MPASM
用户指南

包括 MPLINK 和 MPLIB

目　录

引 言

介绍

在开始使用 MPASM，MPLINK，MPLIB 之前，看一下本章介绍的通用信息将很有好处。

重点

本章你会得到如下信息：

- 关于本指南
- 质量保证注册
- MICROCHIP 的互联网站
- 开发工具用户信息更新服务
- 客户支持

关于本指南

文档资料安排如下：

本手册介绍如何使用 MPASM，MPLINK 和 MPLIB 来开发 PICmicro 系列微控制器的应用程序。所有的软件均包含于 MPLIB-IDE（INTEGRATED DEVELOPMENT ENVIRONMENT）集成开发环境软件包中。欲知 MPLAB 软件包的详细情节，请参考手册《MPLIB 用户指南》，MICROCHIP 文档资料 DS51025。

本指南的内容安排如下：

第一部分-MPASM 宏汇编

- **第 1 章 MPASM 概述** - 定义 MPASM，它能干什么以及它如何配合其他开发工具进行开发工作。
- **第 2 章 MPASM-安装与入门** - 描述如何安装 MPASM 并简述 MPASM 的使用。
- **第 3 章 基于 DOS 的 MPASM 汇编** - 描述如何通过命令行或 DOS SHELL 界面在 DOS 环境下使用 MPASM 进行开发工作。
- **第 4 章 基于 WINDOWS 和 MPLAB 的 MPASM 汇编** - 描述如何在 WINDOWS SHELL 界面或 MPLAB 环境下使用 MPASM 进行开发。
- **第 5 章 指示语言** - 描述 MPASM 编程语言，包括 STATEMENT，操作符，变量以及其他的 ELEMENT。
- **第 6 章 使用 MPASM 建立可重定位目标代码** - 描述如何使用 MPASM 与 MICROCHIP 的链接器 MPLINK 联合工作。
- **第 7 章 宏语言** - 描述如何使用 MPLAB 软件包内建的宏指令处理器
- **第 8 章 表达式语法与操作** - 指导如何在 MPASM 的源文件中使用复杂的表达式。
- **第 9 章 初始化代码范例** - 列出一些初始化 PIC16CXX，PIC17CXX，PIC18CXX 系列单片机的初始化程序。

第二部分 MPLINK 链接程序

- **第 1 章 MPLINK 概述** - 定义 MPLINK，它能干什么以及它如何配合其他开发工具进行开发工作。
- **第 2 章 MPLINK 的安装与入门** - 描述如何安装 MPASM 并简述 MPASM 的使用。

- **第 3 章　在 DOS 下使用 MPLINK** - 描述如何通过命令行指令在 DOS 下使用 MPLINK。
- **第 4 章　在 WINDOWS 下 MPLAB 软件包中使用 MPLINK** - 描述如何在 WINDOWS 的 DOS 状态或 MPLAB 环境下运行 MPLINK。
- **第 5 章　MPLINK 链接器命令** - 描述如何生成链接命令以及如何用这些命令控制链接器的操作。
- **第 6 章　链接器的处理过程** - 描述链接器如何处理文件。
- **第 7 章　应用范例 1** - 讲解如何把程序代码放到不同的存储区域，如何在 ROM 中放置数据表格，以及在 C 环境下如何配置设置寄存器。
- **第 8 章　应用范例 2** - 讲述如何为引导装载器（BOOT LOADER）划分存储器，如何编译将会调到外部 RAM 中运行的源代码。
- **第 9 章　应用范例 3** - 讲述如何访问存储器映射的外设，以及如何建立新的区段（SECTION）。
- **第 10 章　应用范例 4** - 讲述如何为程序应用手动分配 RAM。

第三部分　MPLIB

- **第 1 章　MPLIB 概述** - 定义 MPLIB，它能干什么以及它如何配合其他开发工具进行开发工作。
- **第 2 章　MPLIB 的安装与入门** - 描述如何安装 MPLIB 并简述 MPLIB 的使用。
- **第 3 章　使用 MPLIB** - 描述如何通过 DOS 命令行或 WINDOWS 的 DOS 方式运行 MPLIB。

附录

- **附录 A　16 进制文件格式** - 描述了可能用到的几种不同的 16 进制文件格式。
- **附录 B　快速参考** - PICmicro 单片机的指令系统，16 进制到 10 进制转换，ASCII 集。
- **附录 C　MPASM 错误/警告/信息** - 详细说明由 MPASM 产生的错误/警告/提示信息。
- **附录 D　MPLINK 错误/警告** - 详细说明由 MPLINK 产生的错误，警告信息。
- **附录 E　MPLIB 出错信息** - 详细说明由 MPLIB 产生的错误提示信息。
- **术语** - 在本指南中用到的术语。
- **索引** - 本手册中涉及到的术语（TERMS），特性（FEATURES）和章节（SECTIONS）。
- **环球销售服务网** - 列出 MICROCHIP 全球的销售服务点的电话，传真以及地址。

本指南中用到的一些表示符号：

本指南中用到了下列约定符号：

描述	说明	例子
代码		
Courier 字体	用户输入代码或例子	`#define ENIGMA`
尖括号 < >	变量，需要你填入参数	<标号>,<表达式>
方括号 [ ]	可选项	`MPASMWIN [main.asm]`
大括号{ } 竖号|	互斥选项；“或”选项	Errorlevel { 0 | 1 }
引号中的小写体	数据类型	“filename”
省略号 …	表示一下而不是一一列出与本例无关的文本项	List [“列表选项”,…,“列表选项”]
0xnnnn	表示 16 进制数	0xFFFF,0x007A
界面		
带下划线和右箭头的斜体字	表示菜单选项	*File>Save*
文本中的粗体字	表示一个按钮	**OK,Cancel**
尖括号中的大写字	表示一些特殊键	<TAB>,<ESC>
文档资料		
斜体字	参考书	*MPLAB User’s Guide*

资料更新

所有的文档资料都有过时的时候，本用户指南也不例外。因为 MPASM，MPLINK，MPLIB 及其他 MICROCHIP 的开发工具需要经常更新，以满足用户的要求。某些实际看到的对话栏和/或工具的使用说明可能与本指南有所不同。请到 MICROCHIP 的网站获取最新的文档资料。

质量保证注册

请填好登记卡并立即邮寄出来，这将保证您得到及时的软件升级，MICROCHIP 的网站上可得到这些更新软件。

推荐阅读

本指南讲述如何使用 MPASM，MPLINK 和 MPLAB。用户也可以在开发软件包里找到某一特定型号单片机的详细资料。

README.ASM，README.LKR

阅读 README 文档（ASCII 文档资料）了解最新的 MPASM 和 MPLINK 信息。该文档资料中包含了本指南没有提及的软件的升级信息。

MPLAB 用户指南（DS51025）

详细地介绍了 MPLAB-IDE 集成开发环境软件包的特性及安装以及编辑器和模拟器的使用。

技术资料 CD-ROM（DS00161）

这张 CD-ROM 中包含了 MICROCHIP 最新的 PICmicro 器件详细数据手册。请联系您最近的 MICROCHIP 办事处索取光碟。也可以从 WWW。MICROCHIP。COM 网站下载。

嵌入控制技术手册（DS00092 和 DS00167）第一卷和第二卷。

这两本手册包含丰富的单片机应用信息。请联系您最近的 MICROCHIP 办事处索取。（见手册后面附录）

这些手册中讲述的应用例子也可以从 MICROCHIP 的销售和办事处得到，还可以从

MICROCHIP 的网站上下载。

MICROSOFT WINDOWS 手册

本指南认为用户已熟悉 WINDOWS 的使用，市面有许多关于 WINDOWS 的书可以作为参考。

MICROCHIP 互联网站

MICROCHIP 在 INTERNET 上提供在线帮助，

MICROCHIP 把 INTERNET 作为一个方便地给用户提供文件和信息的手段。用户需要有 INTERNET 浏览器，比如 NETSCAPE NAVIGATOR 或者 MICROSOFT INTERNET EXPLORER。用户也可以从我们的 FTP 服务器上下载。

连接到 MICROCHIP 的 INTERNET 站点

你可以在你熟悉的浏览器中输入 MICROCHIP 的网站地址：

HTTP：//WWW.MICROCHIP.COM

(中文网站为： HTTP：//WWW.MICROCHIP.COM.CN – 译者)

若想连接到 FTP 服务器，请输入：

FTP：//FTP.MICROCHIP.COM

WEB 网和 FTP 站提供很多服务。用户可以下载最新的开发软件，数据手册，应用笔记，用户指南，技术文章和例程。还有许多 MICROCHIP 的商业信息，包括办事处，分销商。其他用户关心的信息还包括：

- MICROCHIP 最新发布消息
- 常见问题解答
- 设计指导与提示
- 器件资料勘误
- 工作机会
- MICROCHIP 咨询顾问成员名单
- 与其他和 MICROCHIP 产品相关网站的链接。
- 更多关于产品，开发系统，技术研讨会的信息。

开发系统用户最新信息提供服务

MICROCHIP 通过用户最新信息提供服务尽力使用户随时得知 MICROCHIP 的最新产品信息。一旦你购买以下产品，我们会随时 EMAIL 通知你关于软件升级，修改或关于产品系列及开发工具的勘误。其他服务请参见 MICROCHIP 网站。

开发系统产品包括：

- 编译器
- 仿真器
- 编程器
- MPLAB 软件包
- 其他工具

假如你对以上产品之一感兴趣的话，可以联系下列 EMAIL 地址订购：

listserv@mail.microchip.com

请使用如下格式：

```
subscribe <listname> yourname
```

给你一个范例：

```
subscribe mplab John Doe
```

假如你想撤消定单的话，可以 EMAIL 给以下地址：

listserc@mail.microchip.com

请使用如下格式：

```
unsubscribe <listname> yourname
```

给你一个范例：

```
unsubscribe mplab Jhon Doe
```

以下将一一介绍现有的开发系统。

仿真器：

MICROCHIP 最新的在线仿真器。包括 MPLAB-ICE，PICMASTER。

假如你对以上产品感兴趣的话，可以联系下列 EMAIL 地址订购：

listserv@mail.microchip.com

请使用如下格式：

```
subscribe emulators yourname
```

编程器：

MICROCHIP 最新的编程器，包括 PROMATE，PICSTART PLUS。

假如你对以上产品感兴趣的话，可以联系下列 EMAIL 地址订购：

listserv@mail.microchip.com

请使用如下格式：

```
subscribe programmers yourname
```

编译器：

MICROCHIP 最新的 C 编译器，链接器，汇编器的信息包括 MPLAB-C17，MPLAB-C18，MPLINK，MPASM 和库，MPLINK 的库 MPLIB。

假如你对以上产品感兴趣的话，可以联系下列 EMAIL 地址订购：

listserv@mail.microchip.com

请使用如下格式：

```
subscribe compilers yourname
```

MPLAB 集成开发软件包：

WINDOWS 集成开发环境 MPLAB 的最新信息，本产品集中于 MPLAB，MPLAB-SIM 模拟器，MPLAB 项目管理器，欲知通用编辑器，调试器。欲知编译器，链接器，汇编器的详细情况，请订阅编译器产品清单（COMPILER LIST）。欲知 MPLAB 模拟器的详细情况，可订阅模拟器产品清单。欲知 MPLAB 编程器的详细情况，可订阅编程器产品清单。

假如你对以上产品清单感兴趣的话，可以联系下列 EMAIL 地址订阅：

listserv@mail.microchip.com

请使用如下格式：

```
subscribe mplab yourname
```

其他开发工具：

MICROCHIP 提供的最新关于其他开发工具的信息。欲知 MPLAB 及其集成开发工具的详细情况。可联系其他 EMAIL 地址。

假如你对以上产品清单感兴趣的话，可以联系下列 EMAIL 地址订阅：

listserv@mail.microchip.com

请使用如下格式：

```
subscribe otools yourname
```

用户支持

使用 MICROCHIP 的产品可以通过以下渠道得到帮助：

- 分销商及代表处
- 当地销售处
- 现场应用工程师（FAE）
- 厂内应用工程师（CAE）

热线电话

用户可以打电话给分销尚，代表处，或现场应用工程师寻求帮助。当地销售处也可接受用户的求助。见手册后面的销售处电话和地址。联系厂内应用工程师（CAE）可打电话：

（602）786-7627

另外，一条专线电话专门为用户提供系统信息和升级消息。用户可以及时得到开发系统软件的最新版本清单。用户还可以得到如何得到最新升级软件的信息。

这些热线电话是：

1-800-755-2345 美国及加拿大的大部分地区

1-602-786-73-2 世界其他地区

第一部分 - MPASM 宏汇编

第1章 MPASM 概述

1．1 介绍

本章就 MPASM 及其功能做一个概述。

1．2 重点

本章内容包括：

- 什么是 MPASM？
- MPASM 是干什么的？
- 软件代码移植
- 兼容性问题

1．3 什么是 MPASM?

MPASM 是基于 DOS 或 WINDOWS 的应用程序，专门为 PICmicro 系列单片机开发，作为汇编调试工具。通常，MPASM 针对所有开发平台，包括宏汇编和应用函数。MPASM 对于硬件的要求是：IBM PC/AT 或兼容计算机，软件支持包括 MA-DOSV5.0 或以上版本，或者是 MICROSOFT 的 WINDOWS95/9 8/NT。

MPASM 支持 MICROCHIP 的所有 PICmicro 系列微控制器，存储器，数据安全类产品。

1．4 MPASM 能做什么？

MPASM 为 MICROCHIP 的 12-BIT，14-BIT，16-BIT 及增强 16-BIT 微控制器的汇编代码开发提供通用解决方案。值得一提的特性是：

- 所有 PICmicro 的指令系统
- 命令行接口
- 控制界面
- 丰富的指示语言
- 灵活的宏语言
- 兼容于 MPLAB

1．5 软件代码移植

由于 MPASM 是 PICmicro 系列产品的通用汇编器，所以给 PIC16C54 开发的软件可以很容易的迁移到 PIC16C71 上。这需要改变一些指令助记符，以适应不同的 PIC。（假设这两个应用所使用的寄存器及外设是相似的）。指示语言和宏语言将是相似的。

1．6 兼容性问题

MPASM 兼容于目前有的所有 MICROCHIP 开发系统。包括 MPLAB-SIM（PIC 单片机离线模拟器），MPLAB-ICE（PIC 单片机在线仿真器），
PRO MATE（MICROCHIP 通用编程器）和 PICSTART PLUS（MICROCHIP 低价位开发编程器）。

MPASM 保持较好的兼容性。尽管由于兼容的原因新型号 PIC 还支持一些旧语法，我们还是鼓励使用本指南中介绍的新代码来编程。

第2章 MPASM-安装与入门

2．1 介绍

本章指导你安装 MPASM 到你的 PC 上。简要介绍 MPASM 汇编器。

2．2 重点

本章内容包括：

- 安装
- 汇编器概述
- 汇编器输入/输出文件

2．3 安装

MPASM 有 3 种版本：

- DOS 版本，MPASM.EXE，运行于 DOS5.0 以上
- 扩展 DOS 版本，MPASM-DP.EXE
- WINDOWS3.X/95/98/NT 版本，MPASMWIN.EXE（推荐使用）

MPASM.EXE 也有命令行接口界面。尽管推荐使用 MPASMWIN.EXE，MPASM.EXE 也可以运行于 DOS 或 WINDOWS3.X 的 DOS 窗口。

MPASMWIN。EXE 有 WINDOWS SHELL 界面。MPASMWIN.EXE 可以运行于 WINDOWS3.X/95/98/NT。你可以在 MPLAB 下使用它。

假如你想在 MPLAB 环境下使用 MPASM，没必要单独安装 MPASM，因为 MPLAB 软件包安装完成后 MPASM 也已经装好了。参考《MPLAB 用户指南》获取更多安装信息。你既可以从我们提供的 CD-ROM，也可以从 WEB 站点上获取 MPLAB 软件包和用户指南。MPASM 文件以 ZIP 形式封装，安装时你应这样操作：

- 在硬盘上建立一个子目录，将文件拷到该目录下
- 用解压缩工具 WINZIP 或 PKZIP 将文件解压缩

2．4 汇编器概述

有两种方式使用 MPASM：

- 产生单片机可运行的绝对代码
- 产生目标代码，以便和其他单独编译或汇编的代码模块链接。

MPASM 的默认输出是绝对代码。这一过程见图 2-1。

当以这种方式汇编时，在源文件里要定义所有用到的参数，或者说已经完全包含在内了。假若过程没有错误，一个目标设备可以执行的 16 进制文件将生成。该 16 进制文件可以用编程器写入单片机中。

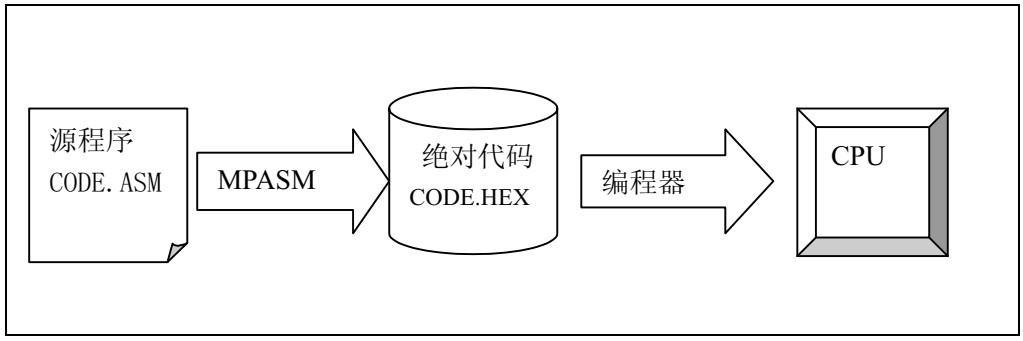
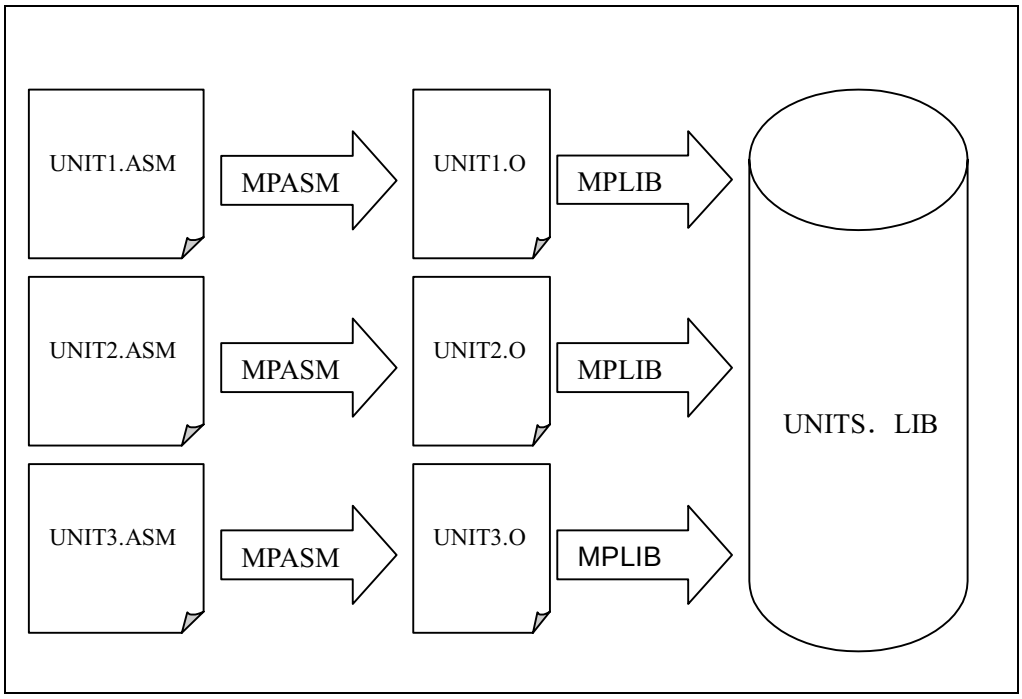


图 2-1：绝对代码的生成

MPASM 还可以产生目标代码，以便用 MICROCHIP 的 MPLINK 程序链接其他目标代码模块，生成一个可执行代码。这种方法产生的模块非常有用，我们就没有必要每次汇编时都去检查 这些模块就可直接使用。一些相关的模块可以组织起来，用 MPLIB 生成库文件。需要的库文件可以在链接的时候指明，而且在最后生成的可执行代码中仅包含了有用的例程。

这一过程可以以一个形象的图示来描述。见**图 2-2，2-3**。关于绝对和目标汇编的详细区别可以参见第 6 章。

图 2-2：建立可以重新使用的目标库



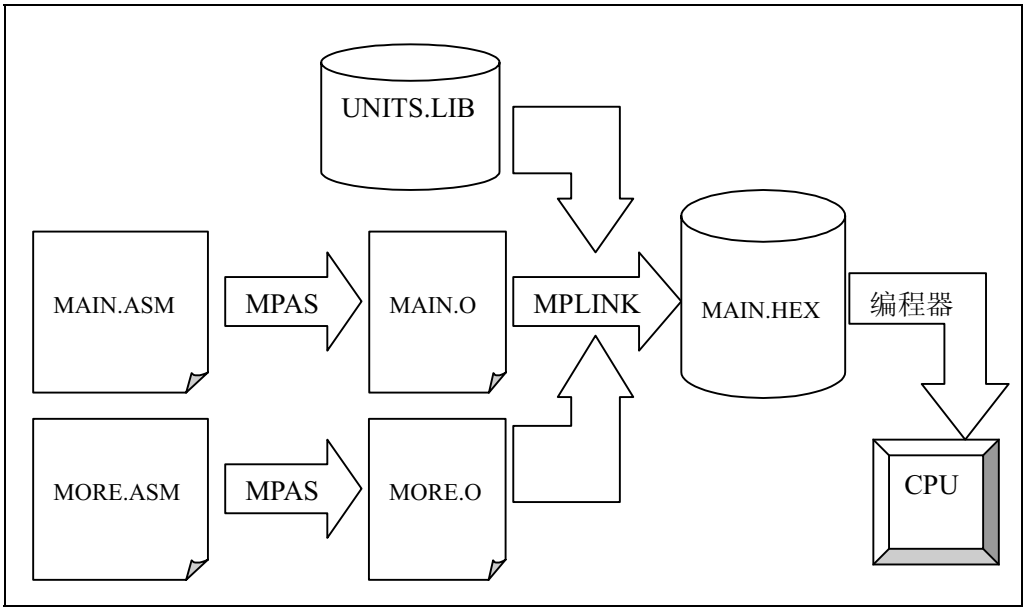


图 2-3：由目标文件生成可执行代码

2．5　汇编器输入 / 输出文件

以下是 MPASM 及相关的应用所使用的缺省文件扩展名列表.

表 2.1 MPASM 缺省扩展名

扩展名	用　　途
.ASM	输入 MPASM 中的缺省源文件扩展名:<源文件名>.ASM
.LST	从 MPASM 或 MPLINK 中产生的列表文件的缺省输出扩展名:<源文件名>.LST
.ERR	从 MPASM 中产生的特定出错文件的缺省输出扩展名:<源文件名>.ERR
.HEX	从 MPASM 或 MPLINK 中产生的 HEX 文件的输出扩展名:<源文件名>.HEX
.HXL/ .HXH	从 MPASM 或 MPLINK 中产生的高低位分开的 HEX 文件的输出扩展名:　<源文件名>.HXL，<源文件名>.HXH
.COD	由 MPASM 或 MPLINK 产生，提供符号和调试信息:<源文件名>.COD
.O	由 MPASM 生成：<源文件名>.O

2．5．1　源代码格式（.ASM）

源代码可以用任何编辑器用 ASCII 码编辑而成。该文件需要满足以下条件：

- 标号
- 助记符
- 操作数
- 命令

这些内容的顺序和位置很重要。标号必须由第一列开始，助记符可以从第二列开始，操作数紧接着助记符，命令可以在操作数，助记符或标号后面，可以从任意列开始。列最多为 255。

标号和助记符间，助记符和操作数间至少用一个空格分隔，多个操作数间需要有一个豆号间隔。例如：

例 2-1　MPASM 源代码范例。 - 列出了多个操作数（OPERAND）

```
;
;  MPASM 源代码范例.仅供演示.
;
        list    p=16c54
Dest    equ     H'0B'

        org     H'01FF'
        goto    Start

        org     H'0000'

Start   movlw   H'0A'
        movwf   Dest
        bcf     Dest, 3
        goto    Start
        end
```

2．5．1．1　标号（Labels）

标号必须开始于第一列，标号后面可以跟一个冒号（:），空格，一个 TAB 的距离，或者行结尾符号。

标号必须用字母或下划线（—）开始，其中可含有数字符号，下划线或者问号。

标号可长达 32 个字符，默认情形下，标号区分大小写。但用一条命令行选项可以忽略大小写。假若标号后面紧跟一个冒号，该冒号只是作为标号操作符，并不是标号的一部分。

2．5．1．2 助记符（Mnemonics）

汇编助记符，指示语言，宏调用语句要至少从第二列开始，假如在某行有标号，指令与该标号间必须间隔一个冒号或者至少一个空格，也可间隔一个 TAB 距离。

2．5．1．3 操作码（Operands）

操作码和助记符间至少用一个空格分隔，也可间隔一个 TAB。多个操作码间要间隔一个豆号

2．5．1．4 注释（Comments）

MPASM 认为：所有分号“；”后面的文本均为注释（comment）。所有分号后面的内容均被忽略。但字符串常量可以包含分号而不至于和注释相混淆。

2．5．2 列表文件格式 (.LST)

