRN 伪指令来完成。

在本章所示的完整示例程序中，file_1.S 包含对象 count（本质上是与保留的存储空间关联的标签）的定义，如下所示。

```
GLOBAL count

PSECT  udata_shr
count:
    DS    1                    ;1 byte in common memory
```

由于必须可从其他模块访问 count，因此除了该符号的定义外，还使用了 GLOBAL count 伪指令（如上所示）以告知链接器其他模块中相同符号的声明可以链接到该模块中的定义。

在 file_2.S 中，则使用了另一个 GLOBAL count 伪指令来声明符号，并将其与链接器最终将在 file_1.o 模块中找到的 count 的定义相链接。此外，也可以使用 EXTRN count 伪指令来执行该功能。该伪指令执行的任务与 GLOBAL 类似，但如果该伪指令出现在包含该符号定义的同一模块中，则将触发错误。

该机制同样适用于需要从多个模块访问的程序所使用的符号。在本示例中，file_2.S 中的代码包含一条 GLOBAL storeLevel 伪指令，允许从该模块外部引用该符号；而 file_1.S 包含相同的伪指令，可将该模块中的 storelevel 符号与其在另一个模块中的定义相链接。同样，file_1.S 中可能已使用 EXTRN storeLevel 伪指令，以确保在所需的模块中定义该符号。

5.2 Psect 连接和分页

必须将所有可执行代码置于 psect 中。psect 基于其定义的方式和位置进行分组，这可能会影响用于跳转到标签或调用其他程序的代码的编写方式。

请注意，在 file_1.S 中，有两个单独的汇编代码块位于 code psect 中（一个定义 readPort，另一个包含 main）。同名 psect 的内容在存储器分配之前由链接器进行连接（除非它们使用 ovrld 标志），因此与标签 main: 关联的指令将直接出现在 readPort 程序中的最后一条指令之后，即使源文件中的这些块之间定义了其他代码和对象也是如此。

但请注意，file_2.S 中 code psect 的内容不会与 file_1.S 中 code psect 已合并的内容连接。psect 连接仅适用于在同一模块中定义的同名 psect。 在这种情况下，file_2.S 中的 code psect 将独立链接到 file_1.S 中的 code psect。

编译示例程序后，映射文件将显示链接到不同地址的两个 code psect。code psect 在 file_1.o（由 file_1.S 生成的目标文件）下的逐模块列表中列出一次，然后在下面的映射文件摘录部分的 file_2.o 下再次列出。（这些 psect 同样按类列出，但此处无法查看定义它们的源文件。）请注意，根据指示 file_1.o 只有一个 code psect，因为该 psect 中的两个代码段已连接。

```
                Name                    Link     Load   Length Selector   Space Scale
file_1.o        resetVec                   0        0        2        0       0
                code                    3E7      3E7       19      7CE       0
                udata_shr                71       71        1       70       1
file_2.o        code                    3DD      3DD        A      7BA       0
                udata_shr                70       70        1       70       1

TOTAL           Name                    Link     Load   Length     Space
        CLASS   CODE
                resetVec                   0        0        2         0
                code                    3E7      3E7       19         0
                code                    3DD      3DD        A         0
```

低档和中档器件上的程序存储器进行了分页，器件数据手册将指示器件的页排列。psect 的连接方式会影响这些器件进行调用或跳转到标签时所需代码的编写方式。PIC18 器件上的程序存储器未进行分页。程序存储器中的所有地址均可通过调用和跳转指令访问，因此以下讨论不适用于为 PIC18 器件编写的程序。

在低档和中档器件上进行调用或跳转之前，PCLATH 寄存器必须包含用于选择目标页的值（地址的高位）。PAGESEL 伪指令可用于初始化 PCLATH 寄存器。

示例代码在 call storeLevel 指令之前使用了 PAGESEL 伪指令，如下所示：

```
loop:
    ;a call to a routine in the same psect
```

```
    call      readPort             ;value returned in WREG
    ;a call to a routine in a different module
    PAGESEL   storeLevel
    call      storeLevel           ;expects argument in WREG
    PAGESEL   $
    ;wait for a few cycles
    movlw     0xFF
delay:
    decfsz    WREG,f
    goto      delay
```

请注意，在调用之后，将使用当前位置计数器$作为参数再次使用该伪指令，以确保 PCLATH 再次指向保存当前所执行代码的页。这样可以在调用后让 goto delay 指令按预期工作。

如上所示，对 readPort 的调用未使用 PAGESEL 伪指令。在这种情况下，PCLATH 寄存器因两种情况而无需更新。首先，如上所述，包含 readPort 程序的 code psect 将与保存 main 程序的 code psect 连接，因此这两个程序（调用方和被调用方）将位于同一连接 psect 中；其次，定义与 code psect 关联的 CODE 链接器类时已确保在其存储器范围内的 psect 永远不会跨越页边界。

编译时使用的链接器选项显示在映射文件中。对于本示例中使用的 16F18446 器件，映射文件的顶部如下：

```
Linker command line:

-W-3 --edf=/Applications/microchip/xc8/v2.31/pic/dat/en_msgs.txt -cs \
  -h+dist/default/debug/asm_curiosity_16F18446_linearMemory.X.debug.sym \
  --cmf=dist/default/debug/asm_curiosity_16F18446_linearMemory.X.debug.cmf \
  -z -Q16F18446 -o/tmp/xckFLd4UW -presetVec=0h -ver=XC8 PIC(R) Assembler \
  -Mfile.map -E1 --acfsm=1493 -ACODE=00h-07FFhx8 -ASTRCODE=00h-03FFFh \  ...
```

CODE 类由以下链接器选项定义： -ACODE=00h-07FFhx8。该选项表明与 CODE 类关联的存储器由 8 个连续的页组成，第一页从地址 0 开始，每页的长度为 0x800 个字。这些范围与 16F18446 器件上的页地址相对应。

链接器绝不允许置于类关联存储器中的 psect 跨越该类的存储器范围边界，这意味着在本示例中，位于 CODE 类中的 psect 将完全包含在器件页中。如果链接到该类的 psect 超出页大小，则将收到“无法找到空间”错误。如果改用-ACODE=0-01FFFh 定义了该类，则该类将覆盖完全相同的存储器，但是该存储器中的边界将不存在。置于这样的类中的 psect 可以链接到任何位置，因此可能跨越器件页边界。

通常情况下会限制链接器放置 psect 的位置，以便之后可以在源代码中做出提高效率的假设。在这种情况下，CODE 类中的页边界意味着，在调用或跳转到在同一 psect 和同一模块中定义的标签时，无需先选择目标页（前提是 PCLATH 已指向该页）。

对于任何低档和中档非相对控制指令（即 goto、call 和 callw 指令），必须考虑页选择。在相对转移指令之前不需要页选择；但当这些指令在转移后修改 PC 时，将需要评估在转移之后进行的调用和跳转是否需要页选择。如果使用任何将 PCL 寄存器指定为目标的指令，也可能需要页选择，因为这些指令也使用 PCLATH 寄存器形成目标地址。

可以使用以下两条伪指令来确保始终进行页选择：ljmp 和 fcall。 这两条伪指令分别通过 goto 或 call 指令之前的必要页选择扩展到 goto 和 call，然后在指令之后选择当前页。由于 ljmp 和 fcall 助记符可以扩展为多条 PIC 指令，因此绝不能紧接在任何执行跳过的指令（例如 btfsc 指令）之后使用它们。可以在汇编列表文件中查看为 ljmp 或 fcall 伪指令生成的操作码。

上面的代码片段可以重新编写为如下形式：

```
loop:
    ;a call to a routine in the same psect
    call      readPort             ;value returned in WREG
    ;a call to a routine in a different module
    fcall      storeLevel          ;expects argument in WREG
    ;wait for a few cycles
    movlw     0xFF
delay:
    decfsz    WREG,f
    goto      delay
```

如果不确定在调用或跳转之前是否需要页选择，最安全的方法是使用 PAGESEL 伪指令或使用 fcall 和 ljmp 指令，但请记住，这样做可能会不必要地增加代码的长度并减慢代码的执行速度。

上述注意事项可能会使您倾向于将大多数代码置于一个模块中的同一 psect 中，以避免使用页选择指令，但请记住，一旦 psect 超出页大小，便不再适合 CODE 类，随后链接器将发出“无法找到空间”错误消息。应考虑将频繁调用的程序以及调用这些程序的程序置于同一模块中的同名 psect 中。将其他程序移至其他模块，或将其置于名称不同的 psect 中，以便单独链接它们并且可以填充其他器件页。

