MPLAB® XC32汇编器、链接器和实用程序用户指南

© 2014 Microchip Technology Inc.

请注意以下有关 Microchip 器件代码保护功能的要点：

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

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

提供本文档的中文版本仅为了便于理解。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。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、 dsPIC、 FlashFlex、 KEELOQ、 KEELOQ 徽标、 MPLAB、 PIC、 PICmicro、 PICSTART、 PIC32 徽标、 rfPIC、 SST、 SST 徽标、 SuperFlash 和 UNI/O 均为 Microchip Technology Inc. 在美国和其他国家或地区的注册商标。

FilterLab、 Hampshire、 HI-TECH C、 Linear Active Thermistor、 MTP、 SEEVAL 和 The Embedded Control Solutions Company 均为 Microchip Technology Inc. 在美国的注册商标。

Silicon Storage Technology 为 Microchip Technology Inc. 在除美国外的国家或地区的注册商标。

Analog-for-the-Digital Age、 Application Maestro、 BodyCom、 chipKIT、 chipKIT 徽标、 CodeGuard、 dsPICDEM、 dsPICDEM.net、 dsPICworks、 dsSPEAK、 ECAN、 ECONOMONITOR、 FanSense、 HI-TIDE、 In-Circuit Serial Programming、 ICSP、 Mindi、 MiWi、 MPASM、 MPF、 MPLAB Certified 徽标、 MPLIB、 MPLINK、 mTouch、 Omniscient Code Generation、 PICC、 PICC-18、 PICDEM、 PICDEM.net、 PICkit、 PICtail、 REAL ICE、 rfLAB、 Select Mode、 SQI、 Serial Quad I/O、 Total Endurance、 TSHARC、 UniWinDriver、 WiperLock、 ZENA 和 Z-Scale 均为 Microchip Technology Inc. 在美国和其他国家或地区的商标。

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

GestIC 和 ULPP 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。

© 2014, Microchip Technology Inc. 版权所有。

ISBN：978-1-62077-821-0

QUALITY MANAGEMENT SYSTEM CERTIFIED BY DNV

═ ISO/TS 16949 ═

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham 的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2009 认证。 Microchip 的 PIC® MCU 与 dsPIC® DSC、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品严格遵守公司的质量体系流程。此外， Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

© 2014 Microchip Technology Inc.

目录

前言 7

第1部分—— MPLAB XC32汇编器

第1章 汇编器概述

1.1 简介 15
1.2 汇编器和其他开发工具 15
1.3 特性集 16
1.4 输入/输出文件 17

第2章 汇编器命令行选项

2.1 简介 23
2.2 汇编器接口语法 23
2.3 编译驱动程序接口语法 24
2.4 修改列表输出的选项 25
2.5 控制信息输出的选项 37
2.6 控制输出文件创建的选项 38
2.7 汇编器符号定义和搜索路径选项 39
2.8 编译驱动程序和预处理程序选项 40

第3章 MPLAB XC32汇编语言

3.1 简介 41
3.2 内部预处理程序 42
3.3 源代码格式 43
3.4 特殊字符 48
3.5 符号 51
3.6 为符号赋予其他值 52
3.7 特殊点符号 52
3.8 表达式 53
3.9 运算符 53
3.10 特殊操作符 55

第4章 汇编器伪指令

4.1 简介 57
4.2 定义段的伪指令 58
4.3 初始化常量的伪指令 62
4.4 声明符号的伪指令 64
4.5 定义符号的伪指令 65
4.6 修改段对齐方式的伪指令 66
4.7 格式化输出列表的伪指令 68

© 2014 Microchip Technology Inc.

4.8 控制条件汇编的伪指令 69
4.9 用于替换/扩展的伪指令 71
4.10 包含其他文件的伪指令 75
4.11 控制诊断输出的伪指令 76
4.12 用于调试信息的伪指令 77
4.13 控制代码生成的伪指令 79

第5章 汇编器错误/警告/消息

5.1 简介 81
5.2 严重错误 82
5.3 错误 83
5.4 警告 90
5.5 消息 94

第2部分—— MPLAB XC32目标链接器

第6章 链接器概述

6.1 简介 97
6.2 链接器和其他开发工具 97
6.3 特性集 98
6.4 输入/输出文件 98

第7章 链接器命令行接口

7.1 简介 105
7.2 链接器接口语法 106
7.3 编译驱动程序链接器接口语法 107
7.4 控制输出文件创建的选项 108
7.5 控制运行时初始化的选项 113
7.6 控制multilib库选择的选项 114
7.7 控制信息输出的选项 115
7.8 修改链接映射输出的选项 118

第8章 链接描述文件

8.1 简介 119
8.2 链接描述文件概述 120
8.3 命令行信息 120
8.4 默认链接描述文件 121
8.5 添加定制链接描述文件到MPLAB X IDE项目 123
8.6 链接描述文件命令语言 124
8.7 链接描述文件中的表达式 140

第9章 链接器处理

9.1 简介 147
9.2 链接器处理概述 148
9.3 链接器分配 150
9.4 全局符号和弱符号 153
9.5 已初始化数据 154

© 2014 Microchip Technology Inc.

9.6 堆栈分配 157
9.7 堆分配 157
9.8 PIC32MX中断向量表 158
9.9 具有专用可编程可变偏移量的PIC32 MCU中断向量表 159

第10章 链接器示例

10.1 简介 163
10.2 重点 163
10.3 存储器地址和可重定位代码 164
10.4 定位变量到特定地址 165
10.5 定位函数到特定地址 165
10.6 定位和保留程序存储器 166

第11章 链接器错误/警告

11.1 简介 167
11.2 严重错误 168
11.3 错误 169
11.4 警告 172

第3部分—— 32位实用程序（包括归档器/库管理器）

第12章 MPLAB XC32目标归档器/库管理器

12.1 简介 175
12.2 归档器/库管理器和其他开发工具 176
12.3 特性集 177
12.4 输入/输出文件 177
12.5 语法 177
12.6 选项 178
12.7 描述文件 180

第13章 其他实用程序

13.1 简介 183
13.2 xc32-bin2hex实用程序 184
13.3 xc32-nm实用程序 185
13.4 xc32-objdump实用程序 188
13.5 xc32-ranlib实用程序 191
13.6 xc32-size实用程序 192
13.7 xc32-strings实用程序 194
13.8 xc32-strip实用程序 195

第4部分—— 附录

附录A 已弃用的功能

A.1 简介 199
A.2 定义段的汇编器伪指令 200

附录B 有用的表格
B.1 简介 201
B.2 ASCII字符集 201
B.3 十六进制到十进制的转换 202
附录C GNU自由文档许可证
术语表 205
索引 225
全球销售及服务网点 234

© 2014 Microchip Technology Inc.

前言

客户须知

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

文档均标记有 **“DS”**编号。该编号出现在每页底部的页码之前。 **DS** 编号的命名约定为 **“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，请参考 **MPLAB® IDE** 在线帮助。从 **Help** （帮助）菜单选择 **Topics**（主题），打开现有在线帮助文件列表。

简介

本章包含使用 32 位语言工具前需要了解的一般信息。内容包括：

- 文档编排
- 本指南使用的约定
- 推荐读物
- Microchip 网站
- myMICROCHIP 个性化通知客户服务
- 客户支持

文档编排

本文档说明了如何使用GNU语言工具来为32位单片机应用编写代码。文档编排如下：

第1部分——MPLAB XC32汇编器

- **第1章汇编器概述**——给出了汇编器操作的概述。
- **第2章汇编器命令行选项**——详细说明了汇编器的命令行选项。
- **第3章MPLAB XC32汇编语言**——讨论了宏汇编器使用的源语言。
- **第4章汇编器伪指令**——说明了源代码中的汇编器命令。
- **第5章汇编器错误/警告/消息**——提供了错误、警告和消息的描述性列表。

第2部分——MPLAB XC32目标链接器

- **第6章链接器概述**——给出了链接器操作的概述。
- **第7章链接器命令行接口**——详细说明了链接器的命令行选项。
- **第8章链接描述文件**——说明了生成和使用链接描述文件控制链接器操作的方法。
- **第9章链接器处理**——讨论了链接器从输入文件构建应用程序的方法。
- **第10章链接器示例**——包含了多个32位特定的链接器示例。
- **第11章链接器错误/警告**——提供了错误和警告的描述性列表。

第3部分——32位实用程序 （包括归档器/库管理器）

- **第12章MPLAB XC32目标归档器/库管理器**——详细说明了归档器/库管理器的命令行选项。
- **第13章其他实用程序**——详细说明了其他的实用程序及其操作。

第4部分——附录

- **附录A已弃用的功能** ——讨论了已被视为作废的功能。
- **附录B有用的表格**——列出了以下实用表格：ASCII码字符集和十六进制到十进制的转换。
- **附录C GNU自由文档许可证**——详细说明了使用GNU语言工具的许可证要求。

© 2014 Microchip Technology Inc.

本指南使用的约定

本指南采用以下文档约定：

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	…… 为*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”
带右尖括号且有下划线的斜体文字	菜单路径	*File>Save*
粗体字	对话框按钮	单击 **OK**
	选项卡	单击 **Power** 选项卡
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`* 可以是任一有效文件名
方括号 []	可选参数	`mcc18 [选项] file [选项]`
花括号和竖线：{|}	选择互斥参数；“或”选择	`errorlevel {0|1}`
省略号 ...	代替重复文字	`var_name [, var_name...]`
	表示由用户提供的代码	`void main (void) { ... }`

推荐读物

本文档介绍如何使用32位语言工具。下面列出了其他有用的文档。以下Microchip文档均已提供，并建议读者作为补充参考资料。

自述文件

有关使用Microchip工具的最新信息，请阅读软件自带的相关自述文件（HTML文件）。

MPLAB® XC32 C/C++编译器用户指南 （DS50001686G_CN）

该文档为使用32位C编译器的指南。 32位链接器与此工具配合使用。

32-Bit Language Tools Libraries （DS51685）

该文档说明了Microchip 32位器件可使用的库。包括标准（包括数学）库以及编译器内建函数。 32位外设库在随每种外设库类型所提供的HTML文件中叙述。

特定器件文档

Microchip网站包含许多描述32位器件功能和特性的文档。其中包括：

- 各器件和器件系列数据手册
- 系列参考手册
- 程序员参考手册

© 2014 Microchip Technology Inc.

MICROCHIP 网站

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

- **产品支持** —— 数据手册和勘误表、应用笔记和示例程序、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 顾问计划成员名单
- **Microchip 业务** —— 产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、 Microchip 销售办事处、代理商以及工厂代表列表

myMICROCHIP 个性化通知客户服务

Microchip 的个人通知客户服务有助于客户了解关于所关注 Microchip 产品的最新信息。注册客户可在指定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。

请访问http://www.microchip.com/pcn开始注册过程，并选择接收个性化通知的首选项。页面上提供了 FAQ 和注册详细信息，可以通过选择上面的链接打开。

在选择首选项时，选择 “Development Systems”（开发系统）将会在列表中填入可用的开发工具。下面列出了主要的工具类别：

开发系统产品的分类如下：

- **编译器**——Microchip C 编译器、汇编器、链接器及其他语言工具的最新信息。包括所有 MPLAB C 编译器、所有 MPLAB 汇编器（包括 MPASM™ 汇编器）、所有 MPLAB 链接器（包括 MPLINK™ 目标链接器），以及所有 MPLAB 库管理器（包括 MPLIB™ 目标库管理器）。
- **仿真器**——Microchip MPLAB REAL ICE™ 在线仿真器的最新信息。
- **在线调试器**——Microchip 在线调试器的最新信息，包括 MPLAB ICD 3 在线调试器和 PICkit™ 3 在线调试器/编程器。
- **MPLAB X IDE**——关于开发系统工具的集成开发环境 Microchip MPLAB X IDE 的最新信息。主要针对 MPLAB X IDE、MPLAB X IDE 项目管理器、MPLAB X 编辑器和 MPLAB X SIM 软件模拟器，以及一般编辑和调试功能。
- **编程器**——Microchip 编程器的最新信息。包括器件（生产）编程器 MPLAB REAL ICE 在线仿真器、MPLAB ICD 3 在线调试器和 MPLAB PM3，以及开发（非生产）编程器 PICkit 3 在线调试器/编程器。
- **入门/演示板**——包括 MPLAB 入门工具包电路板、PICDEM™ 演示板以及各种其他评估板。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

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

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

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

如果您发现本文档有任何错误或者对本文档有任何建议，请通过电子邮件联系我公司 TRC 经理，电子邮件地址为 CTRC@microchip.com。

© 2014 Microchip Technology Inc.

第1部分—— MPLAB XC32汇编器

第1章 汇编器概述 15
第2章 汇编器命令行选项 23
第3章 MPLAB XC32汇编语言 41
第4章 汇编器伪指令 57
第5章 汇编器错误/警告/消息 81

注：

© 2014 Microchip Technology Inc.

第1章 汇编器概述

1.1 简介

MPLAB® XC32汇编器（xc32-as）将汇编语言符号转换成可供PIC32 MCU系列器件识别的可重定位机器代码。该汇编器是一个Windows®操作系统控制台应用程序，它为开发汇编语言代码提供了平台。该汇编器是来自Free Software Foundation的GNU汇编器的一部分。

本章涵盖以下主题：

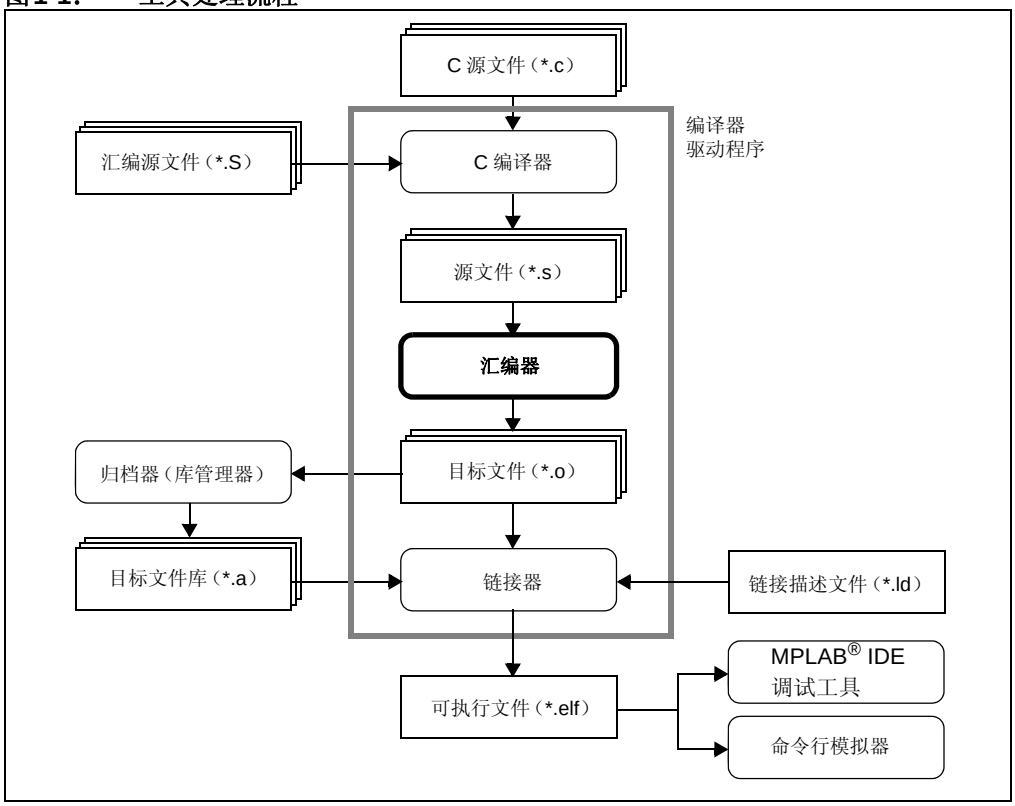
- 汇编器和其他开发工具
- 特性集
- 输入/输出文件

1.2 汇编器和其他开发工具

MPLAB XC32汇编器用于翻译用户汇编源文件。此外，MPLAB XC32 C/C++编译器用该汇编器来生成目标文件。

在 C 预处理程序处理完汇编源文件（*.S）后，该汇编器生成可重定位的目标文件，随后这些目标文件可以被归档或者由链接器将这些目标文件与其他可重定位目标文件及归档文件相链接，形成一个可执行文件。工具处理流程的概况请参见图1-1。

图1-1： 工具处理流程



1.3 特性集

此汇编器的突出特性包括：

- 支持MIPS32、MIPS16e和microMIPS指令集
- 支持ELF目标格式
- 可在Linux®、OS X®和Windows操作系统下使用
- 丰富的伪指令集
- 灵活的宏语言
- 命令行接口
- 是MPLAB X IDE的集成组件

© 2014 Microchip Technology Inc.

1.4 输入/输出文件

下表列出了标准的汇编器输入和输出文件。

扩展名	说明
输入	
.S	要预处理的汇编源文件（推荐）
.s	源文件
输出	
.o	目标文件
.1st	列表文件

与MPASM™汇编器（用于8位PIC® MCU）不同，MPLAB XC32汇编器不会生成错误文件、hex 文件或符号和调试文件。XC32 汇编器能够创建列表文件和可重定位的目标文件（可能包含也可能不包含调试信息）。MPLAB XC32 链接器与该汇编器配合使用可产生用于在 MPLAB X IDE 中进行调试的最终目标文件、映射文件和最终可执行文件（见图1-1）。

1.4.1 源文件

该汇编器可接受的输入是由 PIC32 指令、汇编器伪指令和注释组成的源文件。例 1-1 给出了源文件示例。

> **注：** Microchip 强烈建议将 .S 作为汇编源文件的扩展名。这使您可以更简单地使用 C 编译器驱动程序，而不需要指定一个选项以告知驱动程序将一个文件视为汇编文件。大写字母“S”还表示源文件应在传送到汇编器之前先由 C 预处理程序进行预处理。更多关于 C 编译器驱动程序的信息，请参见《MPLAB® XC32 C/C++编译器用户指南》（DS50001686G_CN）。

例 1-1： 汇编程序代码示例

更新后的示例代码：

```
#include <xc.h>
#define IOPORT_BIT_7 (1 << 7)
    .global main    /* define all global symbols here */
    .text
    /* define which section (for example "text")
     * does this portion of code resides in. Typically,
     * all your code will reside in .text section as
     * shown below.
     */
    .set noreorder
    /* This is important for an assembly programmer. This
     * directive tells the assembler not to optimize
     * the order of the instructions as well as not to insert
     * 'nop' instructions after jumps and branches.
     */
/*******************************************************************
     * main()
     * This is where the PIC32 start-up code will jump to after initial
     * set-up.
     *******************************************************************/
```

```
   .ent main /* directive that marks symbol 'main' as function in the ELF
              * output
              */
main:
   /* Call function to clear bit relevant to pin 7 of port A.
    * The 'jal' instruction places the return address in the $ra
    * register.
    */
   ori      a0, $0, IOPORT_BIT_7
   jal      mPORTAClearBits
   nop
   /* endless loop */
endless:
   j endless
   nop
   .end main /* directive that marks end of 'main' function and its
              * size in the ELF output
              */
/*******************************************************************
 * mPORTAClearBits(int bits)
 * This function clears the specified bits of IOPORT A.
 *
 * pre-condition: $ra contains return address
 * Input: Bit mask in $a0
 * Output: none
 * Side effect: clears bits in IOPORT A
 *******************************************************************/
   .ent mPORTAClearBits
mPORTAClearBits:
   /* function prologue - save registers used in this function
    * on stack and adjust stack-pointer
    */
   addiu   sp, sp, -4
   sw      s0, 0(sp)
   la      s0, LATACLR
   sw      a0, 0(s0)      /* clear specified bits */
   /* function epilogue - restore registers used in this function
    * from stack and adjust stack-pointer
    */
   lw      s0, 0(sp)
   addiu   sp, sp, 4
   /* return to caller */
   jr      ra
   nop
   .end mPORTAClearBits
```


© 2014 Microchip Technology Inc.

1.4.2 目标文件

汇编器创建可重定位 ELF 目标文件。这些目标文件的地址尚未被分配，因此必须先链接这些目标文件后才能生成可执行文件。

默认情况下，创建的目标文件名为a.out。在命令行中指定-o选项（见**第2章“汇编器命令行选项”**）可改写此默认名称。

1.4.3 列表文件

汇编器能够生成列表文件。这些列表文件不是绝对列表文件，列表中出现的地址均是相对于段的起始地址而言的。

默认情况下，列表文件显示在标准输出中。在命令行上指定-a=<file>选项（见**第2章“汇编器命令行选项”**）可以将列表文件发送到指定的文件file中。

汇编器生成的列表文件由下列元素组成。例1-2给出了列表文件示例。

1.4.3.1 开头

开头包含汇编器的名称、被汇编的文件名称以及页码。如果指定了-an选项，将不会显示该信息。

1.4.3.2 标题

标题行包含由.title伪指令指定的标题。如果指定了-an选项，将不会显示该信息。

1.4.3.3 副标题

副标题行包含由.sbttl伪指令指定的副标题。如果指定了-an选项，将不会显示该信息。

1.4.3.4 高级语言源代码

如果为汇编器指定了-ah选项，将显示高级语言源代码。高级语言源代码的格式为：

```
<line #>:<filename>       **** <source>
```

例如：

```
1:hello.c       **** #include <stdio.h>
```

1.4.3.5 汇编语言源代码

如果为汇编器指定了-al选项，将显示汇编语言源代码。汇编语言源代码的格式为：

```
<line #> <addr> <encoded bytes> <source>
```

例如：

```
35 0000 80000434    ori      $a0, $zero, IOPORT_BIT_7
```

注 1： 行号可以重复。

2： 地址相对于其所在模块中的段而非绝对地址。

3： 指令按照“小尾数法”（little endian）的顺序编码。

1.4.3.6 符号表

如果为汇编器指定了-as选项，将显示符号表。已定义和未定义符号的列表都将显示。

已定义符号的格式为：

```
DEFINED SYMBOLS
<filename>:<line #> <section>:<addr> <symbol>
```

例如：

```
DEFINED SYMBOLS
foo.S:79     .text:00000000 main
foo.S:107    .text:00000014 mPORTAClearBits
```

未定义符号的格式为：

```
UNDEFINED SYMBOLS
<symbol>
```

例如：

```
UNDEFINED SYMBOLS
WDTCON
WDTCONCLR
```

例1-2： 汇编器列表文件示例

```
GAS LISTING foo.s       page 1

   1              # 1 "foo.S"
   2              # 1 "<built-in>"
   1                 .nolist
   0
   0
   3                 .list
   4
   5              #define IOPORT_BIT_7 (1 << 7)
   6
   8                 .global baz /* define all global symbols here */
   9
  10                 /* define which section (for example "text")
  11                  * does this portion of code resides in.
  12                  * Typically, all of your code will reside in
                       * the .text section as shown.
  14                  */
  15                 .text
  16
  17                 /* This is important for an assembly programmer.
  18                  * This directive tells the assembler not to
  19                  * optimize the order of the instructions
  20                  * as well as not to insert 'nop' instructions
  21                  * after jumps and branches.
  22                  */
  23                 .set noreorder
  24
  25              .ent baz /* directive that marks symbol 'baz' as
  26                        * a function in ELF output
  27                        */
  28
  29              baz:
  30
  31                 /* Call function to clear bit relevant to pin
  32                  * 7 in port A. The 'jal' instruction places
```


© 2014 Microchip Technology Inc.

```
  33                  * the return address in $ra.
  34                  */
  35 0000 80000434    ori      $a0, $zero, IOPORT_BIT_7
  36 0004 0500000C    jal      mPORTAClearBits
  37 0008 00000000    nop
  38
  39                 /* endless loop */
  40              endless:
  41 000c 03000008    j      endless
  42 0010 00000000    nop
  43
  44              .end baz /* directive that marks end of 'baz'
  45                       * function and registers size in ELF
  46                       * output
  47                       */
DEFINED SYMBOLS
                           *ABS*:00000000 foo.S
                           *ABS*:00000001 __DEBUG
               foo.S:56    .text:00000014 mPORTAClearBits
               foo.S:38    .text:0000000c endless
```

注：

© 2014 Microchip Technology Inc.

第2章 汇编器命令行选项

2.1 简介

除了与MPLAB X IDE项目管理器配合使用外，MPLAB XC32汇编器（xc32-as）也可以通过主机PC的命令行接口（例如cmd.exe）使用。

编译项目时，MPLAB X IDE项目管理器通过xc32-gcc编译驱动程序自动调用汇编器。本文列出的许多常用选项在MPLAB X IDE Project Properties（项目属性）对话框中以复选框形式提供。但是，对于较高级的选项，可能需要在该对话框的“Additional Options”（其他选项）字段中指定。在MPLAB X IDE中编译了项目后，Output窗口会显示传递给汇编器的选项。

本章涵盖以下主题：

- 汇编器接口语法
- 编译驱动程序接口语法
- 修改列表输出的选项
- 控制信息输出的选项
- 控制输出文件创建的选项
- 汇编器符号定义和搜索路径选项
- 编译驱动程序和预处理程序选项

2.2 汇编器接口语法

汇编器命令行可以包含选项和文件名。选项可以以任意顺序出现，可以位于文件名之前、之后或是两个文件名之间。文件名的顺序决定它们将要被汇编的顺序。

```
xc32-as [options|sourcefiles]...
```

只有“`--`”（两个连字号）时将明确指定标准输入文件作为汇编器要汇编的文件之一。除了“`--`”，任何以连字号（“`-`”）开始的命令行参数都是选项。每个选项均可改变汇编器的工作方式，但是任何选项均不能改变其他选项的工作方式。

某些选项严格要求在其后跟有一个文件名。文件名可以紧接在选项字母的后面，也可以作为该命令行的下一个参数。例如，要指定一个名为`test.o`的输出文件，以下两种形式均可接受：

- `-o test.o`
- `-otest.o`

> **注：** 命令行选项区分大小写。

2.3 编译驱动程序接口语法

编译驱动程序（xc32-gcc）预处理、编译、汇编和链接C语言和汇编语言模块及库归档文件。此驱动程序协调整个编译过程，因此您无需了解哪些个别程序进行预处理、编译、汇编和链接。此驱动程序调用各相应的工具来完成请求的编译过程。

实际上，汇编器通常通过xc32-gcc来调用，确定它应汇编文件扩展名为`*.S`或`*.s`的输入文件。编译驱动程序将带有 `*.S`（大写字母）扩展名的文件传递到汇编器之前通过CPP样式的预处理程序进行发送，但是驱动程序却将带有`*.s`（小写字母）扩展名的文件直接发送到汇编器。

编译驱动程序命令行的基本格式为：

```
xc32-gcc [options] files
```

> **注：** 命令行选项和文件扩展名区分大小写。

要将汇编器选项从编译驱动程序传递到汇编器，请使用 `-Wa,option` 选项。此选项参数不应包含空格。

例2-1： 编译驱动程序命令行示例

```
xc32-gcc -mprocessor=32MX360F512L -I"./include" ASMfile.S
  -o"ASMfile.o" -DMYMACRO=1 -Wa,-ah="ASMfile.lst"
```

更多关于编译驱动程序的信息，请参见《MPLAB® XC32 C/C++编译器用户指南》（DS50001686G_CN）。

> **注：** 要在MPLAB X IDE中使用xc32-gcc编译驱动程序，请确保为项目选择了XC32编译器工具链。

© 2014 Microchip Technology Inc.

2.4 修改列表输出的选项

以下选项用于控制列表输出。列表文件有助于调试代码和对代码操作进行常规分析。可使用下列选项构造一个包含有用信息的列表文件。

- `-a[suboption] [=file]`
- `--listing-lhs-width num`
- `--listing-lhs-width2 num`
- `--listing-rhs-width num`
- `--listing-cont-lines num`

2.4.1 -a[*suboption*] [=*file*]

`-a`选项使能列表输出。`-a`选项支持以下子选项以进一步控制汇编列表所包含的内容：

`-ac`	省略false条件
`-ad`	省略调试伪指令
`-ah`	包含高级语言源代码
`-al`	包含汇编代码
`-am`	包含宏扩展
`-an`	省略格式处理
`-as`	包含符号
`-a=file`	将列表输出到指定文件（该文件必须在当前目录中）

如果没有指定子选项，默认使用的子选项为`hls`；只有`-a`选项时要求列出高级语言代码、汇编代码和符号。可以使用其他字母选择特定的选项来控制列表输出。

跟在`-a`后的字母可以被组合为一个选项。因此，例如您可以在命令行中指定`-aln`来代替`-al -an`。

2.4.1.1 -ac

-ac 在输出列表时省略 false 条件。任何由于为假的 .if 或者 .ifdef（以及 .if 或者 .ifdef 为真时的 .else）而不汇编的代码行都将从列表文件中省略。例 2-2 给出了没有使用 -ac 选项时生成的列表文件；例 2-3 给出了相同源代码在使用了 -ac 选项时生成的列表文件。

例 2-2： 使用 -al 命令行选项生成的列表文件

```
GAS LISTING asm.s                                page 1

   1                # 1 "asm.S"
   2                # 1 "<built-in>"
   1                       .data
   0
   2                       .if 0
   3                         .if 1
   4                         .endif
   5                         .long 0
   6                         .if 0
   7                           .long 0
   8                         .endif
   9                       .else
  10                         .if 1
  11                         .endif
  12 0000 02000000            .long 2
  13                         .if 0
  14                           .long 3
  15                         .else
  16 0004 04000000              .long 4
  17                         .endif
  18                       .endif
  19                       .if 0
  20                         .long 5
  21                       .elseif 1
  22                         .if 0
  23                           .long 6
  24                         .elseif 1
  25 0008 07000000              .long 7
  26                         .endif
  27                       .elseif 1
  28                         .long 8
  29                       .else
  30                         .long 9
  31                       .endif
```


© 2014 Microchip Technology Inc.

例2-3：　使用-alc命令行选项生成的列表文件

```
GAS LISTING asm.s page 1

  1              # 1 "asm.S"
  2              # 1 "<built-in>"
  1                    .data
  0
  0
  2                    .if 0
  9                    .else
 10                      .if 1
 11                      .endif
 12 0000 02000000         .long 2
 13                      .if 0
 15                      .else
 16 0004 04000000           .long 4
 17                      .endif
 18                   .endif
 19                   .if 0
 21                   .elseif 1
 22                      .if 0
 24                      .elseif 1
 25 0008 07000000           .long 7
 26                      .endif
 27                   .elseif 1
 29                   .else
 31                   .endif
```

注：　由于使用-ac选项而省略了一些行。

2.4.1.2 -ad

-ad在输出列表时省略调试伪指令。在处理编译器生成的包含调试信息的汇编代码时，此选项很有用，编译器生成的调试伪指令不会使列表文件变得杂乱。例2-4给出了使用d和h子选项时生成的列表文件。与仅使用h子选项（见下一节）相比，该列表文件看上去整洁多了。

例2-4： 使用-alhd命令行选项生成的列表文件

```
GAS LISTING test.s page 1

   1              .section .mdebug.abi32
   2              .previous
  10              .Ltext0:
  11              .align2
  12              .globlmain
  13              .LFB0:
  14              .file 1 "src\\test.c"
   1:src/test.c    **** #include <xc.h>
   2:src/test.c    **** volatile unsigned int testval;
   3:src/test.c    ****
   4:src/test.c    **** int
   5:src/test.c    **** main (void)
   6:src/test.c    **** {
  15              .loc 1 6 0
  16              .setnomips16
  17              .entmain
  18             main:
  19         .frame$fp,8,$31# vars= 0, regs= 1/0, args= 0, gp= 0
  20         .mask0x40000000,-8
  21         .fmask0x00000000,0
  22              .setnoreorder
  23              .setnomacro
  24
  25 0000 F8FFBD27 addiu$sp,$sp,-8
  26              .LCFI0:
  27 0004 0000BEAF sw$fp,0($sp)
  28              .LCFI1:
  29 0008 21F0A003 move$fp,$sp
  30              .LCFI2:
   7:src/test.c    ****   testval += 1;
  31              .loc 1 7 0
  32 000c 0000828F lw$2,%gp_rel(testval)($28)
  33 0010 01004224 addiu$2,$2,1
  34 0014 000082AF sw$2,%gp_rel(testval)($28)
   8:src/test.c    ****   return 0;
  35              .loc 1 8 0
  36 0018 21100000 move$2,$0
   9:src/test.c    **** }
  37              .loc 1 9 0
  38 001c 21E8C003 move$sp,$fp
  39 0020 0000BE8F lw$fp,0($sp)
  40 0024 0800BD27 addiu$sp,$sp,8
  41 0028 0800E003 j$31
  42 002c 00000000 nop
```


© 2014 Microchip Technology Inc.

```
43
44              .setmacro
45              .setreorder
46              .endmain
47              .LFE0:
49
50              .commtestval,4,4
88              .Letext0:
```

2.4.1.3 -ah

-ah要求在列表文件中显示高级语言源代码。高级语言代码列表要求编译器已经生成了汇编语言源代码、已向编译器给定了类似于 -g 的调试选项，并且还必须同时使用汇编列表选项（-al）。-al请求输出程序的汇编代码列表。例2-5给出了使用-alh命令行选项生成的列表文件。

例2-5： 使用-alh命令行选项生成的列表文件

```
GAS LISTING tempfile.s page 1

   1              .section .mdebug.abi32
   2              .previous
   3              .section.debug_abbrev,"",@progbits
   4              .Ldebug_abbrev0:
   5              .section.debug_info,"",@progbits
   6              .Ldebug_info0:
   7              .section.debug_line,"",@progbits
   8              .Ldebug_line0:
   9 0000 34000000 .text
  11              .align2
  12              .globlmain
  13              .LFB0:
  14              .file 1 "src/test.c"
   1:src/test.c    **** #include <xc.h>
   2:src/test.c    **** volatile unsigned int testval;
   3:src/test.c    ****
   4:src/test.c    **** int
   5:src/test.c    **** main (void)
   6:src/test.c    **** {
  15              .loc 1 6 0
  16              .setnomips16
  17              .entmain
  18             main:
  19        .frame$sp,0,$31 # vars= 0, regs= 0/0, args= 0, gp= 0
  20              .mask0x00000000,0
  21              .fmask0x00000000,0
  22              .setnoreorder
  23              .setnomacro
  24
   7:src/test.c    ****   testval += 1;
  25              .loc 1 7 0
  26 0000 0000848F lw$4,%gp_rel(testval)($28)
   8:src/test.c    ****   return 0;
   9:src/test.c    **** }
  27              .loc 1 9 0
  28 0004 21100000 move$2,$0
  29              .loc 1 7 0
  30 0008 01008324 addiu$3,$4,1
  31 000c 000083AF sw$3,%gp_rel(testval)($28)
  32              .loc 1 9 0
  33 0010 0800E003 j$31
  34 0014 00000000 nop
```


© 2014 Microchip Technology Inc.

```
35
36              .setmacro
37              .setreorder
38              .endmain
39              .LFE0:
40              .sizemain, .-main
41
42              .commtestval,4,4
```

2.4.1.4 -al

-al要求输出汇编列表。该子选项可以和其他子选项一起使用。请参见本节中其他示例。

2.4.1.5 -am

-am将列表文件中的宏展开。例2-6给出了在没有使用-am选项时生成的列表文件；例2-7给出了相同源代码在使用-am选项时生成的列表文件。

例2-6： 使用-al命令行选项生成的列表文件

```
GAS LISTING foo.s                        page 1


   1              # 1 "foo.S"
   2              # 1 "<built-in>"
   1                      .macro  sum from=0, to=5
   0
   0
   2                      .long   \from
   3                      .if     \to-\from
   4                      sum     "(\from+1)",\to
   5                      .endif
   6                      .endm
   7
   8                      .data
   9 0000 00000000         .long 0
  10 0004 0A000000         sum 10, 14
  10      0B000000
  10      0C000000
  10      0D000000
  10      0E000000
  11 0018 00000000         .long 0
```


© 2014 Microchip Technology Inc.

例2-7：　使用-alm命令行选项生成的列表文件

```
GAS LISTING foo.s                          page 1

   1              # 1 "foo.S"
   2              # 1 "<built-in>"
   1                      .macro  sum from=0, to=5
   0
   0
   2                      .long   \from
   3                      .if     \to-\from
   4                      sum     "(\from+1)",\to
   5                      .endif
   6                      .endm
   7
   8                      .data
   9 0000 00000000         .long 0
  10                      sum 10, 14
  10 0004 0A000000 > .long 10
  10              > .if 14-10
  10              > sum "(10+1)",14
  10 0008 0B000000 >> .long (10+1)
  10              >> .if 14-(10+1)
  10              >> sum "((10+1)+1)",14
  10 000c 0C000000 >>> .long ((10+1)+1)
  10              >>> .if 14-((10+1)+1)
  10              >>> sum "(((10+1)+1)+1)",14
  10 0010 0D000000 >>>> .long (((10+1)+1)+1)
  10              >>>> .if 14-(((10+1)+1)+1)
  10              >>>> sum "((((10+1)+1)+1)+1)",14
  10 0014 0E000000 >>>>> .long ((((10+1)+1)+1)+1)
  10              >>>>> .if 14-((((10+1)+1)+1)+1)
  10              >>>>> sum "(((((10+1)+1)+1)+1)+1)",14
  10              >>>>> .endif
  10              >>>> .endif
  10              >>> .endif
  10              >> .endif
  10              > .endif
  11 0018 00000000         .long 0
```

注：　>表示扩展的宏指令。

2.4.1.6 -an

-an关闭所有由列表伪指令.psize、.eject、.title和.sbttl执行的格式处理。例2-8给出了没有使用-an选项时生成的列表文件；例2-9给出了相同源代码在使用了-an选项时生成的列表文件。

例2-8： 使用-al命令行选项生成的列表文件

```
GAS LISTING foo.s                       page 1
User's Guide Example
Listing Options
   1              # 1 "foo.S"
   2              # 1 "<built-in>"
   1                 .text
   0
   0
   2                 .title "User's Guide Example"
   3                 .sbttl "Listing Options"
GAS LISTING foo.s                       page 2
User's Guide Example
Listing Options
   4                 .psize 10
   5
   6 0000 01001A3C    lui $k0, 1
   7 0004 02001A3C    lui $k0, 2
   8 0008 03001A3C    lui $k0, 3
   9                 .eject
GAS LISTING foo.s                       page 3
User's Guide Example
Listing Options
  10 000c 04001A3C    lui $k0, 4
  11 0010 05001A3C    lui $k0, 5
```

例2-9： 使用-aln命令行选项生成的列表文件

```
   1              # 1 "foo.S"
   2              # 1 "<built-in>"
   1                 .text
   0
   0
   2                 .title "User's Guide Example"
   3                 .sbttl "Listing Options"
   4                 .psize 10
   5
   6 0000 01001A3C    lui $k0, 1
   7 0004 02001A3C    lui $k0, 2
   8 0008 03001A3C    lui $k0, 3
   9                 .eject
  10 000c 04001A3C    lui $k0, 4
  11 0010 05001A3C    lui $k0, 5
```


© 2014 Microchip Technology Inc.

2.4.1.7 -as

-as要求列出符号表。例2-10给出了使用-as命令行选项生成的列表文件。注意已定义和未定义的符号都将被列出。

例2-10： 使用-as命令行选项生成的列表文件

```
GAS LISTING example.s page 1

DEFINED SYMBOLS
                            *ABS*:00000000 src\example.c
           example.s:18     .text:00000000 main
                            *COM*:00000004 testval

UNDEFINED SYMBOLS
bar
```

2.4.1.8 -a=*file*

=file定义了输出文件的名称。该文件必须位于当前目录中。

2.4.2 --listing-lhs-width *num*

--listing-lhs-width 选项用于设置列表文件输出数据列的宽度。默认情况下，它被设置为1个字。以下是从列表文件中抽取的行。输出数据列以粗体文本显示。

```
2 0000 54686973  .ascii "This is an example"
2      20697320
2      616E2065
2      78616D70
2      6C650000
```

如果使用--listing-lhs-width 2选项，则相同的行在列表文件中将显示如下：

```
2 0000 54686973 20697320  .ascii "This is an example"
2      616E2065 78616D70
2      6C650000
```

2.4.3 --listing-lhs-width2 *num*

--listing-lhs-width2 选项用于设置列表文件输出数据列的连续行宽度。默认情况下，它被设置为 1。如果指定的宽度小于第一行，那么该选项将被忽略。以下是从列表文件中抽取的行。输出数据列以粗体显示。

```
2 0000 54686973  .ascii "This is an example"
2      20697320
2      616E2065
2      78616D70
2      6C650000
```

如果使用--listing-lhs-width2 3选项，则相同的行在列表文件中将显示如下：

```
2 0000 54686973  .ascii "This is an example"
2      20697320 616E2065 78616D70
2      6C650000
```

2.4.4 --listing-rhs-width *num*

--listing-rhs-width 选项用于设置源代码文件的行以字符数表示的最大宽度。默认情况下，它被设置为 100。以下是从没有使用 --listing-rhs-width 选项生成的列表文件中抽取的行。以粗体显示的文本是源代码文件中的行。

```
2 0000 54686973  .ascii "This line is long"
2      206C696E
2      65206973
2      206C6F6E
2      67000000
```

如果使用--listing-rhs-width 22选项，则相同的行在列表文件中将显示如下：

```
2 0000 54686973  .ascii "This line is
2      206C696E
2      65206973
2      206C6F6E
2      67000000
```

在列表文件中行会被截短（而非换行），但是数据仍然存在。

2.4.5 --listing-cont-lines *num*

--listing-cont-lines 选项用于设置列表文件输出数据列的最大连续行数。默认情况下，它被设置为8。以下是从没有使用--listing-cont-lines选项生成的列表文件中抽取的行。粗体文本显示了列表文件输出数据列中的连续行。

```
2 0000 54686973  .ascii "This is a long character
                         sequence"
2      20697320
2      61206C6F
2      6E672063
2      68617261
```

注意显示的字节数与ASCII字符串中的字节数相符，但是如果使用了 --listing-cont-lines 2选项，输出数据将会在显示了两个连续行之后被截断，如下所示。

```
2 0000 54686973  .ascii "This is a long character
                         sequence"
2      20697320
2      61206C6F
```


© 2014 Microchip Technology Inc.