例2-2 列表文件格式（.LST）范例：

```
MPASM 01.99.21 Intermediate MANUAL.ASM 5-30-1997 15:31:05 PAGE 1

LOC OBJECT CODE LINE SOURCE TEXT
  VALUE

           00001                              ; MPASM 源代码范例.仅供演示.
           00002                              ;
           00003                              ;
           00004        list p=16c54
0000000B   00005  Dest  equ H'0B'
           00006
01FF       00007        org H'01FF'
01FF 0A00  00008        goto Start
           00009
0000       00010        org H'0000'
           00011
0000 0C0A  00012 Start  movlw H'0A'
0001 002B  00013        movwf Dest
0002 0A00  00014        goto Start
           00015
           16           end

MPASM 01.99.21 Intermediate MANUAL.ASM 5-30-1997 15:31:05 PAGE 2

SYMBOL   TABLE
  LABEL                      VALUE

  Dest                       0000000B
  Start                      00000000
  __16C54                    00000001

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

0000 : XXX------------ ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used: 4
Program Memory Words Free: 508

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

列表文件格式（.LST）由 MPASM 汇编完成后直接生成：包括软件的名称与版本，汇编的日期，每一页的页号。

第一列区域的数字显示程序代码放置在存储器中的起始的地址。第二列区域的 32-BIT 数值显示任何由 SET、EQU、VARIABLE、CONSTANT、CBLOCK 等指示语言生成的符号。第三列区域显示的是可由 PICmicro 微控制器直接运行的机器码。第四列区域显示源文件中本行的行号。剩下的区间留给产生机器码的源文件。

错误，警告，提示信息被插入到源文件中。

符号表列出了程序中涉及到的所有符号。内存使用表以图示方式列出了内存使用情况，“X”表示已被使用，“-”表示可用。假如生成目标文件，则不生成内存使用表。

2．5．3　　错误文件格式（.ERR）

默认情况下 MPASM 产生错误报告文件。这个文件对于调试非常有用，MPASM 源文件级调试产生错误时将自动调用该文件，错误文件格式如下：

<错误类型>[<错误号>] <文件> <错误所在行> <错误说明>

例如：

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

附录 C 列出了 MPASM 产生的错误信息。

2．5．4　　16 进制文件格式（.HEX、HXL、.HXH）

MPASM 可以产生不同的文件格式。附录 A 详细列出了这些信息。

2．5．5　　符号与调试文件格式（.COD）

当 MPASM 产生绝对代码时，将生成。COD 文件，以供 MPLAB 调试之用。

2．5．6　　目标文件格式（.O）

目标文件是由源文件生成的可重定位代码。

第3章　基于 DOS 的 MPASM 汇编

3．1　介绍

本章介绍 DOS 版本的 MPASM 汇编程序（MPASM.EXE 和 MPASM_DP.EXE）。DOS 版本的 MPASM.EXE 可运行于 DOS 命令行或 DOS SHELL，或者 WINDOWS 的 DOS 状态下。增强型 MPASM_DP.EXE 和 DOS 版本一样，只不过它可以在 DOS 内存用尽（OUT OF MEMERY）的情况下运行.

3．2　重点

本章内容包括：

- 命令行界面
- DOS SHELL 界面

3．3　命令行界面

MPASM 可以象以下所示那样通过命令行来调用:

MPASM **[/<选项>[/<选项>....]] [<文件名>]**

或者：

MPASM_DP **[/<选项>[/<选项>....]] [<文件名>]**

其中 /<选项> - 指命令行选项的之一

<文件名> - 指将要被汇编的文件

例如.如果 TEST.ASM 存在于当前目录中, 它可以用以下命令行来汇编:

汇编程序缺省可通过选项来重定义:

- / 选项 打开该选项
- / 选项+ 打开该选项
- / 选项- 关闭该选项
- / 选项 <文件名> 如果合适则打开该选项并指定输出到指定文件

如果源文件名被省略, 一个相应的界面会被调用。

表 3.1 汇编程序命令行选项

选项	缺省	描　　述
?	N/A	显示帮助屏幕
a	INH8M	设置文件格式:/a<hex-format>,这里<hex-format>格式指[INHX8M | INHX8S | INHX32]之一
c	On	打开/关闭大小写敏感
d	None	定义符号：/dDebug /dMax=5 /dString="abc"
e	On	允许/禁止/设置错误文件路径 /e 允许 /e + 允许 /e - 禁止 /e <路径>error.file 允许/设置路径
h	N/A	显示 MPASM 帮助屏幕
l	On	允许/禁止/设置列表文件路径 /l 允许 /l + 允许 /l - 禁止 /l <路径>list.file 允许/设置路径

选项	缺省	描　　述
m	On	允许/禁止宏扩展
O	Off	允许/禁止/设置目标文件路径 /o　　允许 /o +　　允许 /o -　　禁止 /o <路径>object.file　允许/设置路径
P	None	设置处理器类型：/p<处理器类型> 这里：<处理器类型>是指一种 PICmicro 器件，例如 PIC16C54
q	Off	允许/禁止静止模式(抑制屏幕输出)
r	Hex	定义缺省基数：/r<基数> ,这里<基数>指 [HEX | DEC | OCT]三者之一
t	8	列表文件宽度：/t<大小>
w	o	设置信息级别：/w<level>这里<level>指[0|1|2]之一, 0 - 所有信息；1 - 错误和警告；2 - 错误
x	Off	允许/禁止/设置交叉引用文件路径

3．4　DOS Shell 界面

MPASM 的 DOS 界面显示了图形文本方式的屏幕,在这个屏幕上你可以填入你想汇编的源文件名以及其它信息.

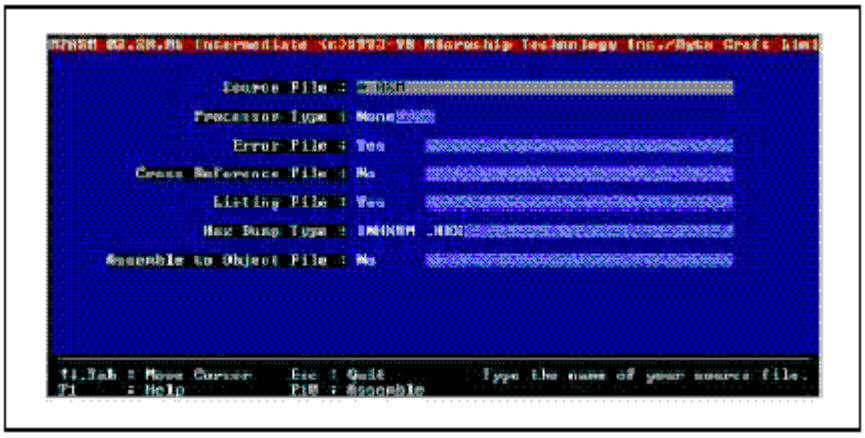


图 3-1 显示图形文本格式

3．4．1　源文件（Source File）

输入你的源文件的文件名, 文件名可以包含一个 DOS 路径以及通配符,你如果使用通配符(*或?之一)所有匹配的文件就会成为一个列表显示供你选择,要在这个区域中自动输入按**<TAB>**键即可。

3．4．2　处理器类型（Processor Type）

如果你在源文件中没有指定处理器,可以用这个字段来选择处理器,用光标键来输入这个字段,然后按　键确定处理器。

3．4．3　错误文件（Error File）

缺省情况下会创建一个出错文件(<源文件名>.ERR)，要关闭出错文件,移动到此项并按“下箭头键 使它移动到 YES 然后敲**<RET>**改变为 NO。出错文件名可以用**<TAB>**键移动到阴影区并输入一个新文件名来改变,不允许通配符。

3．4．4　交叉引用文件（Cross Reference File）

交叉引用文件(<源文件名>.XRF)缺省情况下并不生成。要关闭交叉引用文件，移动到此项并按“下箭头键 使它移动到 YES 然后敲**<RET>**改变为 NO。交叉引用文件名可以用**<TAB>**键移动到阴影区并输入一个新文件名来改变，不允许通配符。

3．4．5　列表文件（Listing File）

列表文件(<源文件名>.LST)可以用同出错文件一样的方法修改此项,它可以用来随意的关 闭列表文件输出,输出文件名可以象出错文件一样修改。

3．4．6　HEX 文件输出类型（HEX Dump Type）

设置此值可以产生想要的 hex 文件格式，可用“下箭头键”键移到此项并按**<RET>**键来滚动可选择的参数项以完成修改此值，要改变 hex 文件名，按**<TAB>** 键移动到阴影区，并输入文件名。

3．4．7　汇编成目标文件（Assemble to Object File）

允许此选项可以产生重定位目标代码以供链接程序使用，同时禁止产生 hex 文件,它可以同修改出错文件一样的方法来改变它。

第4章　基于 WINDOWS 和 MPLAB 的 MPASM 汇编

4.1　介绍

本章介绍基于 WINDOWS 的 MPASM 版本（MPASMWIN.EXE）。该版本可以独立运行，也可以在 MPLAB 界面下运行。

4.2　重点

本章内容包括：

- WINDOWS SHELL 界面
- MPLAB 项目和 MPASM
- 产生输出文件
- MPLAB/MPASM 疑难解答。

4.3　WINDOWS 界面

MPASM 设计了一个图形界面，其中可以修改汇编器的选项。在 WINDOW 中通过执行 MPASMWIN.EXE 来调用的。

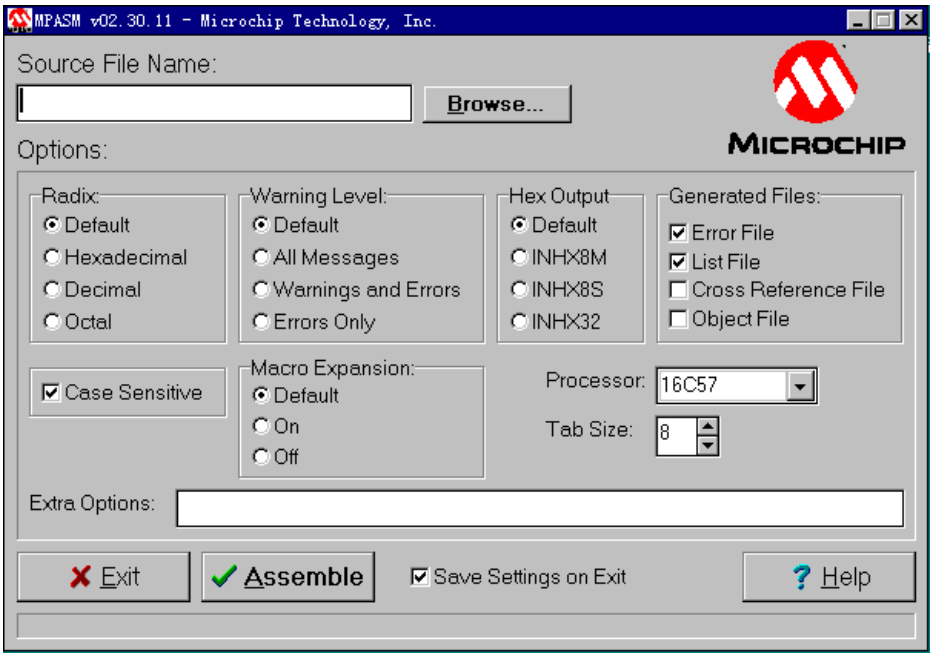


图 4-1：MPASM 的 WINDOWS 界面

可以用键入文件名的方法或用 **Browse**(浏览)按钮来选择一个源文件，同下表描述一样设置可变选项,然后按 **Assemble**(汇编)按钮来汇编源文件.

注意：当 MPASM 的 WINDOWS 版本在 MPLAB 中被调用时，将没有以上的选择窗口。参考《MPLAB 用户指南》获取在 MPLAB 中如何选择汇编选项的信息。

选　项	用　　法
基数	重写源文件中基数设置
警告级别	重写源文件中的信息级别设置
16 进制输出	重写源文件中的 16 进制文件格式设置
生成文件	允许/禁止各种输出文件
大小写	允许/禁止大小写区分功能
处理器	重写源文件中的处理器设置
TAB 宽度	设置列表文件中 TAB 的宽度
其他选项	其他命令行选项见选择 3-3 节
退出前保存设置	退出时在 MPLAB.INI 中保存上面这些设置，下次启动时将使用这些设置

4．4　MPLAB 项目（PROJECT）和 MPASM

MPLAB 的“项目”由“节点”（NODES）生成。(见图 4-2)

- 目标节点 - 最终输出
  - 16 进制文件
- 项目节点 – 元件
  - 源汇编文件

本章我们重点介绍 MPLAB 和 MPASM 的关系。你可以参见《MPLAB 用户指南》获取关于 MPLAB 的一般信息。（**文档资料编号：DS51025**）

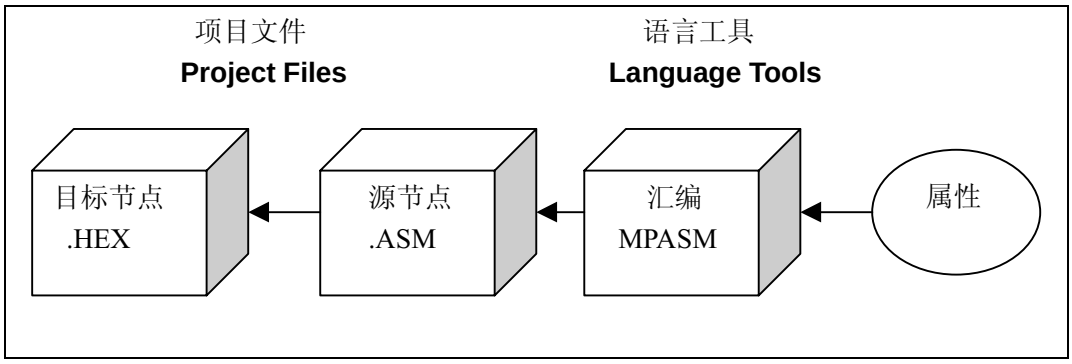


图 4-2：“项目”与源文件及 MPASM 的关系

“项目”采用一些语言工具，比如汇编、编译器、链接器等将源文件转化成可执行文件（.HEX）。本图表显示了最终的.HEX 文件和生成它的.ASM 文件的关系。

4．5　设置 MPLAB 以运行 MPASM

按如下步骤设置 MPLAB，以便使用 MPASM：

1．建立项目后（***Project>New>Newproject***），一个编辑项目会话窗口将出现。在项目列表中选择一个项目（比如：`tutor84.hex`），则“节点属性”（**Node Properties**）按钮将被激活，然后按该按钮。

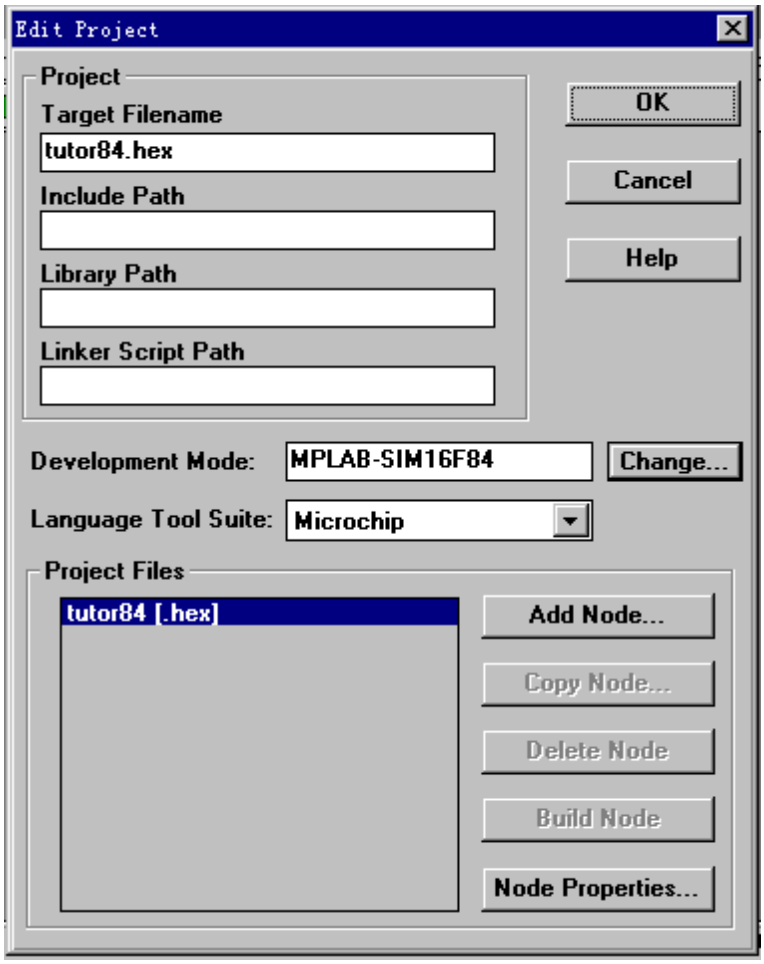


图 4-3： 编辑项目会话窗口

2．在节点属性(Node Property)会话窗口里，选择 MPASM 作为语言工具，如果需　　要，选择其它汇编器选项。

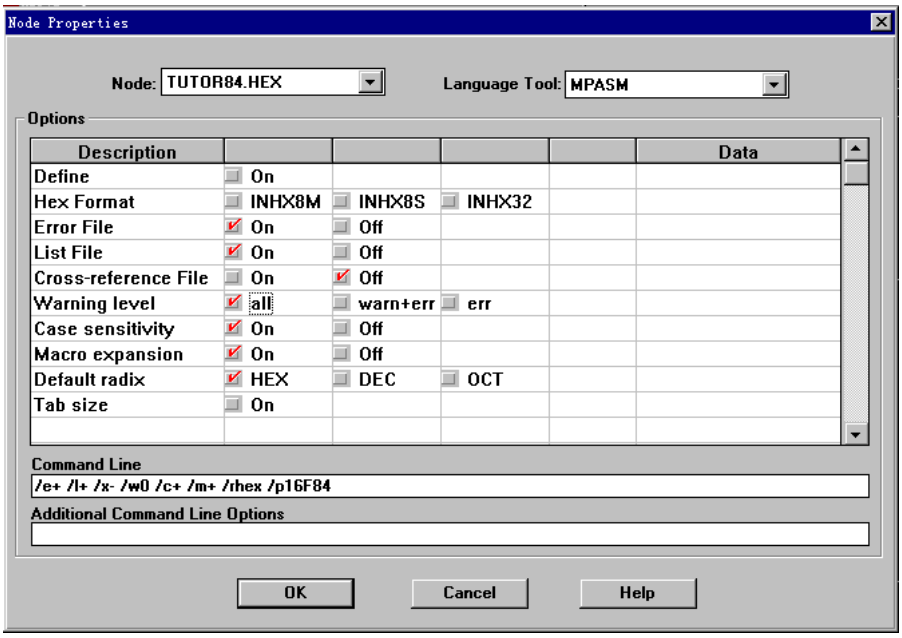


图 4-4： 节点属性会话窗口

3．给你建立的项目添加源文件。点击添加节点（**Add Node**）按钮，会弹出如下会话窗口：

图 4-5： 添加节点会话窗口确 - 源文件

4．完成后点击编辑项目会话窗口上的“**OK**”按钮。

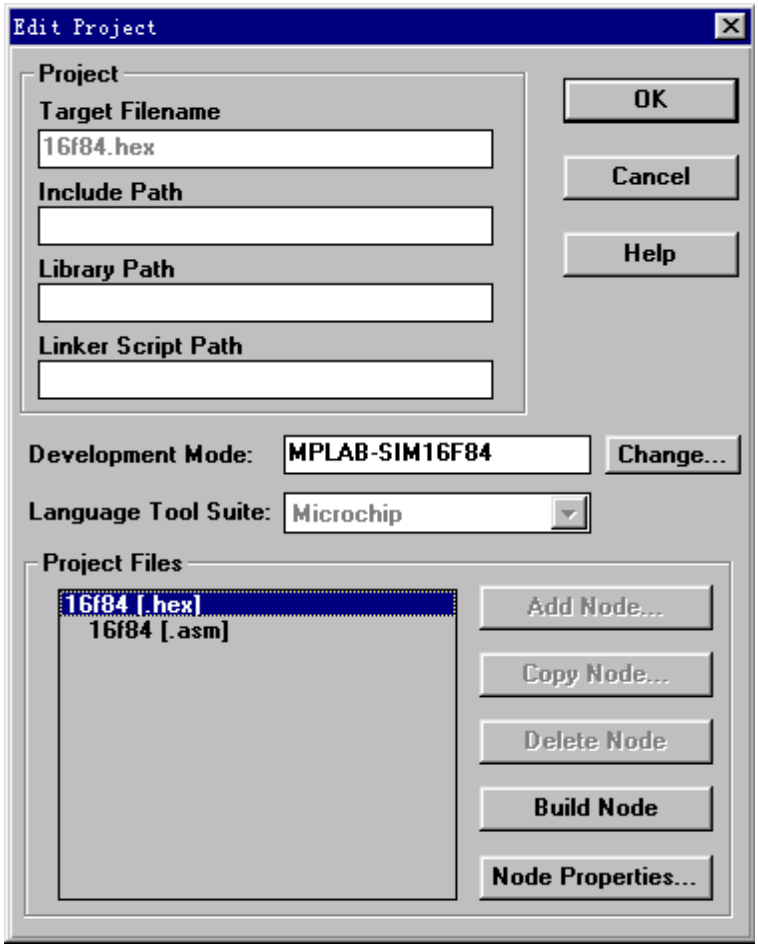


图 4-6： 编辑项目会话窗口 - 添加节点

4．6 生成输出文件

一旦 MPLAB 的属性被设定后，你就可以开始建立项目了。选择以下格式：***Project>Make project***。16 进制文件将被自动调入程序存储器。另外，除了 16 进制文件，MPASM 还产生其他文件以帮助你调试代码。这些文件的信息和具体功能，请参见 2-5 节。

4．7 MPLAB/MPASM 疑难解答

假如有错误，请检查以下状况：

选择：***Project>Install Language Tool…***检查在 MPLAB 的安装目录中，MPASM 是否指向文件 MPASMWIN.EXE，并确信被选中。另一方面，也可以选中基于 DOS 的 MPASM.EXE 文件，但需要同时选中命令行选项。

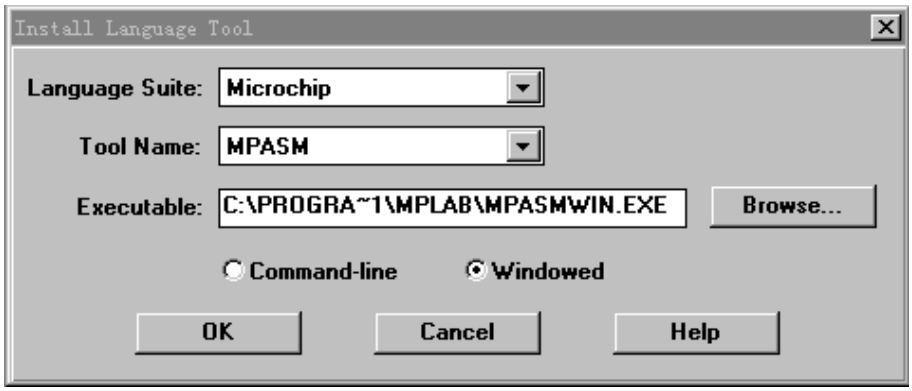


图 4-7： 安装语言工具会话窗口 - MPASM.EXE

增加初试环境的大小。通常 2048 已经足够了。但假如在你的 AUTOEXEC。BAT 文件中有许多设置变量的应用程序的路径申明，你可能需要把该参数设得大一些。

图 4-8 属性会话窗口 – MPASM.EXE

第5章　指示语言

5．1　介绍

本章介绍 MPASM 指示语言（directive language）。

指示符（Directive）是出现在源文件里的汇编命令，但它们并不会直接翻译成操作码。它们用来控制汇编器的操作，包括：输入、输出、数据分配。

许多汇编器指示符有多种形式和名称。这可能是 MICROCHIP 为提供向后兼容，以及和个人编程经验相兼容而设计的。如果想生成紧骤代码，请使用本手册中介绍的编程方法。

5．2　重点

MPASM 提供 5 种基本类型的指示符：

- **控制指示符**（Control Directives） - 控制指示符允许允许条件汇编区段。
- **数据指示符**（Data Directives） - 数据指示符控制存储器分配，而且提供了一种将数据和符号相联系的方法，即：给数据提供有意义的名称。
- **列表指示符**（Listing Directives）- 列表指示符控制 MPASM 列表文件格式。允许指定标题、分页、列表控制。
- **宏指示符**（Macro Directives）- 在宏体（macro body）定义期间控制执行（execution）和数据分配。
- **目标文件指示符**（Object File Directives）- 只在生成目标文件时使用。

5．3　指示符号汇总

表 5-1 MPASM 支持的所有指示符。本章接下来的章节将详细介绍 MPASM 支持的指示符。每个指示符的介绍都将按以下部分介绍：

。语法。介绍。范例

表 5-1：指示语言一览表

指示符	描　述	语　　法
_ _BADRAM	申明不可用的 RAM	_ _badram<表达式>[-<表达式>][,<表达式>[-<表达式>]]
BANKISEL	生成 RAM 区选择码（间接寻址）	bankisel <标号>
BANKSEL	生成 RAM 区选择码	banksel <标号>
CBLOCK	定义常量数据块	cblock [<表达式>]
CODE	可执行代码的开始	[<名称>] code [<地址>]
_ _CONFIG	设置配置寄存器	_ _config <表达式> OR _ _config <地址>, <表达式>
CONSTANT	说明符号常量	constant <标号>[=<表达式>,...,<标号>[=<表达式>]]
DA	存字符串于 ROM 中	[<标号>] da <表达式> [, <表达式 2>, ..., <表达式 n>]
DATA	建立文本数字数据	[<标号>] data <表达式>,[,<表达式>,...,<表达式>] [<标号>]data"<字符串>"[,"<字符串>",...]
DB	定义一字节数据	[<标号>] db <表达式>[,<表达式>,...,<表达式>] [<标号>] db "<字符串>"[,"<字符串>",...]
DE	定义一字节数据 EEPROM 数据	[<标号>] de <表达式>[,<表达式>,...,<表达式>] [<标号>] de "<字符串>"[,"<字符串>",...]
#DEFINE	定义文本替换符	define <名称> [<值>] define <名称> [<参数>,...,<参数>] <值>
DT	表格定义	[<标号>] dt <表达式>[,<表达式>,...,<表达式>] [<标号>] dt "<字符串>"[,"<字符串>",...]
DW	定义一个字	[<标号>] dw <表达式>[,<表达式>,...,<表达式>] [<标号>] dw "<字符串>"[,"<字符串>",...]
ELSE	执行 IF 块的另一块	Else
END	结束程序块	End
ENDC	结束自动常量块	Endc
ENDIF	结束条件汇编块	Endif
ENDM	结束宏定义	Endm
ENDW	结束 WHILE 循环	Endw
EQU	定义汇编常量	<标号> equ <表达式>
ERROR	生成错误信息	error "<字符串>"
ERRORLEVEL	设置错误级别	errorlevel 0|1|2| <+|-><消息号码>

指示语言一览表（续表）：

指示符	描 述	语 法
EXITM	推出宏	Exitm
EXPAND	展开宏列表	Expand
EXTERN	定义外部定义标号	extern <标号>[,<标号>]
FILL	指定内存填充值	[<标号>] fill <表达式>, <计数值>
GLOBAL	出口标号	global <标号>[,<标号>]
IDATA	开始目标文件初始数据	[<名称>] idata [<地址>]
_ _IDLOCS	设置处理器 ID 位置	_ _idlocs <表达式>
IF	开始条件汇编	if <表达式>
IFDEF	如果符号被定义则执行	ifdef <标号>
IFNDEF	如果符号被定义则执行	ifndef <标号>
#INCLUDE	包含另外的源文件	include <<包含文件>> |"<包含文件>"
LIST	列表选项	list [<列表选项>,...,<列表选项>]
LOCAL	说明局部变量	local <标号>[,<标号>]
MACRO	宏定义	<标号> macro [<参数>,...,<参数>]
_ _MAXRAM	定义最大的 RAM 位置	_ _maxram <表达式>
MESSG	建立用户自定义消息	messg "<消息文本>"
NOEXPAND	关闭宏扩展	Noexpand
NOLIST	关闭列表选项	Nolist
ORG	设置程序起始地址	<标号> org <表达式>
PAGE	插入页到列表中	Page
PAGESEL	产生页选择码	pagesel <标号>
PROCESSOR	设置处理器类型	processor <处理器类型>
RADIX	定义默认基数	radix <默认基数>
RES	保留存储器	[<标号>] res <存储器单元>
SET	定义一个汇编变量	<标号> set <表达式>
SPACE	在列表中插入空	space <表达式>
SUBTITLE	指定程序子标题	subtitle "<子标题>"

指示语言一览表（续表）：

指示符	描 述	语 法
TITLE	指定程序标题	title "<标题>"
UDATA	开始目标文件未初始化数据位置	[<名称>] udata [<地址>]
UDATA_ACS	访问目标文件未初始化数据位置	[<名称>] udata_acs [<地址>]
UDATA_OVR	覆盖目标文件未初始化数据位置	[<名称>] udata_ovr [<地址>]
UDATA_SHR	共享目标文件未初始化数据位置	[<名称>] udata_shr [<地址>]
#UNDEFINE	删除一个替换符号	#undefine <标号>
VARIABLE	说明符号变量	variable<标号>[=<表达式>,...,<标号>[=<表达式>]]
WHILE	条件为真时执行循环体	while <表达式>

5．4 __BADRAM - 标注不可用 RAM

5．4．1 语法

_ _badram <表达式>[-<表达式>][, <表达式>[-<表达式>]]

5．4．2 描述

_ _MAXRAM 和 _ _BADRAM 指示语句一起允许你标识使用未设置的寄存器。_ _BADRAM 定义无效 RAM 的位置。该指示符设计来和_ _MAXRAM 一起使用，在任何_ _BADRAM 指示符之前必须有一条_ _MAXRAM。语法中表达式的值必须不大于_ _MAXRAM 中指定的数值。一旦用于_ _MAXRAM 就可以用_ _BADRAM 指定的映象进行严格的地址检查。指定无效位置的范围，可以用指令：<minloc> - <maxloc>。

5．4．3 范例

参见 _ _MAXRAM 的范例。

5．4．4 参见

_ _MAXRAM

5．5 BANKISEL - 产生间接堆（BANK）选择码

5．5．1 语法

```
bankisel <标号>
```

5．5．2 描述

生成目标文件时使用。链接器根据它生成相应的堆选择代码，以便间接寻址访问以<标号>标识的地址。只能有一个<标号>被指定的。对于<标号>不能被操作。<标号>.必须预先定义。

链接器将产生相应的堆选择代码。例如，14-BIT 核的单片机，其 STATUS 寄存器的 IRP 位将会填入相应的 0 或 1。对于 16-BIT 核的单片机，将生 成 MOVLB 或 MOVLR 指令。假如用户自己可以指定这些间接地址而不用这 些指令的话，将不产生代码。

更多信息参考第六章。

5．5．3 范例

```
movlw Var1
movwf FSR
bankisel Var1
...
movwf INDF
```

5．5．4 参见

BANKSEL PAGESEL

5．6 BANKSEL - 产生堆（BANK）选择码

5．6．1 语法

```
banksel <标号>
```

5．6．2 描述

生成目标文件时使用。链接器根据它生成相应的堆（BANK）选择代码，把堆设置在含有<标号>的堆上。只能有一个<标号>被指定的。对于<标号>不能被操作。<标号>.必须预先定义。

链接器将产生相应的堆选择代码。例如，12-BIT 核的单片机，其 FSR 寄存器的相应位将会填入相应的 0 或 1，14-BIT 核的单片机，其 STATUS 寄存器的 IRP 位将会填入相应的 0 或 1，对于 16-BIT 核的单片机，将生成 MOVLB 或 MOVLR 指令，对于增强型 16-BIT 核的单片机，将生成 MOVLB 指令。假如 MCU 只有一个段，则不生成任何指令。

参见第 6 章获取更多信息

5．6．3 范例

```
banksel Var1
movwf Var1
```

5．6．4 参见

```
BANKISEL PAGESEL
```

5．7 CBLOCK-定义常量数据块

5．7．1 语法

```
cblock [<表达式>]
  <标号>[:<increment>][,<标号>[:<increment>]]
endc
```

5．7．2 描述

定义一个命名的常量表。每一个<标号>都赋予了比前一个大的数值。这条指示符主要用来给许多标号赋予偏移量。常数表遇到 ENDC 指令时结束。

<表达式>指定块的第一个标号的开始值。假若没有表达式，第一个名称就接受比前一个 CBLOCK 里最后一个名称大的数值。给定数值从 0 开始。

假若<increment>被指定，则下一个<标号>将比前一个<标号>大，而且被赋予了<increment>的数值。

一行可以给出多个名称，由豆号分隔。

在程序和数据存储区定义常量时，cblock 非常有用。

5．6．3 范例

```
cblock 0x20 ; name_1 被赋值20
```

```
    name_1, name_2 ; name_2 被赋值21，依此类推
    name_3, name_4 ; name_4 被赋值23.
  Endc
  cblock 0x30
    TwoByteVar: 0, TwoByteHigh, TwoByteLow
    Queue: QUEUE_SIZE
    QueueHead, QueueTail
    Double1:2, Double2:2
  Endc
```

5．6．4 参见

ENDC

5．8 CODE - 开始一个目标代码的选项

5．8．1 语法

[<标号>] code [<ROM 地址>]

5．8．2 描述

生成目标文件时使用。申明一个程序代码区段的开始。若<标号>未指定，区段被命名为.COD。开始地址被初始化到指定的地址，如果没有指定地址，则将在链接时分配。

注意：同一源文件里的两个区段不能有相同的名称。

参考第 6 章看详细说明。

5．8．3 范例

```
RESET code H’01FF’
      goto START
```

5．6．4 参见

EXTERN GLOBAL IDATA UDATA UDATA_ACS UDATA_OVR UDATA_SHR

5．9 _ _CONFIG - 设置处理器配置位

5．9．1 语法

_ _config <表达式> 或： _ _config <地址>, <表达式>

5．9．2 描述

设置处理器配置位的数值为<表达式>所给定的。对于 PIC18CXX 系列 MCU，合法的配置字节地址也应该由<表达式>.来指定。对于每一个 MCU，请参考 PICmicro 微处理器数据手册，了解各自配置位的详细情况。

本指示语句使用之前，必须先通过命令行指令，LIST 指示符，或者 PROCESSOR 指示符来说明处理器。本指示符用于 17CXX 系列 MCU 时，

HEX 文件的输出格式必须通过这些方法设置为 INHX32 格式。

5．9．3　范例

```
list p=17c42,f=INHX32
 _ _config H'FFFF' ;默认配置位
```

5．9．4　参见

_ _IDLOCS LIST PROCESSOR

5．10　CONSTANT - 说明符号常量

5．10．1　语法

constant <标号>=<表达式> [...,<标号>=<表达式>]

5．6．2　描述

按 MPASM 的表达式规则建立常量符号。一旦建立常量后，常量不能被重新设置，表达式必须在指定前完全确定具体的数值。这是 CONSTANT 定义符号常量的方法和用 VARIABLE 或用 SET 指示符定义的主要不同点。否则，常量和变量的定义就没什么不同了。

5．6．3　范例

```
variable RecLength=64     ; 设置默认记录长度
                          ;
constant BufLength=512    ; 初始化缓冲区长度
                         . ; 记录长度可能随后被复位：
                         . ;
                         . ;  RecLength=128
                         . ;
constant MaxMem=RecLength+BufLength ;计算最大存储器单元数
```

5．6．4　**参见**

SET VARIABLE

5．11　DA - 将字符串存入程序存储器中

5．11．1　语法

[<标号>] da <表达式> [, <表达式 2>, ..., <表达式 n>]

5．11．2　描述

产生一帧 14-BIT 的数据，代表两个 7-BIT 的 ASCII 码。这对于 PICmicro 的 FLASH 型 MCU 在内存中存储字符串特别有用。

5．11．3　范例

```
da "abcdef"
将把30E2 31E4 32E6 3380 放入程序存储里
da "12345678" ,0
将把18B2 19B4 1AB6 0000 放入程序存储里
```

```
da 0xFFFF
```

将把 0x3FFF 放入程序存储里

5．12 DATA - 建立数字和文本数据

5．12．1 语法

```
[<标号>] data <表达式>,[,<表达式>,...,<表达式>]
[<标号>] data "<字符串>"[,"<字符串>",...]
```

5．12．2 描述

初始化程序存储器中的一个或多个字（WORD）。数据可能是常量，可 重定位或外部标号，或者是以上几个方面之一。数据还可能包含 ASCII 字符串。如果<字符串>上用了单引号，表示一个字符数据；用双引号，则表示字符串。单个字符放在一个字（WORD）的低位字节，而字符串则每两个字符组合占有一个字。假如字符串有奇数个字符，则最后一个字节被赋予 0。除了 PIC18CXX，所有 PIC 系列字符串的存储方式都是：第一个字符放在字的高字节，而对于 PIC18CXX，第一个字符放在字的低字节。当生成目标文件时，该指示符也可以用来申明初始数据。参考 IDATA 以获取更多信息。

<字符串>

5．6．3 范例

```
data reloc_label+10   ; 常量
data 1,2,ext_label    ; 常量, 外部
data "testing 1,2,3"  ; 字符串
data ‘N’              ; 单个字符
data start_of_program  ; 可重定位标号
```

5．6．4 参见

DB DE DT DW IDATA

5．13 DB - 说明一个字节数据

5．13．1 语法

```
[<标号>] db <表达式>[,<表达式>,...,<表达式>]
```

5．13．2 描述

按压缩的 8-BIT 字节保留程序存储器字。多个表达式情况下，在存储器中字节将顺序存放，一直到表达式结束。假如有奇数个表达式，则最后一个字节被赋予 0。当生成目标文件时，该指示符也可以用来申明初始数据。参考 IDATA 以获取更多信息。

5．13．3 范例

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

5．13．4 参见

DATA DE DT DW IDATA

5．14 DE - 说明一个 EEPROM 字节

5．14．1 语法

[<标号>] de <表达式> [, <表达式>, ..., <表达式>]

5．6．2 描述

按 8-BIT 数据保留存储器字。每个<表达式>都可以作为一个 8-BIT 数值。程序字的高位字节是 0，因而字符串中每一个字节都存储在单独的字（WORD）里面。

尽管该指示符是用来初始化 PIC16C8X 的 EEPROM 数据的，该指示符还可以用在任何处理器类型的应用程序中任何位置。

5．6．3 范例

```
org H’2100’                    ; 初始化 EEPROM 数据
de "My Program, v1.0", 0
```

5．6．4 参见

DATA DB DT DW

5．15 #DEFINE - 定义一个文本替换符号

5．15．1 语法

#define <名称> [<字符串>]

5．15．2 描述

本指示符定义一个文本替换符号。汇编中任何有<名称>的地方都将被<string>所替换。

如果使用此指示符时没有<string>则会产生一个内部标记的<名称>，可以用于 IFDEF 指示符。

该指示符模仿 ANSI ’C’标准里的#define。用这种方式定义的符号在 MPLAB 里是无法看到的。

5．15．3 范例

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

5．6．4 参见

```
#UNDEFINE IFDEF IFNDEF
```

5．16 DT - 定义表格

5．16．1 语法

```
[<标号>] dt <表达式> [, <表达式>, ..., <表达式>]
```

5．16．2 描述

生成一系列的 RETLW 指令，每个<表达式>产生一条指令。而且每个<表达式>.必须是一个 8-BIT 数值。字符串中的每个字符存储到与之对应的 RETLW 指令里的 W 寄存器里。

5．6．3 范例

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

5．6．4 参见

DATA DB DE DW

5．17 DW - 说明一个字数据

5．17．1 语法

```
[<标号>] dw <表达式>[,<表达式>,...,<表达式>]
```

5．17．2 描述

为数据保留程序存储器，初始化这些空间为指定的数值。对于 PIC16CXX 系列 MCU，DW 的功能类似 DB，数值被依此存储到中响应位置，同时位置记数器加 1。表达式可以是字符串，而且可以和 DATA 指示符描述的一样存储。

当生成目标文件时，该指示符也可以用来申明初始数据。参考 IDATA 以获取更多信息。

5．17．3 范例

```
dw 39, "diagnostic 39", (d_list*2+d_offset)
dw diagbase-1
```

5．17．4 参见

DATA DB IDATA

5．18 ELSE - 开始 IF 汇编块的另一分支

5．18．1 语法

```
else
```

5．6．2 描述

和 IF 指示符一起使用，当 IF 条件不满足时则执行这一块汇编代码。ELSE 可以在正规的程序或宏中使用。

5．6．3　范例

```
 speed macro rate
     if rate < 50
         dw slow
     else
     dw fast
     endif
endm
```

5．6．4　参见

ENDIF IF

5．19　END-程序结束标志

5．19．1 语法

```
end
```

5．19．2 描述

指示出程序结束。

5．19．3 范例

```
list p=17c42
:                ; 可执行代码
:                ;
end              ; 指令结束
```

5．20　ENDC-结束一个自动常量块

5．20．1 语法

```
endc
```

5．20．2 描述

ENDC 作为结束所有 CBLOCK 列出程序的标志。必须用它来结束所有代码序列。

5．20．3 参见

CBLOCK

5．21　ENDIF-结束条件汇编

5．21．1　语法

```
endif
```

5．6．2　描述

本指示符标识一个条件汇编块的结束。ENDIF 可以在正规的程序块或宏中使用。

5．6．3　参见

ELSE IF

5．22　ENDM - 结束宏定义

5．22．1　语法

endm

5．22．2　描述

结束以 MACRO 开始的宏定义。

5．22．3　范例

```
make_table macro arg1, arg2
           dw arg1, 0          ; 以 0 结束表格名
           res arg2            ; 保留存储器
           endm
```

5．6．4　参见

MACRO EXITM

5．23　ENDW - WHILE 循环的结尾

5．23．1　语法

endw

5．23．2　描述

ENDW 结束一个 WHILE 循环。汇编时，只要 WHILE 指定的条件是真，WHILE 和 ENDW 之间的源代码将被不断地查入到汇编源代码中去。本指示符可以在正规的程序块或宏中使用。

5．23．3　范例

参考 WHILE 指示符范例。

5．23．4　参见

WHILE

5．24　EQU-定义一个汇编常量

5．24．1　语法

<标号> equ <表达式>

5．24．2　描述

<表达式> 的值被赋予<标号>。

5．24．3　范例

```
four equ 4 ; 给标号four赋予数值4 。
```

5．24．4　参见

SET

5．25　ERROR - 产生一条错误信息

5．25．1　语法

```
error "<字符串>"
```

5．25．2　描述

<字符串>将被打印出来，格式与任何 MPLAB 错误信息相同。<字符串>可以是 1 到 80 个字符。

5．25．3　范例

```
error_checking macro arg1
    if arg1 >= 55                ; 如果参数超出范围
       error "error_checking-01 arg out of range"
    endif
endm
```

5．6．4　参见

MESSG

5．26　ERRORLEVEL - 设置信息优先级

5．26．1　语法

```
errorlevel {0|1|2|+<消息号>|-<消息号>} [, ...]
```

5．26．2　描述

设置在列表文件和错误文件中打印的信息类型。

设置	作用
0	打印消息，警告，错误
1	打印警告，错误
2	打印错误
-<消息号>	禁止打印指定的< 消息号>
+<消息号>	允许打印指定的<消息号>

<消息号>的数值在附录 C 中有详细列表。错误信息不能屏蔽。设置参数 0，1，或 2 可以允许或禁止某个单独的打印信息。

5．26．3　范例

```
errorlevel 1, -202
```

5．26．4　参见

LIST

5．27 EXITM - 退出宏

5．27．1 语法

```
exitm
```

5．27．2 描述

在汇编的时候强迫从宏扩展立即返回。它的结果和遇到一条 ENDM 指示符一样。

5．27．3 范例

```
test macro filereg
    if filereg == 1 ; 检查合法的文件
        exitm
    else
        error "bad file assignment"
    endif
endm
```

5．6．4 参见

ENDM MACRO

5．28 EXPAND - 展开宏列表

5．28．1 语法

```
expand
```

5．28．2 描述

在列表文件里展开所有的宏。这条指令大致等同于 MPASM 命令行选项的/m 参数。

5．28．3 参见

MACRO NOEXPAND

5．29 EXTERN - 定义外部定义标号

5．29．1 语法

```
extern <标号> [, <标号>...]
```

5．6．2 描述

当产生目标文件时使用。申明可能用在当前模块里但在其它模块里却被定义成全局变量的符号名称。EXTERN 必须在<标号>使用前就被包含（INCLUDE）。在这一行里，至少有一个标号被定义。假如<标号>在当前模块被定义，MPASM 将会产生一个重复标号错误。

更详细信息参考第 6 章。

5．6．3　范例

```
extern Function
...
call Function
```

5．6．4　参见

GLOBAL IDATA TEXT UDATA UDATA_ACS UDATA_OVR UDATA_SHR

5．30　FILL - 指定内存填充值

5．30．1　语法

[<标号>] fill <表达式>,<计数值>

5．30．2　描述

产生<计数值>个程序字，或者字节（PIC18CXX），具体值为<表达式>。假如它被圆括号括起来，<表达式>可以是一个汇编指令。

5．30．3　范例

```
fill 0x1009, 5                          ; 填入常量
fill (GOTO RESET_VECTOR), NEXT_BLOCK-$
```

5．6．4　参见

DATA DW ORG

5．31　GLOBAL - 出口标号

5．31．1　语法

global <标号> [, <标号>...]

5．31．2　描述

当生成目标文件时使用。申明在当前模被定义，而且在别的模块也可以使用的符号名称。GLOBAL 必须在<标号>之后定义。在这一行必须有至少一个标号被定义。

更多信息，参考第 6 章。

5．31．3　范例

```
            udata
Var1        res 1
Var2        res 1
            global Var1, Var2
            code
AddThree
```

```
global AddThree
addlw 3
return
```

5．6．4　参见

EXTERN IDATA TEXT UDATA UDATA_ACS UDATA_OVR UDATA_SHR

5．32　IDATA - 开始目标文件初始数据

5．32．1　语法

[<标号>] idata [<RAM address>]

5．6．2　描述

当生成目标文件时使用．申明初始化区段的开始，假若<标号>没有指定，则区段被命名为.idata．开始地址被初始化到指定的地址；假若没有指定地址，则会在链接时分配．在这一段（SEGMENT）将不能产生代码．链接器将为在 IDATA 区段指定的每个字节生成一个查询表格．用户必须链接，或者包含(INCLUDE)相应的初始化代码．参考第 9 章了解 PICmicro 系列ＭＣＵ的初始化代码范例．值得注意的是：本指示符不适用于 12－BIT 核的 MCU．

给变量(VARIABLE)保留空间可以用到 RESDB,DW 指示符．RES 将会生成一个初始数值 0；DB 将会在 RAM 里依次初始化一系列字节(BYTE)；DW 将会在 RAM 里初始化成一系列的字（WORD），每次一个字节，顺序是：低字节／高字节．

参考第 6 章获取更多信息．

5．6．3　范例

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

5．6．4　参见

EXTERN GLOBAL TEXT UDATA UDATA_ACS UDATA_OVR UDATA_SHR

5．33　__IDLOCS　- 设置处理器 ID 位置

5．33．1　语法

_ _idlocs <表达式> or _ _idlocs <表达式 1>, <表达式 2>

5．33．2　描述

对于 PIC12CXX,PIC14000,和 PIC16CXX 系列 MCU，_ _idlocs 按

照<表达式>的 1 6 进制数值设置四个 I D 位置。对于 PIC18CXX 系列 MCU，该指示符将 2 字节的 ID<表达式 1>设置为 16 进制形式的<表达式 2>该指示符不适用于 17CXX 系列 MCU。例如：假如<表达式>被赋值为 1AF，则第一个 ID（最低地址）为 0，第二个 ID 为 1，第三个 ID 为 1 0，第四个 ID 为 1 5.

使用本指示符时，必须通过命令行， LIST 或者 PROCESSOR 指示符说明处理器.

5. 33. 3 范例

```
_ _idlocs H'1234'
```

5. 33. 4 参见

```
LIST PROCESSOR _ _CONFIG
```

5. 34 IF - 开始条件汇编

5. 34. 1 语法

```
if <表达式>
```

5. 34. 2 描述

开始一个条件汇编块的执行. 假如<表达式>的值为真（true），则紧跟 IF 的代码被汇编. 否则，直到遇见 ELSE 或者 ENDIF 指示符，代码都将被忽略.

假若表达式的运算值为 0，则在逻辑上被认为是 " 假 " （FALSE）；任何其他运算结果均被认为是 " 真 " （TRUE）. 所以，IF 和 WHILE 靠表达式的逻辑值来决定程序的汇编代码. 运算结果为真的表达式返回逻辑值为 1，反之则为 0.

5. 34. 3 范例

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

5. 6. 4 参见

```
ELSE ENDIF
```

5. 35 IFDEF - 如果符号已被定义则执行

5. 35. 1 语法

```
ifdef <标号>
```

5．35．2　描述

假如<标号>已经预先定义－通常是用一条#DEFINE 指示符或者在 MPASM 里用命令行参数来设置（选择条件路径）．汇编将一直进行，直到遇见与之匹配的 ELSE 或者 ENDIF 指示符．

5．35．3　范例

```
#define testing 1           ; 设置测试位"on"
 :
 :
 ifdef testing
       <execute test code>  ; 测试代码块
 endif                      ;
```

5．6．4　参见

#DEFINE ELSE ENDIF IFNDEF #UNDEFINE

5．36　IFNDEF-如果符号未定义则执行

5．36．1 语法

ifndef <标号>

5．36．2 描述

假如<标号>没有预先定义，或者没有用一条#UDEFINE 指示符来取消定义，则该指示符后面的代码将被汇编，直到遇见与之匹配的 ELSE 或者 ENDIF 指示符．

5．36．3　范例

