用户手册

NEC

CC78K0R Ver. 1.00

C 编译器

操作篇

目标设备

78K0R 微控制器

文档编号 U17838CA1V0UM00 (第 1 版)
出版日期 2007 年 12 月 CP(K)

日本印刷

[备忘录]

MS-DOS，Windows 和 Windows NT 是微软公司在美国及/或在其它国家的注册商标或商标。
PC/AT 是 IBM（国际商用机器）公司的注册商标。
i386 是 Intel 公司的注册商标。

- **本文档所刊登的内容有效期截至 2007 年 12 月。将来可能未经预先通知而更改。在实际进行生产设计时，请参阅各产品最新的数据表或数据手册等相关资料以获取本公司产品的最新规格。**
- 并非所有的产品和/或型号都向每个国家供应。请向本公司销售代表查询产品供应及其他信息。
- 未经本公司事先书面许可，禁止复制或转载本文件中的内容。否则因本文档所登载内容引发的错误，本公司概不负责。
- 本公司对于因使用本文件中列明的本公司产品而引起的，对第三者的专利、版权以及其它知识产权的侵权行为概不负责。本文件登载的内容不应视为本公司对本公司或其他人所有的专利、版权以及其它知识产权作出任何明示或默示的许可及授权。
- 本文件中的电路、软件以及相关信息仅用以说明半导体产品的运作和应用实例。用户如在设备设计中应用本文件中的电路、软件以及相关信息，应自行负责。对于用户或其他人因使用了上述电路、软件以及相关信息而引起的任何损失，本公司概不负责。
- 虽然本公司致力于提高半导体产品的质量及可靠性，但用户应同意并知晓，我们仍然无法完全消除出现产品缺陷的可能。为了最大限度地减少因本公司半导体产品故障而引起的对人身、财产造成损害（包括死亡）的危险，用户务必在其设计中采用必要的安全措施，如冗余度、防火和防故障等安全设计。
- 本公司产品质量分为：

  “标准等级”、“专业等级”以及“特殊等级”三种质量等级。

  “特殊等级”仅适用于为特定用途而根据用户指定的质量保证程序所开发的日电电子产品。另外，各种日电电子产品的推荐用途取决于其质量等级，详见如下。用户在选用本公司的产品时，请事先确认产品的质量等级。

  “标准等级”： 计算机，办公自动化设备，通信设备，测试和测量设备，音频·视频设备，家电，加工机械以及产业用机器人。

  “专业等级”： 运输设备（汽车、火车、船舶等），交通用信号控制设备，防灾装置，防止犯罪装置，各种安全装置以及医疗设备（不包括专门为维持生命而设计的设备）。

  “特殊等级： 航空器械，宇航设备，海底中继设备，原子能控制系统，为了维持生命的医疗设备、用于维持生命的装置或系统等。

  除在本公司半导体产品的数据表或数据手册等资料中另有特别规定以外，本公司半导体产品的质量等级均为“标准等级”。如果用户希望在本公司设计意图以外使用本公司半导体产品，务必事先与本公司销售代表联系以确认本公司是否同意为该项应用提供支持。

（注）

（1） 本声明中的“本公司”是指日本电气电子株式会社（NEC Electronics Corporation）及其控股公司。

（2） 本声明中的“本公司产品”是指所有由日本电气电子株式会社所开发或制造，或为日本电气电子株式会社（定义如上）开发或制造的产品。

[备忘录]

前言

该手册的目的是帮助您深入理解 CC78K0R 的功能和操作（78K0R 系列 C 编译器）。

该手册没有解释如何编写 CC78K0R 源程序。因此，在阅读手册之前，请先阅读**“CC78K0R C 编译器用户手册语言篇（U17837E）”**(下文称作”语言手册”)。

[目标设备]

借助 CC78K0R 可以开发面向 78K0R 系列微控制器的软件。为了配合使用这个软件，还需要 RA78K0R(78K0R 系列汇编程序包)(另外销售)和目标模块设备文件。

[目标读者]

写作这本手册是为了让读者通过阅读手册获得设备相关使用知识，读者最好具有软件开发经验。但是，关于 C 编译器和 C 语言的知识并不是一定需要的，所以第一次使用 C 编译器的用户可以使用该手册。

[组织结构]

这本手册的结构组织如下描述。

第 1 章　概述

介绍在微控制器开发中 CC78K0R 的作用和位置。

第 2 章　产品概述和安装

介绍如何安装 CC78K0R，所提供程序的文件名，以及程序的操作环境。

第 3 章　从编译到连接的过程

使用样例程序来描述如何运行 CC78K0R，并举例说明从编译到连接的详细处理过程。

第 4 章　CC78K0R 函数

介绍在 CC78K0R 中的优化方法和 ROMization 函数。

第 5 章　编译选项

介绍编译选项，具体的规格说明方法和各个选项的优先级别。

第 6 章　C 编译器输出文件

介绍 CC78K0R 输出的各种列表文件。

第 7 章　C 编译器的使用方法

介绍一些技巧，帮助读者更熟练的使用 CC78K0R。

第 8 章　启动例程

CC78K0R 提供了启动例程作为样例。介绍了启动例程的使用，并提供了关于如何改善的建议。

第 9 章　错误信息

介绍 CC78K0R 输出的错误信息。

附录

附录提供了一个样例程序，一份使用时的注意事项列表，一个命令参数列表和一组索引。

[如何阅读这本手册]

首先，对于希望知道如何开始使用 CC78K0R 的用户，请先阅读**第 3 章从编译到连接的过程**。

理解 C 编译器一般知识的用户或者已经阅读语言手册的用户可以跳过**第 1 章　概述**。

[相关资料]

下面的表格显示了这本手册的相关文档（如用户手册）。在出版物中出现的相关资料可能会包括初稿版本。但是，并未对初稿版本作特殊标注。

开发工具的相关文档（用户手册）

资料名		资料编号
CC78K0R V 1.00 C 编译器	操作篇	本文档
	语言篇	U17837E
RA78K0R V1.00 汇编程序包	操作篇	U17836E
	语言篇	U17835E
SM+ 系统模拟器	操作篇	U18010E
PM + V 6.20 版本		U17990E
ID78K0R-QB 集成调试器 V3.20 版本		U17839E

注意　上述列出的相关资料如有变动恕不另行通知，请务必使用最新版本的设计文件。

[约定]

下面解释本手册中所使用符号的意思。

RTOS	78K0R 系列的实时操作系统 RX78K0R
…:	重复相同的格式。
[ ]:	在括号中的字符可以被忽略。
「 」:	如括号中的字符所示(字符串)。
“ ”:	如括号中的字符所示(字符串)。
‘ ’:	如括号中的字符所示(字符串)。
Boldface:	如粗体字符所示(字符串)。
_ :	在重要的位置或实例中的下划线都属于输入字符序列。
Δ :	至少一个空间间隔。
. :	在程序描述中表示省略。
( ) :	如圆括号中之间的字符所示(字符串)。
/ :	分界符
\:	反斜杠

[文件名称约定]

在命令行中指定输入文件名的约定如下所示。

(1) 指定磁盘文件名

[驱动名] [\] [[路径名]...] 主文件名 [.[文件类型]]
<1> <2> <3> <4> <5>

<1> 指定存储文件的驱动名(A: 到 Z:)。

<2> 指定根目录名。

<3> 指定子目录名。

指定的字符串长度应该在操作系统允许范围内。

可以使用的字符:

操作系统允许圆括号(())，分号(:)，逗号(,)之外的所有字符。

注意，连字符(-)不能被当作路径名的首字符。

<4> 主文件名

指定的字符串长度应该在操作系统允许范围内。

可以使用的字符：

操作系统允许除了圆括号(())，分号(:)，逗号(,)以外的所有字符。

注意，连字符(-)不能被当作路径名的首字符。

<5> 文件类型

指定的字符串长度应该在操作系统允许范围内。

可以使用的字符：

操作系统允许除了圆括号(())，分号(:)，逗号(,)以外的所有字符。

例： C:\Program Files\NEC Electronics Tools\CC78K0R\V1.00\smp78k0r\cc78k0r

注意 **1.** 在‘:’, ‘.’和 ‘\’之前或之后不能有空格。

2. 不区分大写和小写 (大小写不敏感)。

(2) 指定设备文件名

可以使用下列逻辑设备。

逻辑设备	描述
CON	输出到控制台。
PRN	输出到打印机。
AUX	输出到辅助输出设备。
NUL	伪输出(没有输出)

目录

插图列表

插图编号　插图标题 ..... 页码

表格列表

表格编号　表格标题..... 页码

第 1 章 概述

CC78K0R C 编译器能够把符合 ANSI-C 注规格或符合 78K0R 系列规格的 C 语言源程序转变成 78K0R 微控制器能够识别的机器语言。

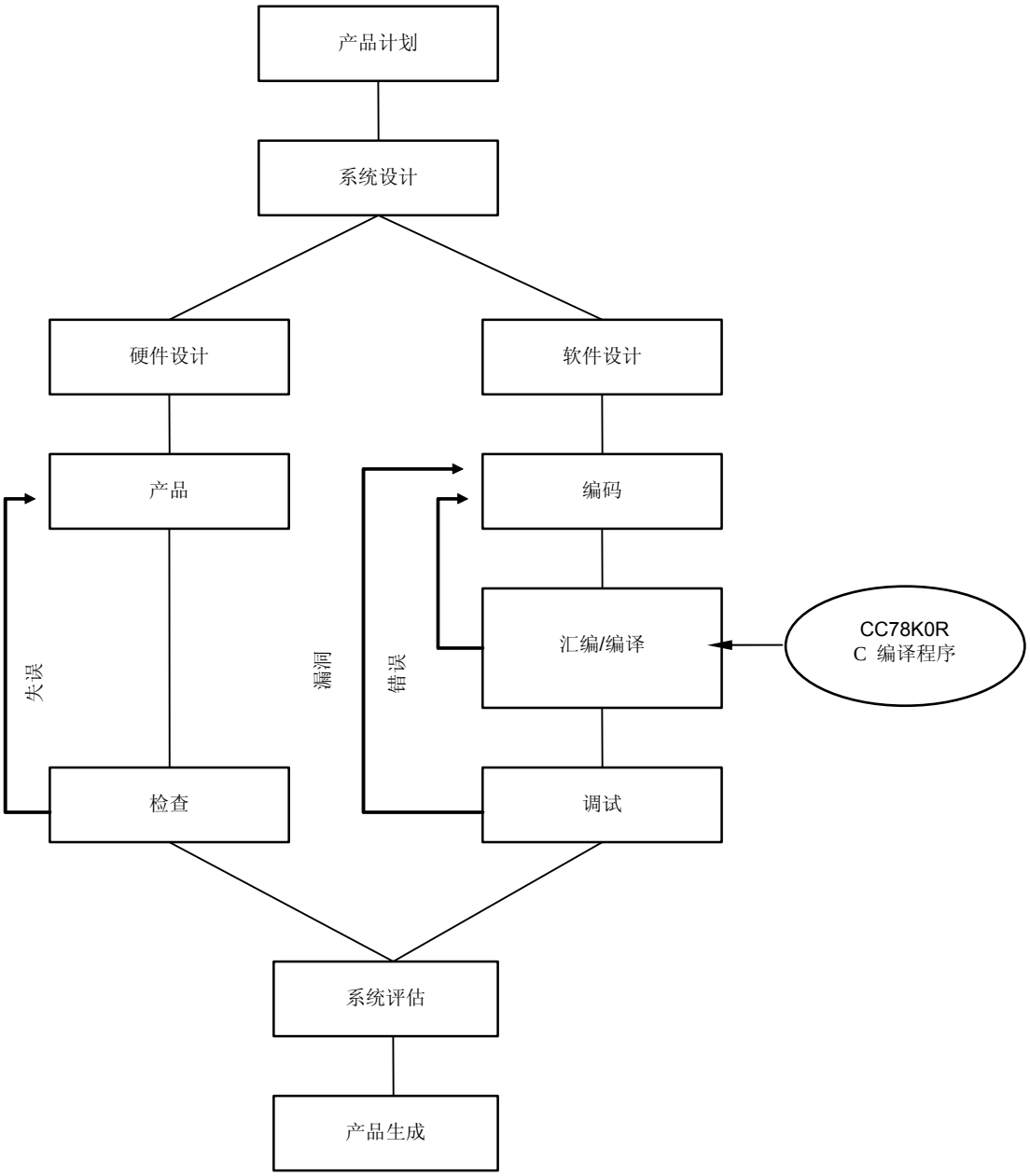
CC78K0R 支持 Windows™系统，必须在 PM +（项目管理器）中才能使用 CC78K0R，PM +工具包含在 CC78K0R 系列的汇编程序安装包中。如果没有使用 PM +，编译程序可以在命令提示符下运行。