如果决定创建自己的 psect 和链接器类来定位代码，请记住，链接器类的定义方式可能会影响代码的编写方式。 如果程序会跨越存储区边界，则它们很可能可以装入程序存储器，但您的程序将需要更多的页选择指令。如果手动定位 psect 来控制需要页选择指令的位置（而不是将它们链接在链接器类中的任何位置），则在调试和开发代码时将需要大量维护。

5.3 线性存储器

线性寻址模式是一种将增强型中档器件上的分区数据存储器作为一个连续线性块进行访问的方法。器件数据手册中会指出器件上是否已实现该存储器，并提供有关操作的更多详细信息。

线性存储器通常用于因过大而无法装入单个存储区的对象，但务必要记住，器件上只有常规的分区存储器，并不存在线性存储器；线性存储器只是一种可以访问该分区存储器的备选方法。对于超出存储区大小的对象，定义方式必须与置于分区存储器中的普通对象有所不同。

本章的示例代码使用 DLABS 伪指令定义了一个超出存储区大小的对象，如下所示。

```
;define NUM_TO_READ bytes of linear memory, at banked address 0x120
DLABS  1,0x120,NUM_TO_READ,levels
```

该伪指令有多个参数。第一个参数是用于定义存储器的地址空间。对于要置于数据存储器中的对象，该值应为 1。第二个参数是用于存储的起始地址。可将其指定为线性地址或分区地址。例如，上述示例中使用的分区地址 0x120 的等效线性地址为 0x20A0。下一个参数是要保留的字节数。在本示例中，使用预处理器宏表示数组大小。该宏在编译选项中定义，如 5.4 编译示例所示。 最后一个参数是符号。该参数是可选的，用于为位于指定地址的对象创建标签。

与大多数伪指令不同的是，DLABS 伪指令不会产生属于当前 psect 的输出，因此它可以出现在源文件中的任何位置。这也意味着不应紧接在该伪指令之前放置标签。如果要定义的对象需要标签，则将标签指定为该伪指令的最后一个参数，如上所示。

在本示例中，编译器将在存储区 2 和存储区 3 中为 levels 对象保留由伪指令使用的地址所指定的存储空间。尽管可以使用普通文件寄存器指令访问该存储器，但如果在数组中的偏移量未知，则此类访问可能会出现问题，因为尚不清楚需要预先选择哪个存储区。但是，增强型中档器件允许使用 FSR 寄存器通过线性地址（无需进行分区）间接读取和写入数据存储器。下面给出了示例中用于间接访问 levels 的部分。

```
storeLevel:
    movwf     tmp                 ;store the parameter
    movf      count,w             ;add the count index to...
    addlw     low(levels)         ;the linear base address of the array...
    movwf     FSR1L               ;storing the result in FSR1
    movlw     high(levels)
    clrf      FSR1H
    addwfc    FSR1H
    movf      tmp,w               ;retrieve the parameter
    movwf     INDF1               ;access levels using linear memory
```

由于符号 levels 是使用 DLABS 伪指令定义的，因此它将始终表示线性地址，即使将分区地址用作该伪指令的第二个参数也依然如此。当需要线性访问时，可以将此类地址直接装入 FSR 寄存器中。如果符号是使用 GLOBAL 伪指令全局访问的，则可以在汇编列表文件以及映射文件中找到符号的线性地址值。

5.4 编译示例

如果本章所示的两个源代码示例均已保存在名为 file_1.S 和 file_2.S 的纯文本文件中，则可以使用下面的单条命令对其进行编译。

```
pic-as -mcpu=16f18446 -Wl,-presetVec=0h -DNUM_TO_READ=100 -Wa,-a -Wl,-Map=linearMemory.map
file_1.S file_2.S
```

在该命令中，包含复位代码的 psect 已链接到地址 0。该命令还包括用于生成映射文件（名为 linearMemory.map）的选项。-Wa,-a 选项用于请求汇编列表文件。对于每个源模块，都将生成一个列表文件，并且上述命令将生成名为 file_1.lst 和 file_2.lst 的文件。

已使用-D 选项将预处理器宏 NUM_TO_READ（用于确定线性数组大小和存储值的数量）定义为 100。在命令行中（而不是使用#define 伪指令）进行此定义意味着可以在每个源文件中更轻松地使用该宏。它还允许更改与宏关联的值，而无需修改源代码。

如果希望单独编译每个文件（例如，通过 make 文件），则还可以使用以下命令：

```
pic-as -mcpu=16f18446 -Wa,-a -c -DNUM_TO_READ=100 file_1.S
pic-as -mcpu=16f18446 -Wa,-a -c -DNUM_TO_READ=100 file_2.S
pic-as -mcpu=16f18446 -Wl,-presetVec=0h -Wl,-Map=linearMemory.map file_1.o file_2.o
```

此处，已使用-c 选项为每个源文件生成一个中间文件。中间（目标）文件的扩展名将为.o。最终命令将链接目标文件并生成最终输出。请注意，前两条汇编命令需要与创建汇编列表文件的选项以及定义预处理器宏的-D 选项配合使用。只有最终编译命令才需要与创建映射文件的选项配合使用。每条命令都必须与指示目标器件的-mcpu 选项一起使用。

如果要在 MPLAB X IDE 中编译，则将以类似于上文中多步命令行示例的方式对每个源文件执行增量编译。要指定-D 选项，应在 **PIC-AS assembler > Preprocessing and Messages**（PIC-AS 汇编器 > 预处理和消息）类别中的 **Define preprocessor symbol**（定义预处理器符号）字段中输入 NUM_TO_READ=100。

6. 编译堆栈示例

本章中的示例代码说明了如何使用链接器将对象置于编译堆栈中。

请不要将编译堆栈与软件堆栈相混淆，软件堆栈是一种通过堆栈指针访问的动态堆栈分配，而编译堆栈则是为本地对象静态分配的存储区域，并且这些本地对象仅在程序执行期间占用存储空间，行为类似于 C 程序中函数的自动和参数对象。

将对象置于编译堆栈中的主要优势在于，每个程序为其本地对象占用的存储空间可供其他程序定义的本地对象重复使用。这极大地减少了程序占用的数据存储空间。

编译堆栈的缺点是编程人员必须始终使用伪指令来指示每个程序的存储器需求以及这些程序的交互方式，否则可能会导致代码失败。

编译堆栈不使用堆栈指针。将对象置于此类堆栈中会为其分配一个静态地址（可通过符号进行访问），因此对普通对象使用编译堆栈不会对代码长度或执行速度造成负面影响。堆栈中的存储空间分配由链接器执行，因此可以确定堆栈的总大小，并且当无法为所请求的堆栈对象找到存储空间时，链接器会发出存储器错误。

由于分配给编译堆栈对象的存储空间是静态分配的，使用这些对象的程序不可重入，因此这些程序既不能递归调用，也不能从主干代码和中断程序（或由中断程序调用的任何内容）进行调用。