```
 #define testing1        ; 打开测试
         :
         :
#undefine testing1       ; 关闭测试
         ifndef testing  ; 假如不是测试模式
         :               ; 则执行这一部分
         :               ;
         endif           ;
                         ;
         end             ; 结束代码
```

5．6．4　参见

#DEFINE ELSE ENDIF IFDEF INE #UNDEF

5．37　INCLUDE-包含另外的源文件

5．37．1　语法

```
include <<包含文件>>
include "<包含文件>"
```

5．37．2　描述

被指定的文件将作为源代码读入．看起来被包含的代码好象是被插入到 INCLUDE 指定的位置一样．遇到文件结束符时，源代码汇编又会恢复到初始源文件处，这种方式允许嵌套，最多达 6 层．<包含文件>可以放在双引号或者尖括号里.假若指定了一个完全的限制路径,则只搜索该路径.否则搜索顺序依此为：当前工作路径；源文件路径；MPASM 可执行文件所在的路径.

5．37．3　范例

```
include "c:\sys\sysdefs.inc" ; 系统定义
include <regs.h>             ; 寄存器定义
```

5．38　LIST - 列表选项

5．38．1 语法

```
list [<列表选项>, ..., <列表选项>]
```

5．6．2　描述

该指示符单独占一行，假若 LIST 指示符被预先关掉的话，则它会将列表输出打开．否则，选用下表中的选项可以控制汇编过程或者格式化列表文件.

选项	默认值	说明
b=nnn	8	设置 TAB 宽度
c=nnn	132	设置列宽度
f=<format>	INHX8M	设置 HEX 文件格式。<format>可以是 INHX32， INHX8M，INHX8S
Free	FIXED	使用自由格式，保证向下兼容
Fixed	FIXED	使用固定格式
mm={ON|OFF}	On	在列表文件中打印存储器映像
n=nnn	60	设置每页的行数
p=<type>	None	设置处理器类型，如：PIC16C54
r=<基数>	hex	设置默认的基数：hex,dec,oct
st={ON|OFF}	On	在列表文件中打印符号表
t={ON|OFF}	Off	截断列表行（否则会出界）
w={0|1|2}	0	设置信息级别，参见 ERRORLEVEL
x={ON|OFF}	On	开/关宏扩展

注意：所有 LIST 选项都被用 10 进制数来求值。

5．6．3　范例

```
list p=17c42, f=INHX32, r=DEC
```

5．6．4　参见

```
ERRORLEVEL EXPAND NOEXPAND NOLIST PROCESSOR RADIX
```

5．39 LOCAL - 说明局部变量

5．39．1 语法

```
local <标号>[,<标号>...]
```

5．39．2 描述

该指示符申明：指定的数据元素只存在于宏的某一局部范围。在宏定义的外部，仍然可以使用与<标号>相同的标号而不会产生冲突。假若宏被递归调用，则每次都有一个局部副本保留下来。

5．39．3 范例

```
<main code segment>
  :
  :
len   equ  10            ; 全局变量
size  equ  20            ; 指定局部变量
                         ; 现在可以被建立和修改
test macro size          ;
      local len, label   ; local len 和 label
len   set  size          ; 修改local len
label res len            ; 保留的缓冲区
len   set len-20         ;
endm                     ; 结束宏
```

5．6．4 参见

```
ENDM MACRO
```

5．40 MACRO - 宏定义

5．40．1 语法

```
<标号> macro [<参数>, ..., <参数>]
```

5．40．2 描述

所谓“宏”指一系列指令集合，可以通过宏调用的方式，将这些代码插入到源代码序列中。但必须首先定义宏，然后才能在以后的程序中调用这个宏。一个宏可以调用另外一个宏，或者递归调用。

想进一步了解，参考第 5 章《宏语言》。

5．40．3 范例

```
Read macro device, buffer, count
     movlw device
     movwf ram_20
```

```
        movlw buffer   ; buffer 地址
        movwf ram_21
        movlw count    ; 字节计数器
        call sys_21    ; 读文件调用
    endm
```

5．40．4　参见

ELSE ENDIF ENDM EXITM IF LOCAL

5．41　_ _MAXRAM-定义最大的 RAM 位置

5．6．1　语法

_ _maxram <表达式>

5．6．2　描述

_ _MAXRAM 和_ _BADRAM 一起，允许用户标识对未设置寄存器的操作。_ _MAXRAM 定义了最大的可用 RAM 地址，并初始化可用 RAM 地址映像为小于<表达式>的所有地址有效。<表达式>必须不小于零页的 RAM 最大的地址，且小于 1000H。这条指示符是和_ _BADRAM 一起使用的，一旦使用了_ _MAXRAM 指示符，就可以用_ _BADRAM 指定的 RAM 映像进行严格的 RAM 检查。

_ _MAXRAM 可以在源文件中多次使用。每次使用都重新定义最大的有效 RAM 地址，同时重新设置 RAM 映像中的所有位置。

5．6．3　范例

```
list p=16c622
_ _maxram H'0BF'
_ _badram H'07'-H'09', H'0D'-H'1E'
_ _badram H'87'-H'89', H'8D', H'8F'-H'9E'
movwf     H'07' ; 生成非法 RAM 警告
movwf     H'87' ; 生成非法 RAM 警告和
                ; 截断信息
```

5．6．4　参见

_ _BADRAM

5．42　MESSG - 建立用户自定义信息

5．42．1　语法

messg "<信息文本>"

5．42．2　描述

使提示信息在列表文件中被打印，文本信息最长 80 字符。使用 MESSG 不设置任何错误代码。

5．42．3 范例

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

5．42．4 参见

ERROR

5．43 NOEXPAND - 关闭宏扩展

5．43．1 语法

nolist

5．6．2 描述

在列表文件中关闭宏扩展。

5．6．3 参见

EXPAND

5．44 NOLIST - 关闭列表选项

5．44．1 语法

nolist

5．44．2 描述

关闭列表文件输出。

5．44．3 参见

LIST

5．45 ORG - 设置程序起始地址

5．45．1 语法

[<标号>] org <表达式>

5．45．2 描述

设置程序代码的开始地址，该地址由<表达式>赋予。假如指定了<标号>，则它会得到<表达式>给定的值。如果没有指定 ORG，代码将从地址 0 开始。

对于 PIC18CXX 系列 MCU，<表达式>只允许是偶数。

当生成目标文件时，不能使用该指示符。

5．45．3 范例

```
int_1 org 0x20
                ; 向量 20 的代码放在这里
int_2 org int_1+0x10
                ; 向量 30 的代码放在这里
```

5．45．4　参见

FILL RES

5．46　PAGE - 插入页到列表中

5．46．1　语法

page

5．46．2　描述

在列表文件中插入新的页。

5．46．3　参见

LIST SUBTITLE TITLE

5．47　PAGESEL - 产生页选择码

5．47．1　语法

pagesel <标号>

5．47．2　描述

产生目标文件时使用。告诉链接器生成页选择代码，以便设置页选位，使得选在<标号>指定的页上。只能设置一个<标号>。对<标号>不能执行操作，而且<标号>需要预先定义。

链接器将生成相应的页选择代码。对 12-BIT 核的 MCU，将会对 STATUS 寄存器相应的位进行置位或复位；14-BIT 和 16-BIT 核的 MCU，MOVLW 和 MOVWF 指令将会生成，用以修改 PCLATH。假如 MCU 只包含一个存储器页，则不产生任何页选代码。

对于 PIC18CXX 系列 MCU，本指令无任何操作。

更多信息请参考第 6 章。

5．47．3　范例

```
pagesel GotoDest
goto GotoDest
....
pagesel CallDest
call CallDest
```

5．47．4　参见

BANKISEL BANKSEL

5．48 PROCESSOR - 设置处理器类型

5．48．1 语法

```
processor <处理器类型>
```

5．48．2 描述

设置处理器类型为<处理器类型>。

5．48．3 范例

```
processor 16C54
```

5．48．4 参见

```
LIST
```

5．49 基数 - 设置默认基数

5．49．1 语法

```
radix <默认基数>
```

5．49．2 描述

为数据设置默认的基数（进制形式）。默认为 HEX。合法的基数有：hex（16 进制），dec（10 进制），或 oct（8 进制）。

5．49．3 范例

```
radix dec
```

5．49．4 参见

```
LIST
```

5．50 RES - 保留存储器

5．50．1 语法

```
[<标号>] res <存储器单元>
```

5．50．2 描述

使程序记数器有当前位置向前移动，移动长度为<存储器单元>个单元。对于非可重定位代码，<标号>指定的地址被认为是程序存储器地址。对于可重定位代码（使用 MPLINK），res 还可以被用来保留数据存储器。

对于 12-, 14- 和 16-bit MCU，地址定位用“字”来定义，对于增强型 16-BIT MCU 则采用“字节”来定义。

5．6．3 范例

```
buffer res 64  ; 保留 64 个存储器地址定位
```

5．6．4 参见

```
FILL ORG
```

5．51 SET - 定义一个汇编变量

5．51．1 语法

<标号> set <表达式>

5．51．2 描述

<标号>被赋予由<表达式>指定的合法的 MPASM 表达式值。SET 指示符在功能上和 EQU 相似，不同的是：SET 指定的值可以在随后用另外一条 SET 指示符取代。

5．51．3 范例

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

5．51．4 参见

EQU VARIABLE

5．52 SPACE - 在列表中插入空行

5．52．1 语法

space <表达式>

5．52．2 描述

在列表文件中插入<表达式>个空行。

5．52．3 范例

```
space 3 ;插入 3 个空行
```

5．52．4 参见

LIST

5．53 SUBTITLE - 指定程序子标题

5．53．1 语法

subtitle "<子标题>"

5．53．2 描述

<子标题>是用双引号包含的 ASCII 字符串，长度不大于 60 个字符。本指示符建立一个第二标题行，作为列表文件的子标题。

5．53．3　范例

```
subtitle "diagnostic section"
```

5．6．4　参见

TITLE

5．54　TITLE - 指定程序标题

5．54．1　语法

```
title "<标题文本>"
```

5．54．2　描述

<标题文本>是用双引号包含的 ASCII 字符串，长度不大于 60 个字符。本指示符在列表输出文件中每页的第一行建立标题行，作为列表文件的标题。

5．54．3　范例

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

5．54．4　参见

LIST SUBTITLE

5．55　UDATA - 开始目标文件未初始化数据区段

5．55．1　语法

[<标号>] udata [<RAM 地址>]

5．55．2　描述

当生成目标文件时使用。申明一个未初始化区段的开始。假如<标号>未指定，则该区段被命名为.udata。起始地址被初始化到指定的地址，或者假如未指定地址的话，地址将在链接时被分配。本段不会产生代码。RES 指示符应当用来保留数据空间。

注意：同一源文件里的两个区段不能有相同的名称。

更多信息请参考第 6 章。

5．6．3　范例

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

5．6．4　参见

EXTERN GLOBAL IDATA UDATA_ACS UDATA_OVR UDATA_SHR TEXT

5．56 UDATA _ ASC - 访问目标文件未初始化数据区段

5．56．1 语法

[<标号>] udata_acs [<RAM 地址>]

5．56．2 描述

当生成目标文件时使用。申明一个访问未初始化区段的开始。假如<标号>未指定，则该区段被命名为.udata_acs。起始地址被初始化到指定的地址；另一方面，假如未指定地址的话，地址将在链接时被分配。本指示符针对 PIC18CXX 系列ＭＣＵ，用来申明定位于ＲＡＭ中的变量。本段不会产生代码。RES 指示符应当用来保留数据空间。

注意：同一源文件里的两个区段不能有相同的名称。

更多信息请参考第 6 章。

5．56．3 范例

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

5．6．4 参见

EXTERN GLOBAL IDATA UDATA UDATA_OVR UDATA_SHR TEXT

5．57 UDATA_OVR - 覆盖目标文件未初始化数据区段

5．57．1 语法

[<标号>] udata_ovr [<RAM 地址>]

5．57．2 描述

当生成目标文件时使用。申明一个访问覆盖未初始化区段的开始。假如<标号>未指定，则该区段被命名为.udata_ovr。起始地址被初始化到指定的地址；另一方面，假如未指定地址的话，地址将在链接时被分配。由这个区段申明的空间将被所有其他与 udata_ovr 区段名称相同的区段所覆盖。本指示符在申明临时变量时非常理想，因为它允许多个变量被申明在同一存储器位置。RES 指示符应当用来保留数据空间。

注意：同一源文件里的两个区段可以有相同的名称。

更多信息请参考第 6 章。

5．57．3 范例

```
 Temps udata_ovr
Temp1  res 1
Temp2  res 1
Temp3  res 1
Temps  udata_ovr
LongTemp1 res 2   ; 将与Temp1和Temp2有相同位置的变量
```

```
LongTemp2 res 2   ; 将与 Temp3 有相同位置的变量
```

5．6．4　参见

EXTERN GLOBAL IDATA UDATA UDATA_ACS UDATA_SHR TEXT

5．58　UDATA_SHR - 共享目标文件未初始化数据区段

5．58．1　语法

[<标号>] udata_shr [<RAM 地址>]

5．58．2　描述

当生成目标文件时使用。申明一个共享未初始化数据区段的开始。假如<标号>未指定，则该区段被命名为.udata_shr。起始地址被初始化到指定的地址；另一方面，假如未指定地址的话，地址将在链接时被分配。本指示符用来申明位于 RAM 中的变量时，这些 RAM 对于所有的 RAM 块（如：未分块 RAM）都是共享的。本段不会产生代码。RES 指示符应当用来保留数据空间。

注意：同一源文件里的两个区段不能有相同的名称。

更多信息请参考第 6 章。

5．58．3　范例

```
Temps udata_shr
Temp1 res 1
Temp2 res 1
Temp3 res 1
```

5．58．4 参见

EXTERN GLOBAL IDATA UDATA UDATA_ACS UDATA_SHR TEXT

5．59　#UNDEFINE - 删除一个替换符号

5．59．1　语法

#undefine <标号>

5．59．2　描述

<标号>是一个预先用#define 指示符定义的标志，必须合乎 MPASM 的标号规则。用该指示符可以将符号从符号表中删除掉。

5．59．3　范例

```
#define length 20
:
:
#undefine length
```

5．6．4　参见

#DEFINE IFDEF INCLUDE IFNDEF

5．60　VARIABLE - 说明符号变量

5．60．1　语法

```
variable <标号>[=<表达式>][,<标号>[=<表达式>]...]
```

5．60．2　描述

建立 MPASM 表达式中用到的符号，表达式中的变量和常量可以在表达式中交替使用。

用指示符 variable 建立的符号和用指示符 set 建立的符号在功能上是相似的。不同之处在于：用指示符 variable 申明变量时不需要符号初始化。

需要指出的是，变量值不能用操作数更新。你必须将变量的赋值，变量的增 1 和减 1 运算放在单独的行里。

5．60．3　范例

请参考指示符 constant 的范例。

5．60．4　参见

CONSTANT SET

5．61　WHILE - 条件为真时执行循环体

5．61．1　语法

```
while <表达式>
:
:
endw
```

5．61．2　描述

当<表达式>表达式的值是真（TRUE）时，位于 while 和 endw 之间的代码将被汇编。表达式的值是 0 时，认为是逻辑“假（FALSE）”。表达式的任何其它值则被认为是逻辑“真（TRUE）”。一个为“真”的表达式其返回值一定不为 0，反之，则返回一个 0。一个 WHILE 块最多可以包含 100 行指令，而且可以重复 256 次。

5．61．3　范例

```
test_mac    macro count
            variable i
i = 0
            while i < count
            movlw i
i += 1
            endw
            endm
start
            test_mac 5
            end
```

5．6．4　参考

ENDW IF

第6章　使用 MPASM 建立可重定位目标代码

6．1 概述

介绍了 MPASM V2.00 和 MPLINK1.00 后，用户就可以开始生成和链接预编译目标模块了。编写将被汇编成目标模块的源程序，和直接生成可执行的 HEX 代码有一些差异。用 MPASM 编写的绝对地址汇编例程稍加修改就可以正确地编译到可重定位目标模块中。

6．2 重点

本章主要内容的标题如下：

- 头文件
- 程序存储器
- 指令助记符
- RAM 的分配
- 配置位及 CPU-ID 位
- 从其他模块来的操作标号
- 分页和分块问题
- 不再使用的指示符
- 生成目标模块
- 代码范例

6．3 头文件

当生成目标模块时，必须使用 MICROCHIP 提供的标准的头文件（形如：p17c756.inc）。这些头文件里专门为处理器的特殊功能寄存器进行了定义。

6．4 程序存储器

程序存储器必须优先由 CODE 进行区段申明。

6．4．1 绝对代码

```
Start CLRW
      OPTION
      :
```

6．4．2 可重定位代码

```
CODE
Start CLRW
      OPTION
      :
```

假若一个源程序中定义了不止一个代码区段，每个区段必须有一个最终的名称。假如没有指定文件名，则会使用默认名为.CODE。在一个源程序里，程序存储器区段必须连续。单个源程序文件里的区段不能被分成片段。

通过选择 CODE 指示符的地址选择参数，即可确定代码的物理地址。

可能需要这样做的情形还有：

- 指定中断向量。
- 确保代码段不会超越边界。

6．4．3 可重定位代码范例

```
Reset CODE  H'0lFF'
      GOTO  Start
Main  CODE
      CLRW
      OPTION
```

6．5 指令操作符

有一些指令操作数的规范需要说明。即，指令操作数需要按如下格式来安排：

```
[HIGH|LOW|UPPER] (<可重定位符号> + <常数偏移量>)
```

其中：

- <可重定位符号> 是任何定义程序或数据存储器的标号
- <常数偏移量> 是表达式，在汇编时会转化为一个介于-32768 到+32768 之间的数据。

<可重定位符号> 或 <常数偏移量>可以被省略。假如两个符号被定义在同一区段里，则形如：

<可重定位符号> - <可重定位符号> 的操作数将变为一个常量。

假如用了参数“HIGH”，则只使用表达式的第 8-15 位；假如用了参数“LOW”，则只使用表达式的第 0-7 位；假如用了参数“UPPER”，则只使用表达式的第 16-21 位。

6．6 RAM 定位

RAM 空间必须定位在数据区段里，数据区段有 5 种类型：

- **UDATA-未初始化数据**。这是最常用的数据区段类型。本区段保留的位置没有初始化，只能通过在本区段定义的符号或间接寻址操作。
- **UDATA_ACS-未初始化读写数据**。本数据区段是给变量用的，这些变量将放置于 PIC18CXX 系列 MCU 的 RAM 里。“读写 RAM”是 MCU 内部特定指令进行数据快速存取的存储器。
- **UDATA_OVR-未初始化覆盖数据**。本数据区段是给变量用的，这些变量可以和同一模块里或者已链接的模块里的其它变量以相同的地址申明。其典型应用是临时变量。
- **UDATA_SHR-未初始化共享数据**。本数据区段是给变量用的，这些变量放置于未分块的 RAM 里或者与所有的块共享。
- **IDATA**-初始化数据。链接器将生成一个查询表格（LOOKUP TABLE），将本区段的变量初始化为特定的数值。只有在本区段定义的标号或间接寻址才可以操作在该区段保留的位置。
  参考以下范例，了解如何进行数据申明。

6．6．1 绝对代码

```
CBLOCK 0x20
   InputGain, OutputGain ;控制循环大小
   HistoryVector         ;应该初始化为 0
   Templ, Temp2, Temp3   ;作为内部运算用
ENDC
```

6．6．2 可重定位代码

```
                IDATA
HistoryVector DB    0

                UDATA
InputGain      RES    1
OutputGain     RES    1
                UDATA_OVR
 Templ         RES    1
 Temp2         RES    1
 Temp3         RES    1
```

假如有必要，区段的位置可以根据可选地址参数固定地放在存储器的某一位置。假如要定义一个以上的区段类型，则每个区段都必需有一个专用名称。如果不提供名称，则会使用默认名称。这些名称有：

.idata, .udata,.udata_acs,
.udata_shr, and .udata_ovr.

当在 IDATA 区段里定义初始化数据，可以使用指示符：DB，DW 和 DATA。DB 将会定义一系列数据存储器字节，DW 将会定义一系列数据存储器字，DATA 将会定义一系列数据存储器字（按格式：低字节-高字节排列）。以下范例显示如何初始化数据：

6．6．3 可重定位代码

```
                    00001   LIST p=17C4
                    00002   IDATA
0000 01 02 03       00003  Bytes  DB 1,2,3
0003 34 12 78 56    00004  Words   DW H’1234’,H’5678’
0007 41 42 43 00    00005  String  DB “ABC”, 0
```

6．7 配置位和 ID 位置

使用_ _CONFIG 和 _ _IDLOCS 指示符，在可重定位目标模块里仍然可以对配置位和 CPU 的 ID 定位进行设置。只有一个已链接模块可以定义这些指示符。他们必须在任何 CODE 区段定义之前被定义。使用这些指示符以后，当前区段被重新定义。

6．8 操作其他模块的标号

要在一个模块里使用另一个模块里定义的标号，必须使用 GLOBAL 指示符。在指定标号为全局（GLOBAL）类型前，该标号必须先定义。使用这些标号的模块必须先使用 EXTERN 指示符来申明这些标号。使用指示符 GLOBAL 和 EXTERN 的范例如下：

6．8．1 可重定位代码，定义模块

```
                UDATA
InputGain       RES 1
OutputGain      RES 1
                GLOBAL InputGain, OutputGain

                CODE
Filter
                GLOBAL Filter
                :       ; Filter code
```

6．8．2 可重定位代码，参考模块

```
                EXTERN InputGain, OutputGain, Filter

                UDATA
        Reading RES 1

                 CODE
                 ...
                 MOVLW GAIN1
                 MOVWF InputGain
                 MOVLW GAIN2
                 MOVWF OutputGain
                 MOVF Reading,W
                 CALL Filter
```

6．9 分页（PAGEING）和分堆（BANKING）的问题

在很多场合，RAM 分配可能会占有多个堆，可执行代码会占有多个页。这就有必要执行相应的堆（BANK）和页（PAGE）的设置，以便标号正确地操作。然而在汇编时这些变量的绝对地址和地址标号都是未知的，而且通常做不到编写刚好够一个页的代码。鉴于这些情形，使用 BANKSEL 和 PAGESEL 指示符可以解决问题。这两个指示符可以指示链接器为指定标号生成正确的块和页选择码。下例显示代码应该如何转换：

6．9．1 绝对代码：

```
                LIST P=12C509
        #include “P12C509.INC”

      Varl      EQU H'10'
      Var2      EQU H'30'
                ...
                MOVLW InitialValue
                BCF FSR, 5
                MOVWF Varl
                BSF FSR, 5
                MOVWF Var2
                BSF STATUS, PA0
                CALL Subroutine
                ...
   Subroutine   CLRW ;In Page 1
                ...
                RETLW 0
```

6．8．2　可重定位代码：

```
        LIST P=12C509
      #include “P12C509.INC”

                UDATA
      Varl      RES 1
      Var2      RES 1
                ...
                CODE
                MOVLW InitialValue
                BANKSEL Varl
                MOVWF Varl
                BANKSEL Var2
                MOVWF Var2
                PAGESEL Subroutine
                CALL Subroutine
                ...
    Subroutine  CLRW
                ...
                RETLW 0
```

6．10　不再使用的指示符

当生成目标文件时，除了 ORG，几乎所有的指示符和宏功能都可以使用。可以象下面的例子，指定一个绝对代码段。

6．10．1　绝对代码

```
Reset ORG   H’01FF’
      GOTO  Start
```

6．10．2　可重定位代码

```
Reset CODE H’0lFF’
      GOTO Start
```

6．11　生成目标模块

一旦代码转换完成，可以从命令行或 SHELL 界面获取目标文件，从而产生目标模块。当使用基于 WINDOWS 的 MPASM 时，查看标记有“目标文件”（OBJECT FILE）的检查盒。当使用 DOS SHELL 界面时，点击“Assemble to Object File”（汇编为目标文件），选中“Yes”。输出文件将带一个“.O”扩展名。

6．12　代码范例

以下是从乘法例程中节选出来，作为 MPASM 的例子。一些注释被简化。

6．12．1　绝对代码：

```
            LIST P=16C54
            #INCLUDE “P16C5x.INC”

            cblock H ‘020’
      mulcnd RES 1              ; 8 bit 被乘数
      mulplr RES 1              ; 8 bit 乘数
      H_byte RES 1              ; 16 bit结果的高位字节
      L_byte RES 1              ; 16 bit结果的低位字节
      count  RES 1              ;  循环计数器

      mpy   clrf H_byte
            clrf L_byte
            movlw 8
            movwf count
            movf mulcnd,w
            bcf STATUS,C        ; 清除进位位
  Loop rrf  mulplr,F
           btfsc STATUS,C
           addwf H_byte,F
           rrf H_byte,F
           rrf L_byte,F
           decfsz count,F
           goto loop

           retlw 0
;*****************************************************
```

```
; Test Program
;*****************************************************
          start  clrw
                 option
          main   movf PORTB,w
          movwf mulplr          ; 乘数(in mulplr) = 05
          movf  PORTB,W
          movwf mulcnd
call_m    call mpy              ; 结果在 F12 和 F13中
                                ; H_byte & L_byte
          goto main

          ORG 01FFh
          goto start
          END
```

由于 8X8 乘法是非常有用的通用例程，如果能将它拿出来作为单独的目标文件，当需要的时候就可以链接它。上面的文件可以分为两个文件：一个是安排功能的调用文件(CALLING FILE),一个是可以放置在库文件里的通用(GENERIC)例程。

6．8．2　可重定位代码，调用文件（CALLING FILE）

```
                    LIST P=16C54
                    #INCLUDE “P16C5x.INC”

                    EXTERN mulcnd, mulplr, H_byte, L_byte
                    EXTERN mpy

                      CODE
                start  clrw
                       option
                main   movf PORTB, W
                movwf  mulplr
                movf   PORTB, W
                movwf  mulcnd
          call_m   call   mpy        ; 结果在 H_byte 和 L_byte
                   goto main
            Reset  CODE H’0lFF’
                   goto start
            END
```

6．8．3　可重定位代码，库例程（LIBRARY ROUTINE）

```
        LIST P=16C54
        #INCLUDE “P16C5x.INC”

        UDATA
mulcnd  RES l             ; 8 bit 被乘数
mulplr  RES 1             ; 8 bit 乘数
H_byte  RES 1             ; 16 bit结果的高位字节
L_byte  RES 1             ; 16 bit结果的低位字节
count   RES 1             ; 循环计数器
        GLOBAL mulcnd, mulplr, H_byte, L_byte

        CODE
mpy
         GLOBAL mpy
         clrf H_byte
         clrf L_byte
         movlw 8
         movwf count
         movf muland, W
         bcf STATUS, C          ;清除进位位
loop     rrf mulplr, F
         btfsc STATUS, C
         addwf H_byte, F
         rrf   H_byte, F
         rrf   L_byte, F
         decfsz count, F
         goto loop

         retlw 0
 END
```


7．1 概述

所谓“宏”是用户自定义的指令集或指示符，无论什么时候调用宏，它都将被逐行插入到源汇编文件之中的相应位置。

宏由一系列的汇编指令和指示符组成。宏可以接受参数（arguments）使之应用更加灵活。其优点有 ：

- 具有较为高级的抽象逻辑，增加了可读性和可靠性。
- 对经常使用的函数提供一致的解决方案；
- 使程序的修改变得简单；
- 提高了程序的可测试性；

应用程序可能包括建立复杂的表格，经常使用的代码以及复杂的运算。

7．2 重点

本章内容主要包括：

- 宏语法
- 宏指示符
- 文本替换
- 宏的用法
- 代码范例

7．3 宏语法

MPASM 的宏语法按如下格式定义：

```
<标号> macro [<参数1>,<参数2> ..., <参数n>]
:
:
 endm
```

其中<标号>是合乎 MPASM 规范的标号， <参数>是赋予宏使用的相应数值的可选参数（argument）。当有宏调用的时候，赋予这些参数的数值将会替换在宏里任何地方用到的这些参数。

在宏体里面，可能包含 MPASM 指示符，PICmicro 系列 MCU 的汇编指令，或者 MPASM 宏指示符（比如：(LOCAL）。参考第 5 章。MPASM 将不断处理宏体里的语句，直到遇见 ENDM 或 EXITM。

注意：不允许对宏进行向前引用（Forward reference）

7．4 宏指示符

宏定义有如下指示符，但这些指示符不能在宏之外使用（关于这些内容请参考第 7 章）：

- MACRO
- LOCAL
- EXITM
- ENDM

注意：旧版本的宏指示符“dot”已经不再使用。鉴于兼容性的原因，使用

该指示符的旧 ASM17 代码仍然可以使用 MPASM 汇编，但正如前所述，你最好使用本手册介绍的方法编程，以确保向上与 MPASM 兼容。

7．5 文本替换

在一个宏里面，可能有字符串的替换和表达式的分析操作（expression evaluation）。

命令	描　　述
<参数>	宏调用时，宏体里的参数名全部被<参数>的值所替换
#v(<表达式>)	返回<表达式>的整型值。通常用来创建带公共前缀或后缀的唯一变量名。不能用 IFDEF，WHILE 等条件汇编

参量（ARGUMENT）可以用在宏体里的任何地方，正常表达式的一部分例外，比如以下宏：

```
define_table macro
            local a = 0
            while a < 3
  entry#v(a) dw 0
  a += 1
            endw
            endm
```

当调用时，生成以下代码：

```
            entry0 dw 0
            entry1 dw 0
            entry2 dw 0
            entry3 dw 0
```

7．6 宏的用法

一旦宏被定义，则可使用宏调用在源程序的任何地方对它进行调用，例如：

<宏名> [<参数>, ..., <参数>]

其中：<宏名>是预先定义的宏名称，并且按要求赋予参量。

宏调用并不占用任何存储器位置。然而，宏扩展将从当前的存储器位置开始。逗号可用来保留参数位置。在这种情况下，参数将是一个空的字符串。参数表由空格（space）或分号（semicolon）结束。

指示符 EXITM（见第 5 章）提供了另外一种结束宏扩展的方法。宏扩展期间，EXITM 使宏扩展停止，所有 EXITM 和 ENDM 之间的语句将被忽略。假如宏嵌套，则 EXITM 会使代码返回到上一级宏扩展。

7．7 代码范例

7．7．1 8X8 乘法：

```
subtitle “macro definitions”
page
;
; 乘法 – 8X8乘法宏, 在内存中运行，优化速度，直行代码
```

```
; 基于 PIC17C42微控制器
;
multiply macro arg1, arg2, dest_hi, dest_lo
                          ;
        local i = 0       ; 建立局部索引变量
                          ; 并初始化
                          ;
         movlw arg1       ; 设置乘数
         movwf mulplr     ;
                          ;
         movlw arg2       ; 在W寄存器中设置被乘数
                          ;
         clrf dest_hi     ;清除目标寄存器
         clrf dest_lo     ;
                          ;
         bcf _carry       ; 清除进位位，以便测试
                          ;
         while i < 8      ; 操作位数为8
         addwf dest_hi    ; 加上被乘数
         rrcf dest_hi     ; 通过进位位右移CY
         rrcf dest_lo     ; 再右移，假若CY被上一次右移置位
                          ; 则将CY清除
  i += 1                  ; 循环计数器加一
         endw             ;8次迭代后退出
         endm             ; 结束宏
```

宏申明所有需要的参数，本范例中共有 4 个参数。用 LOCAL 伪指令建立局部变量“i”，它被用作索引计器，并被初始化为 0。

随即包含许多汇编指令。执行宏的时候，这些指令将会逐行插入源汇编代码之中。

宏使用一种算法来进行乘运算，这种算法的原理是：对乘数进行右移和逐位相加。WHILE 指示符担当此功能，当 i>=8 之前继续循环。

循环的终点由 ENDW 来指示，当 WHILE 的条件为真（TRUE）时，则继续执行 ENDW 后面的语句。整个宏由 ENDM 指示符结束。

7.7.2 常量比较

另外一个范例。假如编写了下列宏：

```
include “16cxx.reg”
;
;文件和常数相比较，如果文件>=常量，则跳转
;
cfl_jge macro file, con, jump_to
        movlw con & 0xff
```

```
            subwf file, w
            btfsc status, carry
            goto jump_to
        endm
```

如果它被以下调用时：

```
        cfl_jge switch_val, max_switch, switch_on
```

则会生成以下代码：

```
        movlw max_switch & 0xff
        subwf switch_val, w
        btfsc status, carry
        goto switch_on
```


8．1　概述

本章介绍 MPASM 的各种表达式，语法和运算。

8．2　重点

本章内容包括：

- 文本字符串
- 数字常量和进制基数（RADIX）
- 运算符号和优先级别
- 高/低/上位及增/减操作符号

8．3　文本字符串

“字符串”是用双引号包含的，一切合法的 ASCII 字符（10 进制 0-127）。

字符串的长度为小于 255 的任何常数。如果找到匹配的引号，就结束字符串；假如未找到引号，则该字符串在行的末尾结束。如果没有找到可以继续到下一行的标志，在下一行使用另一个“DW”指示符一般不会有问题。

DW 指示符将会把整个字符串存储到存储器中地址连续的字（WORD）里面。假如该串有奇数个字符（BYTE），则 DW 和 DATA 指示符会在字符串末尾填充为字节“00”

假如字符串用作文本操作数，则必须是一个严格的字符长度，否则会出错。

参考下例，了解与字符串相关的不同语句所生成的目标代码：

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

汇编程序还接受某些特殊控制字符的 ANSI 转义操作：

表 8-1 ANSI‘C’转义命令符：

转义命令符	描 述	HEX 值
\a	振铃（报警）	07
\b	退格	08
\f	换页	0C
\n	换行	0A
\r	回车	0D
\t	水平制表	09
\v	垂直制表	0B
\\	反斜杠	5C
\?	问号	3F
\′	单引号	27
\″	双引号	22
\0OO	8 进制数（格式：0，8 进制数，8 进制数）	
\xHH	16 进制数	

8．4 数值常量和进制基数

MPASM 支持以下进制基数（RADIX）：16 进制（HEXDECIMAL），10 进制（DECIMAL），8 进制（OCTAL），2 进制（BINARY）和 ASCII 字符。默认的进制是 16 进制；如果没有使用进制说明符规定进制基数，就使用默认进制给目标文件分配常量数值。

常量可以带一个正“+”号或者负“-”号。如果没有指定符号，则认为是正“+”。

注意：常量表达式中的立即数被当作 32-BIT 无符号数处理。任何时候如果试图把常量放置到一个不能容纳该常量的区域里，则会输出截断（TRUNCATION）警告。

下表是各种进制基数的详细规定：

表 8-2 进制基数：

类型	语法	范例
10 进制	D′<数字>′	D′100′
16 进制	H′<16 进制数字>′ 0x<16 进制数字>	H′9f′ 0x9f
8 进制	O′<8 进制数字>′	O′777′
2 进制	B′<2 进制数字>′	B′00111001′
ASCII	′<字符>′ A′<字符>′	′C′ A′C′

表 8-3 运算符与优先级别：

运算符		范例
$	返回程序计数器	goto $ + 3
(	左括号	1 + (d * 4)
)	右括号	(Length + 1) * 256
!	逻辑“非”	if ! (a == b)
-	负	-1 * Length
~	按位取反	flags = ~flags
high	返回高字节	movlw high CTR_Table
low	返回低字节	movlw low CTR_Table
upper	返回上位字节	movlw upper CTR_Table
*	乘	a = b * c
/	除	a = b / c
%	取模	entry_len = tot_len % 16
+	加	tot_len = entry_len * 8 + 1
-	减	entry_len = (tot - 1) / 8
<<	左移	flags = flags << 1
>>	右移	flags = flags >> 1
>=	不小于	if entry_idx >= num_entries
>	大于	if entry_idx > num_entries
<	小于	if entry_idx < num_entries
<=	不大于	if entry_idx <= num_entries
==	等于	if entry_idx == num_entries
! =	不等于	if entry_idx != num_entries
&	位“与”	flags = flags & ERROR_BIT
^	位“异或”	flags = flags ^ ERROR_BIT
|	位“同或”	flags = flags | ERROR_BIT
&&	逻辑“与”	if (len == 512) && (b == c)
||	逻辑“或”	if (len == 512) || (b == c)
=	赋值	entry_index = 0
+=	加，再赋予	entry_index += 1
-=	减，再赋予	entry_index -= 1
*=	乘，再赋予	entry_index *= entry_length
/=	除，再赋予	entry_total /= entry_length
%=	取模，再赋予	entry_index %= 8
<<=	左移，再赋予	flags <<= 3
>>=	右移，再赋予	flags >>= 3
&=	与，再赋予	flags &= ERROR_FLAG
|=	同或，再赋予	flags |= ERROR_FLAG
^=	异或，再赋予	flags ^= ERROR_FLAG
++	加 1	i ++
--	减 1	i --

8．5　字节的高/低/上位

8．5．1 语法

```
high <操作数>
low <操作数>
upper <操作数>
```

8．5．2 描述

该运算符用于返回一个针对多字节标号（multi-byte label）的字节。这可以用于表格读写时进行动态指针运算。

8．5．3 范例

```
movlw low size ; handle the lsb’s
movpf wreg, low size_lo
movlw high size ; handle the msb’s
movpf wreg, high size_hi
```

8．6 增/减操作（++/--）

8．6．1 语法

```
<变量>++
<变量>--
```

8．6．2 描述

使变量值加 1 或减 1 操作。这些运算符号只能在一行上由其本身使用。但它不能嵌入其他的表达式中。

8．6．3 范例