注 ANSI-C 是美国国家标准局所制定的 C 语言标准。

1.1 CC78K0R的作用

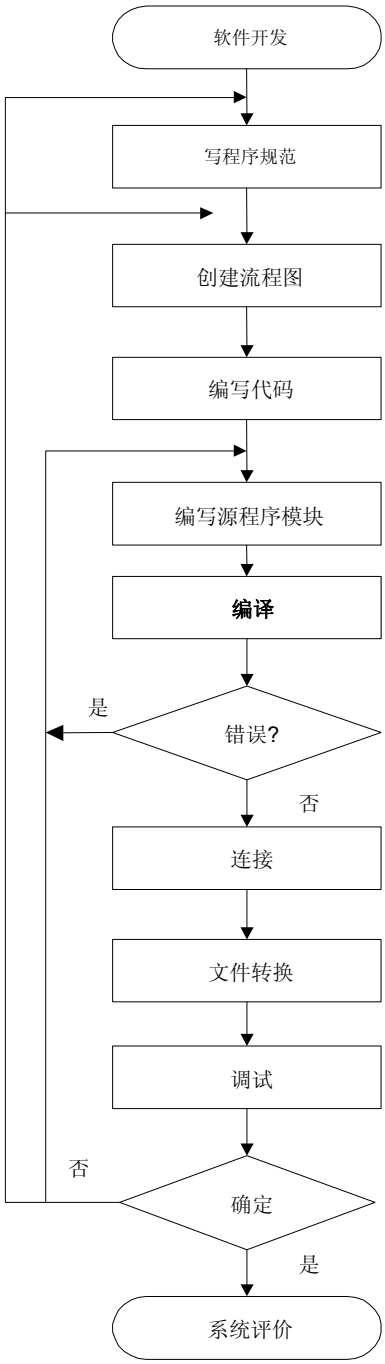
CC78K0R 在产品开发中的位置如下图所示。

图 1-1.开发过程



软件开发过程如下图所示。

图 1-2. 软件开发过程



… 使用 78K0R 系列 C 语言或 ANSI-C。

… 使用编辑器创件建 C 源程序模块文件。

… 连接到引用库和函数库。

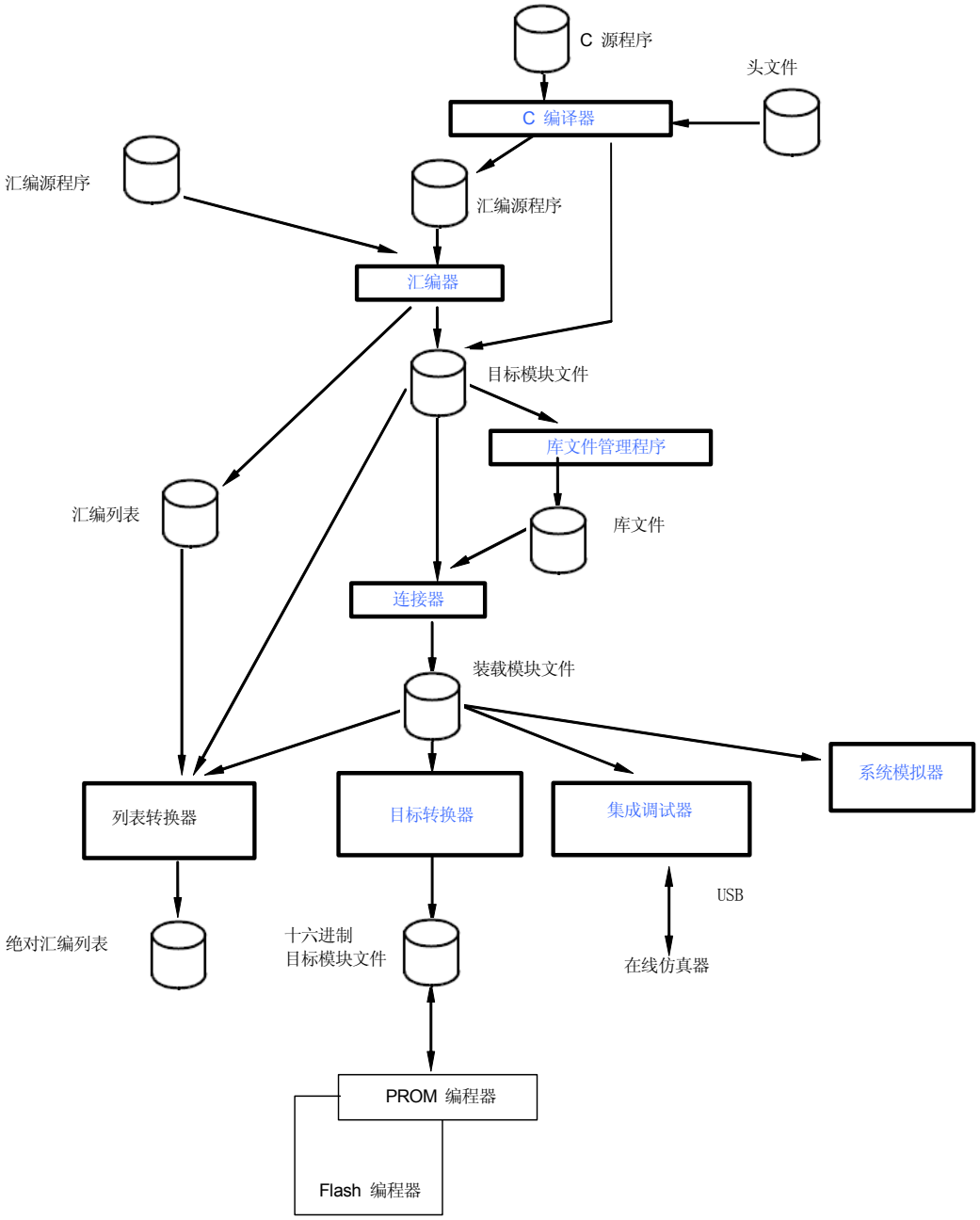
… 将文件转换成十六进制格式。

… 使用硬件调试器（在线仿真器等）来检验系统的运行。

1.2 使用CC78K0R的开发过程

使用 CC78K0R 的开发过程如下图所示。

图 1-3. 使用 CC78K0R 的程序开发过程



1.2.1 使用编辑器创建源程序模块

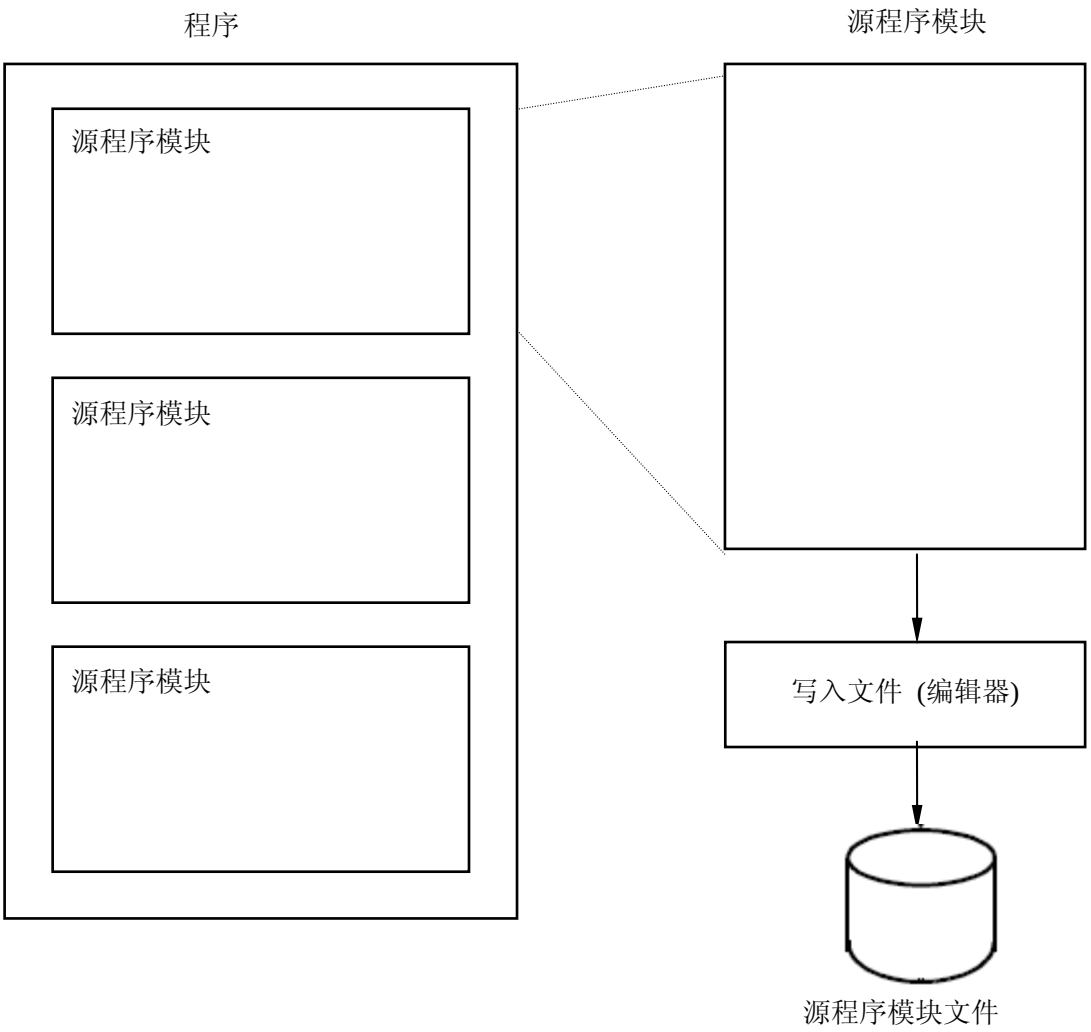
一个程序被划分为若干个功能模块。

其中一个模块是代码单元，也是编译器的输入单元。输入到 C 编译器的模块被称作 C 源程序模块。

当所有的 C 源程序模块编写完成后，使用编辑器将源程序模块存入某个文件中。以这种方式创建的文件被称作 C 源程序模块文件。

这个 C 源程序模块文件就是 CC78K0R 的输入文件。

图 1-4. 创建源程序模块文件

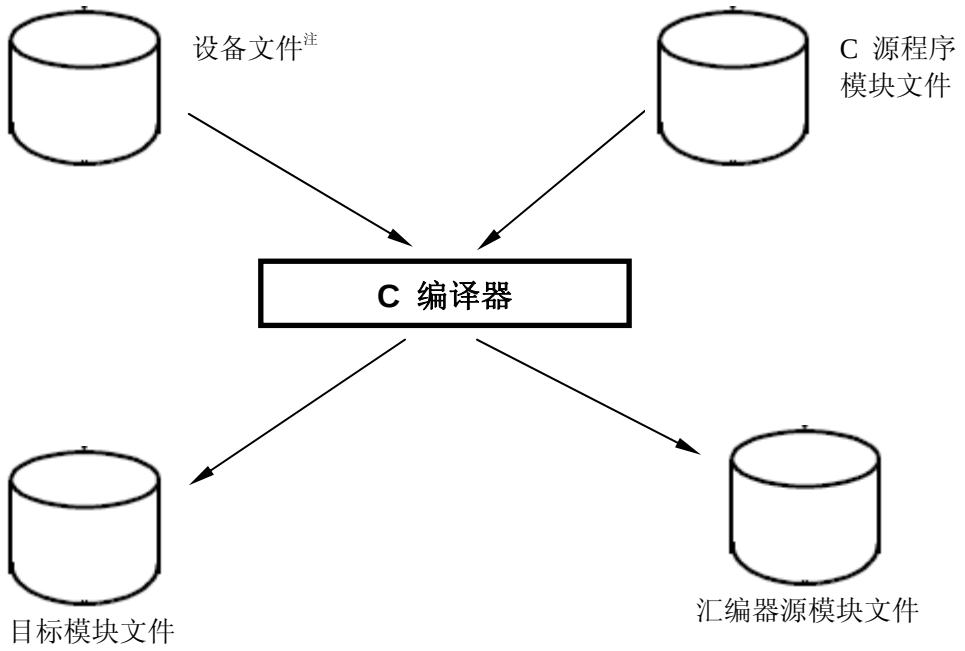


1.2.2 C 编译器

C 编译器读入 C 源程序模块后，将 C 语言转换成机器语言。如果在 C 源程序模块中发现描述错误，就会输出编译错误信息。如果没有编译错误，就会生成目标模块文件。

为了在汇编语言中对程序进行校正和检查，需要生成汇编源程序模块文件。如果要输出汇编源程序模块文件，当编译时在编译选项中选中了-A 或-SA 选项，就可以创建汇编源程序模块源文件。（要查看选项的相关信息，可参阅“**第 5 章 编译选项**”）

图 1-5. C 编译器功能



注 设备文件要从在线发送服务（ODS）下载获取，ODS 可以从如下网址登录进入。

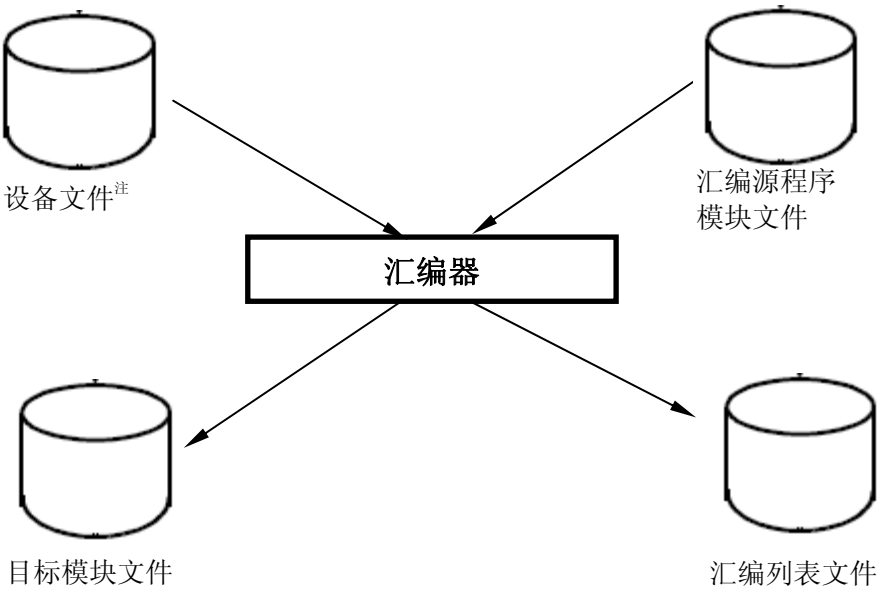
http://www.necel.com/micro/ods/eng/index.html

1.2.3 汇编器

汇编工作是通过使用 RA78K0R 汇编程序安装包（单独销售）中的汇编器来执行的。

汇编器可以对汇编源程序模块文件进行处理，是将汇编语言转化成机器语言的一种程序。如果在源程序模块中发现描述错误，就会输出汇编错误。如果没有汇编错误，则会生成目标模块文件，这个目标文件模块中包括机器语言信息和位置分配信息，例如各条机器语言代码被放在哪个内存单元地址。除此以外，在汇编过程中的信息会以汇编列表文件的形式输出。

图 1-6. 汇编器功能



注　设备文件要从在线发送服务（ODS）下载获取，ODS 可以从如下网址登录进入。

http://www.necel.com/micro/ods/eng/index.html

1.2.4 连接器

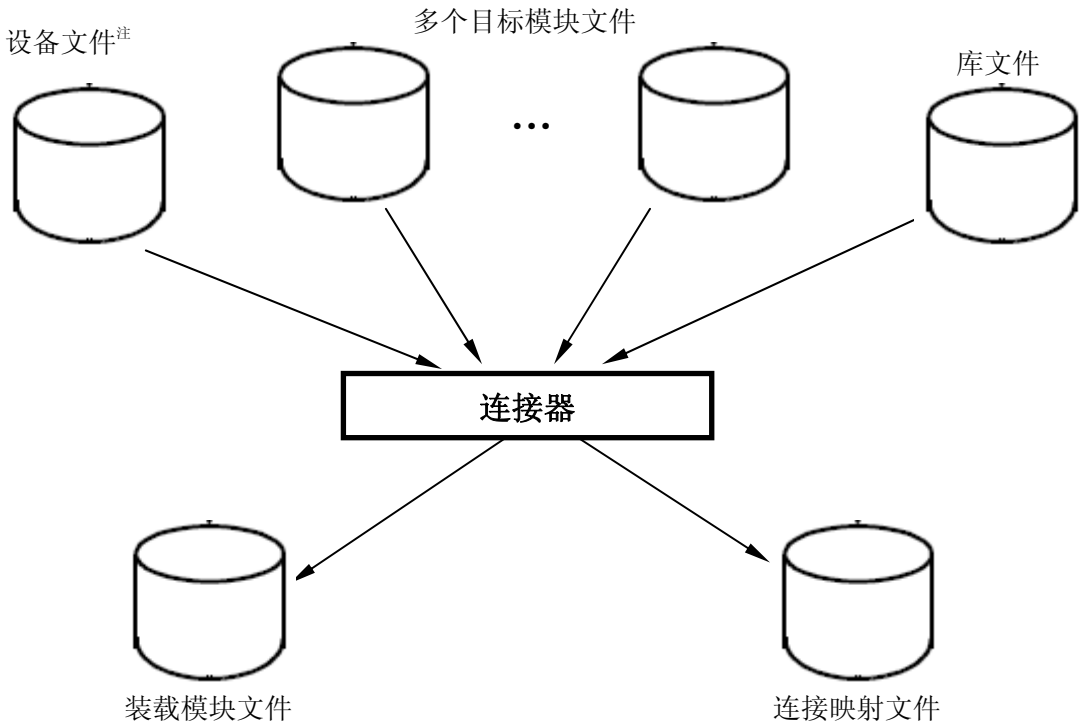
连接操作是通过使用 RA78K0R 汇编程序包（单独销售）中的连接器来执行的。

连接器的输入文件有编译器输出的目标模块文件，也有汇编器输出的目标模块文件，同时将它们和库文件连接起来(即使只有一个目标模块，也必须执行连接操作)。

会输出一个装载模块文件。

在这种情况下，连接器决定输入模块中的重定位段的地址。同时也决定了重定位符号的值和外部引用符号的值，并将正确的值嵌入到装载模块文件中。连接器将连接信息输出到连接映射文件（link map）。

图 1-7. 连接器功能



注　设备文件要从在线发送服务（ODS）下载获取，ODS 可以从如下网址登录进入。

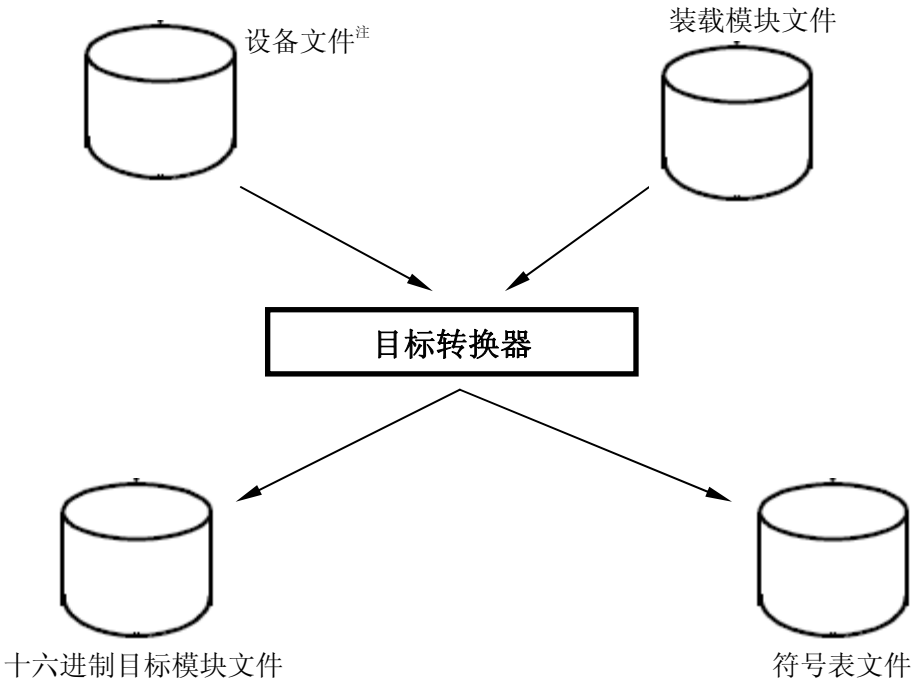
http://www.necel.com/micro/ods/eng/index.html