2.5 控制信息输出的选项

本节中的选项控制信息的输出方式。通过本节中的几个选项可以对与代码转换和执行有关的错误、警告和消息进行控制。

圆括号中的项给出了指定选项的快捷方式（如`--no-warn`也可以被指定为`-W`）。

2.5.1 `--fatal-warnings`

将警告视作错误处理。

2.5.2 `--no-warn (-W)`

不产生警告。如果使用该选项，将不会发出警告。此选项仅影响警告消息， 它并不会更改文件的汇编。仍然会报告错误。

2.5.3 `--warn`

在适当的时候会发出警告。这是默认的行为。

2.5.4 `-J`

不会发出有关有符号溢出的警告。

2.5.5 `--help`

汇编器将显示有关命令行用法和选项的消息。然后汇编器退出。

2.5.6 `--target-help`

汇编器将显示有关特定于PIC32目标器件的命令行选项的消息。然后汇编器退出。

2.5.7 `--version`

显示汇编器版本号。然后汇编器退出。

2.5.8 `--verbose (-v)`

显示汇编器版本号。汇编器不会退出。如果只使用了该命令行选项，那么汇编器将打印出其版本号并等待通过标准输入进入汇编源代码。使用 `<CTRL>-D` 可发送一个 EOF 字符以结束汇编。

2.6 控制输出文件创建的选项

本节中的选项控制输出文件的创建方式。例如，使用 -o 选项可更改输出目标文件的名称。

圆括号中的项给出了指定选项的快捷方式（如--keep-locals也可以被指定为-L）。

2.6.1 -g

生成符号调试信息。

2.6.2 --keep-locals (-L)

保留局部符号，即以 .L（只能是大写字母）开始的标号。您在调试的过程中通常无法看见这样的标号，因为它们专供用于生成汇编程序的程序（例如编译器）使用。通常，汇编器和链接器都会丢弃此类符号。该选项告知汇编器在目标文件中保留这些符号。

2.6.3 -o *objfile*

将输出目标文件命名为*objfile*。如果没有错误，当您在运行汇编器时总会输出一个目标文件。它的默认名为 a.out。使用该选项（仅带有一个文件名）可为目标文件指定不同的名称。无论目标文件叫什么名字，汇编器都将覆盖任何与其同名的现有文件。

2.6.4 -Z

在错误产生后仍然生成目标文件。通常，当错误消息出现后，汇编器将不再产生输出。如果由于某种原因，在汇编器给出错误消息之后，您仍然对目标文件输出感兴趣，那么就应该使用 -Z 选项。如果存在任何错误，汇编器仍将继续运行，在显示了“n errors, m warnings, generating bad object file”（n个错误，m条警告，生成了错误的目标文件）形式的最终警告消息之后生成一个目标文件。

2.6.5 -MD *file*

将相关性信息写入 *file*。汇编器可生成相关性文件。该文件由适用于描述主源代码文件相关性的一个规则组成。该规则被写入在该选项的参数中指定的文件中。该功能可被用于Makefile的自动更新。

© 2014 Microchip Technology Inc.

2.7 汇编器符号定义和搜索路径选项

本节中的选项将执行上述各节中没有定义的功能。

2.7.1 --defsym *sym=value*

定义符号*sym*并给其赋值*value*。

2.7.2 -I *dir*

使用该选项在目录列表中添加一条名为*dir*的目录，供汇编器在其中查找在.include伪指令中指定的文件。有必要的话，您可多次使用 -I，以使目录列表包含多条路径。总是首先搜索当前的工作目录，然后汇编器将按在命令行中指定的顺序（从左到右）搜索所有-I目录。

直接传递到汇编器时，此选项影响汇编器的.include伪指令使用的搜索路径。要影响C预处理程序的#include伪指令使用的搜索路径，请将相应的选项传递到xc32-gcc编译驱动程序。

2.8 编译驱动程序和预处理程序选项

本节中的编译驱动程序（xc32-gcc）和 C 预处理程序选项可用于汇编代码项目。编译驱动程序会根据需要将选项传递到预处理程序。更多关于编译驱动程序的信息以及更加全面的驱动程序选项列表，请参见《MPLAB® XC32 C/C++编译器用户指南》（DS50001686G_CN）。

2.8.1 -mprocessor=*device*

选择编译所针对的器件（例如-mprocessor=32MX360F512L）。

2.8.2 -Wa,*option*

将*option*作为选项传递给汇编器。如果*option*包含逗号，则表示以逗号分隔多个汇编器选项。*option*参数不应包含空格。

2.8.3 -Dmacro=*defn*

将宏macro定义为*defn*。在任何-U选项之前处理命令行中的所有-D实例。

2.8.4 -Dmacro

将宏macro定义为1。在任何-U选项之前处理命令行中的所有-D实例。

2.8.5 -Umacro

未定义宏macro。-U选项在所有-D选项之后、所有-include和-imacros选项之前处理。

2.8.6 -*I dir*

将目录*dir*添加到要搜索的目录列表的开头，以供预处理程序#include伪指令在其中搜索头文件。这用于覆盖系统头文件，替换为自己的头文件，因为这些目录在系统头文件目录之前进行搜索。如果使用了多个 -I 选项，将按照从左到右的顺序扫描这些目录。然后扫描标准系统目录。

传递到编译驱动程序时，此选项影响预处理程序的#include伪指令使用的搜索路径。要影响汇编器的.include伪指令使用的搜索路径，请使用-Wa选项将相应的选项传递到汇编器。

2.8.7 -save-temps

不要删除中间文件。将它们放在当前目录中并根据源文件为其命名。因此，使用 -c -save-temps编译foo.c时会产生以下文件：

- foo.i（预处理文件）
- foo.s（汇编语言文件）
- foo.o（目标文件）

2.8.8 -v

打印在编译的每个阶段执行的命令。

2.8.9 --help

打印命令行选项的说明。

© 2014 Microchip Technology Inc.

第3章 MPLAB XC32汇编语言

3.1 简介

下面将介绍宏汇编器所接受的源语言。所有操作码助记符和操作数语法均特定于目标器件。使用相同的汇编器应用程序处理编译器生成的中间汇编代码和手写汇编源代码。

本章涵盖以下主题：

- 内部预处理程序
- 源代码格式
- 特殊字符
- 符号
- 为符号赋予其他值
- 特殊点符号
- 表达式
- 运算符
- 特殊操作符

3.2 内部预处理程序

该汇编器带有一个内部预处理程序。该内部预处理程序执行以下操作。

1. 调整和删除多余的空白。它会在每行上的关键字前面留有一个空格或制表符，并将这一行上的任何其他空白替换为一个空格。
2. 删除所有注释，用一个空格或适当数量的新行代替。
3. 将字符常量转换为相应的数字值。

   如果在源代码中有一个字符（如'b'），它就会变为相应的数字值。如果这个字符发生了语法错误，汇编器将不会显示'b'，而是显示此字符相应的十进制数的首位数字。

   例如，如果源代码中有`.global mybuf`, 'b'，就会有错误消息"Error: Rest of line ignored. First ignored character is ?"（错误：行的其余部分被忽略。第一个被忽略的字符是9）。注意错误消息显示的是9。这是因为b被转换成了相应的十进制数98。汇编器实际上是在解析`.global mybuf,98`。

该内部预处理程序**不会**执行以下操作。

1. 宏预处理
2. 包含文件处理
3. C编译器预处理程序所执行的任何其他操作

可以使用`.include`伪指令来进行包含文件预处理。请参见**第4章"汇编器伪指令"**。

可以通过给输入文件加上一个`.S`后缀来使用C编译器驱动程序进行其他C样式的预处理。更多信息，请参见《MPLAB® XC32 C/C++编译器用户指南》（DS50001686G_CN）。

如果输入文件的第一行是`#NO_APP`或者您使用了`-f`选项，空白和注释将不会从输入文件中删除。在输入文件中，可以通过在可能包含空白或注释的文本前加上一个写有`#APP`的行，并在该文本之后加上一个写有`#NO_APP`的行来请求删除某些部分的空白和注释。此功能主要是为了在编译器中支持汇编语句，编译器的输出不含有注释和空白。

> **注：** 不能在未预处理的输入文本的某些部分中使用过多的空白、注释和字符常量。

© 2014 Microchip Technology Inc.

3.3 源代码格式

汇编源代码由语句和空白组成。

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

每条*语句*的一般格式如下，后面均换行。

```
[label:][mnemonic[operands] ][; comment]
```

或

```
[label:][directive[arguments] ][; comment]
```

- 标号
- 助记符
- 伪指令
- 操作数
- 参数
- 注释

3.3.1 标号

标号是从所有字母、数字、下划线字符（_）和句点（.）组成的集合中挑选出来的一个或多个字符。除非是局部符号这一特殊情况，否则标号不能以十进制数开始（更多信息，请参见**第 3.5.1 节“局部符号”**）。标号是区分大小写的；它没有长度限制，并且所有字符都有意义。

标号定义之后必须紧跟一个冒号。冒号后面可跟有空格、制表符、行结束符、汇编助记符或伪指令。

标号定义本身可以单独放在一行，将引用下一个地址。

链接完成后，标号的值为存储器中单元的绝对地址。

3.3.2 助记符

助记符告诉汇编器对哪些机器指令进行汇编。例如，加（ADD）、跳转（J）或装载（LUI）。与您自己创建的标号不同，助记符由 PIC32 MCU 汇编语言提供。助记符不区分大小写。

更多关于器件可用的 CPU 指令集助记符的详细信息，请参见您所使用目标 PIC32 MCU的数据手册。

汇编器还支持许多用于简化汇编代码编写的合成/宏指令。LI（立即数装载）指令就是合成宏指令的一个示例。汇编器产生两条机器指令以通过这一条合成指令将 32 位常量值装入寄存器。

> **注：** 不能在未预处理的输入文本的某些部分中使用过多的空白、注释和字符常量。

```
[label:] [mnemonic [operands] ] [; comment]
```

或

```
[label:] [directive [arguments] ] [; comment]
```

3.3.3 汇编器语法

汇编器为以下操作合成指令：

- 32位立即数装载
- 从存储单元装载
- 扩展的条件跳转
- 某些三操作数指令的双操作数形式
- 未对齐的装载/存储指令

汇编伪指令（例如`.set noat`、`.set nomacro`和`.set noreorder`）在您需要对生成的代码进行完全控制时禁止这些正常情况下有用的功能。请参见**第 4.13 节“控制代码生成的伪指令”**。

3.3.4 伪指令

汇编器伪指令是出现在源代码中但不直接翻译为机器代码的命令。伪指令用于控制汇编器的输入、输出和数据分配。伪指令的第一个字符是点（`.`）。更多关于可用伪指令的详细信息，请参见**第4章“汇编器伪指令”**。

3.3.5 操作数

每条机器指令可使用 0 到 4 个操作数 （完整的机器指令列表，请参见您所使用目标 PIC32 MCU 的相应数据手册）。这些操作数为指令提供相关信息，包括指令应该使用的数据和存储单元。操作数和助记符之间必须用一个或多个空格或制表符隔开。

应该用逗号隔开多个操作数。如果没有用逗号隔开操作数，汇编器就会显示警告并且按照最佳猜测分隔操作数。对于大多数 PIC32 MCU 指令，操作数由内核通用寄存器、标号、立即数或基址寄存器+偏移量组成。

3.3.5.1 通用寄存器操作数

PIC32 MCU内核包含用于整型操作和地址计算的32个32位通用寄存器。大多数PIC32 MCU指令需要一个或多个用于源和/或目标寄存器的GPR操作数。

寄存器操作数的标志是以美元符号开头（“`$`”）。寄存器编号紧跟在美元符号后。例3-1给出了使用寄存器编号操作数的汇编源代码。

但是，如果在汇编之前使用编译驱动程序（xc32-gcc）通过 CPP 样式的预处理程序对源代码进行预处理，那么可以利用C编译器随附的`xc.h`头文件中的宏。这些宏可将传统寄存器名称映射到相应的寄存器编号。例3-2给出了使用传统寄存器名称作为操作数的汇编源代码。更多有关PIC32 MCU寄存器约定和编译器运行时环境的信息，请参见《MPLAB® XC32 C/C++编译器用户指南》（DS50001686G_CN）。

例3-1： 使用寄存器编号操作数的汇编源代码

```
.text
# Add Word
li   $2, 123
li   $3, 456
add  $4, $2, $3
```


© 2014 Microchip Technology Inc.

例3-2： 使用传统寄存器名称的汇编源代码

```
#include <xc.h>
  .text
  /* Add Word */
  li   v0, 123      /* v0 is a return-value register */
  li   v1, 456      /* v1 is a return-value register */
  add  a0, v0, v1   /* a0 is an argument register */
```

3.3.5.2 立即数值操作数

立即数值可以是十六进制、八进制、二进制或十进制格式。十六进制数的标志是以 0x 开头。八进制数的标志是以0开头。二进制数的标志是以0B或0b开头。十进制数不需要特殊的开头或结尾字符。

示例：

0xe、016、0b1110和14都代表立即数值14。

-5代表立即数值-5。

symbol代表symbol的值。

3.3.5.3 基址寄存器+偏移量操作数

装载和存储操作使用基址寄存器+偏移量操作数选择存储单元。对于此类型的操作数，有效地址由32位有符号偏移量与基址寄存器的内容相加而成。PIC32 MCU数据手册将此类型的操作数显示为Mem[R+offset]。

例3-3： 使用带基址寄存器+偏移量操作数的汇编源代码

```
#include <xc.h>
        .data
        .align 4
MY_WORD_DATA:
        .word 0x10203040, 0x8090a0b0
        .text
        .global example
        /* Store Word */
example:
        la   v0, MY_WORD_DATA
        lui         v1,0x1111
        ori         v1,v1,0x4432
        lui         a0,0x5555
        ori         a0,a0,0x1123
        sw v1, 0(v0)     /* Mem[GPR[v0]+0] <- GPR[v1] */
        sw a0, 4(v0)     /* Mem[GPR[v0]+4] <- GPR[a0] */
        lw a1, 0(v0)     /* GPR[a1] <- Mem[GPR[v0]+0] */
        b .
```

C编译器支持全局指针相对（gp-rel）寻址。可在使用gp寄存器作为基址寄存器的一条指令中，执行对位于该寄存器中存储的地址任一侧32 KB范围（共64 KB）内的数据的装载和存储操作。

C编译器的-Gnum选项控制可用一条指令而非两条指令寻址的全局数据项和静态数据项的最大大小。编译器的默认gnum值为8字节，足以容纳所有简单的标量变量。

> **注：** 要利用gp相对寻址，编译器和汇编器必须将所有的“小”变量和常量分组到“小”段之一。更多关于全局指针和 -G 选项的信息，请参见《MPLAB® XC32 C/C++编译器用户指南》（DS50001686G_CN）。

例3-4： 使用GP相对寻址的汇编源代码

```
    .align 2
    .globl foo
    .set   nomips32
    .ent   foo
foo:
    .set   noreorder
    .set   nomacro

    lw     $3,%gp_rel(testval)($28)
    addiu  $2,$3,1
    lw     $2,%gp_rel(testval)($28)
    j      $31
    nop

    .set   macro
    .set   reorder
    .end   foo
```

使用gp相对寻址有几个潜在的隐患：

- 在编写汇编代码以正确声明全局（即公共或外部）数据项时必须特别小心：
  - gnum字节或更少字节的可写已初始化数据必须明确放入到.sdata段，例如：
    ```
                .sdata
     small:     .word  0x12345678
    ```
  - 全局常用数据必须使用正确的大小进行声明，例如：
    ```
                .comm small, 4
                .comm big, 100
    ```
  - 小的外部变量也必须正确声明，例如：
    ```
                .extern smallext, 4
    ```
- 如果程序具有大量的小数据项或常量，那么C编译器的-G8选项可能仍会尝试将64 KB以上的数据放入“小”区域；链接时将看到重定位错误（“重定位截短”）的现象。通过使用编译器的-G0选项禁止gp相对寻址和/或减少汇编代码的小数据段（即.sbss和.sdata）中保留的空间可解决这一问题。

3.3.6 参数

每条伪指令可使用 0 到 3 个参数。这些参数为伪指令提供其他有关如何执行命令的信息。参数和伪指令之间必须用一个或多个空格或制表符隔开。必须用逗号隔开多个参数。更多关于可用伪指令的详细信息，请参见**第4章“汇编器伪指令”**。

© 2014 Microchip Technology Inc.

3.3.7 注释

在汇编器中，注释可用下述两种方法之一来表示。

3.3.7.1 单行注释

这种类型的注释从注释字符开始一直到行末。单行注释使用数字/井号（#）开始。

> **注：** 这种注释字符与由MPASM汇编器及MPLAB XC16汇编器识别的字符不同。

3.3.7.2 多行注释

这种类型的注释可跨越多行。多行注释使用`/* ... */`。

这种注释不能嵌套。

示例：

```
/* All
of these
lines
are
comments */
```

3.4 特殊字符

常量是被写入的值，因此其值可以被直观地获知而无需了解上下文。例如：

```
.byte 74, 0112, 0b01001010, 0x4A, 0x4a, 'J', '\J'#All the same value
.ascii "Ring the bell\7" #A string constant
.float 0f-314159265358979323846264338327950288419 71.693993751E-40
```

3.4.1 数字常量

汇编器根据数字在机器中存储的方式将数字区分为两种。整数是符合C语言中long型的数字。浮点数是IEEE 754浮点数。

3.4.1.1 整数

二进制整数以0b或0B开头，其后跟有一个或多个二进制数字01。

八进制整数以0开头，其后跟有一个或多个八进制数字01234567。

十进制整数以非零的数字开头，其后跟有0或多个十进制数字0123456789。

十六进制整数以0x或0X开头，其后跟有一个或多个十六进制数字0123456789abcdefABCDEF。

使用前缀运算符“-”来表示负整数。

3.4.1.2 浮点数

浮点数以IEEE 754格式表示。通过写操作按顺序写浮点数：

- 可选前缀，由数字0开始，后面跟有字母e、f或d（大小写均可）。由于浮点常量仅与.float和.double伪指令一起使用，因此二进制数表示形式的精度与前缀无关。
- 可选符号：“+”或“-”。
- 可选整数部分：0或多个十进制数字。
- 可选小数部分：“.”后面跟有0或多个十进制数字。
- 可选指数，由以下部分组成：
  - “E”或“e”。
  - 可选符号：“+”或“-”。
  - 一个或多个十进制数字。

整数部分或小数部分中至少必须有一个存在。浮点数通常是以10为基数的值。

浮点数的计算与运行汇编器的计算机中的任何浮点硬件无关。

© 2014 Microchip Technology Inc.

3.4.2 字符常量

有两种类型的字符常量。*字符*用一个字节表示一个字符，并且其值可以在数字表达式中使用。*字符串*可能包含多个字节，并且它们的值不能在算术表达式中使用。

3.4.2.1 字符

单个字符可以被写成一个单引号后面紧跟着该字符的形式，或是使用一对单引号引用该字符的形式。汇编器接受以下转义字符来代表特殊控制字符：

表3-1： 特殊字符和用法

转义字符	说明	十六进制值
`\a`	报警字符	07
`\b`	后退字符	08
`\f`	换页符	0C
`\n`	换行符	0A
`\r`	回车符	0D
`\t`	水平制表符	09
`\v`	垂直制表符	0B
`\\`	反斜杠	5C
`\?`	问号字符	3F
`\"`	双引号	22
`\digit digit digit`	八进制字符代码。数字代码为3个八进制数字。	
`\x hex-digits`	十六进制字符代码。所有在该字符后的十六进制数字是被组合到一起的。大写或小写的x均有效。	

数字表达式中字符常量的值是该字符的机器字节宽度的代码。汇编器假设字符代码是ASCII码。

3.4.2.2 字符串

字符串被写在一对双引号之间。可能包括双引号或空字符。在字符串中添加特殊字符的方法是在这些特殊字符前用反斜杠 \ 进行转义。应用于字符串的转义序列同样也适用于字符。

3.4.2.3 一般语法规则

表3-2对适用于汇编器的一般语法规则做了总结：

表3-2： 语法规则

字符	字符说明	语法用法
.	句点	开始伪指令
#	数字或井号	开始单行注释
/*	斜杠和星号	开始多行注释
*/	星号和斜杠	结束多行注释
:	冒号	结束标号定义
	不需要字符	开始立即数值
'c'	单引号中的字符	指定单个字符值
"string"	双引号中的字符串	指定一个字符串

© 2014 Microchip Technology Inc.

3.5 符号

符号是从所有字母、数字、下划线字符（_）和句点（.）组成的集合中挑选出来的一个或多个字符。符号不能以数字开头， 并且是区分大小写的（例如，foo与Foo是两个不同的符号）。符号没有长度限制，并且所有字符都有意义。

每个符号都有且仅有一个名称。在汇编语言程序中，每个名称都仅代表一个符号。您可在一个程序中多次使用同一个符号名。

- 局部符号
- 为符号赋予其他值
- 特殊点符号
- 预定义符号

3.5.1 局部符号

当需要在临时作用域内引用标号时，使用局部符号。局部符号的名称共有十个，并且在整个程序中可以被重复使用。可以使用名称“0”、“1”、...和“9”来引用这些符号。要定义局部符号，可以用格式‘N’, ‘N’, ..., ‘N’写一个标号（其中N代表0到9的任何数字）。要引用该符号的前一个定义，请写“Nb”，使用在定义标号时所使用的同一个数字。要引用局部标号的下一个定义时，请写“Nf”。“b”表示“往回”（backwards），“f”表示“往前”（forwards）。

对于使用这些标号的方法没有限制，您也可重复使用这些标号。可使用相同的数字“N”重复定义相同的局部标号，尽管只可引用该数字的上一个定义的局部标号（对于“往回”引用）或特定局部标号的下一个定义（对于“往前”引用）。

还请注意，头10个局部标号（“0:”. . . “9:”）的实现方法比其他局部标号的实现方法稍为高效。

此处给出了一个示例：

例3-5： 符号用法

```
1: b 1f
2: b 1b
1: b 2f
2: b 1b
```

等同于以下标号：

```
label_1: b label_3
label_2: b label_1
label_3: b label_4
label_4: b label_3
```

局部符号名称仅仅是一个符号表示。它们在被汇编器使用前被立即转换成更传统的符号名称。这些传统符号名称被存储在符号表中，出现在错误消息中，并且可以选择性地被发送到目标文件中。

3.6 为符号赋予其他值

可以通过在符号后面添加一个等号 “=”，然后再添加一个表达式来给符号赋予任意值。

示例：

```
VAR = 4
```

3.7 特殊点符号

特殊符号“.”指汇编器正在处理的当前地址。因此，表达式“melvin: .long .”定义 melvin 包含自身的地址。给 . 赋值的效果和 .org 伪指令相同。因此，表达式“.=.+4”与“.space 4”相同。

当在可执行代码段中使用时，“.”指程序计数器地址。在 PIC32 MCU 中，对于每个32位指令字，程序计数器都会递增4。在修改点符号后，用户代码应注意正确对齐指令。

3.7.1 为符号赋予其他值

可以通过在符号后面添加一个等号“=”，然后再添加一个表达式来给符号赋予任意值。

示例：

```
VAR = 4
```

3.7.2 特殊点符号

特殊符号“.”指汇编器正在处理的当前地址。因此，表达式“melvin: .long .”定义 melvin 包含自身的地址。给 . 赋值的效果和 .org 伪指令相同。因此，表达式“.=.+4”与“.space 4”相同。

当在可执行代码段中使用时，“.”指程序计数器地址。在 PIC32 MCU 中，对于每个32位指令字，程序计数器都会递增4。在修改点符号后，用户代码应注意正确对齐指令。

3.7.3 预定义符号

汇编器预定义了一些可通过源代码中的条件伪指令测试的符号。

表3-3： 预定义符号

符号	定义
P32MX	PIC32MX目标器件系列
P32MZ	PIC32MZ目标器件系列
HAS_MIPS32R2	器件支持MIPS32r2指令集
HAS_MIPS16	器件支持MIPS16e指令集
HAS_MICROMIPS	器件支持microMIPS指令集
HAS_DSPR2	器件支持DSPr2引擎
HAS_MCU	器件支持MIPS MCU扩展
HAS_L1CACHE	器件具有L1数据和程序缓存
HAS_VECTOROFFSETS	器件使用向量表的可配置偏移量

© 2014 Microchip Technology Inc.

3.8 表达式

表达式指定地址或数字值。在表达式前面和 / 或后面可以有空白。表达式的结果必须是一个绝对数字，或是到特定段的偏移量。如果表达式的值不是绝对的，并且表达式未提供足够的信息使汇编器能确定其所在的段，汇编器将终止并生成一条错误消息。

3.8.1 空表达式

空表达式没有值：它为空白或空值。任何需要绝对表达式之处，都可以省略此表达式，并且汇编器将假定此表达式的值为（绝对）0。

3.8.2 整数表达式

整数表达式是由运算符分隔的一个或多个参数。参数可以是符号、数字或子表达式。子表达式是一个左括号“(”后面跟着一个整数表达式和一个右括号“)”；或者是一个后面有参数的前缀运算符。

在整数表达式中，如果包含程序存储器中的符号，将按照程序计数器（PC）的单位进行求值。在 MIPS32 模式下，程序计数器每执行一个指令字就会递增 4。例如，要跳转到L标号所在指令的下一条指令，就应指定目标为L+4。

示例：

```
b  L+4
```

3.9 运算符

运算符是诸如+或%之类的算术函数。前缀运算符后面跟有一个参数。中缀运算符在参数之间。运算符的前面和/或后面可以是空白。

前缀运算符具有比中缀运算符高的优先级。中缀运算符的优先级顺序由其类型决定。

3.9.1 前缀运算符

汇编器有以下前缀运算符。每个运算符使用一个参数，参数必须是绝对的。

表3-4： 前缀运算符

运算符	说明	示例
-	求补。二进制补码。	-1
~	按位非。二进制反码。	~flags

3.9.2 中缀运算符

中缀运算符的两边各有一个参数。运算符根据其类型具有不同的优先级，如下表所示，但是优先级相同的运算按照从左到右的顺序执行。除了 + 或 – 之外，运算符都必须是绝对的，而且结果也必须是绝对的。

表3-5： 中缀运算符

运算符	说明	示例
算术运算		
*	乘	5 * 4 (=20)
/	除。截取方式与C运算符“/”相同。	23 / 4 (=5)
%	求余	30 % 4 (=2)
<<	左移。与C运算符“<<”相同	2 << 1 (=4)
>>	右移。与C运算符“>>”相同	2 >> 1 (=1)
按位操作		
&	按位与	4 & 6 (=4)
^	按位异或	4 ^ 6 (=2)
!	按位或非	0x1010 ! 0x5050 (=0xBFBF)
|	按位或	2 | 4 (=6)
简单算术运算		
+	加。如果某个参数是绝对的，结果就会包含另一个参数的段。不可以将来自不同段的参数相加。	4 + 10 (=14)
-	减。如果右边的参数是绝对的，结果就会包含左边参数的段。如果两个参数在同一个段中，结果就是绝对的。不可以将来自不同段的参数相减。	14 - 4 (=10)
关系运算		
==	等于	.if (x == y)
!=	不等于（也可表示为<>）	.if (x != y)
<	小于	.if (x < 5)
<=	小于或等于	.if (y <= 0)
>	大于	.if (x > a)
>=	大于或等于	.if (x >= b)
逻辑运算		
&&	逻辑与	.if ((x > 1) && (x < 10))
||	逻辑或	.if ((y != x) || (y < 100))

© 2014 Microchip Technology Inc.

3.10 特殊操作符

汇编器为下列操作提供了一组特殊操作符：

- 获取特定段的大小
- 获取特定段的起始地址
- 获取特定段的结束地址

DD 表3-6： 特殊操作符

操作符	说明
.sizeof.(*name*)	获取地址单元中段*name*的大小
.startof.(*name*)	获取段*name*的起始地址
.endof.(name)	获取段*name*的结束地址

3.10.1 获取特定段的大小

.sizeof.(section_name) 操作符可用于在发生链接过程之后获取特定段的大小（以字节为单位）。例如，要获取.data段的最终大小，使用：

```
.word .sizeof.(.data)
```

3.10.2 获取特定段的起始地址

.startof.(section_name)操作符可用于在发生链接过程之后获取特定段的起始地址。例如，要获取.data段的起始地址，使用：

```
.word .startof.(.data)
```

3.10.3 获取特定段的结束地址

.endof.(section_name) 操作符可用于在发生链接过程之后获取特定段的结束地址。例如，要获取.data段的结束地址，使用：

```
.word .endof.(.data)
```

注：

© 2014 Microchip Technology Inc.

第4章 汇编器伪指令

4.1 简介

伪指令是出现在源代码中的汇编器命令，通常不直接翻译为操作码。伪指令用于控制汇编器的输入、输出和数据分配。

> **注：** 汇编器伪指令*并非*目标指令（ADD、XOR和JAL等）。有关指令集的信息，请参见目标器件的数据手册。

MPLAB XC32 汇编器（xc32-as）支持的伪指令集虽然与 MPLAB XC16 汇编器（xc16-as）支持的伪指令有很多类似之处，但是也有许多不同之处。

本章涵盖以下主题：

- 定义段的伪指令
- 初始化常量的伪指令
- 声明符号的伪指令
- 定义符号的伪指令
- 修改段对齐方式的伪指令
- 格式化输出列表的伪指令
- 控制条件汇编的伪指令
- 用于替换/扩展的伪指令
- 包含其他文件的伪指令
- 控制诊断输出的伪指令
- 用于调试信息的伪指令
- 控制代码生成的伪指令

© 2014 Microchip Technology Inc.

4.2 定义段的伪指令

段为可定位的代码或数据块，它们占用 32 位器件存储器中连续的存储单元。预定义的段有三个：.text用于可执行代码段、.data用于初始化数据段以及.bss用于未初始化数据段。还可以定义其他段；链接器定义了几个有助于在 32 位存储器的特定区域内定位数据的段。

段伪指令如下：

- .bss
- .data
- .pushsection name
- .popsection
- .section name [, “flags”] (deprecated)
- .section name [, attr1[,…,attrn]]
- .text

.bss

定义

将后面的语句汇编到.bss（未初始化数据）段的末尾。

bss 段用于局部常用变量存储。可在 bss 段内分配地址空间，但是在程序执行之前不得命令将数据装入其中。程序开始运行时，bss段的所有内容都为全0字节。

使用.bss伪指令切换到bss段，然后如常定义符号。可以仅将全0值汇编到该段内。通常情况下，该段将仅包含符号定义和.skip伪指令。

示例

```
   # The following symbols (B1 and B2) will be placed in
   # the uninitialized data section.
.bss
B1:   .space 4   # 4 bytes reserved for B1
B2:   .space 1   # 1 byte reserved for B2
```

.data

定义

将后面的语句汇编到.data（已初始化数据）段的末尾。

示例

```
   # The following symbols (D1 and D2) will be placed in
   # the initialized data section.
.data
D1:.long 0x12345678   # 4 bytes
D2:.byte 0xFF         # 1 byte
```

链接器收集 .data 段（以及通过数据属性定义的其他段）的初始值并创建数据初始化模板。此模板可在应用程序启动期间进行处理以将初始值传送到存储器中。对于C语言应用程序，将自动调用库函数来实现。汇编项目可通过与 libpic32 库链接来使用此库。更多信息，请参见**已初始化数据**中对**第9.5.3节“运行时库支持”**的讨论。

© 2014 Microchip Technology Inc.

.pushsection *name*

该伪指令将当前段的内容压入段堆栈的栈顶，并用*name*替换当前段。每个.pushsection应有一个匹配的.popsection。

.popsection

用段堆栈栈顶的段替换当前段内容。栈顶段从堆栈弹出。

.section name [, “flags”] (deprecated)
.section name [, attr1[,...,attrn]]

将后面的代码汇编到名为 name 的段中。如果字符 * 被指定为名称，则汇编器将根据输入文件名以“文件名.s.scnn”（其中n表示自动生成的段名数）格式生成唯一的段名。

命名为 * 的段可用于节省存储器空间，因为汇编器不会向这些段添加对齐填充字符。未命名为 * 的段可能跨多个文件进行组合，因此为了确保所需对齐，汇编器必须添加对齐填充字符。

如果不存在可选参数，则段属性取决于段名。带隐含属性的保留段名表格中给出了带有隐含属性的保留段名。如果段名与保留段名匹配，则隐含属性将分配给该段。如果段名未被识别为保留段名，则默认属性将为data（数据存储器中的初始化存储）。

不宜使用除[.text, .data, .bss]之外的保留段名隐含属性。

如果使用了这些保留段的隐含属性，汇编器将发出警告。

如果第一个可选参数加有引号，该参数将被视为描述段属性的一个或多个标志。不建议使用加有引号的段标志。（见**附录A“已弃用的功能”**）。如果使用了加引号的段标志，则汇编器将发出警告。

如果第一个可选参数未加引号，则该参数将被视为属性列表的第一个元素。属性可以按任意顺序进行指定，且不区分大小写。存在两类段属性：表示段类型的属性和修改段类型的属性。

4.2.1 表示段类型的属性

表示段类型的属性是互斥的。对于给定的段，最多只能为其指定以下列出的属性之一。

表4-1： 表示段类型的属性

属性	说明
code	程序存储器中的可执行代码
data	数据存储器中的已初始化存储
bss	数据存储器中未初始化的存储
persist	数据存储器中的持久存储
ramfunc	数据存储器中的函数

4.2.2 修改段类型的属性

根据属性的不同，可通过其修改部分或全部段类型，如下面的表4-2。

表4-2： 修改段类型的属性

属性	说明	属性应用到：				
		code	**data**	**bss**	**persist**	**ramfunc**
address(a)	定位在绝对地址a处	x	x	x	x	
near	定位在存储器的前64k		x	x	x	
reverse	对齐结束地址+1		x	x	x	
align(n)	对齐起始地址	x	x	x	x	x
noload	分配，不装载	x	x	x	x	x
keep	保持段不受垃圾收集	x	x	x	x	x

4.2.3 组合修改段类型的属性

表4-3： 组合修改段类型的属性

	address	near	reverse	align	noload	keep
address		x	x		x	x
near	x		x	x	x	x
reverse		x			x	x
align	x	x			x	x
noload	x	x	x	x		x
keep	x	x	x	x	x	

以下段名可供用户应用程序使用：

表4-4： 保留段名

段名	生成者	在链接描述文件中映射到	隐含属性
.text	编译器或汇编器生成的指令		code
.text.*	使用-ffunction-sections编译时函数输出到此形式的独特指定段		code
.startup	留在链接描述文件中用于向后兼容性的C启动代码	kseg0_boot_mem	code
.app_excpt	通用异常处理程序	kseg0_boot_mem	code
.reset	复位处理程序	kseg0_boot_mem	code
.bev_excpt	BEV异常处理程序	kseg0_boot_mem	code
.vector_n	中断向量n	kseg0_boot_mem	code
.rodata	声明为const的字符串和C数据		code
.rodata.*	使用-fdata-sections编译时常量数据输出到此形式的独特指定段		code
.data	具有初始值且字节数>n（由采用-Gn选项编译确定）的变量。		data
.data.*	使用-fdata-sections编译的已初始化大变量		data

© 2014 Microchip Technology Inc.

表4-4： 保留段名 （续）

.ramfunc	RAM函数		data
.bss	未初始化的数据		bss
.lit4 / .lit8	汇编器决定存储在存储器中而不是指令流中的常量（通常为浮点数）。用于gp相对寻址。		data
.sdata	具有初始值且字节数<=n（由采用-Gn选项编译确定）的变量。用于gp相对寻址。		data
.sdata.*	使用-fdata-sections编译的小变量。用于gp相对寻址		data
.sbss	字节数<=n（由采用-Gn选项编译确定）的未初始化变量。用于gp相对寻址。		data
.sbss.*	使用-fdata-sections编译的未初始化小变量。用于gp相对寻址。		data
.bss	未初始化大变量		data
.bss.*	使用-fdata-sections编译的未初始化变量。		data
.heap	动态存储使用的堆		data
.stack	为堆栈保留的最小区间		data
.debug*	DWARF调试信息		info
.line	DWARF调试信息		info
.comment	#ident/.ident字符串		info
.reginfo	信息段		info

段伪指令示例

```
.section foo                        ;foo is initialized data memory.
.section bar,bss,align(256)         ;bar is uninitialized data memory, aligned.
.section *,data,near                ;section is near initialized data memory.
.section buf1,bss,address(0xa0000800);buf1 is uninitialized data memory at 0xa0000800.
.section *, code                    ;section is in program memory
```

.text

定义

将后面的语句汇编到.text（可执行代码）段的末尾。

示例

```
            .text
            .ent _main_entry
_main_entry:
            jal main
            nop
            jal exit
            nop
1:
            b  1b
            nop
            .end _main_entry
```

4.3 初始化常量的伪指令

常量初始化伪指令如下：

- .ascii “$string_1$” [, ..., “$string_n$”]
- .asciz “$string_1$” [, ..., “$string_n$”]
- .byte $expr_1$[, ..., $expr_n$]
- .double $value_1$[, ..., $value_n$]
- .float $value_1$[, ..., $value_n$]
- .single $value_1$[, ..., $value_n$]
- .hword $expr_1$[, ..., $expr_n$]
- .int $expr_1$[, ..., $expr_n$]
- .long $expr_1$[, ..., $expr_n$]
- .short $expr_1$[, ..., $expr_n$]
- .string “str”
- .word $expr_1$[, ..., $expr_n$]

.ascii “$string_1$” [, ..., “$string_n$”]

.ascii 使用用逗号隔开的 0 个或多个字符串字面量。它将每个字符串（没有自动在尾部追加全零字节）汇编到连续地址。

.asciz “$string_1$” [, ..., “$string_n$”]

.asciz 与 .ascii 相似，但是每个字符串后跟一个全零字节。.asciz 中的“z”表示“零”。该伪指令与 .string 同义。

.byte $expr_1$[, ..., $expr_n$]

.byte使用用逗号隔开的0个或多个表达式。每个表达式被汇编到当前段的下一个字节中。

.double $value_1$[, ..., $value_n$]

将一个或多个双精度（64 位）浮点常量以小尾数法（little-endian）格式汇编到连续的地址中。浮点数是IEEE格式的（见**第3.4.1.2节“浮点数”**）。

以下语句等效：

```
.double 12345.67
.double 1.234567e4
.double 1.234567e04
.double 1.234567e+04
.double 1.234567E4
.double 1.234567E04
.double 1.234567E+04
```

也可以指定浮点常量的十六进制编码形式。以下语句等效并且将值 12345.67 编码为 64 位双精度数：

```
.double 0e:40C81CD5C28F5C29
.double 0f:40C81CD5C28F5C29
.double 0d:40C81CD5C28F5C29
```


© 2014 Microchip Technology Inc.

.float *value*$_1$[, ..., *value*$_n$]

将一个或多个单精度（32 位）浮点常量以小尾数法格式汇编到连续的地址中。它等效于.single。浮点数是IEEE格式的（见**第3.4.1.2节“浮点数”**）。

以下语句等效：

```
.float 12345.67
.float 1.234567e4
.float 1.234567e04
.float 1.234567e+04
.float 1.234567E4
.float 1.234567E04
.float 1.234567E+04
```

也可以指定浮点常量的十六进制编码形式。以下语句等效并且将值12345.67编码为32位双精度数：

```
.float 0e:4640E6AE
.float 0f:4640E6AE
.float 0d:4640E6AE
```

.single *value*$_1$[, ..., *value*$_n$]

将一个或多个单精度（32 位）浮点常量以小尾数法格式汇编到连续的地址中。该伪指令与.float同义。浮点数是IEEE格式的（见**第3.4.1.2节“浮点数”**）。

.hword *expr*$_1$[, ..., *expr*$_n$]

将一个或多个2字节数以小尾数法格式汇编到连续的地址中。该伪指令与.short同义。

.int *expr*$_1$[, ..., *expr*$_n$]

将一个或多个4字节数以小尾数法格式汇编到连续的地址中。该伪指令与.long同义。

.long *expr*$_1$[, ..., *expr*$_n$]

将一个或多个4字节数以小尾数法格式汇编到连续的地址中。该伪指令与.int同义。

.short *expr*$_1$[, ..., *expr*$_n$]

将一个或多个2字节数以小尾数法格式汇编到连续的地址中。该伪指令与.hword同义。

.string *“str”*

该伪指令与.asciz同义。

.word *expr*$_1$[, ..., *expr*$_n$]

将一个或多个4字节数以小尾数法格式汇编到连续的地址中。

4.4 声明符号的伪指令

声明符号的伪指令如下：

- .comm symbol, length [, algn]
- .extern symbol
- .global symbol .globl symbol
- .lcomm symbol, length
- .weak symbol

.comm *symbol, length* [, *algn*]

.comm声明了一个名为*symbol*的共用符号。链接时，在某个目标文件中的共用符号可与另一个目标文件中定义的同名符号或同名的共用符号合并。如果链接器未发现该符号的定义（仅一个或多个共用符号），那么它将会分配一个*length*个字节的未初始化存储区。*length* 必须是绝对表达式。如果链接器发现有多个具有相同名称的共用符号，但是它们的大小不同，那么它将使用其中最大的长度分配存储空间。

.comm伪指令可使用可选的第三个参数。如果指定了*algn*，那么该值是symbol要求的地址对齐值（指定为字节边界），例如，对齐值16意味着该地址的低4位应为零。对齐值必须是绝对表达式，且必须是2的整数次幂。如果链接器为共用符号分配了未初始化存储地址，在存放该符号时它将使用对齐。如果未指定对齐，汇编器会将对齐值设置为小于或等于符号大小的2的最大幂，最大值为1。

.extern *symbol*

.extern 伪指令声明一个在其他模块中定义为全局、但可在当前模块中使用的符号名称。但是，所有符号默认为extern，因此该伪指令是可选的。

.global *symbol*
.globl *symbol*

.global 伪指令声明一个在当前模块中定义并可在其他模块中使用的符号。.global 使链接器可以看到该符号。如果在部分程序中定义了符号，与其链接的其他部分程序就都可使用该符号的值。否则，符号从链接到相同程序的其他文件中的同名符号获取其属性。

.globl和.global形式都是可接受的，以与其他汇编器兼容。

.lcomm *symbol, length*

为由*symbol*表示的局部共用符号保留*length*个字节。symbol的段和值都与新定义的局部共用符号相同。地址在 .bss 段中分配，因此在运行时，该字节的初始值为全 0。*symbol*没有被声明为全局变量，因此通常链接器看不到它。

.weak *symbol*

将名为 *symbol* 的符号标记为“弱”。当将定义为弱的符号和常规定义的符号相链接时，常规定义符号的使用不会产生错误。当链接尚未定义的弱符号时，弱符号的值将变为0，并且不会产生错误。

© 2014 Microchip Technology Inc.