```
LoopCount = 4
   while LoopCount > 0
         rlf Reg, f
LoopCount --
   endw
```

第9章 初始化代码范例

9．1 介绍

本章介绍给各种 PICmicro 系列 MCU 的初始化代码。

9．2 重点

本章介绍：

- 初始化代码

9．3 初始化代码范例

假如生成目标模块时使用了 IDATA 指示符，用户必须编制初始化代码。这些代码可以使用并根据需要修改。你可以在 MPLINK 提供的应用范例里找到，也可以在 WEB 网（www.microchip.com）上下载。

第二部分 MPLINK 链接程序

第1章　MPLINK 概述

1．1　介绍

本章说明什么是 MPLINK 以及如何用它来开发 PIC 单片机支持软件。

1．2　重点

本章内容包括：

- 什么是 MPLINK
- MPLINK 是干什么的？
- MPLINK 能帮你干什么？
- 开发工具及支持平台

1．3　什么是 MPLINK？

MPLINK 是 MICROCHIP 的编译器 MPASM 以及 MPLAB C17/C18 高级语言编译器的链接器软件。详细内容参见本指南的第一部分。关于 MPLAB C17/C18 的详细内容，参见《MPLAB-C17/C18 用户指南》（文档资料编号 DS51112）。

MPLINK 也可以和 MICROCHIP 的库函数 MPLIB 一起使用。

MPLINK 被设计来与 MPLAB 一起使用，但也可以单独使用。当一个应用程序建立以后，最好还是用 MPLAB 里的模拟器或仿真器验证和调试，并用编程调试。详细细节参见 MPLAB 用户指南（文档号 DS51025）。语言

1．4　MPLINK 是干什么的？

MPLINK 有许多功能：

- **定位代码和数据：**作为输入可重定位源文件，使用链接器可以把代码放到 ROM 中相应的位置，把变量放到 RAM 中相应的位置。
- **决定地址：**一个源文件中的外部参考信息在目标文件中生成可重定位路径。当 MPLINK 把代码和数据定位后，就使用此重定位信息来更新所有外部参考，使它们转化为实际的物理地址。
- **产生可执行代码：**产生一个扩展名为.HEX 的文件，该文件可以用编程器写入 PIC 单片机中，也可以被调入模拟器或仿真器中运行。
- **设立堆栈的大小和位置：**允许 MPLAB C17/C18 在 RAM 空间中设置动态堆栈。
- **检查地址冲突：**检查是否有与已经分配或保留的程序/数据空间相冲突的情形。
- **提供符号调试信息：**输出一个文件，该文件可以为 MPLAB 在源程序级调试时候提供地址标号，变量定位，行/文件信息。

1．5　MPLINK 如何帮助你？

MPLINK 允许你处理模块,可再用代码.在命令行状态下,使用链接描述语言可以控制链接过程.MPLINK 保证所有符号参考都被处理,所有代码和数据都被放置到 PICmicro 之中。

MPLINK 可以帮助你：

- 可再用代码.你可以把应用程序建立成较小的可再用代码模块.
- 库函数.你可以建立相关功能的库函数.这样一来效率更高,兼容性和可靠性更好。
- MPLAB-C17/C18.MICROCHIP 的 MPLAB-C17/C18 编译器需要使用 MPLINK,,它还可以使用预编译库和宏汇编 MPASM。
- CENTRALIZED 存储器定位.通过使用特定功能的链接指令,可以在链接时而不时在编译和汇编时将经过预先处理的目标代码和库与新模块组合起来,有效地放置到可用的存储器中去。
- 加速开发过程.因为已测试模块和库函数不必在每次代码变化时都做编译,所以加快了编译速度。

1．6 MPLINK 范例

本部分结尾你将看到四个范例。教你如何使用 MPLINK。这些范例也可以作为你自己应用程序的参考。

- 范例 1

如何把程序代码放到不同的存储区
如何把数据表放置到 ROM 里
如何在 C 程序中设置配置位

- 范例 2

如何为启动引导器（BOOT LOADER）在内存中定位
如何编译将要调入外部 RAM 中执行的程序

- 范例 3

如何操作存储器映射的外设
如何建立新的区段

- 范例 4

如何为程序手动配置 RAM

1．7 支持平台

MPLINK 可以支持两者操作平台：一种是适合于 WINDOWS95/98/NT 的版本，另一种是适合于 WINDOWS3.X 和 DOS 平台版本。基于 DOS 平台的应用文件的文件名以“d”结尾，以便和 WINDOWS 版本相区别。DOS 版本需要 DOS 扩展程序“DOS4GW.EXE“支持，该程序和 MPLINK 一起包含在软件包中。

第2章 MPLINK 的安装与入门

2．1 介绍

本章介绍如何安装 MPLINK 到你的机器上；初步介绍 MPLINK 链接器的工作原理。

2．2 重点

本章内容包括：

- 安装
- 链接器概述
- 链接器输入/输出文件

2．3 安装 MPLINK

MPLINK 有两种版本，MPLINKD.EXE 是 DOS 扩展版本建议只在 DOS 和 WINDOWS3.X 环境下运行；MPLINK.EXE 是 WINDOWS32 位应用程序，工作于 WINDOWS95/98/NT 环境。

MPLINKD.EXE 是命令行应用程序，用于 DOS 和 WINDOWS3.X 环境，MPLINK.EXE 可以用于 WINDOWS95/98/NT。用户可以在 MPLAB 环境中设置，使用它。

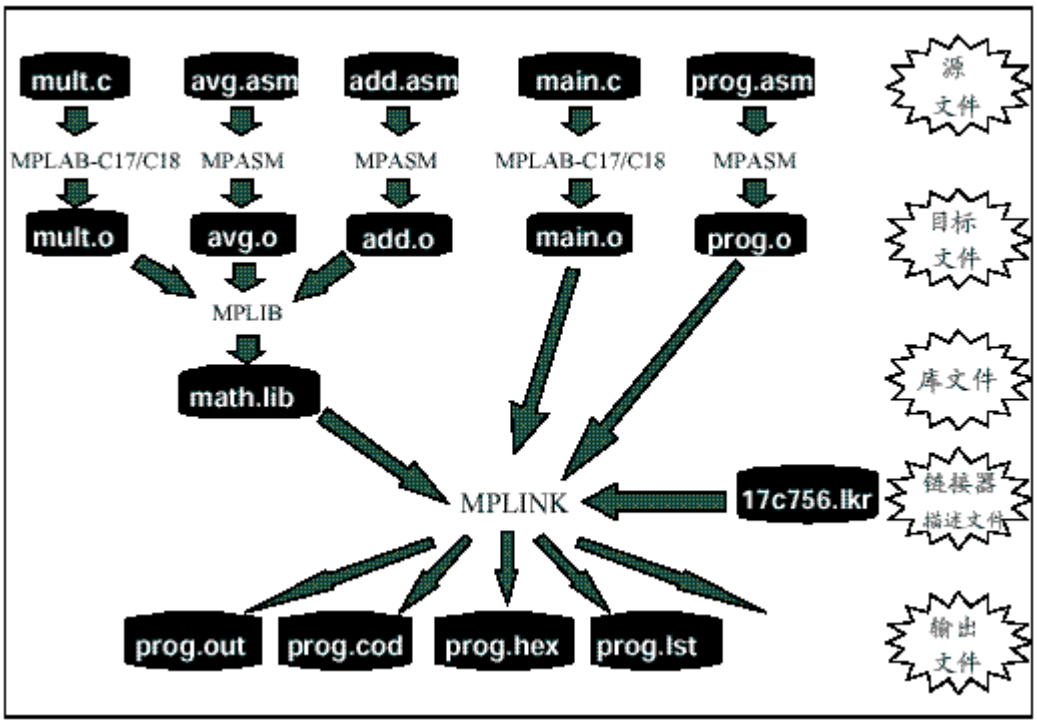
若你想在 MPLAB 中使用 MPLINK，没有必要单独安装 MPLINK，当 MPLAB 安装完后 MPLINK 也已经安装完毕。参见《MPLAB 用户指南》（文档资料号 DS51025）获取详细内容。你既可以从 MICROCHIP 的 WEB 网站下载，也可以索取技术资料 CD-ROM 得到 MPLAB 软件包。

若你只想得到 MPLINK 应用程序，也可以采用以上方法。MPLINK 在 MPLAB 软件包中以 ZIP 形式的文件存在，安装时按以下步骤：

- 建立一个目录将文件放在其中
- 用 WINZIP 或 PKZIP

2．4 链接器概述

图 2-1：MPLINK 和 MICROCHIP 其他开发工具配合工作



MPLINK 将 MPASM 以及 MPLAB-C17/C18 产生的多个输入目标模块合并为一个可执行文件（.HEX）,链接指令告诉链接器如何把这些模块合并起来。

在用 MPASM 汇编或 MPLAB-C17/C18 编译生成可重定位目标模块以后将分配数据的实际地址和函数的定位，也就是说，你可以通过链接器指令来控制链接器，把代码和数据放到指定的存储器器区域内某个地方。这些指定区域可能是特定的物理位置，也可能不是。

链接器指令告诉 MPLINK 关于现有的目标设备上可用的 ROM 和 RAM 资源。这样一来 MPLINK 才可以分析所有输入文件，再把程序代码放到相应的 ROM 中去，数据放到相应的 RAM 里。若输入的代码太大或者变量太多以至超出限定，MPLINK 将给出错误信息。

MPLINK 还可以灵活地指定数据存储器中某些可以重新使用的块。这样一来不同的例程可以享用有限的 RAM 空间。（注意：这些例程应互相不调用，而且当程序之间切换时，不依靠这些空间作为数据保护的介质。

有许多 PICmicro 的外设函数和 C 函数都有现成的库函数可供调用，（更多信息见第 2 章）链接器只从包含库文件中取出当前应用程序中用到的目标文件进行链接。这意味着较大的库函数可以有效率地被使用。、

MPLINK 把所有输入文件合并，产生可执行代码，而且确保所有地址都被分配。假如在许多输入模块中的任何函数试图访问一个没有定位或建立的例程，都将导致 MPLINK 生成一个错误信息。

2．5 链接器输入/输出文件

MPLINK 是一个将多个目标文件合并为一个可执行文件（.HEX）的链接器。链接器用到的输入文件类型有：

- 目标文件（.O）由源程序产生的可重定位代码。
- 库文件（.LIB）.由一系列应用目标文件组成，集合成库，方便调用。

- 链接器描述文件（.LKR）针对某一处理器或项目，对存储器的安排作一个描述。

注意：当建立列表文件时，要用到源文件（C 或汇编）

链接器用到的输出文件类型有：

- **COFF 文件（.OUT，.COF）：**MPLINK 生成.HEX 文件和列表文件时用到的中间文件。
- **代码文件（.COD）：**MPLAB 用到的调试文件。
- **HEX 文件（.HEX）：**二进制可执行代码，不包含调试信息。
- **列表文件（.LST）：**初始源代码和对应的二进制代码 一起列出。
- **映射文件（.MAP）：**显示链接后存储器分布图，显示出已经使用的和还未使用的存储器区域。

2．5．1 目标文件（.O）

目标文件是由源程序产生的可重定位代码。链接器的工作就是把源文件和苦文件组合起来，遵照链接器命令，转化为目标文件。

2．5．2 库文件（.LIB）

MPLAB 可以把目标文件转化为库文件，也可以由现有的标准库文件生成。关于库文件的详细信息，请看第三部分。

2．5．3 链接器描述文件（.LKR）

链接器描述文件是 MPLINK 的命令文件，详细情况请看第 5 章。

2．5．4 COFF 文件（.OUT，.COF）

MPLINK 产生一个.COFF 的中间文件文件，MP2COD.EXE 产生 COD 文件，MP2HEX.EXE 产生 HEX 文件。

2．5．5 代码与调试文件（.COD）

MPLINK 生成.COD 文件，供 MPLAB 调试代码使用。

2．5．6 HEX 文件（.HEX）

MPLINK 可以产生不同的 HEX 文件格式。附录 A 里有详细的介绍。

2．5．7 绝对列表文件（.LST）

绝对列表文件由 MPLINK 产生，假如想要查看该文件请选：***Window>Absolute Listing.*** 它提供源代码和机器指令间的映射关系。该窗口在每次重新创建（REBUILD）项目时被自动重新调入。欲知更多关于该文件格式的信息，请参考第一部分的 2.5.2 节。

2．5．8 映射文件（.MAP）

映射文件由 MPLINK 生成，选择 ***File>Open*** 并选择你在 MPLINK 选项中指定的文件名，就可以阅读该文件。它提供最终输出的源文件代

码符号的绝对位置信息，此外还提供内存情况信息，包括已经使用的和还未使用的存储器区域。该窗口在每次重新创建（REBUILD）时被自动重新调入。

映射文件包含 3 个表，第一个表（区段信息）显示每个区段的信息。信息包括每个区段的名称，类型，开始地址，该区段是否驻留在程序存储器或数据存储器中，该区段的大小（节数）。

区段有四种类型：

- 代码
- 已初始化数据（IDATA）
- 未初始化数据（UDATA）
- 已初始化 ROM 数据

以下范例是一个映射文件里的区段表：

Section Info

Section	Type	地址	Location	Size(Bytes)
Reset	code	0x000000	program	0x000002
.cinit	romdata	0x000021	program	0x000004
.code	code	0x000023	program	0x000026
.udata	udata	0x000020	data	0x000005

映射文件里的第二个表（符号-根据名称索引）提供关于输出模块里的符号信息。表格根据符号的名称来索引，其中还包括符号的地址，以及符号是否驻留在 ROM 或 RAM 中,符号是否有外部或静态链接(STATIC LINKAGE)，在哪里定义文件名。下列范例是一个映射文件中按符号名称索引的符号表：

Symbols - Sorted by Name

Name	Address	Location	Storage	File
call_m	0x000026	program	static	C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
loop	0x00002e	program	static	C:\MPASMV2\MUL8X8.ASM
main	0x000024	program	static	C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
mpy	0x000028	program	extern	C:\MPASMV2\MUL8X8.ASM
start	0x000023	program	static	C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
H_byte	0x000022	data	extern	C:\MPASMV2\MUL8X8.ASM
L_byte	0x000023	data	extern	C:\MPASMV2\MUL8X8.ASM
count	0x000024	data	static	C:\MPASMV2\MUL8X8.ASM
mulcnd	0x000020	data	extern	C:\MPASMV2\MUL8X8.ASM
mulplr	0x000021	data	extern	C:\MPASMV2\MUL8X8.ASM

映射文件里的第三个表（符号-根据地址索引）提供关于输出模块里的符号信息。表格根据符号的地址来索引而不是根据符号的名称来索引。下列范例是一个映射文件中按符号地址来索引的符号表：

```
Symbols - Sorted by Address
Name     Address   Location  Storage    File
-------- -------- -------- -------- ---------
start   0x000023 program static C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
main    0x000024 program static C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
call_m  0x000026 program static C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
mpy     0x000028 program extern C:\MPASMV2\MUL8X8.ASM
loop    0x00002e program static C:\MPASMV2\MUL8X8.ASM
mulcnd  0x000020 data extern    C:\MPASMV2\MUL8X8.ASM
mulplr  0x000021 data extern    C:\MPASMV2\MUL8X8.ASM
H_byte  0x000022 data extern    C:\MPASMV2\MUL8X8.ASM
L_byte  0x000023 data extern    C:\MPASMV2\MUL8X8.ASM
count   0x000024 data static    C:\MPASMV2\MUL8X8.ASM
```

如果链接有错，就不会生成一个完整的映射文件。然而，如果在选项里有“-m”参数，错误映射文件将生成，该文件只包含区段信息，而没有符号信息。错误映射文件提供足够的信息，以便分析为什么一个区段无法定位。

第3章　在 DOS 下使用 MPLINK

3．1　概述

本章介绍如何在 DOS 命令行下使用 MPLINK。

3．2　重点

本章介绍的内容：

- 链接器命令行选项

3．3 链接器命令行的参数选择

使用 MPLAB 时，在编辑项目会话窗口（**Edit Project**）里的节点属性栏（**Node Property**）里，可以找到所有 MPLINK 的选项。要想知道更多在 MPLAB 中使用 MPLINK 的信息，请看下一章。

当以批处理方式或者直接在 DOS 下使用 MPLINK 时，可以使用以下语法来调用 MPLINK：

```
mplink { cmdfile | [objfile] | [libfile] | [option] } . . .
```

要求必须有一个命令文件（cmdfile）和至少一个目标文件(objfile)。

“cmdfile”是链接器命令文件。所有链接器命令文件都有扩展名“.lkr”。

“objfile”是汇编器或编译器生成的目标文件。所有目标文件都必须有扩展名“.O”。

“libfile”是生成的库文件。所有库文件都有扩展名“.lib”。

“option”是链接器命令行选项。见表 3-1：

表 3-1：链接器命令行选项

选项	描述
/o 文件名	指定输出文件名（filename），默认为 a.out
/m 文件名	建立映射文件（filename）
/l 路径	增加库文件搜索路径（pathlist）
/k 路径	增加链接器命令搜索路径（pathlist）
/n 长度	设置列表页的行数（length）
/h, /?	显示帮助菜单
/a hex 格式	设置 16 进制文件输出格式
/q	静默模式
/d	不生成绝对列表文件

命令行参数没有严格的先后顺序，然而，改变一个顺序可能影响到链接器的操作。具体地说，增加的库/目标文件目录搜索路径被附加到当前库/目标文件目录搜索路径的后面-正如在命令行和命令文件里遇到的一样。

对库/目标文件所在目录的搜索顺序在库/目标文件目录搜索路径里设置。因此，改变目录的顺序可能改变所选择的文件。

参数“/o”给 COFF 输出文件提供文件名。链接器还生成一个供 MPLAB 调试用的 COD 文件和一个 INTEL 格式的 16 进制编程文件。这两个文件和 COFF 文件有相同的文件名，只不过扩展名分别为“.cod”和“hex”。假如“/o”参数没有指定，默认的 COFF 输出文件名为“a.out”。相应的 COD 和 HEX 文件的名称为“a.cod”和“a.hex”。

例 3-1：

```
MPLINK 17C756.LKR MAIN.O FUNCT.O ADC17.LIB /m MAIN.MAP /o MAIN.OUT
```

该指令指示 MPLINK 使用 17C756.LKR 链接命令文件将输入模块 MAIN.O，FUNCT.O 和预编译库 ADC17.LIB 链接起来。它还指令 MPLINK 产生一个名为 MAIN.MAP 的映像文件。MAIN.O 和 FUNCT.O 必须使用 MPLAB-C17/C18 和 MPASM 进行预编译或汇编。假如链接过程中没有出现错误，则会生成输出文件 MAIN.HEX 和 MAIN.COD。

第4章　在 WINDOWS 下 MPLAB 软件包中使用 MPLINK

4．1　概述

本章讲述如何在 MPLAB 软件包环境下使用 MPLINK。

4．2　重点

本章内容包含：

- MPLAB 项目和 MPLINK
- 设置 MPLAB 以便使用 MPLINK
- 生成输出文件
- MPLAB/MPLINK 故障排除

4．3　项目（PROJECT）和链接（MPLINK）

MPLAB 的项目由节点（NODE）生成，这一系列文件都被项目所使用。

- 目标节点-元件
  16 进制文件
- 项目节点-元件
  C 源文件
  汇编源文件
  库文件
  预编译目标文件
  链接器命令文件

本章将讨论 MPLAB 和 MPLINK 的关系。关于 MPLAB 项目更详细的信息请参考**《MPLAB 用户指南》**（文档资料号 DS51025）。

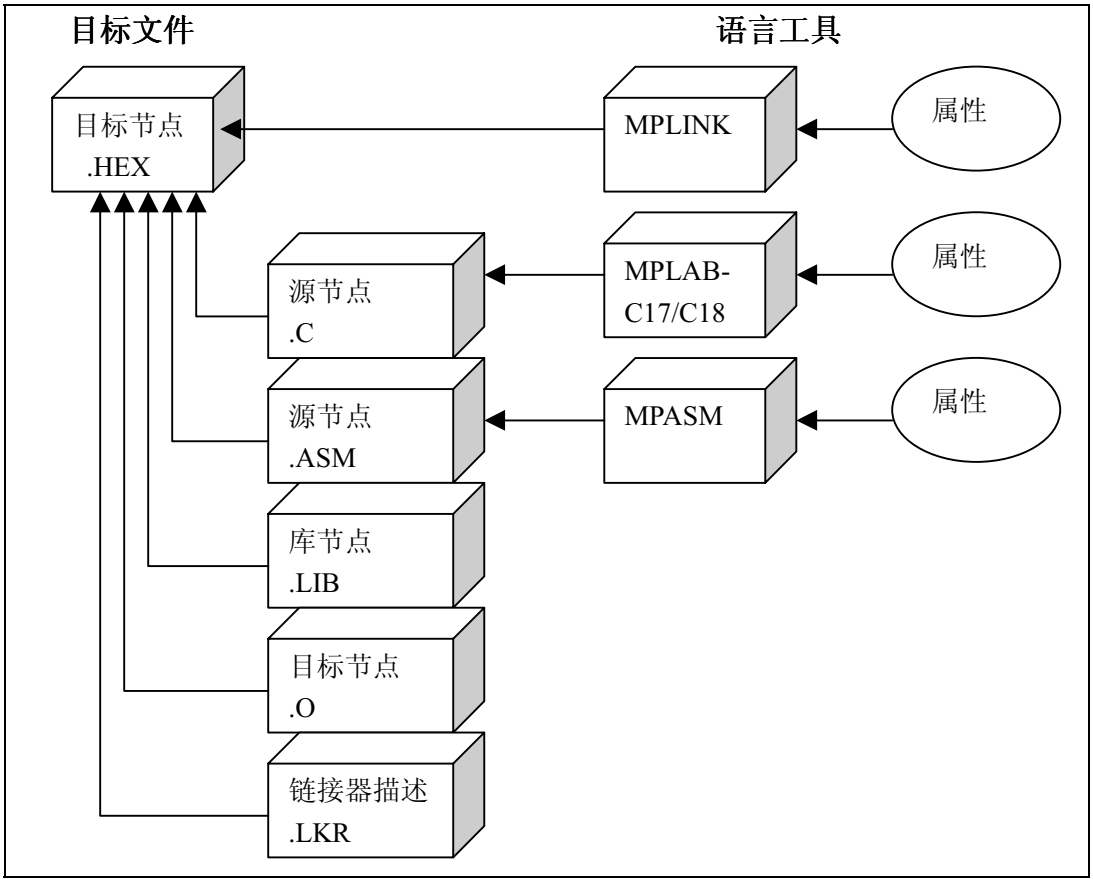


图 4-1　项目内部关系 – MPLINK

项目选用一些语言工具，比如：汇编，编译器，链接器来处理源程序，从而生成可执行文件（.HEX）。本图表显示最终的.HEX 文件和用来生成该文件的元件。

4．4 设置 MPLAB 以便使用 MPLINK

按照如下步骤设置 MPLINK 以便在 MPLAB 环境下使用。

1．建立项目以后（***Project>New Project***），编辑项目会话窗口将会出现。在项目文件列表里选择一个项目名称（比如：AD.HEX），则立即会激活属性（***Node Properties***）按钮（文字由虚变实）。并点击钮。

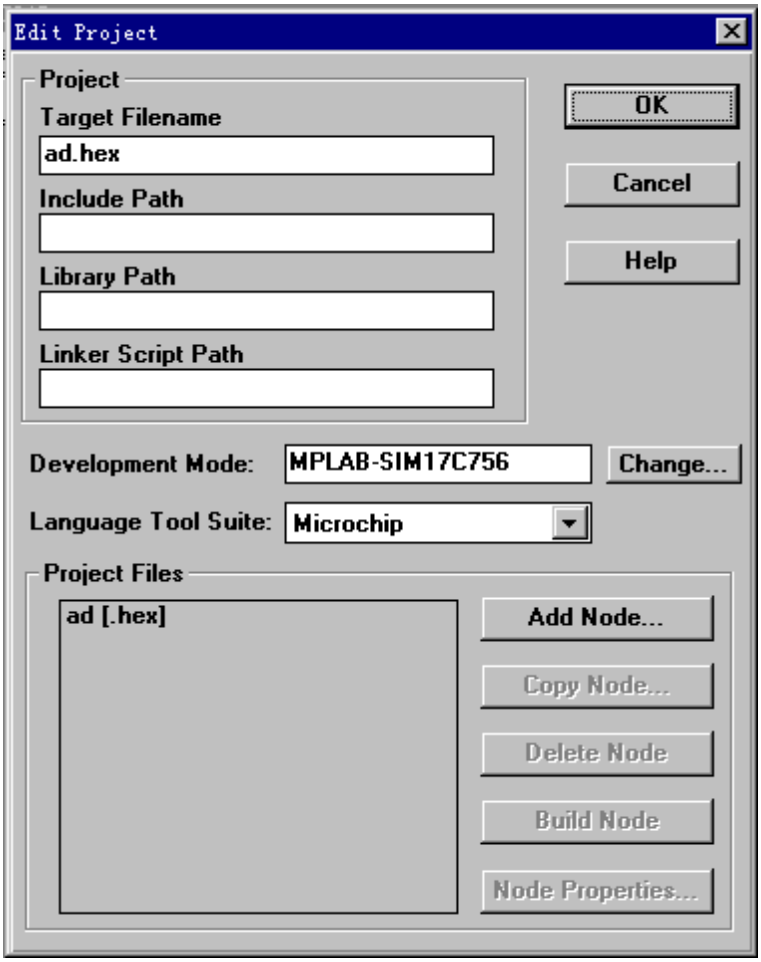


图 4-2 编辑项目会话窗口

2. 在“节点属性（***Node Properties***）”会话窗口里，选择 MPLINK 作为语言工具，如果需要，还可以选择其他链接器选项。

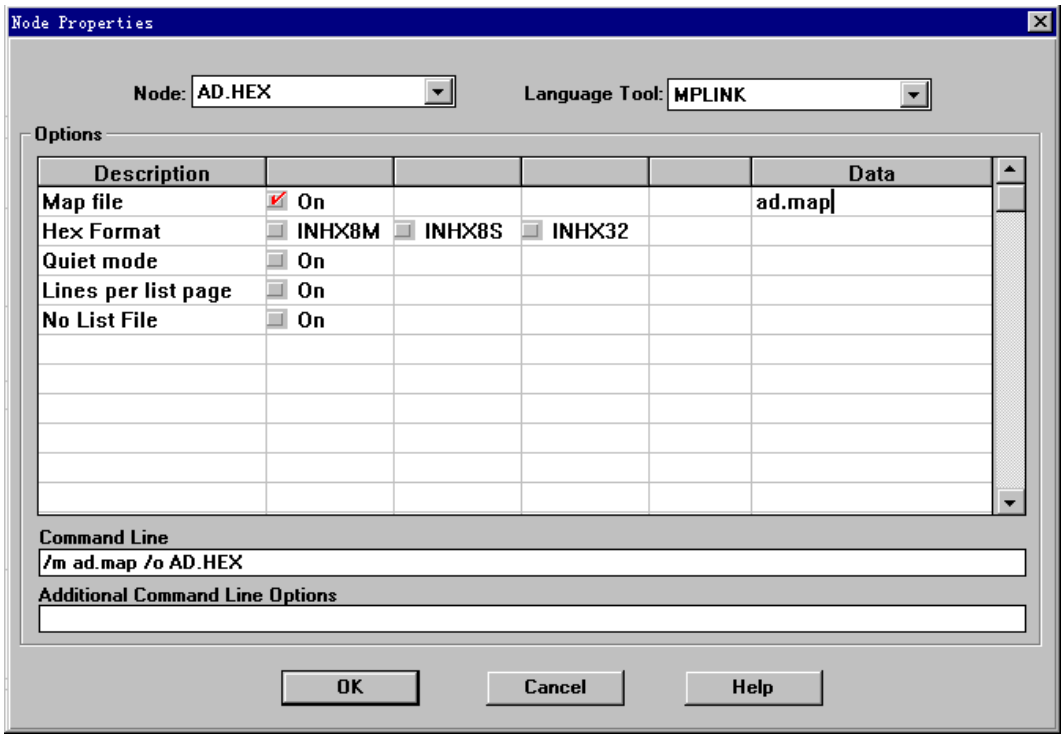


图 4-3 节点属性会话窗口

3. 在项目中加入文件。在“编辑项目（***Edit Project***）”会话窗口里选择“增加节点（***Add Node***）”按钮，该会话窗口就会弹出。选择一个源文件并点击“节点属性（***Node Properties***）”按钮，设置用来将源文件编译为目标文件的工具。

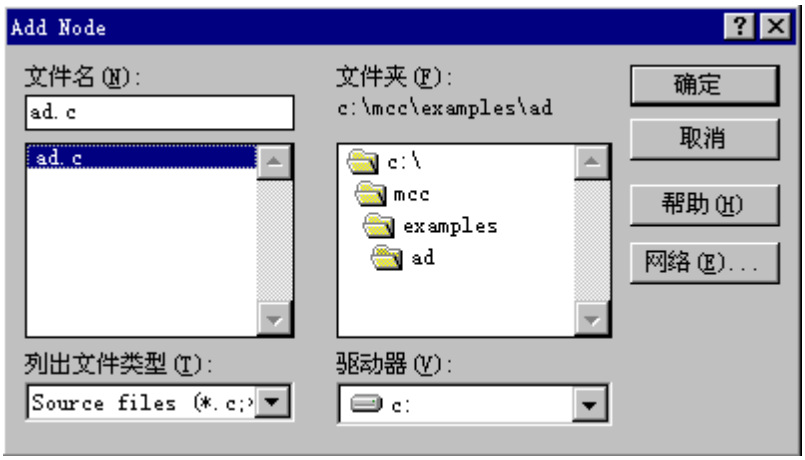


图 4-4 增加节点会话窗口 – 源文件

4. 添加预编译的目标文件和库文件（比如：包含于 MPLAB-C17/C18 中的库文件）。你可以选择一个以上的文件，方法是：点击多个文件时按住“**CTRL**” 键不放，可以选中多个文件。

图 4-5　增加节点会话窗口 – 目标文件

5. 将链接器描述文件作为一个节点加入到项目中。

图 4-6　增加节点会话窗口 – 链接器描述文件

6. 完成以上操作后，在编辑项目会话窗口里点击“**OK**”。

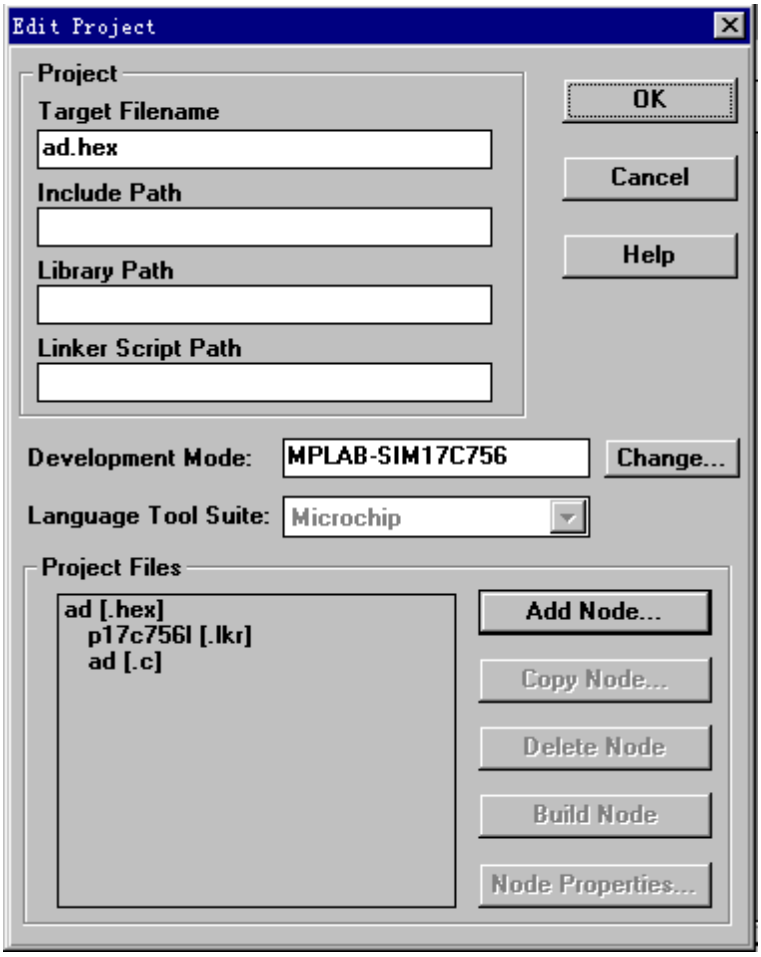


图 4-7　编辑项目会话窗口 – 接点已添加

4．5　生成输出文件

一旦MPLAB项目设置好后，就可以“建立（BUILD）”项目了。选择“项目>建立项目（***Project>Make>Project***）”就可以了。16进制文件被自动装载到程序存储器里。关于16进制文件需要说明的是，MPLINK生成其他文件帮助你调试代码。关于这些文件和它们的功能，详细情节见2.5节。

4．6　MPLAB/MPLINK 疑难解答

假如遇到问题，请按下面步骤检查：

选择“项目>安装语言工具...（***Project>Install Language Tool...***）”，检查 MPLINK 是否指向 MPLAB 安装目录里的文件 MPLINK.EXE，而且命令行选项已经设置好。

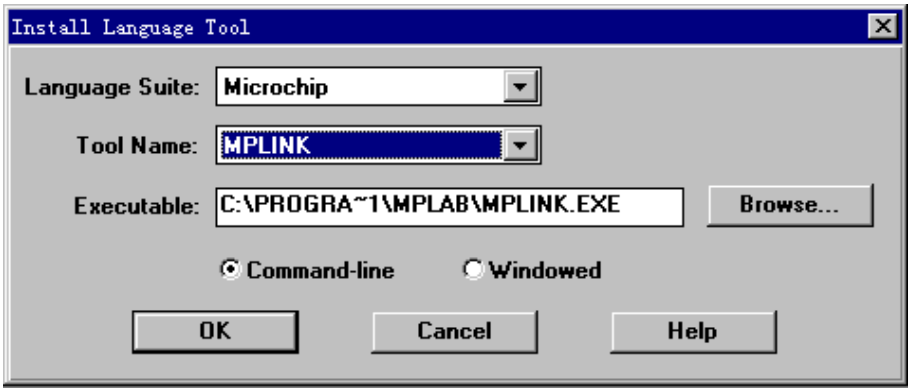


图 4-8　安装语言工具会话窗口 - MPLINK

第5章 MPLINK 链接器命令

5．1 概述

本章介绍如何制作和修改链接器命令，以便和 MPLINK 一起工作。

5．2 重点

本章内容包括：

- 链接器命令定义
- 命令行信息
- 存储器区定义
- 逻辑区段定义
- 堆栈定义
- 链接器命令信息

5．3 链接器命令定义

链接器命令文件是 MPLINK 的命令文件。它们设置以下参数：

- 目标系统的程序存储器和数据存储器
- 堆栈的深度和位置（针对 MPLAB-C17/C18）
- 源代码中用来放置代码和数据的区段

控制语言中的链接器命令指示符用来控制链接器的操作方式的，它主要分为四类基本类型。以下几节将详细介绍这些指示符和一些有用的链接器命令指示符。

链接器命令的注释内容用‘//’指定，比如：任何‘//’一直到该行末尾的文本均被忽略。

5．4 命令行信息

MPLAB 项目管理器可以直接设置该信息。假如从命令行进行链接，可能你只需使用这些信息就够了。

库文件路径（LIBPATH）：

对于没有路径的库和目标文件，将按照“库/目标”文件搜索路径进行寻找。以下指示符给“库/目标”文件搜索路径添加额外的搜索目录。

```
LIBPATH 'libpath'
```

其中：`'libpath'`是用分号界定的目录列表。

例如：

如果要将当前目录（**directory**）和目录 `C:\PROJECTS\INCLUD`E添加到“库/目标”文件搜索路径中，则要在链接器命令文件（linker command file）中加入以下信息行：

```
LIBPATH .;C:\PROJECTS\INCLUDE
```

链接路径（LKRPATH）

对于没有路径的链接器命令文件，将按照链接器命令文件搜索路径进行寻找。以下指示符给链接器命令文件搜索路径添加额外的搜索目录：

```
LKRPATH 'lkrpath'
```

其中：`'lkrpath'` 是用分号界定的目录列表。

例如：

如果要将当前目录的父目录和目录 C:\PROJECTS\SCRIPTS 添加到链接器命令文件搜索路径中，则要在链接器命令文件（linker command file）中加入以下信息行：

```
LKRPATH ..;C:\PROJECTS\SCRIPTS
```

文件（FILES）

以下指令为链接器指定目标或库文件：

```
FILES 'objfile/libfile' ['objfile/libfile'...]
```

其中：'objfile/libfile'是目标文件或者库文件。需要注意的是，一个 FILES 指示符可以指定一个以上的目标或库文件。

例如：

假若要将目标模块 main.o 和库文件 math.lib 链接在一起，则要在链接器命令文件中加入以下行：

```
FILES main.o math.lib
```

包含（INCLUDE）：

以下指示符包含一个附加的链接器命令文件：

```
INCLUDE 'cmdfile'
```

其中：'cmdfile'是将要包含（INCLUDE）的链接器命令文件名称。

例如：

包含一个名为 mylink.lkr 的链接器命令文件，则要在链接器命令文件中加入以下行：

```
INCLUDE mylink.lkr
```

5．5 存储器区段定义

链接器命令（linker script）用来描述PICmicro系列MCU的存储器结构。这样链接器才可以把代码放置到可用的ROM当中，把变量放置到可用的RAM当中。凡是标志有PROTECTED（已保护）的区域将不被程序或数据的通用部分（general allocation）所使用。这些区域被做了标记，另有特殊用途。除非在你的源代码中使用指示符#pragma code/udata（MPLAB-C17/C18）或code/udata（MPASM）来明确地标记了代码和数据的属性后，才能使用这些区域。

5．5．1 定义ROM存储器区域

指示符 CODEPAGE是为说明程序代码，已初始化数据值，常量数据值和PIC17CXX系列的外部存储器设计的。说明格式如下：