1.2.5 目标转换器

目标转换操作是通过 RA78K0R 汇编程序包(单独销售)中的转换器来执行的。

目标转换器读入连接器产生的装载模块文件，转换文件的格式，产生 Intel 标准格式的十六进制模块文件。符号信息输出到符号表文件中。

图 1-8. 目标转换器功能



注　设备文件要从在线发送服务（ODS）下载获取，ODS 可以从如下网址登录进入。

http://www.necel.com/micro/ods/eng/index.html

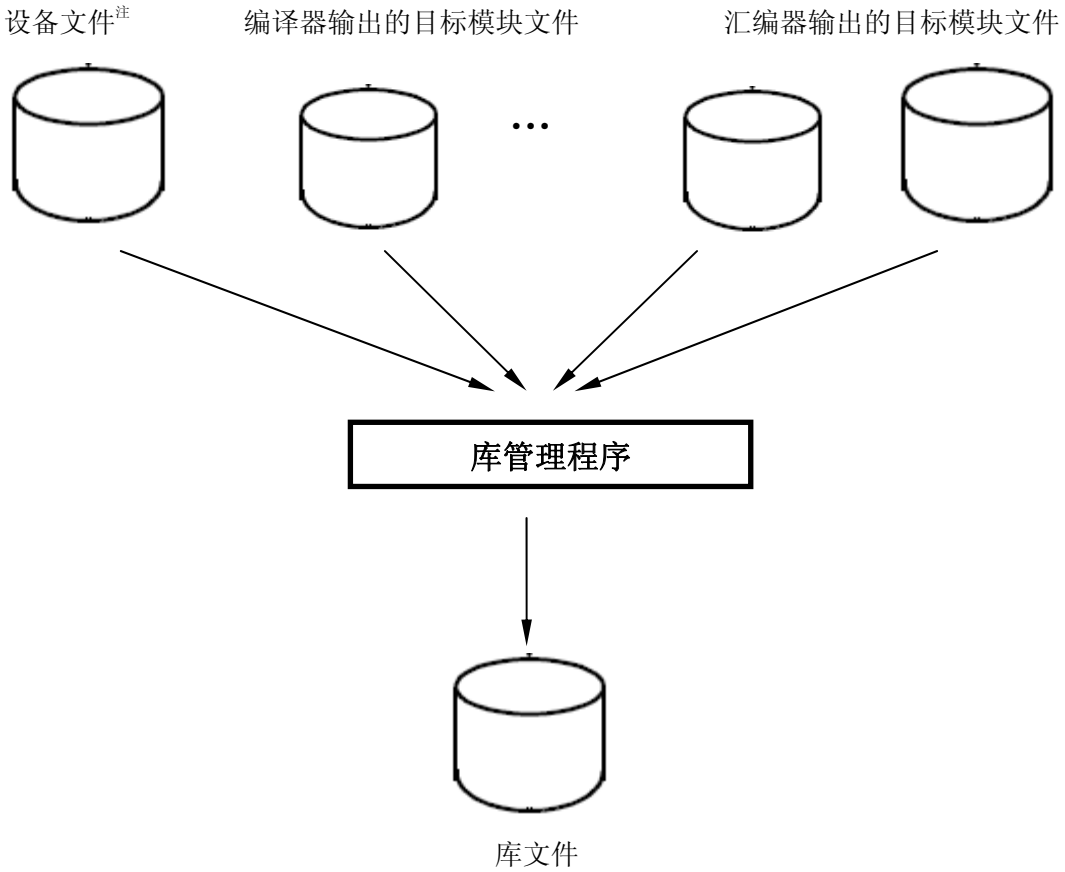
1.2.6 库管理程序

为了方便起见，拥有通用接口并被清晰定义的模块被做成库。通过创建库，许多目标模块组成一个文件，更容易处理。

连接器可以从库文件中提取出需要的模块并将它们连接起来。因此，如果一个库文件中包含多个模块，当每个模块的连接无需单独指定参数时，就需要使用模块文件的名称。

库管理程序用来创建和更新库文件。库管理功能通过 RA78K0R 汇编程序包（单独销售）中的库管理程序来执行。

图 1-9. 库管理程序功能



注　设备文件要从在线发送服务（ODS）下载获取，ODS 可以从如下网址登录进入。

http://www.necel.com/micro/ods/eng/index.html

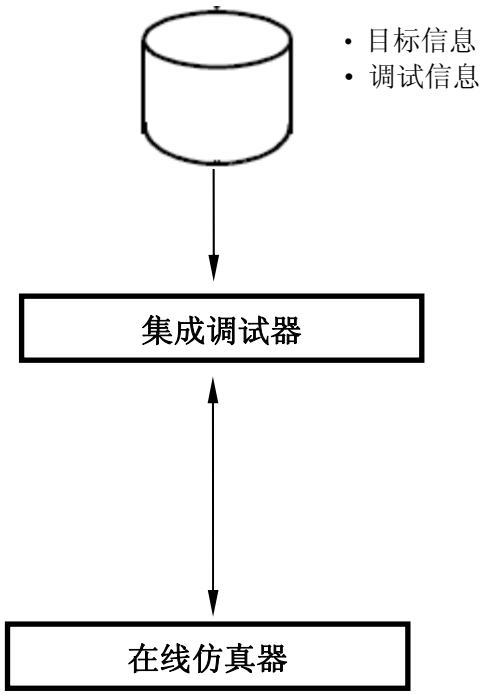
1.2.7 调试器

将连接器输出的装载模块文件通过 ID78K0R-NS/ID78K0R-QB（78K0R 系列集成调试器）下载到 IE（在线仿真器）中，就可以使用图形用户接口对源程序进行调试。

调试时，当目标源程序被编译时(-G 是缺省选项)，指定了-G 选项就可以输出调试信息。指定这个参数，调试中所需的符号和行号就会被加入到目标模块中。对于编译选项的信息，请查阅“第 5 章 编译选项”。

调试器和仿真器是独立包装并各自销售的不同设备。

图 1-10. 调试器的功能



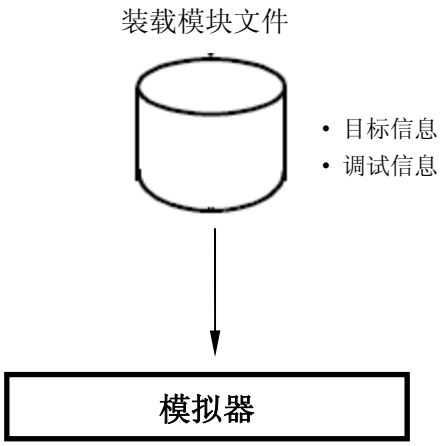
1.2.8 系统模拟器

将连接器输出的装载模块文件通过 SM78K0R（78K0R 系列集成模拟器）下载，就可以使用图形用户接口对源程序进行调试

SM78K0R 是在主机上进行模拟的软件，和 ID78K0R-QB 有着同样的操作界面。

除了在 SM78K0R 中模拟机器指令，同时也可以模拟 MCU 的片上外围设备和中断。因为外围部件和过程用来构建虚拟的目标系统，所以在开发早期阶段就可以对包含目标系统操作的程序进行调试，并且可以脱离硬件系统独立进行。

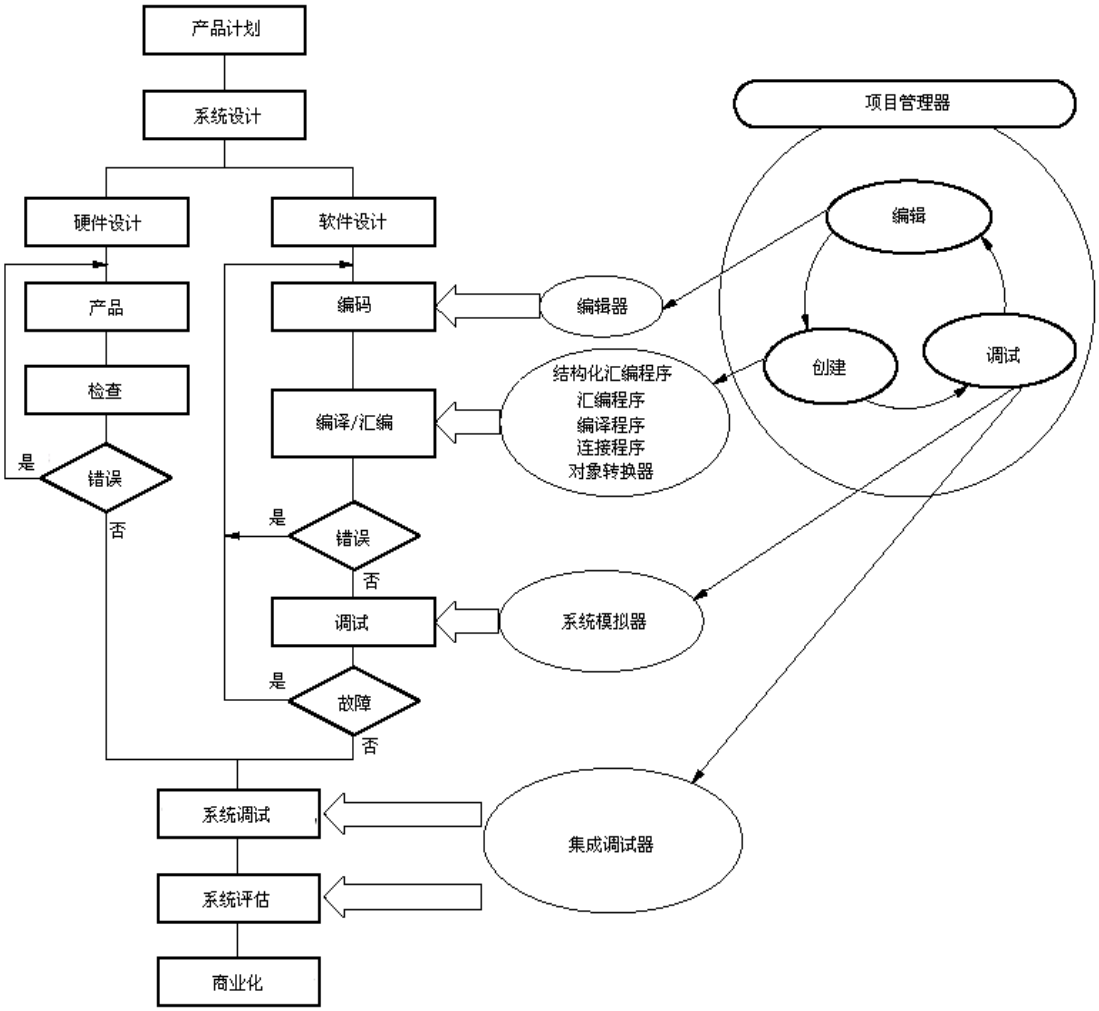
图 1-11. 模拟器功能



1.2.9 PM +

PM +（项目管理器）提供集成化开发环境，可以帮用户高效开发程序。因此，可以使用 PM +的图形用户界面来进行编辑至调试的一系列开发工作过程。

图 1-12. PM +功能



第 2 章 产品概述和安装

本章介绍了将 CC78K0R 的文件安装到用户开发环境（主机）的过程，以及从用户开发环境中卸载的过程。

2.1 主机和供应媒介

C 编译器支持表 2-1 中列出的开发环境。

表 2-1. C 编译器的供应媒介和记录格式

主机	操作系统	供应媒介
IBM PC/AT™ 及兼容机	Windows(2000/XP)注	CD-ROM

注 如果要在 Windows 环境中使用 C 编译器，必须使用 PM +。如果不使用 PM +，也可以在命令提示符下启动 CC78K0R 编译器。

2.2 安装

将供应媒体中的 CC78K0R 文件安装到主机的过程说明如下。

(1) 启动 Windows
为主机和外部设备供电并启动 Windows。

(2) 设置供应媒体
将 CC78K0R 的供应媒体放入主机的驱动器(CD-ROM 驱动器)中。安装程序将自动启动。根据显示的提示信息逐步安装。

警告 如果安装程序没有自动启动，请执行 CC78K0R 文件夹中的 SETUP.EXE 文件。

(3) 文件确认
使用 Windows 资源管理器等，检查 CC78K0R 供应媒体中的文件是否已经安装到主机上。
每个文件夹的详细信息，参考 **2.4 Windows 版本目录结构**。

2.3 设备文件的安装

请从在线传送服务（ODS）自行下载对应型号的最新版本设备文件，ODS 的入口网址为：

http://www.necel.com/micro/ods/eng/index.html

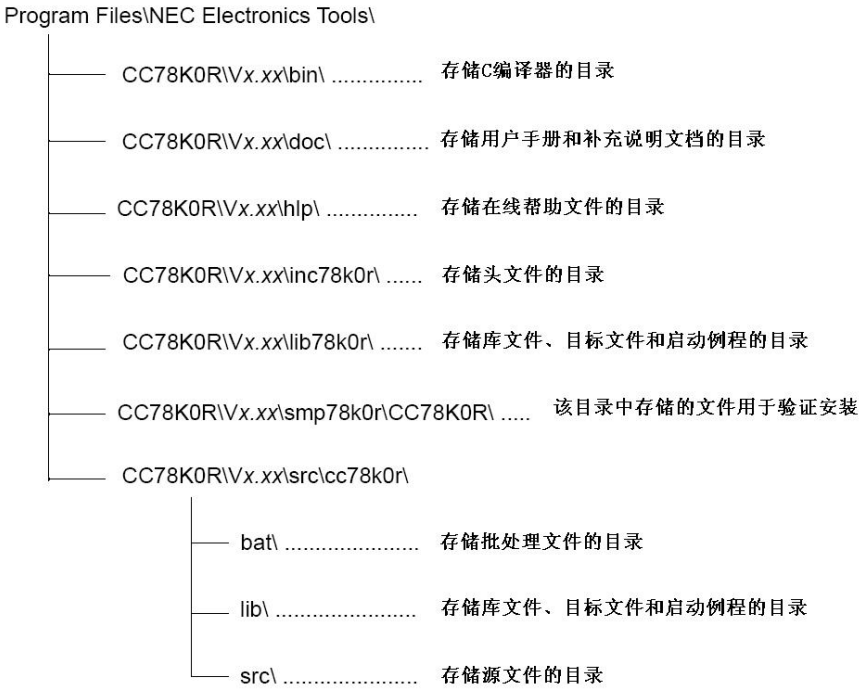
使用安装文件来进行设备文件的安装。设备安装文件应该和 CC78K0R 同时安装。

2.4 目录结构

Windows 系统安装过程中的标准目录是“Program Files\NEC Electronics Tools”。安装目录中的文档结构如下，注意在安装过程中可以改变驱动器和安装目录。当使用 PM +执行 MAKE 操作时，相关的工具（CC78K0R，RA78K0R）也被安装到这个驱动器和目录。

本手册中假定的标准目录就是“Program Files\NEC Electronics Tools”，这是默认的程序名称，也是安装的默认路径。

图 2-1 目录结构



2.5 文档结构

下面的表格列出了每个目录的内容。

目录结构和文档组织是在安装时就已经决定了。

表 2-2. 文档组织结构(* = 字母数字符号)