4.5 定义符号的伪指令

定义符号的伪指令如下：

- .equ symbol, expression
- .equiv symbol, expression

.equ *symbol, expression*

该伪指令将*symbol*的值设置为*expression*。在汇编过程中，可以多次设置符号值。如果设置了全局变量，那么存储在目标文件中的值是最后一次赋给该变量的值。

.equiv *symbol, expression*

与 .equ 类似，不同之处在于使用本伪指令时，如果符号已被定义，汇编器将报错。注意，已被引用但实际上未被定义的符号视为未定义。

除了给出错误消息的内容之外，该伪指令近似等效于：

```
.ifdef SYM
.err
.endif
.equ SYM,VAL
```

4.6 修改段对齐方式的伪指令

下面列出了显式修改段对齐方式的伪指令。

> **注：** 用户代码必须小心以正确对齐修改段对齐方式或位置计数器的伪指令之后的下一条指令。

- `.align [algn[, fill]]`
- `.fill repeat[, size[, value]]`
- `.org new-lc[, fill]`
- `.skip size[, fill]`
- `.space size[, fill]`
- `.struct expression`

.align [*algn*[, *fill*]]

`.align`伪指令将位置计数器（在当前子段中）填充至特定的存储边界。第一个表达式（必须是绝对的）是位置计数器在递增之后必须具有的对齐方式，要求指定为低位为零的数。

汇编器接受的*algn*值的范围为0至最高15。`.align`为0会关闭创建伪操作数的数据使用的自动对齐方式。必须确保数据正确对齐。使用`.align`伪指令恢复自动对齐。

第二个表达式（也是绝对的）提供存储在填充字节中的*fill*的值。可以省略第二个表达式（及逗号）。如果省略了它，那么填充字节将默认为零。可以对存储器的闪存区使用0xFF。

.fill *repeat*[, *size*[, *value*]]

保留repeat个size字节的副本。*repeat*可以大于或等于0。*size*也可以大于等于0，但是如果大于8的话，它将被认为是8。每repeat个字节都来自于一个8字节的数。最高的4字节为0。最低的4字节是以小尾数法字节顺序存储的值。每个要被重复的size字节来自于该数最低的size个字节。

*size*是可选的。如果没有第一个逗号及后续的标记，size将假定为1。

*value*是可选的。如果没有第二个逗号及*value*，那么value将假定为零。

示例：

```
          .text
          .fill 0x3, 1, 0xFF
          .align   2
mylabel:b .
```


© 2014 Microchip Technology Inc.

.org *new-lc*[, *fill*]

.org伪指令将当前段的位置计数器递增至*new-lc*。*new-lc*可以是绝对表达式，或是与当前子段所在段相同的表达式。也就是说，不能使用 .org 伪指令跨段定位：如果*new-lc* 具有错误的段，那么 .org 伪指令被忽略。如果 *new-lc* 的段是绝对的，同时xc32-as发出了一条警告，那么只能说是*new-lc*的段与当前子段相同。

.org只能使位置计数器递增或保持不变，不能使用.org使位置计数器往回计数。

由于汇编器尝试在一次计数内汇编程序，因此*new-lc*不能为未定义。

请注意起始地址相对于段的起始地址，而不是相对于子段的起始地址。

当当前子段的位置计数器递增时，在当前位置计数器值与新位置计数器值之间的字节使用*fill*（应为绝对表达式）填充。如果省略了逗号和*fill*，那么fill默认为零。

.skip *size*[, *fill*]

.space *size*[, *fill*]

这两条伪指令分配*size*个字节，每个字节使用*fill*值填充。*size*和*fill*都是绝对表达式。如果省略了逗号和*fill*，那么*fill*假定为零。

.struct *expression*

切换到绝对段，并将段偏移量设置为 *expression*，它必须是一个绝对表达式。您可以如下方式使用它：

```
          .struct 0
field1:
          .struct field1 + 4
field2:
          .struct field2 + 4
field3:
```

这样可将符号 field1 的值定义为 0 ；将符号 field2 的值定义为 4 ；并将符号 field3 的值定义为 8。汇编将在绝对段中进行，在进一步汇编之前，您需要使用某种.section伪指令以切换到其他某个段。

4.7 格式化输出列表的伪指令

输出列表格式化伪指令如下：

- .eject
- .list
- .nolist
- .psize lines[, columns]
- .sbttl “subheading”
- .title “heading”

.eject

在生成汇编列表时在该点强制换页。

.list

控制（与 .nolist 结合）是否生成汇编列表。该伪指令会使内部计数器递增（初始值为1）。如果计数器的值大于零，则将生成汇编列表。

只有当使用 -a 命令行选项使能了列表输出，并且未使用 -an 命令行选项禁止格式处理时才有效。

.nolist

控制（与 .list 结合）是否生成汇编列表。该伪指令会使内部计数器递减（初始值为1）。如果计数器的值大于零，则将生成汇编列表。

只有当使用 -a 命令行选项使能了列表输出，并且未使用 -an 命令行选项禁止格式处理时才有效。

.psize *lines*[, *columns*]

声明在生成列表时每页显示的行数和列数（列数为可选）。

如果不使用.psize，列表使用默认行数60。可以省略逗号和列指定；默认宽度为200列。

当超过了指定的行数（或者使用.eject明确请求换页）时，汇编器将生成换页。

如果指定行数为0，则不会生成换页，保存使用.eject明确指定的换页。

.sbttl *“subheading”*

当生成汇编列表时，使用subheading作为副标题（第3行，紧接在标题行之后）。如果该伪指令出现在最开头的10行之内，它将影响当前页和后续页。

.title *“heading”*

当生成汇编列表时，使用heading作为标题（第2行，紧接在源文件名和页码行之后）。

© 2014 Microchip Technology Inc.

4.8 控制条件汇编的伪指令

条件汇编伪指令如下：

- .else
- .elseif expr
- .endif
- .if expr

.else

与.if伪指令结合使用，提供当.if值为false时汇编代码的另一条途径。

.elseif *expr*

与.if伪指令结合使用，提供当.if值为false而第2种条件存在时汇编代码的另一条途径。

.endif

标记仅被条件汇编的代码块的末尾。

.if *expr*

标记仅当参数*expr*不为零时才被视为被汇编的源程序一部分的代码段的开头。必须使用.endif标记条件代码段的结尾；也可选择包含由.else标记的另一种条件的代码。

汇编器也支持.if的以下变体。

.ifdecl *symbol*

如果已定义了指定符号，则将汇编其后的代码段。注意，已被引用但尚未定义的符号被视为未定义。

.ifc *string1*,*string2*

如果两个字符串相同，该伪指令汇编其后的代码段。可选择用单引号将字符串引起来。如果没有这样做，那么第一个字符串在第一个逗号处停止，第二个字符串在行末停止。包含空白的字符串应该在两边加单引号。字符串比较区分大小写。

.ifeq *absolute-expression*

如果参数为零，该伪指令汇编其后的代码段。

.ifeqs *string1*,*string2*

该伪指令是.ifc的另一种形式。必须在字符串两边加双引号。

.ifge *absolute-expression*

如果参数大于等于零，该伪指令汇编其后的代码段。

.ifgt *absolute-expression*

如果参数大于零，该伪指令汇编其后的代码段。

.ifle *absolute-expression*

如果参数小于等于零，该伪指令汇编其后的代码段。

.iflt *absolute-expression*

如果参数小于零，该伪指令汇编其后的代码段。

.ifnc *string1*,*string2*

该伪指令与 `.ifc` 相似，但是测试的意义相反：如果两个字符串不相同，该伪指令汇编其后的代码段。

.ifndef *symbol*

如果未定义指定符号，该伪指令汇编其后的代码段。大小写拼写形式都是等效的。注意，已被引用但尚未定义的符号被视为未定义。

.ifnotdef *symbol*

该伪指令与 `.ifndef` 相同。

.ifne *absolute-expression*

如果参数不等于零，该伪指令汇编其后的代码段（换句话说，它等效于 `.if` 伪指令）。

.ifnes *string1*,*string2*

该伪指令与 `.ifeqs` 相似，但是测试的意义相反：如果两个字符串不相同，该伪指令汇编其后的代码段。

© 2014 Microchip Technology Inc.

4.9 用于替换/扩展的伪指令

替换/扩展伪指令如下：

- .exitm
- .irp symbol, $value_1$ [, ..., $value_n$] ... .endr
- .irpc symbol, value ... .endr
- .macro

.exitm

提前从当前宏定义退出。请参见.macro伪指令。

.irp *symbol, $value_1$* [, ..., *$value_n$*] ... .endr

求给 *symbol* 赋不同值的一系列语句的值。这一系列语句以 .irp 伪指令开始，以 .endr 伪指令结束。对于每一个value，都会将*symbol*设置为*value*，并且汇编这一系列语句。如果没有列出任何 value，那么这一系列语句只被汇编一次，并且将symbol设置为空字符串。要引用这一系列语句中的symbol，请使用\symbol。

例如，汇编

```
.irp reg,0,1,2,3
lw $a\reg, 1032+\reg($sp)
.endr
```

等同于汇编

```
lw $a0,1032+0($sp)
lw $a1,1032+1($sp)
lw $a2,1032+2($sp)
lw $a3,1032+3($sp)
```

.irpc *symbol, value* ... .endr

求给 *symbol* 赋不同值的一系列语句的值。这一系列语句以 .irpc 伪指令开始，以 .endr伪指令结束。对于*value*中的每一个字符，都会将*symbol*设置为相应的字符，并且汇编这一系列语句。如果没有列出任何 *value*，那么这一系列语句只被汇编一次，并且将symbol设置为空字符串。要引用这一系列语句中的symbol，请使用\symbol。

例如，汇编

```
.irpc reg,0123
lw $a\reg, 1032+\reg($sp)
.endr
```

等同于汇编

```
lw $a0,1032+0($sp)
lw $a1,1032+1($sp)
lw $a2,1032+2($sp)
lw $a3,1032+3($sp)
```

.macro

伪指令.macro和.endm允许定义生成汇编输出的宏。例如，以下定义指定将一系列数字保存到存储器的宏SUM：

```
.macro SUM from=0, to=5
.long \from
.if    \+o-\from
SUM "(\from+1)",   \+o
.endif
.endm
```

通过此定义，“SUM 0,5”等同于以下汇编输入：

```
.long 0
.long 1
.long 2
.long 3
.long 4
.long 5
```

.macro *macname*

.macro *macname macargs* …

开始名为 *macname* 的宏的定义。如果宏定义需要参数，那么请在宏名后指定这些参数的名称，并以逗号或空格分隔。可通过在名称后跟=deflt为任何宏参数提供默认值。例如，这些是所有有效的.macro语句：

- .macro comm
  开始名为comm的宏的定义，它不带有参数。
- .macro plus1 p, p1
  .macro plus1 p p1
  上面两条语句中任一语句都可开始名为plus1的宏的定义，它带有两个参数；在宏定义内，请写\p或\p1以对参数求值。
- .macro reserve_str p1=0 p2
  开始名为reserve_str的宏的定义，它带有两个参数。第一个参数有默认值，但第二个参数没有。完成定义后，可以“reserve_str a,b”（使用\p1求值到a和\p2求值到b）或以“reserve_str ,b”（使用\p1求值为默认值（这里指“0”）和\p2求值到b）形式调用宏。

调用宏时，可按位置或按关键字指定参数值。例如，“SUM 9,17”等同于“sum to=9, from=17”。

.endm

标记宏定义的结束。

.exitm

提前从当前宏定义退出。

© 2014 Microchip Technology Inc.

\@

汇编器在此伪变量中保存已被执行过的宏的数量的计数；可使用 \@ 将该值复制到输出中，但是必须位于宏定义内。在下面的示例中，使用了递归宏来分配设定数量的带有标号的缓冲区：

```
       .macro make_buffers num,size
BUF\@: .space \size
       .if (\num - 1)
       make_buffers (\num - 1),\size
       .endif
       .endm

       .bss
       # create BUF0..BUF3, 16 bytes each
       make_buffers 4,16
```

此示例宏展开如以下列表所示：

```
   6            make_buffers (\num - 1),\size
   7            .endif
   8            .endm
   9
  10            .bss
  11            # create BUF0..BUF3, 16 bytes each
  12            make_buffers 4,16
  12     > BUF0:.space 16
  12 0000 > .space 16
  12     > .if (4-1)
  12     > make_buffers (4-1),16
  12     >> BUF1:.space 16
  12 0010 >> .space 16
  12     >> .if ((4-1)-1)
  12     >> make_buffers ((4-1)-1),16
  12     >>> BUF2:.space 16
  12 0020 >>> .space 16
  12     >>> .if (((4-1)-1)-1)
  12     >>> make_buffers (((4-1)-1)-1),16
  12     >>>> BUF3:.space 16
  12 0030 >>>> .space 16
  12     >>>> .if ((((4-1)-1)-1)-1)
  12     >>>> make_buffers ((((4-1)-1)-1)-1),16
  12     >>>> .endif
  12     >>> .endif
  12     >> .endif
  12     > .endif
```

.purgem "*name*"

取消定义宏 name，这样在以后使用该字符串时将不会将其展开。请参见上一页的 .macro伪指令。

.rept *count* ... .endr

将在 `.rept` 伪指令和下一条 `.endr` 伪指令之间的一系列行重复count遍。

例如，汇编

```
.rept 3
.long 0
.endr
```

等同于汇编

```
.long 0
.long 0
.long 0
```


© 2014 Microchip Technology Inc.

4.10 包含其他文件的伪指令

包含来自其他文件的数据的伪指令有：

- .incbin "file"[,skip[,count]]
- .include "file"

.incbin "*file*"[,*skip*[,*count*]]

.incbin伪指令在当前位置逐字包含*file*内容。假设该文件包含二进制数据。可使用-I命令行选项指定搜索路径（见**第2章“汇编器命令行选项”**）。要求在*file*两边加上双引号。

*skip*参数从文件起始处跳过数个字节。*count*参数指定要读取的最大字节数。注意该数据未以任何方式对齐，因此用户必须负责确保在.incbin伪指令之前和之后提供正确的对齐方式。

.include "*file*"

提供了一种在您的源代码中的指定点包含支持文件的方法。代码被当作跟在.include点以后被汇编。当到达包含的文件的末尾时，将会从.include之后的语句开始继续汇编原文件。

4.11 控制诊断输出的伪指令

其他伪指令如下：

- .abort
- .err
- .error "string"
- .fail expression
- .ident "comment"
- .print "string"
- .version "string"
- .warning "string"

.abort

显示消息“.abort detected. Abandoning ship.”（检测到.abort，放弃。），然后退出程序。

.err

当汇编器发现.err伪指令时，它将显示一条错误消息，并且除非使用了-Z选项，否则将不会生成目标文件。可以使用该伪指令报告在条件编译的代码中存在错误。

.error "*string*"

与.err相似，不同之处在于该伪指令将显示指定的字符串。

.fail *expression*

产生错误或警告。如果*expression*的值大于等于500，xc32-as将显示一条警告消息。如果其值小于500，as将显示一条错误消息。消息将包含*expression*的值。该伪指令有时在复杂的嵌套宏或条件汇编中有用。

.ident "*comment*"

给名为.comment的段添加注释。如果该段不存在则将创建它。链接器在分配存储空间时将忽略此段，但是会以链接的顺序将所有.comment段组合到一起。

.print "*string*"

在汇编过程中在标准输出中显示字符串*string*。

.version "*string*"

该伪指令创建一个.note段并将其放入一个ELF格式的NT_VERSION类型文本中。该文本的名称被设置为string。当输出文件的格式为ELF时，支持.version，否则它将被忽略。

.warning "*string*"

与伪指令.error类似，但发出的是警告。

© 2014 Microchip Technology Inc.

4.12 用于调试信息的伪指令

调试信息伪指令如下：

- .ent function
- .end
- .file fileno "filename"
- .fmask mask, offset
- .frame framereg, frameoffset, retreg
- .loc fileno, lineno [columnno]
- .mask mask, offset
- .size name, expression
- .sleb128 $expr_1$ [, ..., $expr_n$]
- .type name, description
- .uleb128 expr1[,...,exprn]

.ent *function*

该伪指令将function符号标记为与通用.type伪指令相似的函数。

.end

结束程序。

.file *fileno* "*filename*"

发出dwarf2行号信息时，.file将文件名分配到.debug_line文件名表。*fileno*操作数应该是用作该表中条目索引的唯一正整数。*filename*操作数是C字符串立即数。

*filename*索引的详细信息应显示给用户，因为文件名表与dwarf2调试信息的.debug_info段共用，因此用户必须知道表项将具有的确切索引。

.fmask *mask*, *offset*

不用于当前的PIC32 MCU。保持mask 0x00000000和offset 0。

.frame *framereg*, *frameoffset*, *retreg*

该伪指令描述了堆栈帧的外形。使用的虚拟帧指针是 *framereg*，通常表示为 $fp 或 $sp。帧指针是规范帧地址（canonical frame address，CFA）下的 *frameoffset* 字节，规范帧地址是进入函数时堆栈指针的值。返回地址最初保存在*retreg*中，直到它根据指示保存到.mask中为止。

.loc *fileno*, *lineno* [*columnno*]

目标文件的调试信息包含将汇编指令与源代码的行和列关联起来的行号矩阵。.loc 伪指令将添加与伪指令后紧跟的汇编指令对应的矩阵行。在添加行之前，*fileno*、*lineno*和*columnno*将应用到调试状态机。

.mask *mask, offset*

指示在当前函数的堆栈帧中保存哪些整型寄存器。*mask*被解释为位掩码，其中位n置1指示保存寄存器n。寄存器保存在自规范帧地址（CFA）开始的 *offset* 字节所在的块中，规范帧地址是进入函数时堆栈指针的值。

.size *name, expression*

该伪指令设置与符号 *name* 关联的大小。大小（以字节计）通过可使用标记算术的 *expression*计算。该伪指令通常用于设置函数符号的大小。

.sleb128 $expr_1$ [, ..., $expr_n$]

sleb128表示“以128为基的有符号小尾数法”。这是DWARF符号调试格式使用的数字的紧凑、长度可变的表示法。

.type *name, description*

该伪指令将符号 *name* 的类型设置为函数符号或目标符号。类型 *description* 字段有五种受支持的不同语法，以提供与各种其他汇编器的兼容性。支持的语法如下：

```
.type <name>,#function
.type <name>,#object
.type <name>,@function
.type <name>,@object
.type <name>,%function
.type <name>,%object
.type <name>,"function"
.type <name>,"object"
.type <name> STT_FUNCTION
.type <name> STT_OBJECT
```

.uleb128 $expr_1$[,...,$expr_n$]

uleb128 表示“以 128 为基的无符号小尾数法”。这是 DWARF 符号调试格式使用的数字的紧凑、长度可变的表示法。

© 2014 Microchip Technology Inc.

4.13 控制代码生成的伪指令

控制汇编器代码生成行为的伪指令如下：

- .set noat
- .set at
- .set noautoextend
- .set autoextend
- .set nomacro
- .set macro
- .set mips16e
- .set nomips16e
- .set noreorder
- .set reorder

.set noat

合成某些地址格式时，汇编器可能需要一个临时寄存器。默认情况下，汇编器将静默使用 at（$1）寄存器，即根据约定保留为汇编器的临时寄存器。在某些情况下，编译器不应使用该寄存器。.set noat伪指令阻止汇编器静默使用该寄存器。

.set at

允许汇编器静默使用at（$1）寄存器。

.set noautoextend

默认情况下，MIPS16指令可根据需要自动扩展为32位。伪指令.set noautoextend会将此功能关闭。当.set noautoextend有效时，任何32位指令必须使用.e修饰符显式地扩展（例如，li.e $4,1000）。伪指令.set autoextend可再使用一次，以根据需要自动扩展指令。

.set autoextend

使能自动将MIPS16指令扩展为32位。

.set nomacro

汇编器支持合成的指令以及合成到多条机器指令中的指令助记符。例如，sleu 指令汇编为sltu指令和xori指令。.set nomacro伪指令使汇编器在一条指令扩展到多条机器指令时发出警告消息。

.set macro

阻止有关合成指令的警告。

.set mips16e

使用MIPS16e ISA扩展进行汇编。

.set nomips16e

不要使用MIPS16e ISA扩展进行汇编。

.set noreorder

默认情况下，汇编器通过对其周围的指令重新排序来填充分支或延迟槽。此功能非常有用。

您有时可能希望对指令排序进行精确控制。请使用`.set noreorder`伪指令告知汇编器在遇到`.set reorder`伪指令之前阻止此功能。

.set reorder

允许汇编器对指令重新排序以填充分支或延迟槽。

.set micromips

使用microMIPS ISA模式进行汇编。

.set nomicromips

不要使用microMIPS ISA模式进行汇编。

© 2014 Microchip Technology Inc.

第5章 汇编器错误/警告/消息

5.1 简介

MPLAB XC32汇编器（xc32-as）可生成错误、警告和消息。此处给出了汇编器最常见的诊断消息的描述性列表。

本章涵盖以下主题：

- 严重错误
- 错误
- 警告
- 消息

5.2 严重错误

以下错误表明汇编器中发生了内部错误。如果汇编器产生下面列出的任何一个严重错误，请联系 Microchip（http://support.microchip.com）以获取支持。请确保提供了导致错误发生的源代码和命令行选项的完整详细信息。

- Bad char = '%c'
- Bad defsym; format is --defsym name=value
- Bad return from bfd_install_relocation: %x
- Broken assembler. No assembly attempted.
- Can't allocate elf private section data: %s
- Can't continue
- Can't create group: %s
- Can't extend frag %u chars
- Can't open a bfd on stdout %s
- Can't start writing .mdebug section: %s
- Cannot write to output file
- Could not write .mdebug section: %s
- Dwarf2 is not supported for this object file format
- Emulations not handled in this configuration
- Error constructing %s pseudo-op table: %s
- Expr.c(operand): bad atof_generic return val %d
- Failed sanity check
- Failed to read instruction table %s\n
- Failed to set up debugging information: %s
- Index into stored_fixups[] out of bounds
- Inserting into symbol table failed: %s
- Internal: bad mips opcode (bits 0x%lx undefined): %s %s.
- Internal: bad mips opcode (mask error): %s %s.
- Internal: bad mips opcode (unknown extension operand type `+%c'): %s %s.
- Internal: bad mips opcode (unknown operand type `%c'): %s %s.
- Internal error, line %d, %s
- Internal error: unknown dwarf2 format
- Internal: can't hash `%s': %s
- Invalid abi -mabi=%s
- Invalid listing option `%c'
- Macros nested too deeply
- Missing emulation mode name
- Multiple emulation names specified
- No compiled in support for 64 bit object file
- No object file generated
- Rva not supported
- Rva without symbol
- Too many fixups
- Unrecognized emulation name `%s'

© 2014 Microchip Technology Inc.

5.3 错误

下面列出的错误通常指示传递给汇编器的汇编源代码或命令行选项中存在错误。

符号

.abort detected. Abandoning ship.

调用 `.abort` 伪指令时出现用户错误。

.else without matching .if

发现一个没有 .if 伪指令前导的 `.else` 伪指令。

.elseif after .else

在 `.else` 伪指令之后指定了 `.elseif` 伪指令。修改您的代码，使得 `.elseif` 伪指令出现在 `.else` 伪指令之前。

.elseif without matching .if

发现一个没有 `.if` 伪指令前导的 `.elseif` 伪指令。

.endfunc missing for previous .func

前面的 `.func` 伪指令缺少 `.endfunc` 伪指令。

.endif without .if

发现一个没有 `.if` 伪指令前导的 `.endif` 伪指令。

.err encountered.

调用 `.err` 伪指令时出现用户错误。

.Ifeqs syntax error

两个用逗号分隔的且用双引号引起的字符串没有作为参数传递给 `.ifeqs` 伪指令。

.Set pop with no .set push

尝试从没有选项的堆栈弹出选项。请在 `.set pop` 之前使用 `.set push`。

.Size expression too complicated to fix up

`.size` 表达式可以为常量或使用标记减法。

A

A bignum with underscores may not have more than 8 hex digits in any word.

任何单词中的大数（bignum）常量不得超过8个十六进制数字。

A bignum with underscores must have exactly 4 words.

使用下划线标注的大数常量必须正好是4个部分，每个部分由8个十六进制数字组成。

Absolute sections are not supported.

此汇编器不支持绝对段命令。

Alignment not a power of 2.

对齐值必须是2的次幂。将该对齐值修改为2的次幂。

Alignment too large: 15. Assumed.

请求对齐长度大于15。汇编器自动继续使用对齐值15。

Arg/static registers overlap.

MIPS32模式的保存/恢复操作将重叠的寄存器用于参数和静态变量。

Argument must be a string.

.error或.warning伪指令的参数必须是用双引号引起的字符串。

Attempt to allocate data in common section.

此伪指令尝试将数据分配到不可分配的段。请将数据分配到其他段。

Attempt to get value of unresolved symbol *name*

汇编器无法获得未解析符号的值。

Attempt to set value of section symbol.

对段符号赋值是非法的。

B

Backward ref to unknown label *label*:

引用的标号找不到或未在此定义。

Bad .common segment name

无法确定`.comm`符号的适当对齐值。之前找到的`.comm`符号可能不正确。

Bad escaped character in string.

该字符串使用了不标准的斜杠转义字符。

Bad expression.

无法确定表达式类型或此表达式类型的操作数类型不正确。

Bad floating literal: %s.

无法将标记转换为浮点值。

Bad floating-point constant: exponent overflow.

由于指数溢出，无法将标记转换为浮点值。

Bad floating-point constant: unknown error code=%d.

无法将标记转换为浮点值。

Bad format for ifc or ifnc.

ifc或ifnc伪指令的参数不正确。它们必须是2个用逗号分隔且用双引号引起的字符串。

Bad or irreducible absolute expression.

绝对表达式使用了不期望的运算符类型。

© 2014 Microchip Technology Inc.

Bad register expression.

DWARF调试伪指令具有无效的寄存器表达式。

Bignum invalid.

在表达式中指定的大数值无效。

C

Can't parse register list.

在MIPS32模式下，寄存器列表无效。

Can't resolve value for symbol `%s'.

汇编器无法获得符号的实际值。

Constant too large.

从基址寄存器进行常量偏移量的符号扩展时，常量过大。

Could not skip to num in file filename

`.incbin`伪指令的跳过参数对于给定文件无效。

D

Duplicate else.

每个.if伪指令有且仅能有一个对应的.else伪指令。

E

End of file inside conditional.

汇编器识别出缺少条件结束伪指令。请在文件结束前终止条件。

End of macro inside conditional.

汇编器识别出缺少宏结束伪指令。请在文件结束前终止宏。

Expected address expression.

表达式非法、缺失或者应为常量地址时却为大数。

Expected comma after %s.

此伪指令的参数必须用逗号分隔。

Expected comma after name `%s' in .size directive.

此伪指令的参数必须用逗号分隔。

Expected quoted string.

参数应为用引号引起的字符串。

Expected simple number.

此参数必须只是数字。

Expected symbol name.

此参数必须为符号名。

Expression out of range.

表达式超出伪指令或指令的范围（例如，需要16位值时却给出了32位值）。

Expression too complex.

表达式应为符号或常量。

F

File not found:%s.

指定给伪指令（例如 `.incbin`）的文件不能按照指定打开。

File number %ld already allocated.

传递给.file伪指令的文件号已在使用。

File number less than one.

传递给 `.file`伪指令的文件号必须> 1。

Floating point number invalid.

浮点数无效。

G

Global symbols not supported in common sections.

外部符号在MRI共用段中不受支持。

I

Ignoring attempt to redefine symbol name

`.weakext`伪指令正在重定义的符号已被定义。

Improper insert size

指定给INS指令的字段宽度对于移位位置无效。

Improper extract size

指定给EXT指令的字段宽度对于移位位置无效。

Instruction insn requires absolute expression.

该指令要求常量表达式。

Invalid astatic register list

MIPS32e的扩展SAVE/RESTORE指令的aregs字段指定了一个无效的astatic寄存器列表。

Invalid arg register list.

MIPS32e的扩展SAVE/RESTORE指令的aregs字段指定了一个无效的参数寄存器列表。

Invalid coprocessor 0 register number.

传递给此指令的协处理器0寄存器编号无效。

Invalid coprocessor sub-selection value (%ld), not in range 0-7.

协处理器子选择值必须在范围0-7内。

© 2014 Microchip Technology Inc.

Invalid frame size

帧大小无效且无法编码。

Invalid identifier for .ifdef.

指定的标识符名称无效。必须以合法的字符开头。

Invalid register list.

在MIPS32模式下，寄存器列表包含了无效的寄存器。

Invalid segment %s.

尝试在无效段中更改位置计数器。

Invalid static register list.

静态寄存器列表应只包含$s2-$s8。

J

Jump to misaligned address (0x%lx).

跳转目标地址未对齐。

Junk at end of line, first unrecognized character is char

期望的输入后跟有无关字符。

Junk at end of line, first unrecognized character valued 0xval

期望的输入后跟有无关字符。

L

Load/store address overflow (max 32 bits).

装载/存储地址大于32位宽。请确保标号正确。

Local label is not defined.

引用的局部标号从未定义。

Lui expression not in range 0..65535.

Load Upper Immediate表达式中加载的立即数应在32位范围内。

N

New line in title.

标题字符串应用双引号引起。

No such section.

在`.global`伪指令中指定的段名不存在（例如`.global foo .myscn`）

Non-constant expression in .elseif statement

`.elseif`语句需要`expr`常量表达式。`.elseif`伪指令的参数必须为常量值，从而可在第一次执行该伪指令时就可以对值进行解析。确保对此参数中使用的符号执行的任何`.equ`操作位于该伪指令之前。更多详细信息，请参见**第4.8部分“控制条件汇编的伪指令”**。

Non-constant expression in .if statement.

`.if`语句需要`expr`常量表达式。`.if`伪指令中的参数必须为常量值，从而可在第一次执行该伪指令时就可以对值进行解析。确保对此参数中使用的符号执行的任何`.equ`操作位于该伪指令之前。更多信息，请参见**第4.8节“控制条件汇编的伪指令”**。

`Noreorder' must be set before `nomacro'.

在设置nomacro之前设置noreorder。

Number (0x%lx) larger than 32 bits.

装载了大于32位宽的值到寄存器。

Number larger than 64 bits.

装载了大于64位宽的值到HI/LO寄存器。

O

Offset too large.

偏移量必须在符号扩展32位范围内。

Opcode not supported on this processor.

该指令操作码在PIC32 MCU上不受支持。

Operand overflow.

操作数不在该指令的允许范围内。

Operation combines symbols in different segments.

表达式的左边和右边位于两个不同的段内。汇编器不知道如何处理该表达式。

R

Register value used as expression.

表达式的运算符不是有效运算符，而是寄存器。

Relocation reloc isn't supported by the current ABI.

该重定位不受PIC32小尾数法ELF输出格式的支持。

S

Seek to end of .incbin file failed `%s'.

找不到`.incbin`指定的文件的末尾。

Skip (%ld) + count (%ld) larger than file size (%ld).

`.incbin`跳过值和计数值之和超过文件的大小。

Store insn found in delay slot of noreorder code.

考虑将存储移至跳转前面，并在延迟中使用nop代替。

Symbol `%s' can not be both weak and common.

在同一个源文件中对同一个符号使用了`.weak`伪指令和`.comm`伪指令。

© 2014 Microchip Technology Inc.

Symbol name is already defined.

符号无法重新定义。

Symbol definition loop encountered at `%s'.

由于遇到自引用循环，而无法定义符号。符号无法根据自身的值进行定义。

Syntax error in .startof. Or .sizeof.

汇编器发现了`.startof.`或`.sizeof.`，但却没有找到左括号“(”或右括号“)”。

T

This string may not contain '\0'.

该字符串必须是有效的C字符串，不能包含“`\0`”。

Treating warnings as errors.

通过`--fatal-warnings`命令行选项指示汇编器将所有警告当作错误处理。

U

Unassigned file number num

`.loc`伪指令指定了尚未使用的文件号。

Unclosed '('.

左括号“(”没有对应的右括号“)”。请添加缺失的“)”。

Unexpected register in list.

在MIPS32模式下，使用了无效的寄存器。检查此指令的操作数。

5.4 警告

当汇编器做出某种假设以得以继续汇编某个有缺陷的程序时，汇编器会产生警告。不应该忽略警告。每条警告都应该被仔细检查并改正以确保汇编器明白操作的目的。有时，警告消息能指出您程序中的错误。

符号

.end directive missing or unknown symbol

缺失 .end 函数调试信息伪指令，或未定义相关的符号。请确保 .end 伪指令位于 .ent 伪指令后的适当位置。

.end directive without a preceding .ent directive.

.end函数调试信息伪指令没有相关的、用于将符号标记为函数的.ent伪指令。请确保 .end 伪指令位于在 .ent 伪指令后的适当位置。

.end not in text section

.end 函数调试信息伪指令必须在带有可执行代码的段内。

.end symbol does not match .ent symbol.

.end 函数调试信息伪指令的函数参数与前面的 .ent 伪指令的函数参数不一致。请确保 .end 伪指令位于其相应的.ent伪指令后的适当位置。

.endr encountered without preceding .rept, .irc, or .irp

.endr 伪指令用于结束 .rept、.irc 或.irp 序列；但是此 .endr 伪指令前面没有 .rept、.irc 或.irp 伪指令。请确保 .endr 伪指令在代码中的位置正确。

.ent or .aent not in text section.

.ent 函数调试信息伪指令必须在包含可执行代码的段内。

.fail *expr* encountered

如果.fail表达式的值为500或更大，汇编器将显示一条警告消息。该消息将包含此表达式的值。

.fill size clamped to 8

.fill size 的值可以为0或更大，但是如果大于8，那么被视为具有值8。

.frame outside of .ent

.frame 伪指令描述了堆栈帧，因此必须在函数内使用。

.incbin count zero, ignoring filename

.incbin 计数应大于0，从文件中读取0个字节没有意义。

© 2014 Microchip Technology Inc.

.mask/.fmask outside of .ent

`.mask/.fmask`堆栈帧信息应在`.ent`函数内定义。请确保`.mask/.fmask`伪指令在源代码中的位置正确。

.popsection without corresponding .pushsection; ignored

汇编器没有将某个段压入堆栈之前，无法从段堆栈中弹出段。

.previous without corresponding .section; ignored

没有提供可与当前段交换的先前段。请确保`.previous`伪指令在源代码中的位置正确。

.space repeat count is negative, ignored.

`.space`的`size`参数参数必须大于0。

.space repeat count is zero, ignored.

`.space`的`size`参数必须大于0。

A

Alignment negative: 0 assumed.

`.align`对齐值必须是非负的、2的次幂。.align为0会关闭创建伪操作数的数据使用的自动对齐方式。

Alignment too large: 15 assumed.

`.align`对齐值大于15。有效范围为[0,15]。

D

Divide by zero.

DIV指令的$zero寄存器为RT。

Division by zero.

表达式尝试除以零。请检查操作数。

E

Extended instruction in delay slot.

MIPS32e 扩展指令不能位于跳转延迟槽中，因为它会导致发生未定义的行为。请将该指令移出延迟槽。

F

Floating point constant too large.

浮点常数的十六进制编码过大。请确保浮点值使用32位或64位IEEE格式进行正确的编码。

I

Ignoring changed section attributes for *name*

如果汇编器第二次看到特定段时指定了段属性，那么段属性应该与汇编器第一次看到时的相同。汇编器假设第一次设置的段属性是正确的。

Ignoring changed section entity size for *name*

段实体大小应与汇编器第二次看到该特定段时的相同。汇编器假设它第一次看到该段时的段实体大小是正确的。

Ignoring changed section type for *name*

段类型应与汇编器第二次看到该特定段时的相同。汇编器假设它第一次看到该段时的段类型是正确的。

Ignoring incorrect section type for *name*

按名称切换到特殊的预定义段时，该段的类型应与预定义类型一致。汇编器使用该段的预定义类型。

Immediate for %s not in range 0..1023 (%lu).

调试器的中断代码不在有效范围内。常规用户代码不应使用此专为调试器保留的指令。

Improper shift amount (%lu).

移位指令（例如SLL、SRA和SRL）的移位值超出了范围。

Instruction sne: Instruction %s: result is always false.

SNE指令测试的条件的结果始终为false。（例如，s操作数为zero寄存器，t为非零常量表达式）。

Instruction seq: result is always true.

SEQ指令测试的条件的结果始终为false。（例如，s操作数为zero寄存器，t为常数0）。

Invalid merge entity size.

段合并实体大小必须非负。

Invalid number.

常量使用了不可识别的格式。请检查该常量的前缀和数制。

J

Jump address range overflow (0x%lx).

跳转指令的目标地址超出了228字节的“页”。

L

Left operand is a bignum; integer 0 assumed.

表达式的左操作数不是整数，而是大数。汇编器只能基于整数对表达式求值，所以它假设此操作数为整数0。

Left operand is a float; integer 0 assumed.

表达式的左操作数不是整数，而是浮点数。汇编器只能基于整数对表达式求值，所以它假设此操作数为整数0。

© 2014 Microchip Technology Inc.

Line numbers must be positive; line number %d rejected.

该伪指令仅接受正整数作为行号。

M

Missing close quote; (assumed).

单字符的引号未正确关闭。

Missing operand; zero assumed.

表达式中缺少操作数（可能是正确大小的操作数）。汇编器假设整数0并继续。

O

Operand overflow.

在（基址寄存器+偏移量）操作数中使用的常量表达式仅接受32位有符号常数。

R

Repeat < 0; .fill ignored.

.fill伪指令的repeat参数必须非负。

Right operand is a bignum; integer 0 assumed.

表达式的右操作数不是整数，而是大数。汇编器只能基于整数对表达式求值，所以它假设此操作数为整数0。

Left operand is a float; integer 0 assumed.

表达式的右操作数不是整数，而是浮点数。汇编器只能基于整数对表达式求值，所以它假设此操作数为整数0。

S

Setting incorrect section attributes for *name*

对特定段设置段属性时，段的属性应与该段的预定义类型一致。汇编器使用该段的预定义类型。

Setting incorrect section type for *name*

对特定段设置段属性时，段的属性应与该段的预定义类型一致。汇编器使用该段的预定义类型。

Size negative; .fill ignored.

`.fill`伪指令的size参数必须非负。

T

Tried to set unrecognized symbol: *name*

.set伪指令中的符号不是可识别的PIC32 MCU汇编器符号。

Truncated file *filename*, *num1* of *num2* bytes read.

从`.incbin`文件中读取的字节数小于counts参数中指定的数。

U

Unknown escape \escape in string; ignored.

该字符串包含无法识别的反斜杠转义字符。请检查反斜杠后的字符是否正确。

Used $at without .set noat.

该代码正在使用 $at（汇编器临时）寄存器，但是汇编器在生成合成的宏指令时可能会使用它。请使用`.set noat`伪指令告知汇编器不要静默使用该寄存器。

5.5 消息

当汇编器做出某种不关键的假设以得以继续汇编某个有缺陷的程序时，汇编器会生成消息。可以忽略消息。但是消息有时能指出您程序中的错误。

© 2014 Microchip Technology Inc.

第2部分—— MPLAB XC32目标链接器

第6章 链接器概述 97
第7章 链接器命令行接口 105
第8章 链接描述文件 119
第9章 链接器处理 147
第10章 链接器示例 163
第11章 链接器错误/警告 167

注：

© 2014 Microchip Technology Inc.

第6章 链接器概述

6.1 简介

MPLAB XC32目标链接器（xc32-ld）将用于PIC32 MCU系列器件的可重定位目标代码和归档文件转换成二进制代码。该32位链接器是一个Windows控制台应用程序，它为开发可执行代码提供了平台。该链接器是来自Free Software Foundation的GNU链接器的一部分。

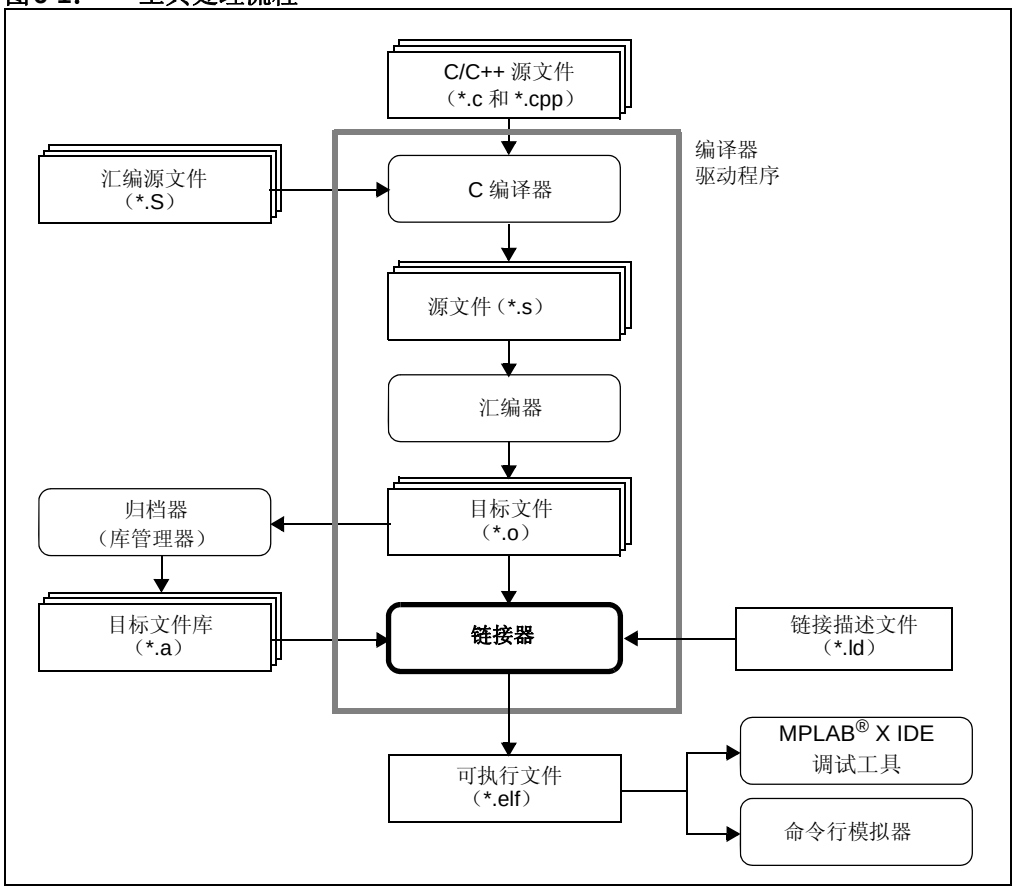
本章涵盖以下主题：

- 链接器和其他开发工具
- 特性集
- 输入/输出文件

6.2 链接器和其他开发工具

XC32链接器将来自XC32汇编器的目标文件和来自XC32归档器/库管理器的归档文件转换成可执行文件。工具处理流程的概况请参见图6-1。

图6-1： 工具处理流程



6.3 特性集

该链接器的显著特性包括：

- 用户定义的最小堆栈分配
- 用户定义的堆分配
- 可在Windows操作系统下使用
- 具有适用于所有当前PIC32器件的链接描述文件
- 命令行接口
- 是MPLAB X IDE的集成组件

6.4 输入/输出文件

下表列出了链接器的输入和输出文件。

表6-1： 链接器文件

扩展名	说明
输入	
.o	目标文件
.a	库文件
.ld	链接描述文件
输出	
.elf和.out	链接器输出文件
.map	映射文件

与Microchip MPLINK™链接器不同，32位链接器不生成绝对列表文件。32位链接器能够创建映射文件和二进制 ELF 文件（可能包含也可能不包含调试信息）。欲获得与 MPLINK 的列表文件相似的文本输出，请通过 xc32-objdump 二进制实用程序运行 ELF 文件。

6.4.1 目标文件

是由汇编器生成的可重定位代码。该链接器接受ELF格式的目标文件。

6.4.2 库文件

为方便起见，将一组目标文件组合在一起的集合。

6.4.3 链接描述文件

链接描述文件或命令文件：

- 指示链接器在何处放置段
- 指定给定器件的存储器地址范围
- 可以自定义将用户定义的段放置在特定的地址

更多有关链接描述文件的信息，请参见**第8章 “链接描述文件”**。

© 2014 Microchip Technology Inc.