```
CODEPAGE NAME=memName START=addr END=addr [PROTECTED] [FILL=fillvalue]
```

其中：

memName是任何用来确定CODEPAGE的ASCII字符串。

addr是用来指定一个地址的10进制或16进制的数字。

fillValue是用来填充存储器块里未使用区域的数值。假如它是一个10进制符号，则假设为一个16-BIT的量。假如它是一个16进制符号（比如0x2346），则是一个可以被16整除的任何长度的值。

可选关键字“PROTECTED”用来指定一个存储器区域，该区域只有明确地请求后才能被程序代码使用。

5．5．2　定义ROM存储器区域-范例

图5-1显示PIC17C42A微控制器的程序存储器布局情况。根据该图，范例5-1显示了用CODEPAGE（代码页）申明存储器的情况。

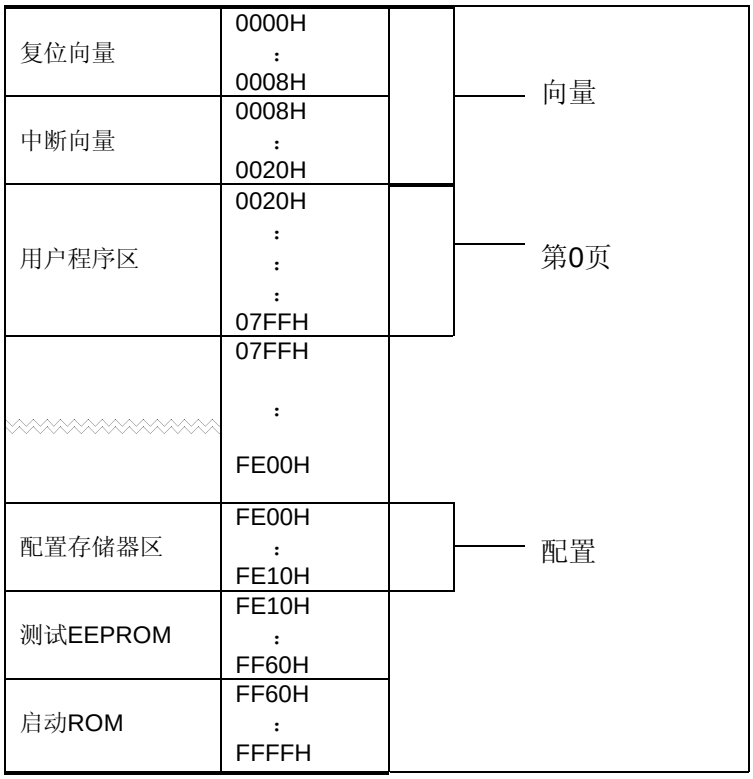


图5-1：PIC17C42A程序存储器映射（微控制器模式）

例5-1：PIC17C42A的ROM存储器申明

```
CODEPAGE NAME=vectors START=0x0     END=0x27 PROTECTED
CODEPAGE NAME=page0    START=0x28    END=0x7FF
CODEPAGE NAME=.config START=0xFE00 END=0xFE0F PROTECTED
```

5．5．3　定义RAM存储器区域

指示符：DATABANK，SHAREBANK 和ACCESSBANK说明内部RAM中的变量数据。其使用格式如下：

. **分段的寄存器**（Banked registers）：

```
DATABANK NAME=memName START=addr END=addr [PROTECTED]
```

。**未分段寄存器**（Unbanked registers）：

```
SHAREBANK NAME=memName START=addr END=addr [PROTECTED]
```

。**读写寄存器**（Access registers）-只针对PIC18CXX：

```
ACCESSBANK NAME=memName START=addr END=addr [PROTECTED]
```

其中：

memName是任何用来确定一个RAM区域的ASCII字符串。

Addr是一个是用来指定一个地址的10进制或16进制数。

可选关键字PROTECTED用来指定一个存储器区域，该区域只有明确地请求后才能被程序代码使用。

5．5．4　定义RAM存储器区域-范例

图5-2所显示的是RAM存储器布局图。正如例5-2和例5-3所示：链接器描述文件（linker script file）中将会出现关于DATABANK 和 SHAREBANK 的条目。

地址	堆0	堆1	堆2	堆3
00h	INDF0			
01h	FSR0			
:	:			
0Eh	TBLPTRH			
0Fh	BSR			
10h	PORTA	DDRC	TMR1	PW1DCL
11h	DDRB	PORTC	TMR2	PW2DCL
:	:			
16h	TXREG	PIR	PR3LCA1L	TCON1
17h	SPBRG	PIE	PR3HCA1H	TCON2
18h	PRODL			
19h	PRODH			
1Ah : 1Fh	通用RAM（未分堆）			
20h : FFh	通用RAM	通用RAM	未使用	

图5-2：PIC17C42A寄存器文件映射

例5-2：针对PIC17C42A的RAM存储器申明-分堆存储器（Banked Memory）

```
DATABANK NAME=sfr0 START=0x10 END=0x17 PROTECTED //Special Function Registers in Bank0
DATABANK NAME=sfr1 START=0x110 END=0x117 PROTECTED //Special Function Registers in Bank1
DATABANK NAME=sfr2 START=0x210 END=0x217 PROTECTED //Special Function Registers in Bank2
DATABANK NAME=sfr3 START=0x310 END=0x317 PROTECTED //Special Function Registers in Bank3
DATABANK NAME=gpr0 START=0x20 END=0xFF //General Purpose RAM in Bank0
DATABANK NAME=gpr1 START=0x120 END=0x1FF //General Purpose RAM in Bank1
```

例5-3：针对PIC17C42A的RAM存储器申明-未分堆存储器（Unbanked Memory

```
SHAREBANK NAME=sfrnobnk START=0x0 END=0xF PROTECTED //INDF0 thru BSR - available in all banks
SHAREBANK NAME=sfrnobnk START=0x100 END=0x10F PROTECTED //INDF0 thru BSR - available in all banks
SHAREBANK NAME=sfrnobnk START=0x200 END=0x20F PROTECTED //INDF0 thru BSR - available in all banks
SHAREBANK NAME=sfrnobnk START=0x300 END=0x30F PROTECTED //INDF0 thru BSR - available in all banks
SHAREBANK NAME=sfrprod START=0x18 END=0x19 PROTECTED //PRODL, PRODH - available in all banks
SHAREBANK NAME=sfrprod START=0x118 END=0x119 PROTECTED //PRODL, PRODH - available in all banks
SHAREBANK NAME=sfrprod START=0x218 END=0x219 PROTECTED //PRODL, PRODH - available in all banks
SHAREBANK NAME=sfrprod START=0x318 END=0x319 PROTECTED //PRODL, PRODH - available in all banks
SHAREBANK NAME=gpr2 START=0x1A END=0x1F PROTECTED //1A thru 1F - available in all banks
SHAREBANK NAME=gpr2 START=0x11A END=0x11F PROTECTED //1A thru 1F - available in all banks
SHAREBANK NAME=gpr2 START=0x21A END=0x21F PROTECTED //1A thru 1F - available in all banks
SHAREBANK NAME=gpr2 START=0x31A END=0x31F PROTECTED //1A thru 1F - available in all banks
```

5．6 逻辑区段定义

逻辑区段用来指定哪一个已定义的存储器区域可以被某一段源代码使用。使用这个逻辑区段，先在链接器描述文件（linker script file）里用指示符 SECTION 定义该区段，然后在源文件里用开发语言的相应方式假设该名称（例如：#pragma section – 针对 MPLAB-C17/C18）。

通过指定一个名称，区段指示符可以定义一个区段。ROM 中的程序存储器块或 RAM 中的数据存储器块包含有区段：

```
SECTION NAME='secName' { ROM='memName' | RAM='memName' }
```

其中：

secName 是 ASCII 字符串，用来命名一个区段（SECTION）。

memName是一个预先定义的ACCESSBANK，SHAREBANK，DATABANK 或者 CODEPAGE。

ROM 的属性取决于预先用 CODEPAGE 指示符设置好的程序存储器。RAM 的属性取决于预先用 ACCESSBANK，DATABANK 或者 SHAREBANK 指示符设置好的数据存储器。

例如：

指定一个名为 filter_coeffs 的区段，将其调入预先定义好的名为 constants 的程序存储器逻辑块中，则在链接器命令文件（linker command file）中应该加入以下行：

例 5-4：定义逻辑区段

```
SECTION NAME=filter_coeffs ROM=constants
```

要想将 MPASM 源代码放置到 filter_coeffs 区段，在相应的源代码之前放置下列行：

例 5-5：使用逻辑区段

```
filter_coeffs CODE
```

5．7 堆栈定义

MPLAB-C17/C18 要求设置软件堆栈。以下所诉表明如何指定堆栈的大小和可选的 DATABANK（堆栈位于其中）。

```
STACK SIZE='allocSize' [RAM='memName']
```

其中：

allocSize 是堆栈的大小（字节），**memName** 是预先用 ACCESSBANK，DATABANK 或 SHAREBANK 定义的存储器名。

例如：

在 RAM 中将堆栈大小设置为'0x20'（RAM 预先用 gpr0 定义），则在链接器命令文件中要加入以下行:

例 5-6：定义堆栈

```
STACK SIZE=0x20 RAM=gpr0
```

5．8 关于链接器描述的注意事项（linker script caveats）：

以下是一些链接器描述的注意事项：

。在使用 MPLINK 之前，有可能修改它包含的链接器描述文件（linker script files）。

。当 MPLAB-C17/C18 和 MPLINK 一起使用时，可能修改堆栈的大小。

。当代码中用到了GOTO或CALL指令，而又没有用PAGESEL指示符，则需要截断存储器页。

。在MPASM中不能在一个文件中向前和向后切换区段。例如，不能象这样做：

```
CODE MY_ROM
       :
       (instructions)
       :
UDATA MY_VARS
       :
       (variables)
       :
CODE MY_ROM
       :
```

第 6 章 链接器处理过程

6.1 介绍

本章介绍链接器如何处理链接工作。

6．2 重点

本章内容包括：

- 链接过程概述
- 链接器分配算法
- 重定位范例
- 初始化代码

6．3 链接过程概述

链接器将多个输入目标文件链接成为一个可执行输出文件。输入目标模块可能包含可重定位或绝对代码，以及链接器将在目标存储器中定位的数据。在链接控制文件中会对目标系统的存储器结构进行描述。这个链接控制文件为说明目标系统存储器块以及这些被说明的存储器的映射区段提供了一种灵活的机制。目标系统存储器块是用来定位一个区段的，假若链接器找不到该文件，则会生成一个错误文件。链接器将具有相似名称的输入区段组合为一个输出文件区段。链接器的定位原理介绍如下。

链接器将所有输入文件模块中的所有区段定位后，将开始对符号进行处理。每个输入区段里的符号地址取决于该区段的开始地址。链接器将根据这些符号在它们分配区段中的最终位置来调整符号地址。

链接器完成对所有输入模块中的符号重定位后，将去处理外部符号。链接器试图将所有外部符号与相应的符号定义进行匹配。假如一个外部符号没有找到相应的符号定义，链接器会转向输入库文件中去寻找，假若仍然没有找到该符号定义，链接器会生成错误报告。

如果外部符号被成功定位后，链接器接下来会处理每个区段里的原始数据。每个区段里都有一个包含可重定位路径的列表，该列表把一个区段里的原始数据定位与可重定位符号联系起来，可重定位符号的地址就会被分配到原始数据块里。重定位符号以及区段分配的处理过程接下来会详细说明。

链接器处理完所有可重定位信息后，会生成可执行输出模块。

6．4 链接器定位原理

链接器将区段（SECTION）定位以便取得对它们的最大控制范围。这些区段是目标系统存储器里的数据和代码。链接器处理四种定位操作。区段可以是绝对代码，也可以是可重定位（相对）代码。它们既可以通过链接器控制文件分配目标存储器块，也可以暂时不分配。所以存在以下定位类型：

1．绝对分配
2．绝对不分配
3．可重定位分配
4．可重定位不分配

链接器先对绝对代码区段进行定位，然后是可重定位代码区段，接下来是可重定位未定义区段。

6．4．1　绝对分配

在链接器控制文件里绝对代码区间可以被分配到目标系统存储器块里，可是由于绝对代码区段的地址是固定的，链接器只能去确认：是否在目标系统存储器块里有一个给绝对区段的已分配的块；目标存储器块是否足够大；以及该绝对区段会不会覆盖其他的区段。假如没有目标存储器块分配给绝对区段，链接器将试图为它找到一个。假如一个区段不能定位，就会有一个错误报告产生。因为绝对区段只能放到固定的地址里，分配和未分配区段的处理没有特定的顺序。

6．4．2　可重定位分配

所有绝对区段被分配后，链接器去分配可重定位分配区段。对于可重定位分配区段，链接器检查已经分配的目标存储器块，看是否有可用的空间，若没有，则会产生错误报告。可重定位分配区段的位置按它们在链接器命令文件中指定的位置而定。

所有可重定位分配区段被分配后，链接器去分配可重定位未分配区段。链接器以最大的可重定位未分配区段开始一直到最小的可重定位未分配区段进行处理。对于每一个分配，链接器在目标存储器中选择最小的可用空间来安排该区段。链接器由最大的区段开始，选择最小的存储器驻留空间，这增加了将所有可重定位未分配区段全部分配出去的机会。

堆栈不是区段，但和区段一起分配。链接器命令文件中可能已经或者还没有给堆栈在目标存储器中指定位置。如果堆栈在目标存储器中被分配位置，它在可重定位分配区段被定位之前被分配。如果堆栈未指定，则会在可重定位分配区段被定义之后，可重定位未分配区段被定义之前被分配。

6．5　重定位范例

以下例子显示链接器如何定位区段。假如以下是一个文件中的源代码段。

```
/* File: ref.c */
char var1;           /* Line 1 */
void setVar1(void) { /* Line 2 */
       var1 = 0xFF; /* Line 3 */
}
```

以上程序被编译为如下汇编语句(注意:本例程故意省略了由 MPLAB-C17/C18 生成的关于函数的进入和退出的语句代码)：

```
0x0000 MOVLW 0xFF
0x0001 MOVLR ?? ;  需要填入var1的段（bank）
0x0002 MOVWF ?? ;  需要填入 var1 的偏移（offset）
```

当编译器处理源程序的第一行时，将按如下格式为标识符 var1 产生一个符号表入口：

```
Symbol[index] => name=var1, value=0, section=.data,
class=extern
```

当编译器处理源程序的第三行时，由于标识符 var1 的最终地址直到链接前是未知的，故编译器将在代码区段为标识符 var1 产生 2 个可重定位入口。其格式如下：

```
Reloc[index] => address=0x0001 symbol=var1 type=bank
Reloc[index] => address=0x0002 symbol=var1 type=offset
```

一旦链接器将每一个区段都放到了目标存储器中，最终的地址就知道了。当所有的标识符都被分配了最终地址后，链接器就会用可重定位入口来填补所有的参考量。上面例子中，被更新的符号现在可能定位是 0x125:

```
Symbol[index] => name=var1, value=0x125, section=.data,
class=extern
```

假如上面的代码区段被放在地址 0x50 开始的地方，被更新的可重定位入口现在将开始于 0x51:

```
Reloc[index] => address=0x0051 symbol=var1 type=bank
Reloc[index] => address=0x0052 symbol=var1 type=offset
```

链接器将一步一步的处理可重定位入口并填补他们相应的区段。这样一来，最终得到以上范例的等价汇编语句：

```
0x0050 MOVLW 0xFF
0x0051 MOVLR 0x1 ; Patched with var1’s bank
0x0052 MOVWF 0x25 ; Patched with var1’s offset
```

6．6 初始化数据

MPLINK 靠初始化数据对输入区段进行特殊处理。初始化数据区段包含给变量赋予初始值（初始化）和定义常量。因为初始化数据区段里的变量和常量驻留在 RAM 里，他们的数据应该存储在非易失性的程序存储器（ROM）里。对于每一个初始化数据区段，链接器在程序存储器里产生一个区段。数据将在程序启动时被初始化程序（由 MPLAB C17/18 或者 MPASM 编写）转移到相应的 RAM 里去。

由链接器生成的初始化程序区段名和初始化数据区段相同，都有一个“_i”结尾。如：如果一个输入目标模块包括一个名为“.idata_main.o”的初始化数据区段，链接器将在程序存储区产生一个名为“.idata_main.o_i”的区段，该区段包含数据。

链接器除了生成初始化程序区段外，还在程序存储区产生一个名为“.cinit”的区段。“.cinit”区段里有每个初始化数据区段的入口表。每个入口都是三个字节，它指出初始化程序区段在程序存储器中的开始地址；初始化数据区段在 RAM 中的开始地址，以及在初始化数据区段里有多少字节。初始化引导程序将操作该表并将它从 ROM 中引导到 RAM 里。

第 7 章 应用范例 1

7.1 重点

本章讲述如下内容：

- 如何在不同的存储器区域放置程序
- 如何在 ROM 中放置数据
- 如何在 C 环境下设置配置位

7.2 概述

本章针对 PIC17C44 微控制器，工作于扩展控制器模式，用汇编写成的中断处理程序放在 0x8000 开始的位置。汇编代码指示符 INTHAND CODE 把代码放在 INTHAND 区段之后。链接器描述语言将 INTHAND 区段和从 0x8000 开始的 CODE 区域映射起来。

在程序存储器的 0x1000 还有 ELEMENT DATA 表，它和中断处理器处于相同的代码页。

注意：PIC17CXXX 系列 MCU 的一个代码页长 8K 字。例如：
0000 –1FFF, 2000 – 3FFF, 等。

在 C 语言环境下，数据表定义用#pragma romdata 指示语句，把表格放在一个称为 DATTBL 的区段里。链接器描述语句将 DATTBL 区段和从 0x8000 开始的 CODE 区域映射起来。

C 语言的主函数放在默认的 CODE 区段，因为没有确切地指定区段指示符。

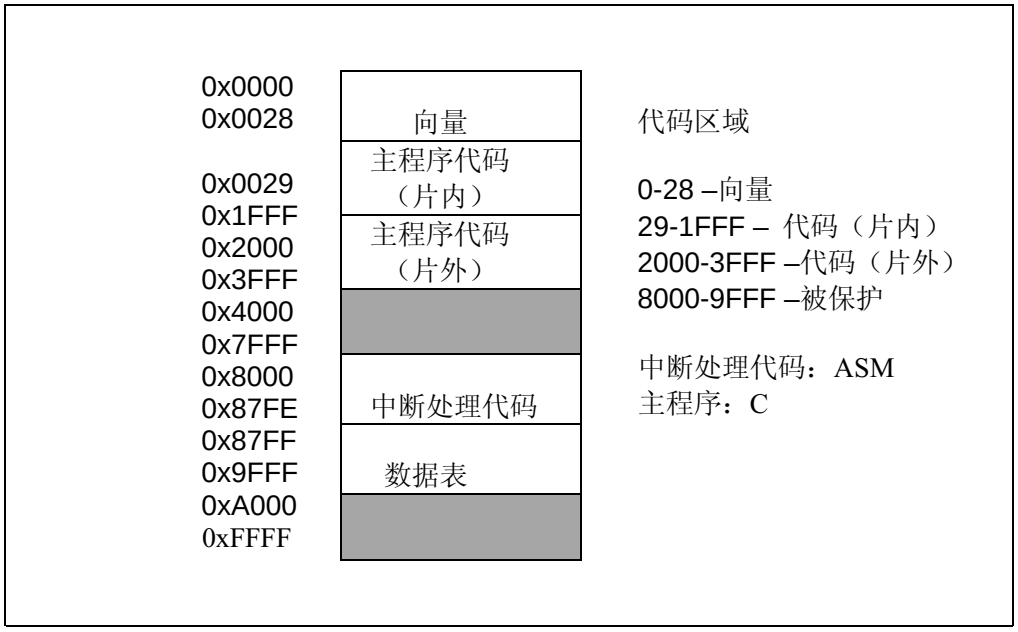


图 7.1：程序存储器映射图 - 应用范例 1

7.3 建立应用

本范例的源文件可由包含的批处理文件或 MPLAB 编译。

批处理文件 app1.bat 可编译本应用程序。该文件认为 MCC17, MPASM 和 MPLINK 可以在你设定的路径（PATH）里找到。同样，MCC_INCLUDE 环境变量应设置指向 MCC\H。你可以在批处理文件 AUTOEXEC.BAT 里设置这些参数，在 DOS 命令行下用 TYPE SET.MPLINK 命令查看，设置库文件路径指向

C:\MCC\LIB。如果 MPLAB-C17/C18 安装在其他的目录里，改变批处理文件中关于 MPLINK 的命令行。

在 MPLAB 项目（PROJECT）里建立此应用：

1. 给新项目取名 APP1
   - APP1.HEX-变为项目的顶端节点
2. 设置 MPLINK 作为 APP1.HEX 的语言工具
3. 在项目中添加这些节点，设置语言工具为：
   - eeprom.asm – 语言工具：MPASMWIN
   - eeprom1.c– 语言工具： MPLAB-C17/C18
   - eeprom.lkr
   - p17C44.o – 该文件在目录： mcc\lib
   - C0L17.O – 该文件在目录： mcc\lib
   - IDATA17.O – 该文件在目录： mcc\lib

4. 编辑APP1的MPLAB项目。因为其中一个文件中包含头文件P17C44.H。
   - Include path = c:\mcc\h

注意：当链接时，你可能遇到以下信息：
"Warning – Could not open source file '<filename>'.
This file will not be present in the list file."
（警告：无法打开源文件"文件名"。）
（这个文件将不会在列表文件里出现）
这是由于使用了预编译的库文件，而这些库的源文件不在默认目录里。

7.4 源代码

EEPROM.ASM –含有一个列表指示语句；一个带 INTHAND 区段的代码指示语句；一个放置中断处理代码的地方。

EEPROM1.C –在程序数据区段 DATTBL 含有一个 0x1000 数据阵，在默认代码区段含有一个主函数。

EEPROM.LKR –包含对 PIC17C44 的描述，0x8000 到 0x9FFF 区域为保护代码。INTHAND 和 DATTBL 被映射到保护代码区域。

7．4．1 eeprom.asm

```
; EEPROM.ASM

LIST p=17c44

#include "p17c44.inc"


INTHAND CODE

; 中断处理代码放在这里
END
```

7．4．2 eeprom1.c

```
/* EEPROM1.C */

#include "p17c44.h"

#define DATA_SIZE 0x1000

#pragma romdata DATTBL // put following romdata into section DATTBL
unsigned rom data[DATA_SIZE];

#pragma romdata // set back to default romdata section
/* 默认代码区段里的主函数代码 */
void main( void )
{
   while( 1 )
   {

   } /* end while */

} /* end main */
```

7．4．3 eeprom.lkr

```
 // 文件名: EEPROM.LKR PIC17C44

LIBPATH .
CODEPAGE NAME=vectors   START=0x0     END=0x27    PROTECTED
CODEPAGE NAME=page     START=0x28    END=0x1FFF
CODEPAGE NAME=eeprom    START=0x8000  END=0x9FFF   PROTECTED

SHAREBANK NAME=gprshare START=0x1A   END=0x1F
SHAREBANK NAME=gprshare START=0x11A END=0x11F
SHAREBANK NAME=gprshare START=0x21A END=0x21F
SHAREBANK NAME=gprshare START=0x31A END=0x31F

DATABANK NAME=gpr0 START=0x20   END=0xFF
DATABANK NAME=gpr1 START=0x120  END=0x1FF

SHAREBANK NAME=sfrnobnk START=0x0    END=0xF   PROTECTED
SHAREBANK NAME=sfrnobnk START=0x100 END=0x10F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x200 END=0x20F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x300 END=0x30F PROTECTED

SHAREBANK NAME=sfrprod START=0x18  END=0x19   PROTECTED
SHAREBANK NAME=sfrprod START=0x118 END=0x119 PROTECTED
SHAREBANK NAME=sfrprod START=0x218 END=0x219 PROTECTED
SHAREBANK NAME=sfrprod START=0x318 END=0x319 PROTECTED

DATABANK NAME=sfr0 START=0x10  END=0x17   PROTECTED
DATABANK NAME=sfr1 START=0x110 END=0x117 PROTECTED
DATABANK NAME=sfr2 START=0x210 END=0x217 PROTECTED
DATABANK NAME=sfr3 START=0x310 END=0x317 PROTECTED

SECTION NAME=STARTUP ROM=vectors   // 复位和中断向量
SECTION NAME=PROG ROM=page         // 主函数代码区域
SECTION NAME=INTHAND ROM=eeprom    // 中断处理代码
SECTION NAME=DATTBL ROM=eeprom     // 数据表格

STACK SIZE=0x3F RAM=gpr0
```

第8章　应用范例 2

8．1　重点

本章内容包括：

- 如何为引导器分配存储器
- 如何编译将要调入外部 RAM 中运行的代码

8．2　概述

很多应用需要现场修改支持软件（FIRMWARE）。PIC17CXX 系列 MPU 支持扩展微控制器工作模式（extended microcontroller mode）。本范例中用到了 PIC17C44 微控制器。

一个简单的启动器引导器（boot loader）放置在芯片内部程序存储器里，可以支持一个或一个以上的被支持软件（firmware）调用的功能。主要的支持软件放置在外部程序存储器里面。程序存储器数据表格作为一个已分配区段（assigned section）位于适当的存储器区。

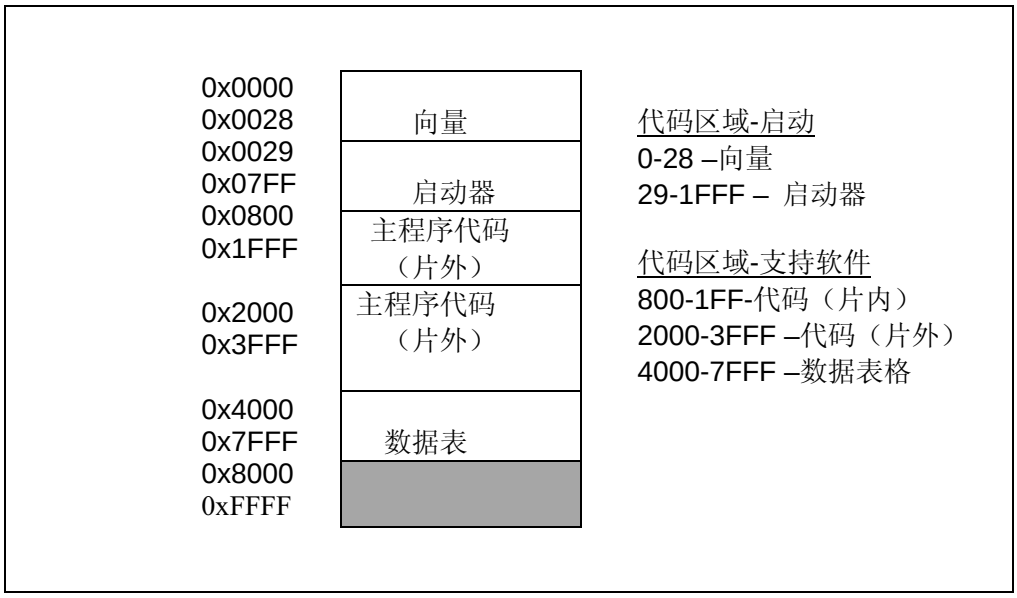


图 8-1：程序存储器映像 - 应用范例 2

8．3　建立应用程序

本应用范例的源文件可以通过批处理文件或 MPLAB 进行编译。

用批处理文件 APP2.BAT 编译该应用。本文件假设在用 PATH 设置的路径里可以找到 MCC17，MPASM 和 MPLINK。而且 MCC_INCLUDE 环境变量应该设置为指向 MCC\H。在批处理文件 AUTOEXEC.BAT 里可以看到这些，DOS 方式下用“SET”命令可以检查这些设置。 MPLINK 设置库的路径为 C:\MCC\LIB。假如 MPLAB-C17/C18 安装在别的地方，则在批处理文件里将关于 MPLINK 行的路径改为指向 MPLAB-C17/C18 安装目录。

本应用范例包含 2 个单独的目标文件，一个是将要烧入 PIC17C44 的文件，另一个是将要下载到外部程序存储器里执行的支持软件。

为了在 MPLAB 的项目里建立该应用，两个目标文件需要被生成（create）和创建（build）。设置第一个目标文件来建立启动引导器（boot loader）。

1. 取一个新项目名称 APP2BOOT
   - APP2BOOT.HEX – 成为项目的顶部节点
2. 设置 MPLINK 作为 APP1.HEX 的开发工具
3. 把这些节点加入到项目里面并设置语言工具
   - boot.c – 设置 MPLAB-C17/C18 作为语言工具
   - loader.lkr
   - C0S17.O – 该文件位于 mcc\lib 中
4. 编辑 APP2BOOT 的 MPLAB 项目
   - 库路径= c:\mcc\lib
5. 创建项目（Build project），然后存项目并退出

建立第二个项目来编译支持软件：

1. 取一个新项目名称 APP2
   - APP2.HEX – 成为项目的顶部节点
2. 设置 MPLINK 作为 APP1.HEX 的开发工具
3. 把这些节点加入到项目里面并设置语言工具
   - fwentry.asm – 设置 MPASMWIN 作为语言工具
   - firmware.c – 设置 MPLAB-C17/C18 作为语言工具
   - fwtables.c – 设置 MPLAB-C17/C18 作为语言工具
   - firmware.lkr
4. 编辑 APP1 的 MPLAB 项目：
   - 库路径= c:\mcc\lib（或者你自己给定的 mcc\lib 路径。
5. 创建项目（Build project），然后保存项目并退出。

注意：当链接时，你可能遇到以下信息：
"Warning – Could not open source file '<filename>'.
This file will not be present in the list file."
（警告：无法打开源文件"文件名"。）
（这个文件将不会在列表文件里出现）
这是由于使用了预编译的库文件，而这些库的源文件不在默认目录里。

8．4 源代码 - 启动引导器（boot loader）

boot.c –启动引导器把支持软件（firmware）拷贝到外部程序存储器中并跳转到一个预先定义好的地址上，该地址即是支持软件的入口。支持软件还在一个固定的地址提供随时支持的功能。

loader.lkr –启动引导器链接描述（boot loader linker script）只定义片内程序存储器的区域。

8．4．1 boot.c

```
#include <p17c44.h>

/* 该支持软件的入口地址定义为 0x4000 */
#define FW_ENTRY 0x4000

#define LGOTO(x) {_asm movlw high (x) movwf PCLATH movlw (x) movwf PCL _endasm}

void load_firmware( void );

void load_firmware( void )
{
    /*执行相应操作，将支持软件调入外部存储器里*/
}

void main( void )
{
    load_firmware();
    LGOTO( FW_ENTRY );
}

/* 我们将提供一个函数给支持软件，以便调用它向LCD显示器输出一个字符
 *
 * 我们需要该函位于一个确切的地址，以便在支持软件中定义入口地址
 */

#pragma code out_lcd=0x1000

void out_lcd( unsigned char );

void out_lcd( unsigned char ch )
{
   /* 执行相应操作*/
}

#pragma romdata CONFIG
// 以下配置位定义来自头文件P17C44.INC:
#define _PMC_MODE 0x7FAF
#define _XMC_MODE 0xFFBF
#define _MC_MODE 0xFFEF
#define _MP_MODE 0xFFFF

#define _WDT_NORM 0xFFF3
#define _WDT_OFF 0xFFF3
#define _WDT_64 0xFFF7
#define _WDT_256 0xFFF
#define _WDT_1 0xFFFF

#define _LF_OSC 0xFFFC
#define _RC_OSC 0xFFFD
#define _XT_OSC 0xFFFE
#define _EC_OSC 0xFFFF

rom const unsigned int my_config = _MC_MODE & _WDT_OFF & _XT_OSC ;
```

8．4．2 loader.lkr

```
LIBPATH . // 给库搜索路径增加一个目录
LIBPATH c:\work\mcc\cc\install\lib
FILES p17c44.o

CODEPAGE NAME=reset_vector   START=0x0000    END=0x0027          // 复位向量
CODEPAGE NAME=page0          START=0x0028    END=0x1FFF          // 片内存储器
CODEPAGE NAME=page1          START=0x2000    END=0x3FFF          // 片内存储器
CODEPAGE NAME=config         START=0xFE00    END=0xFE0F          PROTECTED

SHAREBANK NAME=sharedsfr     START=0x00      END=0x0f            PROTECTED
DATABANK NAME=sfr0           START=0x10      END=0x17            PROTECTED
DATABANK NAME=sfr1           START=0x110     END=0x117           PROTECTED
DATABANK NAME=sfr2           START=0x210     END=0x217           PROTECTED
DATABANK NAME=sfr3           START=0x310     END=0x317           PROTECTED
DATABANK NAME=sfr4           START=0x410     END=0x417           PROTECTED
DATABANK NAME=sfr5           START=0x510     END=0x517           PROTECTED
DATABANK NAME=sfr6           START=0x610     END=0x617           PROTECTED
DATABANK NAME=sfr7 S        TART=0x710      END=0x717           PROTECTED
SHAREBANK NAME=registers     START=0x18      END=0x1f            PROTECTED
DATABANK NAME=gpr0           START=0x20      END=0xEF        // 通用 RAM 第0块
DATABANK NAME=gpr1           START=0x120     END=0x1FB       // 通用 RAM 第1块

DATABANK NAME=stack START=0xF0 END=0xFF PROTECTED // 堆栈RAM

SECTION NAME=CONFIG ROM=config// 配置位（configuration bits）位置

STACK SIZE=0x0F RAM=stack
```

8．5 源代码 - 支持软件（firmware）

firmware.c –入口函数（entry function）位于一个绝对区段内，其地址是预先确定的支持软件入口地址。支持软件的其他部分由链接器定位。

fwtables.c–程序存储器数据表格放在一个由链接器定位的数据区段里。

fwentry.h –这些支持函数被设计得和其他的原型 C 函数一样，可供调用。

fwentry.asm –对于支持软件，链接器需要在每个函数的入口地址处有一个符号。

firmware.lkr –支持软件链接器描述（firmware linker script）从程序存储器数据表和片内支持函数定义保护区域（protected regions）。其他外部存储器被定义为“可以使用”。

8．4．1 firmware.c

```
#include <p17c44.h>
#include "fwentry.h"

void fw_entry_func( void );
void signon( void );

#pragma code fw_entry = 0x4000
void fw_entry_func( void )
   {
       /* signon message */
    signon();
      /* 执行相应的操作... */
    while(1);
}
#pragma code // we don’t care where the rest of stuff goes

void signon( void )
{
    static const rom char msg[] = "FW v1.00";
    rom char *p;
    for( p=msg ; *p != 0 ; p++ )
       out_lcd( *p );
}
```

8．4．2 fwtables.c

```
#include <p17c44.h>

#pragma romdata fw_tables

rom unsigned datatable[] = {
      0x0000,
      0x0001,
      0x0002,
      0x0003,
      0x0004,
      0x0005,
      0x0006,
      0x0007,
      0x0008,
};
```

8．4．3 fwentry.h

```
#ifndef FWENTRY_H
#define FWENTRY_H

void out_lcd( unsigned char ch );

#endif
```

8．4．4 fwentry.asm

```
; 为ROM里的系统例程以相应的地址提供给符号，这些系统例程可被支持软件调用。
; 这些例程的实际代码在ROM里的启动引导器里，这些符号表示它们是这些地址位置的占
；有者（place-holders）
LIST P=17c44
GLOBAL out_lcd
OUTLCDENTRY CODE 0x1000
out_lcd
END
```

8．4．5 firmware.lk

```
LIBPATH .      // 为库搜索路径添加一个目录
LIBPATH c:\work\mcc\cc\install\lib
FILES p17c44.o

//CODEPAGE NAME=reset_vector START=0x0000 END=0x0027 // 复位向量
//
//
// 我们并不希望任何片外支持程序（off-chip firmware）被放置在片内任何地址空间里
// 我们把芯片某些空间设置为“已保护(PROTECTED)”属性，因为我们把一些入口点地址放置
// 到启动引导器(bootloader)里。支持程序(firmware)从这里调用应用程序。这些地址空间
// 并没有实际的代码。但我们需要在这里安排一些空的区段(sections)以便于重定位。
//
CODEPAGE NAME=entrypoints START=0x1000  END=0x3fff  PROTECTED

CODEPAGE NAME=firmware    START=0x4000  END=0x5FFF
CODEPAGE NAME=tables      START=0x6000  END=0x7FFF  PROTECTED

SHAREBANK NAME=sharedsfr  START=0x00    END=0x0f    PROTECTED
DATABANK NAME=sfr0        START=0x10    END=0x17    PROTECTED
DATABANK NAME=sfr1        START=0x110   END=0x117   PROTECTED
DATABANK NAME=sfr2        START=0x210   END=0x217   PROTECTED
DATABANK NAME=sfr3        START=0x310   END=0x317   PROTECTED
DATABANK NAME=sfr4        START=0x410   END=0x417   PROTECTED
DATABANK NAME=sfr5        START=0x510   END=0x517   PROTECTED
DATABANK NAME=sfr6        START=0x610   END=0x617   PROTECTED
DATABANK NAME=sfr7        START=0x710   END=0x717   PROTECTED
SHAREBANK NAME=registers  START=0x18    END=0x1f    PROTECTED
DATABANK NAME=gpr0        START=0x20    END=0xEF    //通用RAM第0堆
DATABANK NAME=gpr1        START=0x120   END=0x1FF   //通用RAM第1堆