目录名	文件名	说明
CC78K0R\V*x.xx*\bin\	CC78K0r.exe	编译器
	CC78K0r.msg	信息文件
	*.hlp	帮助文件
	*.dll	DLL 文件
CC78K0R\V*x.xx*\hlp\	cc78k0rp.chm	在线帮助文件
CC78K0R\V*x.xx*\inc78k0r\	*.h 注 1	标准库的头文件
CC78K0R\V*x.xx*\lib78k0r\ (For link) 注 2,3	cl0r*.lib	库（运行库和标准库）
	s0r*.rel	服务于启动例程的目标文件
CC78K0R\V*x.xx*\smp78k0r\ CC78K0R\	prime.c	用来验证安装的源程序
	sample.bat	验证安装所需的批处理文件
	readme.doc	验证安装所需文件的说明文件
	lk78k0r.dr	用于引用的连接命令文件
CC78K0R\V*x.xx*\src\cc78k0r\bat \注 4	mkstup.bat	启动例程的汇编批处理文件
	reprom.bat	用于更新 rom.asm
	*.bat 注 5	更新标准函数的批处理文件（部分的）
CC78K0R\Vx.xx\src\cc78k0r\lib\(修改用) 注 2	cl0r*.lib	库（运行库和标准库）
	s0r*.rel	服务于启动例程的目标文件
CC78K0R\V*x.xx*\src\cc78k0r\src\	cstart*.asm 注 4	启动例程的源文件
	rom.asm	ROMization 例程的源文件
	*.asm 注 5	标准函数的源文件（部分的）

备注 *：字母数字符号。

注 1. 参见 **CC78K0R C 编译器 语言篇用户手册**。

注 2. 为了修改启动例程，修改 CC78K0R\Vx.xx\src\cc78k0r\lib 目录中的源文件。该文件和存储在 CC78K0R\Vx.xx\src\cc78k0r\lib 目录下的某个批处理文件共同编译，所以将该文件拷贝到 CC78K0R\Vx.xx\lib78k0r 目录下，并和用户程序连接。

注 3. 参考“**2.5.1 库文件**”。

注 4. 本目录中的批处理文件不能在 PM +中使用，他们只能通过命令提示符来执行。只有在源文件必须被修改时才使用这些文件。

注 5. 参考**表 8-1 的内容**。

注 6. * = B | E | N（B：指定根区域，E：指定闪存区域，N：未使用标准程序库）。

注 7. 参考**表 8-2 的内容**。

2.5.1 库文件

·这些文件由标准库、运行时刻库和启动例程组成。

下表列出了目录内容。

表 2-3 程序库文件

目录名	文件名			文件作用
	普通	Boot区域	Flash 区域	
LIB78K0R\	cl0rm.lib cl0rl.lib cl0rmf.lib cl0rlf.lib cl0rxm.lib注 3 cl0rxl.lib注 3	cl0rm.lib cl0rl.lib cl0rmf.lib cl0rlf.lib cl0rxm.lib注 3 cl0rxl.lib注 3	cl0rme.lib cl0rle.lib cl0rmfe.lib cl0rlfe.lib cl0rxme.lib注 3 cl0rxle.lib注 3	库（运行时刻库和标准库）注 1
	s0rm.rel s0rml.rel s0rl.rel s0rll.rel	s0rmb.rel s0rmlb.rel s0rlb.rel s0rllb.rel	s0rme.rel s0rmle.rel s0rle.rel s0rlle.rel	启动例程的目标文件注 2

注 **1.** 命名程序库的规则如下。

lib78K0R\cl0r<mul><model><float><flash>.lib

<mul >
None 未使用乘法器
X 使用乘法器

<model>
m 小型模式或中等模式
l 紧凑模式或大型模式

<float>
None 标准库和运行时刻库 （不支持移动指针程序库没有使用）
f 浮点指针库

<flash>
None 普通/boot 区域
e flash 存储器区域

注 **2.** 命名启动例程的规则如下。

lib78K0R\s0r<model><lib><flash>.rel

<model>
m 中等模式（同样可以指定为小型模式）
l 大型模式（同样可以指定为紧凑模式）

<lib>
None 未使用标准库函数
l 使用了标准库函数

<flash>
None 普通
b boot 区域
e flash 存储器区域

注 3. CC78K0R的库和下列乘法器设备兼容。
但是，在计算乘除法时可能会发生中断，某些计算结果是禁止被打断的，以保证得到正确的计算结果。
关于库函数和中断响应时间的关系，敬请参阅**CC78K0R C编译器语言篇用户手册**。

[特殊功能寄存器]

功能	保留字	地址	大小
乘法输入数据 A	MULA	FFFF0H	16 位
乘法输入数据 B	MULB	FFFF2H	16 位
乘法结果数据	MULOH，MULOL	FFFF4H，FFF6H	16 位 * 2

<乘法时的寄存器配置>

<乘数 A> <被乘数 B> <乘积结果>
MULA (第15至0位) * MULB (第15至0位) = MULOH（高位）(15至0位)，MULOL (低位) (15至0位)

2.6 卸载步骤

下面介绍从主机中卸载安装的步骤。

(1) Windows 启动

启动主机和外围设备电源，并启动Windows。

(2) 删除CC78K0R

双击<控制面板>窗口中的[添加或删除程序]图标，并选中“NECEL CC78K0R Vx.xx”

(3) 文件确认

使用Windows 资源管理器等，检查主机中安装的文件是否被卸载。关于每个文件夹的详细情况，请参阅“2.4 目录配置”

2.7 环境设置

2.7.1 主机

CC78K0R 处理 32 位系统，并运行于装备了 i386™CPU 或更高版本的模型上。

- Windows 2000/XP
- Windows 2000/XP 下的命令提示符

2.7.2 环境变量

为名称提示符操作设置以下环境变量。

表 2-4 环境变量

环境变量	描述
PATH	指定编译器的存储目录。
TMP	指定用于创建临时文件的目录
LANG78K	指定源文件中汉字码（2 字节代码）。 sjis： Shift JIS（默认） euc： EUC none： 没有2 字节代码
INC78K0R	指定C编译器标准头文件的存储目录。
LIB78K0R	指定C编译器库的存储目录。

[规格示例**]**

```
PATH=%PATH%;    C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\bin
set     TMP=C:\tmp
set     LANG78K=sjis
set     INC78K0R=C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\inc78k0r
set     LIB78K0R=C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r
```

第 3 章 编译到连接的过程

本章用 CC78K0R 和 RA78K0R 汇编程序包来描述编译到连接的整个过程。

实际上，对'prime.c' 这个样例程序按照本章节描述的方法执行编译到连接整个过程，你可以熟悉编译、汇编和连接（见**附录 A 样例程序**的相关资料）。

本章介绍了如何在 PM +中执行，以及如何从命令提示符执行。（安装信息参见 **2.2 安装**）。

3.1 PM +

本节描述了在 PM +中启动 CC78K0R 的用户接口，PM +包括在 RA78K0R 汇编程序安装包中。

如果在 PM +中启动 CC78K0R，会引用 CC78K0R 中的 CC78K0RP.DLL 动态连接文件。

3.1.1 CC78K0RP.DLL的位置（工具动态连接文件）

这些工具相关的动态连接文件，比如 CC78K0RP.DLL，是从 WINDOWS 系统下的 PM +中运行 78K0R 系列 C 编译器（CC78K0R）所必需的。

3.1.2 执行的环境

这个环境由 PM +决定。

3.1.3 CC78K0R选项设置菜单

(1) 选项菜单条目

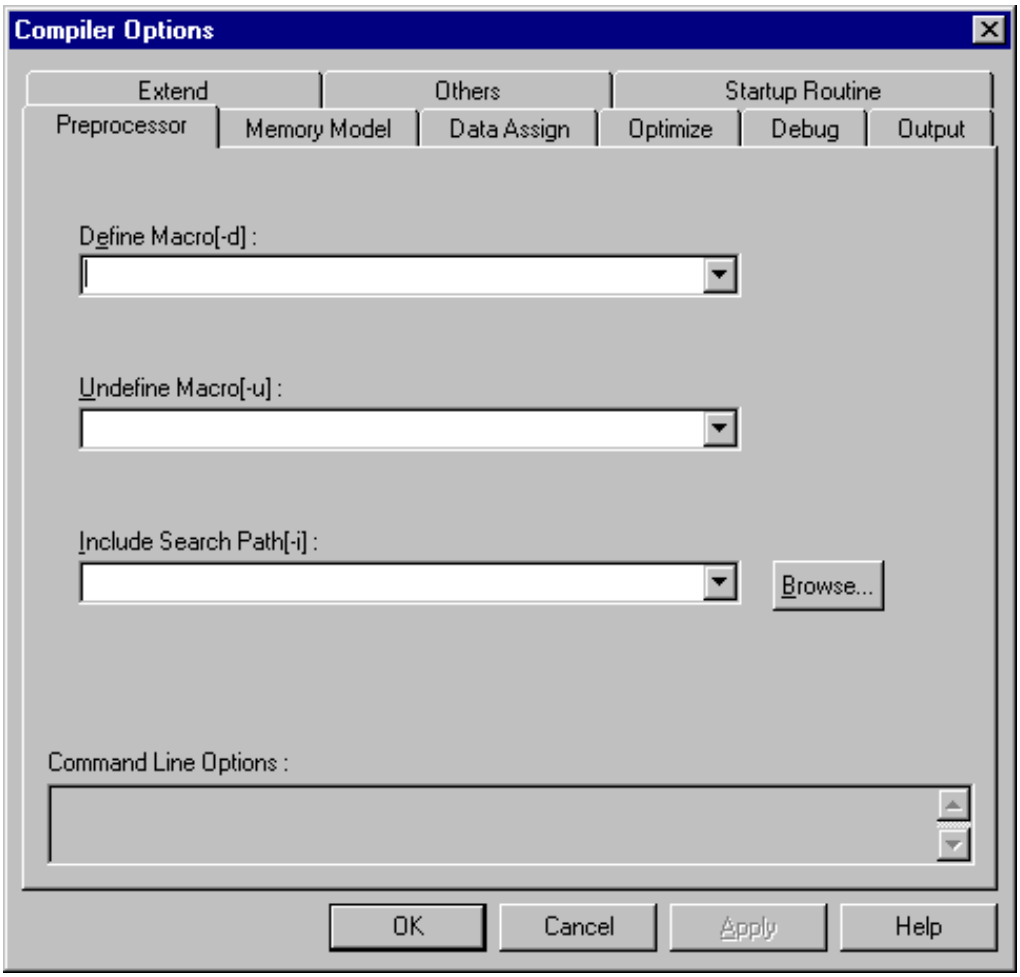
CC78K0R C 编译器安装包中的工具动态连接文件 DLL 文件将在 PM +中的[Tools]菜单中添加“Compiler Options…”项。

(2) < Compiler Options > 对话框

在 PM +中，选择[Tools]下的[Compiler Options…]菜单来为 DLL 工具调用选项设置功能。

< Compiler Options >对话框如下所示。

图 3-1 < Compiler Options >对话框

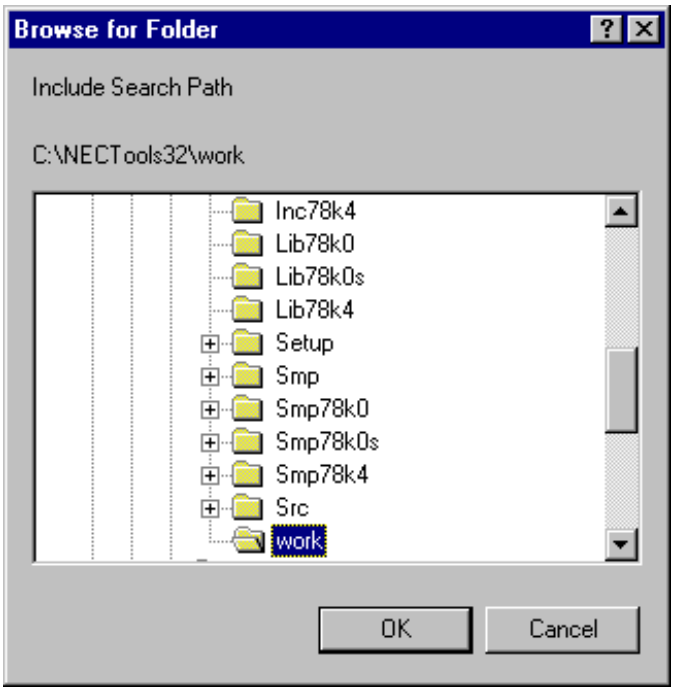


(a) [Browse for Folder]对话框

在< Compiler Options > 对话框中，当单击[Browse...]按钮设置路径时，会显示下列对话框。在本对话框中只能选文件夹。

- 目标模块文件输出路径
- 汇编文件模块文件输出路径
- 错误列表文件输出文件
- 交叉引用列表文件输出路径
- 预处理列表文件输出路径
- 临时文件路径

图 3-2 < Browse for Folder >对话框



(b) [Browse for Folder]对话框

在指定参数文件时单击[Browse…]按钮，会出现下列对话框。

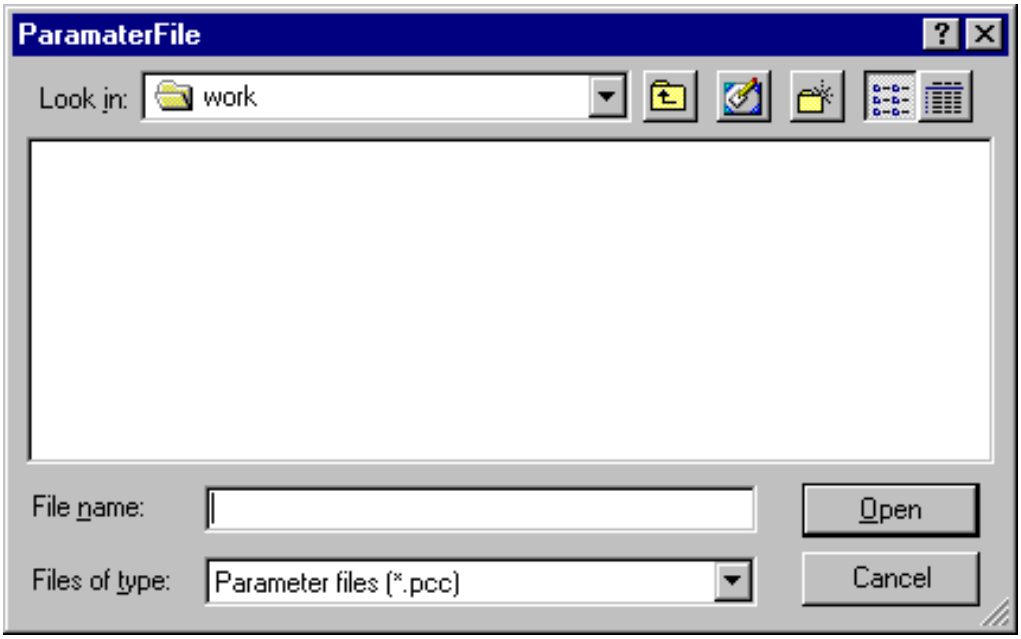
- 在[Others]标签页下的参数文件

对话框的情况如下：

当前目录： 项目文件目录

文件类型： 参数文件 (*.pcc)