如下所示的汇编代码说明了在 PIC18F47K42 器件上使用编译堆栈的基本原理。

编译堆栈示例

```
#include <xc.inc>

CONFIG FEXTOSC = OFF        // External Oscillator Selection->Oscillator not enabled
CONFIG RSTOSC = HFINTOSC_1MHZ    // Reset Oscillator Selection->HFINTOSC, HFFRQ=4MHz, CDIV=4:1
CONFIG CLKOUTEN = OFF       // Clock out Enable bit->CLKOUT function is disabled
CONFIG PR1WAY = ON          // PRLOCKED One-Way Set Enable->PRLOCK cleared/set only once
CONFIG CSWEN = ON           // Clock Switch Enable bit->Writing to NOSC and NDIV is allowed
CONFIG FCMEN = ON           // Fail-Safe Clock Monitor Enable->Clock Monitor enabled

CONFIG MCLRE=EXTMCLR        // If LVP=0, MCLR pin is MCLR; If LVP=1, RE3 pin function is MCLR
CONFIG PWRTS=PWRT_OFF       // PWRT is disabled
CONFIG MVECEN=OFF           // Vector table isn't used to prioritize interrupts
CONFIG IVT1WAY=ON           // IVTLOCK bit can be cleared and set only once
CONFIG LPBOREN=OFF          // ULPBOR disabled
CONFIG BOREN=SBORDIS        // Brown-out Reset enabled, SBOREN bit is ignored
CONFIG BORV=VBOR_2P45       // Brown-out Reset Voltage (VBOR) set to 2.45V
CONFIG ZCD=OFF              // ZCD disabled, enable by setting the ZCDSEN bit of ZCDCON
CONFIG PPS1WAY=ON           // PPSLOCK cleared/set only once; PPS locked after clear/set cycle
CONFIG STVREN=ON            // Stack full/underflow will cause Reset
CONFIG DEBUG=OFF            // Background debugger disabled
CONFIG XINST=OFF            // Extended Instruction Set and Indexed Addressing Mode disabled

CONFIG WDTCPS=WDTCPS_31     // Divider ratio 1:65536; software control of WDTPS
CONFIG WDTE=OFF             // WDT Disabled; SWDTEN is ignored
CONFIG WDTCWS=WDTCWS_7      // window open 100%; software control; keyed access not required
CONFIG WDTCCS=SC            // Software Control

CONFIG BBSIZE=BBSIZE_512    // Boot Block size is 512 words
CONFIG BBEN=OFF             // Boot block disabled
CONFIG SAFEN=OFF            // SAF disabled
CONFIG WRTAPP=OFF           // Application Block not incr protected
CONFIG WRTB=OFF             // Configuration registers (300000-30000Bh) not incr-protected
CONFIG WRTC=OFF             // Boot Block (000000-0007FFh) not incr-protected
CONFIG WRTD=OFF             // Data EEPROM not incr-protected
CONFIG WRTSAF=OFF           // SAF not Write Protected
CONFIG LVP=ON               // Low voltage programming enabled, MCLR pin, MCLRE ignored

CONFIG CP=OFF               // PFM and Data EEPROM code protection disabled


;place the compiled stack in Access bank memory (udata_acs psect)
;use the ?au_  prefix for autos, the ?pa_  prefix for parameters
FNCONF udata_acs,?au_,?pa_

PSECT resetVec,class=CODE,reloc=2
resetVec:
```

```
    goto       main

PSECT code
;add needs 4 bytes of parameters, but no autos
FNSIZE add,0,4       ;two 2-byte parameters
GLOBAL ?pa_add
;add the two 'int' parameters, returning the result in the first parameter location
add:
    movf       ?pa_add+2,w,c
    addwf      ?pa_add+0,f,c
    movf       ?pa_add+3,w,c
    addwfc     ?pa_add+1,f,c
    return

;incr needs one 2-byte parameter
FNSIZE incr,0,2
GLOBAL ?pa_incr
;return the additional of the 2-byte parameter with the value in the W register
incr:
    addwf      ?pa_incr+0,c
    movlw      0h
    addwfc     ?pa_incr+1,c
    return

GLOBAL ?au_main
GLOBAL result
result  EQU    ?au_main+0          ;create an alias for this auto location

GLOBAL incval
incval EQU     ?au_main+2          ;create an alias for this auto location

FNROOT main                        ;this is the root of a call graph
FNSIZE main,4,0                    ;main needs two 2-byte 'autos' (for result and incval)
FNCALL main,add                    ;main calls add
FNCALL main,incr                   ;main calls incr

PSECT code
main:
    clrf       result+0,c
    clrf       result+1,c
    movlw      2                       ;increment amount
    movwf      incval+0,c
    clrf       incval+1,c
loop:
    movff      result+0,?pa_add+0      ;load 1st parameter for add routine
    movff      result+1,?pa_add+1
    movff      incval+0,?pa_add+2      ;load 2nd parameter for add routine
    movff      incval+1,?pa_add+3
    call       add                     ;add result and incval
    movff      ?pa_add+0,result+0      ;store add's return value back to result
    movff      ?pa_add+1,result+1

    movff      incval+0,?pa_incr+0     ;load the parameter for incr routine
    movff      incval+1,?pa_incr+1
    movlw      2
    call       incr                    ;add 2 to incval
    movff      ?pa_incr+0,incval+0     ;store the result of incr back to incval
    movff      ?pa_incr+1,incval+1
    goto       loop

    END        resetVec
```

上面汇编代码的功能与下面的 C 代码类似。

```
int add(int a, int b) {
    return a + b;
}

void incr(int val, char amount) {
    return val + amount;
}

void main(void) {
    int result, incval;

    result = 0;
```

```
    incval = 2;
    while(1) {
        result = add(result, incval);
        incval = incr(incval, 2);
    }
}
```

6.1 编译堆栈伪指令

用于控制编译堆栈的汇编器伪指令均以字母 FN 开头。

每个程序中都会使用一次 FNCONF 伪指令。该伪指令有三个参数，分别指示用于保存编译堆栈的 psect 的名称、用于自动对象的符号前缀以及用于参数对象的符号前缀。

在本示例中，该伪指令如下所示：

```
FNCONF udata_acs,?au_,?pa_
```

这表明 udata_acs psect（位于快速操作数据存储区中）将用于保存堆栈中的所有对象。链接器将在一个连续的存储块中形成堆栈。保存堆栈的 psect 的位置将影响堆栈对象的访问方式。如果存在大量基于堆栈的对象（尤其是经常访问这些对象时），将堆栈置于 PIC18 快速操作存储区中意味着无需任何存储区选择指令即可访问这些对象。但是，如果堆栈过大，则需要将其置于分区存储器中。中档和低档器件几乎没有公共存储器，即便有也可能需要用于其他用途，因此在这些器件上通常将分区存储器用于编译堆栈。

FNCONF 伪指令的第二个参数?au_符号将作为程序名称的前缀，用以创建该程序的自动对象的基本符号。?pa_前缀将用于形成每个程序的参数对象的基本符号。这些符号在 add 程序中进行了说明，该程序开头的代码如下所示：

```
PSECT code
;add needs 4 bytes of parameters, but no autos
FNSIZE add,0,4       ;two 2-byte parameters
GLOBAL ?pa_add
;add the two 'int' parameters, returning the result in the first parameter location
add:
    movf       ?pa_add+2,w,c
    addwf      ?pa_add+0,f,c
```

FNSIZE 伪指令有三个参数，分别指示程序的名称、该程序的自动对象所需的总字节数以及该程序的参数对象所需的总字节数。在本示例中，FNSIZE 伪指令指示 add 程序不需要自动对象，需要 4 字节参数。该伪指令可以置于代码中的任何位置，但通常位于其配置的程序附近。

链接器将自动创建一个与程序的参数使用的 4 字节块关联的符号。在本示例中，该符号将为?pa_add（基于 FNCONF 伪指令中使用的前缀）。尽管该符号由链接器定义，但在每个模块中需要使用该符号时仍需要对其进行声明。这在上述代码中是通过 GLOBAL ?pa_add 伪指令实现的。通过使用相对于?pa_add 符号的偏移量，可以访问参数存储器的每个字节。上述代码显示了所访问参数存储器的第一个和第三个字节。这 4 个字节的含义可完全自定义。在本示例中，参数存储器用于保存两个 2 字节宽的对象，但相同的 FNSIZE 参数也可用于创建一个 4 字节宽的对象。

在示例代码的后面，main 程序的定义以下列代码开头。

```
GLOBAL ?au_main
GLOBAL result
result  EQU    ?au_main+0          ;create an alias for this auto location

GLOBAL incval
incval EQU     ?au_main+2          ;create an alias for this auto location

FNROOT main                        ;this is the root of a call graph
FNSIZE main,4,0                    ;main needs two 2-byte 'autos' (for result and incval)
FNCALL main,add                    ;main calls add
FNCALL main,incr                   ;main calls incr

PSECT code
main:
    clrf       result+0,c
    clrf       result+1,c
```

main 程序需要四个字节的自动对象，因此再次使用了 FNSIZE 伪指令进行指示。main 的自动对象存储区域的第一个字节可以使用符号?au_main 访问，但本示例中定义了 EQU，这样做的目的是使用更易读的名称 result，如 clrf 指令中所示。

为了能够在堆栈中分配存储空间，链接器需要了解程序调用结构。为了能够形成程序的调用图，使用了一些其他的伪指令。

FNROOT 伪指令（如上所示）指示调用图中的根节点由指定的程序形成。（或者，也可以使用 resetVec 标签作为根节点，在本示例中没有区别。）分配给堆栈对象的存储空间可以与同一调用图中的其他程序的存储空间重叠，但是，不同调用图中的程序的堆栈对象之间不会发生重叠。通常，将为程序的主要部分定义一个调用图根，然后为每个中断程序定义一个调用图根。这样，与中断程序相关联的堆栈存储器将保持独立，并且不会发生数据损坏。

FNCALL 伪指令可根据需要多次使用，以指示从哪个位置调用哪些程序。通过这种方法，链接器可以形成调用图节点。在上面的代码序列中，FNCALL 伪指令使用了两次。第一次指示 main 程序调用 add;；第二次指示 main 调用 incr。开发程序时，需要确保代码中发生的每次惟一调用都有 FNCALL 伪指令。如果所调用的程序未定义任何编译堆栈对象，则不需要该伪指令，但如果对程序进行了更改，则无论如何都最好包含该伪指令。