DATABANK NAME=stack START=0xF0 END=0xFF PROTECTED // 堆栈RAM

STACK SIZE=0x0F RAM=stack
```

第9章　应用范例 3

9.1　重点

本章内容包括：

- 如何操作存储器映射的外设
- 如何建立新的区段

9.2　概述

本范例有两个外部设备：位于 0x8000 的 DAC 和位于 0x9000 的 IRD。在 0x8000 和 0x9000 两个位置，代码都申明多个变量。当这些变量中的某一个被读或写时，变量所代表位置的设备就被读或写。`#pragma romdata` 指示符在指定存储器区域建立一个绝对区段。

链接器描述文件（linker script file）为每个设备和程序存储器申明代码页。每个设备的代码页都用“PROTECTED”标明，防止链接器将未分配可重定位区段（unassigned relocatable section）放入存储器块中。链接器描述文件还有为代码 0，代码 1 和代码页申明的区段。区段名和指示符`#pragma code` 一起使用，这样链接器将代码放在该区段的存储器位置`#pragma` 后面。

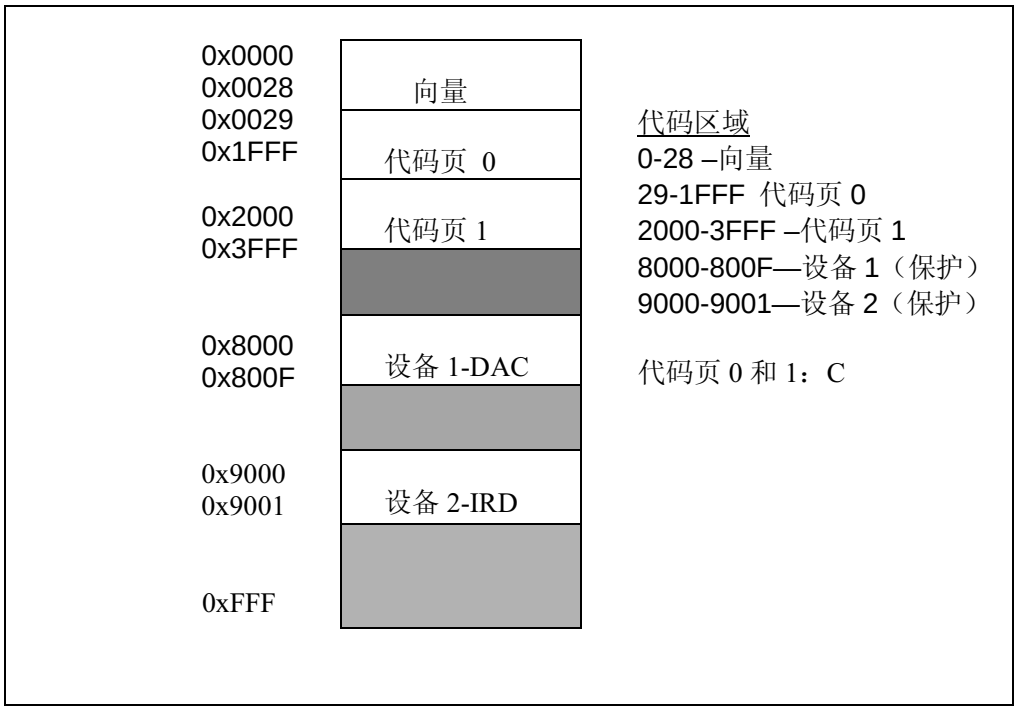


图 9-1：程序存储器映射

9.3 创建应用程序(Building the Application)

本应用范例的源文件可以通过批处理文件或 MPLAB 进行编译。

批处理文件 app3.bat 被用来编译该应用文件。本文件假设在用 PATH 设置的路径里可以找到 MCC17，MPASM 和 MPLINK。而且 MCC_INCLUDE 环境变量应该设置为指向 MCC\H。在批处理文件 AUTOEXEC.BAT 里可以看到这些，DOS 方式下用“SET”命令可以检查这些设置。 MPLINK 设置库的路径为 C:\MCC\LIB。假如 MPLAB-C17/C18 安装在别的地方，则在批处理文件里将关于 MPLINK 行的路径改为指向 MPLAB-C17/C18 安装目录。

在 MPLAB 的项目里创建该应用。

1. 取一个新项目名称 APP3
   - APP3.HEX – 成为项目的顶部节点
2. 设置 MPLINK 作为 APP1.HEX 的开发工具
3. 把这些节点加入到项目里面并设置语言工具
   - memmapio.c – 设置 MPLAB-C17/C18 作为语言工具
   - memmapio.lkr
   - p17C756.o –该文件位于 mcc\lib 中
4. 编辑 APP1 的 MPLAB 项目
   - 库路径= c:\mcc\lib

注意：当链接时，你可能遇到以下信息：
“Warning – Could not open source file '<filename>'.
This file will not be present in the list file.”
（警告：无法打开源文件“文件名”。）
（这个文件将不会在列表文件里出现）
这是由于使用了预编译的库文件，而这些库的源文件不在默认目录里。

注意：你还可能看到以下信息：
“Warning[2003]...\MEMMAPIO.C 25 :
'rom' and 'volatile' in same declaration”
这个信息很常见，它提醒你：链接器将把该变量看作应用的需要。

9.4 源代码

9.4.1 memmapio.c

```
 #include <P17C756.H>

void main(void);

#pragma romdata dev1 = 0x8000
unsigned rom loc0;
unsigned rom loc1;
unsigned rom loc2;
unsigned rom loc3;
unsigned rom loc4;
unsigned rom loc5;
unsigned rom loc6;
unsigned rom loc7;
unsigned rom loc8;
unsigned rom loc9;
unsigned rom loca;
unsigned rom locb;
unsigned rom locc;
unsigned rom locd;
unsigned rom loce;
unsigned rom locf;

#pragma romdata dev2 = 0x9000
const volatile unsigned rom inp;
unsigned rom outp;

#pragma code code0
void main(void)
{
   unsigned int input;

    loc0 = 0;
   loc1 = 1;

    input = inp;
    outp = 0x7;
}
```

9.4.2 memmapio.lkr

```
// 文件名: memmapio.lkr
// 写给17C756, 17C756A, 17C766的链接器命令文件
// 12/16/97
LIBPATH .

CODEPAGE NAME=vectors   START=0x0    END=0x27   PROTECTED
CODEPAGE NAME=code0     START=0x28   END=0x1fff
CODEPAGE NAME=code1     START=0x2000 END=0x3FFF
CODEPAGE NAME=device1  START=0x8000  END=0x800F  PROTECTED
CODEPAGE NAME=device2  START=0x9000  END=0x9001  PROTECTED
CODEPAGE NAME=config   START=0xFE00  END=0xFE0F  PROTECTED

SHAREBANK NAME=sfrnobnk START=0x0   END=0xF   PROTECTED
SHAREBANK NAME=sfrnobnk START=0x100 END=0x10F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x200 END=0x20F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x300 END=0x30F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x400 END=0x40F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x500 END=0x50F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x600 END=0x60F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x700 END=0x70F PROTECTED

DATABANK NAME=sfr0 START=0x10   END=0x17  PROTECTED
DATABANK NAME=sfr1 START=0x110 END=0x117 PROTECTED
DATABANK NAME=sfr2 START=0x210 END=0x217 PROTECTED
DATABANK NAME=sfr3 START=0x310 END=0x317 PROTECTED
DATABANK NAME=sfr4 START=0x410 END=0x417 PROTECTED
DATABANK NAME=sfr5 START=0x510 END=0x517 PROTECTED
DATABANK NAME=sfr6 START=0x610 END=0x617 PROTECTED
DATABANK NAME=sfr7 START=0x710 END=0x717 PROTECTED

SHAREBANK NAME=sfrprod START=0x18   END=0x19  PROTECTED
SHAREBANK NAME=sfrprod START=0x118 END=0x119 PROTECTED
SHAREBANK NAME=sfrprod START=0x218 END=0x219 PROTECTED
SHAREBANK NAME=sfrprod START=0x318 END=0x319 PROTECTED
SHAREBANK NAME=sfrprod START=0x418 END=0x419 PROTECTED
SHAREBANK NAME=sfrprod START=0x518 END=0x519 PROTECTED
SHAREBANK NAME=sfrprod START=0x618 END=0x619 PROTECTED
SHAREBANK NAME=sfrprod START=0x718 END=0x719 PROTECTED

SHAREBANK NAME=registers START=0x1A  END=0x1F   PROTECTED
SHAREBANK NAME=registers START=0x11A END=0x11F PROTECTED
SHAREBANK NAME=registers START=0x21A END=0x21F PROTECTED
SHAREBANK NAME=registers START=0x31A END=0x31F PROTECTED
SHAREBANK NAME=registers START=0x41A END=0x41F PROTECTED
SHAREBANK NAME=registers START=0x51A END=0x51F PROTECTED
SHAREBANK NAME=registers START=0x61A END=0x61F PROTECTED
SHAREBANK NAME=registers START=0x71A END=0x71F PROTECTED

DATABANK NAME=gpr0 START=0x20   END=0xFF
DATABANK NAME=gpr1 START=0x120 END=0x1FF
DATABANK NAME=gpr2 START=0x220 END=0x2FF
DATABANK NAME=gpr3 START=0x320 END=0x3FF

SECTION NAME=code0 ROM=code0
SECTION NAME=code1 ROM=code1

//以下的堆栈指示符是使用 MPLAB-C17/C18时用到
STACK SIZE=0x20
```

第10章　应用范例 4

10．1　重点

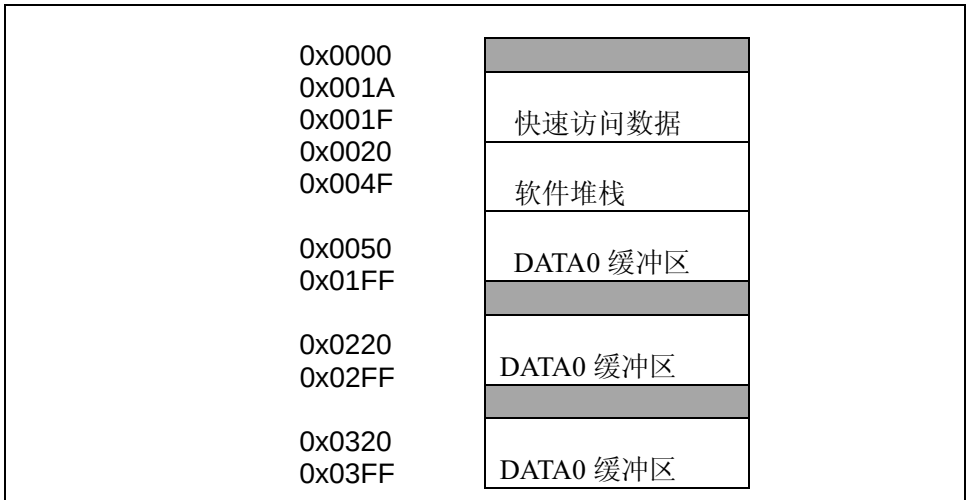
本章主要讲述：如何为程序应用手动设置 RAM 区。

10．2　概述

给出一个 PIC17C756 的小模式，微控制器模式的项目：morse.pjt。用这个项目从 USART1 接受莫尔斯代码，并将其输出到端口 B（PORTB）。它包含以下源文件：

main.c	主程序，输出莫尔斯代码到端口 B
morse.c	子程序，输出一个莫尔斯代码到端口 B
morse.h	Morse.c 的头文件-定义函数原型，重要常量（如：输入模式的表示）
usart.c	子程序，处理从 USART1 的输入
usart.h	usart.c 的头文件-定义函数原型
portb.c	子程序的头文件，处理从 PORTB 的输出
portb.h	portb.c 的头文件-定义函数原型
delayms.asm	子程序，负责延时（基于 4MHZ 时钟）
delayms.h	Delayms.asm 的头文件-定义函数原型
morse756.lkr	链接器描述文件，不包含区段，设置为小存储器模式

链接器描述文件和源代码按图 10.1.分配存储器：



数据区：
1A-1F-全局指针（被保护）-当前缓冲器读和写位置
20 - 4F –软件堆栈（大小应增加到 0x30 个字节）
50 - 1FF- morse.c 里的 data 0
220 - 2FF- delayms.asm 里的 data 1
320 - 3FF-所有其他 data 2
缓冲区访问代码：ASM
缓冲区数据：ASM
主程序代码：C
主数据声明：C
快速访问数据：C

图 10.1　寄存器文件映射 - 应用范例 4

10．3　建立应用程序

本范例的源文件可由包含的批处理文件或 MPLAB 编译。

批处理文件 app4.bat 可编译本应用程序。该文件认为 MCC17, MPASM 和 MPLINK 可以在你设定的路径（PATH）里找到。同样，MCC_INCLUDE 环境变量应设置指向 MCC\H。你可以在批处理文件 AUTOEXEC.BAT 里设置这些参数，在 DOS 命令行下用 TYPE SET.MPLINK 命令查看，设置库文件路径指向 C:\MCC\LIB。如果 MPLAB-C17/C18 安装在其他的目录里，改变批处理文件中关于 MPLINK 的命令行。

在 MPLAB 项目（PROJECT）里创建此应用：

1．给新项目取名 APP1
   - APP4.HEX-变为项目的顶端节点

2. 设置 MPLINK 作为 APP4.HEX 的语言工具
3. 在项目中添加这些节点，设置语言工具为：
   - delayms.asm – 语言工具：MPASM
   - main.c – 语言工具： MPLAB-C17/C18
   - morse.c – 语言工具： MPLAB-C17/C18
   - portb.c – 语言工具： MPLAB-C17/C18
   - usart.c – 语言工具： MPLAB-C17/C18
   - morse756.lkr
   - p17C756.o – 该文件在目录： mcc\lib
   - C0L17.O – 该文件在目录： mcc\lib
   - IDATA17.O – 该文件在目录： mcc\lib
   - PMC756L.LIB – 该文件在目录： mcc\lib
   - INT756L.LIB – 该文件在目录： mcc\lib

注意：当链接时，你可能遇到以下信息：
“Warning – Could not open source file '<filename>'.
This file will not be present in the list file.”
（警告：无法打开源文件“文件名”。）
（这个文件将不会在列表文件里出现）
这是由于使用了预编译的库文件，而这些库的源文件不在默认目录里。

10．4 源代码

10.4.1 morse756.lkr

```
// File: morse756.lkr
LIBPATH .
CODEPAGE NAME=vectors START=0x0     END=0x27    PROTECTED
CODEPAGE NAME=page0   START=0x28    END=0x1FFF
CODEPAGE NAME=page1   START=0x2000  END=0x3FFF   PROTECTED
CODEPAGE NAME=config  START=0xFE00  END=0xFE0F   PROTECTED

SHAREBANK NAME=sfrnobnk START=0x0   END=0xF   PROTECTED
SHAREBANK NAME=sfrnobnk START=0x100 END=0x10F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x200 END=0x20F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x300 END=0x30F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x400 END=0x40F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x500 END=0x50F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x600 END=0x60F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x700 END=0x70F PROTECTED

DATABANK NAME=sfr0 START=0x10   END=0x17  PROTECTED
DATABANK NAME=sfr1 START=0x110 END=0x117 PROTECTED
DATABANK NAME=sfr2 START=0x210 END=0x217 PROTECTED
DATABANK NAME=sfr3 START=0x310 END=0x317 PROTECTED
DATABANK NAME=sfr4 START=0x410 END=0x417 PROTECTED
DATABANK NAME=sfr5 START=0x510 END=0x517 PROTECTED
DATABANK NAME=sfr6 START=0x610 END=0x617 PROTECTED
DATABANK NAME=sfr7 START=0x710 END=0x717 PROTECTED

SHAREBANK NAME=sfrprod START=0x18   END=0x19  PROTECTED
SHAREBANK NAME=sfrprod START=0x118 END=0x119 PROTECTED
SHAREBANK NAME=sfrprod START=0x218 END=0x219 PROTECTED
SHAREBANK NAME=sfrprod START=0x318 END=0x319 PROTECTED
SHAREBANK NAME=sfrprod START=0x418 END=0x419 PROTECTED
SHAREBANK NAME=sfrprod START=0x518 END=0x519 PROTECTED
SHAREBANK NAME=sfrprod START=0x618 END=0x619 PROTECTED
SHAREBANK NAME=sfrprod START=0x718 END=0x719 PROTECTED
SHAREBANK NAME=share0  START=0x1A   END=0x1F  PROTECTED
SHAREBANK NAME=share0  START=0x11A END=0x11F PROTECTED
SHAREBANK NAME=share0  START=0x21A END=0x21F PROTECTED
SHAREBANK NAME=share0  START=0x31A END=0x31F PROTECTED
SHAREBANK NAME=share0  START=0x41A END=0x41F PROTECTED
SHAREBANK NAME=share0  START=0x51A END=0x51F PROTECTED
SHAREBANK NAME=share0  START=0x61A END=0x61F PROTECTED
SHAREBANK NAME=share0  START=0x71A END=0x71F PROTECTED

DATABANK NAME=stackram START=0x20  END=0x4F   PROTECTED
DATABANK NAME=gpr0     START=0x50  END=0xFF   PROTECTED
DATABANK NAME=gpr1     START=0x120 END=0x1FF PROTECTED
DATABANK NAME=gpr2     START=0x220 END=0x2FF PROTECTED
DATABANK NAME=gpr3     START=0x320 END=0x3FF

// 在此放置你的区段
SECTION NAME=global_vars   RAM=share0
SECTION NAME=morse_dat    RAM=gpr0
SECTION NAME=morse_buf    RAM=gpr1
SECTION NAME=delay_dat    RAM=gpr2

// 以下的堆栈指示符是使用 MPLAB-C17/C18时用到
STACK SIZE=0x30 RAM=stackram
```

10.4.2 main.c

```
#include "usart.h"
#include "morse.h"
#include "delayms.h"
#include <p17c756.h>

void main( void )
{
   InitializeMorseTable();
   Install_INT( GetMorseFromUSART );
   InitializeUSART();
   while( DisplayNextMorseVal() )
  {
   DelayXMS( 100 );
  }
  TerminateUSARTInput();
}
```

10.4.3 morse.h

```
#ifndef _MORSE_H_
#define _MORSE_H_

#define MORSE_BUFFER_SIZE 220

#define DOT       1
#define DASH      2
#define TERMINATE 3

#define DOT_LENGTH  100
#define DASH_LENGTH  250

void InitializeMorseTable( void );
unsigned char DisplayNextMorseVal( void );

#endif
```

10.4.4 morse.c

```
#include "portb.h"
#include "morse.h"
#include "delayms.h"

#pragma udata shared global_vars
unsigned char *CurrentBufferWritePos;
unsigned char *CurrentBufferReadPos;

#pragma udata morse_buf
unsigned char USARTBuffer[MORSE_BUFFER_SIZE];

#pragma udata morse_dat
void InitializeMorseTable( void )
{
   CurrentBufferWritePos = USARTBuffer;
   CurrentBufferReadPos = USARTBuffer;
   PortBInitialize();
}

unsigned char DisplayNextMorseVal( void )
{
   unsigned int HoldTime;
   if( CurrentBufferReadPos == CurrentBufferWritePos )
   return 1;

   if( CurrentBufferReadPos == USARTBuffer )
CurrentBufferReadPos = USARTBuffer + MORSE_BUFFER_SIZE - 1;
else
CurrentBufferReadPos--;

switch( *CurrentBufferReadPos )
  {
   case TERMINATE:
      return 0;

   case DOT:
      HoldTime = DOT_LENGTH;
   break;

   case DASH:
      HoldTime = DASH_LENGTH;
   break;

   default:
      return 1;
  }

   PortBOutput( 0xFF );
   DelayXMS( HoldTime );
   PortBOutput( 0x00 );
   DelayXMS( HoldTime );
   return 1;
}
```

10.4.5 portb.h

```
#ifndef _PORTB_H_
#define _PORTB_H_

void PortBInitialize( void );
void PortBOutput( unsigned char OutputValue );

#endif
```

10.4.6 portb.c

```
#include "portb.h"
#include <p17C756.h>

void PortBInitialize( void )
{
    PORTB = 0xff                    ; // 清除端口 Port B 寄存器
    DDRB = 0x00                     ; // 设置端口 Port B 为输出
}

void PortBOutput( unsigned char OutputValue )
{
    PORTB = 0xff                    ; // 清除端口 Port B 寄存器
    PORTB = OutputValue             ; // 写输出值
}
```

10.4.7 usart.h

```
#ifndef _USART_H_
#define _USART_H_
void InitializeUSART( void );
void TerminateUSARTInput( void );
void GetMorseFromUSART( void );
#endif
```

10.4.8 usart.c

```
#include "usart.h"
#include "morse.h"
#include <usart16.h>
#include <p17C756.h>

extern unsigned char USARTBuffer[];
extern unsigned char *CurrentBufferWritePos;
extern unsigned char *CurrentBufferReadPos;

void InitializeUSART( void )
{
   OpenUSART1( USART_TX_INT_ON & USART_ASYNCH_MODE &
   USART_EIGHT_BIT & USART_CONT_RX, 1 );
}

void TerminateUSARTInput( void )
{
   CloseUSART1();
}

void GetMorseFromUSART( void )
{
   /* Wrap if we are at the end of the buffer */
   if( CurrentBufferWritePos >= (USARTBuffer + MORSE_BUFFER_SIZE - 1) )
{
   /* Unless buffer is full, increment position */
if( CurrentBufferReadPos != USARTBuffer )
   CurrentBufferWritePos = USARTBuffer;
  }
else
{
/* Unless buffer is full, increment position */
if( CurrentBufferReadPos != (CurrentBufferWritePos + 1) )
  {
   CurrentBufferWritePos++;
}
}
*CurrentBufferWritePos = ReadUSART1();
```

10.4.9 delayms.h

```
#ifndef _DELAYMS_H_
#define _DELAYMS_H_

void DelayXMS( unsigned int time );

#endif
```

10.4.10 delayms.asm

```
          list p=17c756
          #include <p17c756.inc>

          EXTERN _stack
          GLOBAL DelayXMS
delay_dat UDATA
MSB        RES 1
LSB        RES 1

          CODE

DelayXMS:
          banksel _stack
          movfp _stack,FSR0
          decf FSR0,f
          movfp INDF0,MSB
          decf FSR0,f
          movfp INDF0,LSB
dly1:
          nop
          nop
          decfsz LSB,1
          goto dly1
          decfsz MSB,1
          goto dly1
          return

          end
```

第三部分　MPLIB

第 1 章　MPLIB 概述

1．1　介绍

见本章所有的标题。

1．2　重点

本章内容包括：

- 什么是 MPLIB
- MPLIB 是干什么的？
- MPLIB 能帮你做什么？

1．3　什么是 MPLIB

MPLIB 是 COFF 目标模块(FILENAME.O)的库。它可由 MPASM v2.0、MPASMWIN v2.0、 MPLAB-C v2.0 或更新版本等工具制作而成。

1．4　MPLIB 是干什么的？

库管理文件(LIBRARIAN)负责库文件的建立和修改。库文件是把很多目标模块结合在一起，形成一个文件。至于为什么要使用库文件，原因如下：

- **库文件使得链接更容易**。因为库文件可以包含很多目标文件，当链接的时候，就不用使用很多目标文件，而只需方便地使用一个库文件即可。
- **库文件使得代码更小**。因为链接器只使用库文件中所需要的目标模块。而不是将所有的模块都链接到输出模块中。
- **库文件使得项目更好维护**。如果项目中包含有库，对该库中目标模块调用的增加和减少不需要改变链接处理过程。
- **库文件帮助了解库文件的相关功能**。因为将功能相近的目标模块组合在一起比单独使用这些模块更容易让人理解。例如：一个名为 math.lib 的库文件比名为 power.lib、ceiling.lib、floor.lib 的库文件更让人容易理解其功能。

1．5　MPLIB 能帮你做什么？

MPLIB 可以帮助你干这些事情：

- MPLIB 使得链接更容易。当链接的时候，就不用使用很多目标文件，而只需方便地包含一个库文件即可。

- MPLIB 将相近功能的模块结合在一起，增加代码的可维护性。
- MPLIB 命令可以控制库的建立、模块的增加、列表、替换、删除或提取（EXTRACT）。

第 2 章 MPLIB 软件包的安装与入门

2．1 介绍

本章讨论如何安装 MPLIB，并初步介绍库管理程序。

2．2 重点

本章内容包括：

- 安装
- 初步介绍库管理程序

2．3 MPLIB 安装

MPLIB 有两种版本：DOS 版本的 MPLIBD.EXE，推荐使用于 DOS 和 WINDOWS3.X 操作系统环境；MPLIB.EXE 一个 32-BIT 的 WINDOWS 环境应用程序，应用于 WINDOWS95/98/NT。

假如你使用 MPLAB 软件包的话，就没有必要单独安装库管理程序。因为 MPLAB 软件包安装完成后，MPLIB 也随即完成了安装。请参考《MPLAB 用户指南》（文档资料号 DS51025）了解 MPLAB 的安装。用户可以从 MICROCHIP 的资料库 CD-ROM 或者 WEB 站点获得 MPLAB 软件包或用户指南。

假如你不使用 MPLAB，则可以可以从 MICROCHIP 的资料库 CD-ROM 或者 WEB 站点单独获得 MPLIB。MPLINK 和 MPLIB 被捆绑在一起，制作成.ZIP 格式的文件。安装的时候，按如下步骤。

- 建立一个子目录，将文件拷贝到该目录之下。
- 使用 WINZIP 或 PKZIP 解压缩程序把 MPLINK 和 MPLIB 解压缩。

2．4 MPLIB 库概述

图 2-1 显示 MPLIB 和 MICROCHIP 的其他开发工具一起工作的框图。

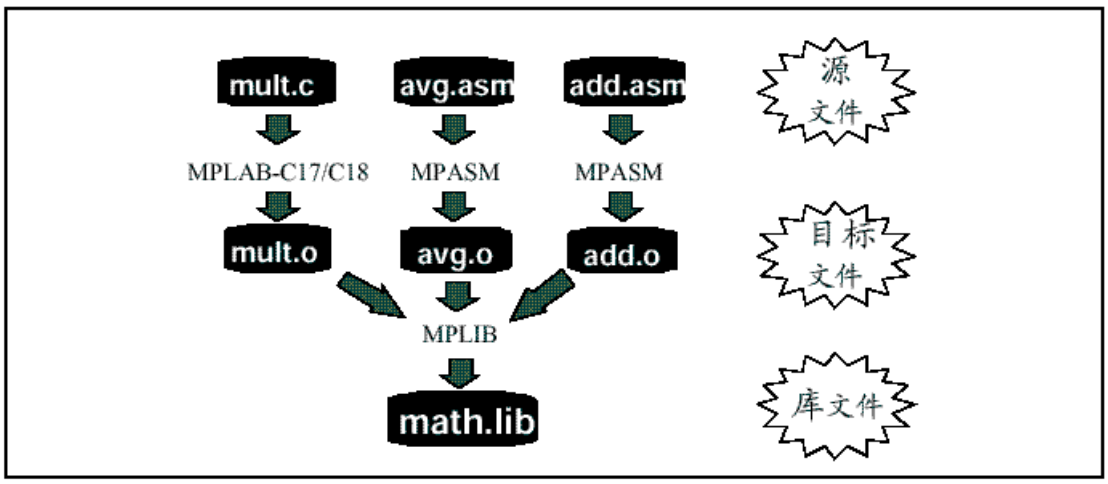


图 2-1：MPLIB 和 MICROCHIP 的其他开发工具

MPLIB 把多个由 MPASM 或 MPLAB-C17/C18 生成的输入模块组合起来，生成一个输出库文件（.LIB）。库文件和链接器一起使用（更多关于 MPLINK 的信息，见第 2 章，“MPLINK-安装与开始使用”），生成可执行文件。

第 3 章　使用 MPLIB 软件包

3．1　介绍

本章讨论如何使用库管理文件（MPLIB）。

3．2　重点

本章内容包括：

- MPLIB 的使用格式
- 使用范例
- 提示与建议

3．3　MPLIB 使用格式

用下列方式调用 MPLIB：

```
mplib [/q] /{ctdrx} LIBRARY [MEMBER…]
```

选项：

/c　**建立库**。用所指定的成员（MEMBER）建立一个新库。

/t　**列出成员**。把库里的成员列表打印出来。

/d　**删除成员**。删除库里的成员，假如不指定成员，库文件没有变化。

/r　**增加/替换成员**。如果库里有所指定的成员，则替换之，否则指定的成员被加到库的尾部。

/x　**提取成员**。如果库里有成员，则它们会被提取（extract），如果没有指定成员，则所有的成员都被提取。

/q　**静默工作模式**（QUIET MODE）。没有任何输出。

3．4　应用范例

假如有一个名叫 dsp.lib 的库文件，由 fft.o、fir.o、iir.o 三个目标模块所生成。使用下列命令就可以达到目的：

```
mplib /c dsp.lib fft.o fir.o iir.o
```

要想显示库文件 dsp.lib 中的目标模块名称，可以使用以下命令：

```
mplib /t dap.lib
```

3．5 一些建议与提示

MPLIB 生成的库文件对任何符号只能有一个外部定义。假如两个目标模块定义了相同的外部符号，这两个目标模块被加入到同一库文件中时，MPLIB 将生成错误报告。

为了优化用链接程序链接库文件以后程序的代码和数据空间，库里的成员应该尽量小。建立只含有单一功能的目标模块将回大大地节省空间。

附录 A 16 进制文件格式

A1 介绍

MPASM 可以生成好几种文件格式。

A2 重点

本章内容包括：

- INTEL-HEX 格式（INHX8M）- 适合于标准编程器。
- INTEL SPLIT HEX 格式（INHX8S）- 适合于奇/偶 ROM 编程器。
- INTEL HEX-32 格式（INHX32）- 适合于 16-BIT 核编程器。

A3 INTEL - HEX 格式（.HEX）

这种格式是低字节和高字节组合的 8-BIT 的 16 进制(HEX)文件。由于 这种格式的每个地址只包含一个 8-BIT 数据，所以所有的地址被加倍了。这种文件格式适合于将 PICmicro 系列 MCU 的代码传送到 PRO MATE，PICSTART 及其他第三方编程器。

每个数据代码由 9 个预先设置的字符前缀(PREFIX)开始，2 个字符的校验和(CHECHSUM)结尾。格式如下：

```
:BBAAAATTHHHH....HHCC
```

其中：

BB- 2 位 16 进制数字，代表一行的数据字节总和。

AAAA- 4 位 16 进制数字，代表代码的开始地址。

TT- 2 位 16 进制数字，代表记录类型，常为 00，只有结尾时是 01。

HH- 2 位 16 进制数字，代表低/高字节组合。

CC- 2 位 16 进制数字，代表记录里所有字节的校验和,是前面所有字节

总和的补码 。

例如：

例 A-1:

```
          <文件名>.HEX
:10000000000000000000000000000000000000000000F0
:0400100000000000EC
:10003200000028004000680OA800E800C80028016D
:100042006801A9018901EA01280208026A02BF02C5
:10005200E002E80228036803BF03E803C8030804B8
:1000620008040804030443050306E807E807FF0839
:06007200FF08FF08190A57
:00000001FF
```

A4 8BIT-切分格式（.HXL/HXH）

这种切分（SPLIT）8-BIT 文件格式生成两种文件：.HXL 和.HXH。除了该字的低位字节存在.HXL 文件中，高位字节存在.HXH，文件中地址被除以 2 外，其格式与通常的 8-BIT 文件格式完全一样。这被用来将 16-BIT 的代码写入一对 EPROM 中。一个文件写入低位字节， 另一个文件写入高位字节。

例 A-2：

```
<文件名>.HXL
:0A000000000000000000000000000000F6
:10001900002840688E8C82868A989EA28086ABFAA
:10002900E0E82868BFE8C8080808034303E8E8FFD0
:03003900FFFF19AD
:00000001FF
<文件名>.HXH
:0A000000000000000000000000000000F6
:10001900000000000000000010101010102020202CA
:10002900020203030303030404040405060707088 3
:0300390008080AAA
:00000001FF
```

A5 32-BIT16 进制格式（.HEX）

除了输出扩展线性地址记录，从而建立数据地址的高 16-BIT 外，扩展 32-BIT 地址 16 进制格式上面介绍的 8-BIT 格式相似。

这主要用在 16-BIT 核的 MCU 上，因为他们的程序存储器器可寻址范围超过了 32K 字。每个数据代码由 9 个预先设置的前缀字符开始，2 个字符校验和结尾。格式如下：