图 3-3 < ParameterFile >对话框



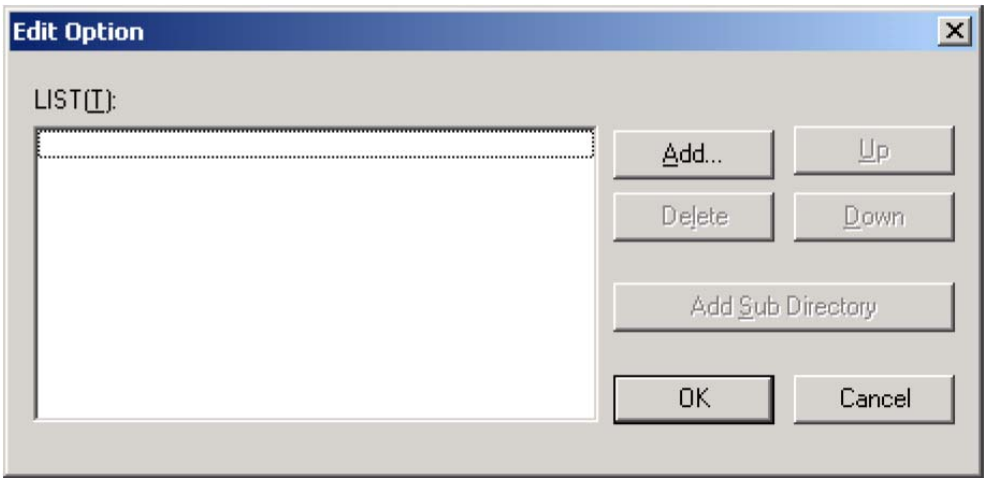
(c) [Edit Option]对话框

在< Compiler Options >对话框中，如果点击了[Edit...]按钮，就会出现下列对话框。

- 在[Preprocessor]标签页下定义宏
- 在[Preprocessor]标签页下取消定义宏
- 在[Preprocessor]标签页下添加搜索目录

在< Edit Option >对话框中的各项都是列表格式进行编辑的。

图 3-4 < Edit Option >对话框



< Edit Option >对话框的描述如下。

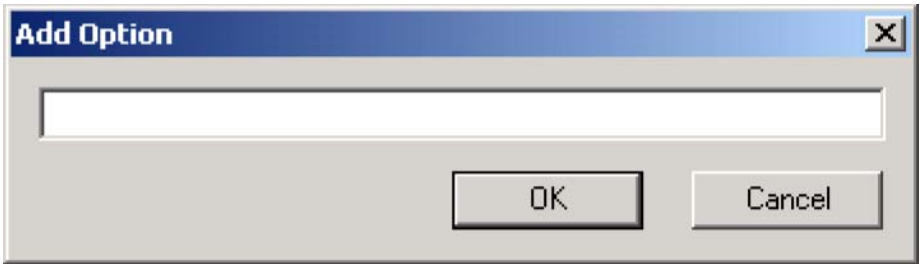
- [Add] 按钮

添加一个列表项。

如果要添加的是一个文件或者目录，就会打开对应的< Browse for Folder >对话框。

在所有其它情况下，都会打开< Add Option >对话框，在该对话框中指定待添加项的具体细节。

图 3-5 <Add Option>对话框



- [Delete] 按钮

删除所选的列表项。

- [Up] 按钮

向上移动所选的列表项。

- [Down] 按钮

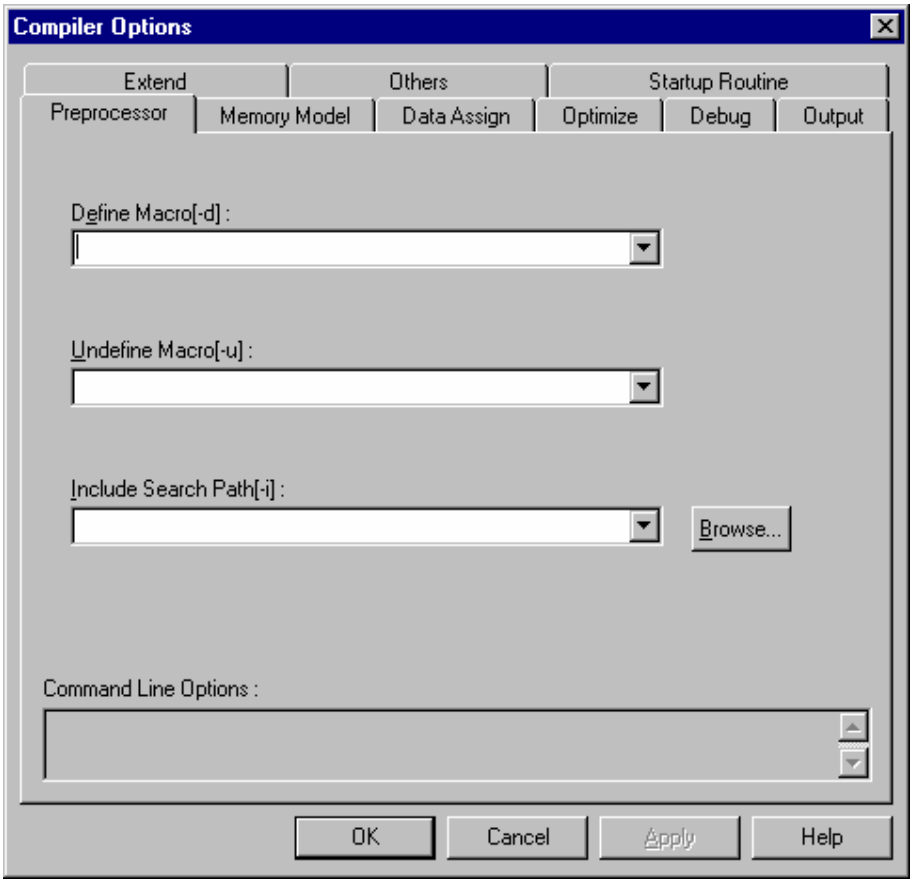
向下移动所选的列表项。

- [Add Sub Directory] 按钮
如果在<< Others >>该项被指定为包含搜索路径[-i](I)，则可以为该列表项添加子目录。

3.1.4 [Compiler Options]对话框的具体描述

下面介绍<Compiler Options>对话框的各个部分。

图 3-6 < Compiler Options >对话框



- 编译器选项的设置

  编译器设置选项分为以下的 9 个标签页并单独设置。

  单击对话框上相应的标签页可以显示每一个个选项类的设置界面。

  [Preprocessor]预处理器标签页（默认）
  [Memory Model]存储器模块标签页
  [Data Assign]数据的分配标签页
  [Optimize]优化标签页
  [Debug]调试标签页
  [Output]输出标签页
  [Extend]扩展标签页
  [Others]其他标签页
  [Startup Routine]启动例程标签页

- 命令行选项：
  显示当前设置的选项字符串。
  在<Others>对话框中的<Other Options>中输入的选项字符串可以实时的显示出来。
  在显示区域无法进行任何输入。
  即使 CC78K0R 的默认选项是“已指定的”状态 (比如，选择框是选中的，等等)，在默认的情况下也不会显示任何东西。
  选项在选项属性显示区域无法完全显示时，可以通过拖动[ScrollBar]滚动条来查看。

- [OK] 按钮
  接受本对话框中的设置选项，并且会关闭<Compiler Options>对话框。
  如果在源程序文件列表中选择了一个源文件，那么这些选项就是为这个文件所设置的。如果没有单独选中其中某些项，那么这些选项对所有的源程序文件都适用。

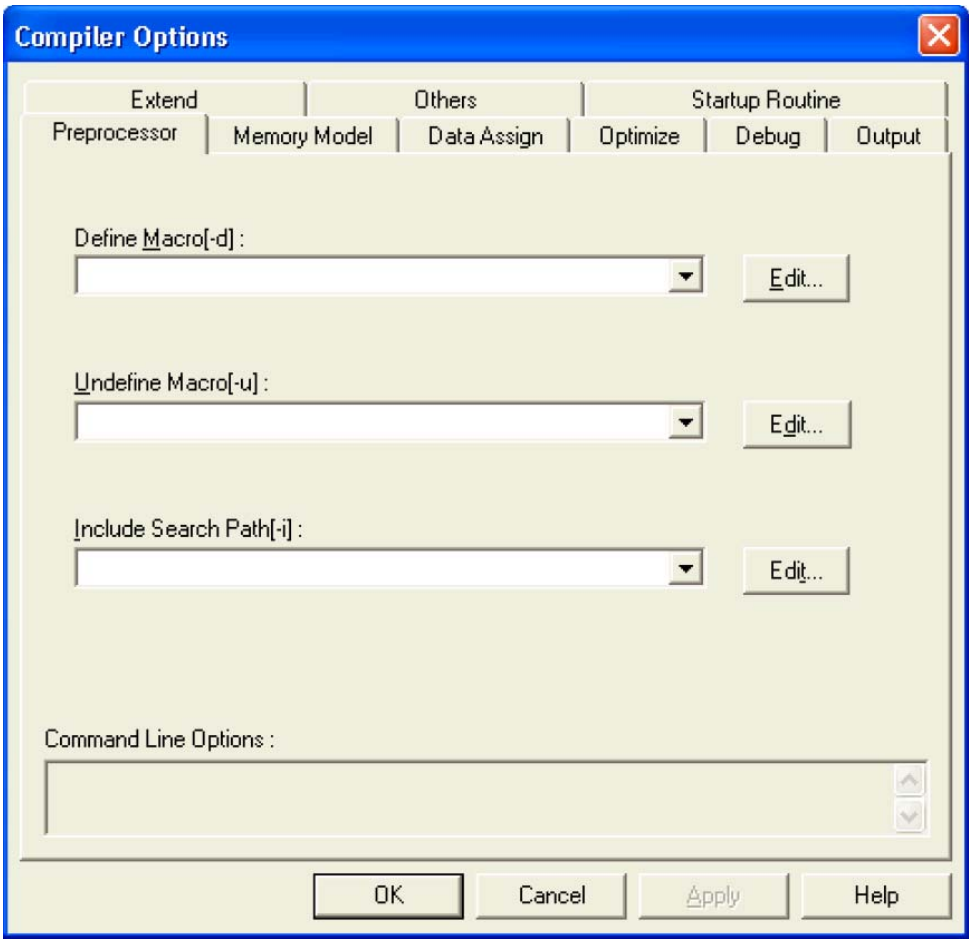
- [Cancel]按钮
  设定的选项不会生效，并关闭这个对话框。
  ESC 键和[Cancel]按钮有着同样的作用，不管目前在哪个对话框中。

- [Apply] 按钮
  只有在设置的选项发生改变时这个按钮才有效。
  应用对话框中编辑的内容，<Compiler Options>对话框继续显示。

- [Help]按钮
  打开关于本对话框的帮助文件。

(1) 选择“Preprocessor”标签页

图 3-7 <Compiler Options>对话框（当选择“Preprocessor”标签页时）



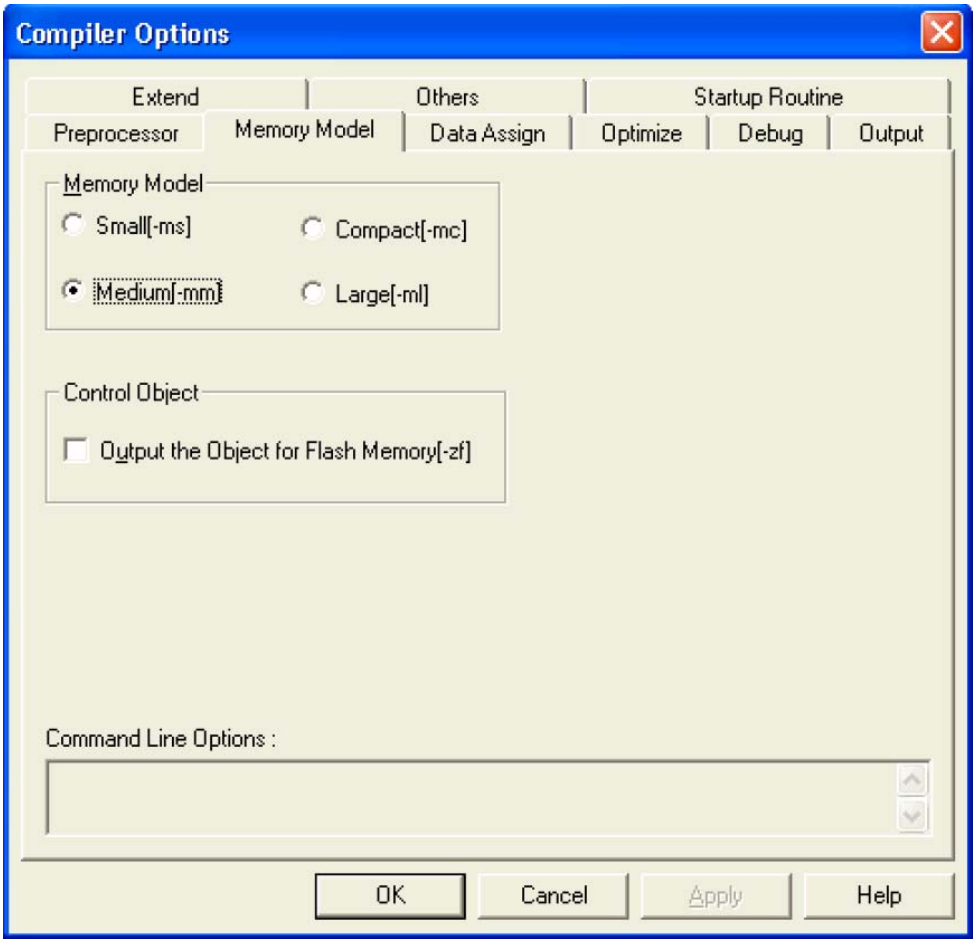
- 定义宏[-d]
  需要用-D 选项来指定的宏名称和定义名，输入组合框即可。
  对于宏名称来说，最多一次可以定义 30 个宏名称，多个宏定义之间用“，”来区分。
  一个定义的宏名称最多可以指定为 256 字符。
  该组合框最多可以接受 7709 个字符。
  可以通过[Edit…]按钮来指定。（打开[Edit Option]对话框）
  如果有某宏定义被指定了两次，则显示错误信息。

- 未定义宏[-u]
  用-u 选项指定的宏定义都输入到组合框中。
  对于宏名称来说，最多一次可以定义 30 个宏名称，多个宏定义之间用“，”来区分。
  一个定义的宏名称最多可以指定为 256 字符。
  该组合框最多可以接受 7709 个字符。
  可以通过[Edit…]按钮来指定。（打开[Edit Option]对话框）
  如果有某个宏被反定义了两次，则显示错误信息。

- 包含查找路径[-i]:
  在这个组合框中输入的内容将被[-i]选项指定为包含文件的目录。
  最多一次可以输入 64 个目录，多个目录之间用“，”来区分。
  指定一个目录时最多接受 259 个字符。
  该组合框最多可以接受 16639 个字符。
  可以通过[Edit…]按钮来指定。（打开[Edit Option]对话框）
  如果有某个宏被反定义了两次，则显示错误信息。
  也可以通过[Edit…]按钮来指定目录。
  不能指定不存在的路径。
  如果有某个目录被指定了两次，则显示错误信息。

(2) 选择“Memory Model”标签页

图 3-8 < Compiler Options >对话框（当选择“Memory Model”界面时）



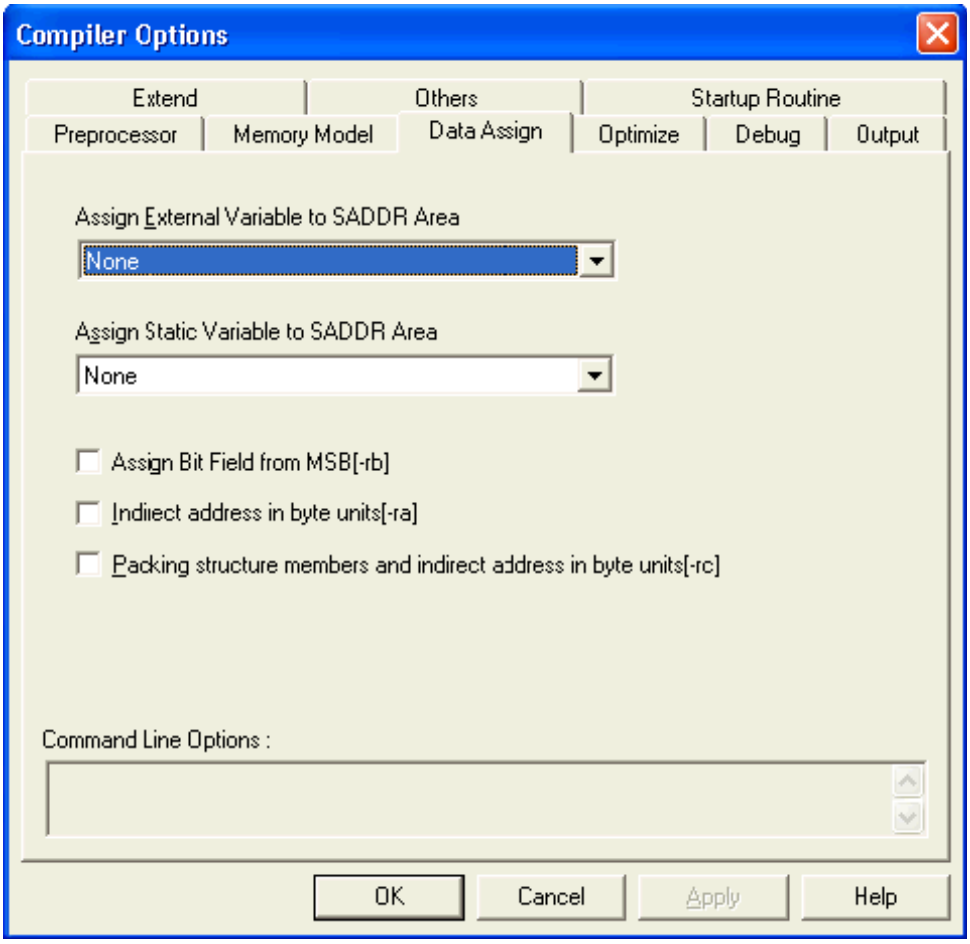
注意 如果对每个源文件都指定了某些特殊选项，则[Memory Model] 标签页有可能无法设置。