用户可以自定义按照何种约定将参数传递给程序。在 add 程序中，第一个参数的 LSB 的偏移量为 0；第一个参数的 MSB 的偏移量为 1，依此类推。因此表达式?pa_add+0 和?pa_add+1 表示第一个“int”参数的两个字节，?pa_add+2 和?pa_add+3 表示第二个参数的两个字节。main 定义的第一个自动对象使用表达式?au_main+0（或 result+0）和?au_main+1（或 result+1）来引用。

通常，需要返回值的程序会将返回值存储到其参数占用的存储空间中。由于程序返回后不应再使用参数，因此重复使用同一存储空间通常不是问题，但在为程序的参数分配存储空间时则需要考虑程序的返回值。使用 FNSIZE 伪指令为程序请求参数存储器时，必须取程序参数总大小与程序返回值大小中的较大者。

6.2 编译器堆栈分配

编译程序时，链接器将处理程序中所有 FN 类型的伪指令，生成程序的调用图，创建堆栈对象（具有 FNSIZE 伪指令的堆栈对象）使用的符号，并以尽可能重叠的方式为这些对象分配存储空间。可以在映射文件中查看该过程的结果。

调用图打印到映射文件顶部，位于链接器选项和编译参数之后。对于本示例，可能如下所示。

```
Call graph: (fully expanded)

*main size 0,4 offset 0
*    add size 4,0 offset 4
     incr size 2,0 offset 4
```

通过缩进指示调用深度。在上面的代码中，可以看到 main 会调用 add，main 还会调用 incr。打印的内容包括为程序的参数和自动对象分配的存储器大小，后跟编译堆栈中的自动参数块（Auto-Parameter Block，APB）偏移量。请注意，偏移量不是地址，只是在堆栈中的相对位置。

请注意，在调用图中，用于 add 和 incr 的 APB 的偏移量相同。这意味着它们共用存储空间，这是有可能的，因为 add 和 incr 在代码中不会相互调用，所以永远不会同时处于活动状态。

程序名称前的星号*指示该程序使用的 APB 存储器处于惟一位置，并且会影响程序占用 RAM 的总大小。这些程序是调用图中的关键路径节点。如果减小这些程序使用的 APB 的大小，程序占用的数据存储器的总大小也会随之减小。未加星号的程序占用的存储块与其他程序的块完全重叠，并且不影响程序占用的数据存储器的总大小。

-mcallgraph 选项可用于自定义映射文件中显示的调用图信息。例如，使用-mcallgraph=crit 将仅显示关键路径上的节点，即图中所有加星号的程序。

在本示例中，使用编译堆栈的优势显而易见：尽管该程序总共需要 10 字节的本地存储空间，但只需分配 8 字节的存储空间，其中 2 个字节可以重复使用。通过这种方法减少存储空间分配时，编程人员不必冒险在程序之间共享对象。

务必要注意，如果由于程序中的 FN 类型伪指令缺失或错误而导致调用图中的信息不准确，则 APB 可能发生错误重叠。可考虑将与调用关联的 FNCALL 伪指令紧接在调用指令本身之前放置，这样便可清楚地看到是否缺少此类型伪指令。不过，FN 类型的伪指令可置于文件中的任何位置，因为这类伪指令不会在 hex 文件中产生内容，并且重复使用同一伪指令不会产生任何坏处。

在映射文件末尾打印的符号表中，还将看到分配给由链接器生成供堆栈使用的特殊符号的地址。对于本程序，可能如下所示。

```
                                Symbol Table

?au_main        udata_acs    000000  ?pa_add         udata_acs    000004
?pa_incr        udata_acs    000004  __Hcode         code         000008
...
incval          udata_acs    000002  result          udata_acs    000000
```

符号表中的值始终为绝对地址，而非调用图中打印的偏移量值。在本示例中，udata_acs psect 链接到地址 0，因此 main 为自动对象（?au_main）占用的存储空间从地址 0 开始且位于 udata_acs psect 中（如我们所指定）。add 为其参数（?pa_add）占用的存储空间从地址 4 开始，与 incr（?pa_incr）使用的块相同。该表中还显示了两个等效的符号（result 和 incval）。

6.3 编译示例

如果本章的示例代码已保存在纯文本文件（例如，名称为 cstack.S）中，则可以使用下面的命令对其进行编译。请注意，文件名使用大写扩展名.S，以便先对文件进行预处理，再进行任何其他处理。

```
pic-as -mcpu=18F47K42 -Wl,-presetVec=0h -Wl,-pudata_acs=COMRAM -Wa,-a -Wl,-Map=cstack.map -
mcallgraph=full cstack.S
```

链接器汇编编译堆栈不需要特殊选项；当传递给链接器的目标文件中存在 FN 类型的伪指令时，将自动创建一个编译堆栈。

上述命令显示了-p 链接器选项，该选项的作用是将用于保存编译堆栈 udata_acs 的 psect 置于合适的链接器类中。此类选项通常不需要使用，但如果不使用，则会抑制在程序中使用 FNCONF 伪指令时产生的警告。

已使用-mcallgraph 选项请求打印完整的调用图。调用图显示在映射文件中，该文件由上述选项-Wl,-Map=cstack.map 请求。

与之前的示例一样，包含复位代码的 psect 已链接到地址 0。该命令还包括用于生成汇编列表文件（cstack.lst）的-Wa,-a 选项，以便浏览生成的代码。

7. 中档器件的中断和位示例

以下示例代码针对 PIC16F18446 编写，将在使用该器件的 Curiosity Nano 开发板上执行。本示例的大部分内容对于其他器件同样适用。该代码将定时器 0 初始化为每 500 ms 产生一次中断，并在每次发生事件时翻转板上 LED（连接到端口位 RA2）。

中断和位示例

```
/*
 * Blink the LED on a PIC16F18446 Curiosity Nano Board
 * using the timer and interrupts to control the flash period.
 */

#include <xc.inc>

CONFIG FEXTOSC = OFF        // External Oscillator mode selection bits->Oscillator not enabled
CONFIG RSTOSC = HFINT1      // Power-up default value for COSC bits->HFINTOSC (1MHz)
CONFIG CLKOUTEN = OFF       // Clock Out Enable->CLKOUT disabled; i/o or osc function on OSC2
CONFIG CSWEN = ON           // Clock Switch Enable bit->Writing to NOSC and NDIV is allowed
CONFIG FCMEN = ON           // Fail-Safe Clock Monitor Enable bit->FSCM timer enabled

CONFIG MCLRE = ON           // Master Clear Enable bit->MCLR pin is Master Clear function
CONFIG PWRTS = OFF          // Power-up Timer Enable bit->PWRT disabled
CONFIG LPBOREN = OFF        // Low-Power BOR enable bit->ULPBOR disabled
CONFIG BOREN = ON           // Brown-out reset enable->Brown-out Reset Enabled, SBOREN ignored
CONFIG BORV = LO            // Brown-out Reset Voltage Selection->VBOR set to 2.45V
CONFIG ZCDDIS = OFF         // Zero-cross detect disable->Zero-cross circuit disabled at POR
CONFIG PPS1WAY = ON         // Peripheral Pin Select control->PPSLOCK cleared/set only once
CONFIG STVREN = ON          // Stack Over/Underflow Reset Enable->Over/Underflow causes reset

CONFIG WDTCPS = WDTCPS_31   // WDT Period Select->Divider 1:65536; software control of WDTPS
CONFIG WDTE = OFF           // WDT operating mode->WDT Disabled, SWDTEN is ignored
CONFIG WDTCWS = WDTCWS_7    // WDT Window Select->window always open (100%); software control
CONFIG WDTCCS = SC          // WDT input clock selector->Software Control

CONFIG BBSIZE = BB512       // Boot Block Size Selection bits->512 words boot block size
CONFIG BBEN = OFF           // Boot Block Enable bit->Boot Block disabled
CONFIG SAFEN = OFF          // SAF Enable bit->SAF disabled
CONFIG WRTAPP = OFF         // Application Block Write Protection->Not write protected
CONFIG WRTB = OFF           // Boot Block Write Protection bit->Block not write protected
CONFIG WRTC = OFF           // Configuration Register Write Protection->Not write protected
CONFIG WRTD = OFF           // Data EEPROM write protection->Data EEPROM NOT write protected
CONFIG WRTSAF = OFF         // Storage Area Flash Write Protection->SAF not write protected
CONFIG LVP = ON             // Low Voltage Programming Enable->LV Programming enabled

CONFIG CP = OFF             // UserNVM Program memory code protection->protection disabled

GLOBAL resetVec,isr
GLOBAL LEDState             ;make this global so it is watchable when debugging

PSECT bitbss,bit,class=BANK1,space=1
LEDState:
    DS          1           ;a single bit used to hold the required LED state

PSECT resetVec,class=CODE,delta=2
resetVec:
    ljmp        start

PSECT isrVec,class=CODE,delta=2
isr:
    ;no context save required in software for this device
    PAGESEL     $           ;select this page for the following goto
    BANKSEL     PIE0        ;for TMR0IE and TMR0IF
    ;for timer interrupts, set the required LED state
    btfsc       TMR0IE
    btfss       TMR0IF
    goto        notTimerInt ;not a timer interrupt
    bcf         TMR0IF
    ;toggle the desired bit state
    movlw       1 shl (LEDState&7)
    BANKSEL     LEDState/8
    xorwf       BANKMASK(LEDState/8),f
```

```
notTimerInt:
    ;code to handle other interrupts could be added here
exitISR:
    ;no context restore required in software
    retfie

PSECT code
start:
    ;set up the state of the oscillator and peripherals with RA2 as a digital output driving
    ;the LED, assuming that other registers have not changed from their reset state
    movlw       0x33
    BANKSEL     TRISA
    movwf       TRISA
    movlw       2
    BANKSEL     OSCFRQ
    movwf       OSCFRQ
    ;configure and start timer using interrupts
    movlw       0x89
    BANKSEL     T0CON1
    movwf       T0CON1
    movlw       0x1D
    movwf       TMR0H
    clrf        TMR0L
    BANKSEL     PIE0        ;for TMR0IE and TMR0IF
    bcf         TMR0IF
    bsf         TMR0IE
    movlw       0x80
    BANKSEL     T0CON0
    movwf       T0CON0
    bsf         GIE
    bsf         PEIE
loop:
    ;copy the desired state to the LED port pin
    BANKSEL     LEDState/8
    btfss       BANKMASK(LEDState/8),LEDState&7
    goto        lightLED
    BANKSEL     PORTA
    bsf         RA2         ;turn LED off
    goto        loop
lightLED:
    BANKSEL     PORTA
    bcf         RA2         ;turn LED on
    goto        loop

    END         resetVec
```