```
:BBAAAATTHHHH....HHCC
```

其中：

BB- 2 位 10 进制数字，代表一行里的数据字节总和。

AAAA- 4 位 16 进制数字，代表数据代码的开始地址。

TT- 2 位 16 进制数字，代表记录类型，具体安排：

00- 数据记录，

01- 文件结尾记录

02- 段地址记录

04- 线性地址记录

HH- 2 位 16 进制数字，代表低/高字节组合。

CC- 2 位 16 进制数字，代表记录里所有字节的校验和,是前面所有字节总和的补码 。

附录 B 快速参考

B1 介绍

本附录列出了关于 MPASM 和 PICmicro 系列 MCU 指令集的缩略语，帮助你用 MPASM，MPLINK 和 MPLIB 开发应用项目时参考。

B2 重点

本章内容包括：

- MPASM 快速参考
- PICmicro 指令集关键字
- 12-BIT 核指令集
- 14-BIT 核指令集
- 16-BIT 核指令集
- 增强型 16-BIT 核 MCU 指令集关键字
- 增强型 16-BIT 核 MCU 指令集
- 16 进制-10 进制代码转换
- ASCII 码表

B3 MPASM 快速参考

下列快速指南给出所有 MICROCHIP MPASM 中的指令，指示语言，命令行选项。

表 B-1：MPASM 指示语言集：

指示语言	描述	语法
控制型指示语言		
CONSTANT	说明符号常量	constant <标号> [= <表达式>,

		...,<标号> [= <表达式>]]
#DEFINE	定义一个文本替换符号	#define <名称> [[(<参数>,...,<参数>)]<数值>]
END	程序结束标志	End
EQU	定义一个汇编常量	<标号> equ <表达式>
ERROR	产生一条错误信息	error "<字符串>"
ERRORLEVEL	设置信息优先级	errorlevel 0|1|2|<+-><msg>
#INCLUDE	包含另外的源文件	include <<包含文件>> include "<包含文件>"
LIST	列表选项	list [<选项>[,...,<选项>]]
MESSG	建立用户自定义信息	messg "<信息文本>"
NOLIST	关闭列表选项	Nolist
ORG	设置程序起始地址	<标号> org <表达式>
PAGE	插入页到列表中	Page
PROCESSOR	设置处理器类型	processor <处理器类型>
RADIX	定义默认基数	radix <默认基数>
SET	定义一个汇编变量	<标号> set <表达式>
SPACE	在列表中插入空行	space [<表达式>]
SUBTITLE	指定程序子标题	subtitl "<子标题>"
TITLE	指定程序标题	title "<标题>"
#UNDEFINE	删除一个替换符号	#undefine <标号>
VARIABLE	说明符号变量	variable <标号> [= <表达式>,..., <标号> [= <表达式>]]
条件汇编		
ELSE	开始条件汇编的另一分支	Else
ENDIF	开始条件汇编的另一分支	Endif
ENDW	WHILE 循环的结尾	Endw
IF	开始条件汇编	if <表达式>
IFDEF	如果符号定义则执行	ifdef <标号>
IFNDEF	如果符号未定义则执行	ifndef <标号>
WHILE	条件为真时执行循环体	while <表达式>
数据		
_ _BADRAM	标注不可用 RAM	_ _badram <表达式>
CBLOCK	标注不可用 RAM	cblock [<表达式>]
_ _CONFIG	设置处理器配置位	_ _config <表达式> 或 _ _config <地址>, <表达式>
DA	字符串存入程序存储器中	[<标号>] da <表达式> [, <表达式2>, ..., <表达式 n>]
DATA	建立数字和文本数据	data <表达式>,[,<表达式>,...,<表达式>] data "<字符串>"[,"<字符串>",...]
DB	说明一个字节数据	db <表达式> [,<表达式>,...,<表达式>]
DE	说明一个 EEPROM 字节	de <表达式> [,<表达式>,...,<表达式>]
DT	定义表格	dt <表达式> [,<表达式>,...,<表达式>]
DW	说明一个字数据	dw <表达式> [,<表达式>,...,<表达式>]
ENDC	结束一个自动常量块	Endc
FILL	指定内存填充值	fill <表达式>, <数量>
_ _ IDLOCS	设置处理器 ID 位置	_ _idlocs <表达式>
_ _MAXRAM	定义最大的 RAM 位置	_ _maxram <表达式>
RES	保留存储器	res <存储器单元>
宏		
ENDM	结束宏	endm
EXITM	退出宏	exitm
EXPAND	展开宏列表	expand
LOCAL	说明局部变量	local <标号> [,<标号>]
MACRO	宏定义	<标号> macro [<参数>,...,<参数>]
NOEXPAND	关闭宏扩展	noexpand

目标文件指示语言		
BANKISEL	产生间接 RAM 堆选择	bankisel <标号>
BANKSEL	产生 RAM 堆选择	banksel <标号>
CODE	开始一个目标代码的选项	[<名称>] code [<地址>]
EXTERN	定义外部定义标号	extern <标号> [,<标号>]
GLOBAL	出口标号	extern <标号> [.<标号>]
IDATA	开始目标文件初始数据	[<名称>] idata [<地址>]
PAGESEL	产生页选择码	pagesel <标号>
UDATA	开始目标文件未初始化数据位置	[<名称>] udata [<地址>]
UDATA_ACS	访问目标文件未初始化数据位置	[<名称>] udata_acs [<地址>]
UDATA_OVR	覆盖目标文件未初始化数据位置	[<名称>] udata_ovr [<地址>]
UDATA_SHR	共享目标文件未初始化数据位置	[<名称>] udata_shr [<地址>]

表 B-2：MPASM 命令行选项：

选项	默认	描述
?	N/A	
a	INHX8M	/a<十六进制格式> 其中 <十六进制格式>是[INHX8M | INHX8S | INHX32]之一
c	On	允许/禁止大小写
d	N/A	/d<标号>[=<数值>]
e	On	/e 允许 /e + 允许 /e - 禁止 /e <路径>error.file 允许/设置路径
h	N/A	
i	On	/l 允许 /l + 允许 /l - 禁止 /l <路径>list.file 允许/设置路径
m	On	
o	Off	/o 允许 /o + 允许 /o - 禁止 /o <路径>object.file 允许/设置路径
p	None	/p<处理器类型> 其中<处理器类型> 是一个PICmicro芯片. 例如, PIC16C54.
q	Off	
r	Hex	/r<基数> 其中<基数> 是以下之一： [HEX | DEC | OCT]
t	8	size: /t<大小>
w	0	设置信息级别: /w<数值> 其中<level> 是以下之一： [0|1|2] 0 – 所有信息 1 – 错误和警告

		2 – 错误
x	off	/x 允许 /x + 允许 /x - 禁止 /x <路径>xref.file 允许/设置路径

表 B-3：进制基数（RADIX）类型支持：

基数	语法	范例
十进制	D'<数字>'	D'100'
十六进制	H'<十六进制数字>' 0x<十六进制数字>	H'9f' 0x9f
八进制	O'<八进制数字>'	O'777'
二进制	B'<二进制数字>'	B'00111001'
字符（ASCII）	'<字符>' A'<字符>'	'C' A'C'

表 B-4：MPASM 数学运算符：

运算符号	描述	范例
$	返回程序计数器	goto $ + 3
(	左括号	1 + (d * 4)
)	右括号	(Length + 1) * 256
!	逻辑“非”	if ! (a - b)
-	负	-1 * Length
~	按位取反	flags = ~flags
high	返回高字节	movlw high CTR_Table
low	返回低字节	movlw low CTR_Table
upper	返回上位字节	movlw upper CTR_Table
*	乘	a = b * c
/	除	a = b / c
%	取模	entry_len = tot_len % 16
+	加	tot_len = entry_len * 8 + 1
-	减	entry_len = (tot - 1) / 8
<<	左移	val = flags << 1
>>	右移	val = flags >> 1
>=	不小于	If entry_idx >= num_entries
>	大于	If entry_idx > num_entries
<	小于	If entry_idx < num_entries
<=	不大于	if entry_idx <= num_entries
==	等于	if entry_idx == num_entries
!=	不等于	if entry_idx != num_entries
&	位“与”	flags = flags & ERROR_BIT
^	位“异或”	flags = flags ^ ERROR_BIT
|	位“同或”	flags = flags | ERROR_BIT
&&	逻辑“与”	if (len == 512) && (b == c)
||	逻辑“或”	if (len == 512) || (b == c)
=	赋值	entry_index = 0
+=	加，再赋予	entry_index += 1

-=	减，再赋予	entry_index -= 1
*=	乘，再赋予	entry_index *= entry_length
/=	除，再赋予	entry_total /= entry_length
%=	取模，再赋予	entry_index %= 8
<<=	左移，再赋予	flags <<= 3
>>=	右移，再赋予	flags >>= 3
&=	与，再赋予	flags &= ERROR_FLAG
|=	同或，再赋予	flags |= ERROR_FLAG
^=	异或，再赋予	flags ^= ERROR_FLAG
++	加 1	i ++
--	减 1	i --

B4 PICMIRO 系列指令集中用到的关键字：

关键字	描 述
B	8 位文件寄存器的某一位地址
D	选择目标； d = 0 结果存于 W (f0A). d = 1 结果存于文件寄存器 f. 默认 d = 1.
F	寄存器文件地址(0x00 to 0xFF)
K	常量区域，常量数据或标号
W	工作寄存器（累加器）
X	不关心区域
i	表指针控制 ；i = 0 不变化 i = 1 执行指令后加一
p	外设寄存器文件地址(0x00 到 0x1f)
t	表字节选择 ；t = 0 在低位字节执行 t = 1 在高位字节执行
PH:PL	乘法结果寄存器

B．5 12-BIT 核指令集

12-BIT 核（指令宽度 12-BIT）MCU 是 MICROCHIP 单片机的基本系列。个别跳转指令除外其指令均为单机器周期。其他未用到的代码均被视为 NOP 操作。其指令集可归为以下几类：

表 B-5：12-BIT 核的常量和控制操作：

Hex 码	助记符号	功能描述	功能示意
Ekk	ANDLW k		k .AND. W →W
9kk	CALL k	子函数调用	PC + 1 →TOS, k →PC
004	CLRWDT	清看门狗记数器	0 →WDT (如果已赋值,清预分频器)
Akk	GOTO k	跳到某地址（k 为 9	k →PC(9 bits)

		位)	
Dkk	IORLW k	常量和 W 同或	k .OR. W →W
Ckk	MOVLW k	将常量送到 W	k →W
002	OPTION	设置配置寄存器	W →OPTION 寄存器
8kk	RETLW k	带着 W 的值返回	k →W, TOS →PC
003	SLEEP	进入睡眠状态	0 →WDT, 停振
00f	TRIS f	F 口设置三态	W →I/O 控制寄存器 f
00f	XORLW k	常量与 W 异或	k .XOR. W →W

表 B-6：12-BIT 核面向“字节”的文件寄存器操作：

Hex 码	助记符号	功能描述	功能示意
1Cf	ADDWF f,d	W 和 f 相加	W + f ••d
14f	ANDWF f,d	W 和 f 相与	W .AND. f ••d
06f	CLRF f	清零 f	0 ••f
040	CLRW	清零 W	0 ••W
24f	COMF f,d	f 取反	.NOT. f ••d
0Cf	DECF f,d	f 减 1	f - 1 ••d
2Cf	DECFSZ f,d	f 减 1, 到 0 时，则跳过下一行	f - 1 ••d, 若为 0，跳过下一条语句
28f	INCF f,d	f 增 1	f + 1 ••d
3Cf	INCFSZ f,d	f 增 1, 到 0 时，则跳过下一行	f + 1 ••d,若为 0，跳过下一条语句
10f	IORWF f,d	W 和 f 相同或	W .OR. f ••d
20f	MOVF f,d	d 赋予 f	f ••d
02f	MOVWF f	W 赋予 f	W ••f
000	NOP	空操作	
18f	XORWF f,d	W 和 f 相异或	W .XOR. f ••d
08f	SUBWF f,d	从 f 中减去 W	f - W ••d
38f	SWAPF f,d	F 的高低四位互换	f(0:3) ••f(4:7) ••d
000	NOP	空操作	

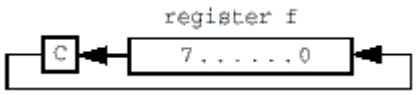
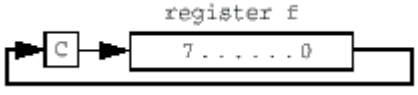
34f	RLF f,d	穿越进位的左移	
30f	RRF f,d	穿越进位的右移	

表 B-7：12-BIT 核面向“位”的文件寄存器操作：

Hex 码	助记符号	功能描述	功能示意
4bf	BCF f,b	清零 f 的第 B 位	0 → f(b)
5bf	BSF f,b	置位 f 的第 B 位	1 → f(b)
6bf	BTFSC f,b	若 f 的第 b 位为 0，则跳过下一条语句	f(b) = 0 跳过下条语句
7bf	BTFSS f,b	若 f 的第 b 位为 1，则跳过下一条语句	f(b) = 1 跳过下条语句

B6 14-BIT 核指令集

MICROCHIP 的中档 MCU 采用 14BIT 指令宽度，指令集有 36 条指令，都为单个 14BIT 宽的字。大多数指令通过文件寄存器 F 和工作寄存器 W（累加器）运算。运算结果既可以存入文件寄存器 F，也可以存入 W 中，有的指令还可以将结果同时存入 F 和 W。个别指令只和工作寄存器打交道（如 BSF）。其指令集可归为以下几类：

表 B-8：14-BIT 核的常量和控制操作：

Hex 码	助记符号	功能描述	功能示意
3Ekk	ADDLW k	常量+W	k + W → W
39kk	ANDLW k	常量 或 W	K .AND. w → W
2kkk	CALL k	子函数调用	PC + 1 → TOS, k → PC
0064	CLRWDT	清看门狗记数器	0 → WDT (如果已赋值,清预分频器)
2kkk	GOTO k	跳到某地址(k :9bits)	k → PC(9 bits)
38kk	IORLW k	常量和 W 同或	k .OR. W → W
30kk	MOVLW k	将常量送到 W	k → W
0062	OPTION	设置配置寄存器	W → OPTION 寄存器
34kk	RETLW k	带着 W 的值返回	k → W, TOS → PC
0008	RETURN	子程序返回	TOS→ PC
0063	SLEEP	进入睡眠状态	0 → WDT, 停振
006f	TRIS f	设置 I/O 状态	W → I/O 控制寄存器 f
3Akk	XORLW k	常量与 W 异或	k .XOR. W → W

表 B-9：14-BIT 核面向“字节”的文件寄存器操作：

Hex 码	助记符号	功能描述	功能示意
07ff	ADDWF f,d	W 和 f 相加	W + f → d
05ff	ANDWF f,d	W 和 f 相与	W .AND. f → d
018f	CLRF f	清零 f	0 → f
0100	CLRW	清零 W	0 → W
09ff	COMF f,d	f 取反	.NOT. f → d
03ff	DECF f,d	f 减 1	f - 1 → d
0Bff	DECFSZ f,d	f 减 1, 到 0 时，则跳过下一行	f - 1 → d, 若为 0, 跳过下一条语句
0Aff	INCF f,d	f 增 1	f + 1 → d

0Fff	INCFSZ f,d	f 增 1, 到 0 时，则跳过下一行	f + 1 →d,若为 0，跳过下一条语句
04ff	IORWF f,d	W 和 f 相同或	W .OR. f →d
08ff	MOVF f,d	d 赋予 f	f →d
008f	MOVWF f	W 赋予 f	W →f
0000	NOP	空操作	
02ff	SUBWF f,d	从 f 中减去 W	f - W →d
0Eff	SWAPF f,d	f 的高低 4 位交换	f(0:3) ↔f(4:7) →d
06ff	XORWF f,d	W 和 f 相异或	W .XOR. f →d
000	NOP	空操作	

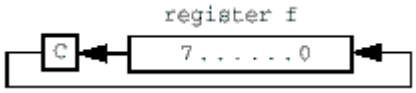
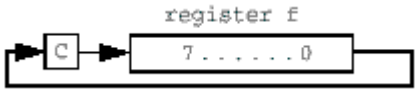
0Dff	RLF f,d	穿越进位的左移	
0Cff	RRF f,d	穿越进位的右移	

表 B-10：14-BIT 核面向“位”的文件寄存器操作：

Dr4Hex 码	助记符号	功能描述	功能示意
1bff	BCF f,b	清零 f 的第 B 位	0 →f(b)
1bff	BSF f,b	置位 f 的第 B 位	1 →f(b)
1bff	BTFSC f,b	若 f 的第 b 位为 0，则跳过下一条语句	f(b) = 0 跳过下条语句
1bff	BTFSS f,b	若 f 的第 b 位为 1，则跳过下一条语句	f(b) = 1 跳过下条语句

表 B-11：12/14-BIT 核的特殊指令：

助记符	功能描述	等价操作	状态位
ADDCF f,d	进位加到文件寄存器	BTFSC 3,0 INCF f,d	Z
ADDDCF f,d	辅助进位加到文件寄存器	BTFSC 3,1 INCF f,d	Z
B k	分支跳转	GOTO k	-
BC k	若进位 C=1 则跳转	BTFSC 3,0 GOTO k	-
BDC k	若辅助进位 DC=1 则跳转	BTFSC 3,1 GOTO k	-
BNC k	若进位 C=0 则跳转	BTFSS 3,0 GOTO k	-
BNDC k	若辅助进位 DC=0 则跳转	BTFSS 3,1 GOTO k	-
BNZ k	若零位 Z=0 则跳转	BTFSS 3,2 GOTO k	-
BZ k	若零位 Z=1 则跳转	BTFSC 3,2 GOTO k	-
CLRC	清零进位 C	BCF 3,0	-
CLRDC	清零辅助进位 DC	BCF 3,1	-
CLRZ	清零零位 Z	BCF 3,2	-
LCALL k			
LGOTO k			
MOVFW f	文件寄存器赋予 W	MOVF f,0	Z

NEGF f,d	文件寄存器取反	COMF f,1 INCF f,d	Z
SETC	置位进位，C=1	BSF 3,0	-
SETDC	置位辅助进位 DC=1	BSF 3,1	-
SETZ	置位零位 Z=1	BSF 3,2	-
SKPC	进位 C=1 则跳过下一语句	BTFSS 3,0	-
SKPDC	辅助进位 DC=1 则跳过下一语句	BTFSS 3,1	-
SKPNC	进位 C=0 则跳过下一语句	BTFSC 3,0	-
SKPNDC	辅助进位 DC=0 则跳过下一语句	BTFSC 3,1	-
SKPNZ	零位 Z=0 则跳过下一语句	BTFSC 3,2	-
SKPZ	零位 Z=1 则跳过下一语句	BTFSS 3,2	-
SUBCF f,d	文件寄存器位 d 减去进位 C	BTFSC 3,0 DECF f,d	Z
SUBDCF f,d	文件寄存器位 d 减去辅助进位 C	BTFSC 3,1 DECF f,d	Z
TSTF f	检测文件寄存器	MOVF f,1	Z

B7 16-BIT 核指令集

MICROCHIP 的高档 MCU 采用 16BIT 指令宽度，指令集有 55 条指令，都为单个 16BIT 宽的字。大多数指令通过文件寄存器 F 和工作寄存器 W（累加器）运算。运算结果既可以存入文件寄存器 f，也可以存入 W 中，有的指令还可以将结果同时存入 F 和 W。一些型号还内建硬件乘法器。个别指令只和工作寄存器打交道（如 BSF）。其指令集可归为以下几类：

表 B-12：16-BIT 核数据传输指令：

Hex 码	助记符号	功能描述	功能示意
6pff	MOVFP f,p	将 f 的内容送入 p	f → p
b8kk	MOVLB k	将常数送入 BSR	k → BSR (3:0)
bakx	MOVLP k	将常数送入 RAM 页选	k → BSR (7:4)
4pff	MOVPF p,f	将 p 的内容送入 f	p → W
01ff	MOVWF f	将 W 的内容送入 F	W → f
a8ff	TABLRD t,i,f	将表锁存器的内容送入 f 然后用表指针所指存储器 的 16-bit 常数更新表锁存器	TBLATH → f 若： t=1, TBLATL → f 若： t=0; ProgMem(TBLPTR) → TBLAT; TBLPTR + 1 → TBLPTR 若： i=1
acff	TABLWT t,i,f	将 f 的内容写入表锁存器里， 再将 16bit 表锁存器内容送入 表指针指向的程序存储器里	f → TBLATH 若： t = 1, f → TBLATL 若： t = 0; TBLAT → ProgMem(TBLPTR); TBLPTR + 1 → TBLPTR 若： i=1
a0ff	TLRD t,f	将表锁存器的内容读入 f （表锁存器内容不变）	TBLATH → f 若： t = 1 TBLATL → f 若： t = 0
a4ff	TLWT t,f	将 f 的内容写入表锁存器	f → TBLATH 若： t = 1 f → TBLATL 若： t = 0

表 B-13：16-BIT 核算数逻辑指令：

Hex 码	助记符号	功能描述	功能示意
b1kk	ADDLW k	将常数加到 W 里	(W + k) → W
0eff	ADDWF f,d	将 W 的内容加到 F 里	(W + f) → d
10ff	ADDWFC f,d	将 W 的内容和 Cy 加到 F 里	(W + f + C) → d
b5kk	ANDLW k	常数和 W 内容相与	(W .AND. k) → W
0aff	ANDWF f,d	W 内容和 f 相与	(W .AND. f) → d

28ff	CLRF f,d	清除 f 和 d	0x00 → f,0x00 → d
12ff	COMF f,d	将 f 取反	.NOT. f → d
2eff	DAW f,d	对 W 做 10 进制调整 结果存入 f 和 d 中	W 调整后 → f and d
06ff	DECF f,d	F 做减 1 操作	(f - 1) → f and d
14ff	INCF f,d	F 做加 1 操作	(f + 1) → f and d
b3kk	IORLW k	常数与 W 做“同或”逻辑	(W .OR. k) → W
08ff	IORWF f,d	W 和 F 做“同或”逻辑	(W .OR. f) → d
b0kk	MOVLW k	将常数送入 W 里	k → W
bckk	MULLW k	常数与 W 相乘	(k x W) → PH:PL
34ff	MULWF f	W 和 F 相乘	(W x f) → PH:PL
2cff	NEGW f,d	W 取反，结果在 f 和 d 中	(W + 1) → f,(W + 1) → d
2aff	SETF f,d	将 f 和 d 置位	0xff → f,0xff → d
b2kk	SUBLW k	从常数中减去 W	(k - W) → W
04ff	SUBWF f,d	从 F 中减去 W	(f - W) → d
02ff	SUBWFB f,d	从 F 中带借位减去 W	(f - W - c) → d
1cff	SWAPF f,d	F 的高、低四位互换	f(0:3) → d(4:7), f(4:7) → d(0:3)
b4kk	XORLW k	常数与 W 做“异或”逻辑	(W .XOR. k) → W
0cff	XORWF f,d	W 和 F 做“异或”逻辑	(W .XOR. f) → d

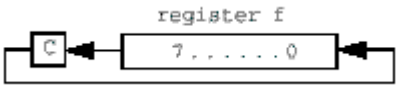
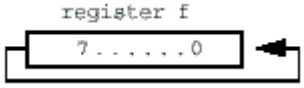
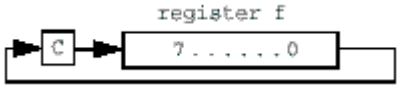
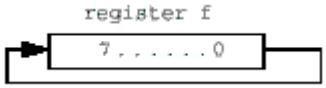
1aff	RLCF f,d	穿越进位的左移	
22ff	RLNCF f,d	不穿越进位的左移	
18ff	RRCF f,d	穿越进位的右移	
20ff	RRNCF f,d	不穿越进位的右移	

表 B-14：16-BIT 核位操作指令：

Hex 码	助记符号	功能描述	功能示意
8bff	BCF f,b	清零 f 的第 b 位	0 → f(b)
8bff	BSF f,b	置位 f 的第 b 位	1 → f(b)
9bff	BTFSC f,b	若 f 的第 b 位为 0，则跳过下一条语句	f(b) = 0 跳过下条语句
9bff	BTFSS f,b	若 f 的第 b 位为 1，则跳过下一条语句	f(b) = 1 跳过下条语句
3bff	BTG f,b	取反 f 的第 b 位	.NOT.f(b) → f(b)

表 B-15：16-BIT 核程序控制指令：

Hex 码	助记符号	功能描述	功能示意
Ekkk	CALL k	子函数调用（8K 页）	PC+1→ TOS, k→ PC(12:0), k(12:8) → PCLATH(4:0), PC(15:13) → PCLATH(7:5)
31ff	CPFSEQ f	F=W 则跳过下一行语句	f-W, 若： f = W, 跳过下一行
32ff	CPFSGT f	F>W 则跳过下一行语句	f-W, 若： f > W,跳过下一行

30ff	CPFSLT f	F<W 则跳过下一行语句	f-W, 若： f < W,跳过下一行
16ff	DECFSZ f,d	若：F 减 1，结果为 0 则：跳过下一行语句	(f-1) →d, 若： 0, 跳过下一行
26ff	DECFSZ f,d	若：F 减 1，结果不为 0 则：跳过下一行语句	(f-1) →d, 若：不为 0, 跳过下一行
Ckkk	GOTO k	无条件转移（8K 范围）	k →PC(12:0) k(12:8) →PCLATH(4:0), PC(15:13) →PCLATH(7:5)
1eff	INCFSZ f,d	若：F 加 1，结果为 0 则：跳过下一行语句	(f+1) →d, 若：为 0, 跳过下一行
24ff	INFSNZ	若：F 加 1，结果不为 0 则：跳过下一行语句	(f+1) →d, 若：不为 0 跳过下一行
B7kk	LCALL k	长调用（64K）	(PC+1)→ TOS;k→ PCL, (PCLATH)→ PCH
0005	RETFIE	中断返回；允许中断	(PCLATH)→ PCH； k→ PCL,0→ GLINTD
B6kk	RETLW k	返回；W 中带回常量	k →W, TOS →PC,(PCLATH 不变)
0002	RETURN	子程序返回	TOS →PC ，(PCLATH 不变)
33ff	TSTFSZ f	F 为 0 则跳过下一行	若： f = 0，跳过下一行

表 B-16：16-BIT 核特殊控制指令：

Hex 码	助记符号	功能描述	功能示意
0004	CLRWT	清零看门狗计数器	0 →WDT,0→ WDT预分频器 , 1 →PD, 1 →TO
0003	SLEEP	进入睡眠状态	停振,关电, 0 →WDT, 0 →WDT 预分频器 1 →PD, 1 →TO

B8 增强型 16-BIT 核指令集中涉及的关键字：

关键字	功能描述
文件地址	
f	8 位文件寄存器地址
fs	12-BIT 源文件寄存器地址
fd	12-BIT 目标文件寄存器地址
dest	若 d=0 存入 W，若 d=1 存入文件寄存器
r	FSR 编号，0，1 或 2
常量	
kk	8-BIT 常量
kb	4-BIT 常量
kc	12-BIT 常量的第 8-11 位
kd	12-BIT 常量的第 0-7 位
偏移量，地址	
nn	8-BIT 相对偏移量（带符号，2 位补码）
nd	11-BIT 相对偏移量（带符号，2 位补码）
ml	20-BIT 程序存储器地址的第 0-7 位
mm	20-BIT 程序存储器地址的第 8-19 位
xx	任何 12-BIT 数值
位（BIT）	
b	第 9-11 位；位地址
d	第 9 位。为 0 时，存入 W 中；为 1 时，存入 f 文件寄存器中
a	第 8 位。为 0 时，选共用块；为 1 时，由 B S R选择区
s	第 0 位（第 8 位供调用）； 0 =不更新； 1 （快速）=更新/存W，STSTUS,BSR

B9 增强型号 16-BIT 核指令集

表 B-17：增强型 16-BIT 核数据操作指令：

Hex 码	助记符号		功能描述	功能示意
0F kk	ADDLW	kk	常数加到 W	W+kk → W
0B kk	ANDLW	kk	常数和 W 与	W .AND. kk → W
0004	CLRWDT		清 WDT	0 → WDT, 0 → WDT postscaler, 1 → TO,1 → PD
0007	DAW		W 做 10 进制调整	若：W<3:0>>9 或DC=1, W<3:0>+ 6→W<3:0>, 否则： W<3:0> → W<3:0>; 若：W<7:4> >9 或 C=1, W<7:4>+6→W<7:4>, 否则：W<7:4> → W<7:4>;
09 kk	IORLW	kk	常数与 W 同或	W .OR. kk → W
EF kc F0 kd	LFSR	r,kd:kc	装载 12bit 常数到 FSR	kd:kc → FSRr
01 kb	MOVLB	kb	常数送入 BSR 的低 nibble(?)	kb → BSR
0E kk	MOVLW	kk	常数送入 W	kk → W
0D kk	MULLW	kk	常数与 W 乘	W * kk → PRODH:PRODL
08 kk	SUBLW	kk	常数里减 W	kk–W → W
0A kk	XORLW	kk	常数与 W 异或	W .XOR. kk → W

表 B-18：增强型 16-BIT 核存储器操作指令：

Hex 码	助记符号		功能描述	功能示意
0008	TBLRD *		读表 （TBLPTR 不变）	程序存储器(TBLPTR) → TABLAT
0009	TBLRD *+		读表 （TBLPTR 加 1）	程序存储器(TBLPTR) → TABLAT TBLPTR +1 → TBLPTR
000A	TBLRD *-		读表 （TBLPTR 减 1）	程序存储器(TBLPTR) → TABLAT TBLPTR -1 → TBLPTR
000B	TBLRD +*		读表 （TBLPTR 先加 1）	TBLPTR +1 → TBLPTR 程序存储器(TBLPTR) → TABLAT
000C	TBLWT *		写表 （TBLPTR 不变）	TABLAT →程序存储器(TBLPTR)
000D	TBLWT *+		写表 （TBLPTR 后加 1）	TABLAT →程序存储器(TBLPTR) TBLPTR +1 → TBLPTR
000E	TBLWT *-		写表 （TBLPTR 后减 1）	TABLAT →程序存储器(TBLPTR) TBLPTR -1 → TBLPTR
000F	TBLWT +*		写表 （TBLPTR 先加 1）	TBLPTR +1 → TBLPTR TABLAT →程序存储器(TBLPTR)

表 B-19：增强型 16-BIT 核 CPU 控制指令：

Hex 码	助记符号		功能描述	功能示意
E2 nn	BC	nn	若 C=1 则相对跳转	若：C=1, PC+2+2*nv→ PC, 否则： PC+2→PC
E6 nn	BN	nn	若 N=1 则相对跳转	若： N=1, PC+2+2*nv→PC, 否则： PC+2→PC
E3 nn	BNC	nn	若 C=0 则相对跳转	若： C=0, PC+2+2*nv→PC, 否则： PC+2→PC
E7 nn	BNN	nn	若 N=0 则相对跳转	若： N=0, PC+2+2*nv→PC, 否则： PC+2→PC
E5 nn	BNOV	nn	若 OV=0 则相对跳转	若： OV=0, PC+2+2*nv→PC, 否则： PC+2→PC
E1 nn	BNZ	nn	若 Z=0 则相对跳转	若： Z=0, PC+2+2*nv→PC, 否则： PC+2→PC
E4 nn	BOV	nn	若 OV=1 则相对跳转	若： OV=1, PC+2+2*nv→PC,

				否则： PC+2→PC
E0 nd	BRA	nd	UNCONDITIONAL 分支	PC+2+2*nv→ PC
E0 nn	BZ	nn	若 Z=1 则相对跳转	若： Z=1, PC+2+2*nv→PC, 否则： PC+2→PC
EC ml F mm	CALL	mm:ml,s	子程序绝对调用（第二个字）	PC+4 → TOS, mm:ml → PC<20:1>, 若： s=1, W → WS, STATUS → STATUSS, BSR → BSRS
EF ml F mm	GOTO	mm:ml	绝对跳转（第二个字）	mm:ml → PC<20:1>
0000	NOP		空操作	无操作
0006	POP		堆栈弹出	TOS-1 → TOS
0005	PUSH		压入堆栈	PC +2→ TOS
D8 nd	RCALL	nd	子程序相对调用	PC+2 → TOS, PC+2+2*nδ→PC
00FF	RESET		产生 RESET 复位信号	和 MCLR 复位功能一样
0010	RETFIE	s	中断返回（被允许的中断）	TOS→ PC,1→ GIE/GI 或 PEIE/GIEL, TOS→ PC,1→ GIE/GIEL, PEIE/GIEL 若 s=1, WS→ W, STATUSS → STATUS, BSRS→BSR,PCLATU/PCLATH 不变
0C kk	RETLW	kk	子程序返回，W 带回常量	kk → W,
0012	RETURN	s	子程序返回	TOS → PC, 若： s=1, WS → W, STATUSS→ STATUS, BSRS → BSR, PCLATU/PCLATH 不变
0003	SLEEP		进入睡眠状态	0 → WDT, 0 → WDT 预分频器, 1 → TO, 0 → PD

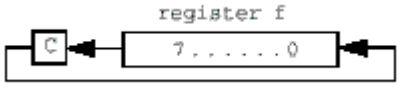
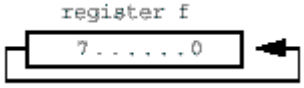
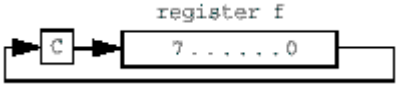
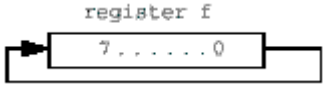
表 B-20：增强型 16-BIT 核位运算指令：

Hex 码	助记符号		功能描述	功能示意
9bf	BCF	f,b,a	清零 f 的第 b 位	0 → f(b)
8bf	BSF	f,b,a	置位 f 的第 b 位	1 → f(b)
Bbf	BTFSC	f,b,a	若 f 的第 b 位为 0，则跳过下一条语句	f(b) = 0 跳过下条语句
Abf	BTFSS	f,b,a	若 f 的第 b 位为 1，则跳过下一条语句	f(b) = 1 跳过下条语句
7bf	BTG	f,b,a	取反 f 的第 b 位	.NOT.f(b) → f(b)

表 B-21：增强型 16-BIT 核文件寄存器运算指令：

Hex 码	助记符号		功能描述	功能示意
24 f	ADDWF	f,d,a	将 W 加到 f 寄存器	W+f → dest
20 f	ADDWFC	f,d,a	将 W 和 C 加到 f 寄存器	W+f+C → dest
14 f	ANDWF	f,d,a	W 和 f 寄存器相与	W .AND. f → dest
6A f	CLRF	f,a	f 寄存器清零	0 → f
1C f	COMF	f,d,a	将 f 寄存器取反	f → dest
62 f	CPFSEQ	f,a	W 与 f 比较 若 W=f,跳过下一条语句	f–W, 若： f=W, PC+4 → PC 否则： PC+2 → PC
64 f	CPFSGT	f,a	W 与 f 比较 若 f>W,跳过下一条语句	f–W, 若： f > W, PC+4 → PC 否则： PC+2 → PC
60 f	CPFSLT	f,a	W 与 f 比较 若 W<f,跳过下一条语句	f–W, 若： f < W, PC+4 → PC 否则： PC+2 → PC
04 f	DECF	f,d,a	f 寄存器减一	f–1 → dest
2C f	DECFSZ	f,d,a	f 寄存器减一 若 f=0,跳过下一条语句	f–1→ dest,若dest=0, PC+4 → PC 否则： PC+2 → PC

4C f	DCFSNZ	f,d,a	f 寄存器减一 若 f≠ 0,跳过下一条语句	f–1→ dest, 若dest ≠ 0, PC+4 → PC 否则： PC+2 → PC
28 f	INCF	f,d,a	f 寄存器加一	f+1 → dest
3C f	INCFSZ	f,d,a	f 寄存器加一 若 f=0,跳过下一条语句	f+1→ dest, 若dest=0, PC+4 → PC 否则： PC+2 → PC
48 f	INFSNZ	f,d,a	f 寄存器加一 若 f≠ 0,跳过下一条语句	f+1→ dest, 若dest ≠ 0, PC+4 → PC 否则： PC+2 → PC
10 f	IORWF	f,d,a	W 和 f 寄存器同或	W .O**R.** f → dest
50 f	MOVF	f,d,a	传送 f	f → dest
C fs F fd	MOVFF	fs,fd	传送 fs 到 fd（第二个字）	fs → fd
02 f	MULWF	f,a	W 和 f 寄存器相乘	W → f
6C f	NEGF	f,a	将 f 寄存器取反	W * f → PRODH:PRODL
48 f	SETF	f,a	f 寄存器置位	0xFF → f
54 f	SUBFWB	f,d,a	从 W 中减去 f 和 C	W–f–C → dest
5C f	SUBWF	f,d,a	从 f 中减去 W	f–W → dest
58 f	SUBWFB	f,d,a	从 f 中减去 W 和 C	f–W–C → dest
38 f	SWAPF	f,d,a	f 高低半字节交换	f<3:0>→ dest<7:4>,f<7:4> → dest<3:0>
66 f	TSTFSZ	f,a	若 f 寄存器为 0，则跳过下一条指令	PC+4 → PC, 若： f=0, 否则： PC+2 → PC
18 f	XORWF	f,d,a	将 W 和 f 寄存器异或	W .XOR. f → dest

34 f	RLCF	f,d,a	穿越进位的左移	
44 f	RLNCF	f,d,a	不穿越进位的左移	
30 f	RRCF	f,d,a	穿越进位的右移	
40 f	RRNCF	f,d,a	不穿越进位的右移	

B10 16 进制到 10 进制转换

字节				字节			
16 进制	10 进制	16 进制	10 进制	16 进制	10 进制	16 进制	10 进制
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

使用本表方法：每个 16 进制码均可找到相应的 10 进制数值。把这些数放在一起可方便查找。例如：16 进制数 A38F 按表转化为 41871。

千位		百位		十位		个位		结果
40960	+	768	+	128	+	15	=	41871
（16 进制）								（10 进制）

B11 ASCII 代码集:

ASCII 码的低位	ASCII 码的高位								
	HEX	0	1	2	3	4	5	6	7
	0	NUL	DLE	Space	0	@	P	‘	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	.	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	Bell	ETB	’	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	|
	D	CR	GS	-	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

附录 C　MPASM 错误/警告/信息

C.1 概述

下列信息由 MPASM 生成。这些信息出现在列表文件之中，每一出错行的正上方。

如果不指定 MPASM 选项，那么信息存储在出错文件（. ERR）中。如果使用/e 选项（关闭出错文件）。那么信息将不出现将在屏幕上。假如/q（静止模式）选项与/e 一起使用，那么信息将不出现在屏幕或出错文件中。信息仍然会出现在列表文件内。

C.2 重点

本附录内容包括：

- 错误
- 警告
- 信息

C.3 错误

101 **ERROR**

错误

用户错误，由 ERROR 指示符调用。

102 **Out of memory**

存储器用尽

当系统在处理宏，#define 或内部处理时没有足够的存储器。去掉所有驻留（TSR），

关闭任何打开的应用，并试图再汇编文件。如果使用 DOS 可执行的实模式时产生这种错误，可试一下使用 Windows 版本（MPASMWIN）或 DPMI 版本（MPASM_DP）。

103 Symboltable full

符号表满

没有足够的存储器可供符号表使用。消除去掉所有驻留（TSR），关闭任何打开的应用，并再次汇编文件。如果使用 DOS 可执行的实模式时产生这种错误，可试用 Windows 版本（MPASMWIN）或 DPMI 版本（MPASM_DP）。

104 Temp file creation error

无法建立临时文件

不能建立临时文件，请检查可用的磁盘空间。

105 Conont open file

无法打开文件

不能打开文件，如果它是源文件，那么文件可能不存在。如果它是输出文件，可能是旧版本有写保护。

106 String substitution too complex

字符串替换太复杂

#defines 的嵌套太多。

107 Illegal digit

非法数字

一个数（NUMBER）中的非法数字（DIGIT）。二进制的有效数字是 0-1，八进制的有效数字是 0-7，十进制的有效数字是 0-9，十六进制的有效数字是 0-9，a-f 和 A-F。

108 Illegal chatacter

非法字符

一个标号（标号）中的非法字符（CHARACTER）。标号的有效字符是（a..f，A..F），数字（0-9），下划线（-）以及问号（？），标号的第一个字符不能是数字。

109 Unmatched “(”

左括号不匹配

左括号没有相匹配的右括号。例如：“DATA（1+2”。

110 Unmatched “)”

右括号不匹配

右括号没有相匹配的左括号。例如：“DATA 1+2)”。

111 MISSING SYMBOL

丢失符号

EQU 或 SET 语句没有赋值的标号。

112 MISSING OPERATOR

丢失运算符

表达式遗漏了运算符。例如：DATA 1 2。

113 Symbol not previously defined

符号未预先定义

引用尚未定义的符号，只有地址可用作向前的引用。常数和变量必须在使用之前声明。

114 Divide by zero

被零除

在表达式求值中遇到除数为零。

115 Duplicate label

多余标号

在多余一个的位置上把标号声明为常量。

116 Address label duplicated or different in second pass

重复地址标号或第二次扫描时地址标号不一致

在两处使用了相同的标号。或者，标号只使用一次但在第 2 遍时对不同的位置求值。这常发生在用户试图写页位（page-bit）设置宏，它根据目标产生不同数目的指令。

117 Address wrapped around 0

地址回复到零

位置计数器只能计数到 FFFF。在此之后，它将回至 0。

118 Overwriting previous address contents

企图覆盖预先定义的地址常量

对于该地址，已经预先产生了代码。

119 Code too fragmented

代码分段太多

代码分为太多的片段。这种错误是非常罕见的，且仅发生在引用地址超过 32K（包括配置位 configuration bits）的源代码中。

120 Call or jump not allowed at this address

该地址不允许调用或跳转

此地址不能进行调用或跳转。例如：PIC165X 系列的 CALL 目录必须在页的下半部。

121 Illegal label

非法标号

在某些指示符行（directive line）不允许标号。可简单地把标号放在它本身行上，在指示符之上。此外，HIGH，LOW，PAGE 和 BANK 不允许作为标号。

122 Illegal opcode

非法操作代码

标号（Token）不是合法的操作码。

123 Illegal directive

非法指示符

对于所选择的处理器，指示符是不允许的。例如 PIC17C42 的_ _IDLOCS 指示符。

124 Illegal argument

非法参数

非法的指示符参数。例如“LIST STUPID”。

125 Illegal condition

非法条件

错误的条件汇编。例如，未匹配的 ENDIF。

126 Argument out of range

参数超出范围

操作码或指示符参数超出有效范围。例如“TRIS10”。

127 Too many arguments

参数太多

给宏调用的参数太多。

128 Missing argument(s)

丢失参数

宏调用或操作码没有赋予足够的参数。

129 Expected

预期

预期某些类型参数。将提供预期的表（expected list）。

130 Processor type previously defined

处理器类型已经定义

将选择不同系列的处理器。

131 Processor type is undefined

处理器类型未定义

在定义处理器之前产生代码。注意，直至处理器被定义为止，操作码集是未知的。

132 Unknown processor

非法处理器类型

所选的处理器不是有效的处理器。

133 Hex file format INHX32 required

需要指明十六进制文件格式

规定 32K 以上的地址。例如，在 PIC17CXX 系列上设置了配置位。

134 Illegal hex file format.

非法十六进制文件格式

在 LIST 指示符中规定了非法的十六进制文件。

135 Macro name missing

丢失宏名

定义了没有名字的宏。

136 Duplicate macro name

重复宏名

宏名是双重的。

137 Macros nested too deep

宏嵌套太深

超出宏嵌套的最大层数。

138 Include files nested too deep

包含文件嵌套太深

超出包含文件嵌套的最大层数。

139 Maximum of 100 lines inside WHILE-ENDW

WHILE-ENDW 里只能有 100 行

WHILE-ENDW 最多能包含 100 行。

140 WHILE must terminate within 256 iterations

WHILE 必须在 256 次迭代后结束

WHILE-ENDW 循环必须在 256 次迭代之内结束，这是为了防止无限的汇编。

141 WHILEs nested too deep

WHILE 嵌套太深

超过 WHILE-ENDW 的最大嵌套层数。

142 IFs nested too deep

IF 嵌套太深

超出 IF 嵌套的最大层数。

143 Illegal nesting

非法嵌套

宏 IF 以及 WHILE 必须完整地嵌套；它们不能交错。如果在 WHILE 循环中有 IF，那么在 ENDW 之前必须有 ENDIF。

144 Unmatched ENDC

ENDC 不匹配

有 ENDC 而没有 CBLOCK 出现。

145 Unmatched ENDM

ENDM 不匹配

出现 ENDM，而没有 MACRO 定义。

146 Unmatched EXITM

EXITM 不匹配

没有宏定义而出现 EXITM。

147 Directive not allowed when generating an object file

生成目标文件时不允许指示符

如果要产生目标文件，那么 ORG 指示符是不允许的。如果需要的话，应该建立数据或代码区段，指定地址。