- 存储器模式
  通过单选按钮指定编译所用的存储器模式类型。

- 控制目标
  Output the Object for Flash Memory[-zf]
  选中复选框来使能-zf 选项。

(3) 选择“Data Assign”标签页

图 3-9 < Compiler Options >对话框（当选择“Data Assign”界面时）



- 将外部变量分配到 SADDR 区域
  通过该下拉框可以选择分配到 saddr 区域的外部变量类型。
  注意 如果对每个源文件都指定了某些特殊选项，则该区域无法指定。

- 将静态变量分配到 SADDR 区域
  通过该下拉框可以选择分配到 saddr 区域的静态变量类型。

- 将局部变量分配到 SADDR 区域[仅静态模式有效]
  选中该复选框来指定-RK 选项。
  用单选按钮可以选择分配到 saddr 区域的局部变量类型。

- 从 MSB 开始分配位域 [-rb]
  选中该复选框来指定-RB 选项。

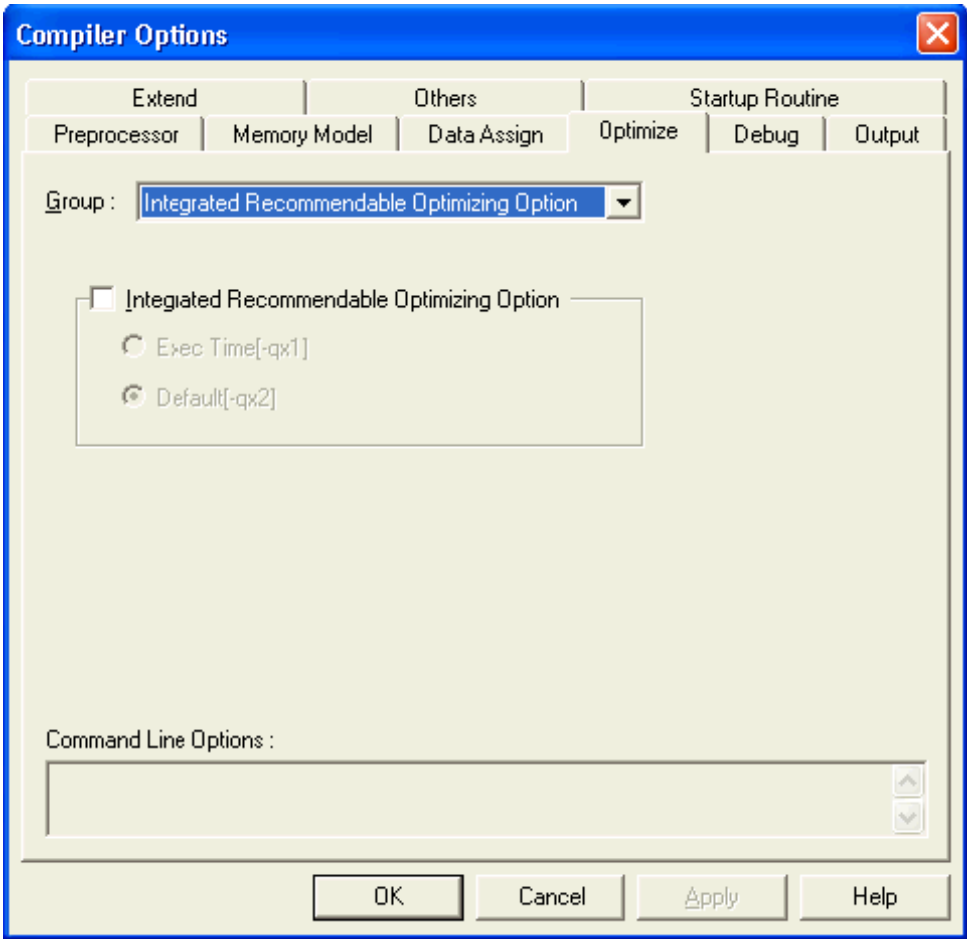
- 按字节单元间接寻址[-ra]
  选中该复选框来指定-RA 选项。

- 结构成员打包[-rc]
  选中该复选框来指定-RC 选项。

(4) 选择“Optimize” 标签页

(1) 在[Group:]下拉菜单中选择“Integrated Recommendable Optimizing Option”时的界面如下：

图 3-10 < Compiler Options >对话框（当选择“集成推荐优化选项”界面时）



- 综合推荐优化选项

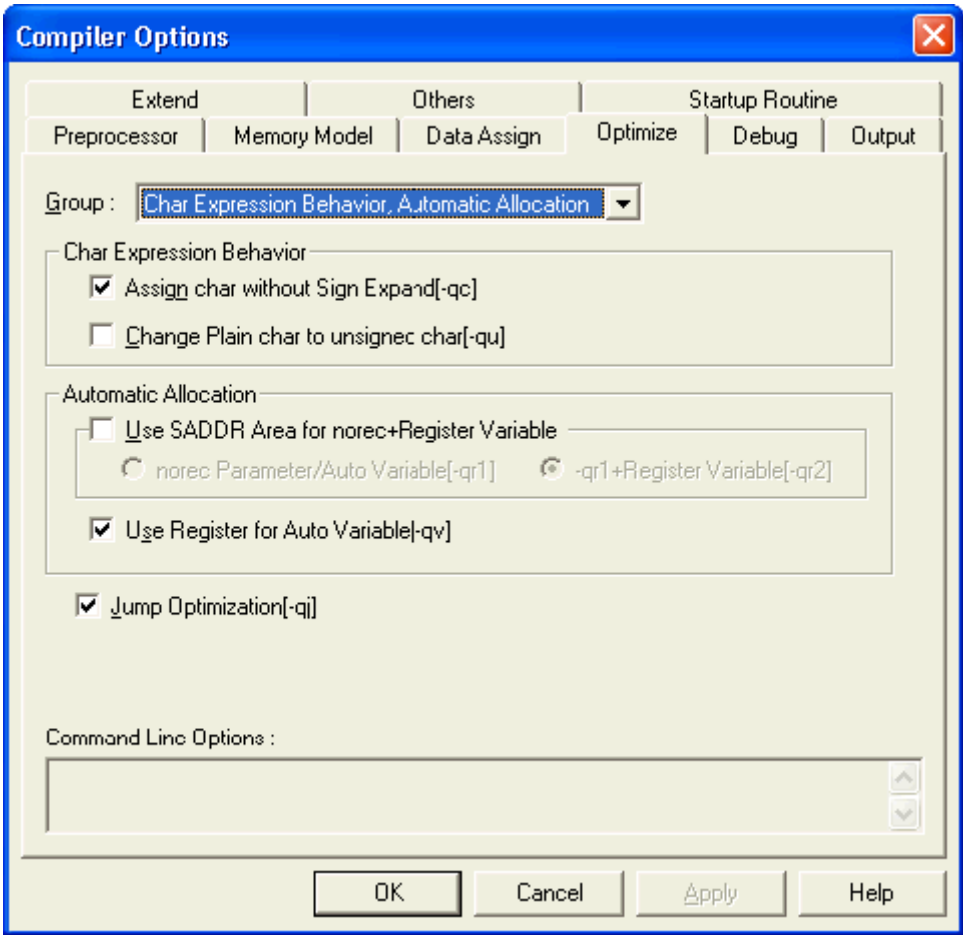
“Integrated Recommendable Optimizing Option”综合优化选项是根据目的来进行优化，而无需单独去指定，这样使得优化参数更方便设置。

共有两个设置项：“执行时间（Exec Time）[-qx1]”，“默认（Default）[-qx2]”。各自的含义如下：

Exec Time[-qx1]: -QX1 选项。看重执行速度效率时，请选择该项。
Default[-qx2]: -QX2 选项。当执行效率和目标代码大小都很重要时，请选择这个选项。

(2) 在[Group:]下拉菜单中选择“Char Expression Behavior, Automatic Allocation”时的界面如下：

图 3-11 < Compiler Options >对话框（当选择“Char Expression Behavior, Automatic Allocation”界面时）



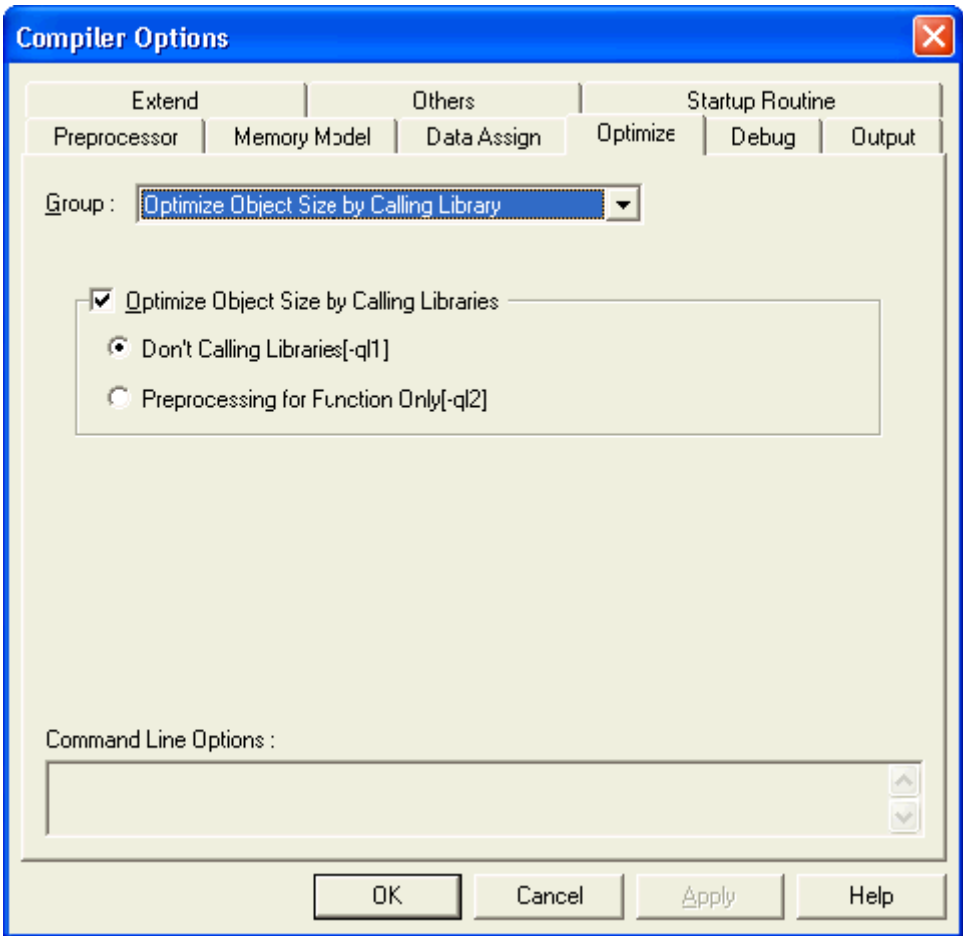
- 字符表达式动作
  指定字符为无符号扩展[-qc]
  选中该复选框来指定-QC 选项。（不执行整型提升时）。
  将无格式字符型变换为无符号字符型[-qu]
  选中该复选框来指定-QU 选项。

- 自动分配
  norec 函数+寄存器变量可以使用 SADDR 区域[-qr]
  选中该复选框来指定-QR 选项，通过选择单选按钮来指定变量类型。
  自动变量使用寄存器来传递[-qv]
  选中该复选框来指定-QV 选项。

- 跳转优化[-qj]
  选中该复选框来指定-QJ 选项。

(3) 在[Group:]下拉菜单中选择“Optimize Object Size by Calling Library”时的界面如下：

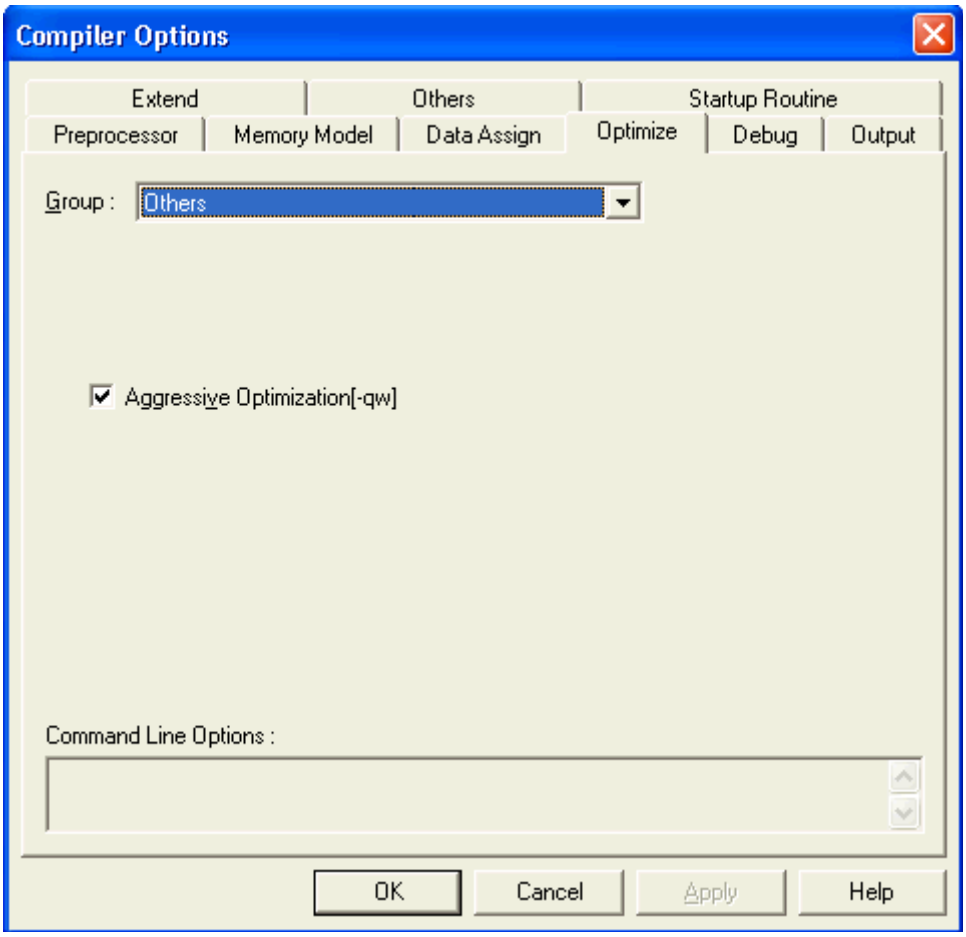
图 3-12 < Compiler Options >对话框（当选择“Optimize Object Size by Calling Library”界面时）



- 通过调用库来对目标程序大小进行优化
  选中该复选框来指定-QL 选项，并点击单选按钮来指定目标优化的优先级别。当-QLn 中的数字 n 越大，目标程序代码就越小，同时执行速度也会越慢。