7.1 中断代码（中档器件）

本章的示例说明了如何编写中断服务程序（Interrupt Service Routine，ISR）。

ISR 的编写方式与任何其他程序大体相同，区别在于 ISR 必须：

- 将其入口点链接到相关中断向量的地址，
- 确保自身与主干代码皆使用的所有寄存器的内容得以保存，
- 确保仅为每个中断源执行相关的代码，并且
- 执行特殊的“从中断返回”指令以结束程序和中断处理。

下面针对本示例中的 ISR 就以上几点进行了讨论，相关代码如下所示。

```
PSECT isrVec,class=CODE,delta=2
isr:
    ;no context save required in software for this device
    PAGESEL     $           ;select this page for the following goto
    BANKSEL     PIE0        ;for TMR0IE and TMR0IF
    ;for timer interrupts, set the required LED state
    btfsc       TMR0IE
    btfss       TMR0IF
    goto        notTimerInt ;not a timer interrupt
    bcf         TMR0IF
    ;toggle the desired bit state
    movlw       1 shl (LEDState&7)
    BANKSEL     LEDState/8
```

```
    xorwf       BANKMASK(LEDState/8),f
notTimerInt:
    ;code to handle other interrupts could be added here
exitISR:
    ;no context restore required in software
    retfie
```

ISR 的入口点（如同在复位后执行的代码的入口点）必须链接到目标器件的相应中断向量地址。为此，应将 ISR 或至少是包含 ISR 入口点的代码置于可以链接到所需地址的惟一 psect 中。在本示例中，`isrVec` psect 保存整个中断程序。某些 PIC18 器件可以使用中断向量表，并对 ISR 具有不同的链接要求，如 8. PIC18 器件的中断和位示例所述。

ISR（及其调用的任何程序）修改的任何寄存器以及在主干代码中使用的任何寄存器均必须在进入 ISR 时保存，然后在退出 ISR 时恢复。对于支持多个中断向量的器件（例如 PIC18 器件），ISR 将需要保存所有已被其他 ISR 修改但未保存的寄存器。用于保存和恢复寄存器的代码通常称为现场切换代码。

与许多现代 PIC 器件一样，16F18446 会在发生中断时自动将内核寄存器的状态保存到影子寄存器中，因此除非存在 ISR 不得修改的其他寄存器或对象，否则通常不需要通过软件执行现场切换。上面的示例代码未包含任何现场切换代码，而是立即处理中断。如果需要编写现场切换代码，请参见 7.3 手动现场切换。

当发生定时器 0 中断时，此处显示的 ISR 将翻转所需 LED 状态。它不对任何其他中断源执行操作，但需要时可以向该 ISR 添加代码以处理尽可能多的中断源。`btfss TMR0IF` 指令通过检查相关的定时器中断标志来确保定时器 0 已触发，但它需与 `btfsc TMR0IE` 指令配合使用，后者检查定时器 0 的中断允许位。只有当这两个位均置 1 时，才能确保定时器 0 是中断源。

要终止执行中断代码并返回到主干代码，应使用所示的 `retfie` 指令。返回指令不会将器件的状态恢复为发生中断时的状态，并且会导致代码失败。

由于该 ISR 包含 `goto` 指令，所以添加了 `PAGESEL` 伪指令以确保选择包含 ISR 的页，`goto` 也因此将到达预期目标。

7.2 定义和使用位

本章所示的中断程序说明了如何修改主干代码用来设置 LED 状态的标志。此标志的状态由 ISR 设置。由于此标志仅需要保存 true 或 false 值，因此已将其定义为位对象，以便节省存储空间和更加高效地检查其内容。

位对象的创建方式与普通对象类似，只是必须为用于保存位对象的 psect 搭配一个特殊的标志。以下代码行定义了一个宽度为一位的对象，名称为 `LEDState`：

```
PSECT    bitbss,bit,class=BANK1,space=1
LEDState:
    DS          1           ;a single bit used to hold the required LED state
```

psect 定义中包含的 `bit` 标志会告知链接器该 psect 中的地址单位是位，而不是字节。这意味着分配一个存储单位的 `DS 1` 伪指令将保留一个位，而不是一个字节。因此，`LEDState` 对象只能保存一个位。

链接器将为存储器的每个字节分配 8 位对象，因此，如果对上述 psect 进行了其他分配（如下所示）：

```
PSECT    bitbss,bit,class=BANK1,space=1
LEDState:
    DS          1           ;a single bit used to hold the required LED state
otherState:
    DS          1           ;a single bit for some other purpose
```

则这两个位将位于存储器的同一字节中，但它们在该字节中的具体位位置当然是不同的。

在位 psect 中定义的任何符号（例如 `LEDState`）都代表位地址，而不是字节地址，这会影响这些符号在指令中的使用方式。执行位操作的 PIC 指令（例如 `bcf` 或 `btfss`）需要字节地址操作数，后跟该字节中的位位置。例如，位地址 0x283 中的位对象位于字节地址 `0x283/8`（即 0x50）和位位置 `0x283&7`（即位置 3）。如果要将位对象与位指令一起使用，则需要将位地址除以 8 以获得指令中使用的字节（文件寄存器）地址，并且需要与 7 进行按位与运算以获得该字节内的位位置。例如，读取所需 LED 状态的主干示例代码使用：

```
    BANKSEL     LEDState/8
    btfss       BANKMASK(LEDState/8),LEDState&7
```

请注意，BANKSEL 伪指令也需要字节地址参数，因此该指令的 LEDState 的位地址也除以 8。BANKMASK()宏也已按常规方式与 btfss 指令的文件寄存器操作数配合使用，但应确保该宏作用于位对象的字节地址（即，位地址除以 8）。

可以根据需要创建预处理器宏，以使对象的文件地址和位位置更易读，例如

```
#define  LEDSTATE  BANKMASK(LEDState/8),LEDState&7
```

随后可按以下方式使用：

```
    BANKSEL LEDState/8
    btfss       LEDSTATE
```

请注意，SFR 中的位按照与上文的 LEDSTATE 类似的方式在提供的头文件中预定义，因此使用方式也相同。例如，中断中的代码序列：

```
    btfsc       TMR0IE
    btfss       TMR0IF
    goto        notTimerInt ;not a timer interrupt
    bcf         TMR0IF
```