148 Expanded source line exceeded 200 characters

替换之后的源程序行最大长度超过了 200 个字符

在#DEFINE 和宏参数替换之后的源程序行的最大长度是 200 个字符。注意，#DEFINE 替换不包括注释，但宏参数替换包括注释。

149 Directive only allowed when generating an object file section

生成目标文件区段时才允许指示符

一些指示符，比如 GLOBAL 和 EXTERN，只有当目标文件生成时才有意义。当生成绝对代码时，它们不能使用。

150 Labels must be defined in a code or data section when making an object file

当生成目标文件时，必须在代码或数据区段里定义标号

当生成目标文件时，所有数据和代码地址标号必须在一个数据或代码区段内定义。符号由EQU定义，SET指示符在区段外定义。

151 Operand contains unresolvable labels or is too complex

操作数里包含不确定的标号或者操作数太复杂

当生成目标文件时，操作数必须以这种格式书写：

[高 | 低] ([<可重定位地址标号>]+[<偏移量>])

152 Executable code and data must be defined in an appropriate section

可执行代码、数据必须在相应的区段里定义

当生成目标文件时，可执行代码及数据的申明必须放在相应的区段里。

153 Page or Bank bits cannot be evaluated for the operand

无法求出操作数的页和堆设置位

指示符PAGESEL、BANKSEL、BANKISEL的操作数要按这样的格式书写：

<可重定位地址标号> 或 <常量>。

154 Each object file section must be continuous

每个目标区段必须连续

除了UDATA_OVR区段，目标文件区段在单个源文件中不能被停止和重启动。为解决问题，可以给每个区段取一个名，或者去掉代码和数据申明以便每个区段是相邻的。如果两个不同性质的区段被赋予同一个名称时，也会产生该错误。

155 All overlaid sections of the same name must have the same starting address

所有名称相同的覆盖区段必须有相同的起始地址

假如多个UDATA_OVR区段申明了相同的名称，它们必须有相同的起始地址。

156 Operand must be an address label

操作数必须是一个地址标号。

当生成目标文件时候，只有代码或数据区段的地址标号可以申明为全局（global）。用SET 或EQU申明的变量不能输出（export）到其他区段。

157 UNKNOWN ERROR

未知错误

遇到了一个MPASM无法理解的错误。在本附录的错误集中没有这种错误。如果你无法找出错误原因，请联系MICROCHIP的现场应用工程师（FAE）。

C.4 Warnings

201 Symbol not previously defined

符号没有预先定义

#undefined（未定义）的符号先前未定义过。

202 Argument out of range. Least significant bits used

参数超出范围。使用分配的最小位

分配的空间容纳不下参数。例如，文字必须为 8 位。

203 Found opcode in column 1

在第一列发现操作码

在第 1 列发现操作码，该列是保留给符号的。

204 Found pseudo-op in column 1

在第一列发现伪操作码

在第 1 列发现伪操作码，该列是保留给符号的。

205 Found directive in column 1

在第一列发现指示符

在第 1 列发现指示符，该列是保留给符号的。

206 Found call to macro in column 1

在第一列发现宏调用

在第 1 列发现宏调用，该列是保留给符号的。

207 Found label after column 1

在第一列以后发现标号

在第 1 列以后发现标号。这常常是拼写错误的操作码引起的。

208 Label truncated at 32 characters

标号在 32 个字符以后被截断

最大的符号长度是 32 个字符。

209 Missing quote

丢失引号

文字字符串或字符遗漏引号。

210 Extra)

多余的右括号

在行的末尾发现额外的逗号。

211 Extraneous arguments on the line

行内发现多余的参数

在行内发现多余的参数。应当调查这些警告，这是语法分析以不同的方式解释某些内容。

212 Expected

期望某些类型的参数

期望某些类型的参数。应当提供描述。对于警告，作有关参数的假设。

213 The EXTERN directive should only be used when making a .O file

只有在生成“a .O”文件时才能使用 EXTERN 指示符

只有创建目录文件时“EXTERN”指示符才有意义。

214 Unmatched (

不匹配的左括号（

发现不匹配的括号。如果括号对于指示求值的次序无用，那么使用此警告。

215 Processor superseded by command line. Verify processor symbol

在命令行上规定的处理器类型具有较高优先级。检查处理器类型

在命令行上与源文件内规定处理器。命令行具有较高优先级。

216 Radix superseded by command line.

在命令行上规定的基数类型具有较高优先级。

在命令行上与源文件内规定了基数。命令行具有较高优先级。

217 Hex file format specified on command line.

在命令行上规定的十六进制文件格式具有较高优先级。

在命令行上与源文件内规定了 HEX 文件格式。命令行具有较高优先级。

218 Expected DEC, OCT, HEX. Will use HEX.

缺少基数定义，将缺省为 HEX 格式

错误的基数规格。

219 Invalid RAM location specified.

非法 RAM 位置指定

如果使用_MAXRAM 和_BADRAM 指示符，那么此警告标志着任何被这些指示符声明为无效的 RAM 位置。请注意所提供的标题文件包含使用于每个处理器

220 Address exceeds maximum range for this processor.

地址超出处理器的最大范围。

规定了超出处理器的存储器大小的 ROM。

221 Invalid message number.

非法信息号

规定的显示或隐藏信息号不是有效的信息号。

222 Error messages cannot be disabled.

无法屏蔽错误信息

出错信息不能用 ERRORLEVEL 命令禁止。

223 Redefining processor

重新选择所选的处理器

将用 LIST 或 PROCESSOR 指示符重新选择所选的处理器。

224 Use of this instruction is not recommended.

不推荐使用该指令

对于 PIC16CXX 器件不推荐使用 TRIS 和 OPTION 指令。

225 Invalid label in operand.

操作数内含有非法标号

操作数非法。例如：试图用 CALL 调用一个宏名。

226 UNKNOWN WARNING

未知警告

遇到 MPASM 不能理解的警告。这不是在本附录内所述的任何警告。如果你不能找出产生警告的原因，那么请与 Microchip 的现场应用工程师（FAE）联系。

C.5 信息

301 MESSAGE:

用户信息，用 MESSG 指示符调用。

302 Register in operand not in bank 0. Ensure that bank bits are correct.

操作数里的寄存器不位于第0堆。检查堆设置位是否正确。

寄存器地址被赋予一个数值，该数值还包含寄存器堆选择位（bank bits）。例如：PIC16CXX 的 RAM 位置由指令里的 7 个 BIT 和 1-2BIT 的寄存器堆选择位组成。

303 Program word too large. Truncated to core size.

程序字太宽。将截断为芯片允许值。

PIC16C5X 的程序字只允许 12 位；PIC16CXX 的程序字允许 14 位。

304 ID Locations value too large. Last four hex digits used.

ID 位的数值太大。使用最后四位十六进制数字

对于 ID 位置只允许 4 个十六进制数字。

305 Using default destination of 1 (file).

使用缺省值（？）

如果不规定目标位，那么使用缺省值。

306 Crossing page boundary – ensure page bits are set.

超越页边界 – 确信页选择位已经设置

产生的代码越过页边界。

307 Setting page bits.

设置页选择位

将用 LCALL 或 LGOTO 伪指令（pseudo-op）设置页位。

308 Warning level superseded by command line value.

命令行上的警告级别具有较高优先级

在命令行上与源文件内规定警告级。命令行具有较高优先级。

309 Macro expansion superseded by command line.

命令行上的宏扩展具有较高优先级

在命令行上与源文件内规定了宏扩展，命令行具有较高优先级。

310 Superseding current maximum RAM and RAM map.

存在优先于当前的最大 RAM 和 RAM 映射

先前已经使用了_ _MAXRAM 指示符。

312 Page or Bank selection not needed for this device. No code generated.

本芯片不必使用页和堆选择。不灰产生任何代码。

假如MCU只有一个ROM页或一个RAM堆，则无需页或堆选择。任何PAGESEL、BANKSEL或BANKISEL指示符将不生成任何代码。

313 CBLOCK constants will start with a value of 0.

CBLOCK 常量将以数值 0 起始

如果源文件里第一个 CBLOCK 没有指定起始值，则会输出这条信息。

314 UNKNOWN MESSAGE

未知信息

遇到 MPASM 不能理解的信息。这不是在本附录内所述的任何信息。如果你不能找出产生错误的原因，那么请与 Micuochip 的现场应用工程师（FAE）联系。

附录 D MPLINK 错误/警告

D1 介绍

MPLINK 可能生成下列错误和警告。

D2 重点

本附录的内容包括：

- 分析错误
- 链接错误
- 链接警告
- 库文件错误
- COFF 文件错误
- COFF 到 COD 转换错误
- COFF 到 COD 转换警告

D3 分析错误

- **Invalid attributes for memory in ‘cmdfile:line’.**
  CODEPAGE、DATABANK 或 SHAREBANK 指示符并不指定NAME、START 或 END属性；或者指定了另外一个非法的属性。
- **Invalid attributes for STACK in ‘cmdfile:line’.**
  STACK 指示符并不指定“SIZE”属性；或者指定了另外一个非法的属性。
- **Invalid attributes for SECTION in ‘cmdfile:line’.**

SECTION 指示符需要有 NAME、RAM 或 ROM 属性。

- **Could not open ‘cmdfile’.**
  不能打开链接器控制文件。检查该文件是否存在于当前搜寻目录里，而且是可读的。
- **Multiple inclusion of linker command file ‘cmdfile’.**
  链接器控制文件只能包含一次。从参考链接器控制文件里将多余的 INCLUDE 指示符移走。
- **Illegal <libpath> for LIBPATH in ‘cmdfile:line’.**
  ‘libpath’ 必须是用分号分隔的目录列表。如果一个附加目录名中有空格分隔，则要用双引号将该目录引起来。
- **Illegal <lkrpath> for LKRPATH in ‘cmdfile:line’.**
  ‘lkrpath' 必须是用分号分隔的目录列表。如果一个附加目录名中有空格分隔，则要用双引号将该目录引起来。
- **Illegal <filename> for FILES in ‘cmdfile:line’.**
  目标文件和库藏文件必须分别用‘.o’ 或 ‘.lib’结尾。
- **Illegal <filename> for INCLUDE in ‘cmdfile:line’.**
  链接器控制文件必须分别用‘.lkr’ 结尾。
- **Unrecognized input in ‘cmdfile:line’.**
  所有链接器控制文件里的注释都必须以一个指示关键字（directive keyword）或“//”开始。
- **-o switch requires <filename>.**
  COFF 输出文件名必须指定。例如：-o main.out。
- **-m switch requires <filename>.**
  映射文件名必须指定。例如：-m main.map
- **-n switch requires <length>.**
  列表每页的行数必须指定。一个为 0 的长度值将会超过列表文件的页面设置。
- **-L switch requires <pathlist>.**
  必须指定由分号分隔的目录路径。如果一个目录名中有空格分隔，则要用双引号将该目录引起来。例如：-L ..;c:\mplab\lib;"c:\program files\mplink"
- **-K switch requires <pathlist>.**
  必须是用分号分隔的目录列表。如果一个附加目录名中有空格分隔，则要用双引号将该目录引起来。例如：-L ..;c:\mplab\lib;"c:\program files\mplink"
- **unknown switch: ’cmdline token’.**
  使用了一个无法识别的命令行开关。参考介绍 MICROCHIP 支持的开关的相关资料。

D4 链接错误

- **Memory ‘memName’ overlaps memory ‘memName’.**
  所有的 CODEPAGE 都必须指定唯一的互不重合存储器地址范围。同样，DATABANK 和 SHAREBANK 块也不能有重合的地址。
- **Duplicate definition of memory ‘memName’.**
  所有CODEPAGE 和DATABANK指示符必须有唯一的NAME（名称）属性。
- **Multiple map files declared: ‘File1’, ‘File2’.**
  指定了一次以上的–m <mapfile>开关。

- **Multiple output files declared: ‘File1’, ‘File2’.**
  指定了一次以上的–o <outfile>开关。
- **Multiple inclusion of object file ‘File1’, ‘File2’.**
  在命令行或者在链接器控制文件里用 FILES 指示符将同一个目标文件包含了多次。将多余的参考项移走。
- **Overlapping definitions of SHAREBANK ‘memName’.**
  用 SHAREBANK 指示符分配的地址空间和前面分配的地址空间有覆盖的区域。覆盖是不允许的。
- **Inconsistent length definitions of SHAREBANK ‘memName’.**
  所有具有相同 NAME 属性的 SHAREBANK 定义必须有相同的长度。
- **Multiple STACK definitions.**
  STACK 指示符在链接器命令文件或者包含的链接器命令文件里出现了多次。将多余的 STACK 指示符移走。
  为 STACK 重新定义：DATABANK/SHAREBANK ‘memName’。
- **Duplicate definitions of SECTION ‘secName’.**
  每个 SECTION 指示符必须有唯一的 NAME 属性。将多余的指定移走。
- **Undefined CODEPAGE ‘memName’ for SECTION ‘secName’.**
  SECTION 指示符所指的 ROM 存储器块没有定义。在链接器控制文件里为未定义存储器块加入 CODEPAGE 指示符。
- **Undefined DATABANK/SHAREBANK ‘memName’ for SECTION ‘secName’.**
  SECTION指示符所指的RAM存储器块没有定义。在链接器控制文件里为未定义存储器块加入DATABANK 或 SHAREBANK指示符。
- **No input object files specified.**
  在命令行或在链接器命令文件里使用FILES指示符指定至少一个目标模块。
- **Could not find file ‘File’.**
  一个指定的库文件或目标文件不存在。或者在链接目录里未找到该文件。
- **Processor types do not agree across all input files.**
  每个目标模块和库文件都必须指定处理器类型或处理器所属系列。所有输入模块的处理器类型或处理器所属系列都必须匹配。
- **ROM width of ‘xx’ not supported.**
  一个输入模块指定了一个ROM宽度不是12、14或16bit。
- **Unknown section type for ‘secName’ in file ‘File’.**
  一个输入目标模块或库文件的文件类型错误或者文件已损坏。
- **Section types for ‘secName’ do not match across input files.**
  名为’secName’的区段在一个以上的输入文件中出现。所有拥有该区段的文件必 须有相同的区段类型。
- **Section ‘secName’ is absolute but occurs in more than one input file.**
  名为’secName’的绝对区段只能出现在一个输入文件里。有相同名称的可重定位区段可以在多个输入文件中出现。要么将源文件中的多余绝对区段去掉，要么使用可重定位区段。
- **Section share types for ‘secName’ do not match across input files.**
  名为’secName’的区段出现在一个以上的输入文件里，然而，一些区段被标记为共享区段，而一些却不是。在源文件里改变区段的共享类型并重新建立目标

模块。

- **Section ‘secName’ contains code and can not have a ‘RAM’ memory attribute specified in the linker command file.**
  在链接器命令文件里定义该区段时只能用ROM属性。
- **Section ‘secName’ contains uninitialized data and can not have a ‘ROM’ memory attribute specified in the linker command file.**
  在链接器命令文件里定义该区段时只能用RAM属性。
- **Section ‘secName’ contains initialized data and can not have a ‘ROM’ memory attribute specified in the linker command file.**
  在链接器命令文件里定义该区段时只能用RAM属性。
- **Section ‘secName’ contains initialized rom data and can not have a ‘RAM’ memory attribute specified in the linker command file.**
  在链接器命令文件里定义该区段时只能用ROM属性。
- **Section ‘secName’ has a memory ‘memName’ which can not fit the section. Section ‘secName’ length=’0xHHHH’.**
  在链接器命令文件里分配给一个区段的存储器要么没有足够的空间容纳该区段，要么将会覆盖其他区段。使用 –m<mapfile>开关来生成一个错误映射文件。错误映射文件将会显示位于该错误之前的那些区段。
- **Section ‘secName’ has a memory ‘memName’' which is not defined in the linker command file.**
  在链接器命令文件里使用CODEPAGE、 DATABANK 或 SHAREBANK 指示符来定义未定义存储器（undefined memory）。
- **Section ‘secName’ can not fit the section. Section ‘secName’ length='0xHHHH'.**
  在链接器命令文件里没有指定存储器的区段不能被定位。使用–m<mapfile>开关来生成一个错误映射文件。错误映射文件将会显示位于该错误之前的那些区段。必须使用CODEPAGE、 DATABANK 或 SHAREBANK 指示符来保证更多可用的存储器；或者去掉“PROTECTED”属性；或者减少输入区段的数量。
- **Section ‘secName’ has a memory ‘memName’ which can not fit the absolute section. Section 'secName' start=0xHHHH, length=0xHHHH.**
  在链接器命令文件里分配给一个区段的存储器要么没有足够的空间容纳该区段，要么将会覆盖其他区段。使用 –m<mapfile>开关来生成一个错误映射文件。错误映射文件将会显示位于该错误之前的那些区段。
- **Section ‘secName’ can not fit the absolute section. Section ‘secName’ start=0xHHHH, length=0xHHHH.**
  在链接器命令文件里没有指定存储器的区段不能被定位。使用–m<mapfile>开关来生成一个错误映射文件。错误映射文件将会显示位于该错误之前的那些区段。必须使用CODEPAGE、 DATABANK 或 SHAREBANK 指示符来保证更多可用的存储器；或者去掉“PROTECTED”属性；或者减少输入区段的数量。

- **Symbol ‘symName’ has multiple definitions.**
  某一符号只能在一个输入模块里被定义。
- **Could not resolve symbol ‘symName’ in file ‘File’.**
  符号‘symName’是一个外部参量（external reference），没有输入模块定义过该

符号。假如在库模块里定义该符号，确信该库模块已被包含在命令行参数里或者链接器命令文件里使用了FILE指示符来指定。

- **Could not open map file ‘File’ for writing.**
  检查文件‘File’是否存在，或者该文件是只读属性。
- **Symbol ‘symName’ out of range of relative branch instruction.**
  相对跳转指令用‘symName’作为它的跳转目标。但是在第二次对‘symName’的偏移量进行编码的时候，发现该偏移量不符合指令宽度的允许范围，而指令的宽度决定了分支指令的目标地址跳转范围。
- **Symbol ‘symName’ is not word-aligned.**
  不能用作branch 、call 或 goto指令的目标符号。这些指令位于奇数地址，但指令译码不能识别不是按照字排列（word-aligned）方式安排的地址。
- **{PCL | TOSH | TOSU | TOSL} cannot be used as the destination of a MOVFF instruction.**
  当MOVFF指令的目的寄存器是PCL、TOSH、TOSU 或 TOSL时会有不可预料的结果。MPLINK不允许MOVFF指令的目的寄存器是这些地址（PCL、TOSH、TOSU 或 TOSL）。
- **Absolute code section ‘secName’ must start at a word-aligned address.**
  程序代码区段只能放置于按照字排列（word-aligned）方式规划的地址上。假如绝对代码区段不是安排在按这种方式规划的地址上，MPLINK将会给出错误信息。

D5 链接警告

- **Fill pattern for memory ‘memName’ doesn’t divide evenly into unused section locations.**
  最后一个值被截断。假如 ROM 区段被指定了填充模式（fill pattern），但这个区段余下的空间却不是填充模式指定大小的偶数倍。这个错误将会出现以警告不完整的模式设置。

D6 库文件错误信息

- **Symbol ‘name’ has multiple external definitions.**
  库文件里一个符号只能定义一次。
- **Could not open library file ‘filename’ for reading.**
  检查文件名为‘filename’的文件是否存在和可读。
- **Could not read archive magic string in library file ‘filename’.**
  文件不是合法的库文件，或该文件已经损坏。
- **File ‘filename’ is not a valid library file.**
  库文件的扩展名必须为：‘.lib’。
- **File ‘filename’ is not a valid library file.**
  文件不是合法的目标文件，或该文件已经损坏。

- **Could not build member ‘memberName’ in library file ‘filename’.**
  文件不是合法的库文件，或该文件已经损坏。
- **Could not open library file ‘filename’ for writing.**

检查文件名为‘filename’的文件是否存在和可读。

- **Could not write archive magic string in library file ‘filename’.**
  该文件可能已经损坏。
- **Could not write member header for ‘memberName’ in library file ‘filename’.**
  该文件可能已经损坏。
- **‘memberName’ is not a member of ‘filename’.**

除非是库的成员之一，‘memberName’不能从库文件里提取（extract）或删除。

D7 COFF 文件错误

- **Unable to find section name in string table.**
  字符串表中找不到区段名。
- **Unable to find symbol name in string table.**
  字符串表中找不到符号名。
- **Unable to find aux_file name in string table.**
  字符串表中找不到辅助文件名。
- **Could not find section name ‘secName’ in string table.**
  字符串表中找不到名为‘secName’的区段名。
- **Could not find symbol name ‘symName’ in string table.**
  字符串表中找不到名为‘symName’的区段名。
- **Coff file ‘filename’ symbol[‘xx’] has an invalid n_scnum.**
  名为‘filename’的COFF文件符号[‘xx’]有非法的n_scnum（？）。
- **Coff file ‘filename’ symbol[‘xx’] has an invalid n_offset.**
  名为‘filename’的COFF文件符号[‘xx’]有非法的n_ offset（？）。
- **Coff file ‘filename’ section[‘xx’] has an invalid s_offset.**
  名为‘filename’的COFF文件区段[‘xx’]有非法的s_ offset（？）。
- **Coff file ‘filename’ has relocation entries but an empty symbol table.**
  名为‘filename’的COFF文件有可重定位路径，但符号表是空的。
- **Coff file ‘filename’, section ‘secName’ reloc[‘xx’] has an invalid r_symndx.**
  名为‘filename’的COFF文件中，‘secName’区段reloc [‘xx’]有一个非法r_symndx。
- **Coff file ‘filename’, symbol['xx'] has an invalid x_tagndx or x_endndx.**
  名为‘filename’的COFF文件，符号[‘xx’]有一个非法tagndx 或 x_endndx。
- **Coff file ‘filename’, section ‘secName’ line[‘xx’] has an invalid l_srcndx.**
  名为‘filename’的COFF文件中，‘secName’区段第[‘xx’]行有一个非法l_scndx。
- **Coff file ‘filename’, section ‘secName’ line[‘xx’] has an invalid l_fcnndx.**
  名为‘filename’的COFF文件中，‘secName’区段第[‘xx’]行有一个非法l_fcnndx。
- **Coff file ‘filename’, cScnHdr.size() != cScnNum.size().**
  名为‘filename’的COFF文件中，参数cScnHdr.size() 与 cScnNum.size()不等。
- **Could not open Coff file ‘filename’ for reading.**
  名为‘filename’的COFF文件不能打开以便读。
- **Coff file ‘filename’ could not read file header.**
  名为‘filename’的COFF文件不能读文件头。
- **Coff file ‘filename’ could not read optional file header.**
  名为‘filename’的COFF文件不能读可选文件头。
- **Coff file ‘filename’ missing optional file header.**

名为‘filename’的COFF文件没有可选文件头。

- **Coff file ‘filename’ could not read string table length.**
  名为‘filename’的COFF文件不能读字符串表长度。
- **Coff file ‘filename’ could not read string table.**
  名为‘filename’的COFF文件不能读字符串表。
- **Coff file ‘filename’ could not read symbol table.**
  名为‘filename’的COFF文件不能读符号表。
- **Coff file ‘filename’ could not read section header.**
  名为‘filename’的COFF文件不能读区段头。
- **Coff file ‘filename’ could not read raw data.**
  名为‘filename’的COFF文件不能读原始数据。
- **Coff file ‘filename’ could not read line numbers.**
  名为‘filename’的COFF文件不能读行号。
- **Coff file ‘filename’ could not read relocation info.**
  名为‘filename’的COFF文件不读可重定位信息。
- **Could not open Coff file ‘filename’ for writing.**
  名为‘filename’的COFF文件不能打开以便写。
- **Coff file ‘filename’ could not write file header.**
  名为‘filename’的COFF文件不能写文件头。
- **Coff file ‘filename’ could not write optional file header.**
  名为‘filename’的COFF文件不能写可选文件头。
- **Coff file ‘filename’ could not write section header.**
  名为‘filename’的 COFF 文件不能写区段头。
- **Coff file ‘filename’ could not write raw data.**
  名为‘filename’的COFF文件不能写原始数据。
- **Coff file ‘filename’ could not write reloc.**
  名为‘filename’的COFF文件不能写可重定位信息。
- **Coff file ‘filename’ could not write lineinfo.**
  名为‘filename’的COFF文件不能写行信息。
- **Coff file ‘filename’ could not write symbol.**
  名为‘filename’的COFF文件不能写符号。
- **Coff file ‘filename’ could not write string table length.**
  名为‘filename’的COFF文件不能写字符串表长度。
- **Coff file ‘filename’ could not write string.**
  名为‘filename’的 COFF 文件不能写字符串。

D8 COFF 到 COD 转换错误

- **Coff file ‘filename’ must contain at least one ‘code’ or ‘romdata’ section.**
  为了将 COFF 文件转化为 COD 文件，COFF 文件必须有一个代码（code）区段或者（romdata）区段。
- **Could not open list file ‘filename’ for writing.**
  检查文件名为‘filename’的文件是否存在和可读。

D9 COFF 到 COD 转换警告

- **Could not open source file ‘filename’. This file will not be present in the list file.**

  给出的源文件不能被打开。假如在另一台目录结构不同的 PC 机上创建（built）一个输入目标或库模块时，可能出现这种现象。如果想在源代码级调试，请在当前 PC 机上重新重新创建（rebuild）。

附录 E MPLIB 出错信息

E1 介绍

MPLIB 检测以下错误的出处并给出报告。

E2 重点

本附录内容包括：

- 分析错误
- 库文件错误
- COFF 文件错误

E3 分析错误

- **invalid switch**

  非法开关

  指定了一个不支持的开关参数。参考 MICROCHIP 支持的命令行选项用法。

- **library filename is required**

  缺少库文件

  所有命令都要求有库文件名。所有库文件名都要有扩展名“.LIB”。

- **invalid object filename**

  非法目标文件名

  所有的目标文件都必须有扩展名“.O”。

E4 库文件错误

请参考 MPLINK 错误信息附录中的与库文件有关的信息。

E5 COFF 文件错误

请参考 MPLINK 错误信息附录中的与 COFF 文件有关的信息。

术语

介绍

为提供通用参考，本附录对本手册中涉及到的一些术语进行定义。

重点

本术语集包含对在MPASM、MPLINK和MPLIB等章节中涉及的术语进行解释。很多术语在这三部分都有使用，但在各个部分可能有其不同的含义。

术语

- **Absolute Section**
  有固定绝对地址的区段，MPLINK 无法改变其地址。
- **Alpha Character**
  字母表中所有包括大小写的字母（a,b,…,z,A,B…,Z）。
- **Alphanumeric**
  阿拉伯数字（0,1,…,9）
- **Application**
  用户开发的一系列软件和硬件，通常是用 PICmicro 控制的产品。
- **Assemble**
  用 MPASM 对源程序进行汇编，生成目标代码的过程。
- **Assembler Source Code**
  由汇编器处理的文本文件，使汇编指令和机器码之间一一对应。
- **Assigned Section**
  在链接器命令文件里被分配了目标存储器的区段。链接器指定一块已分 配区段到指定的存储器块里。
- **Build**

将应用系统的所有源文件重新编译。

- **COFF**(Common Object File Format)
  共用目标文件格式 - 一种目标/可执行文件的格式。用于 MPASM 或 MPLAB-C17/C18 生成的可重定位目标文件。
- **Command Line Interface**
  命令行界面用于执行带参数的程序。带命令行参数执行 MPASM 或者直接执行 MPASM 都将调用汇编器。如果缺少命令行参数，则会有输入界面提示用户输入相应的参数值。
- **Control Directives**
  控制指示符允许条件汇编代码。
- **Data Directives**
  数据指示符控制存储器的分配，提供数据和符号之间的对应，即：用具有含义的符号代表数据。
- **Data RAM**
  仿真 PICmicro 中的通用 RAM 文件存储器。文件寄存器窗口显示数据 RAM 的值。
- **Directives**
  指示符控制汇编器的操作过程，它指示 MPASM 如何处理助记符，定义数据，如何格式化输出列表文件。指示符使得代码更容易理解，而且为特殊的需求提供用户自定义的输出。
- **Expressions**
  表达式用于源文件中的操作数区域，它可以包含常量，符号，或者任何用算术运算符组合起来的常量和符号的组合。每个常量或符号都可以冠以一个正号或负号，以表示其正负性质。

  注意：表达式被用 32 位整型求值运算（目前尚不支持浮点运算）。
- **External Linkage**
  如果在其他模块定义了可以操作一个函数或变量，则称它具有外部连接的功能。
- **External Symbol**
  给有外部连接功能的变量或函数定义的符号。
- **External Symbol Definition**
  在当前模块定义的变量或函数的外部符号。
- **External Symbol Reference**
  一个外部符号，它指向一个在其他模块定义的函数或变量。
- **External Symbol Resolution**
  链接器的一个处理过程。在此过程中，来自所有输入模块的外部符号定义都试图去更新所有外部符号参考。任何没有得到更新的外部符号参考将引起链接错误报告。
- **Hex Code**
  可执行的 16 进制指令代码。是由源文件通过汇编器或编译器的处理生成的。16 进制代码可以直接转化为目标代码。它包含在一个 16 进制文件中。
- **Hex File**
  一个 ASCII 文件。它包含 16 进制地址和代码值，便于编程器使用。该格式的文件可以由编程器读出。
- **Identifier**

一个函数或变量的名称。

- **Initialized Data**
  数据被初始化为特定的值。在 C 语言里，`int myVar=5` 这条语句定义了一个变量，该变量将驻留在一个已初始化的区段里。
- **Internal Linkage**
  不能从其他模块访问一个在该模块定义的函数或变量，则称函数或变量内部可连接。
- **Library**
  库是一些可重定位的目标模块的集合。它由多个源文件经汇编生成目标文件，然后使用库管理程序将这些目标模块和库文件链接起来。一个库文件可以和目标模块和者其他库链接，生成可执行代码。MICROCHIP 的库管理程序 MPLIB 可以建立库并维护库。
- **Link**
  将目标模块和库文件链接起来，生成可执行代码的过程。链接工作有 MICROCHIP 的链接器 MPLINK 完成。
- **Listing Directives**
  列表指示符是用来控制 MPASM 列表文件的格式。它们允许指定标题、分页及其他控制。
- **Listing File**
  列表文件是一个 ASCII 格式的文件，它显示出源文件中每条汇编指令、MPASM 指示符、宏语言和机器码之间的对应情况。
- **Local Label**
  局部标号是用 `LOCAL` 指示符定义的标号。这些标号在宏指令的某一特定的时间内存在，换言之：定义为局部性质的标号和符号在执行完 `ENDM` 宏指令后就从符号表里撤出了。
- **Macro**
  宏是一系列汇编指令的集合。当在源代码中遇到宏时，这些指令便会插入到汇编序列里。宏在使用前必须定义，宏不允许前向引用。
  所有跟在 MACRO 指示符后的参数都是宏定义的一部分，宏里使用的标号必须是局部类型，
- **Macro Directives**
  这些指示符在宏提内控制执行和数据的分配。
- **Make Project**
  一条命令，它控制重新创建（REBUILD））一个应用程序，只编译那些自上次建立以来曾经改变过的源文件。
- **Mnemonics**
  这些指令被直接转化为机器码。对 MCU 内部的 RAM 或 ROM 里的数据执行算术或逻辑运算。它们还能把数据从寄存器和存储器里放入、取出，也能改变程序执行流程。也称为 OPCODES。
- **MPASM**
  MICROCHIP 可重定位宏汇编器。
- **MPLAB**
  主要的可执行程序，支持集成调试环境（**IDE**），内涵文本编辑器（**EDITOR**）、项目管理器、仿真/模拟调试器。MPLAB 软件驻留在用户的 PC 主机里。可执

行文件名是 MPLAB.EXE，它可以调用许多其他的文件。

- **MPLAB-C17/C18**
  MICROCHIP 为 PIC17CXX 和 PIC18CXX 设计的 C 编译器。
- **MPLIB**
  和 MPLINK 链接程序一起使用的 MICROCHIP 库文件管理器。
- **MPLINK**
  MICROCHIP 链接器。
- **Nesting Depth**
  宏嵌套深度可达 16 级。
- **Node**
  MPLAB 的项目元件。
- **Object Code**
  汇编器或编译器对源程序处理后生成的机器码。该代码驻留在 PICmicro 的存储器里面，执行用户的应用程序。可重定位代码是由 MPASM 或 MPLAB-C17/C18 生成，通过 MPLINK 链接程序后生成可执行代码。目标代码放置在目标文件中。
- **Object File**
  包含可重定位代码或数据及外部代码和数据的参考目标模块。多个目标模块可以链接为一个可执行文件。当生成目标文件时，源代码里需要有特殊的指示符。目标代码放置在目标文件中。
- **Object File Directives**
  只有生成目标文件时，才使用的指示符。
- **Operators**
  运算符号是一系列形如加号“+”、减号“-”的符号，用以组成运算表达式。每个运算符号都有自己的优先级。
- **PC**
  任何 IBM 及兼容的计算机。MPLAB 要求 486 以上的计算机。
- **PC Host**
  计算机运行于 WINDOWS3.X 或 WINDOWS95/98 环境。
- **PICmicro**
  PICmicro 指 MICROCHIP 的 PIC12CXX、PIC14000、PIC16CXX、PIC17CXX、PIC18CXX 系列微控制器。
- **Precedence**
  优先级指表达式里一些部件比另外一些先进行求值运算。相同优先级别的运算符号其运算顺序由左到右。
- **Program Memory**
  下载了目标系统支持软件的仿真器或者模拟器的存储器。
- **Project**
  一系列源文件和指令，为一个应用系统建立目标代码。
- **Radix**
  进制基数是汇编器求值的时候使用的基本计数机制。默认的进制基数是 16 进制。用户可以使用基数操作符改变默认进制基数。
- **RAM**
  随机存取存储器（RRANDOM ACCESS MEMERY），即数据存储器。
- **Raw Data**

与一个区段相关的二进制代码或数据。

- **Recursion**
  一个已经定义的宏可以调用它自己。当书写递归宏的时候应当特别留意，很容易导致无穷循环而无法退出。
- **Relocatable Section**
  地址还未最后确定的区段。链接器通过重定位过程给可重定位区段分配地址。
- **Relocation**
  链接器的处理过程，在此过程中，地址被分配给可重定位区段；所有可重定位区段里的标识符定义（dentifier symbol definitions）被更新为新的地址。
- **ROM**
  只读存储器（READ ONLY MEMERY）。
- **Script**
  链接器描述文件是 MPLINK 的控制文件。
- **Section**
  具有名称、大小和地址的一些数据或代码的集合。
- **Shared Section**
  驻留在共享（未分堆）数据 RAM 里的区段。
- **Shell**
  MPASM SHELL 是提供给宏汇编器的输入接口界面。有两种界面类型：一种是 DOS 界面，另一种是 WINDOWS 界面。
- **Software Stack**
  MPLAB-C17/C18 的软件堆栈。
- **Source Code**
  源代码是由将会翻译成机器码的 PICmicro 指令系统、MPASM 指示符和宏组成。这种代码适应于 PICmicro 系列 MCU 或者像 MPLAB 之类的 MICROCHIP 开发系统。
- **Source File**
  是由将会翻译成机器码的 PICmicro 指令系统、MPASM 指示符和宏的 ASCII 文本文件的源代码。它可以用任何 ASCII 文本编辑器进行编辑。
- **Stack**
  是数据存储器里的一块区域。用以存储函数参数、返回值、局部变量和返回地址。
- **Symbol**
  用来描述程序里的各种“部件”的通用方法。这些“部件”包括：函数名、变量名、区段名、文件名、结构名等。
- **Unassigned Section**
  在链接器命令文件里没有分配特定的目标存储器块的区段。链接器必须找到一个目标存储器块，在这里安排未定义区段。
- **Uninitialized Data**
  没有初始化赋值的数据。在 C 语言里，<int myVar;> 语句的含义是：定义一个位于未初始化数据区段的变量。