例6-1： 链接描述文件

> **注：** 此简化的链接描述文件示例不是一个完整、可实际使用的有效链接描述文件，仅用于举例说明。

```
OUTPUT_FORMAT("elf32-tradlittlemips")
OUTPUT_ARCH(pic32mx)
ENTRY(_reset)

MEMORY
{
 kseg0_program_mem(rx): ORIGIN=0x9D000000, LENGTH=0x8000
 kseg0_boot_mem       : ORIGIN=0x9FC00490, LENGTH=0x970
 exception_mem        : ORIGIN=0x9FC01000, LENGTH=0x1000
 kseg1_boot_mem       : ORIGIN=0xBFC00000, LENGTH=0x490
 debug_exec_mem       : ORIGIN=0xBFC02000, LENGTH=0xFF0
 config3              : ORIGIN=0xBFC02FF0, LENGTH=0x4
 config2              : ORIGIN=0xBFC02FF4, LENGTH=0x4
 config1              : ORIGIN=0xBFC02FF8, LENGTH=0x4
 config0              : ORIGIN=0xBFC02FFC, LENGTH=0x4
 kseg1_data_mem  (w!x): ORIGIN=0xA0000000, LENGTH=0x2000
 sfrs                 : ORIGIN=0xBF800000, LENGTH=0x100000
}

SECTIONS
{
  .text ORIGIN(kseg0_program_mem)  :
  {
    _text_begin = . ;
    *(.text .stub .text.* )
    *(.mips16.fn.*)
    *(.mips16.call.*)
    _text_end = . ;
  } >kseg0_program_mem =0
  .data :
  {
    _data_begin = . ;
    *(.data .data.* .gnu.linkonce.d.*)
    KEEP (*(.gnu.linkonce.d.*personality*))
    *(.data1)
  } >kseg1_data_mem AT>kseg0_program_mem
  .bss :
  {
    *(.dynbss)
    *(.bss .bss.* )
    *(COMMON)
    . = ALIGN(32 / 8) ;
  } >kseg1_data_mem
  .stack ALIGN(4) :
  {
    . += _min_stack_size ;
  } >kseg1_data_mem
}
```

6.4.4 链接器输出文件

默认情况下，链接器输出二进制文件的名称是`a.out`。通过在命令行上指定`-o`选项可以改写该默认名称。MPLAB X IDE 的项目管理器使用 `-o` 选项来命名输出文件*`projectname`*`.elf`，其中*`projectname`*是MPLAB X IDE项目的名称。

二进制文件格式为可执行和链接格式（Executable and Linking Format，ELF）文件。可执行和链接格式最初由 UNIX 系统实验室（UNIX System Laboratories，USL）作为应用程序二进制接口（Application Binary Interface，ABI）的一部分开发并发布。ELF规范是工具接口标准（Tool Interface Standards，TIS）委员会（一个致力于标准化开发工具可视的软件接口的微计算机行业成员的联合组织）努力的结晶。

ELF文件内的调试信息采用DWARF调试信息格式。DWARF格式是TIS委员会的又一个努力的结晶，它使用一系列调试条目来定义源程序的低级表示。DWARF 的使用者（例如MPLAB X IDE）可使用这种表示法来创建原始源程序的精确描述。

6.4.5 映射文件

链接器生成的映射文件由以下部分组成：

- 归档成员表——列出包含在该链接中的归档文件中所有成员的名称。
- 存储器使用情况报告——给出在程序存储器和数据存储器中所有输出段的起始地址和长度。它还显示存储区中的存储器使用百分比。
- 存储器配置——列出为链接定义的所有存储区。
- 链接描述文件和存储器映射——显示根据链接描述文件所指定的包含在链接中的模块、段和符号。
- 外部交叉引用表（可选）——显示根据名称排序的符号。为每个符号都给出了一个文件名列表。如果符号已被定义，那么列出的第一个文件是该定义的位置。列出的其余文件包含对该符号的引用。

© 2014 Microchip Technology Inc.

例6-2：　映射文件

```
Archive member included because of file (symbol)

size\libc.a(general-exception.o)
         size/crt0.o (_general_exception_context)
size\libc.a(default-general-exception-handler.o)
         size\libc.a(general-exception.o) (_general_exception_handler)
size\libc.a(default-bootstrap-exception-handler.o)
         size/crt0.o (_bootstrap_exception_handler)
size\libc.a(default-on-reset.o)
         size/crt0.o (_on_reset)
size\libc.a(default-on-bootstrap.o)
         size/crt0.o (_on_bootstrap)
size\libc.a(default-nmi-handler.o)
         size/crt0.o (_nmi_handler)

Microchip PIC32 Memory-Usage Report

kseg0 Program-Memory Usage
section    address  length   (dec)  Description
------- ----------  --------------  -----------
.text   0x9d000000   0x678    1656  Application's executable code
.rodata 0x9d000678    0x14      20  Read-only constant data
.data   0x9d00068c     0xf     244  Data-initialization template
.sdata  0x9d000780     0x4       4  Small data-initialization template
Total kseg0_program_mem used:
                     0x784    1924  0.4% of 0x80000

kseg0 Boot-Memory Usage
section    address  length   (dec)  Description
-------  ---------- --------------  -----------
.startup 0x9fc00490  0x1e0     480  C startup code
Total kseg0_boot_mem used:
                     0x1e0     480  19.9% of 0x970

Exception-Memory Usage
section    address     length   (dec)  Description
-------    ---------- --------------  -----------
.app_excpt 0x9fc01180   0x10      16  General-Exception handler
.vector_1  0x9fc01220    0x8       8  Interrupt Vector 1
Total exception_mem used  :
                        0x18      24  0.6% of 0x1000

kseg1 Boot-Memory Usage
section    address     length   (dec)  Description
-------    ----------  --------------  -----------
.reset     0xbfc00000    0x10      16  Reset handler
.bev_excpt 0xbfc00380    0x10      16  BEV-Exception handler
Total kseg1_boot_mem used  :
                         0x20      32  2.7% of 0x490
        ----------------------------------------------
         Total Program Memory used  :
                        0x99c    2460  0.5% of 0x81e00
        ----------------------------------------------
```

```
kseg1 Data-Memory Usage
section    address     length   (dec)  Description
-------    ----------  --------------  -----------
.data      0xa0000000    0xf4     244  Initialized data
.sdata     0xa00000f4     0x4       4  Small initialized data
.sbss      0xa00000f8     0x4       4  Small uninitialized data
.bss       0xa00000fc   0x10c     268  Uninitialized data
.heap      0xa0000208   0x800    2048  Dynamic Memory heap
.stack     0xa0000a08   0x400    1024  Min space reserved for stack
         Total kseg1_data_mem used  :
                        0xe08    3592  11.0% of 0x8000
        ------------------------------------------------------------
            Total Data Memory used  :
                       0xe08     3592  11.0% of 0x8000
        ------------------------------------------------------------
Memory Configuration

Name             Origin             Length             Attributes
kseg0_program_mem 0x9d000000         0x00080000         xr
kseg0_boot_mem   0x9fc00490         0x00000970
exception_mem    0x9fc01000         0x00001000
kseg1_boot_mem   0xbfc00000         0x00000490
config0          0xbfc02ffc         0x00000004
kseg1_data_mem   0xa0000000         0x00008000         w !x
sfrs             0xbf800000         0x00100000
*default*        0x00000000         0xffffffff

Linker script and memory map

LOAD size/crt0.o
                0x00000800       _min_heap_size = 0x800
START GROUP
LOAD size\libc.a
LOAD size\libm.a
LOAD size\libmchp_peripheral_32MX360F512L.a
END GROUP
LOAD C:/xc32-Tools/bin/../lib/gcc/pic32mx/3.4.4/size\libgcc.a
              0x00000400    PROVIDE (_min_stack_size, 0x400)
              0x00000000    PROVIDE (_min_heap_size, 0x0)
LOAD ./proc/32MX360F512L\processor.o
              0x00000001    PROVIDE (_vector_spacing, 0x1)
              0x9fc01000    _ebase_address = 0x9fc01000
              0xbfc00000    _RESET_ADDR = 0xbfc00000
              0xbfc00380    _BEV_EXCPT_ADDR = 0xbfc00380
              0x9fc01180    _GEN_EXCPT_ADDR = (_ebase_address + 0x180)

.reset        0xbfc00000    0x10
 *(.reset)
 .reset       0xbfc00000    0x10 size/crt0.o
              0xbfc00000        _reset
.bev_excpt    0xbfc00380    0x10
 *(.bev_handler)
 .bev_handler 0xbfc00380    0x10 size/crt0.o
.vector_0     0x9fc01200    0x0
 *(.vector_0)

.startup      0x9fc00490    0x1e0
 *(.startup)
 .startup     0x9fc00490    0x1e0 size/crt0.o
```


© 2014 Microchip Technology Inc.

```
.text         0x9d000000    0x678
              0x9d000000         _text_begin = .
 *(.text .stub .text.* .gnu.linkonce.t.*)
 .text        0x9d000000    0x18 size/crt0.o
 .text        0x9d000018    0x110 intermediate\object.o
              0x9d000089         testfunct
              0x9d0000a0         main
              0x9d000018         foo
 .text        0x9d000128    0xc intermediate   est.o
              0x9d000128         mylabel
 .text.general_exception
              0x9d000134    0xd0 size\libc.a(general-exception.o)
              0x9d000134         _general_exception_context
 .text._general_exception_handler
       0x9d0005bc 0x8 size\libc.a(default-general-exception-handler.o)
            0x9d0005bc         _general_exception_handler
 .text._bootstrap_exception_handler
    0x9d0005c4  0x8 size\libc.a(default-bootstrap-exception-handler.o)
              0x9d0005c4         _bootstrap_exception_handler
 .text._on_reset
              0x9d0005cc    0x8 size\libc.a(default-on-reset.o)
              0x9d0005cc         _on_reset
 .text._on_bootstrap
              0x9d0005d4    0x8 size\libc.a(default-on-bootstrap.o)
              0x9d0005d4         _on_bootstrap
 .text        0x9d0005dc    0x18 size\libc.a(default-nmi-handler.o)
              0x9d0005dc         _nmi_handler
.sdata        0xa00000f4    0x4 load address 0x9d000780
              0xa00000f4         _sdata_begin = .
.heap         0xa0000208    0x800
              0xa0000208         _heap = .
              0xa0000a08         . = (. + _min_heap_size)
 *fill*       0xa0000208    0x800 00

.stack        0xa0000a08    0x400
              0xa0000e08         . = (. + _min_stack_size)
 *fill*       0xa0000a08    0x400 00

.ramfunc      0xa0001000    0x0 load address 0x9d000784
              0xa0001000         _ramfunc_begin = .
 *(.ramfunc .ramfunc.*)
              0xa0001000         . = ALIGN (0x4)
              0xa0008000         _stack =
                                 (_ramfunc_length >0x0)?
                                 (_ramfunc_begin - 0x4):0xa0008000

OUTPUT(test-2.elf elf32-tradlittlemips)

Cross Reference Table

Symbol         File
PORTE          ./proc/32MX360F512L\processor.o
               size\libc.a(default-nmi-handler.o)
               size\libc.a(general-exception.o)
               intermediate/test.o
               size/crt0.o
foo            intermediate\cobject.o
main           intermediate\cobject.o
               size/crt0.o
mylabel        intermediate\asmobject.o
funct          intermediate\cobject.o
```

注：

© 2014 Microchip Technology Inc.

第7章 链接器命令行接口

7.1 简介

除了与MPLAB X IDE配合使用，MPLAB XC32目标链接器（xc32-ld）也可以通过命令行接口使用。

本章涵盖以下主题：

- 链接器接口语法
- 编译驱动程序链接器接口语法
- 控制输出文件创建的选项
- 控制运行时初始化的选项
- 控制信息输出的选项
- 修改链接映射输出的选项

7.2 链接器接口语法

该链接器支持大量的命令行选项，但是在实际应用中，在具体情况下仅会使用其中很少选项。

```
xc32-ld [options] file...
```

> **注：** 命令行选项区分大小写。

例如，xc32-ld 经常被用来链接目标文件和归档文件来生成二进制文件。要链接文件hello.o：

```
xc32-ld -o output hello.o -lpic32
```

此操作使xc32-ld生成一个被称为output的文件，作为将文件hello.o与归档文件libpic32.a链接的结果。

链接C语言应用程序时，通常会链接几个归档文件（亦称为“库”），它们包含在链接命令中。归档文件列表可在--start-group和--end-group选项中指定，以帮助解决循环引用的问题：

```
xc32-ld -o output hello.o --start-group -lpic32 -lm -lc --end-group
```

可以用任何顺序指定传递给 xc32-ld 的命令行选项，并且可以任意重复。使用不同的参数重复大部分选项要么不起作用，要么会覆盖前面出现（在此命令行之前出现）的此选项。可以被有效指定多次的选项将说明如下。

非选项类参数是要被链接在一起的目标文件。目标文件参数除了不能放在一个命令行选项及该命令行选项的参数之间外，可以放在命令行选项的前面、中间或后面。

通常链接器至少要调用一个目标文件，但您可以使用 -l 和描述文件命令语言指定其他类型的二进制输入文件。如果没有指定二进制输入文件，链接器就不会产生任何输出，并发出“No input files”（没有输入文件）消息。

如果链接器不能识别目标文件的格式，就会将该文件假定为一个链接描述文件。以这种方式指定的描述文件将增大链接所要使用的主链接描述文件（默认链接描述文件或用 -T 指定的描述文件）。此功能使链接器可以不链接看似目标文件或归档的文件，而实际上只定义一些符号值，或使用INPUT或GROUP来加载其他目标。

对于名称为单个字母的选项，其参数必须紧跟在选项字母的后面，并且选项与参数之间不插入空白，或者作为独立参数，紧跟在需要它们的选项后面。

对于名称为多个字母的选项，选项名称前面可加上一条或两条短线；例如-trace-symbol和--trace-symbol相同。但此规则有一个例外。以字母o开头的多字母选项前面只能加两条短线。

多字母选项的参数和选项名称之间必须用等号隔开，或者作为独立参数，紧跟在需要它们的选项后面。例如，--trace-symbol srec和--trace-symbol=srec相同。多字母选项名称的独特缩写是可以接受的。

© 2014 Microchip Technology Inc.

7.3 编译驱动程序链接器接口语法

实际上，经常通过编译驱动程序 xc32-gcc 调用链接器。编译驱动程序命令行的基本格式为：

```
xc32-gcc [options] files
```

> **注：** 命令行选项和文件扩展名区分大小写。

要将链接器选项从编译驱动程序传递到链接器，请使用 `-Wl,option` 选项。

例 7-1： 编译驱动程序命令行

```
xc32-gcc -mprocessor=32MX360F512L "input.o" -o"output.elf"
   -Os -Wl,--defsym=_min_heap_size=2048,-Map="mapfile.map",
   --cref,--report-mem
```

相比直接调用链接器，通过编译驱动程序调用链接器有几个优点。

- 驱动程序的 `-mprocessor` 选项允许驱动程序将正确的器件特定头文件和库搜索路径传递给链接器。例如，指定 `-mprocessor=32MX360F512L` 时，驱动程序将相应的器件特定库搜索路径 `pic32mx/lib/proc/32MX360F512L` 传递给链接器。此路径允许链接器为目标器件查找正确的默认链接描述文件和处理器库。
- 驱动程序接受选择适当multilib置换所需的C编译器的优化、ISA模式和浮点支持选项。例如，传递 `-Os` 长度优化选项时，驱动程序将 `pic32mx/lib/size` 作为库搜索路径传递，以便链接器使用对长度进行过优化的预编译库。更多关于C编译器 multilib 特性的信息，请参见《MPLAB® XC32 C/C++ 编译器用户指南》（DS50001686G_CN）。

7.4 控制输出文件创建的选项

输出文件创建选项如下：

- -(archives -), --start-group archives, --end-group
- -d, -dc, -dp
- --defsym sym=expr
- --discard-all (-x)
- --discard-locals (-X)
- --fill=option
- --gc-sections
- --library name (-l name)
- --library-path <dir> (-L <dir>)
- -nodefaultlibs
- -nostartfiles
- -nostdlib
- --output file (-o file)
- --p PROC
- --relocatable (-r, -i, -Ur)
- --retain-symbols-file file
- --section-start sectionname=org
- --script file (-T file)
- --strip-all (-s)
- --strip-debug (-S)
- -Tbss address
- -Tdata address
- -Ttext address
- --undefined symbol (-u symbol)
- --no-undefined
- --wrap symbol

7.4.1 -(*archives* -), --start-group *archives*, --end-group

启动和终止组。

归档应该是一个归档文件列表。它们可以是明确的文件名，也可以是 -l 选项。指定的归档会被重复搜索直至无新的未定义的引用被创建。通常情况下，仅使用命令行中指定的顺序对归档搜索一次。如果需要该归档中的某个符号来解析由某一目标引用的未定义的符号，而该目标位于本命令行中后面出现的归档中，链接器将不能解析该引用。通过将归档分组，可以重复搜索所有的归档，直到所有可能的引用都被解析。使用此选项会大大降低链接器的性能，因此最好是仅在两个或两个以上归档之间发生不可避免的循环引用时，才使用此选项。

7.4.2 -d, -dc, -dp

强制定义共用符号。

即使指定了可重定位输出文件（使用 -r 选项），仍然为共用符号分配空间。描述文件命令FORCE_COMMON_ALLOCATION具有相同的作用。

© 2014 Microchip Technology Inc.

7.4.3 --defsym *sym=expr*

定义符号。

在输出文件中创建全局符号，其中包含由 *expr* 给出的绝对地址。可以根据需要多次使用此选项来在命令行中定义多个符号。在以下情况下对 *expr* 支持有限算术形式：您可以给出一个十六进制常数或现有符号的名称，或者使用 + 和 - 来对十六进制常数或符号进行加减运算。

> **注：** *sym*、等号（=）和*expr*之间不应该有空白。

7.4.4 --discard-all (-x)

丢弃所有局部符号。

7.4.5 --discard-locals (-X)

丢弃临时局部符号。

7.4.6 --fill=option

`--fill=option`

填充未用的程序存储器。格式为：

```
--fill=[wn:]expression[@address[:end_address] | unused]
```

address和end_address将指定要填充的程序存储器地址范围。如果未提供end_address，则expression将被写入地址为address的特定存储单元。可指定未使用的可选立即数值以指示将填充所有未使用的存储器。如果未提供存储单元参数，则将填充所有未使用的存储器。expression将说明如何填充指定的存储器。以下选项可用：

单个值：

```
xc32-ld --fill=0x12345678@unused
```

一系列值：

```
xc32-ld --fill=1,2,3,4,097@0x9d000650:0x9d000750
```

递增值：

```
xc32-ld --fill=7+=711@unused
```

默认情况下，链接器将使用指令字长度的数据进行填充。对于 32 位器件，默认的填充宽度为32位。但是，您可以使用 [wn:] 指定值宽度，其中n为填充值的宽度，n属于 [1, 2, 4, 8]。可在命令行上指定多个填充选项；链接器将始终先处理特定存储单元的填充选项。

7.4.7 --gc-sections

使能未使用输入段的垃圾收集。此选项与 -r 不兼容。可通过在命令行上指定 *--no-gc-sections* 来恢复默认行为（不执行此垃圾收集）。

链接期间使用垃圾收集时，标记不应删除的段通常非常有用。可通过用KEEP()括起输入段的通配项（如在KEEP(*(.init))或KEEP(SORT_BY_NAME(*)(.ctors))中一样）的方式标记段。

7.4.8 --library *name* (-l *name*)

搜索库*name*。

将归档文件*name*添加到要链接的文件的列表中。该选项可以被任意多次使用。对于指定的每个*name*，xc32-ld均会在其路径列表中搜索lib*name*.a文件出现的次数。此链接器只会在由命令行指定的位置中对归档搜索一次。如果归档定义了一个符号，而该符号在出现在命令行中该归档前的目标中并未被定义，那么链接器将包含来自该归档的相应文件。但是，在命令行后面出现的目标中未定义的符号不会使链接器再次搜索归档。有关强制链接器多次搜索归档的方法，请参见 -(选项。您可以在命令行中多次列出同一归档。

如果链接器不能识别归档文件的格式，它将忽略此文件。因此，库和链接器之间的版本不匹配可能导致“undefined symbol”（符号未定义）错误。

7.4.9 --library-path <*dir*> (-L <*dir*>)

将<*dir*>添加到库搜索路径中。

将路径<*dir*>添加到xc32-ld搜索归档库和xc32-ld控制描述文件的路径列表中。可以多次使用此选项。搜索目录是按照命令行中指定的顺序来进行的。所有的 -L 选项都适用于所有的-l选项，而不管选项以怎样的顺序出现。也可以使用SEARCH_DIR命令在链接描述文件中指定库路径。按这种方式指定的目录在命令行中出现链接描述文件之处进行搜索。

7.4.10 -nodefaultlibs

链接时不要使用标准系统库。只有指定的库才被传递到链接器。编译器可生成对memcmp、memset和memcpy的调用。这些入口通常由标准编译器库中的入口解析。指定了此选项时，应通过某个其他机制提供这些入口点。

7.4.11 -nostartfiles

不要将默认的预编译C启动文件（pic32mx/lib/crt0.o）传递给链接器。要为该应用程序提供您自己的启动代码版本。

7.4.12 -nostdlib

链接时不要使用标准系统启动文件或库。不传递启动文件给链接器，只传递指定的库给链接器。编译器可生成对memcmp、memset和memcpy的调用。这些入口通常由标准编译器库中的入口解析。指定了此选项时，应通过某个其他机制提供这些入口点。

7.4.13 --output *file* (-o *file*)

设置输出ELF文件名。

使用*file*作为xc32-ld生成的程序的名称；如果未指定此选项，默认情况下使用名称a.out。

7.4.14 --p PROC

指定目标处理器（例如，32MX795F512L）。

为链接指定目标处理器。

© 2014 Microchip Technology Inc.

7.4.15 --relocatable (-r, -i, -Ur)

生成可重定位输出文件。

即，生成可以作为 xc32-ld 的输入文件的输出文件。这通常被称为部分链接（Partial Linking）。如果未指定此选项，将生成绝对文件。

7.4.16 --retain-symbols-file *file*

仅保留在*file*中列出的符号。

仅保留*file*文件中列出的符号，丢弃所有其他符号。*file*只是一个纯文本文件，每行有一个符号名称。此选项在一个大的全局符号表被逐渐累加的环境中尤其有用，可以节省运行时存储器空间。--retain-symbols-file 不会丢弃未定义的符号或需要重定位的符号。您只能在命令行中指定一次--retain-symbols-file。它将覆盖-s和-S。

7.4.17 --section-start sectionname=***org***

在输出文件中*org*给出的绝对地址处定位段。可根据需要多次使用此选项来在命令行中定位多个段。*org*必须是一个十六进制整数；为了与其他链接器兼容，可以省略通常与十六进制值连在一起的前导0x字符。

注： sectionname、等号（=）和*org*之间不应该有空白。

7.4.18 --script *file* (-T *file*)

读取链接描述文件。

从文件 *file* 读取链接命令。此命令将替换 xc32-ld 的默认链接描述文件（而不是添加一个链接描述文件），这样 *file* 就必须指定所有说明目标格式的必需内容。如果*file*不存在，*xc32-ld*会在任何前面出现的-L选项指定的目录中查找它。多个-T选项将逐个累加。

7.4.19 --strip-all (-s)

删除所有符号。省略所有来自输出文件的符号信息。

7.4.20 --strip-debug (-S)

删除调试符号。省略来自输出文件的调试器符号信息（但不是所有符号）。

7.4.21 -Tbss *address*

设置.bss段的地址。

使用*address*作为输出文件的bss段的起始地址。*address*必须是一个十六进制整数；为了与其他链接器兼容，可以省略通常与十六进制值连在一起的前导0x字符。

此段的地址通常在链接描述文件中指定。

7.4.22 -Tdata *address*

设置.data段的地址。

使用 *address* 作为输出文件的数据段的起始地址。*address* 必须是一个十六进制整数；为了与其他链接器兼容，可以省略通常与十六进制值连在一起的前导0x字符。

此段的地址通常在链接描述文件中指定。

7.4.23 -Ttext *address*

设置.text段的地址。

使用 *address* 作为输出文件的文本段的起始地址。*address* 必须是一个十六进制整数；为了与其他链接器兼容，可以省略通常与十六进制值连在一起的前导0x字符。

此段的地址通常在链接描述文件中指定。

7.4.24 --undefined *symbol* (-u *symbol*)

从对*symbol*的未定义引用开始。

强制将*symbol*作为未定义的符号输入到输出文件。例如，这样做可触发标准库中其他模块的链接。可以使用不同的选项参数重复使用-u来输入其他未定义的符号。

7.4.25 --no-undefined

不允许未定义的符号。

7.4.26 --wrap *symbol*

为*symbol*使用包装（wrapper）函数。

为符号使用包装函数。任何对符号的未定义引用都将被解析为__wrap_symbol。任何对__real_symbol的未定义引用都将被解析为符号。这可被用于为系统函数提供包装。包装函数应被称为__wrap_symbol。如果需要调用系统函数，应该调用__real_symbol。

这里有一个小例子：

```
void *
__wrap_malloc (int c)
{
  printf ("malloc called with %ld\n", c);
  return __real_malloc (c);
}
```

如果使用--wrap malloc将此文件与其他代码相链接，所有对malloc的调用都会转而调用函数__wrap_malloc。在__wrap_malloc中调用__real_malloc将调用真正的malloc函数。您可能还希望提供一个__real_malloc函数，这样在没有--wrap选项的情况下链接也能成功。如果这样做的话，就不应该将__real_malloc的定义与__wrap_malloc放在相同的文件中，否则在链接器有机会将调用包装为malloc之前，汇编器就已将调用解析了。

© 2014 Microchip Technology Inc.

7.5 控制运行时初始化的选项

运行时初始化选项如下：

- `--data-init`
- `--no-data-init`
- `--defsym=_min_stack_size=size`
- `--defsym=_min_heap_size=size`

7.5.1 `--data-init`

支持已初始化数据。（这是默认选项）。

创建一个名为`.dinit`的特殊输出段作为运行时数据初始化的模板。libpic32.a中的C启动模块将解释此模板，并将初始值复制到已初始化的数据段。在调用 main() 函数之前，其他数据段（如 `.bss`）将被清零。请注意，持久数据段（`.pbss`）不受此选项影响。

7.5.2 `--no-data-init`

不支持已初始化数据。

禁止通常为支持运行时数据初始化所创建的模板。指定此选项时，链接器将在 libpic32.a 中选择较短形式的C启动模块。如果应用程序包含需要初始化的数据段，则将生成一条警告消息并丢弃初始数据值。数据段的存储将如常分配。

7.5.3 `--defsym=_min_stack_size=`***size***

默认链接描述文件提供 1024 字节的最小堆栈大小。可使用 `--defsym` 选项定义 `_min_stack_size`符号以更改默认***size***值。请注意，实际的有效堆栈大小可能比最小大小大。

```
xc32-gcc foo.c -Wl,--defsym=_min_stack_size=1536
```

7.5.4 `--defsym=_min_heap_size=`***size***

默认链接描述文件提供0字节的堆大小。可使用--defsym选项定义_min_heap_size符号以更改此默认size值。链接器使用此值定义的大小创建堆。

```
xc32-gcc foo.c -Wl,--defsym=_min_heap_size=2048
```

7.6 控制MULTILIB库选择的选项

multilib 是一组预编译的目标库。multilib 中的每个目标库使用一组不同的编译器选项进行编译。multilib 为链接器提供了使目标库与编译应用程序所使用的编译器选项相匹配的功能。预编译的目标库代表着编译器选项的最常见组合。

当调用编译驱动程序来链接应用程序时，驱动程序选择与应用程序选项对应的目标库版本。这些选项应传递到编译驱动程序，而不是相应的链接器。然后编译驱动程序将选项转换为调用链接器时的合适的 -L 库搜索路径。

长度与速度（-Os与-O3）

选择 -Os 以优化长度，或选择 -O0 至 -O3 以优化速度。

-O0

选择未经优化的 multilib 目标库置换。（这是命令行接口的默认选项，但是 MPLAB X IDE的项目管理器默认传递一个其他的优化选项）。

-O1

选择使用优化级别1编译的multilib目标库置换。

-O2

选择使用优化级别2编译的multilib目标库置换。此优化级别提供执行速度和代码长度之间的良好平衡。此multilib优化级别是大多数应用程序的理想选择。

-O3

选择使用优化级别3编译的multilib目标库置换。此优化级别可使执行速度达到最大。

-Os

选择对代码长度进行优化编译的multilib目标库置换。例如：

```
xc32-gcc foo.o -Os -o project.elf
```

7.6.1 指令集模式（MIPS32/MIPS16E/microMIPS）

根据-mips16、-mmicromips或-mips32r2选择multilib置换，例如：

```
xc32-gcc foo.o -O3 -mips16 -o project.elf
```

7.6.2 软件浮点与无浮点支持

无浮点库置换的开销通常比浮点库置换的开销要小的多。如果应用程序不需要浮点支持，使用此选项。

-mno-float

选择不支持软件浮点操作的multilib目标库置换。

示例：

```
xc32-gcc foo.o -Os -mno-mips16 -mno-float -o project.elf
```

-msoft-float

选择完全支持软件浮点的multilib目标库置换。

© 2014 Microchip Technology Inc.

7.7 控制信息输出的选项

信息输出选项如下：

- `--check-sections`
- `--no-check-sections`
- `--help`
- `--no-warn-mismatch`
- `--report-mem`
- `--trace (-t)`
- `--trace-symbol symbol (-y symbol)`
- `-V`
- `--verbose`
- `--version (-v)`
- `--warn-common`
- `--warn-once`
- `--warn-section-align`

7.7.1 `--check-sections`

检查是否有覆盖的段地址（**这是默认选项**）。正常情况下，链接器执行此检查，如果发现有覆盖情况，将产生一条合适的错误消息。

7.7.2 `--no-check-sections`

不检查段地址是否重叠。仅用于诊断存储器分配问题。

7.7.3 `--help`

显示选项帮助。

在标准输出中显示命令行选项的汇总并退出。

7.7.4 `--no-warn-mismatch`

在输入文件不匹配时不发出警告。

由于某些原因会导致输入文件不匹配（可能由于文件已针对不同处理器编译或文件被编译时的字节存储顺序不同），通常如果尝试将这些不匹配的输入文件链接在一起，`xc32-ld`就会报错。此选项告知`xc32-ld`应该默许这种可能的错误。只有在您已经采取了一些特殊措施以确保链接器不应报错的情况下，才可小心使用该选项。

7.7.5 `--report-mem`

显示存储器使用报告。

将链接过程中存储器使用情况的汇总输出到标准输出中。此报告也会出现在链接映射文件中。

7.7.6 `--trace (-t)`

跟踪文件。

当`xc32-ld`处理输入文件时显示输入文件的名称。

7.7.7 --trace-symbol *symbol* (-y *symbol*)

跟踪提及*symbol*之处。

显示每个含有*symbol*的被链接文件的名称。可以多次使用此选项。链接中有未定义的符号但您又不知道这个符号是从哪里引用的时，此选项非常有用。

7.7.8 -V

显示版本和其他信息。

7.7.9 --verbose

在链接过程中输出大量信息。

显示 xc32-ld 的版本号。显示可以打开和不能打开的输入文件。如果使用默认的内建描述文件，则显示该链接描述文件。

7.7.10 --version (-v)

显示版本信息。

7.7.11 --warn-common

警告重复的共用符号。

当共用符号和另一个共用符号或符号定义结合在一起时发出警告。此选项使您可以从组合全局符号中找出潜在问题。有三种全局符号，在此以C语言示例来说明：

```
int i = 1;
```

一个定义，它出现在输出文件的已初始化数据段中。

```
extern int i;
```

一个未定义的引用，它不分配空间。在某个地方必定有此变量的定义或共用符号。

```
int i;
```

一个共用符号。如果变量只有（一个或多个）共用符号，它将出现在输出文件的未初始化数据区域中。

此链接器将把同一个变量的多个共用符号合并为一个符号。如果它们大小不同，生成的符号将使用最大的那个。如果有同一个变量的定义，链接器将把共用符号转为声明。

--warn-common选项可以产生五种警告。每个警告都由两行组成：第一行说明刚刚遇到的符号，第二行说明上一次遇到的名称相同的符号。两个符号中至少有一个是共用符号。

将一个共用符号转换为引用，因为该符号已经有定义了。

```
file(section): warning: common of ‘symbol’ overridden by definition
file(section): warning:defined here
```


© 2014 Microchip Technology Inc.

将一个共用符号转换为引用，因为在后面遇到了该符号的定义。除遇到符号的顺序不同之外，这与上一种情况相同。

```
file(section): warning: definition of ‘symbol’ overriding common
file(section): warning: common is here
```

将某个共用符号与上一个大小相同的共用符号合并。

```
file(section): warning: multiple common of ‘symbol’
file(section): warning: previous common is here
```

将某个共用符号与上一个更大的共用符号合并。

```
file(section): warning: common of ‘symbol’ overridden by larger common
file(section): warning: larger common is here
```

将某个共用符号与上一个更小的共用符号合并。除遇到符号的顺序不同之外，这与上一种情况相同。

```
file(section): warning: common of ‘symbol’ overriding smaller common
file(section): warning: smaller common is here
```

7.7.12 --warn-once

对每个未定义的符号仅警告一次，而不是对每个引用该符号的模块都警告一次。

7.7.13 --warn-section-align

请注意，段对齐的空隙是正常的。此选项有助于确定最小化空隙的方法。

如果段的开头由于对齐而发生更改，发出警告。这意味着（通常按顺序）分配的存储器中引入了一个空隙。通常情况下，输入段会设置对齐。只有在没有明确指定地址（也就是说SECTIONS命令没有指定段的起始地址）时，地址才会发生改变。

7.8 修改链接映射输出的选项

链接映射输出修改选项如下：

- `--cref`
- `--print-map (-M)`
- `-Map file`

7.8.1 `--cref`

输出交叉引用表。

如果生成链接器映射文件，交叉引用表将被输出到映射文件中。否则，它将输出到标准输出中。此表的格式非常简单，从而使链接描述文件可以在必要的时候很容易地处理它。符号将显示出来并按名称排序。为每个符号都给出了一个文件名列表。如果符号已被定义，那么列出的第一个文件是该定义的位置。剩余的文件包含对该符号的引用。

7.8.2 `--print-map (-M)`

在标准输出中显示映射文件。链接映射提供有关链接的信息，包括：

- 目标文件和符号被映射到存储器中的位置。
- 共用符号的分配。
- 链接中包含的所有归档成员，以及导致这些归档成员被包含进来的符号。

7.8.3 `-Map` *`file`*

写映射文件。

将链接映射输出到文件*`file`*。请参见`--print-map (-M)`选项的说明。

© 2014 Microchip Technology Inc.

第8章 链接描述文件

8.1 简介

链接描述文件用于控制MPLAB XC32目标链接器（xc32-ld）的功能。默认情况下，该链接器将内建的链接描述文件与器件特定的头文件一起使用。但是，您也可以定制链接描述文件以在应用程序中对链接器进行专门控制。

本章涵盖以下主题：

- 链接描述文件概述
- 命令行信息
- 默认链接描述文件
- 添加定制链接描述文件到MPLAB X IDE项目
- 链接描述文件命令语言
- 链接描述文件中的表达式

8.2 链接描述文件概述

链接描述文件控制链接过程的各个方面，包括：

- 分配数据存储器和程序存储器
- 将段从输入文件映射到输出文件
- 构造特殊数据结构（如中断向量表）

链接描述文件为包含一系列命令的文本文件。每条命令要么是可能跟有参数的关键字，要么是对符号的赋值。

8.3 命令行信息

可使用`-T`选项或`--script`选项在命令行上指定链接描述文件（见**第7.4节“控制输出文件创建的选项”**）：

```
xc32-ld -o output.elf input.o --script mylinkerscript.ld
```

如果通过`xc32-gcc`调用链接器，添加`-Wl,`前缀允许将该选项传递给链接器：

```
xc32-gcc -o output.elf input.o -Wl,--script,mylinkerscript.ld
```


© 2014 Microchip Technology Inc.

8.4 默认链接描述文件

仅对于PIC32MX器件：

如果没有在命令行中指定任何链接描述文件，链接器将使用内部版本（内建的默认链接描述文件）。默认的链接描述文件具有适用于所有 PIC32 MCU 的段映射。它使用 `INCLUDE` 伪指令来包含器件特定存储区。

默认的链接描述文件适用于大多数的 PIC32 MCU 应用程序。只有需要特定存储器分配的应用程序才需要应用程序特定的链接描述文件。可通过使用 `--verbose` 选项调用链接器来检查默认的链接描述文件：

```
xc32-ld --verbose
```

在正常的工具套件安装中，默认的链接描述文件的副本位于 `\pic32mx\lib\ldscripts\elf32pic32mx.x` 中。请注意，此文件只是默认的链接描述文件的副本。链接器使用的是链接器内部的链接描述文件。

链接描述文件的器件特定部分位于 `\pic32mx\lib\proc\`*`device`*`\procdefs.ld` 中，其中 *`device`* 是指定给 `-mprocessor` 编译驱动程序（xc32-gcc）选项的器件值。

对于PIC32MZ及以后的器件：

适用于PIC32MZ及以后器件的单文件链接描述文件：用于PIC32MZ器件的链接描述文件包含在单个文件（例如，pic32mx/lib/proc/32MZ2048ECH100/p32MZ2048ECH100.ld）中。这消除了对旧版链接描述文件模型所用的两个文件（elf32pic32mx.x 和 procdefs.ld）的依赖。像以前一样，当 xc32-gcc 编译驱动程序使用 -mprocessor=device 选项进行编译时，会将器件特定的链接描述文件传递给链接器。

> **注：** 《MPLAB® XC32 C/C++编译器用户指南》（DS50001686G_CN）中详细讨论了默认链接描述文件的内容。该讨论内容适用于汇编代码项目和 C 代码项目。

默认的链接描述文件将每个标准输入段映射到一个或多个特定的存储区。而每个存储区映射到PIC32 MCU中的地址段（例如kseg0和kseg1）。有关用户/内核地址段的完整说明，请参见《PIC32MX 系列参考手册》的第3章。

下表显示了默认的链接描述文件如何将标准段映射到存储区。

表8-1： 默认链接描述文件中的PIC32保留标准段名

段名	生成者	最终地址	默认的链接描述文件存储区
.reset	复位处理程序	可执行引导代码段	kseg0_boot_mem
.bev_excpt	BEV异常处理程序	可执行引导代码段	kseg0_boot_mem
.app_excpt	通用异常处理程序	可执行引导代码段	kseg0_boot_mem
.vector_n	中断向量n	可执行引导代码段	kseg0_boot_mem
.startup	C启动代码	可执行引导代码段	kseg0_boot_mem
.text	编译器或汇编器生成的指令	可执行代码段	kseg0_program_mem
.rodata	声明为const的字符串和C数据	只读数据段	kseg0_program_mem
.sdata2	小的已初始化全局常量和静态数据	只读数据段	kseg0_program_mem
.sbss2	未初始化的全局常量和静态数据（即始终为零的变量）	只读数据段	kseg0_program_mem
.data	具有初始值且字节数>n（由采用-Gn选项编译确定）的变量。C启动期间从程序存储器复制到数据存储器的值。	初始化数据段	kseg1_data_mem & kseg0_program_mem
.sdata	具有初始值且字节数<=n（由采用-Gn选项编译确定）的变量。用于gp相对寻址。	小的已初始化数据段	kseg1_data_mem & kseg0_program_mem
.lit4 / .lit8	汇编器决定存储在存储器中而不是指令流中的常量（通常为浮点数）。用于gp相对寻址。	小的已初始化数据段	kseg1_data_mem & kseg0_program_mem
.sbss	字节数<=n（由采用-Gn选项编译确定）的未初始化变量。用于gp相对寻址。	小的零填充段	kseg1_data_mem
.bss	未初始化大变量	零填充段	kseg1_data_mem
.heap	动态存储使用的堆	链接描述文件保留的存储区	kseg1_data_mem
.stack	为堆栈保留的最小空间	链接描述文件保留的存储区	kseg1_data_mem
.ramfunc	C启动期间从程序存储器复制到数据存储器的RAM函数	初始化数据段	kseg1_data_mem & kseg0_program_mem
.reginfo .stab* .debug*	调试信息	不在装载映像中	n/a
.line	DWARF调试信息	不在装载映像中	n/a
.comment	#ident/.ident字符串	不在装载映像中	n/a

注：上表包含不再映射到链接描述文件的段。从XC32 v2.00开始，这些段由最佳适应分配器来分配。

© 2014 Microchip Technology Inc.