使用了 TMR0IE 和 TMR0IF SFR 位，这两个位均扩展到保存它们的字节地址（分别是 PIE0 和 PIR0 的地址）及其在这些字节中的位位置。在源文件中包含<xc.inc>后，即可使用这些 SFR 位。

在位 psect 中定义的任何符号的链接地址都作为位地址打印在列表和映射文件中。检查存储器分配时，请勿将此类地址与其他字节地址进行比较。可以通过在映射文件中查找 **Scale** 值来确认哪些 psect 使用了 bit 标志。对于位 psect，Scale 值指示为 8；对于非位 psect，Scale 值将留空。在为本示例项目生成的映射文件中显示了 bitbss psect。

```
        Name                                Link     Load   Length Selector   Space Scale
build/default/debug/main.o
        config                              8007     8007        5        0       4
        isrVec                                 4        4        A        8       0
        resetVec                               0        0        3        0       0
        bitbss                               500       A0        1       A0       1     8
        code                                   E        E       1D        8       0
```

请注意链接地址为 0x500。这是位地址。但是，装载地址将转换为字节地址，并显示为 0xA0。

位 psect 可以链接到数据存储器中的任何位置。为了突出显示如何针对位对象使用存储器分区，本示例中的 psect 包含相应的标志以与 BANK1 链接器类相关联，这意味着它自动置于数据存储器存储区 1 中的某个位置。为了更加高效地访问位对象，可尝试将其置于 PIC18 器件的公共存储器或快速操作存储区中。

7.3 手动现场切换

如果器件在发生中断时未自动将内核寄存器的状态保存到影子寄存器，或者除了硬件保存的寄存器或对象之外还需要保存其他寄存器或对象，则需要执行以下步骤。

- 在 RAM 中为必须保存的寄存器定义足够的存储空间
- 在修改这些寄存器之前，让 ISR 将相关寄存器的内容复制到存储区
- 当不再需要使用这些寄存器后，让 ISR 从存储区恢复这些寄存器的内容

以下代码段针对在发生中断时不会自动保存现场的中档器件（例如 PIC12F609）编写。该代码假定器件实现了公共存储器。该代码说明了以上几点，具体将在以下段落中进行介绍。该代码可根据具体的器件和希望保存的寄存器进行调整。

```
#include <xc.inc>

PSECT udata_shr
saved_WREG:
    DS      1               ;space for WREG in common memory

PSECT udata_bank0
mainSaveArea:               ;space for other registers in banked memory
saved_STATUS:
    DS      1
saved_PCLATH:
```

```
    DS      1

PSECT isrVec,class=CODE,delta=2
isr:
    ;save context
    movwf   saved_WREG      ;WREG saved to common memory
    movf    STATUS,w        ;STATUS (including bank selection bits) copied to WREG...
    BANKSEL mainSaveArea    ;...allowing a different bank to be selected...
    movwf   saved_STATUS    ;STATUS saved
    movf    PCLATH,w
    movwf   saved_PCLATH    ;PCLATH saved

;interrupt code that can use STATUS, WREG, PCLATH goes here

exitISR:
    ;restore context
    BANKSEL mainSaveArea
    movf    saved_PCLATH,w
    movwf   PCLATH          ;PCLATH restored
    movf    saved_STATUS,w
    movwf   STATUS          ;STATUS restored
    swapf   saved_WREG,f
    swapf   saved_WREG,w    ;WREG restored without altering STATUS

    retfie
```

必须为要保存的所有寄存器分配空间。该空间不必是连续的，甚至不必在同一存储区中。将现场存储在公共存储器或 PIC18 快速操作存储区中可使现场切换代码变得更加简单，但这样会造成公共存储空间不足，无法供所有寄存器使用，而且将这种宝贵的存储器资源仅用于中断有些得不偿失。

在本示例中，已（使用 DS 伪指令）在公共存储器中分配了一个字节（udata_shr psect）来保存 W 寄存器的状态，并且在存储区 0 的一个存储块（udata_bank0 psect）中分配了存储器空间来存储 STATUS 和 PCLATH 寄存器的状态。这种配置不仅可简化现场切换代码，还不会占用太多的公共存储空间，具体如下所述。通过为存储空间中的每个字节定义一个标签（如本示例中所示）可以惟一引用每个字节，或者也可以仅创建一个标签，然后使用相对于该标签的偏移量来访问该存储块中的每个字节。例如，mainSaveArea + 1 和 saved_PCLATH 表示同一个存储单元。

ISR 开头的代码将寄存器复制到分配的存储空间中。必须确保该代码不会改写必须保存的任何寄存器，直到这些寄存器的内容完成保存为止。这意味着可能需要按特定顺序保存寄存器。

上例中的第一条指令将 W 寄存器的当前内容传送到 saved_WREG 对象中。由于该对象的存储空间已分配到公共存储器中，因此无需选择该存储器的存储区。这一点很重要，因为在中档器件上，存储区选择位位于 STATUS 寄存器中，而该寄存器是必须保存的寄存器之一。不破坏 STATUS 寄存器就无法选择新的存储区；同理，先保存 STATUS 寄存器，代码就会破坏 W 寄存器。

保存 W 寄存器后，随后的指令即可将 STATUS 寄存器的内容装载到 W 寄存器中。目前 STATUS 寄存器已复制（尽管尚未存储），因此可以使用 BANKSEL 伪指令选择主存储块的存储区。随后，后续代码可以使用常规 movf/movwf 指令保存剩余寄存器。

ISR 末尾的恢复代码恢复寄存器的顺序通常与保存寄存器时的顺序相反。必须确保恢复后的寄存器不会再有代码写入，这再次说明恢复操作需要按照严格的顺序执行并且可能无法使用某些指令。

本示例中的恢复代码先选择主要分配空间的存储区，然后使用 movf/movwf 指令将保存的值复制回 PCLATH 和 STATUS 寄存器。请记住，当 STATUS 寄存器恢复后，当前选择的存储区也将恢复到中断发生时的状态。由于本示例中要恢复的剩余寄存器的内容已存储在公共存储器中，因此无需担心。

恢复 W 寄存器更加困难。无法使用指令 movf saved_WREG,w，因为它会影响已恢复 STATUS 寄存器中的零标志（Z）位。示例代码改用不会影响 STATUS 寄存器的两条 swapf 指令恢复 W 寄存器。

7.4 编译示例

如果本章的源代码已保存在纯文本文件（例如，名称为 timedLED.S）中，则可以使用下面的命令对其进行编译。请注意，文件名使用大写扩展名.S，以便先对文件进行预处理，再进行任何其他处理。

```
pic-as -mcpu=16F18446 -Wl,-presetVec=0h -Wl,-pisrVec=04h -Wa,-a -Wl,-Map=timedLED.map
timedLED.S
```

包含复位代码的 psect 已链接到地址 0。此外，保存中断程序的 psect 已链接到器件的向量位置（地址 0x4）。该命令还包括-Wl,-Map=timedLED.map 选项（用于生成映射文件）和-Wa,-a 选项（用于生成名为 timedLED.lst 的汇编列表文件）。

8. PIC18 器件的中断和位示例

以下示例代码针对 PIC18F47K42 编写，将在使用该器件的 Curiosity Nano 开发板上执行。该代码将定时器 0 初始化为每 500 ms 产生一次中断，并在每次发生事件时翻转板上 LED（连接到端口位 RE0）。