(4) 在[Group]下拉菜单中选择“Others” 时的界面如下：

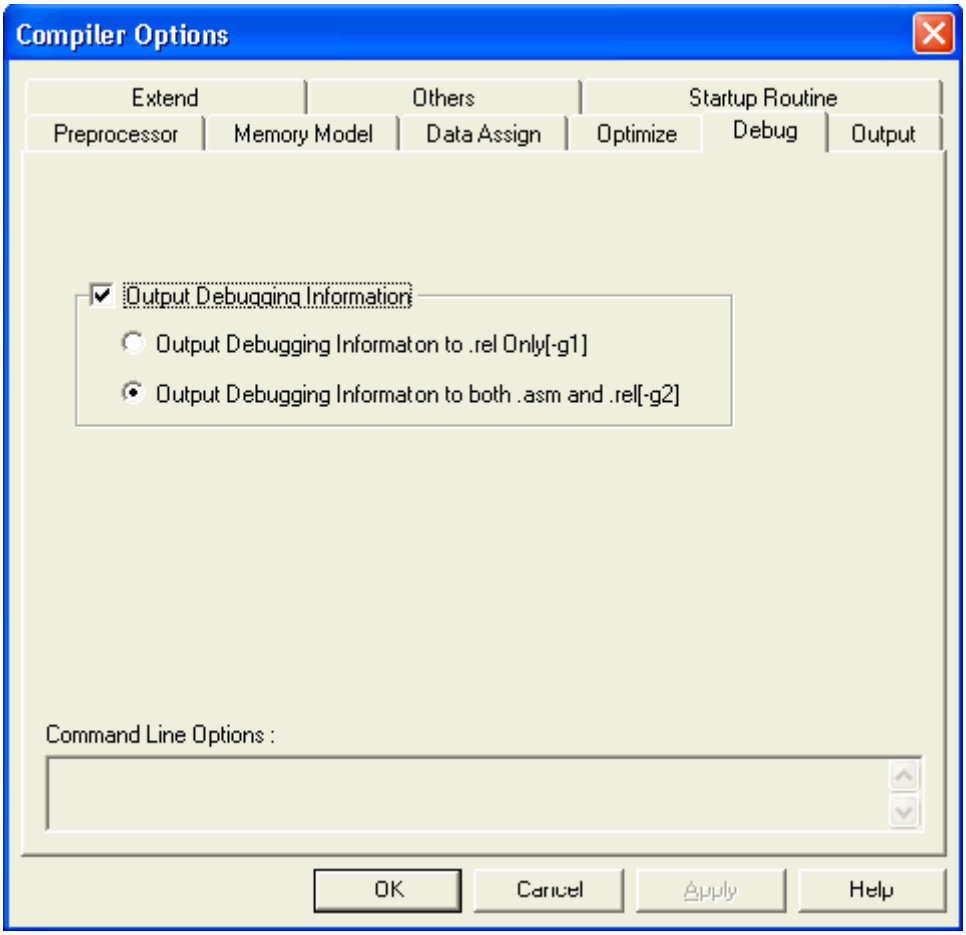
图 3-13 < Compiler Options >对话框（当选择“Others”界面时）



- 积极的优化
  选中该复选框来指定-QW 选项。

(5) 选择“Debug” 标签页

图 3-14 < Compiler Options >对话框（当选择“Debug”界面时）

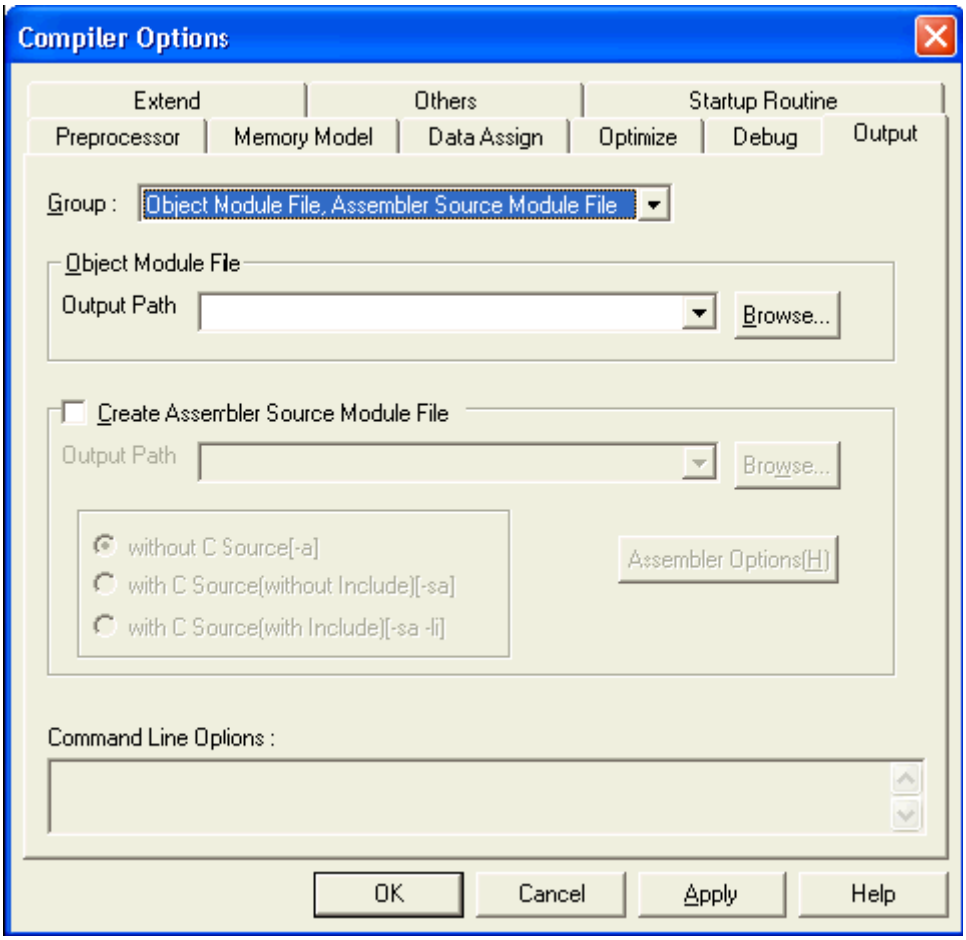


- 输出调试信息
  选中该复选框来指定-G 选项，并且通过点击单选按钮选择一个存放调试输出信息的文件。

(6) 选择“Output” 标签页

(1) 在[Group:]下拉菜单中选择"Object Module File, Assembler Source Module File"时的界面如下：

图 3-15< Compiler Options >对话框（当选择"Object Module File, Assembler Source Module File" 界面时）



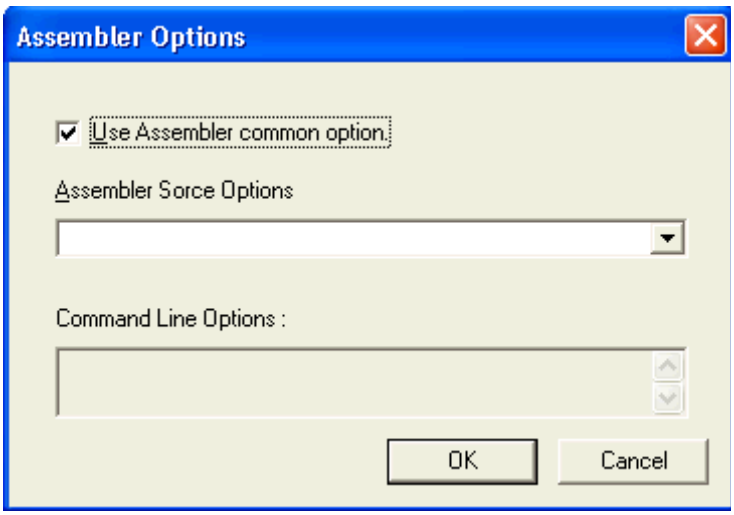
- 目标模块文件
  在这个组合框中输入的内容将被指定为目标模块文件的输出路径。
  该组合框最多可以接受 259 个字符。
  可以通过[Browse…]按钮来指定。（打开[Browse for Folder]对话框）
  在 PM +里指定了通用选项时，总是默认假定路径名已经指定。
  指定了源文件后，如果这个路径确实存在，则按照这个路径名来进行处理；如果路径不存在，则被当做文件名来进行处理。

- 创建汇编源模块文件
  选择这个复选框来使能-A/-SA/-LI 选项。可以选择在汇编程序源文件模块文件中包含/不包含 C 源程序，也可以通过对应的单选按钮来选择 C 源程序是否包含/不包含头文件。
  在组合框里输入汇编程序源模块文件的输出路径，对于指定的源文件名称，统一使用扩展后缀“asm”。
  该组合框最多可以接受 259 个字符。

可以通过[Browse...]按钮来指定。（打开[Browse for Folder]对话框）
在 PM +里指定了通用选项时，总是默认假定路径名已经指定。
指定了源文件后，如果这个路径确实存在，则按照这个路径名来进行处理；如果路径不存在，则被当做文件名来进行处理。

- [Assembler Options[H]] 按钮
  为汇编源模块文件指定汇编选项。
  如果没有指定任何选项，则进行处理时认为指定了所有的汇编器选项。
  在<Compiler Options>对话框中的<Output>标签中单击[Assembler Options[H]]按钮时，会出现以下对话框。

图 3-16 <Assembler Options>对话框



- 使用汇编器共用选项
  选择这个复选框，来使能<Assembler Options>对话框中设置的所有选项。

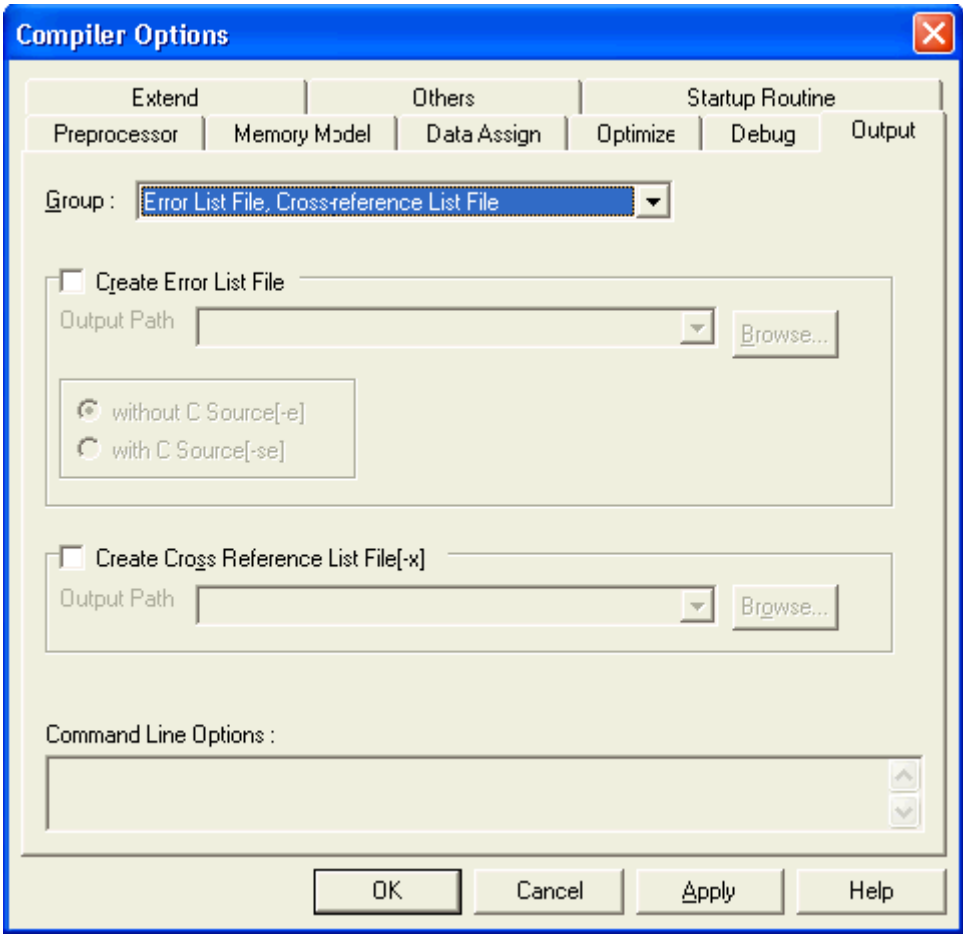
- 汇编器源文件选项
  使能编译器的输出汇编源文件选项，在组合框里输入的字符串必须包括选项名称。
  该组合框最多可以接受 259 个字符。

警告 不要描述芯片类型说明（-C）、设备文件说明（-Y）和参数文件说明（-F），因为它们和工具动态连接库是独立的。

- 命令行选项
  此编辑对话框是只读对话框。
  此编辑对话框中的字符串显示当前的选项。
  所有的汇编共用选项和输出汇编选项都是目标选项。
  所有通过点击按钮或在组合框里输入的字符串都会立即显示在此编辑对话框中。

(2) 在[Group:]下拉菜单中选择“ Error List File, Cross-reference List File”时的界面如下：

图 3-17< Compiler Options >对话框（当选择“ Error List File, Cross-reference List File”界面时）



- 创建错误列表文件

  选中该复选框来指定-E/-SE 选项。可以通过选择对应的的单选按钮来决定是否将 C 源程序加入错误列表。

  在组合框里输入具体路径作为错误列表文件的输出路径。

  该组合框最多可以接受 259 个字符。

  可以通过[Browse…]按钮来指定。（打开[Browse for Folder]对话框）

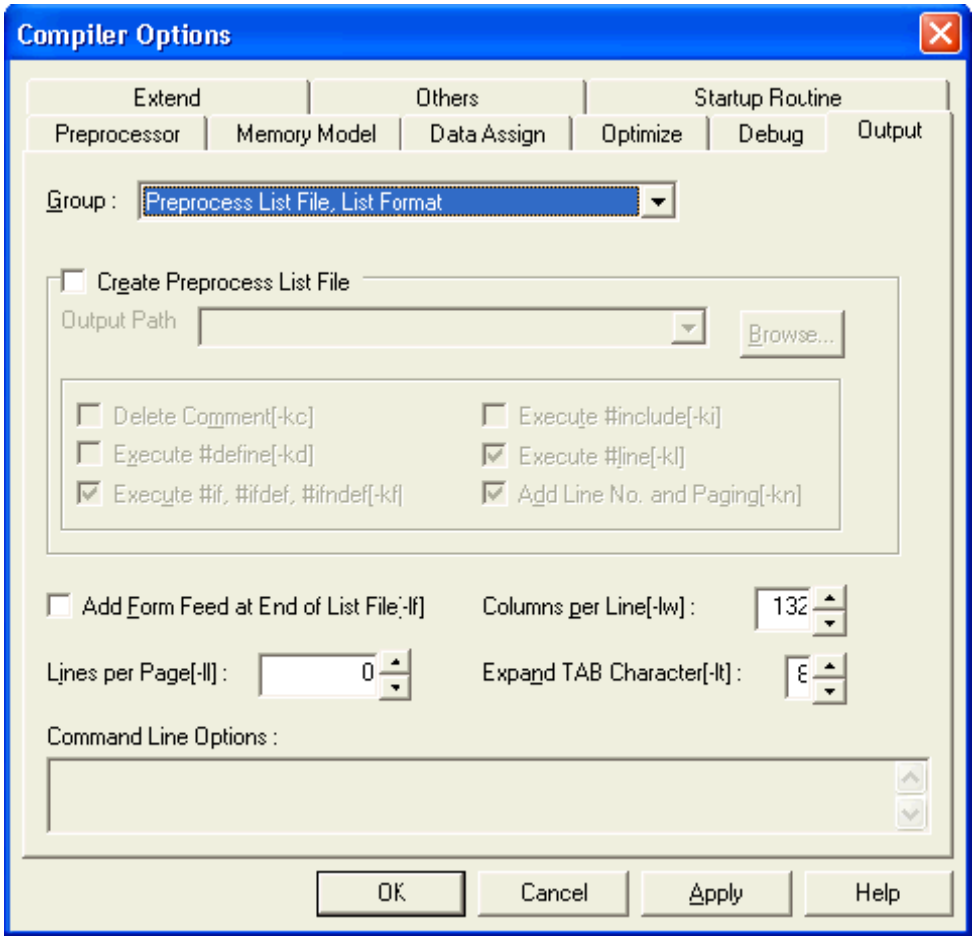
  在 PM +里指定了通用选项时，总是默认假定路径名已经指定。

  指定了源文件后，如果这个路径确实存在，则按照这个路径名来进行处理；如果路径不存在，则被当做文件名来进行处理。

- 创建交叉引用列表文件 [-x]
  选中该复选框来指定-X 选项。在组合框里输入具体路径来指定交叉引用列表文件的输出路径。
  该组合框最多可以接受 259 个字符。
  可以通过[Browse…]按钮来指定。（打开[Browse for Folder]对话框）
  在 PM +里指定了通用选项时，总是默认假定路径名已经指定。
  指定了源文件后，如果这个路径确实存在，则按照这个路径名来进行处理；如果路径不存在，则被当做文件名来进行处理。