8.5 添加定制链接描述文件到MPLAB X IDE项目

标准的默认 32 位链接描述文件是通用的并且可满足大多数应用程序的需求。但是，有时可能需要使用定制的链接描述文件。

将默认的链接描述文件（例如，`pic32mx/lib/proc/32MZ2048ECH100/p32MZ2048ECH100.ld`）复制到应用程序的项目目录中。添加这个新的 `*.ld` 文件到项目。它现在应出现在项目树中的“Linker Files”（链接器文件）下。

对新的`*.ld`文件的定制现在应该对项目产生影响。

由于未使用的段不会影响应用程序的存储器利用，因此希望将未使用的段保留在定制链接描述文件中。如果某段必须从定制描述文件中删除，则可以使用 C 样式注释来禁止它。

8.6 链接描述文件命令语言

链接描述文件为包含一系列命令的文本文件。每条命令要么是可能跟有参数的关键字，要么是对符号的赋值。使用分号分隔多条命令。通常可忽略空白，但有些情况下空白是有意义的。例如，运算符周围需要空白。

通常可直接输入文件名或格式名称之类的字符串。如果文件名包含用于分隔各个文件名的字符（如逗号），那么该文件名必须加有双引号。不得在文件名中使用双引号字符。

与C语言一样，也可以包含注释，并由`/*`和`*/`定界。在C语言中，注释在句法上与空白等同。

8.6.1 链接描述文件基本概念

链接器将输入文件组合成一个输出文件。输出文件和每个输入文件都采用了一种特殊的数据格式，称为 ELF 目标文件格式。每个文件都称为目标文件。特别值得一提的是，每个目标文件都具有一个段列表。在输入文件中的段被称为输入段；类似的，在输出文件中的段为输出段。

目标文件中的每个段都具有名称和大小。大多数段还具有一个相关的数据块，称为段内容。可将段标记为可加载，这意味着当输出文件运行时，段的内容会被加载到存储器中。没有任何内容的段也可为可分配的，这意味着应该保留存储区，但是并不装载任何特殊内容（在某些情况下，这些存储区必须清零）。

每个可装载的或可分配的输出段都具有两个地址。第一个为 VMA 或虚拟存储器地址。这是当输出文件运行时，段将具有的地址。第二个为 LMA 或装载存储器地址。这是装载段的地址。在大多数情况下，这两个地址相同。这两个地址也可能不同，例如当一个段要包含 RAM 中的函数（例如默认的 `.ramfunc` 段）时。这种情况下，程序存储器地址为LMA，而数据存储器地址为VMA。

> **注：** VMA和LMA都使用PIC32 MCU的虚拟地址。有关PIC32MX的虚拟存储器到物理存储器的固定存储器映射的说明，请参见《PIC32MX 系列参考手册》。此外，该参考手册还描述了PIC32存储器的构成。

可使用带`-h`选项的`xc32-objdump`程序查看目标文件中的段。

每个目标文件都包含一个符号列表，称为符号表。符号可能已定义也可能未定义。每个符号都具有一个名称，并且每个已定义的符号还具有一个地址和其他信息。如果将一个 C 程序编译成一个目标文件，则将为每个已定义的函数和全局或静态变量创建一个已定义的符号。每个在输入文件中引用的未定义函数或全局变量将成为未定义的符号。

可使用`xc32-nm`程序或带`-t`选项的`xc32-objdump`程序查看目标文件中的符号。

© 2014 Microchip Technology Inc.

8.6.2 处理文件的命令

有几个链接描述文件命令用于处理文件。

INCLUDE filename

在此处包含链接描述文件名。将在当前目录或在使用 -L 选项指定的任意目录中搜索该文件。对INCLUDE的调用最多可嵌套10层。

INPUT(file, file, ...)
INPUT(file file ...)

INPUT 命令指示链接器在链接过程中包含已指定的文件（虽然这些文件是在命令行中指定的）。链接器将首先尝试打开当前目录中的文件。如果未找到，链接器将在归档库搜索路径中搜索该文件。请参见**第7.4.9节“--library-path <dir> (-L <dir>)”**中-L的说明。

如果使用了INPUT (-lfile)，由于添加了命令行参数-l，xc32-ld将把文件名转换为libfile.a。

如果 INPUT 命令出现在隐含的链接描述文件中，那么在链接过程中相应的文件将在包含链接描述文件的地方被包含进来。这样会影响归档搜索。

GROUP(file, file, ...)
GROUP(file file ...)

GROUP命令与INPUT相似，不同之处在于对于GROUP命令，所有的指定文件都必须是归档文件，并且它们将被重复搜索直到没有新创建的未定义引用为止。请参见**第7.4.1节“-(archives -), --start-group archives, --end-group”**中归档的说明。

OPTIONAL(file, file, ...)
OPTIONAL(file file ...)

OPTIONAL命令与INPUT命令类似，不同之处在于OPTIONAL命令无需指定文件即可成功链接。这对于指定可能与编译器一起安装，也可能不与编译器一起安装的归档（或库）特别有用。随XC32编译器提供的默认链接描述文件使用OPTIONAL伪指令来链接器件特定外设库。

OUTPUT(filename)

OUTPUT命令指定输出文件。在链接描述文件中使用OUTPUT(filename)与在命令行中使用-o filename完全相同（见**第7.4.13节“--output file (-o file)”**）。如果两者同时使用，命令行选项优先。

SEARCH_DIR(path)

SEARCH_DIR命令向链接器搜索归档库的路径列表中添加路径。使用SEARCH_DIR(path) 与在命令行中使用 -L path 完全相同（见**第 7.4.9 节“--library-path <dir> (-L <dir>)”**）。如果两者同时使用，链接器将搜索这两个路径。首先搜索的是使用命令行选项指定的路径。

STARTUP(filename)

STARTUP命令与INPUT命令相似，不同之处在于STARTUP指定的文件名将会成为第一个被链接的输入文件，就好像该文件是在命令行中第一个被指定。

8.6.3 给符号赋值

可在链接描述文件中给符号赋值。这样会将符号定义为全局符号。

8.6.3.1 简单赋值

可使用任一C赋值运算符给符号赋值：

```
符号 = 表达式;
符号 += 表达式;
符号 -= 表达式;
符号 *= 表达式;
符号 /= 表达式;
符号 <<= 表达式;
符号 >>= 表达式;
符号 &= 表达式;
符号 |= 表达式;
```

第一种情形，将符号定义为表达式的值。在其他情形中，必须已先定义符号，并对值做相应的调整。

特殊符号名“.”表示位置计数器。该符号只能在SECTIONS命令中使用。

每个表达式后必须跟有分号。

第8.7节“链接描述文件中的表达式”中定义了表达式。

符号赋值可以是一条命令、在SECTIONS命令中作为语句出现或作为SECTIONS命令中输出段说明的一部分。

可通过表达式段对符号段进行设置；更多信息，请参见**第8.7.6节“表达式的段”**。

此处给出了一个可使用符号赋值的三种不同情况示例：

```
floating_point = 0;
SECTIONS
{
  .text ORIGIN(kseg0_program_mem)  :
  {
    _text_begin = . ;
    *(.text .stub .text.* )
    _text_end = . ;
  } >kseg0_program_mem =0
   _bdata = (. + 3) & ~ 3;
  .data : { *(.data) }
}
```

在本例中，符号floating_point将被定义为0。符号_text_end将被定义为上一个.text输入段后的地址。符号 _bdata将被定义为.text输出段后的地址，该地址向上与4字节地址边界对齐。

© 2014 Microchip Technology Inc.

8.6.3.2 PROVIDE

在某些情况下，我们希望链接描述文件只在一个符号被引用时才定义它，并且不由包含在链接中的任何目标文件来定义。例如，传统的链接器定义符号 etext。但是，ANSI C要求将etext用作函数名，而不遇到任何错误。PROVIDE关键字只有在引用未定义的符号时才用来定义符号，比如etext。语法是PROVIDE(符号 = 表达式)

此处给出了使用PROVIDE定义etext的示例：

```
   SECTIONS
{
  .text :
    {
      *(.text)
      _etext = .;
      PROVIDE(etext = .);
    }
}
```

PIC32默认链接描述文件使用PROVIDE命令定义默认的_min_stack_size、_min_heap_size和_vector_spacing符号值。

```
PROVIDE(_min_stack_size = 0x400) ;
PROVIDE(_min_heap_size = 0) ;
PROVIDE(_vector_spacing = 0x00000001);
```

8.6.4 MEMORY命令

链接器的默认配置允许分配所有的可用存储器。使用MEMORY命令可改写该配置。

MEMORY命令说明目标器件中存储块的位置和大小。该命令可被用来说明链接器可使用哪些存储区以及必须避免使用哪些存储区。之后即可将段分配到特定的存储区。链接器将根据存储区设置段地址并对过满的区域发出警告。链接器不会移动段以使之适合可用的存储区。

MEMORY命令的语法为：

```
  MEMORY
{
  name [(attr)] : ORIGIN = origin, LENGTH = len
  ...
}
```

name 就是在链接描述文件中用来指代存储区的名称。存储区名在链接描述文件外毫无意义。存储区名被存储在单独的名称空间中，并且不与符号名、文件名或段名相冲突。每个存储区的名称必须互不相同。

*attr*字符串只能由以下字符组成：

R	只读段
W	读/写段
X	可执行段
A	可分配段
I	已初始化段
L	与I相同
!	将上述任一属性的意义取反

如果一个未映射段与除！以外的任何一个列出的属性相符，它就将被放入存储区中。！属性对测试取反，因此只有未映射的段与任何一个所列出的属性不符时，才会被放入存储区中。

origin 是存储区起始地址的表达式。在执行存储区分配前，必须将该表达式求值为一个常量，这意味着不能使用段相对符号。关键字ORIGIN可被缩写为org或o（但不能写成例如ORG的形式）。

len是以字节表示的存储区大小的表达式。和origin表达式一样，在执行存储区分配之前也必须将该表达式求值为一个常量。关键字LENGTH可被缩写为len或l。

在下例中，我们指定了两个用于分配的存储区：一个从地址0开始，大小为48 KB；另一个从地址0x800开始，大小为2 KB。链接器会将每个没有明确映射到存储区且为只读或可执行的段放入rom存储区，而将其他没有明确映射到存储区的段放入ram存储区。

```
  MEMORY
{
  rom (rx)  : ORIGIN = 0, LENGTH = 48K
  ram (!rx) : org = 0x800, l = 2K
}
```


© 2014 Microchip Technology Inc.

一旦定义了存储区，可指示链接器通过使用 `>region` 输出段属性将特定的输出段放入该存储区。例如，在输出段定义中使用 `>mem` 可指定名为 `mem` 的存储区。如果没有为输出段指定任何地址，链接器则会将地址设置为存储区内的下一个可用地址。如果要存入存储区的组合输出段对于该存储区来说太大了，链接器将发出一条错误消息。

8.6.5 SECTIONS命令

SECTIONS命令告知链接器如何将输入段映射到输出段以及如何在存储器中放置输出段。

SECTIONS命令的格式为：

```
   SECTIONS
{
  sections-command
  sections-command
  ...
}
```

每个SECTIONS命令可为以下形式之一：

- 一条ENTRY命令（见**第8.6.6节“其他链接描述文件命令”**）
- 符号赋值（见**第8.6.3节“给符号赋值”**）
- 一个输出段说明
- 覆盖说明

为了便于在SECTIONS命令中使用位置计数器，允许在这些命令中使用ENTRY命令和符号赋值。由于这些命令可在输出文件中有意义的位置使用，因此还可使链接描述文件便于理解。

输出段说明和覆盖说明将在下面进行描述。

如果链接描述文件中没有出现SECTIONS命令，那么链接器将以每个输入段在输入文件中首先出现的顺序将这些段放入与它们同名的输出段中。例如，如果所有的输入段都在第一个文件中，那么在输出文件中这些段的顺序将与在第一个输入文件中这些段的顺序相同。第一个段位于地址0处。

8.6.5.1 输入段说明

最常见的输出段命令是输入段说明。

输入段说明是最基本的链接描述文件操作。输出段告知链接器如何在存储器中放置程序。输入段说明告知链接器如何将输入文件映射到存储器布局中。

输入段说明包括一个文件名，后面跟有一系列可选段名（用括号括起）。

文件名和段名可以是通配符模式的，这一点将在下面进一步说明。

最常见的输入段说明是包含在输出段中具有特定名称的所有输入段。例如，要包含所有的.text输入段，可以这样写：

```
*(.text)
```

这里*为通配符，可与任何文件名匹配。要排除一系列与文件名通配符相符的文件，可以使用EXCLUDE_FILE以便与除在EXCLUDE_FILE列表中指定的文件之外的所有文件匹配。例如：

```
*(EXCLUDE_FILE (*crtend.o *otherfile.o) .ctors)
```

将导致包含除crtend.o和otherfile.o文件外的所有文件中的所有.ctors段。

有两种方法可以包含多个段：

```
*(.text .rodata)
*(.text) *(.rodata)
```


© 2014 Microchip Technology Inc.

上述两者的区别在于 .text 和 .rodata 输入段在输出段中出现的顺序不同。在第一个示例中，它们是混合出现的。而在第二个示例中，所有的 .text 输入段都将首先出现，然后才是所有的 .rodata 输入段。

可以指定文件名以包含特定文件中的段。如果某个文件包含特殊数据并需将此数据放在存储器内特定单元时，这个选项很有用。例如：

```
data.o(.data)
```

如果指定的文件名不带有段列表，那么输入文件中的所有段都将被包含到输出段中。这种操作并不常见，但有时会很有用。例如：

```
data.o
```

当指定了一个不含有任何通配符的文件名时，链接器将首先查看是否在链接器命令行或在 INPUT 命令中指定了同样的文件名。如果没有，链接器会尝试将该文件打开为输入文件，就好像文件出现在命令行中一样。由于链接器不会在归档搜索路径中搜索该文件，因此该操作与 INPUT 命令不同。

8.6.5.2 输入段通配符模式

在输入段说明中，文件名、段名或这两者都可以采用通配符模式。

在许多示例中出现的 * 文件名是文件名的简单通配符模式。

这些通配符模式与 UNIX shell 使用的通配符模式相类似。

*	与任意数量的字符匹配
?	与任意单个字符匹配
[*chars*]	与任何 *chars* 的单个实例相匹配；- 字符可被用来指定一个字符范围，如 [a-z] 与任何小写字母相匹配
\	引用以下字符

当文件名与某个通配符匹配时，通配符不与 / 字符（用于在 UNIX 中分隔目录名）匹配。只由一个 * 字符组成的模式是个例外；它总是与任何文件名匹配，而无论该文件名是否含有 /。在段名中通配符将与 / 字符匹配。

文件名通配符模式只与那些在命令行中或在 INPUT 命令中明确指定的文件匹配。链接器不会搜索目录以扩展通配符的匹配范围。

如果某个文件名与多种通配符模式匹配，或者如果一个文件名明确出现，并且它也与一种通配符模式匹配，链接器将在链接描述文件中使用第一个匹配的文件名。例如，这个输入段说明序列可能存在错误，下面的 data.o 规则将不会被使用：

```
.data  : { *(.data) }
.data1 : { data.o(.data) }
```

通常，链接器会以文件和段在链接过程中出现的顺序放置这些与通配符匹配的文件和段。可使用将通配符模式用圆括号括起并在前面加上 SORT 关键字（例如，SORT(.text*)）更改文件和段的排序。当使用 SORT 关键字时，在将文件或段放入输出文件之前，链接器会将文件或段以名称的升序排列。

要验证输入段的放置位置，请使用 -M 链接器选项来生成一个映射文件。映射文件精确地给出了输入段映射到输出段的方式。

本示例给出了如何使用通配符模式来分配文件。链接描述文件指示链接器将所有的.text段放入.text，而将所有的.bss段放入.bss。链接器会将所有以大写字母开头的文件中的.data段放入.DATA；而对于所有其他文件，链接器会将其中的.data段放入.data。

```
   SECTIONS {
  .text : { *(.text) }
  .DATA : { [A-Z]*(.data) }
  .data : { *(.data) }
  .bss : { *(.bss) }
}
```

8.6.5.3 共用符号的输入段

共用符号需要一个特殊的记号（notation），因为共用符号不具有特定的输入段。链接器将共用符号视为如同位于名为COMMON的输入段中的符号。

文件名可与COMMON段一起使用，就像任何其他输入段一样。链接器会将来自特定输入文件的共用符号放入某个段，而将来自其他输入文件的共用符号放入另一个段中。

在大多数情况下，输入文件中的共用符号将被放入输出文件中的.bss段中。例如：

```
   .bss { *(.bss) *(COMMON) }
```

如果没有另行指定，共用符号将被分配到.bss段中。

8.6.5.4 输入段示例

下例为一个完整的链接描述文件。它告知链接器从文件 all.o 中读出所有的段，并将这些段放入起始地址为 0x10000 的输出段 outputa 的开头。来自文件 foo.o 的整个.input1段在同一个输出段中紧跟其后。来自foo.o的整个.input2段被放入输出段outputb中，后跟来自foo1.o的.input1段。所有其余的来自任意文件的.input1和.input2段将被写入输出段outputc。

```
SECTIONS {
  outputa 0x10000 :
    {
    all.o
    foo.o (.input1)
    }
  outputb :
    {
    foo.o (.input2)
    foo1.o (.input1)
    }
  outputc :
    {
    *(.input1)
    *(.input2)
    }
}
```


© 2014 Microchip Technology Inc.

8.6.5.5 输出段说明

输出段的完整描述类似于：

```
   name [address] [(type)] : [AT(lma)]
  {
    output-section-command
    output-section-command
    ...
  } [>region] [AT>lma_region] [=fillexp]
```

大多数输出段仅使用少部分可选段属性。

需要在*name*和*address*之间留有空白。还需要冒号和花括号。可以有选择地换行和添加其他空白。

段名可以由任何字符序列组成，但是包含有特殊字符（如逗号）的段名必须添加引号。

每个输出段命令可为下列形式之一：

- 符号赋值（见**第8.6.3节“给符号赋值”**）
- 输入段说明（见**第8.6.5.1节“输入段说明”**）
- 要直接包含的数据值（见**第8.6.5.7节“输出段数据”**）

8.6.5.6 输出段地址

address 为输出段的VMA（虚拟存储器地址）表达式。如果未提供地址，链接器将根据存储区（如果存在）设置一个，否则，如果不存在存储区，将根据位置计数器的当前值设置地址。

如果提供了 *address*，输出段的地址将被精确地设置为该地址。如果既没有提供 *address* 也没有提供 *region*，那么输出段的地址将被设置为按照输出段对齐要求对齐的位置计数器当前值。输出段对齐要求是在输出段内包含的所有输入段的最严格对齐方式。

例如：

```
.text . : { *(.text) }
```

和

```
.text : { *(.text) }
```

稍有不同。第一种将 `.text` 输出段的地址设置为位置计数器的当前值。而第二种将该地址设置为按照`.text`输入段的最严格对齐方式对齐的位置计数器当前值。

该地址可以是任意表达式（见**第8.7节“链接描述文件中的表达式”**）。例如，要将段与`0x10`字节边界对齐，以使段地址的低`4`位为`0`，可用类似于下面的命令：

```
.text ALIGN(0x10) : { *(.text) }
```

该操作的原理是`ALIGN`返回向上与指定值对齐的位置计数器当前值。

指定段*address*将改变位置计数器的值。

8.6.5.7 输出段数据

可通过使用BYTE、SHORT、LONG或QUAD作为输出段命令在输出段中插入指定字节的数据。每个关键字都跟有一个用圆括号括起的表达式，用来提供要存储的值。表达式的值存储在位置计数器的当前值中。

BYTE、SHORT、LONG和QUAD命令可分别存储一个、两个、四个和八个字节。例如，该命令将存储符号addr的4字节值：

```
LONG(addr)
```

存储完字节后，位置计数器将以所存储的字节数递增。当在程序存储器段中使用数据命令时，必须注意链接器会认为程序存储器为 32 位宽，尽管程序存储器只物理实现了24位，这是很重要的。因此，LONG数据值的高8位不会被装载到器件存储器中。

只能在一个段说明中使用数据命令，但不能在段说明之间使用，因此以下段将导致链接器产生一个错误：

```
SECTIONS { .text : { *(.text) } LONG(1) .data : { *(.data) } }
```

但下面这段可行：

```
SECTIONS { .text : { *(.text) ; LONG(1) } .data : { *(.data) } }
```

FILL 命令可被用来设置当前段的填充模式。该命令后跟带有圆括号的表达式。将使用该表达式的两个最低字节填充段中任何其他没有指定的存储区（例如，由于所要求的输入段对齐方式而留下的空隙），必要时可重复填充。FILL 语句将覆盖它在段定义中出现的位置之后的存储单元；通过包含多条FILL语句，可对输出段的不同部分使用不同的填充模式。

本例给出如何使用值0x9090填充存储器中未定义区域的方法：

```
FILL(0x9090)
```

FILL命令与=fillexp输出段属性类似（见**第8.6.5.9节“输出段属性”**），但是它只影响段中FILL命令后的部分而不是整个段。如果两者同时使用，FILL命令优先。

8.6.5.8 丢弃输出段

链接器不会创建没有任何内容的输出段。这样便于引用任何输入文件中可能存在也可能不存在的输入段。例如：

```
.foo { *(.foo) }
```

仅当至少一个输入文件中包含.foo段时，才会在输出文件中创建一个.foo段。

如果将除输入段说明以外的任何其他项用作输出段命令，如符号赋值，那么将始终创建输出段，即便没有匹配的输入段也是如此。

可使用特殊输出段名/DISCARD/来丢弃输入段。输出文件中不包含任何分配给名为/DISCARD/的输出段的输入段。

© 2014 Microchip Technology Inc.

8.6.5.9 输出段属性

回顾一下，输出段的完整描述如下：

```
name [address] [(type)] : [AT(lma)]
  {
    output-section-command
    output-section-command
    ...
  } [>region] [AT>lma_region] [:phdr :phdr ...] [=fillexp]
```

已经对*name*、*address*和output-section-command进行了说明。在以下部分，将对其余的段属性进行说明。

8.6.5.10 输出段类型

每个输出段都有一个类型。类型为在圆括号中的关键字。定义了以下类型：

NOLOAD

该段应该被标记为不可装载，因此当运行程序时，它将不被装载到存储器中。

DSECT, COPY, INFO, OVERLAY

这些类型的名称支持与旧版MIPS和GNU汇编器的向后兼容，很少被使用。它们的效果相同：将段标记为不可分配，因此当程序运行时，不会为该段分配存储区。

链接器通常根据映射到输出段中的输入段设置输出段的属性。通过使用段类型可改写该配置。例如，在下面的描述文件示例中，ROM段位于存储单元0中，并且当程序运行时，它不需要被装载。通常，ROM段的内容将出现在链接器输出文件中。

```
SECTIONS {
  ROM 0 (NOLOAD) : { ... }
  ...
}
```

8.6.5.11 输出段LMA

每个段都具有一个虚拟地址（VMA）和一个装载地址（LMA）。在输出段说明中的地址表达式设置VMA。

链接器通常会将LMA设置得与VMA相同。通过使用AT关键字可改变该设置。AT关键字后的表达式lma指定段的装载地址。或者，使用AT>lma_region表达式，指定段的装载地址所在的存储区。请参见**第8.6.4节“MEMORY命令”**。

该特性可使建立ROM映像变得简便。例如，以下链接描述文件创建了三个输出段：一个称为.text，地址从0xBFC00000处开始；一个称为.mdata，它被装载到.text段的尾部，即便其VMA位于0xA0000000也是如此；还有一个称为.bss，用于将未初始化数据保存在地址0xA0001000处。使用值0xA0000000定义符号_data，从而使位置计数器保存VMA值，而不是LMA值。

```
SECTIONS
  {
  .text 0xBFC00000: { *(.text) _etext = . ; }
  .mdata 0xA0000000:
    AT ( ADDR (.text) + SIZEOF (.text) )
    { _data = . ; *(.data); _edata = . ;  }
  .bss 0xA0001000:
    { _bstart = . ;  *(.bss) *(COMMON) ; _bend = . ;}
}
```

由链接描述文件生成的用于程序的运行时初始化代码包含一个函数，该函数将来自ROM 映像的已初始化数据复制到运行时地址。初始化函数将利用由链接描述文件定义的符号。

但很少需要编写这样的函数。这些函数由 C 编译器的启动代码和初始化代码提供。更多关于随编译器提供的启动代码的信息，请参见《MPLAB® XC32 C/C++ 编译器用户指南》（DS50001686G_CN）。启动程序的汇编源代码在\pic32-libs\c\startup\crt0.S中提供。

8.6.5.12 输出段存储区

可以将某个段分配到先前使用>region定义的存储区中。请参见**第8.6.4节“MEMORY命令”**。

这里有一个简单的示例：

```
MEMORY { rom : ORIGIN = 0x1000, LENGTH = 0x1000 }
SECTIONS { ROM : { *(.text) } >rom }
```

8.6.5.13 填充输出段

可通过使用 =fillexp 设置整个段的填充模式。fillexp 为一个表达式。将使用该值的两个最低字节填充输出段中任何其他未指定的存储区（例如，由于所要求的输入段对齐方式而留下的空隙），有必要的话可重复填充。

可使用输出段命令中的FILL命令更改填充值；请参见**第8.6.5.7节“输出段数据”**。

这里有一个简单的示例：

```
SECTIONS { .text : { *(.text) } =0x9090 }
```


© 2014 Microchip Technology Inc.

8.6.5.14 覆盖说明

覆盖说明提供了一种简便的方法，来说明将要作为一个存储区映像的一部分装载而在相同的存储器地址运行的段。运行时，某类覆盖管理器将根据需要复制运行时存储器地址内外的覆盖段，这可能仅通过操作寻址位实现。

使用OVERLAY命令说明覆盖操作。和输出段说明一样，OVERLAY命令也在SECTIONS命令中使用。OVERLAY命令的完整语法如下：

```
 OVERLAY [start] : [NOCROSSREFS] [AT ( ldaddr )]
{
  secname1
    {
      output-section-command
      output-section-command
      ...
    } [:phdr...] [=fill]
  secname2
    {
      output-section-command
      output-section-command
      ...
    } [:phdr...] [=fill]
  ...
} [>region] [:phdr...] [=fill]
```

除了 OVERLAY（关键字）以外，所有项都是可选的，并且每个段都必须有一个名称（上面的secname1和secname2）。OVERLAY结构中的段定义与在常规SECTIONS结构中的相同，不同之处在于在OVERLAY中可不为段定义任何地址和存储区。

这些段都被定义为具有相同的起始地址。总体上，段的装载地址的设置要使段在存储器中连续排列，以 OVERLAY 的装载地址为起始地址（对于常规段定义，装载地址是可选的，并且默认为起始地址；起始地址也是可选的，并且默认为位置计数器的当前值）。

如果使用了NOCROSSREFS关键字，并且有段之间的相互引用，链接器将报错。由于段都在同一个地址运行，通常一个段直接引用另一个段是没有意义的。

对于在OVERLAY中的每个段，链接器将自动定义两个符号。符号__load_start_secname定义为段的起始装载地址。符号__load_stop_secname定义为段的末尾装载地址。在C标识符中段名中任何不合法的字符都将被删除。C（或汇编）代码可使用这些符号根据需要将覆盖的段转移。

在覆盖段的末尾，位置计数器的值被设置为覆盖段的起始地址加上最大段的大小。

下面给出了一个示例。切记它将会出现在SECTIONS结构内部。

```
OVERLAY 0x9D001000 : AT (0xA0004000)
 {
   .text0 { o1/*.o(.text) }
   .text1 { o2/*.o(.text) }
 }
```

上述语句会将.text0和.text1的起始地址都定义为0x9D001000。.text0被装载到地址0x9D001000，而.text1将被装载到紧跟.text0之后。将定义以下符号：__load_start_text0、__load_stop_text0、__load_start_text1和__load_stop_text1。

将覆盖段.text1复制到覆盖区的C代码如下：

```
extern char __load_start_text1, __load_stop_text1;
memcpy ((char *) 0x9D001000, &__load_start_text1,
        &__load_stop_text1 - &__load_start_text1);
```

OVERLAY 命令只是一种便捷方式，它所能完成的所有操作都可使用更基本的命令实现。上述示例可以写为如下的等效形式。

```
.text0 0x9D001000: AT (0x9D004000) { o1/*.o(.text) }
__load_start_text0 = LOADADDR (.text0);
__load_stop_text0 = LOADADDR (.text0) + SIZEOF (.text0);
.text1 0x9D001000: AT(0x9D004000+SIZEOF(.text0))
{o2/*.o(.text) }
__load_start_text1 = LOADADDR (.text1);
__load_stop_text1 = LOADADDR (.text1) + SIZEOF (.text1);
. = 0x9D001000+ MAX (SIZEOF (.text0), SIZEOF (.text1));
```

8.6.6 其他链接描述文件命令

还有几个其他的链接描述文件命令，将对它们做简要说明：

ASSERT(*exp*, *message*)

确保*exp*非零。如果为零，将退出链接器并显示错误代码，同时显示错误消息*message*。

ENTRY(*symbol*)

将*symbol*指定为要在程序中执行的第一条指令。链接器将在输出目标文件标题中记录该符号的地址。该命令不会影响位于地址0处的Reset指令，该指令必须以其他方式生成。按照约定，32位链接描述文件将GOTO __reset指令定位到地址0处。

EXTERN(*symbol symbol* ...)

强制将*symbol*作为未定义的符号输入到输出文件。例如，这样做可触发标准库中其他模块的链接。可为每个EXTERN列出几个符号，并且EXTERN可出现多次。该命令与-u命令行选项等效。

FORCE_COMMON_ALLOCATION

该命令与-d命令行选项等效：即使已指定了可重定位输出文件（-r），该命令仍可使32位链接器将空间分配给共用符号。

NOCROSSREFS(*section section* ...)

可使用该命令告知 32 位链接器发出一条有关在某些输出段之间相互引用的错误。在某些类型的程序中，一旦向存储区中装载了一个段，另一个段将不被装载。两个段之间的直接引用将导致错误。

NOCROSSREFS命令带有一系列输出段名。如果链接器检测到段之间的交叉引用，它将报错并返回一个非零的退出状态。NOCROSSREFS命令使用输出段名而不是输入段名。

© 2014 Microchip Technology Inc.

OUTPUT_ARCH(*bfdarch*)

指定具体的输出器件架构。对于Microchip PIC32 MCU，bfdarch值始终为pic32mx。

OUTPUT_FORMAT(*format_name*)

OUTPUT_FORMAT命令指定供输出文件使用的目标文件格式。对于Microchip PIC32 MCU，format_name值始终为elf32-tradlittlemips。

TARGET(*format_name*)

TARGET命令指定在读取输入文件时使用的目标文件格式。它影响后续的INPUT和GROUP命令。对于Microchip PIC32 MCU，*format_name*值应保持为elf32-tradlittlemips。

8.7 链接描述文件中的表达式

在链接描述文件语言中的表达式语法与C表达式的语法相同。所有表达式都被求值为32位整型数据。

可在表达式中使用和设置符号值。

链接器定义了几个特殊用途的内建函数，供在表达式中使用。

8.7.1 常量

所有常量都是整数。

在C语言中，链接器将以0开头的整数视为八进制数，而将以0x或0X开头的整数视为十六进制数。链接器将其他整数视为十进制数。

此外，您可以使用后缀K和M以将常量分别扩大为1024倍或1024*1024倍。例如，以下各项均指的是同一个数：

```
_fourk_1 = 4K;
_fourk_2 = 4096;
_fourk_3 = 0x1000;
```

8.7.2 符号名

除非加有引号，否则符号名称必须以字母、下划线或句点开始，并且可包含字母、数字、下划线、句点和连字符。未加引号的符号名不得与任何关键字相冲突。您可以通过为符号名加上双引号来指定包含特殊字符或与关键字同名的符号。

```
"SECTION" = 9;
"with a space" = "also with a space" + 10;
```

由于符号可包含许多非字母字符，将符号用空格分开是最安全的。例如，A-B为一个符号，而A - B为一个减法表达式。

© 2014 Microchip Technology Inc.

8.7.3 位置计数器

特殊链接器变量点“.”始终包含输出位置计数器的当前值。由于 . 始终指向输出段中的位置，它只能出现在SECTIONS命令中的表达式中。“.”符号可以出现在任何允许使用普通符号的表达式中。

给“.”赋值将导致位置计数器的值被更改。可使用该变量在输出段中创建空隙。位置计数器不能往回移动。

```
SECTIONS
{
  output :
    {
      file1(.text)
      . = . + 1000;
      file2(.text)
      . += 1000;
      file3(.text)
    } = 0x1234;
}
```

在前面的示例中，file1 中的 .text 段将被定位到输出段 output 的开头。后面跟有1000个字节的空隙。然后是来自file2的.text段，同样跟有1000个字节的空隙，之后是来自file3的.text段。记号= 0x1234指定将向空隙写入什么数据。

“.”实际上指的是相对于包含目标文件的当前段起始地址的字节偏移量。通常，它是SECTIONS 语句，其起始地址为 0，因此“.”可被用作绝对地址。当在段说明内部使用“.”时，“.”指的是相对于该段起始地址的字节偏移量，而不是绝对地址。因此在下面的描述文件中：

```
SECTIONS
{
    . = 0x100
    .text: {
      *(.text)
      . = 0x200
    }
    . = 0x500
    .data: {
      *(.data)
      . += 0x600
    }
}
```

为 .text 段指定起始地址0x100，并分配正好为0x200个字节的长度，即使在 .text 输入段中没有足够的数据填充该区域时也是如此 （如果段中的数据过多，由于操作将尝试使“.”往回移动，因此会产生一条错误）。.data段将起始于地址0x500，并在 .data 输入段的值末尾之后和 .data 输出段本身的末尾之前添加额外的0x600个字节的空间。

8.7.4 运算符

链接器可识别具有标准绑定和优先级的标准C算术运算符集：

表8-2： 运算符的优先级

优先级	结合方式	运算符	说明
1（最高）	左	! - ~	前缀运算符
2	左	* / %	乘、除和模
3	左	+ -	加和减
4	左	>> <<	位右移和位左移
5	左	== != > < <= >=	关系运算符
6	左	&	按位与
7	左	|	按位或
8	左	&&	逻辑与
9	左	||	逻辑或
10	右	? :	条件
11（最低）	右	&= += -= *= /=	符号赋值

8.7.5 求值

链接器不会主动对表达式求值。只有绝对必需时，它才计算表达式的值。

链接器需要第一个段的起始地址值和存储区的起始地址和长度等信息才能进行链接。链接器一读链接描述文件就将计算这些值。

但是，其他值（如符号值）都是未知的或在存储区分配完毕之后才需要。当供符号赋值表达式使用的其他信息（如输出段的大小）可用时，才会计算这些值。

在分配完成之前段的大小都是未知的，因此在这之前，取决于段大小的赋值操作将不会执行。

某些表达式，如那些依靠位置计数器“.”的表达式，必须在段分配的过程中求值。

如果需要表达式的结果，但该值不可用，将导致错误。例如，类似于以下的描述文件：

```
SECTIONS
  {
    .text 9+this_isnt_constant :
      { *(.text) }
  }
```

将产生错误消息“non-constant expression for initial address”（初始地址的表达式的值不是常量）。

© 2014 Microchip Technology Inc.

8.7.6 表达式的段

当链接器对表达式求值时，结果可以是绝对的或相对于某个段的值。相对表达式表示为相对于段基地址的固定偏移量。

表达式在链接描述文件中的位置决定它是绝对的还是相对的。出现在输出段定义中的表达式是相对于输出段的基地址。出现在任何其他位置的表达式都是绝对的。

如果使用 `-r` 选项要求可重定位输出，那么设置给相对表达式的符号将是可重定位的。这意味着进一步的链接操作可能更改该符号的值。符号所在的段将是相对表达式的段。

设置给绝对表达式的符号的值在任何进一步的链接操作中都将保持不变。符号将是绝对的，并且没有与之相关的特定段。

可以使用内建函数`ABSOLUTE`将本来为相对的表达式强制为绝对表达式。例如，创建一个绝对符号并使其等于输出段`.data`的末尾地址：

```
SECTIONS
  {
    .data : { *(.data) _edata = ABSOLUTE(.); }
  }
```

如果未使用`ABSOLUTE`，`_edata`将相对于`.data`段。

8.7.7 内建函数

链接描述文件语言包含大量可在链接描述文件表达式中使用的内建函数。

8.7.7.1 ABSOLUTE(*EXP)*

返回表达式 *exp* 的绝对（不可重定位的，相对于非负的）值。此内建函数主要用于在段定义中给符号赋绝对值，其中符号值在正常情况下是相对于段的。

8.7.7.2 ADDR(*SECTION*)

返回指定段的绝对地址（VMA）。链接描述文件必须预先定义了该段的位置。在下例中，`symbol_1`和`symbol_2`被赋以相同的值：

```
SECTIONS { ...
  .output1 :
    {
    start_of_output_1 = ABSOLUTE(.);
    ...
    }
  .output :
    {
    symbol_1 = ADDR(.output1);
    symbol_2 = start_of_output_1;
    }
...
}
```

8.7.7.3 ALIGN(*EXP*)

返回与下一个表达式边界对齐的位置计数器值（.）。*exp* 必须为值为 2 的次幂的表达式。它等效于

```
(. + exp - 1) & ~(exp - 1)
```

ALIGN 不会改变位置计数器的值，它仅对该值进行算术运算。下面给出了一个示例，该示例将输出.data段与前一个段后的下一个0x2000字节边界对齐，并将该段内的变量设置到输入段之后的下一个0x8000边界。

```
SECTIONS { ...
  .data ALIGN(0x2000): {
    *(.data)
    variable = ALIGN(0x8000);
  }
...
}
```

由于本例中第一次使用的 ALIGN 被用作段定义中的可选地址属性，因此它指定段的位置（见**第8.6.5节“SECTIONS命令”**）。第二次使用的ALIGN被用来定义符号的值。

内建函数NEXT与ALIGN紧密相关。

8.7.7.4 BLOCK(*EXP*)

该函数与ALIGN同义，用于与旧的链接描述文件兼容。在设置输出段地址时经常使用它。

8.7.7.5 DEFINED(*SYMBOL*)

如果符号位于链接器全局符号表中并且已被定义则返回1；否则返回0。您可使用该函数为符号提供默认值。例如，下面的描述文件片断给出了将全局符号 begin 设置到 .text 段中第一个存储单元的方法，但是如果名为 begin 的符号已经存在，将保存该符号的值。

```
SECTIONS{ ...
  .text : {
    begin = DEFINED(begin) ? begin : . ;
    ...
  }
  ...
}
```

8.7.7.6 KEEP(*SECTION*)

使用链接时垃圾收集（--gc-sections）时，标记不应删除的段通常非常有用。可通过用KEEP()括起输入段的通配条目（如在KEEP(*(.init))或KEEP(SORT_BY_NAME(*)(.ctors))中）的方式来完成此操作。

8.7.7.7 LOADADDR(*SECTION*)

返回指定段的绝对LMA。它通常与ADDR相同，但是如果在输出段定义中使用了AT属性，则将有所不同（见**第8.6.5节“SECTIONS命令”**）。

8.7.7.8 MAX(*EXP1*, *EXP2*)

返回*exp1*和*exp2*中的最大值。

© 2014 Microchip Technology Inc.

8.7.7.9 MIN(*EXP1*, *EXP2*)

返回*exp1*和*exp2*中的最小值。

8.7.7.10 NEXT(*EXP*)

返回下一个未分配的地址，该地址为*exp*的倍数。该函数与*ALIGN(exp)*等效。

8.7.7.11 SIZEOF(*SECTION*)

返回以字节数表示的指定段的大小（如果段已被分配的话）。如果在计算段大小时，段尚未分配，链接器将报告一个错误。在下例中，symbol_1和symbol_2被赋以相同的值：

```
SECTIONS{ ...
  .output {
    .start = . ;
    ...
    .end = . ;
    }
  symbol_1 = .end - .start ;
  symbol_2 = SIZEOF(.output);
...
}
```

注：

© 2014 Microchip Technology Inc.

第9章 链接器处理

9.1 简介

本章将讨论MPLAB XC32目标链接器（xc32-ld）从输入文件编译应用程序的方法。

本章涵盖以下主题：

- 链接器处理概述
- 链接器分配
- 全局符号和弱符号
- 已初始化数据
- 堆栈分配
- 堆分配
- PIC32MX中断向量表
- 具有专用可编程可变偏移量的PIC32 MCU中断向量表

9.2 链接器处理概述

链接器将一个或多个目标文件与可选的归档文件组合到一个可执行输出文件中。目标文件包含可重定位代码段和数据段，这些段将被链接器分配到目标存储器中。整个过程由链接描述文件（也被称为链接命令文件）控制。每次链接都需要链接描述文件。

链接过程可以分解为5个步骤：

1. 加载输入文件
2. 分配存储区
3. 解析符号
4. 计算绝对地址
5. 编译输出文件

9.2.1 加载输入文件

链接器的首要任务就是解析链接命令选项并加载输入文件。如果指定了链接描述文件，则将打开并解析该文件。否则将使用内部默认的链接描述文件。在这两种情况下，链接描述文件均提供目标器件的说明，包括特定的存储区信息。更多信息，请参见**第8章“链接描述文件”**。

接下来，链接器将打开所有的输入目标文件。检查每个输入文件以确保目标格式是兼容的。如果目标格式不兼容，就会产生错误。然后每个输入文件的内容都被装入内部数据结构。通常每个输入文件都包含多个代码段或数据段。每个段均包含一系列重定位项，这些项将段的原数据中的地址与可重定位符号相关联。

9.2.2 分配存储区

在加载了所有输入文件之后，链接器将分配存储区。这通过将每个输入段分配给一个输出段来完成。输入段和输出段之间的关系由链接描述文件中的段映射来定义。输出段和输入段的名称可能相同也可能不同。然后每个输出段将被分配到目标器件中的存储区中。

注：	输入段是通过编译器或汇编器由源代码产生的。输出段是由链接器创建的。

如果没有明确地将输入段分配给输出段，链接器将根据段属性来分配未分配的段。更多有关链接器分配的信息，请参见**第9.3节“链接器分配”**。

9.2.3 解析符号

一旦分配了存储区，链接器就开始解析符号的过程。在每个输入段中定义的符号都具有相对于该段起始地址的偏移量。链接器将这些值转换为输出段偏移量。

接下来，链接器尝试将所有外部符号引用和相应的符号定义进行匹配。对同一个外部符号进行多次定义将导致错误。如果未找到外部符号，将尝试在归档文件中查找该符号的定义。如果在归档中找到了该符号的定义，就会载入相应的归档模块。

从归档中载入的模块可能包含其他符号引用，因此处理过程将继续到所有的外部符号引用都与其定义相匹配为止。被定义为“弱”的外部符号将得到特殊处理，如**第9.4节“全局符号和弱符号”**所述。如果仍有外部符号引用尚未定义，就会产生错误。

© 2014 Microchip Technology Inc.