中断和位示例

```
/*
 * Blink the LED on a PIC18F47K42 Curiosity Nano Board
 * using the timer and interrupts to control the flash period.
 */

#include <xc.inc>

CONFIG FEXTOSC = OFF          // External Oscillator Selection->Oscillator not enabled
CONFIG RSTOSC = HFINTOSC_1MHZ // Reset Oscillator Selection->HFINTOSC, HFFRQ=4MHz, CDIV=4:1
CONFIG CLKOUTEN = OFF         // Clock out Enable bit->CLKOUT function is disabled
CONFIG PR1WAY = ON            // PRLOCKED One-Way Set Enable->PRLOCK cleared/set only once
CONFIG CSWEN = ON             // Clock Switch Enable->Writing to NOSC and NDIV is allowed
CONFIG FCMEN = ON             // Fail-Safe Clock Monitor Enable->Clock Monitor enabled

CONFIG MCLRE = EXTMCLR        // MCLR Enable->LVP=0=>MCLR pin is MCLR; LVP=1=>RE3 pin is MCLR
CONFIG PWRTS = PWRT_OFF       // Power-up timer selection bits->PWRT is disabled
CONFIG MVECEN = ON            // Multi-vector enable->Multi-vector table enabled
CONFIG IVT1WAY = ON           // IVTLOCK One-way set enable->IVTLOCK cleared/set only once
CONFIG LPBOREN = OFF          // Low Power BOR Enable bit->ULPBOR disabled
CONFIG BOREN = SBORDIS        // Brown-out Reset Enable->BOR enabled, SBOREN ignored
CONFIG BORV = VBOR_2P45       // Brown-out Reset Voltage Selection->VBOR set to 2.45V
CONFIG ZCD = OFF              // ZCD Disable->ZCD disabled; enable by setting ZCDSEN
CONFIG PPS1WAY = ON           // PPSLOCK One-Way Set Enable->PPSLOCK cleared/set only once
CONFIG STVREN = ON            // Stack Full/Underflow Reset Enable->Full/underflow => Reset
CONFIG DEBUG = OFF            // Debugger Enable bit->Background debugger disabled
CONFIG XINST = OFF            // Extended Instruction Set Enable->Extended Set disabled

CONFIG WDTCPS = WDTCPS_31     // WDT Period selection->Divider 1:65536; s/w control of WDTPS
CONFIG WDTE = OFF             // WDT operating mode->WDT Disabled; SWDTEN is ignored
CONFIG WDTCWS = WDTCWS_7      // WDT Window Select->window open (100%); software control
CONFIG WDTCCS = SC            // WDT input clock selector->Software Control

CONFIG BBSIZE = BBSIZE_512    // Boot Block Size selection->Boot Block size is 512 words
CONFIG BBEN = OFF             // Boot Block enable bit->Boot block disabled
CONFIG SAFEN = OFF            // Storage Area Flash enable bit->SAF disabled
CONFIG WRTAPP = OFF           // Application Block write protection->Block not protected
CONFIG WRTB = OFF             // Configuration Register Write Protection->Not protected
CONFIG WRTC = OFF             // Boot Block Write Protection->Boot Block not write-protected
CONFIG WRTD = OFF             // Data EEPROM Write Protection->Data EEPROM not write-protected
CONFIG WRTSAF = OFF           // SAF Write protection bit->SAF not Write Protected
CONFIG LVP = ON               // Low Voltage Programming Enable->LVP enabled
CONFIG CP = OFF               // PFM and Data EEPROM Code Protection->code protection disabled

GLOBAL resetVec
GLOBAL LEDState               ;make this global so it is watchable when debugging
GLOBAL __Livt                 ;defined by the linker but used in this code

PSECT bitbssCOMMON,bit,class=COMRAM,space=1
LEDState:
    DS          1             ;a single bit used to hold the required LED state

PSECT resetVec,class=CODE,reloc=2
resetVec:
    goto        start

PSECT ivt,class=CODE,reloc=2,ovrld
ivtbase:
    ORG         31*2          ;timer 0 vector position
    DW          tmr0Isr shr 2

PSECT code
tmr0Isr:
    bcf         TMR0IF
    ;toggle the desired LED state
    movlw       1 shl (LEDState&7)
    xorwf       LEDState/(0+8),c
```

```
    retfie      f

PSECT code
start:
    bsf         BANKMASK(INTCON0),INTCON0_IPEN_POSN,c        ;set IPEN bit
    ;use the unlock sequence to set the vector table position
    ;based on where the ivt psect is linked
    bcf         GIE
    movlw       0x55
    movwf       BANKMASK(IVTLOCK),c
    movlw       0xAA
    movwf       BANKMASK(IVTLOCK),c
    bcf         IVTLOCKED
    movlw       low highword __Livt
    movwf       BANKMASK(IVTBASEU),c
    movlw       high __Livt
    movwf       BANKMASK(IVTBASEH),c
    movlw       low __Livt
    movwf       BANKMASK(IVTBASEL),c
    movlw       0x55
    movwf       BANKMASK(IVTLOCK),c
    movlw       0xAA
    movwf       BANKMASK(IVTLOCK),c
    bsf         IVTLOCKED
    ;set up the state of the oscillator and peripherals with RE0 as a digital output driving
    ;the LED, assuming that other registers have not changed from their reset state
    movlw       6
    movwf       BANKMASK(TRISE),c
    movlw       0x62
    movlb       57
    movwf       BANKMASK(OSCCON1),b
    clrf        BANKMASK(OSCCON3),b
    clrf        BANKMASK(OSCEN),b
    movlw       2
    movwf       BANKMASK(OSCFRQ),b
    clrf        BANKMASK(OSCTUNE),b
    ;configure and start timer interrupts
    movlb       57
    bsf         TMR0IP
    movlw       0x6D
    movwf       BANKMASK(T0CON1),c
    movlw       0xF3
    movwf       BANKMASK(TMR0H),c
    clrf        BANKMASK(TMR0L),c
    movlb       57
    bcf         TMR0IF
    bsf         TMR0IE
    movlw       0x80
    movwf       BANKMASK(T0CON0),c
    bsf         GIEH

loop:
    ;set LED state to be that requested by the interrupt code
    btfss       LEDState/8,LEDState&7,c
    goto        lightLED
    bsf         RE0          ;turn LED off
    goto        loop
lightLED:
    bcf         RE0          ;turn LED on
    goto        loop

    END         resetVec
```

8.1 中断代码（PIC18）

7.1 中断代码（中档器件）中的大部分讨论对于 PIC18 中断代码同样适用，此处不再赘述。本节将讨论 PIC18 器件特定的中断相关细节。

本示例中使用的某些 PIC18 器件（例如 PIC18F47K42 器件）可配置为使用中断向量表，而不是使用两个固定的中断向量地址。如果 PIC18 器件不支持向量表，或者以传统模式使用该器件，则需要定义一个或两个中断服务程序（每个

中断优先级使用一个），其入口点链接到中断向量地址。这种方法与中档器件示例中所示的方法类似；但是，中档器件仅支持一个中断向量地址。

本章中的示例假定使用向量中断控制器。开启 MVECEN 配置位可使能该工作模式。

使用向量表时，将为每个要处理的中断源编写一个独立的中断服务程序（ISR）。创建包含这些 ISR 地址的向量表并将其置于存储器中。向量在表中的位置决定它将处理哪个中断源。表的基址必须存储在 IVTBASE 寄存器中，以便在发生中断时，器件可装载正确的中断向量。IVTBASE 寄存器的内容可在运行时更改，这意味着一个程序可以定义多个中断向量表，因此可以在程序的不同位置选择不同的 ISR。

本示例中的中断代码（如下所示）给出了一个 ISR 及其在向量表中的对应地址。

```
PSECT ivt,class=CODE,reloc=2,ovrld
ivtbase:
    ORG         31*2          ;timer 0 vector position
    DW          tmr0Isr shr 2

PSECT code
tmr0Isr:
    bcf         TMR0IF
    ;toggle the desired LED state
    movlw       1 shl (LEDState&7)
    xorwf       LEDState/(0+8),c

    retfie      f
```

本示例中用于处理定时器 0 的 ISR 使用标签 tmr0Isr，并且位于 code psect 中。该 psect 可以链接到程序存储器中的任何位置，因此不需要用户定义的 psect 或特殊的链接器选项。

已使用一条 retfie f 指令从中断返回。与许多现代器件一样，PIC18F47K42 会在发生中断时自动将内核寄存器的状态保存到影子寄存器中，因此除非存在 ISR 不得修改的其他寄存器或对象，否则通常不需要通过软件执行现场切换。上面的示例代码未包含任何现场切换代码，而是立即处理中断。如果需要编写现场切换代码，请参见 7.3 手动现场切换。f 操作数表示 retfie 指令的快速形式，该指令会将影子寄存器的内容复制回原先的寄存器。

向量表已置于名为 ivt 的用户定义 psect 中，该 psect 随后可独立链接到程序存储器中的其他 psect，如 8.3 编译示例所示。已使用 reloc=2 标志确保 psect 的起始地址按字对齐，以便正常工作，具体如器件数据手册中所述。ovrld psect 标志的用法将稍后介绍。

已使用汇编器的 DW 伪指令将 ISR 的地址 tmr0Isr 置于向量表中。器件期望该地址右移 2 位，因此使用了汇编器的 shr 运算符来获取该值。定时器 0 的中断向量必须位于表中的位置 31；但是，表本身可以在程序存储器中移动，因此该向量（或任何其他向量）没有绝对位置。为了便于修改程序，使用了 ORG 伪指令来定位向量。在这种情况下，建议使用该伪指令。请注意，ORG 伪指令将位置计数器*相对于当前 psect 的起始位置*移动（而不使用绝对位置），但这正好符合这种情况下的要求。如果用于保存向量表的 psect 链接到所需的地址，则 ORG 伪指令将确保向量处于正确的偏移量。由于每个向量的宽度为两个字节，因此向量的偏移量为 31 * 2。