9.2.4 创建特殊段

符号解析完成后，链接器会构造所需的任何特殊输入段或输出段。例如，链接器构造一个名为`.dinit`的特殊输入段来支持已初始化的数据。段.dinit是由C运行时库解释的初始化模板。更多有关已初始化数据的信息，请参见**第9.5节“已初始化数据”**。

9.2.5 计算绝对地址

在创建了特殊段之后，所有输出段的最终大小就已知了。然后链接器会计算所有输出段和外部符号的绝对地址。检查每个输出段以确保它位于所分配的存储区内。如果有任何段位于它的存储区之外，就会产生错误。在链接描述文件中定义的所有符号也会被计算。

9.2.6 编译输出文件

最后，链接器将编译输出文件。将使用绝对地址修改每个段中的重定位项。如果为符号计算的地址与重定位项不符，会导致链接错误。例如，当一个模块引用了它认为在“小数据”段中而其他模块将其定义在非小数据段中的变量，就可能发生这种情况。

也可以使用适当的选项请求生成链接映射。链接映射包括存储器使用报告，此报告显示了数据存储器和程序存储器中所有段的起始地址和长度。更多有关链接映射的信息，请参见**第6.4.5节“映射文件”**。

9.3 链接器分配

链接器分配由链接描述文件控制，并且分为以下三个步骤来进行：

1. 将输入段映射到输出段
2. 将输出段分配到存储区
3. 分配未映射段

步骤1和步骤2通过连续存储器分配器执行。链接描述文件中出现的输入段分配给目标器件中的特定存储区。存储区中的地址是按顺序分配的，从最低地址开始，向高地址延伸。

步骤3通过最佳适应储器分配器执行。未出现在链接描述文件中的输入段按属性分配给存储区。最佳适应分配器会有效利用任何剩余的存储器（包括可能由连续分配器在输出段之间遗留的空隙）。

9.3.1 将输入段映射到输出段

根据段映射，输入段将被组合并映射到输出段。当输出段包含几个不同的输入段时，输入段的确切顺序可能会很重要。例如，以下输出段定义：

```
/* Code Sections */
.text ORIGIN(kseg0_program_mem)  :
{
  *(.text .stub .text.* .gnu.linkonce.t.*)
  *(.mips16.fn.*)
  *(.mips16.call.*)
} >kseg0_program_mem =0
```

这里定义了名为.text的输出段。注意此段的内容是用花括号{}指定的。右括号之后的>kseg0_program_mem表示应该将此输出段分配到存储区kseg0_program_mem。

可以将输出段.text的内容解析如下：

- 收集名为.text和.stub的输入段以及与通配符模式.text.*和.gnu.linkonce.t.*匹配的输入段并将其映射到输出段。将这些段组合在一起以确保引用的局部性。
- 收集与通配符模式.mips16.fn.*相匹配的输入段并将它们映射到输出段。
- 收集与通配符模式.mips16.call.*相匹配的输入段并将它们映射到输出段。

© 2014 Microchip Technology Inc.

9.3.2 将输出段分配到存储区

一旦获知了所有输出段的大小，就可以将它们分配到存储区。通常，存储区在输出段的定义中指定。如果未指定存储区，则将使用首个定义的存储区。

存储区是按顺序填充的，从低地址到高地址，以段在段映射中出现的顺序进行。每个存储区中唯一的位置计数器跟踪下一个可用的存储单元。有两种情况会导致在一个存储区中分配存储器时产生空隙：

1. 段映射为输出段指定了绝对地址，或
2. 输出段具有特殊的对齐要求。

在这两种情况下，当前位置计数器和绝对（或对齐的）地址之间的任何存储空间将被跳过。分配到存储区中的所有项的确切地址将通过链接映射文件确定。

包含一个对齐的存储块的段（在 C 中具有 `aligned` 属性或在汇编中具有 `.align` 伪指令）也必须与相同（或更大）的对齐值对齐。如果两个或多个输入段有不同的对齐要求，对输出段使用最大的对齐。

9.3.3 分配未映射段

出现在段映射中的所有段分配完毕后，任何剩余的段均被视为未映射。未映射段根据段属性进行分配。链接器使用最佳适应存储器分配器来确定最有效的存储器排列方式。最佳适应分配器的重点是减少或消除由地址对齐限制产生的存储器空隙。按照约定，`.text`、`.data`、`.bss` 或 `.ramfunc` 段等大部分标准段不会显式映射到链接描述文件中，因此为最佳适应存储器分配器提供了最大灵活性。用于 gp 相对寻址的“小”数据段是一个例外。由于这些段必须组合在一起，因此将它们映射到链接描述文件中。将来的工具链版本可能允许使用最佳适应分配器来分配“小”数据段。

段属性对存储器分配的影响如下文所述。有关段属性的一般讨论，请参见**第 A.2 节“定义段的汇编器伪指令”**。

code

code 属性指定应按照链接描述文件中存储区 kseg0_program_mem 的定义，在程序存储器中分配段。以下属性可以与code配合使用，并进一步指定分配：

- address()指定绝对地址
- align()指定段起始地址的对齐方式

data

data属性指定应按照链接描述文件中存储区kseg0_data_mem和kseg1_data_mem的定义，在数据存储器中将段分配为已初始化存储。以下属性可以与 data 配合使用，并进一步指定分配：

- address()指定绝对地址
- near指定数据存储器的前64K
- align()指定段起始地址的对齐方式
- reverse()指定段结束地址 + 1的对齐方式

bss

bss属性指定应按照链接描述文件中存储区kseg0_data_mem和kseg1_data_mem的定义，在数据存储器中将段分配为未初始化存储。以下属性可以与bss配合使用，并进一步指定分配：

- address()指定绝对地址
- near指定数据存储器的前64K
- align()指定段起始地址的对齐方式
- reverse()指定段结束地址 + 1的对齐方式

persist

persist 属性指定应按照链接描述文件中存储区 kseg0_data_mem 和 kseg1_data_mem 的定义，在数据存储器中将段分配为持久存储。持久存储不会被C运行时库清零或初始化。以下属性可以与persist配合使用，并进一步指定分配：

- address()指定绝对地址
- near指定数据存储器的前64K
- align()指定段起始地址的对齐方式
- reverse()指定段结束地址 + 1的对齐方式

© 2014 Microchip Technology Inc.

9.4 全局符号和弱符号

当符号引用出现在目标文件中而其中没有相应的定义时，该符号将被声明为外部符号。在默认情况下，外部符号具有全局绑定并且作为全局符号来引用。可以通过在C语言中使用 `__weak__` 属性或在汇编语言中使用 `.weak` 伪指令进行弱绑定来显式声明外部符号。

如同其名称所暗示的那样，全局符号对于链接中涉及的全部输入文件都可见。每个引用的全局符号都必须有且只有一个定义。如果在输入文件中未找到全局定义，将搜索归档，并且将装入搜索到的第一个包含所需定义的归档模块。如果没有找到全局符号的定义，将报告链接错误。

弱符号与全局符号共享同一个名称空间，但是处理方式不同。允许对一个弱符号进行多次定义。如果在输入文件中找不到弱定义，并不会搜索归档，而是假设弱符号的所有引用的值均为 0。同名全局符号定义的优先级高于弱定义（或没有定义）。基本上，弱符号是可选的，可以被全局符号替代或者被完全忽略。

9.5 已初始化数据

链接器可自动支持数据存储器中的已初始化变量。变量在段中分配。每个数据段的声明都带有一个标志，指示其是否已初始化。

为了控制各个数据段的初始化，链接器构造了一个数据初始化模板。该模板在程序存储器中分配，并在启动时由运行时库处理。当应用程序主程序获得控制权时，数据存储器中的所有变量均已初始化。

- 标准数据段名称
- 数据初始化模板
- 运行时库支持

9.5.1 标准数据段名称

传统上，基于GNU技术的链接器支持所链接二进制文件中的三种段：

表9-1： 传统段名称

段名	说明	属性
.text	可执行代码	code
.data	接收初始值的数据存储器	data
.bss	未初始化的数据存储器	bss

名称“bss”可追溯到几十年前，表示存储器“以符号开始的块”。按照约定，程序启动期间，bss 存储器使用零填充。传统段名称被视为具有表 9-1 中列出的隐含属性。code属性指示该段包含可执行代码，应加载到程序存储器中。bss属性指示该段包含未初始化的数据存储，但在程序启动时将使用零填充。data 属性指示该段包含将在启动时接收初始值的数据存储。

汇编应用程序可使用段伪指令（“定义段的伪指令”一节中说明）来定义其他具有显式属性的段。对于 C 应用程序，32 位编译器将自动定义包含所需变量和函数的段。更多有关可引起自动段定义的变量和函数的属性的信息，请参见《MPLAB® XC32 C/C++编译器用户指南》（DS50001686G_CN）。

> **注：** 无论何时使用段伪指令，所有后续声明均汇编到指定段中。这会持续到另一个段伪指令出现或到达文件末尾。更多有关定义段和段属性的信息，请参见**第X.Y节“定义段的伪指令”**。

© 2014 Microchip Technology Inc.

9.5.2 数据初始化模板

如**第 9.5.1 节“标准数据段名称”**所述，32 位语言工具支持 BSS 段（未初始化的存储器）和数据段（接收初始值的存储器）。数据段在启动时接收初始值，BSS 段则使用零填充。使用支持任意数量的任意 BSS 或数据段的通用数据初始化模板。数据初始化模板由链接器创建，并加载到程序存储器中名为 `.dinit` 的输出段。运行时库中的启动代码将解释模板并相应地初始化数据存储器。

对于数据存储器中的每个输出段，数据初始化模板都包含一条记录。模板由一个空指令字终止。数据初始化记录的格式为：

```
/* data init record */
struct data_record {
char *dst; /* destination address */
unsigned int len; /* length in bytes */
unsigned int format:7; /* format code */
char dat[0]; /* variable length data */
};
```

记录的第一个元素是指向数据存储器中的段的指针。第二个和第三个元素分别是段长度和格式代码。最后一个元素是一个可选的数据字节数组。对于 BSS 段，则无需数据字节。

格式代码有两个可能的值。

表9-2： 格式代码值

格式代码	说明
0	用零填充输出段
1	从数据数组中的每个指令字复制4字节数据

9.5.3 运行时库支持

为了初始化数据存储器中的变量，必须先在启动时处理数据初始化模板，然后应用程序主函数才能进行控制。对于C程序，此任务由运行时库中的C启动模块执行。汇编语言程序还可通过链接libpic32.a来使用C启动模块。

要使用启动模块，应用程序必须允许运行时库在器件复位时获得控制权。对于C程序，此过程会自动进行。启动模块完成其工作后，调用应用程序的 main() 函数。汇编语言程序应使用以下命名约定来指定在器件复位时获得控制权的程序。

表9-3： 主入口点表格

主入口点名称	说明
`_reset`	器件复位后立即获得控制权
`main`	启动模块完成其工作后获得控制权

注意第一个入口点名称（_reset）包含一个前导下划线字符。第二个入口点名称（main）不包含前导下划线字符。器件复位时，将调用启动模块，该模块执行以下操作：

1. 初始化堆栈指针。
2. 读取`.dinit`段中的数据初始化模板，这会使所有未初始化的段清零，并且使所有已初始化的段使用从程序存储器读取的值进行初始化。
3. 将RAM函数从程序闪存复制到数据存储器并初始化总线矩阵寄存器。
4. 不带参数调用函数main。
5. 如果main返回，处理器将复位。

指定--no-data-init选项时，将链接备用启动模块。

它将执行相同操作，除了会忽略步骤（2）。备用启动模块比主启动模块小，如果无需数据初始化，可选择备用启动模块来节省程序存储器空间。

两个模块的源代码均在MPLAB XC32 C编译器安装目录的src目录中提供。必要时可修改启动模块。例如，如果应用程序需要带参数调用 main，可以切换条件汇编伪指令来提供此支持。

© 2014 Microchip Technology Inc.

9.6 堆栈分配

MPLAB XC32 C编译器将通用寄存器29专门用作软件堆栈指针。所有处理器堆栈操作（包括函数调用、中断和异常）都使用该软件堆栈。堆栈从高地址向低地址向下增长。

默认情况下，32 位链接器在未使用的数据存储器中动态分配可能的最大堆栈。以前版本使用链接描述文件中指定的输出段来分配堆栈。

堆栈的位置和大小在链接映射输出文件和“存储器使用情况报告”中的“动态存储器使用情况”标题下报告。应用程序可通过在链接器命令行上使用--defsym=_min_stack_size=size 链接器命令行选项指定大小来确保至少可使用最小堆栈。使用命令行为堆栈分配2048字节的示例如下：

```
xc32-gcc foo.c -Wl,--defsym=_min_stack_size=2048。
```

链接描述文件定义默认的`_min_stack_size`为1024。

> **注：** 更多关于编译器堆栈使用的信息，请参见《MPLAB® XC32 C/C++编译器用户指南》（DS50001686G_CN）。

链接器报告的`.stack`段的大小是避免发生链接错误所需的最小大小。有效的堆栈大小通常比报告的`.stack`段大小要大。

9.7 堆分配

C运行时堆是数据存储器的未初始化区域，用于使用标准C函数库的动态存储器管理函数calloc、malloc和realloc进行动态存储器分配。如果未直接或间接使用任何这些函数，那么无需分配堆。**默认的堆大小为0**。

如果希望通过调用其中一个存储器分配函数来直接使用动态存储器分配，或通过利用其中一个存储器分配函数的标准 C 库函数来间接使用动态存储器分配，那么必须创建堆。使用`--defsym=_min_heap_size`链接器命令行选项在链接器命令行上指定堆大小来创建堆。使用命令行分配512字节堆的示例如下：

```
xc32-gcc foo.c -Wl,--defsym=_min_heap_size=512
```

链接器紧接在分配堆栈之前分配堆。堆的位置和大小在链接映射输出文件和“存储器使用情况报告”中的“动态存储器使用情况”标题下报告。如果请求的大小不可用，链接器会报告错误。

现在堆由链接器动态分配。以前版本使用链接描述文件中指定的输出段来分配堆。

9.8 PIC32MX中断向量表

使用异常基址（EBASE <31:12>）寄存器计算给定中断的向量地址，而该异常基址寄存器在内核段（kseg）地址空间提供 4 KB 页对齐的基地址值（EBASE 是 CPU 寄存器。）使用EBASE和VS（INTCTL <9:5>）值计算地址。VS位提供两个相邻向量地址之间的向量间距。

链接描述文件如下创建相应的中断向量表：

```
PROVIDE(_vector_spacing = 0x00000001);
_ebase_address  = 0x9FC01000;

SECTIONS
{
  .app_excpt _GEN_EXCPT_ADDR :
  {
    KEEP(*(.gen_handler))
  } > exception_mem
  .vector_0 _ebase_address + 0x200 :
  {
    KEEP(*(.vector_0))
  } > exception_mem
  ASSERT (_vector_spacing == 0 || SIZEOF(.vector_0) <=
     (_vector_spacing << 5),
     "function at exception vector 0 too large")
  .vector_1 _ebase_address + 0x200 +
     (_vector_spacing << 5) * 1 :
  {
    KEEP(*(.vector_1))
  } > exception_mem
  ASSERT (_vector_spacing == 0 || SIZEOF(.vector_1) <=
     (_vector_spacing << 5),
     "function at exception vector 1 too large")
  .vector_2 _ebase_address + 0x200 +
     (_vector_spacing << 5) * 2 :
  {
    KEEP(*(.vector_2))
  } > exception_mem
/* … */
  .vector_63 _ebase_address + 0x200 +
     (_vector_spacing << 5) * 63 :
  {
    KEEP(*(.vector_63))
  } > exception_mem
  ASSERT (_vector_spacing == 0 || SIZEOF(.vector_63) <=
     (_vector_spacing << 5),
     "function at exception vector 63 too large")
}
```

表中的每个向量都根据`_ebase_address`和`_vector_spacing`符号的值创建为绝对地址处的输出段。对于表中64个向量中的每一个向量，都有一个输出段。

© 2014 Microchip Technology Inc.

9.9 具有专用可编程可变偏移量的PIC32 MCU中断向量表

一些PIC32系列针对向量间距提供了可变偏移量。此特性允许根据应用需求配置中断向量间距。可以使用与各个向量相关的OFFxxx寄存器为向量设置特定的中断向量偏移量。有关中断向量表可变偏移量特性的详细信息，请参见《PIC32系列参考手册》以及具体PIC32 MCU的数据手册。

XC32 工具链提供了与默认运行时启动代码配合使用的器件特定的默认链接描述文件和相应的目标文件。下表显示了 /pic32mx/lib/proc/<devicename> 下用于初始化向量表偏移量寄存器的文件。

器件链接描述文件	*<devicename>*.ld
向量偏移量初始化	vector_offset_init.o
器件运行时启动代码	crt0_*<boot_isa>*.o

9.9.1 器件特定的链接描述文件

应用程序源代码负责为每个中断向量创建一个.vector_*n*输入段。C/C++编译器在vector(n)或at_vector(n)属性应用于中断服务程序时创建此段。在汇编代码中，使用.section伪指令创建新命名段。

器件特定的链接描述文件创建名为.vectors的单个输出段，该输出段将项目中的所有各个.vector_*n*输入段组合在一起。中断向量表的起始处映射到地址（_ebase_address + 0x200）。链接描述文件还提供了_ebase_address符号的默认值。

对于每个向量，链接描述文件还创建一个名为 __vector_offset_*n* 的符号，该符号的值为向量地址与_ebase_address地址之间的偏移量。

```
PROVIDE(_ebase_address = 0x9D000000);

SECTIONS
{
 /* Interrupt vector table with vector offsets */
 .vectors _ebase_address + 0x200 :
 {
  /*  Symbol __vector_offset_n points to .vector_n if it exists,
   *  otherwise points to the default handler. The
   *  vector_offset_init.o module then provides a .data section
   *  containing values used to initialize the vector-offset SFRs
   *  in the crt0 startup code.
   */
 __vector_offset_0 = (DEFINED(__vector_dispatch_0) ? (. - _ebase_address) : __vector_offset_default);
 KEEP(*(.vector_0))
 __vector_offset_1 = (DEFINED(__vector_dispatch_1) ? (. - _ebase_address) : __vector_offset_default);
 KEEP(*(.vector_1))
 __vector_offset_2 = (DEFINED(__vector_dispatch_2) ? (. - _ebase_address) : __vector_offset_default);
 KEEP(*(.vector_2))

 /* … */

 __vector_offset_190 = (DEFINED(__vector_dispatch_190) ? (. - _ebase_address) : __vector_offset_default);
 KEEP(*(.vector_190))
 }
}
```

9.9.2 向量偏移量初始化模块

向量偏移量初始化模块（vector_offset_init.o）使用默认链接描述文件中定义的 __vector_offset_*n* 个符号。各个符号的值是向量地址与 ebase 寄存器地址之间的偏移量。向量偏移量初始化模块使用符号值来创建采用相应OFFxxx特殊功能寄存器地址的.data段。这意味着链接器生成的标准数据初始化模板包含用于初始化OFFxxx寄存器的值。

```
  .section
.data.__vector_offset_BF810540,data,keep,address(0xBF810540)
  .word __vector_offset_0
  .word __vector_offset_1
  .word __vector_offset_2
  .word __vector_offset_3
  .word __vector_offset_4
  .word __vector_offset_5
  .word __vector_offset_6
```

9.9.3 运行时启动代码数据初始化

通过这些添加到项目的 .data 段以及链接器生成的数据初始化模板，标准运行时启动代码将OFFxxx特殊功能寄存器初始化为常规已初始化数据。启动代码中无需特殊代码便可初始化OFFxxx寄存器。

© 2014 Microchip Technology Inc.

9.9.4 向量表汇编源代码示例

以下示例说明了如何为中断向量0创建向量分派。向量分派是从向量表到实际中断服务程序（Interrupt Service Routine，ISR）的跳转。

```
/*  Input section .vector_0 is mapped to the .vectors output
       *  section in the linker script.
       */
   .globl _vector_dispatch_0
   .section .vector_0,code
   .align 2
   .set nomips16
   .ent __vector_dispatch_0
__vector_dispatch_0:
      /* Jump to the actual ISR code */
   j  isrvector0
   nop
   .end __vector_dispatch_0
   .size __vector_dispatch_0, .-__vector_dispatch_0
```

以下示例说明了如何将中断服务程序直接置于向量表中。链接描述文件中的映射将自动调整向量间距以适应函数代码。

```
/*  Input section .vector_0 is mapped to the .vectors output
       *  section in the linker script.
       */
      .section .vector_0,code
   .align 2
   .globl isrvector0
   .set nomips16
   .set nomicromips
   .ent isrvector0
isrvector0:
   .set noat

      /*  Interrupt Service Routine code directly in the vector table.
       *  Be sure to preserve registers as appropriate for an ISR.
       */

   eret
   .set at
   .end isrvector0
   .size isrvector0, .-isrvector0
```

在XC32 C代码中，使用ISR函数中的标准`vector(n)`和`at_vector(n)`函数属性。更多关于这些函数属性的信息，请参见《MPLAB® XC32 C/C++ 编译器用户指南》（DS50001686G_CN）。

注：

© 2014 Microchip Technology Inc.

第10章 链接器示例

10.1 简介

32 位编译器和汇编器均提供了用于指示应用程序中要进行特殊处理的特定元素的语法。在 C 语言中，提供了丰富的属性来修改变量和函数的定义（请参见《MPLAB® XC32 C/C++ 编译器用户指南》（DS50001686G_CN））。在汇编语言中，变量和函数抽象成存储器段，存储器段成为链接器的输入。汇编器提供了另一组可用于修改段定义的属性（请参见**第4.6节“修改段对齐方式的伪指令”**）。

本章包括大量特定于32位链接器的示例，并介绍了C语言和汇编语言中的等效语法。

10.2 重点

本章涉及以下主题：

- 存储器地址和可重定位代码
- 定位变量到特定地址
- 定位函数到特定地址
- 定位和保留程序存储器

10.3 存储器地址和可重定位代码

对于大多数应用程序，最好编写完全可重定位的源代码，从而使链接器确定在存储器中放置函数和变量的准确地址。外部符号在存储器中的最终地址可通过链接映射输出确定，如下所示：

```
...
.text                       0x9d0000f0   0x64
 .text                      0x9d0000f0   0x64 test.o
                            0x9d0000f0   myfunc
                            0x9d000110   main

.text._DefaultInterrupt     0x9d000154   0x48
 .text._DefaultInterrupt    0x9d000154   0x48 c:/program files/
                                         microchip/xc32/v1.20/bin/
                                         ../lib/gcc/pic32mx/4.5.2/
                                         ../../../../pic32mx/
                                         lib\libpic32.a
                                         defaultinterrupt.o)
                            0x9d000154   _DefaultInterrupt...
```

在某些情况下，编程人员需要指定特定变量或函数所在的地址。传统上，这通过创建用户定义的段和编写自定义链接描述文件来实现。32 位汇编器和编译器提供了一组可用于在源代码中直接指定绝对地址和存储空间的属性。使用这些属性时，无需自定义链接描述文件。

> **注：** 编程人员通过指定绝对地址来负责确保指定的地址合理且可用。如果指定的地址超出范围，或与静态分配的资源发生冲突，则将出现链接错误。

© 2014 Microchip Technology Inc.

10.4 定位变量到特定地址

在此示例中，数组buf1定位到数据存储器中的特定地址。buf1的地址可通过在软件模拟器中执行程序或检查链接映射来确认。

```
#include "stdio.h"
int __attribute__((address(0xa0000200))) buf1[128];
```

汇编语言中的等效数组定义如下所示。.align 伪指令可选，它代表数据存储器中的默认对齐方式。使用 * 作为段名称会使汇编器根据源文件名称生成唯一名称。

```
.globl                  buf1
                        .section    *,address(0xa0000200),bss
                        .align      2
                        .type       buf1, @object
                        .size       buf1, 512
buf1:
                        .space      512
```

10.5 定位函数到特定地址

在此示例中，函数func定位到特定地址。

```
#include "stdio.h"
void __attribute__((address(0x9d002000))) func()
{}
```

汇编语言中的等效函数定义如下所示。.align 伪指令可选，它代表程序存储器中的默认对齐方式。使用 * 作为段名称会使汇编器根据源文件名称生成唯一名称。

```
.section                *,address(0x9d002000),code
                        .align      2
                        .globl      func
func:
                        ......
```

10.6 定位和保留程序存储器

在此示例中，保留了一个程序存储器块用于特殊目的，如自举程序。块中分配了任意大小的函数，剩余空间保留用于扩展或其他用途。

以下输出段定义将添加到自定义链接描述文件中：

```
BOOT_START = 0x9d00A200;
BOOT_LEN = 0x400;
my_boot BOOT_START :
{
*(my_boot);
. = BOOT_LEN; /* advance dot to the maximum length */
} > kseg0_program_mem
```

注意输入段之后的段定义中出现的“点赋值”（.=）。点是代表当前段中的位置计数器或下一填充点的特殊变量。它是相对于段起始地址的偏移量。该语句实际表示“无论输入段大小为何，都应确保输出段为完整大小。”

以下C函数将分配到保留的块中：

```
void __attribute__((section("my_boot"))) func1()
{
  /* etc. */
}
```

等效的汇编语言为：

```
 .section        my_boot,code
        .align   2
        .globl  func1
func1:
                              ......
```


© 2014 Microchip Technology Inc.

第11章 链接器错误/警告

11.1 简介

MPLAB XC32 目标链接器（xc32-ld）可生成错误和警告。在此给出了这些输出的描述性列表。该列表仅给出了链接器的最常见的诊断消息。

本附录涵盖以下主题：

- 严重错误
- 错误
- 警告

11.2 严重错误

以下错误表明链接器中发生了内部错误。如果链接器发出了下面列出的任何一个严重错误且您正在使用自定义链接描述文件，那么请检查描述文件是否指定了OUTPUT_FORMAT（elf32-tradlittlemips）和OUTPUT_ARCH（pic32mx）。其他值可导致链接器在不受支持的模式下工作。还请检查是否在链接器的命令行中只传递了完全受支持的选项。最后，请确保没有其他应用程序将链接器的输入或输出文件锁定。

如果在使用了正确的OUTPUT_FORMAT和OUTPUT_ARCH以及命令行选项后仍发生严重错误，请与 Microchip Technology（http://support.microchip.com）联系以获取工程师支持。请确保提供了导致错误发生的源代码和命令行选项的完整详细信息。

- Bfd backend error: bfd_reloc_ctor unsupported
- Bfd_hash_allocate failed creating symbol %s
- Bfd_hash_lookup failed: %e
- Bfd_hash_lookup for insertion failed: %e
- Bfd_hash_table_init failed: %e
- Bfd_hash_table_init of cref table failed: %e
- Bfd_link_hash_lookup failed: %e
- Bfd_new_link_order failed
- Bfd_record_phdr failed: %e
- Can't set bfd default target to `%s': %e
- Can not create link hash table: %e
- Can not make object file: %e
- Cannot represent machine `%s'
- Could not read relocs: %e
- Could not read symbols
- Cref_hash_lookup failed: %e
- Error closing file `%s'
- Error writing file `%s'
- Failed to create hash table
- Failed to merge target specific data
- File not recognized: %e
- Final close failed: %e
- Final link failed: %e
- Hash creation failed
- Out of memory during initialization
- Symbol `%t' missing from main hash table
- Target %s not found
- Target architecture respecified
- Unknown architecture: %s
- Unknown demangling style `%s'
- Unknown language `%s' in version information

© 2014 Microchip Technology Inc.

11.3 错误

下面列出的链接器错误通常表示传递给链接器的链接描述文件或命令行选项中存在错误。错误也可表示一个或多个输入目标文件或归档文件有问题。

符号

--gc-sections and -r may not be used together

垃圾收集段选项和可重定位输出选项不兼容。请删除 `--gc-sections` 选项或 `--relocatable`选项。

--relax and -r may not be used together

松散（relaxation）选项和可重定位输出选项不兼容。请删除任一选项。

A

A heap is required, but has not been specified.

使用标准C输入/输出函数时必须指定堆。

Assignment to location counter invalid outside of SECTION

只能在分配期间在 `SECTION` 内对特殊点符号进行赋值。请在链接描述文件中检查赋值语句的位置。

B

Bad --unresolved-symbols option: *option*

`--unresolved-symbols` 方法选项无效。注意，该选项不受支持。尝试使用默认的 `--unresolved-symbols=report-all`代替。

C

Can not PROVIDE assignment to location counter

只能在分配期间对特殊点符号进行赋值。*PROVIDE* 命令无法对位置计数器赋值。请从链接描述文件中删除错误的语句。

Can not set architecture: *arch_name*

如果使用了自定义链接描述文件，请检查该链接描述文件中是否存在 `OUTPUT_ARCH`（`pic32mx`）命令。PIC32 MCU链接器目前仅支持“pic32mx”架构。

Cannot move location counter backwards (from *addr1* to *addr2*)

下一个点符号值必须大于当前点符号值。

Could not allocate data memory.

链接器无法分配所有已分配到“kseg0_data_memory/kseg1_data_memory”区的段。

Could not allocate program memory.

链接器无法分配所有已分配到“kseg0_program_memory”区的段。

D

Dangerous relocation: *relocation_type*

符号被解析，但是使用时存在危险。例如，当代码使用GP相对寻址，但未定义_gp初始化符号，就会发生这种情况。_gp符号通常在链接描述文件中定义。

--data-init and --no-data-init options can not be used together.

--data-init创建了一个名为.dinit的特殊输出段作为数据运行时初始化的模板，而--no-data-init没有创建。只能使用一个选项。

F

File format not recognized; treating as linker script

其中一个输入文件不是可识别的ELF目标或归档文件。链接器假设它是链接描述文件。

G

Group ended before it began (--help for usage)

在命令行中，-)选项出现在-(选项前面。请检查链接器命令行中是否正确指定了此组。

I

Illegal use of *name* section

段名被保留。例如，可使用特殊输出段名/DISCARD/来丢弃输入段。输出文件中不包含任何分配给名为/DISCARD/的输出段的输入段。不应创建自己的名为/DISCARD/的输出段。

Includes nested too deeply

最大包含深度为10。

Invalid argument to option --section-start

--section-start的参数必须是sectionname=*org*。*org*必须是一个十六进制整数。sectionname、等号（=）和*org*之间不应该有空白。

Invalid assignment to location counter

对特殊点符号的赋值无效。

Invalid syntax in flags

段标志无效。接受的标志为：a r w x l。

M

Macros nested too deeply

宏最多只能嵌套10层。

May not nest groups (--help for usage)

归档组已启动。请在使用-(选项启动新的组之前，使用-)选项关闭当前组。

© 2014 Microchip Technology Inc.

Member %b in archive is not an object

归档成员不是有效目标。请检查用于Microchip MPLAB XC32 C/C++编译器的库归档是否正确。

Missing argument(s) to option --section-start

`--section-start`的必需参数必须为`sectionname=`*`org`*。

Multiple definition of *name*

链接器发现多次对某个符号进行定义。请删除多余的定义。

Multiple startup files

链接描述文件尝试设置启动文件，但已设置了某个启动文件。在链接描述文件中只能指定一个启动文件。

N

No input files

链接器找不到在命令行中指定的输入文件。链接器无法执行任何操作。请检查是否为链接器传递了正确的目标文件名。

Nonconstant expression for *name*

*`name`*必须是非常量表达式。

Not enough memory for stack (num bytes available).

没有足够的空余存储器来分配最小堆栈。

R

region *region* is full (*filename* section secname).

存储区*`region`*已满，但是已将段*`secname`*分配给了该区域。

Reloc refers to symbol *name* which is not being output

指令引用了没有被输出的符号。

Relocation truncated to fit *relocation_type name*.

此错误表明name的重定位值相对于预期用途过大。当地址超出了请求的指令的范围，就会发生这种情况。请检查是否在预期的段内声明并定义了该符号。例如，变量的声明和定义必须是常数或非常数。

Relocation truncated to fit: *relocation_type name* against undefined symbol *name*

如果符号不存在，就会发生这种错误。例如，代码调用未定义的函数。

U

Undefined MEMORY region *region* referenced in expression

表达式引用了链接描述文件中不存在的`MEMORY`区。

11.4 警告

当链接器做出某种假设以得以继续链接某个有缺陷的程序时，链接器会产生警告。不应该忽略警告。每条警告都应该被仔细检查并改正以确保链接器明白操作的目的。有时，警告消息能指出您程序中的错误。

C

Cannot find entry symbol *name*

如果链接器找不到指定的入口符号且该符号不是数字，请使用文本段中的第一个地址。

Changing start of section name by *num* bytes

由于对齐的原因，链接器正在更改指定段的起始地址。

D

data initialization has been turned off, therefore section secname will not be initialized.

指定的段需要初始化，但数据初始化已关闭；因此，将丢弃初始数据值。数据段的存储将如常分配。

I

initial values were specified for a non-loadable data section (name). These values will be ignored.

按照定义，持久数据段意味着数据未初始化；因此将丢弃这些值。段的存储将如常分配。

R

Redeclaration of memory region *name*

在链接描述文件中多次声明了`MEMORY`区。

U

Undefined reference to *name*

此符号是未定义的。

© 2014 Microchip Technology Inc.

第3部分——32位实用程序（包括归档器/库管理器）

第12章 MPLAB XC32目标归档器/库管理器 175
第13章 其他实用程序 183

注：

© 2014 Microchip Technology Inc.

第12章 MPLAB XC32目标归档器/库管理器

12.1 简介

MPLAB XC32目标归档器/库管理器（xc32-ar）可创建、修改和抽取归档中的文件。此工具是几个实用程序之一。“归档”是一个文件，它将其他一些文件的集合保存在一个结构中，并可从该结构中获取各原始文件（称为归档的“成员”）。

该32位归档器/库管理器能维护成员名为任意长度的归档，但如果使用了f修饰符，文件名就会被截短为15个字符。

由于该类归档通常用作包含常用子程序的“库”，因此归档器被认为是一个二进制实用程序。

当指定修饰符s时，归档器会在归档中为可重定位目标模块内定义的符号创建索引。创建后归档内的索引就会在归档器更改其内容（执行q更新操作而保存）时被更新。具有这种索引的归档可以加速与库的链接，并且让库中的程序可以相互调用而不必考虑它们在归档中的位置。

可以使用`xc32-nm -s`或`xc32-nm --print-armap`列出此索引表。如果归档缺少索引表，该32位归档器/库管理器的另一种称为`xc32-ranlib`的形式可用于只添加该表。

该32位归档器/库管理器设计为与两种不同的工具兼容。可以使用命令行选项控制其工作，或者，如果指定单个命令行选项 `-M`，则可以通过标准输入提供的描述文件对它进行控制。

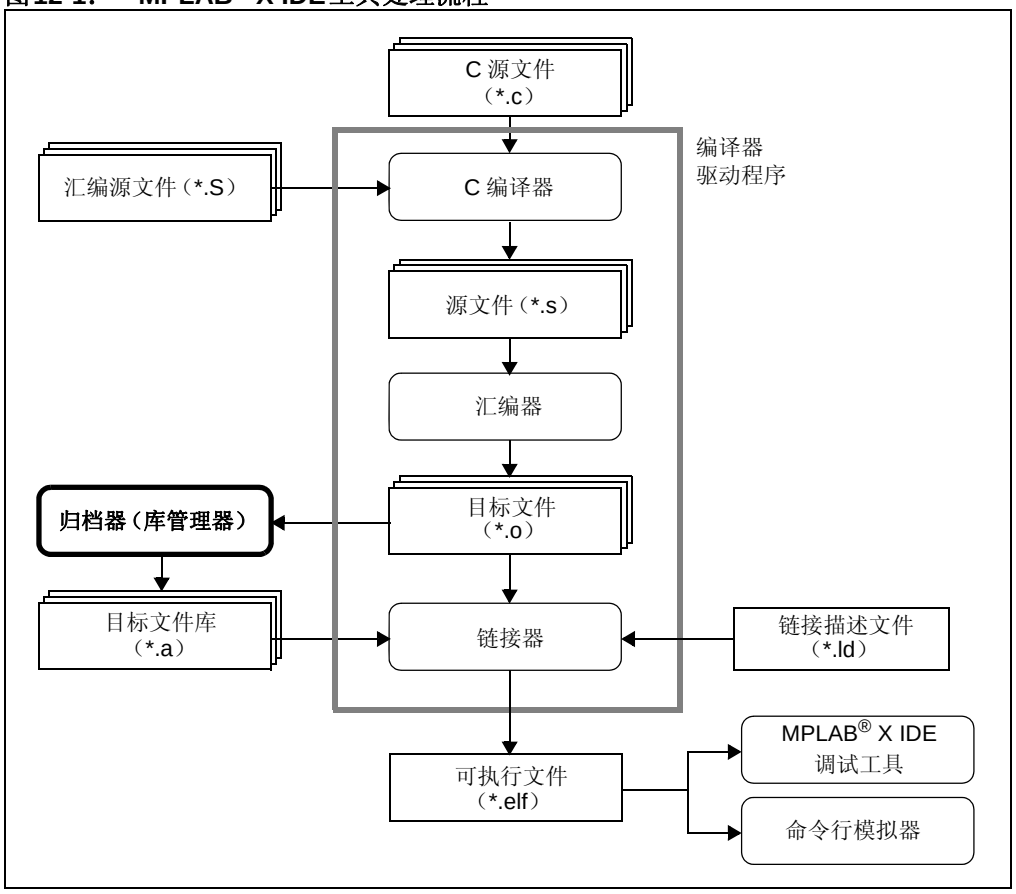
本章涵盖以下主题：

- 归档器/库管理器和其他开发工具
- 特性集
- 输入/输出文件
- 语法
- 选项
- 描述文件

12.2 归档器/库管理器和其他开发工具

32位库管理器从32位汇编器创建的目标文件来创建归档文件。然后32位链接器将归档文件与其他可重定位目标文件链接到一起，以创建一个可执行文件。工具处理流程的概况请参见图12-1。

图12-1： MPLAB® X IDE工具处理流程



© 2014 Microchip Technology Inc.

12.3 特性集

此汇编器的突出特性包括：

- 可在Linux x86、Mac OS X和Windows下使用
- 命令行接口

12.4 输入/输出文件

32位归档器/库管理器生成归档文件（.a）。归档文件是一个将其他文件集中保存在一个结构中的文件，这样就可从该结构中获取各原始文件（称为归档的成员）。使用ELF目标文件格式对所有目标文件进行处理。

由于该类归档通常用作包含常用子程序的库，因此xc32-ar被认为是一个二进制实用程序。

12.5 语法

```
xc32-ar [-]P[MOD [RELPOS] [COUNT]] ARCHIVE [MEMBER...]
xc32-ar -M [ <mri-script ]
```

12.6 选项

当通过命令行选项使用32位归档器/库管理器时，归档器执行必须至少要有两个参数：一个指定操作的关键字（可以与其他指定修饰符的关键字一起使用）和归档的名称。

xc32-ar [-]P[*MOD* [*RELPOS*][*COUNT*]] ARCHIVE [*MEMBER...*]

> **注：** 命令行选项区分大小写。

大多数操作还能进一步接受指定归档成员的*MEMBER*参数。若没有指定成员，将使用整个归档。

32位归档器/库管理器允许在第一个命令行参数中以任意次序混合操作码P和修饰符标志*MOD*。如果您愿意，也可以在第一个命令行参数前添加连字符（-）。

P关键字指定要执行的操作；它可以是下面任意一个，但是您只能指定其中一个。

表12-1： 要执行的操作

选项	功能
d	删除归档中的模块。指定要删除的模块名为MEMBER...；如果没有指定要删除的文件，将不对归档执行任何操作。 如果指定v修饰符，32位归档器/库管理器将列出删除的每个模块。
m	使用该操作移动归档中的成员。 如果在多个成员中对同一个符号进行了定义，归档中成员的次序会影响使用库链接程序的方式。 如果没有修饰符与m一起使用，在MEMBER参数中指定的任何成员会被移动到归档的末尾；可以使用a、b或i修饰符将成员移动到指定的位置。
p	将指定的归档成员输出到标准输出文件中。如果指定了修饰符v，在将成员的内容复制到标准输出前显示成员的名称。如果没有指定MEMBER参数，将输出归档中的所有文件。
q	将文件MEMBER...追加到ARCHIVE中。
r	将文件MEMBER...插入到ARCHIVE中（带替换）。 如果在MEMBER...中指定的某个文件不存在，归档器就会显示一条错误消息，而对现有的与该名称匹配的归档的成员没有影响。默认情况下，会将新成员添加到文件末尾；但也可以使用修饰符a、b或i中的一个来请求将新成员添加到相对于某个现有成员特定距离的位置上。可在该操作中使用修饰符v为每一个插入的文件引出一行输出，该修饰符与字母a或r连用以指示文件是被追加（不删除旧成员）还是被替换。
t	列出一张表，显示ARCHIVE的内容，或是归档中MEMBER...内所列文件的内容。通常只显示成员名称；如果还想看到模式（权限）、时间标记、作者、组和容量，也可以通过指定修饰符v来请求这些功能。如果没有指定MEMBER，将会列出归档中所有的文件。 例如，如果归档（b.a）中有多个文件名称相同（fie），xc32-ar t b.a fie只会列出第一个实例；要查看所有文件，必须用xc32-ar t b.a要求完整的列表。
x	从归档抽取成员（指定的MEMBER）。您可以在该操作中使用修饰符v请求归档器列出它所抽取的每一个成员的名称。 如果没有指定MEMBER，就会抽取归档中所有的文件。

© 2014 Microchip Technology Inc.

许多修饰符（MOD）可以紧跟在P关键字之后以指定一个操作的不同行为。

表12-2： 修饰符

选项	功能
a	在归档中的现有成员后添加新文件。如果使用修饰符 a，现有归档成员的名称必须在ARCHIVE说明前作为RELPOS参数出现。
b	在归档中的现有成员前添加新文件。如果使用修饰符 b，现有归档成员的名称必须在ARCHIVE说明前作为RELPOS参数出现（与i相同）。
c	创建归档。当请求更新时，如果指定的归档 ARCHIVE 之前不存在，则都将创建一个。但除非您在使用该修饰符前预先指明希望创建归档，否则将发出一个警告。
f	截短归档中的名称。32位归档器/库管理器通常允许任何长度的文件名。这使得创建的归档与一些系统上的本地归档器程序不兼容。如果要考虑这个问题，可以使用f修饰符在将这些文件放入归档时截短它们的名称。
i	在归档中现有成员前插入新文件。如果使用修饰符i，现有归档成员的名称必须在ARCHIVE说明前作为RELPOS参数出现（与b相同）。
l	接受该修饰符但不使用。
N	使用COUNT参数。如果在归档中有多项名称相同，就使用该选项。从归档中抽取或删除COUNT个给定名称的实例。
o	当抽取成员时保存成员的原始日期。如果不指定该修饰符，从归档中抽取的文件会标有抽取时的时间标记。
P	当寻找归档中名称匹配的成员时使用完整路径名。32 位归档器 / 库管理器不能创建带完整路径名的归档（这些归档与POSIX不兼容），但其他归档器则可以。该选项使得归档器使用完整路径名搜索匹配的文件名，这对于从由另一个工具创建的归档中抽取单个文件非常方便。
s	将目标文件索引写入归档，或更新现有的索引，即使归档没有其他更改也是如此。您可以在任何操作中使用该修饰符标志，也可以单独使用该修饰符标志。对一个归档运行xc32-ar s等同于对其运行ranlib。
S	不生成归档符号表。这能在某些步骤中加快构建大型库文件的速度。生成的归档不能被链接器使用。要建立符号表，必须在归档器执行的最后一步省略修饰符 S 或对归档运行ranlib命令。
u	通常，xc32-ar r...将列出的所有文件插入归档。如果希望仅插入那些列出的、比现有同名成员更新的文件，就应使用该修饰符。只能在 r（替换）操作中使用修饰符u。尤其不能使用qu组合，因为检查时间标记会损失q操作的速度优势。
v	该修饰符请求显示操作的冗长版本。当追加了 v 修饰符时，许多操作会显示其他信息，如会显示处理过的文件名。
V	该修饰符显示32位归档器/库管理器的版本号。

12.7 描述文件

如果对归档器使用单个命令行选项-M，就能使用基本命令语言对其操作进行控制。

```
xc32-ar -M [ <SCRIPT ]
```

如果标准输入直接来自终端，该形式的32位归档器/库管理器就会与终端进行交互。在交互过程中，归档器提示输入（提示符为 AR >），即使有错误，它仍将继续执行。如果重新将标准输入定位到一个描述文件，将不会发出任何提示，发生任何错误时，32位归档器/库管理器都会放弃执行（返回一个非0的退出码）。

归档器命令语言**并不是**专为与命令行选项等效而设计的；事实上，它在一定程度上减少了对归档的控制。命令语言的唯一用途是使已经为 MRI“库管理器”程序编写了描述文件的开发人员能较容易地过渡到32位归档器/库管理器。

32位归档器/库管理器命令语言的语法非常明了：

- 大写或小写的命令均能被识别；例如，LIST等同于list。为了明了起见，在下面的说明中使用大写命令。
- 可在每一行上显示一条命令；命令是每行的第一个词。
- 允许空行，但空行不起作用。
- 允许注释；“*”或“;”后的文本被忽略。
- 当使用一列名称作为 xc32-ar 命令的参数部分时，可以使用逗号或空格分隔这些名称。为明了起见，在下面的说明中采用逗号。
- “+”用作行连接符；如果“+”出现在行尾，下一行文本就将被认为是当前命令的一部分。

下表给出了可在归档器描述文件中或在使用归档器交互操作时使用的命令。其中3个具有特殊意义。

归档器描述文件命令

选项	功能
OPEN或CREATE	指定“当前归档”，这是大多数其他命令所需的临时文件。
SAVE	提交到目前为止描述文件指定的更改。执行 SAVE 命令前的命令仅影响当前归档的临时副本。
ADDLIB ARCHIVE ADDLIB ARCHIVE (MODULE, MODULE,...MODULE)	把 ARCHIVE 的所有内容（或在指定时，将 ARCHIVE 中所有指定的MODULE）添加到当前归档中。 要求在使用OPEN或CREATE后使用。
ADDMOD MEMBER, MEMBER, ... MEMBER	在当前归档中添加所有名为MEMBER的成员作为模块。 要求在使用OPEN或CREATE后使用。
CLEAR	丢弃当前归档的内容，取消执行最后一次 SAVE 命令后的任何操作的影响。即使没有指定当前归档，该选项可能仍会被执行（但不起作用）。

© 2014 Microchip Technology Inc.

归档器描述文件命令 （续）

选项	功能
CREATE ARCHIVE	创建归档并把它作为当前归档（许多其他命令需要使用该归档）。用临时名称创建新归档；在使用 SAVE 前它并没有真正存为ARCHIVE。可以改写现有归档；类似地，任何现有的名为 ARCHIVE 的文件的内容在SAVE前不会被破坏。
DELETE MODULE, MODULE, ... MODULE	从当前归档中删除每一个列出的MODULE；等同于xc32-ar -d ARCHIVE MODULE ... MODULE。 要求在使用OPEN或CREATE后使用。
DIRECTORY ARCHIVE (MODULE, ... MODULE) [OUTPUTFILE]	列出ARCHIVE中名为MODULE的所有模块。独立的命令 VERBOSE 指定输出的形式：关闭冗长输出时，输出形式如同xc32-ar -t ARCHIVE MODULE... 开启冗长输出时，列表如同 xc32-ar -tv ARCHIVE MODULE... 通常会输出到标准输出流；但是如果指定OUTPUTFILE为最后一个参数，32 位归档器 / 库管理器会指示输出到该文件。
END	退出归档器，使用 0 退出码来指示成功完成。该命令不保存输出文件；如果在上一次 SAVE 命令后更改了当前归档，这些更改将会丢失。
EXTRACT MODULE, MODULE, ...MODULE	从当前归档抽取每一个名为MODULE的模块，并作为独立文件写入当前目录。等同于xc32-ar -x ARCHIVE MODULE... 要求在使用OPEN或CREATE后使用。
LIST	在“冗长”格式下（不管 VERBOSE 的状态）显示当前归档的所有内容。作用如同xc32-ar tv ARCHIVE。（该命令是一个 32 位归档器 / 库管理器增强命令，而不与MRI兼容）。 要求在使用OPEN或CREATE后使用。
OPEN ARCHIVE	打开现有的归档作为当前归档使用（许多其他命令需要使用该归档）。在下次使用 SAVE 命令前，OPEN ARCHIVE 后的命令产生的任何更改将不会真正影响ARCHIVE。
REPLACE MODULE, MODULE, ...MODULE	用当前工作目录中的文件替换当前归档中每一个现有的名为MODULE的模块（在REPLACE参数中指定）。要无错误地执行该命令，在当前归档中必须存在文件和模块。 要求在使用OPEN或CREATE后使用。
VERBOSE	切换控制从目录（DIRECTORY）输出的内部标志。当标志启用时，DIRECTORY输出与从xc32-ar -tv ... 的输出匹配。
SAVE	将更改提交到当前归档，实际使用在上一次CREATE或OPEN命令中指定的名称另存文件。 要求在使用OPEN或CREATE后使用。

注：

© 2014 Microchip Technology Inc.

第13章 其他实用程序

13.1 简介

除了MPLAB XC32目标归档器/库管理器（xc32-ar）之外，还有几个其他的二进制实用程序可供MPLAB XC32汇编器和链接器使用。

本章涵盖以下主题：

实用程序	说明
xc32-bin2hex实用程序	将链接的目标文件转换为Intel® hex文件。
xc32-nm实用程序	列出目标文件中的符号。
xc32-objdump实用程序	显示有关目标文件的信息。
xc32-ranlib实用程序	生成归档内容的索引并将它存储在归档中。
xc32-size实用程序	列出文件段大小和总大小。
xc32-strings实用程序	显示可显示的字符序列。
xc32-strip实用程序	丢弃来自一个目标文件的所有符号。

13.2 xc32-bin2hex实用程序

二进制-十六进制实用程序（xc32-bin2hex）将二进制文件（来自32位链接器）转换为Intel十六进制格式的文件，以便加载到器件编程器中。

- 输入/输出文件
- 语法
- 选项

13.2.1 输入/输出文件

- 输入：ELF格式的二进制目标文件
- 输出：Intel十六进制文件

13.2.2 语法

命令行语法为：

xc32-bin2hex [options] *file*

例13.1： hello.elf

将绝对ELF可执行文件hello.elf转换为hello.hex

```
xc32-bin2hex hello.elf
```

13.2.3 选项

支持以下选项。

表13-1： xc32-bin2hex选项

选项	功能
-a, --sort	根据地址来对段排序
-i, --virtual	使用虚拟地址
-p, --physical	使用物理地址（默认）
-v, --verbose	打印出冗长消息
-?, --help	打印帮助屏幕

> **注：** 有关PIC32MX的虚拟存储器到物理存储器的固定存储器映射的说明，请参见《PIC32MX系列参考手册》。

例13-2： -V选项输出

```
writing hello.hex

section  PC address   byte address   length (w/pad)   actual length  (dec)
-------   ----------   ------------   --------------   --------------------
.reset             0              0              0x8             0x6  (6)
.text          0x100          0x200           0x6a28          0x4f9e  (20382)
.dinit        0x3614         0x6c28            0xda4           0xa3b  (2619)
.const        0x3ce6         0x79cc             0x40            0x30  (48)
.ivt             0x4            0x8             0xf8            0xba  (186)
.aivt           0x84          0x108             0xf8            0xba  (186)

Total program memory used (bytes):                            0x5b83  (23427)
```


© 2014 Microchip Technology Inc.

13.3 xc32-nm 实用程序

xc32-nm 实用程序会生成来自目标文件的符号列表。列表中的每项均由符号值、符号类型和符号名称组成。

- 输入文件
- 语法
- 选项
- 输出格式

13.3.1 输入文件

- 输入：ELF 目标文件。如果没有作为参数列出的目标文件，xc32-nm 假设此文件为a.out。

13.3.2 语法

命令行语法为：

```
xc32-nm [ -A | -o | --print-file-name ]
      [ -a | --debug-syms ] [ -B ]
      [ --defined-only ] [ -u | --undefined-only ]
      [ -f format | --format=format ] [ -g | --extern-only ]
      [ --help ] [-l | --line-numbers ]
      [ -n | -v | --numeric-sort ]  [-omf=format]
      [ -p | --no-sort ]
      [ -P | --portability ] [ -r | --reverse-sort ]
      [ -s  --print-armap ] [ --size-sort ]
      [ -t radix | --radix=radix ] [ -V | --version ]
      [ OBJFILE... ]
```

13.3.3 选项

表13-2中给出的长短形式的选项是等效的，可以互换。

表13-2： xc32-nm选项

选项	功能
-A -o --print-file-name	在每个符号之前加上符号所在的输入文件（或归档成员）名，而不是仅仅在所有符号前面标识一下输入文件。
-a --debug-syms	显示所有符号，甚至仅供调试器使用的符号，这些符号通常不列出。
-B	与--format=bsd相同。
--defined-only	仅显示为每个目标文件定义的符号。
-u --undefined-only	仅显示未定义的符号（每个目标文件外部的符号）。
-f *format* --format=*format*	使用输出格式*format*，可以是bsd、sysv或posix。默认为bsd。只有 *format* 的第一个字符有意义；它既可以是大写也可以是小写。
-g --extern-only	仅显示外部符号。
--help	显示xc32-nm选项的汇总并退出。
-l --line-numbers	使用调试信息尝试找到每个符号所在文件的文件名和行号。对于定义的符号，找到符号地址的行号。对于未定义的符号，找到引用该符号的重定位项的行号。如果找到行号信息，则将其显示在其他符号信息之后。
-n -v --numeric-sort	用符号地址的数值顺序对符号进行排序，而不是使用其名称的字母顺序来排序。
-p --no-sort	不必以任何顺序将符号排序，只要按照其出现的顺序显示它们就可以了。
-P --portability	使用POSIX.2标准输出格式而不是默认格式。与-f posix等效。
-r --reverse-sort	反转排序的顺序（不管是数字还是字母顺序）；把排在最后面的放在最前面。
-s --print-armap	在列出归档成员中的符号时，还要列出索引：包含符号名称定义的模块的映射（用xc32-ar或xc32-ranlib存储在归档中）。
--size-sort	根据大小来对符号排序。此大小是符号值与下一个值更大的符号的值之差。显示的是符号的大小，而不是它的值。
-t *radix* --radix=*radix*	使用*radix*作为基数来显示符号值。如果是十进制数为d，八进制数则为o，十六进制数则为x。
-V --version	显示xc32-nm的版本号并退出。

© 2014 Microchip Technology Inc.

13.3.4 输出格式

符号值采用选项选择的基数，或者默认使用十六进制。

如果符号类型是小写的，此符号为局部符号；如果是大写的，此符号为全局符号（外部）。表13-3给出了符号类型：

表13-3： 符号类型

符号	说明
A	符号值是绝对的，而且不会被进一步的链接更改。
B	符号在未初始化的数据段（称为BSS）中。
C	此符号是共用的。共用符号是未初始化的数据。在链接时，会出现多个使用同一名称的共用符号。如果此符号在任何地方定义，该共用符号将被视为未定义的引用。
D	此符号在已初始化数据段中。
N	此符号是调试符号。
R	此符号在只读的数据段中。
T	此符号在文本（代码）段中。
U	此符号是未定义的。
V	此符号是一个弱对象。当将弱定义的符号和正常定义的符号相链接时，使用正常定义的符号不会产生错误。当链接尚未定义的弱符号时，弱符号的值将变为0，并且不会产生错误。
W	此符号是一个没有被特别标记为弱对象符号的弱符号。当将弱定义的符号和正常定义的符号相链接时，使用正常定义的符号不会产生错误。当链接尚未定义的弱符号时，弱符号的值将变为0，并且不会产生错误。
?	此符号类型未知或由目标文件格式决定。

13.4 xc32-objdump实用程序

xc32-objdump 实用程序显示有关一个或多个目标文件的信息。此选项控制要显示的特定信息。输出可提供类似于反汇编列表之类的信息。

- 输入文件
- 语法
- 选项

13.4.1 输入文件

- 输入：目标归档文件。如果没有作为参数列出的目标文件，xc32-objdump 假设此文件为a.out。

13.4.2 语法

命令行语法为：

```
xc32-objdump [ -a | --archive-headers ]
      [ -d | --disassemble ]
      [ -D | --disassemble-all ]
      [ -f | --file-headers ]
      [ --file-start-context ]
      [ -g | --debugging ]
      [ -h | --section-headers | --headers ]
      [ -H | --help ]
      [ -j name | --section=name ]
      [ -l | --line-numbers ]
      [ -M options | --disassembler-options=options]
      [ --prefix-addresses]
      [ -r | --reloc ]
      [ -s | --full-contents ]
      [ -S | --source ]
      [ --[no-]show-raw-insn ]
      [ --start-address=address ]
      [ --stop-address=address ]
      [ -t | --syms ]
      [ -V | --version ]
      [ -w | --wide ]
      [ -x | --all-headers ]
      [ -z | --disassemble-zeros ]
      OBJFILE...
```

OBJFILE… 是要分析的目标文件。在指定归档时，xc32-objdump 显示关于每个成员目标文件的信息。

© 2014 Microchip Technology Inc.

13.4.3 选项

表13-4中给出的长短形式的选项是等效的，可以互换。必须至少给出以下选项中的一个：-a、-d、-D、-f、-g、-G、-h、-H、-p、-r、-R、-S、-t、-T、-V或-x。

表13-4： xc32-objdump选项

选项	功能
-a --archive-header	如果任何目标文件OBJFILE都是归档的，则显示归档标头信息（格式类似于ls -l）。除了可以用xc32-ar tv列出的信息之外，xc32-objdump -a 还将显示每个归档成员的目标文件格式。
-d --disassemble	显示目标文件OBJFILE中机器指令的汇编助记符。此选项只会反汇编那些希望包含指令的段。
-D --disassemble-all	和-d一样，但是会反汇编所有段的内容，而不仅仅是希望包含指令的段。
-f --file-header	显示每个目标文件OBJFILE总标头的汇总信息。
--file-start-context	指定当显示尚未显示的文件的内部列出的源代码/反汇编代码（使用-S）时，将显示的内容扩展到文件的开头部分。
-g --debugging	显示调试信息。尝试解析存储在文件中的调试信息并使用和C 语言类似的语法将其显示出来。仅可实现某些类型的调试信息。
-h --section-header --header	显示目标文件的段标头的汇总信息。 文件段可以被重定位到非标准地址，例如使用ld的-Ttext、-Tdata 或 -Tbss 选项。但是，某些目标文件格式，如a.out，不存储文件段的起始地址。在这种情况下，虽然ld会正确地重定位段，但使用xc32-objdump -h列出的文件段标头就不会显示正确的地址，而是显示常规地址，此地址对于目标器件是隐含的。
-H --help	显示xc32-objdump选项的汇总并退出。
-j *name* --section=*name*	仅显示段*name*的信息。
-l --line-numbers	用文件名和目标代码或可重定位代码所对应的源代码行号来标记显示的内容（使用调试信息）。只可和-d、-D或-r一起使用。
-M *options* --disassembler-options=*options*	将特定于目标器件的信息传递到反汇编程序。PIC32 器件支持以下特定于目标器件的选项： symbolic：将执行符号反汇编。
--prefix-addresses	在反汇编时，在每行显示完整的地址。这是较早的反汇编格式。
-r --reloc	显示文件的重定位项。如果和-d或-D配合使用，这些重定位信息将和反汇编一起显示。
-s --full-contents	显示所请求的任何段的完整内容。

表13-4： xc32-objdump选项（续）

选项	功能
-S --source	可能的话，将源代码和反汇编代码混合显示。隐含-d。
--show-raw-insn	在反汇编指令时，用十六进制格式以及符号形式显示指令。除了使用--prefix-addresses的情况之外，这是默认情况。
--no-show-raw-insn	在反汇编指令时，不显示指令字节。在使用--prefix-addresses时，这是默认情况。
--start-address=*address*	开始在指定的地址显示数据。这将影响-d、-r和-s选项的输出。
--stop-address=*address*	停止在指定的地址显示数据。这将影响-d、-r和-s选项的输出。
-t --syms	显示文件的符号表项。这与xc32-nm程序提供的信息相似。
-V --version	显示xc32-objdump的版本号并退出。
-w --wide	为具有大于80列的输出器件格式化一些行。
-x --all-header	显示所有可用的标头信息，包括符号表和重定位项。使用-x和指定所有-a -f -h -r -t等效。
-z --disassemble-zeroes	通常，反汇编输出会跳过为零的块。此选项让反汇编程序对这些块进行反汇编，就和其他数据一样。

© 2014 Microchip Technology Inc.

13.5 xc32-ranlib实用程序

xc32-ranlib实用程序生成归档内容的索引，并将它存储在归档中。索引列出由归档成员（可重定位的目标文件）定义的每个符号。可以使用xc32-nm -s或xc32-nm --print-armap列出此索引。具有这种索引的归档可以加速与库的链接，并且让库中的程序可以相互调用而不必考虑它们在归档中的位置。

运行xc32-ranlib与执行xc32-ar -s（即，使用-s选项的32位归档器/库管理器）完全等效。

- 输入/输出文件
- 语法
- 选项

13.5.1 输入/输出文件

- 输入：归档文件
- 输出：归档文件

13.5.2 语法

命令行语法为：

```
xc32-ranlib [-v | -V | --version] ARCHIVE
xc32-ranlib [-h | --help]
```

13.5.3 选项

下表给出的长短形式的选项是等效的，可以互换。

表13-5： xc32-ranlib选项

选项	功能
-v -V --version	显示xc32-ranlib的版本号
-h --help	输出帮助消息

13.6 xc32-size实用程序

xc32-size 实用程序在参数列表中列出每个目标文件或归档文件的段大小和总大小。默认情况下，为每个目标文件或归档中的每个模块生成一行输出。

> **注：** 链接器的--report-mem存储器使用报告提供关于存储器使用的更多信息。

- 输入/输出文件
- 语法
- 选项
- 示例

13.6.1 输入/输出文件

- 输入：目标文件或归档文件
- 输出：标准输出

13.6.2 语法

xc32-size的命令行语法为：

```
xc32-size [ -A | -B | --format=compatibility ]
[ --help ]
[ -d | -o | -x | --radix= number]
[ -t | --totals ]
[ -V | --version ]
[objfile...]
```

13.6.3 选项

xc32-size选项显示如下。

表13-6： xc32-size选项

选项	功能
-A -B --format=*compatibility*	使用这些选项之一，可以选择gnu大小的输出是与System V大小的输出（使用 -A 或 --format=sysv）还是 Berkeley 大小（使用 -B 或 --format=berkeley）的输出类似。默认格式是与Berkeley相似的一行格式。
--help	显示可接受参数和选项的汇总。
-d -o -x --radix=number	使用这些选项之一，可以控制每个段的大小是采用十进制（-d或--radix=10）、八进制（-o或--radix=8）还是十六进制（-x或--radix=16）。在--radix=*number*中，仅支持三个值（8、10和16）。总大小始终以两种数制表示：-d或-x输出为十进制和十六进制；如果使用-o，则为八进制和十六进制。
-t --totals	显示列出的所有对象的总大小（仅限Berkeley格式的列表模式）。
-V --version	显示xc32-size的版本号。

© 2014 Microchip Technology Inc.

13.6.4 示例

这里给出了大小的Berkeley（默认）格式输出的示例：

```
xc32-size --format=Berkeley ranlib size
text   data  bss   dec    hex   filename
294880 81920 11592 388392 5ed28 ranlib
294880 81920 11888 388688 5ee50 size
```

下面显示的是相同数据，但更接近于System V约定：

```
xc32-size --format=SysV ranlib size
ranlib :
section   size   addr
.text   294880   8192
.data    81920 303104
.bss     11592 385024

Total 388392
size :
section   size   addr
.text   294880   8192
.data    81920 303104
.bss     11888 385024
Total 388688
```

13.7 xc32-strings实用程序

对于每一个给定的文件，xc32-strings实用程序输出至少4个字符长（或选项中给出的长度）的可显示字符序列，并后跟一个不可显示字符。默认情况下，它只输出目标文件内已初始化和加载段内的字符串；对于其他类型的文件，它将输出整个文件中的字符串。

xc32-strings主要用于确定非文本文件的内容。

- 输入/输出文件
- 语法
- 选项

13.7.1 输入/输出文件

- 输入：ELF目标文件
- 输出：标准输出

13.7.2 语法

命令行语法为：

```
xc32-strings [-a | --all | -] [-f | --print-file-name]
      [--help] [-min-len | -n min-len | --bytes=min-len]
      [-t radix | --radix=radix] [-v | --version] FILE...
```

13.7.3 选项

表13-7中给出的长短形式的选项是等效的，可以互换。

表13-7： xc32-strings选项

选项	功能
-a --all -	不要仅扫描目标文件中已初始化和加载的段，而是扫描整个文件。
-f --print-file-name	在每个字符串前显示文件名。
--help	在标准输出上显示程序用法的汇总并退出。
-*min-len* -n *min-len* --bytes=*min-len*	显示至少-min-*len*个字符长的字符序列而不是默认的4个字符长。
-t *radix* --radix=*radix*	在文件中的每个字符串之前显示偏移量。单字符参数指定偏移量的数制，o为八进制，x为十六进制，d为十进制。
-v --version	在标准输出上显示程序版本号并退出。

© 2014 Microchip Technology Inc.

13.8 xc32-strip实用程序

xc32-strip 实用程序丢弃来自指定的目标文件和归档文件中的所有符号。必须给定至少一个文件。xc32-strip 修改在其所带参数中指定的文件，而不是写以不同的名称命名的改写副本。

- 输入/输出文件
- 语法
- 选项

13.8.1 输入/输出文件

- 输入：目标或归档文件
- 输出：目标或归档文件。如果没有列出作为参数的目标文件或归档文件，xc32-strip假设此文件为a.out。

13.8.2 语法

命令行语法为：

```
xc32-strip [ -g | -S | --strip-debug ] [ --help ]
      [ -K symbolname | --keep-symbol=symbolname ]
      [ -N symbolname | --strip-symbol=symbolname ]
      [ -o file ]
      [ -p | --preserve-dates ]
      [ -R sectionname | --remove-section=sectionname ]
      [ -s | --strip-all ] [--strip-unneeded]
      [ -v | --verbose ]  [ -V | --version ]
      [ -x | --discard-all ] [ -X | --discard-locals ]
      OBJFILE...
```

13.8.3 选项

表13-8中给出的长短形式的选项是等效的，可以互换。

表13-8： xc32-strip选项

选项	功能
-g -S --strip-debug	仅删除调试符号。
--help	显示xc32-strip选项的汇总并退出。
-K *symbolname* --keep-symbol=*symbolname*	仅保留源文件中的符号 *symbolname*。可以多次使用该选项。
-N *symbolname* --strip-symbol=*symbolname*	从源文件中删除符号 *symbolname*。可以多次使用该选项，并且可以与除-K外的删除选项组合。
-o *file*	把删除的输出文件放入 *file*，而非替换现有的文件。当使用该参数时，只可指定一个OBJFILE参数。
-p --preserve-dates	保留访问和修改文件的日期。
-R *sectionname* --remove-section=*sectionname*	从输出文件中删除任何名为 *sectionname* 的段。可以多次使用该选项。注意若使用该选项不当会造成输出文件不可用。
-s --strip-all	删除所有符号。
--strip-unneeded	删除重定位处理不需要的所有符号。
-v --verbose	冗长输出：列出所有修改过的目标文件。对于归档，xc32-strip -v列出归档的所有成员。
-V --version	显示xc32-strip的版本号。
-x --discard-all	删除非全局符号。
-X --discard-locals	删除编译器生成的局部符号。 （这些符号的开头通常带有L或“.”）。

© 2014 Microchip Technology Inc.

第4部分—— 附录

附录A 已弃用的功能 199
附录B 有用的表格 201
附录C GNU自由文档许可证 203

注：

© 2014 Microchip Technology Inc.

附录A 已弃用的功能

A.1 简介

下面说明的功能被视为作废，并且已被更高级的功能取代。依赖于已弃用功能的项目可与所使用的语言工具版本正常配合使用。使用弃用的功能会导致出现警告；建议编程人员对其项目进行修订，以消除任何对已弃用功能的依赖。未来的版本可能完全删除对这些功能的支持。

A.2 定义段的汇编器伪指令

XC32 v2.00中已弃用下面的.section伪指令格式。新的伪指令格式请参见**第4.2节“定义段的伪指令”**。

.section *name* [, *flags*] [, @*type*]

使用.section伪指令将后面的代码汇编到名为name的段中。可选的flags参数是一个用引号括起的字符串，可包含以下字符的任何组合：

a 表示段是可分配的

w 表示段是可写的

x 表示段是可执行的

@*type*参数可以是以下其中之一：

@progbits 带有内容的正常段

@nobits 不包含数据的段（即段只占用空间）

以下段名可被识别：

表A-1： 段名

段名	默认标志
.text	x
.data	d
.bss	b

> **注：** 确保用双引号括住flags参数。如果.section伪指令的可选参数未用引号括住，则将被视作子段号。记住，单引号括住的单个字符（如‘b’）将被预处理程序转换为一个数字。

段伪指令示例

```
.section foo,"aw",@progbits #foo is initialized
#data memory.
.section fob,"aw",@nobits #fob is uninitialized
#(but also not zeroed)
#data memory.
.section bar,"ax",@progbits #bar is in program memory
```


© 2014 Microchip Technology Inc.

附录B 有用的表格

B.1 简介

以下是一些供参考的有用表格。

- ASCII字符集
- 十六进制到十进制的转换

B.2 ASCII字符集

此表格列出了ASCII标准字符集。

	最高有效半字节								
	十六进制	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
最低有效半字节	**0**	NUL	DLE	Space	0	@	P	`	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	Bell	ETB	'	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	|
	D	CR	GS	–	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

B.3 十六进制到十进制的转换

此表格说明了十六进制转换到十进制的方法。对于每个十六进制数字，查找与之相应的十进制值，并将这些十进制数相加。

高字节				低字节			
十六进制数的千位	十进制数	十六进制数的百位	十进制数	十六进制数的十位	十进制数	十六进制数的个位	十进制数
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

例如，十六进制数A38F按如下方式转换为41871：

十六进制数的千位	十六进制数的百位	十六进制数的十位	十六进制数的个位	结果
40960	768	128	15	十进制数41871

© 2014 Microchip Technology Inc.

附录C GNU自由文档许可证

Copyright (C) 2010 Microchip Technology Inc.

允许在Free Software Foundation发布的GNU自由文档许可证（版本1.3或更高版本）的条款下复制、分发和/或修改本文档；不包括不可变章节、封面文本和封底文本。

注：

© 2014 Microchip Technology Inc.

术语表

A

ANSI

美国国家标准学会（American National Standards Institute，ANSI）是美国负责制定和核准标准的组织。

ASCII

美国信息交换标准码是使用7位二进制数来表示每个字符的字符集编码。它包括大小写字母、数字、符号和控制符。

B

八进制（Octal）

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

本地数据大小（Native Data Size）

对于本地跟踪，Watch（观察）窗口中使用的变量长度必须与所选器件的数据存储器长度相同：对于PIC18器件，为字节长度；对于16位器件，为字长度。

编译（Build）

为应用程序编译和链接所有源文件。

编译器（Compiler）

将用高级语言编写的源文件翻译成机器码的程序。

标识符（Identifier）

函数或变量名。

表达式（Expressions）

用算术或逻辑运算符分隔开来的常数和/或符号的组合。

C

C\C++

C语言是一种通用的编程语言，具有精简的表达式、现代的控制流和数据结构，以及丰富的运算符集。C++是C语言的面向对象版本。

COFF

公共目标文件格式。此格式的目标文件包含机器码、调试信息和其他信息。

CPU

*请参见*中央处理单元（Central Processing Unit）。

操作码（Opcodes）

操作码。请参见助记符（Mnemonics）。

插件（Plug-ins）

MPLAB IDE/MPLAB X IDE 使用内置组件和插件模块来为各种软件和硬件工具配置系统。在Tools（工具）菜单下可找到几个插件工具。

场景（Scenario）

在MPLAB SIM软件模拟器中，用于激励控制的特殊设置。

程序存储器（Program Memory）

MPLAB IDE/MPLAB X IDE——器件中存储指令的存储区。也是仿真器或软件模拟器中包含已下载目标应用固件的存储器。

MPLAB XC16汇编器/编译器——器件中存储指令的存储区。

程序计数器（Program Counter）

包含当前正在执行的指令的地址的存储单元。

程序计数器单元（Program Counter Unit）

MPLAB XC16 汇编器 —— 程序存储器布局的概念化表示。每个指令字使程序计数器递增2。在可执行段中，2个程序计数器单元等于3个字节。在只读段中，2个程序计数器单元等于2个字节。

持久数据（Persistent Data）

从不被清零或初始化的数据。它的主要用途为在器件复位时应用依然能保存数据。

重定位（Relocation）

是链接器执行的一种过程，在此过程中，绝对地址被分配给可重定位段，而且可重定位段中的所有符号均被更新到新地址。

触发输出（Trigger Output）

触发输出是指可在任何地址或地址范围产生的仿真器输出信号，它独立于跟踪和断点设置。可以设置任意多个触发输出点。

次数计数器（Pass Counter）

每当事件（如在某个特定地址执行指令）发生时进行递减计数的计数器。当次数计数值达到零时，对该事件做出响应。可以将次数计数器分配给中断和跟踪逻辑，也可以将它分配给Complex Trigger（复杂触发）对话框中的任何连续事件。

存储类别（Storage Class）

它决定与所标识对象相关的存储器的寿命。

存储器模型（Memory Model）

对于C编译器，为可供应用程序使用的存储器的表示。对于PIC18 C编译器，为指定指向程序存储器的指针长度的描述。

© 2014 Microchip Technology Inc.

存储限定符（Storage Qualifier）

指示被声明对象的特殊属性（例如`const`）。

错误/错误文件（Error/Error File）

错误能报告无法继续处理您的程序的问题。可能情况下，错误能识别有明显问题的源文件名和行号。错误文件包含语言工具产生的错误消息和诊断。

D

DWARF

用任意记录格式调试。DWARF是ELF文件的一种调试信息格式。

带序列号的快速批量编程（Serialized Quick Turn Programming）

序列化允许将序列号烧写到器件编程器要编程的每个单片机器件中。此序列号可用作入口码、密码或ID号。

单步进入（Step Into）

此命令与Single Step相同。Step Into（与Step Over相对）在CALL指令之后，单步执行子程序。

单步跳出（Step Out）

Step Out允许单步跳出您正在单步执行的子程序。该命令执行子程序中代码的其余部分，然后在子程序的返回地址处停止执行。

单步跳过（Step Over）

Step Over使您调试代码时不单步执行子程序就可调试代码。当Step Over一条CALL指令时，下一个断点将设置在CALL指令后的下一条指令处。如果由于某种原因，子程序陷入无限循环或返回不正确，下一个断点将永远执行不到。除处理 CALL 指令以外，Step Over命令与Single Step相同。

单步执行（Single Step）

此命令以一次执行一条指令的方式执行代码。每条指令执行后，MPLAB IDE/MPLAB X IDE会更新寄存器窗口、观察变量和状态显示，从而使您能分析和调试指令执行。也可以单步执行C编译器的源代码，但是MPLAB IDE/MPLAB X IDE并不是执行单条指令而是执行所有由这行高级C语句生成的汇编指令。

单片机（Microcontroller）

一种高度集成的芯片，包含CPU、RAM、程序存储器、I/O端口和定时器。

单片机模式（Microcontroller Mode）

PIC18单片机的一种可能的程序存储器配置。在单片机模式下，只允许执行内部存储器中的指令。因此，在单片机模式下只能使用片上程序存储器。

导出（Export）

以标准化的格式将数据从MPLAB IDE/MPLAB X IDE发送出。

导入（Import）

将数据从外部源（例如，hex文件）传送到MPLAB IDE/MPLAB X IDE中。

递增调用（Recursive Calls）

直接或间接调用自身的函数。

递归（Recursion）

递归为一个已定义的函数或宏可以调用自身的概念。写递归的宏时务必小心；当没有从递归退出的出口时，很容易陷入无限循环。

地址（Address）

标识存储器中位置的值。

段（Section）

OCG psect的GCC等效形式。被链接器视为整体的代码块或数据块。

段属性（Section Attribute）

赋予某个段（例如access段）的GCC特性。

断点（Breakpoint）

硬件断点：执行后将导致程序暂停的事件。

软件断点：固件将暂停执行处的地址。通常由特殊的中断指令获取。

堆（Heap）

用于动态存储器分配的存储区，其中存储器块的分配或释放按运行时确定的任意顺序进行。

E

EEPROM

电可擦除可编程只读存储器。一种可电擦除的特殊类型的 PROM。一次可写入或擦除一个字节的数据。即使关断电源，EEPROM仍能保留自身内容。

ELF

可执行和链接格式。此格式的目标文件包含机器码。使用 DWARF 指定调试及其他信息。ELF/DWARF能比COFF提供对优化代码的更好调试。

Epilogue

编译器生成的代码的一部分，负责释放堆栈空间、恢复寄存器和满足运行时模型中指定的任何其他机器特定要求。此代码在给定函数的任何用户代码后、紧接着函数返回之前执行。

EPROM

可擦除可编程只读存储器。通常可通过紫外线照射而擦除的可编程只读存储器。

二进制（Binary）

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

F

FNOP

强制性空操作。强制性 NOP 周期是双周期指令的第二个周期。由于 PIC 单片机采用流水线架构，所以当它执行当前指令时，会预取物理地址空间内的下一条指令。然而，如果当前指令更改了程序计数器，这条预取的指令将被明确忽略，引起一个强制性的NOP周期。

© 2014 Microchip Technology Inc.

Free-Standing

一种接受任何不使用复杂类型的严格符合程序的实现，而且在这种实现中，对库条款（ANSI 89 标准条款第 7 条）中规定的特性的使用，仅限于标准头文件 `<float.h>`、`<iso646.h>`、`<limits.h>`、`<stdarg.h>`、`<stdbool.h>`、`<stddef.h>` 和 `<stdint.h>`的内容。

访问入口点（Access Entry Point）

访问入口点提供了一种方法，可将跨段控制传送到链接时可能未定义的函数。它们支持独立链接引导应用程序段和安全应用程序段。

仿真/仿真器（Emulation/Emulator）

*请参见*ICE/ICD。

放宽（Relaxation）

将某一指令转换为相同但较小指令的过程。这对节省代码大小很有用。MPLAB XC32 目前知道如何将`CALL`指令放宽（`relax`）为`RCALL`指令。当被调用的符号距当前指令 +/- 32K个指令字以内时可以这样做。

非扩展模式（Non-Extended Mode）（PIC18 MCU）

在非扩展模式下，编译器不使用扩展指令，也不使用立即数偏移量变址寻址。

非实时（Non Real Time）

指处理器处于断点或正在单步执行指令或者MPLAB IDE/MPLAB X IDE正运行在软件模拟器模式下。

非易失性存储器（Non-Volatile Storage）

关断电源后，能保存自身内容的存储器件。

符号（Symbol）

符号是用于描述组成程序的各个部分的通用机制。这些部分包括函数名、变量名、段名、文件名、结构/枚举/联合标记名等。MPLAB IDE/MPLAB X IDE中的符号主要指变量名、函数名和汇编标号。链接后符号的值是其在存储器中的值。

符号，绝对（Symbol，Absolute）

立即值的表示，如通过汇编伪指令.equ所给出的定义。

G

GPR

通用寄存器。器件数据存储器（RAM）的一部分，用于常规用途。

高级语言（High Level Language）

用于编写程序的语言，它比汇编语言更加不依赖于具体的处理器。

跟踪（Trace）

记录程序执行的仿真器或软件模拟器功能。仿真器将程序执行记录到其跟踪缓冲区中，该缓冲区的内容将被上载到MPLAB IDE/MPLAB X IDE的trace（跟踪）窗口。

跟踪存储器（Trace Memory）

跟踪存储器包含在仿真器内。跟踪存储器有时也称为跟踪缓冲区。

跟踪宏（Trace Macro）

提供仿真器数据跟踪信息的宏。由于这是软件跟踪，必须将宏添加到代码，必须重新编译或重新汇编代码，而且必须先使用此代码对目标器件编程，跟踪才会生效。

工作簿（Workbook）

MPLAB SIM软件模拟器中用于生成SCL激励的一种设置。

工具栏（Tool Bar）

一行或一列图标，单击这些图标时将执行MPLAB IDE/MPLAB X IDE功能。

观察变量（Watch Variable）

调试会话期间可在Watches窗口中监控的变量。

归档/归档器（Archive/Archiver）

归档/库是可重定位目标模块的集合。通过将多个源文件汇编为目标文件，然后使用归档器/库管理器将这些目标文件组合为一个归档/库文件来创建。归档/库文件可以与目标模块和其他归档/库文件链接以生成可执行代码。

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

制订许多行业和技术（包括计算和通信）方面的标准的组织。也称为ISO。

过滤器（Filter）

通过选择确定在跟踪显示或数据文件中将包含/排斥哪些数据。

H

Hex代码\Hex文件（Hex Code\Hex File）

Hex代码是以十六进制格式代码存储的可执行指令。Hex代码包含在Hex文件中。

宏（Macro）

宏指令。以缩写形式表示指令序列的指令。

宏伪指令（Macro Directives）

在宏定义体内部控制执行和数据分配的伪指令。

环回测试板（Loop-Back Test Board）

环回测试板用于测试MPLAB REAL ICE在线仿真器的功能。

环境（Environment）

MPLAB PM3——包含关于如何编程器件的文件的文件夹。该文件夹可被传送到SD/MMC卡。

汇编/汇编器（Assembly/Assemble）

汇编是用符号形式描述二进制机器码的编程语言。汇编器是将汇编语言源代码翻译成机器码的语言工具。

© 2014 Microchip Technology Inc.