随着程序不断开发，可能需要其他中断，此时可以编写其他 ISR，并将这些中断服务程序的地址添加到向量表中。用于保存向量表的 psect 可以按段和跨多个文件编译。此外，也可以在一个专属的源文件中提供一个 ISR 及其向量表。

为了确保 psect 中的向量以正确的顺序出现（而不管每个向量在何时何处添加），已为 ivt psect 搭配使用了 ovrld 标志。该标志会告知链接器重叠而不是连接 psect 的每一部分内容。例如，以下附加代码（可能位于不同的源文件中）用于放置 UART1 接收中断程序的向量。

```
PSECT ivt,class=CODE,reloc=2,ovrld
    ORG         27*2          ;UART1 receive vector position
    DW          uart1Isr shr 2

PSECT code
uart1Isr:
    ;code to process uart1 receive goes here
```

请注意，该向量所处的 psect（ivt）也用于保存定时器中断的向量。由于已为该 psect 搭配使用了 ovrld 标志，因此即使已经处理了第一个示例中 ivt psect 的内容，ORG 伪指令也会再次移动随后的地址（相对于 psect 的起始地址）。这样便无需确保向量以任何特定顺序进行处理。

如果检查映射文件中的向量表 psect，则应看到链接地址指示其所处的地址（以下示例中为地址 8），其长度将包括单条或多条 ORG 伪指令引入的任何间隙。尽管本示例的向量表中只有一个向量，但由于定时器 0 向量位于距 psect 开头的偏移量为 31 * 2 的位置且长度为 2 字节，因此 psect 长度已显示为 0x40。

```
TOTAL          Name                    Link     Load   Length     Space
  CLASS   CODE
                ivt                       8        8       40         0
```

即使项目不需要来自其他中断源的中断，建议也要考虑这些中断源中的任何一个意外触发时将会发生什么。此外，也可以使用能够处理这些情况的默认 ISR 的地址来填充未使用的向量表位置。

通过将适当的表地址写入 IVTBASE 寄存器，可以在程序存储器中定位（和重新定位）向量表。为了避免在决定将向量表链接到其他地址时必须修改源代码，可以使用由链接器定义的特殊符号装载 IVTBASE 寄存器。下面的示例摘录部分给出了 IVTBASE 寄存器的解锁序列（如器件数据手册中所述）以及装载的全部三个 IVTBASE 寄存器。

```
GLOBAL __Livt                ;defined by the linker but used in this code
...
    movlw       0x55
    movwf       BANKMASK(IVTLOCK),c
    movlw       0xAA
    movwf       BANKMASK(IVTLOCK),c
    bcf         IVTLOCKED
    movlw       low highword __Livt
    movwf       BANKMASK(IVTBASEU),c
    movlw       high __Livt
    movwf       BANKMASK(IVTBASEH),c
    movlw       low __Livt
    movwf       BANKMASK(IVTBASEL),c
```

对于使用-p 链接器选项的任何 psect 分配的存储器，链接器将定义特殊的符号来表示这些 psect 在存储器中的链接位置。此类 psect 的起始地址以__L*psectname* 形式的符号表示（注意开头有两个下划线字符）。（链接器还定义了__H*psectname* 和__B*psectname* 符号。）在本示例中，已使用__Livt 符号（ivt psect 的下界）装载 IVTBASE 寄存器。已使用 low、high 和 highword 操作符为每个寄存器获取该完整地址的相应字节。如果稍后要更改向量表的位置，无需修改此代码；只需将程序重新链接到其他 ivt 地址即可。

8.2 定义和使用位

本章所示的中断程序说明了如何修改主干代码用来设置 LED 状态的标志。此标志的状态由 ISR 设置。由于此标志仅需要保存 true 或 false 值，因此已将其定义为位对象，以便节省存储空间和更加高效地检查其内容。

PIC18 器件位对象的创建方式与中档器件相同（在 7.2 定义和使用位中讨论），将为保存位对象的 psect 搭配使用 bit psect 标志。与中档器件相比，以下代码摘录部分的惟一区别是 psect 置于快速操作存储区中。

```
PSECT bitbssCOMMON,bit,class=COMRAM,space=1
LEDState:
    DS          1               ;a single bit used to hold the required LED state
```

PIC18 代码中的位符号的使用方式与中档器件相同。可以通过将位地址除以 8 来获得位符号的文件寄存器地址，并且可以通过与 7 进行按位与运算来获得位位置。此外，一旦在程序中包含<xc.inc>，就会为示例代码中使用的特殊功能寄存器（例如 TMR0IF 和 GIEH）中的位定义位符号，这一点与中档器件相同。

您可能会发现示例（如下所示）中未使用 bsf IPEN 等指令来将 INTCON0 寄存器中的 IPEN 位置 1，而是将该指令的操作数指定为 INTCON0 文件寄存器地址和用于定位该寄存器中 IPEN 位的特殊符号。

```
PSECT code
start:
    bsf         BANKMASK(INTCON0),INTCON0_IPEN_POSN,c        ;set IPEN bit
```

在这种特殊情况下，尝试直接使用 IPEN 位将导致未定义的符号错误。这是因为 18F47K42 器件的 SFR 中有多个名为 IPEN 的位。例如，ADCON1 寄存器中也有 IPEN 位。在这种情况下，名称 IPEN 不是惟一的，并且所提供的头文件中没有位符号。必须使用 SFR 名称访问该位，但是头文件确实提供了表示位位置的宏，例如上面使用的 INTCON0_IPEN_POSN。这些头文件还提供了指示 SFR 中的位数的宏（例如_INTCON0_IPEN_SIZE）和可用于按位

逻辑运算的掩码（例如_INTCON0_IPEN_MASK）。这些宏可用于每个 SFR 位，如果需要，可优先使用宏来代替位名称。

8.3 编译示例

如果本章所示的源代码已保存在纯文本文件（例如，名称为 timedLED.S）中，则可以使用下面的命令对其进行编译。请注意，文件名使用大写扩展名.S，以便先对文件进行预处理，再进行任何其他处理。

```
pic-as -mcpu=18F47K42 -Wl,-presetVec=0h -Wl,-pivt=08h -Wa,-a -Wl,-Map=timedLED.map timedLED.S
```

用于保存向量表的 ivt psect 链接到地址 0x8。该 psect 的链接地址在源代码中用于装载 IVTBASE 寄存器中包含的值，以确保在发生中断时器件读取正确的向量。为该 psect 指定的地址必须等于 2 的整数数，以确保满足与该 psect 搭配使用的 reloc 标志的要求。如果尝试将 psect 链接到一个奇地址，则将报错。

上述命令还包括-Wl,-Map=timedLED.map 选项（用于生成映射文件）和-Wa,-a 选项（用于生成名为 timedLED.lst 的汇编列表文件）。

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指标。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿意与关心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。

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担

保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。Microchip 对因这些信息及使用这些信息而引起的后果不承担任何责任。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伤害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任，并加以赔偿。除非另外声明，否则在 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、TimeSource、tinyAVR、UNI/O、Vectron 及 XMEGA 均为 Microchip Technology Incorporated 在美国和其他国家或地区的注册商标。

AgileSwitch、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、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、Augmented Switching、BlueSky、BodyCom、CodeGuard、CryptoAuthentication、CryptoAutomotive、CryptoCompanion、CryptoController、dsPICDEM、dsPICDEM.net、Dynamic Average Matching、DAM、ECAN、Espresso T1S、EtherGREEN、IdealBridge、In-Circuit Serial Programming、ICSP、INICnet、Intelligent Paralleling、Inter-Chip Connectivity、JitterBlocker、maxCrypto、maxView、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、RTAX、RTG4、SAM-ICE、Serial Quad I/O、simpleMAP、SimpliPHY、SmartBuffer、SMART-I.S.、storClad、SQI、SuperSwitcher、SuperSwitcher II、Switchtec、SynchroPHY、Total Endurance、TSHARC、USBCheck、VariSense、VectorBlox、VeriPHY、ViewSpan、WiperLock、XpressConnect 和 ZENA 均为 Microchip Technology Incorporated 在美国和其他国家或地区的商标。

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

Adaptec 徽标、Frequency on Demand、Silicon Storage Technology 和 Symmcom 均为 Microchip Technology Inc.在除美国外的国家或地区的注册商标。

GestIC 为 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： 978-1-5224-8125-6

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