I

ICE/ICD

在线仿真器 / 在线调试器：用于对目标器件进行调试和编程的硬件工具。仿真器的功能多于调试器，如跟踪。

在线仿真/在线调试：使用在线仿真器或调试器进行仿真或调试的行为。

-ICE/-ICD：具有内置在线仿真或调试电路的器件（MCU或DSC）。此器件总是安装在仿真头板上，并用于通过在线仿真器或调试器进行调试。

ICSP

在线串行编程。一种使用串行通信及最少数量的器件引脚编程 Microchip 嵌入式器件的方法。

IDE

集成开发环境，如MPLAB IDE/MPLAB X IDE。

IEEE

国际电子电气工程师协会。

J

机器码（Machine Code）

处理器实际读取和解释的计算机程序表示。用二进制机器码编写的程序由一系列机器指令组成（可能还带有数据）。某个特定处理器的所有可用指令的集合称为它的“指令集”。

机器语言（Machine Language）

用于特定中央处理单元的一组指令，旨在无需翻译即可由处理器使用。

激励（Stimulus）

软件模拟器的输入，即，为模拟对外部信号的响应而生成的数据。通常这些数据以动作列表的形式放在文本文件中。激励可能是异步、同步（引脚）、时钟激励和寄存器激励。

交叉引用文件（Cross Reference File）

引用符号表的文件和引用符号的文件列表。如果符号已被定义，那么列出的第一个文件是该定义的位置。剩余的文件包含对该符号的引用。

校准存储区（Calibration Memory）

用来保存 PIC 单片机内置 RC 振荡器或其他器件外设的校准值的特殊功能寄存器或通用寄存器。

节点（Node）

MPLAB IDE/MPLAB X IDE 项目组件。

警告（Warning）

MPLAB IDE/MPLAB X IDE——提供警告，警告您发生了可能导致器件、软件文件或设备物理损坏的情况。

MPLAB XC16汇编器/编译器——警告能报告可能表明发生问题的情况，但并不停止处理。在 MPLAB XC16 C 编译器中，警告消息报告源文件名和行号，但其中带有文本“`warning:`”以与错误消息区别开来。

静态RAM或SRAM（Static RAM or SRAM）

静态随机访问存储器。可在目标板上读写的程序存储器，无需经常刷新。

局部标号（Local Label）

局部标号是用 LOCAL 伪指令在宏内部定义的标号。这些标号特定于宏实例化的一个给定实例。换句话说，当遇到ENDM宏之后，将不再能访问声明为local的符号和标号。

绝对段（Absolute Section）

具有链接器不能更改的固定（绝对）地址的GCC编译器段。

绝对变量/函数（Absolute Variable/Function）

使用OCG编译器的`@ address`语法放置在绝对地址的变量或函数。

K

看门狗定时器（Watchdog Timer，WDT）

PIC单片机上的一个定时器，它可在可选的时间长度之后复位处理器。可通过使用配置位使能、禁止或设置WDT。

可重定位（Relocatable）

其地址尚未被分配到存储器中固定位置的对象。

可重定位段（Relocatable Section）

MPLAB XC16汇编器——地址不固定（绝对）的段。链接器通过称为重定位的过程将地址分配给可重定位段。

可重入函数（Reentrant）

有多个同时活动的实例的函数。这可能是由于直接或间接递归或通过中断处理期间执行造成的。

可执行代码（Executable Code）

可被装载用以执行的软件。

控制伪指令（Control Directives）

汇编语言代码中的伪指令，它能根据指定表达式的汇编时值包含或省略代码。

库/库管理器（Library/Librarian）

*请参见*归档/归档器。

快速操作存储区（Access Memory）

仅限PIC18——PIC18器件中的一些特殊寄存器，对这些寄存器的访问与存储区选择寄存器（Bank Select Register， BSR）的设置无关。

扩展单片机模式（Extended Microcontroller Mode）

在扩展单片机模式下，片上程序存储器和外部存储器都可用。如果程序存储器的地址超出了PIC18器件的内部存储空间，程序执行将自动切换到外部存储器。

扩展模式（Extended Mode）（PIC18 MCU）

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

© 2014 Microchip Technology Inc.

L

LVDS

低压差分信号传输。一种通过铜线进行高速（每秒千兆位）数据传输的低噪声、低功耗和低幅值方法。

对于标准I/O信号，数据存储取决于实际电压大小。电压大小可能受到传输线长度的影响（线路越长，电阻就越高，这会使电压下降）。但对于LVDS，数据存储仅通过正负电压值区分，而不是实际电压。因此，数据可以传输更长的线路距离，同时保持干净、一致的数据流。

来自网址：http://www.webopedia.com/TERM/L/LVDS.html。

链接描述文件（Linker Script Files）

链接描述文件是链接器的命令文件。它们定义链接器选项并描述目标平台上的可用存储器。

链接器（Linker）

通过解析从一个模块到另一个模块的引用，将目标文件和库组合起来以生成可执行代码的语言工具。

列表伪指令（Listing Directives）

列表伪指令是那些控制汇编器列表文件格式的伪指令。它们允许指定标题、分页和其他列表控制。

列表文件（Listing File）

列表文件是一种ASCII文本文件，它显示了为每个C源语句、汇编指令、汇编器伪指令或在源文件中遇到的宏生成的机器码。

逻辑探针（Logic Probes）

最多14个逻辑探针可被连接到某些Microchip仿真器。逻辑探针提供外部跟踪输入、触发输出信号、+5V电压以及一个公共接地端。

M

Makefile

导出到某个文件，此指令用于Make项目。使用该文件可在MPLAB IDE/MPLAB X IDE外Make项目（即使用`make`）。

Make项目（Make Project）

重新编译应用程序的命令，仅编译自上次编译完成后更改了的源文件。

MCU

单片机器件。microcontroller的缩写，也写作uC。

MPASM™汇编器（MPASM™ Assembler）

Microchip Technology用于PIC单片机、KeeLoq®器件和Microchip存储器件的可重定位宏汇编器。

MPLAB ICD

Microchip的在线调试器，与MPLAB IDE/MPLAB X IDE配合使用。*请参见*ICE/ICD。

MPLAB IDE/MPLAB X IDE

Microchip 的集成开发环境。MPLAB IDE/MPLAB X IDE 随附编辑器、项目管理器和软件模拟器。

MPLAB PM3

Microchip 的器件编程器。用于编程 PIC18 单片机和 dsPIC 数字信号控制器。可与 MPLAB IDE/MPLAB X IDE配合使用，也可单独使用。取代PRO MATE II。

MPLAB REAL ICE™ 在线仿真器（MPLAB REAL ICE™ In-Circuit Emulator）

Microchip 的新一代在线仿真器，与 MPLAB IDE/MPLAB X IDE 配合使用。*请参见* ICE/ICD。

MPLAB SIM

Microchip的软件模拟器，可与MPLAB IDE/MPLAB X IDE配合使用，支持PIC MCU和dsPIC DSC器件。

MPLIB™ 目标库管理器（MPLIB™ Object Librarian）

Microchip的库管理器，可与MPLAB IDE/MPLAB X IDE配合使用。MPLIB库管理器是一种目标库管理器，用于组合目标模块，目标模块是使用 MPASM 汇编器（mpasmx）或MPLAB XC8 C编译器创建的。

MPLINK™ 目标链接器（MPLINK™ Object Linker）

MPLINK链接器是Microchip MPASM汇编器和Microchip MPLAB XC8 C编译器的目标链接器。MPLINK 链接器还可以与Microchip MPLIB库管理器配合使用。MPLINK 链接器设计为与MPLAB IDE/MPLAB X IDE配合使用，但是它并不一定要与MPLAB IDE/MPLAB X IDE配合使用。

MRU

最近使用过的。指可从MPLAB IDE/MPLAB X IDE主下拉菜单中选择的文件和窗口。

命令行接口（Command Line Interface）

仅基于文本输入和输出，在程序和其用户之间进行通信的一种方式。

模板（Template）

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

模块（Module）

执行预处理器伪指令后，源文件的预处理输出。也称为转换单元。

目标（Target）

指用户硬件。

目标板（Target Board）

构成目标应用的电路和可编程器件。

目标处理器（Target Processor）

目标应用板上的单片机器件。

© 2014 Microchip Technology Inc.

目标代码/目标文件（Object Code/Object File）

目标代码是汇编器或编译器生成的机器码。目标文件是包含机器码（可能还有调试信息）的文件。它可以直接执行；或为可重定位的，需要与其他目标文件（如库文件）链接，以生成完整的可执行程序。

目标文件伪指令（Object File Directives）

仅当创建目标文件时使用的伪指令。

目标应用程序（Target Application）

目标板上的软件。

N

NOP

空操作。除了让程序计数器加1外，没有其他作用的指令。

内部链接（Internal Linkage）

如果某个函数或变量不可以从定义它的模块外访问，则称该函数或变量有内部链接。

匿名结构（Anonymous Structure）

MPLAB XC16 C编译器——未命名的结构。

MPLAB XC8 C编译器——作为C联合成员的未命名结构。可以像访问包含各结构的联合的成员一样访问匿名结构的成员。例如，在下列代码中，`hi` 和 `lo` 都是联合 `caster` 中匿名结构的成员。

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

O

OTP

一次可编程。非窗口封装的 EPROM 器件。由于 EPROM 需要紫外线才能擦除其存储器，因此只有窗口式器件才可被擦除。

P

PC

个人计算机或程序计数器。

PC主机（PC Host）

任何运行受支持的Windows操作系统的PC。

PIC MCU

PIC单片机（MCU）是指所有的Microchip单片机系列。

PICkit 2和3

Microchip的器件开发编程器，通过Debug Express实现调试功能。要了解支持哪些器件，请参见各工具的自述文件。

Pragma

对特定编译器有意义的伪指令。通常 pragma 用于将实现定义的信息传递给编译器。MPLAB XC16 C编译器使用属性传递该信息。

Prologue

编译器生成的代码的一部分，负责分配堆栈空间、保存寄存器和满足运行时模型中指定的任何其他机器特定要求。在给定函数中，此代码在所有用户代码之前执行。

Psect

GCC段的OCG等效形式，程序段的简称。被链接器视为整体的代码块或数据块。

PWM信号（PWM Signals）

脉宽调制信号。某些PIC MCU器件具有PWM外设。

跑表（Stopwatch）

测量执行周期的计数器。

配置位（Configuration Bits）

可通过编程来设置PIC单片机工作模式的专用位。配置位可预编程也可不预编程。

配置文件（Profile）

在MPLAB SIM软件模拟器中，是寄存器已执行激励的汇总列表。

片外存储器（Off-Chip Memory）

片外存储器是指PIC18器件的一种存储器选择选项，存储器可驻留在目标板上，也可由仿真器提供所有的程序存储器。访问 *Options>Development Mode*（选项 > 开发模式）的**Memory**（存储器）选项卡可打开Off-Chip Memory selection（片外存储器选择）对话框。

Q

器件编程器（Device Programmer）

用来对可电编程的半导体器件（如单片机）进行编程的工具。

嵌套深度（Nesting Depth）

一个宏可以包含其他宏的最大层数。

清除（Clean）

清除操作删除所有中间项目文件，例如活动项目的目标文件、hex文件和调试文件。项目编译时会通过其他文件重新创建这些文件。

R

Radix（基数）

基数，十六进制或十进制，用于指定地址。

RAM

随机访问存储器（数据存储器）。可以任意顺序访问其中信息的存储器。

© 2014 Microchip Technology Inc.

ROM

只读存储器（程序存储器）。不能被修改的存储器。

软件堆栈（Stack, Software）

供应用程序用来存储返回地址、函数参数和局部变量的存储器。此存储器由程序中的指令在运行时动态分配。它支持可重入函数调用。

软件模拟器（Simulator）

模拟器件操作的软件程序。

S

Shell

MPASM汇编器shell是宏汇编器的提示性输入接口。有两个MPASM汇编器shell：一个针对DOS版本，一个针对Windows版本。

Skew

不同时间出现在处理器总线上与指令执行相关的信息。例如，在执行前一条指令的过程中取指时，被执行的操作码出现在总线上；当实际执行该操作码时，总线上出现源数据地址及其值以及目标数据地址。而目标数据值在下一条指令执行时出现在总线上。跟踪缓冲区一次捕捉总线上的这些信息。因此，跟踪缓冲区的一条记录将包含三条指令的执行信息。执行一条指令时，从一条信息到另一条信息之间捕捉的周期数称为Skew。

Skid

当使用硬件断点来暂停处理器时，在处理器暂停之前可能再执行一条或多条额外的指令。在预期断点之后执行的额外指令数称为Skid。

SQTP

*请参见*带序列号的快速批量编程（Serialized Quick Turn Programming）。

三字母组合（Trigraphs）

ISO C定义的、用于代替单个字符的三字符序列，均以??开头。

闪存（Flash）

一种EEPROM，其数据以块形式而不是以字节形式写入或擦除。

上电复位仿真（Power-on-Reset Emulation）

在数据RAM区域中写入随机值以模拟最初给应用上电时RAM中未初始化的值的软件随机过程。

上载（Upload）

上载功能将数据从一个工具（如仿真器或编程器）传输到主机 PC 或者从目标板传输到仿真器。

生产编程器（Production Programmer）

生产编程器是一种编程工具，其中含有可对器件进行快速编程的资源。它能在各种电压下进行编程，并完全符合编程规范。在由于应用电路要经过装配线而使时间极为重要的生产环境中，尽可能快地对器件编程最为重要。

十六进制（Hexadecimal）

使用数字0-9加上字母A-F（或a-f）的以16为基数的计数体制。数字A-F表示（十进制中）值为10-15的十六进制数。最右边的数字表示1的倍数，右侧第二位数字表示16的倍数，右侧第三位数字表示16^2 = 256，以此类推。

实时（Real-Time）

当在线仿真器或调试器从暂停状态释放时，处理器将在实时模式下运行并且与芯片的正常操作完全相同。在实时模式下，仿真器的实时跟踪缓冲区被使能并持续捕捉所有选择的周期，同时允许所有的中断逻辑。在在线仿真器或调试器中，处理器将实时执行直到由于遇到有效的断点而暂停，或者直到用户暂停程序执行。

对于软件模拟器，实时只不过意味着单片机指令的执行速度与主机 CPU 模拟这些指令的速度相同。

事件（Event）

对总线周期的描述，可能包括地址、数据、次数计数、外部输入、周期类型（取指或读/写）以及时间标记。事件可用来描述触发、断点和中断。

书签（Bookmarks）

使用书签可轻松地查找文件中的指定行。

在Editor（编辑器）工具栏中选择Toggle Bookmarks（开关书签）可添加/删除书签。单击该工具栏上的其他图标可移动到下一个或上一个书签。

属性（Attribute）

C程序中用来描述特定于机器的属性的变量或函数的GCC特性。

属性，段（Attribute, Section）

段的GCC特性，如“可执行”、“只读”或“数据”，它们可在汇编器`.section`伪指令中指定为标志。

数据存储器（Data Memory）

在Microchip MCU和DSC器件上，数据存储器（RAM）由通用寄存器（General Purpose Register，GPR）和特殊功能寄存器（Special Function Register，SFR）组成。有些器件还具有EEPROM数据存储器。

数据监视和控制界面（Data Monitor and Control Interface，DMCI）

数据监视和控制界面（DMCI）是MPLAB X IDE中的工具。该界面提供对项目中应用程序变量的动态输入控制。可使用4个可动态分配的图形窗口中的任一个以图形方式查看应用程序生成的数据。

数据伪指令（Data Directives）

数据伪指令是那些控制汇编器对程序或数据存储器进行分配的指示性语句，它提供了用符号（即有意义的名称）引用数据项的方法。

数字信号处理\数字信号处理器（Digital Signal Processing\Digital Signal Processor）

数字信号处理（DSP）是对通常由模拟信号（声音或图像）转换成数字形式（采样）的数字信号进行的计算机处理。数字信号处理器是为数字信号处理而设计的微处理器。

© 2014 Microchip Technology Inc.

数字信号控制器（Digital Signal Controller）

数字信号控制器（DSC）是带数字信号处理功能的单片机器件，即Microchip dsPIC DSC器件。

顺序断点（Sequenced Breakpoints）

按顺序发生的断点。断点执行顺序是从下到上的；最先执行序列中的最后一个断点。

T

特殊功能寄存器（Special Function Register，SFR）

数据存储器（RAM）的一部分，专用于控制 I/O 处理功能、I/O 状态、定时器或者其他模式或外设的寄存器。

条件编译（Conditional Compilation）

仅在预处理器伪指令指定的某个常量表达式为true时编译程序段的操作。

条件汇编（Conditional Assembly）

根据指定表达式汇编时的值包含或忽略的汇编语言代码。

调试/调试器（Debug/Debugger）

*请参见*ICE/ICD。

调试信息（Debugging Information）

编译器和汇编器选项，选中时提供调试应用代码所需的多种信息。关于选择调试选项的详细信息，请参见编译器或汇编器文档。

U

USB

通用串行总线。是计算机和外设之间以双串行传输方式在电缆上通信的外设接口标准。USB 1.0/1.1支持12 Mbps的数据传输速率。USB 2.0（也称为高速USB）支持高达480 Mbps的数据传输速率。

V

Volatile

一种变量限定符，用于阻止编译器应用会影响存储器中变量访问方式的优化。

W

Watches窗口（Watches Window）

Watches窗口包含一系列观察变量，这些变量在每次执行到断点时更新。

外部RAM（External RAM）

片外读/写存储器。

外部标号（External Label）

有外部链接的标号。

外部符号（External Symbol）

有外部链接的标识符符号。这可能是一个引用，也可能是一个定义。

外部符号解析（External Symbol Resolution）

由链接器执行的过程，在该过程中将收集来自所有输入模块的外部符号定义以尝试解析所有的外部符号引用。任何没有对应定义的外部符号引用将会导致报告发生链接器错误。

外部链接（External Linkage）

如果某个函数或变量可以从定义它的模块外被引用，则称该函数或变量有外部链接。

外部输入线（External Input Line）

用于根据外部信号设置事件的外部输入信号逻辑探针线（TRIGIN）。

微处理器模式（Microprocessor Mode）

PIC18单片机的一种可能的程序存储器配置。在微处理器模式下，不使用片内程序存储器。整个程序存储器映射到外部。

伪指令（Directives）

源代码中提供对语言工具的操作进行控制的语句。

未初始化的数据（Uninitialized Data）

未定义初始值的数据。在C中，

```
int myVar;
```

定义了将存放在未初始化数据段中的变量。

未分配段（Unassigned Section）

在链接器命令文件中尚未被分配给特定目标存储器块的段。链接器必须找到一个可将未分配段分配到的目标存储块。

文件寄存器（File Registers）

片内数据存储器，包括通用寄存器（GPR）和特殊功能寄存器（SFR）。

X

系统窗口控件（System Window Control）

系统窗口控件位于窗口和某些对话框的右上角。单击该控件时通常会弹出包含“Minimize”（最小化）、“Maximize”（最大化）和“Close”（关闭）项的菜单。

下载（Download）

下载是将数据从主机传送到另一个器件（如仿真器、编程器或目标板）的过程。

限定符（Qualifier）

次数计数器使用的地址或地址范围，或作为复杂触发中发生在另一个操作前的事件。

响应延时（Latency）

事件与其得到响应之间的时间间隔。

向量（Vector）

复位或中断发生时应用程序将跳转到的存储器地址。

项目（Project）

项目包含编译应用程序（源代码和链接描述文件等）所需的文件及其与各种编译工具和编译选项的关联。

© 2014 Microchip Technology Inc.

消息（Message）

显示出来的文本，警告在语言工具的操作中可能存在的问题。消息不会停止操作。

小尾数法（Little Endian）

是多字节数据的数据存储顺序方案，最低有效字节存储在低地址中。

修正（Fixup）

链接器重定位后将目标文件符号引用替换为绝对地址的过程。

虚拟字节（Phantom Byte）

dsPIC 架构中未实现的字节，用于将 24 位指令字当作 32 位指令字处理。虚拟字节出现在dsPIC hex文件中。

Y

严重错误（Fatal Error）

导致编译立即停止的错误。不产生进一步的消息。

样机系统（Prototype System）

指用户的目标应用或目标板的术语。

已编译堆栈（Compiled Stack）

编译器管理的存储区，在其中为变量静态分配空间。当软件堆栈或硬件堆栈机制无法在目标器件上高效实现时，将替换为已编译堆栈。

已编译堆栈（Stack, Compiled）

编译器管理和分配的存储区，在其中为变量的静态分配空间。当软件堆栈等机制无法在目标器件上高效实现时，将替换为已编译堆栈。它会阻止重入。

已初始化数据（Initialized Data）

以一个初始值定义的数据。在C 中，

```
int myVar=5;
```

定义了将存放在已初始化数据段中的变量。

已分配段（Assigned Section）

在链接器命令文件中已分配到目标存储器块的GCC编译器段。

已弃用的功能（Deprecated Features）

由于历史原因仍支持，但将逐步被淘汰而不再使用的功能。

异步（Asynchronously）

不同时发生的多个事件。通常用于指处理器执行期间可能会发生的中断。

异步激励（Asynchronous Stimulus）

为模拟到被模拟器件的外部输入而生成的数据。

应用（Application）

可以由PIC® 单片机控制的一组软件和硬件。

硬件堆栈（Stack, Hardware）

当调用函数时，在PIC单片机中存储返回地址的存储单元。

用于*器件*的MPLAB入门工具包（MPLAB Starter Kit for *Device*）

Microchip 的入门工具包中包含开始探索指定器件所需的全部内容。查看正在工作的应用程序，然后对您自己的更改进行调试和编程。

用于*器件*的MPLAB *语言工具*（MPLAB *Language Tool* for *Device*）

用于指定器件的 Microchip 的 C 编译器、汇编器和链接器。根据您用于应用的器件来选择语言工具的类型，例如，如果要为 PIC18 MCU 创建 C 代码，那么选择用于 MPLAB XC8 C编译器。

优先级（Precedence）

定义表达式中求值顺序的规则。

与断点（ANDed Breakpoints）

设置断点的“与”条件（即在程序暂停之前必须同时发生断点 1 和断点 2）。只有数据断点和程序存储器断点同时发生时才能实现此操作。

原始数据（Raw Data）

与段相关的代码或数据的二进制表示。

源代码（Source Code）

由编程人员编写的计算机程序的形式。源代码以某种正规的编程语言编写，可被翻译成机器码或由解释程序执行。

源文件（Source File）

包含源代码的ASCII文本文件。

运算符（Operators）

加号“+”和减号“-”之类的符号，它们在构成定义明确的表达式时使用。每个运算符都有已分配的优先级，用来决定求值的顺序。

运行（Run）

将仿真器从暂停释放以允许它实时运行应用程序代码并更改或响应I/O的命令。

运行时观察（Runtime Watch）

一个Watches窗口，在该窗口中，变量随应用程序运行而更改。可参见相应的工具文档，以确定如何设置运行时观察。不是所有的工具都支持运行时观察。

运行时模型（Run-time Model）

描述目标架构资源的使用。

Z

暂停（Halt）

停止程序执行。执行暂停与在断点处停止相同。

帧指针（Frame Pointer）

指向堆栈中地址的指针，它将基于堆栈的参数和基于堆栈的局部变量分隔开。为访问当前函数的局部变量和其他值提供了一个便利的方法。

只读存储器（Read Only Memory）

允许快速访问永久存储的数据但阻止添加或修改数据的存储硬件。

指令（Instructions）

告知中央处理单元执行特定操作并可包含操作中要使用的数据的位序列。

© 2014 Microchip Technology Inc.

指令集（Instruction Set）

特定处理器能够理解的机器语言指令的集合。

中断（Interrupt）

传递到CPU的信号，它使CPU暂停执行正在运行的应用程序，把控制权转交给中断服务程序（Interrupt Service Routine，ISR），从而能对该事件进行处理。ISR 结束后，应用程序将继续正常执行。

中断处理程序（Interrupt Handler）

发生中断时，处理特殊代码的程序。

中断服务程序（Interrupt Service Routine，ISR）

语言工具——用于处理中断的函数。

MPLAB IDE/MPLAB X IDE——在产生中断时进入的用户生成的代码。代码在程序存储器中的位置通常取决于所产生的中断类型。

中断服务请求（Interrupt Service Request，IRQ）

使处理器暂停正常的指令执行并开始执行中断处理程序的事件。某些处理器具备几种中断请求事件，以允许产生不同优先级的中断。

中断向量（Interrupt Vector）

中断服务程序或中断处理程序的地址。

中央处理单元（Central Processing Unit）

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

主机（Pod）

在线仿真器或调试器的外壳。其他名称还有“Puck”（如果外壳是圆的）和“Probe”（不要与逻辑探头混淆）。

助记符（Mnemonics）

可以直接翻译成机器码的文本指令。也称为操作码。

状态栏（Status Bar）

状态栏位于MPLAB IDE/MPLAB X IDE窗口的底部，指示诸如光标位置、开发模式、器件和有效工具栏之类的当前信息。

字节存储顺序（Endianness）

多字节对象中字节的存储顺序。

字母数字字符（Alphanumeric）

字母数字字符由字母字符和十进制数字（0,1, …,9）组成。

字母字符（Alphabetic Character）

字母字符指属于罗马字母表（a, b, …, z, A, B, …, Z）中字母的字符。

左值（L-value）

引用可被检查和/或修改的对象的表达式。l-value表达式用于赋值的左侧。

注：

© 2014 Microchip Technology Inc.

索引

符号

.abort....76
.align....66
.ascii....62
.asciz....62
.bss....58
.bss段....111, 132
.byte....62
.comm....64
.comm symbol, length....64
.data....58
.data段....111
.double....62
.eject....68
.else....69
.elseif....69
.end....77
.endif....69
.endm....72
.endr....71, 74
.ent....77
.equ....65
.equiv....65
.err....76
.error....76
.exitm....71, 72
.extern....64
.fail....76
.file....77
.fill....66
.float....63
.fmask....77
.frame....77
.global....64
.hword....63
.ident....76
.if....69
.ifc....69
.ifdecl....69
.ifeq....69
.ifeqs....69
.ifge....69
.ifgt....70
.ifle....70
.iflt....70
.ifnc....70
.ifndef....70
.ifnes....70
.ifnotdef....70
.incbin....75
.include....39, 42, 75
.int....63
.irp....71
.irpc....71
.lcomm....64
.list....68
.loc....77
.long....63
.macro....72
.mask....78
.nolist....68
.org....67
.popsection....59
.print....76
.psize....68
.purgem....73
.pushsection....59
.rept....74
.sbttl....68
.section name....59, 200
.set at....79
.set autoextend....79
.set macro....79
.set mips16e....79
.set noat....79
.set noautoextend....79
.set nomacro....79
.set nomips16e....80
.set noreorder....80
.set reorder....80
.short....63
.single....63
.size....78
.sizeof.....55
.sleb128....78
.space....67
.startof.....55
.string....63
.struct....67
.text....61
.text段....112
.title....68
.type....78
.warning....76
.weak....64, 153
.version....76
.uleb128....78
.word....63
-a....25
-a=file....35
-ac....26
-ad....28

-ah 30
-al 32
-am 32
-an 34
-as 35
-d 108
-dc 108
-dp 108
-I 39
-i 111
-J 37
-L 38, 110
-l 110
-M 118
-Map 118
-MD 38
-nodefaultlibs 110
-nostartfiles 110
-nostdlib 110
-o 38, 110
-r 111
-S 111
-s 111
-T 111
-t 115
-Tbss 111
-Tdata 111
-Ttext 112
-u 112
-V 116
-v 37, 116
-W 37
-Ur 111
-x 109
-y 116
-Z 38
--check-sections 115
--cref 118
--defsym 39, 109
--defsym=_min_heap_size 113
--defsym=_min_stack_size 113
--discard-all 109
--discard-locals 109
--end-group 108
--fatal-warnings 37
--gc-sections 109
--help 37, 115
--keep-locals 38
--library 110
--library-path 110
--listing-cont-lines 36
--listing-lhs-width 35
--listing-lhs-width2 35
--listing-rhs-width 36
--no-check-sections 115
--no-warn-mismatch 115
--no-undefined 112
--output 110
--p PROC 110
--print-map 118
--relocateable 111
--report-mem 115
--retain-symbols-file 111
--script 111
--section-start 111
--start-group 108
--strip-all 111
--strip-debug 111
--target-help 37
--trace 115
--trace-symbol 116
--warn 37
--warn-common 116
--warn-once 117
--warn-section-align 117
--verbose 37, 116
--version 37, 116
--undefined 112
--wrap 112

A

a.out 19, 38, 110
ADDR 143
ALIGN 144
ar实用程序 175
ASCII字符集 201
ASSERT 138

B

bin2hex实用程序 184
包含文件伪指令
    .incbin 75
    .include 75
保留名称 51
变量，定位 165
编译输出文件 149
表达式 53
表达式，空 53
表达式，整数 53
表达式的段 143
标号 43, 51
标题 68
标题行 19

C

COPY 135
参数 46
常量 140
    浮点数 48
    数字 48
    整数 48
操作数 44
    基址寄存器+偏移量 45
    立即数值 45
    通用寄存器 44
程序存储器，定位和保留 166
处理流程
    汇编器 15
    库管理器 176
    链接器 97

© 2014 Microchip Technology Inc.

初始化伪指令
.ascii....62
.asciz....62
.byte....62
.double....62
.float....63
.hword....63
.int....63
.long....63
.short....63
.single....63
.string....63
.word....63

D

DEFINED....144
DSECT....135
代码控制伪指令
.set at....79
.set autoextend....79
.set macro....79
.set mips16e....79
.set noat....79
.set noautoextend....79
.set nomacro....79
.set nomips16e....80
.set noreorder....80
.set reorder....80
点变量....141
点符号....52
定义符号的伪指令
.equ....65
.equiv....65
读/写段....128
段伪指令
.bss....58
.data....58
.popsection....59
.pushsection....59
.section name....59, 200
.text....61
堆分配....157
堆栈分配....157

E

ENTRY....138
EXCLUDE_FILE....130
EXTERN....138
二进制文件....100

F

FORCE_COMMON_ALLOCATION....138
访问数据....55
分配存储区....148
浮点数....48
符号表....20, 35, 124, 144
符号名....140
副标题....19, 68
赋值....126

G

GPR....44
GROUP....125
概述
汇编器....15
链接器....97
高级语言源代码....19, 30
归档器....175
描述文件....180
命令行接口....178

H

函数，定位....165
汇编器
概述....15
命令行接口....23
伪指令....44, 57
源代码....19

I

INCLUDE....125
INFO....135
INPUT....125
ifne....70

J

基址寄存器+偏移量....45
计算绝对地址....149
加载存储器地址....124, 136, 144
加载输入文件....148
简单赋值....126
将输出段分配到存储区....151
解析符号....148

K

KEEP....144
K后缀....140
开头....19
看门狗定时器....212
可分配段....128
可执行段....128
可重定位....19
可重定位代码....164
客户支持....12
空表达式....53
空白....43
库管理器....175
描述文件....180
命令行接口....178
库文件....98

L

LENGTH....128
LMA....124, 136, 144
LOADADDR....144
垃圾收集....144
立即数....45
链接描述文件....119
表达式....140
概念....124
命令行信息....120

命令语言 124
其他命令 138
文件 98
文件命令 125
链接器
处理 147
分配 150
概述 97
命令行接口 105
示例 163
输出文件 100
文件扩展名 98
链接器处理过程 147
链接映射选项，链接器
-M 118
-Map 118
--cref 118
--print-map 118
列表输出选项，汇编器 25
-a=file 35
-ac 26
-ad 28
-ah 30
-al 32
-am 32
-an 34, 35
--listing-cont-lines 36
--listing-lhs-width 35
--listing-lhs-width2 35
--listing-rhs-width 36
列表文件 19

M

MAX 144
MEMORY命令 128
! 128
A 128
I 128
L 128
R 128
W 128
X 128
M后缀 140
Microchip网站 11
MIN 145
myMICROCHIP个性化通知客户服务 11
描述文件
库管理器 180
描述文件，归档器/库管理器
ADDLIB 180
ADDMOD 180
CLEAR 180
CREATE 180, 181
DELETE 181
DIRECTORY 181
END 181
EXTRACT 181
LIST 181
OPEN 180, 181
REPLACE 181
SAVE 180, 181
VERBOSE 181
命令行接口
归档器/库管理器 178
汇编器 23
链接器 105
命令行信息
链接描述文件 120
目标文件 19, 98

N

NEXT 145
NOLOAD 135
nm实用程序 185
内部预处理程序 42
内建函数 143
ADDR 143
ALIGN 144
DEFINED 144
KEEP 144
LOADADDR 144
MAX 144
MIN 145
NEXT 145
SIZEOF 145

O

ORG 128
ORIGIN 128
OVERLAY 135
OUTPUT 125
OUTPUT_FORMAT 139
objdump实用程序 188

P

PIC32MX中断
向量表 158
PROVIDE 127

Q

起始地址 55
其他链接描述文件命令
ASSERT 138
ENTRY 138
EXTERN 138
FORCE_COMMON_ALLOCATION 138
OUTPUT_FORMAT 139
TARGET 139
其他选项，汇编器
-I 39
--defsym 39
前缀运算符 53
求值 142
全局符号 153

© 2014 Microchip Technology Inc.

R

ranlib实用程序 191
弱符号 153

S

SEARCH_DIR 125
SECTIONS命令 130
SIZEOF 145
size实用程序 192
skip 67
STARTUP 125
strings实用程序 194
strip实用程序 195
声明符号的伪指令
- .comm 64
- .extern 64
- .global 64
- .globl 64
- .lcomm 64
- .weak 64

十六进制转换到十进制 202
实用程序 173
示例，链接器 163
输出端
- 存储区 136

输出段
- LMA 136
- 存储区 136
- 地址 133
- 丢弃 134
- 类型 135
  - COPY 135
  - INFO 135
  - NOLOAD 135
  - OVERLAY 135
- 数据 134
- 说明 133
- 填充 136

输出格式，xc32-nm
- ? 187
- A 187
- B 187
- C 187
- D 187
- N 187
- R 187
- T 187
- U 187
- V 187
- W 187

输出列表伪指令
- .eject 68
- .list 68
- .nolist 68
- .psize 68
- .sbttl 68
- .title 68

输出文件创建选项，汇编器
- -L 38
- -MD 38
- -o 38
- -Z 38
- --keep-locals 38

输出文件选项，链接器 108
- -d 108
- -dc 108
- -dp 108
- -i 111
- -L 110
- -l 110
- -nostartfiles 110
- -nostdlib 110
- -o 110
- -r 111
- -S 111
- -s 111
- -T 111
- -Tbss 111
- -Tdata 111
- -Ttext 112
- -u 112
- -Ur 111
- -x 109
- --defsym 109
- --discard-all 109
- --discard-locals 109
- --end-group 108
- --gc-sections 109
- --library 110
- --library-path 110
- -nodefaultlibs 110
- --no-undefined 112
- --output 110
- --relocateable 111
- --retain-symbols-file 111
- --script 111
- --section-start 111
- --start-group 108
- --strip-all 111
- --strip-debug 111
- --undefined 112
- --wrap 112

输入段
- 共用符号 132
- 示例 132
- 通配符模式 131

数字常量 48

T

TARGET 139
特殊运算符 55
- .endof.(name) 55
- .sizeof 55
- .startof 55

替换/扩展伪指令
.endm....72
.endr....71, 74
.exitm....71, 72
.irpc....71
.macro....72
.purgem....73
.rept....74
/@....73
irp....71
条件汇编伪指令
.else....69
.elseif....69
.endif....69
.if....69
.ifc....69
.ifdecl....69
.ifeq....69
.ifeqs....69
.ifge....69
.ifgt....70
.ifle....70
.iflt....70
.ifnc....70
.ifndef....70
.ifne....70
.ifnes....70
.ifnotdef....70
调试信息伪指令
.end....77
.ent....77
.file....77
.fmask....77
.frame....77
.loc....77
.mask....78
.size....78
.sleb128....78
.type....78
.uleb128....78
推荐读物....10

U

USB....219

V

VMA....124, 136

W

位置计数器....141
位置计数器伪指令
.align....66
.fill....66
.org....67
.skip....67
.space....67
.struct....67
伪指令....44
初始化....62
定义符号....65
段....58
对齐方式....66
汇编器....57
其他....76
声明符号....64
输出列表....68
替换/扩展....71
条件....69
调试信息....77
文档
编排....8
约定....9
文件
库....98
链接描述文件....98
链接器输出....100
列表....19
目标....98
映射....100
源文件....17
文件扩展名
汇编器....17, 98
链接器....98
文件命令，链接描述文件
GROUP....125
INCLUDE....125
INPUT....125
OUTPUT....125
SEARCH_DIR....125
STARTUP....125

X

xc32....189, 196
xc32-ar实用程序....175
xc32-bin2hex实用程序....184
xc32-nm实用程序....185
xc32-objdump实用程序....188
xc32-ranlib实用程序....191
xc32-size实用程序....192
xc32-strings实用程序....194
xc32-strip实用程序....195
信息输出选项，汇编器
-J....37
-v....37
-W....37
--fatal-warnings....37
--help....37
--no-warn....37
--target-help....37
--warn....37
--verbose....37
--version....37
信息输出选项，链接器
-t....115
-V....116
-v....116
-y....116
--check-sections....115
--help....115
--no-check-sections....115
--no-warn-mismatch....115
--report-mem....115

© 2014 Microchip Technology Inc.

--trace 115
--trace-symbol 116
--warn-common 116
--warn-once 117
--warn-section-align 117
--verbose 116
--version 116
修饰选项，归档器/库管理器
a 179
b 179
c 179
f 179
i 179
l 179
N 179
o 179
P 179
S 179
s 179
u 179
V 179
v 179
虚拟存储器地址 124, 136
选项，归档器/库管理器
d 178
m 178
p 178
q 178
r 178
t 178
x 178
选项，汇编器
符号定义 39
列表输出 25
输出文件创建 38
搜索路径 39
信息输出 37
选项，链接器
链接映射输出 118
输出文件创建 108
信息输出 115
运行时初始化 113
选项，pic320-objdump--disassemble-zeroes 190
选项，pic32-nm--no-sort 186
选项，xc32-nm
-A 186
-a 186
-B 186
-f 186
-g 186
-l 186
-n 186
-o 186
-P 186
-p 186
-r 186
-s 186
-t 186
-u 186
-V 186
-v 186
--debug-syms 186
--defined-only 186
--extern-only 186
--format= 186
--help 186
--line-numbers 186
--numeric-sort 186
--portability 186
--print-armap 186
--print-file-name 186
--radix= 186
--reverse-sort 186
--size-sort 186
--version 186
--undefined-only 186
选项，xc32-objdump
-a 189
-D 189
-d 189
-f 189
-g 189
-H 189
-h 189
-j 189
-l 189
-M 189
-r 189
-S 190
-s 189
-t 190
-V 190
-w 190
-x 190
-z 190
--all-header 190
--archive-header 189
--debugging 189
--disassemble 189
--disassemble-all 189
--disassembler-options= 189
--file-header 189
--file-start-context 189
--full-contents 189
--header 189
--help 189
--line-numbers 189
--no-show-raw-insn 190
--prefix-addresses 189
--reloc 189
--section= 189
--section-header 189
--show-raw-insn 190
--source 190
--start-address= 190
--stop-address= 190
--syms 190
--version 190
--wide 190

选项，xc32-ranlib
-V ... 191
-v ... 191
--version ... 191
选项，xc32-strings
- ... 194
-a ... 194
-f ... 194
-n ... 194
-t ... 194
-v ... 194
--all ... 194
--bytes= ... 194
--help ... 194
--print-file-name ... 194
--radix= ... 194
--version ... 194
选项，xc32-strip
-K ... 196
-g ... 196
-N ... 196
-o ... 196
-p ... 196
-R ... 196
-S ... 196
-s ... 196
-V ... 196
-v ... 196
-X ... 196
-x ... 196
--discard-all ... 196
--discard-locals ... 196
--help ... 196
--keep-symbol= ... 196
--remove-section= ... 196
--preserve-dates ... 196
--strip-all ... 196
--strip-debug ... 196
--strip-symbol= ... 196
--strip-unneeded ... 196
--verbose ... 196
--version ... 196

Y

已初始化段 ... 128
意义取反 ... 128
映射段 ... 150
映射文件 ... 100
预处理程序，内部 ... 42
语法
xc32-bin2hex ... 184
xc32-nm ... 185
xc32-objdump ... 188
xc32-ranlib ... 191
xc32-strings ... 194
xc32-strip ... 195
归档器/库管理器 ... 177
汇编器 ... 23
链接器 ... 106
语句格式 ... 43
源代码 ... 43
源文件 ... 17
运算符 ... 53, 142
前缀 ... 53
中缀 ... 54
运算符优先级 ... 54, 142
运行时初始化选项，链接器
--defsym=_min_heap_size ... 113
--defsym=_min_stack_size ... 113

Z

诊断控制伪指令
.abort ... 76
.err ... 76
.error ... 76
.fail ... 76
.ident ... 76
.print ... 76
.warning ... 76
.version ... 76
整数 ... 48
整数表达式 ... 53
只读段 ... 128
中缀运算符 ... 54
助记符 ... 43
注释 ... 47, 124
转义字符 ... 49
字符 ... 49
字符常量 ... 49
字符串 ... 49

© 2014 Microchip Technology Inc.

注：

全球销售及服务网点

美洲

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

亚特兰大 **Atlanta**
Duluth, GA
Tel: 1-678-957-9614
Fax: 1-678-957-1455

奥斯汀 **Austin, TX**
Tel: 1-512-257-3370

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

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

克里夫兰 **Cleveland**
Independence, OH
Tel: 1-216-447-0464
Fax: 1-216-447-0643

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

底特律 **Detroit**
Novi, MI
Tel: 1-248-848-4000

休斯敦 **Houston, TX**
Tel: 1-281-894-5983

印第安纳波利斯 **Indianapolis**
Noblesville, IN
Tel: 1-317-773-8323
Fax: 1-317-773-5453

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

纽约 **New York, NY**
Tel: 1-631-435-6000

圣何塞 **San Jose, CA**
Tel: 1-408-735-9110

加拿大多伦多 **Toronto**
Tel: 1-905-673-0699
Fax: 1-905-673-6509

亚太地区

亚太总部 **Asia Pacific Office**
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

中国 - 北京
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

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

中国 - 重庆
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

中国 - 杭州
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

中国 - 香港特别行政区
Tel: 852-2943-5100
Fax: 852-2401-3431

中国 - 南京
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

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

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

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

中国 - 深圳
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

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

中国 - 西安
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

中国 - 厦门
Tel: 86-592-238-8138
Fax: 86-592-238-8130

中国 - 珠海
Tel: 86-756-321-0040
Fax: 86-756-321-0049

亚太地区

台湾地区 - 高雄
Tel: 886-7-213-7830

台湾地区 - 台北
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

台湾地区 - 新竹
Tel: 886-3-5778-366
Fax: 886-3-5770-955

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

印度 **India - Bangalore**
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

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

印度 **India - Pune**
Tel: 91-20-3019-1500

日本 **Japan - Osaka**
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

日本 **Japan - Tokyo**
Tel: 81-3-6880-3770
Fax: 81-3-6880-3771

韩国 **Korea - Daegu**
Tel: 82-53-744-4301
Fax: 82-53-744-4302

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

马来西亚 **Malaysia - Kuala Lumpur**
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

马来西亚 **Malaysia - Penang**
Tel: 60-4-227-8870
Fax: 60-4-227-4068

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

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

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

欧洲

奥地利 **Austria - Wels**
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

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

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

德国 **Germany - Dusseldorf**
Tel: 49-2129-3766400

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

德国 **Germany - Pforzheim**
Tel: 49-7231-424750

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

意大利 **Italy - Venice**
Tel: 39-049-7625286

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

波兰 **Poland - Warsaw**
Tel: 48-22-3325737

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

瑞典 **Sweden - Stockholm**
Tel: 46-8-5090-4654

英国 **UK - Wokingham**
Tel: 44-118-921-5800
Fax: 44-118-921-5820

03/25/14

© 2014 Microchip Technology Inc.