(3) 在[Group:]下拉菜单中选择“Preprocess List File, List Format”时的界面如下：

图 3-18 < Compiler Options >对话框（当选择“Preprocess List File, List Format”界面时）



- 创建预处理列表文件
  选中该复选框来指定-P 选项，并根据实际需要来决定预处理列表文件中的下列内容。

  删除注释[-kc]
  　选中该复选框来指定-KC 选项。
  执行 #define[-kd]
  　选中该复选框来指定-KD 选项。
  执行 #if, #ifdef, #ifndef[-kf]
  　选中该复选框来指定-KF 选项。
  执行 #include[-ki]
  　选中该复选框来指定-KI 选项。
  执行 #line[-kl]
  　选中该复选框来指定-KL 选项。
  添加行号并分页[-kn]
  　选中该复选框来指定-KN 选项。

在组合框里输入具体路径作为预处理列表文件输出路径。
该组合框最多可以接受 259 个字符。
可以通过[Browse…]按钮来指定。（打开[Browse for Folder]对话框）
在 PM +里指定了通用选项时，总是默认假定路径名已经指定。
指定了源文件后，如果这个路径确实存在，则按照这个路径名来进行处理；如果路径不存在，则被当做文件名来进行处理。

- 在列表文件的最后添加换页符[-lf]
  选中该复选框来指定-LF 选项。

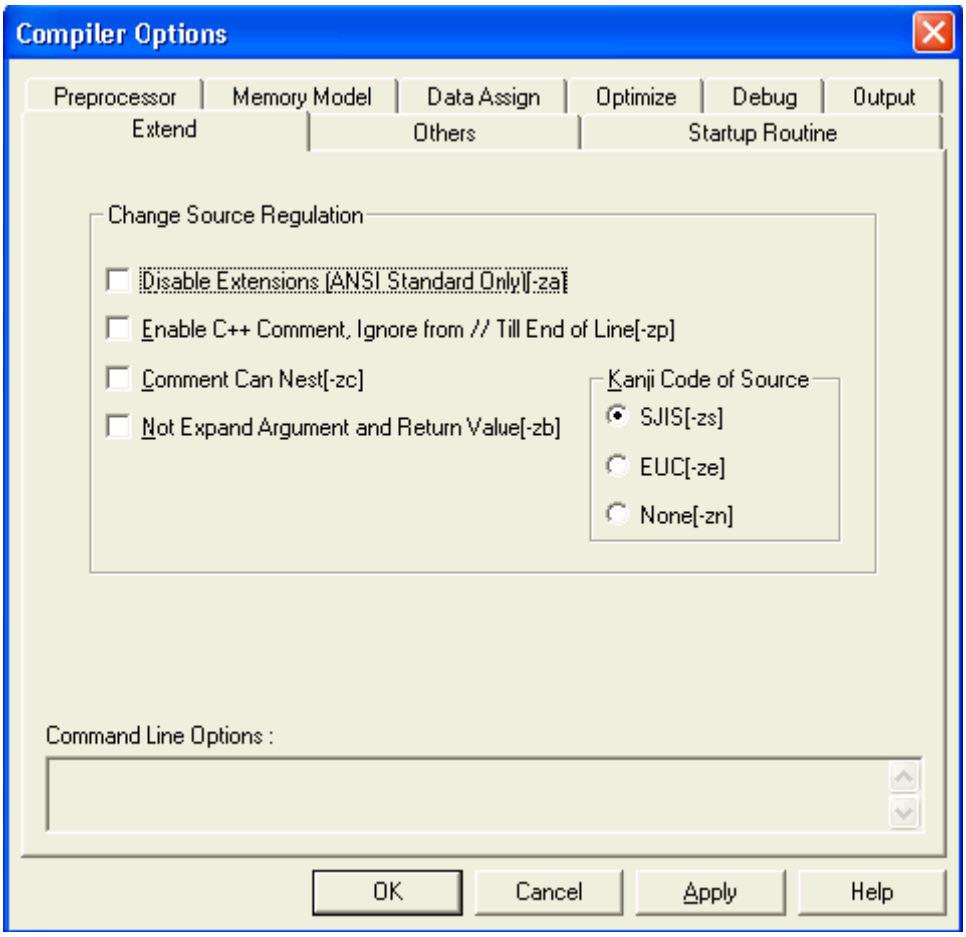
- 每行的列数 [-lw]
  使用-LW 选项指定每一行字符的数量。
  可以指定的字符数量是 0 和 72 至 132。

- 每页的行数 [-ll]
  使用-LL 选项来指定一页中的行数。
  可以指定的行数是 0 和 20 至 32767。

- 扩展 TAB 字符 [-lt]
  使用-LT 选项指定 tab 字符的跨度。
  可以指定的 TAB 字符跨度是 0 至 8。

(7) 选择 “Extend” 标签页

图 3-19 < Compiler Options >对话框（当选择“Extend”界面时）



- 改变源文件的规则

禁止扩展(仅符合 ANSI 标准)[-za]

选中该复选框来指定-ZA 选项。

允许 C++注释格式，忽略//到行末尾的内容[-zp]

选中该复选框来指定-ZP 选项。

注释可以嵌套[-zc]

选中该复选框来指定-ZC 选项。

没有扩展参数和返回值[-zb]

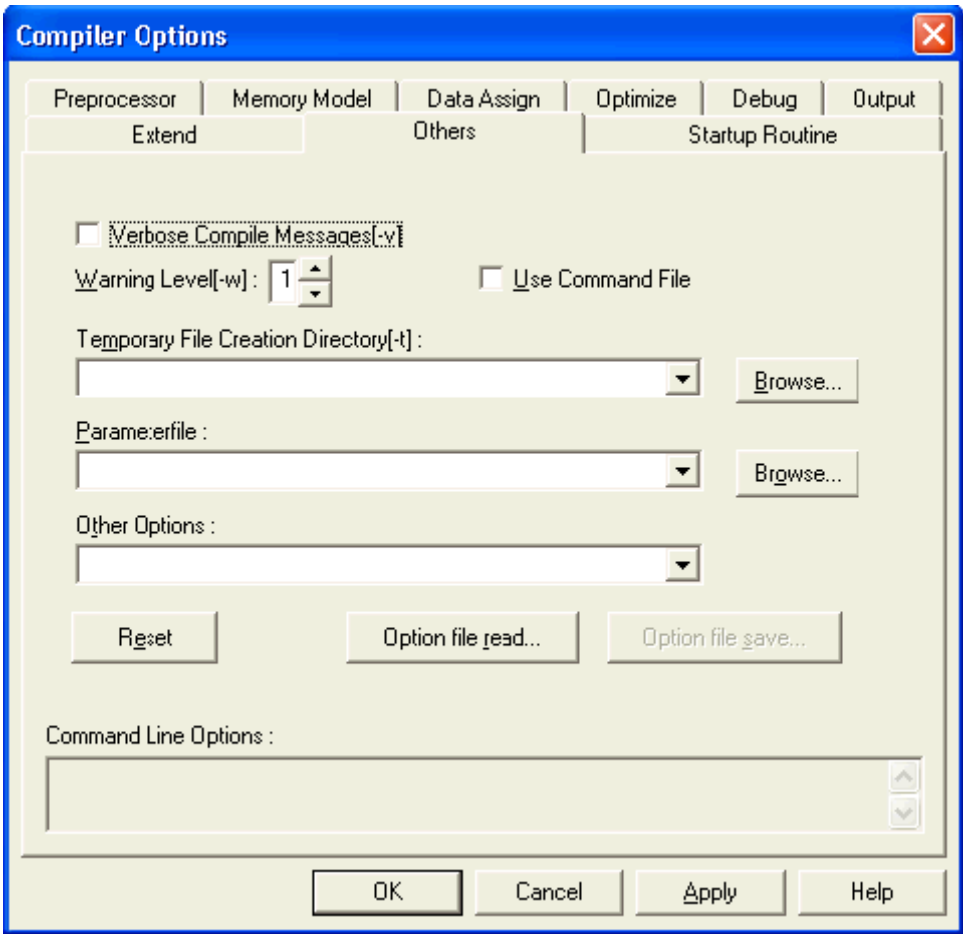
选中该复选框来指定-ZB 选项。

源文件中的日文汉字

选择对应的按钮来指定在源文件的注释中使用的日文汉字编码（2 字节编码）类型，可以为 SJIS/EUC/None 类型。

(8) 选择“Others”标签页

图 3-20 < Compiler Options >对话框（当选择“Others”界面时）



- 冗余编译信息[-v]
  选中该复选框来指定-V 选项。

- 警告级别[-w]
  使用-V 选项来指定警告级别。
  可以指定的级别范围是 0-2。

级别	描述
0	不输出警告信息
1	输出普通的警告信息
2	输出详细的警告信息

- 使用指令文件
  选中这个复选框，选项字符串就会输出到命令文件中，所以无需关心选项字符串的长度。
  注意 如果对每个源文件都指定了某些特殊选项，则该复选框无法指定。

- 临时文件存放目录[-t]:
  在组合框里输入文件夹名称会被-t 选项指定为存放临时文件的目录。
  在该组合框内只能指定一个文件夹目录。
  该组合框最多可以接受 259 个字符。
  可以通过[Browse…]按钮来指定。（打开[Browse for Folder]对话框）

- 参数文件
  用 -F 选项将组合框中输入的名称指定为参数文件名称。
  在该组合框内只能指定一个文件夹目录。
  该组合框最多可以接受 259 个字符。
  可以通过[Browse…]按钮来指定。（打开[Browse for Folder]对话框）

- 其他选项：
  如果需要指定某些规格条目之外的编译器选项，请将选项输入组合框中。
  该组合框最多可以接受 259 个字符。

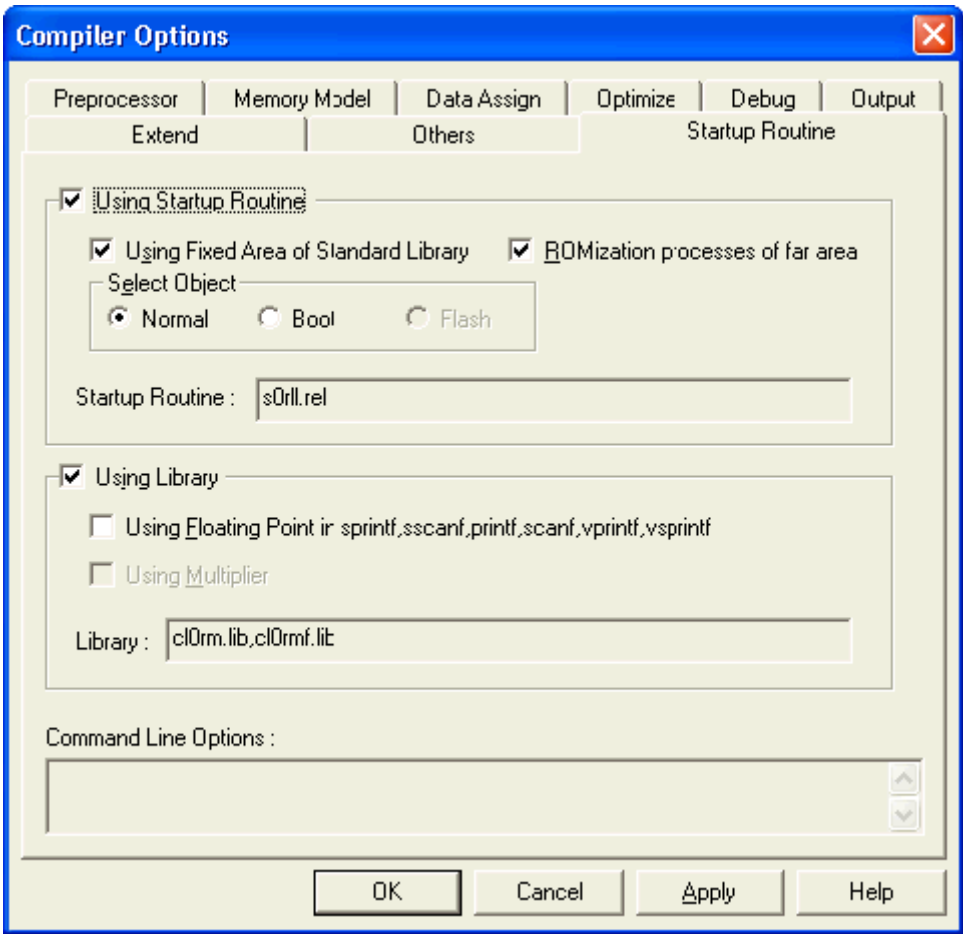
- [Reset] 按钮
  单击这个按钮恢复默认选项的设置。

- [Option file read…] 按钮
  单击这个按钮读入包含选项设置的选项信息文件。

- [Option file save…] 按钮
  选项设置被存入一个选项信息文件中。
  只有点击[OK]或[Apply]按钮之后这个按钮才有效，选项的设置被存入选项信息文件中。

(9) 选择 “Startup Routine” 标签页

图 3-21 < Compiler Options >对话框（当选择“Startup Routine”界面时）



注意 针对每个源文件指定了特殊编译参数时，<Startup Routine>标签页无法设置。

- 使用启动例程

选择这个复选框来启用 C 编译器提供的标准启动例程。

使用标准库的固定区域

选定复选框来确定标准库征用的固定区域。

ROMization 对 far 区域的处理

选定复选框来进行 ROMization 对 far 区域的处理。

选择目标

选定对应的单选按钮来指定启动例程的目标：normal，boot 区域或 flash 区域。

如果<Memory Model>标签页中的[Output the Object for Flash Memory[-zf]] 复选框未被选中，启动例程的目标可以选择普通或 boot 区域，如果这个复选框已经被选中，则启动例程的目标只能选择为 flash 区域。

启动例程：

显示将要使用的启动例程文件名称。

- 使用库

  选中复选框来启用 C 编译器提供的标准库。

  在 sprintf, sscanf, printf, scanf, vprintf, vsprintf 中使用浮点

  选择这个复选框使 sprintf, sscanf, printf, scanf, vprintf 和 vsprintf 函数支持浮点。

  使用乘法器

  选择复选框来使用产品中内嵌的乘法器。

  注意　产品类型如果没有内嵌的乘法器，则无法选择该项。

  库：

  显示使用的库文件名。

3.2 从编译到连接的过程（未使用自写入模式时）

3.2.1 从PM +中MAKE

在 PM +中进行 MAKE 的方法描述如下。

PM +是一个软件程序，作为开发环境的核心来进行工具的集成管理。使用 PM +能够把应用程序和环境设置当作工程来处理。可以使用编辑器、源文件管理、编译和调试一系列步骤来创建源程序。

（1） 启动 PM +

开发的工具包正确安装之后，在[开始] 按钮的“所有程序”中会创建[NEC Electronics Tools]菜单，并且 PM +和其他程序会注册到该菜单下。

在菜单中单击[PM +]就会启动 PM +。

（2） 创建工程

注册一个工程首先要用 PM +进行一系列的开发操作。

注册一个工程，首先创建工程管理的工作区。关于创建工作区的过程，敬请参考 PM + 用户手册。

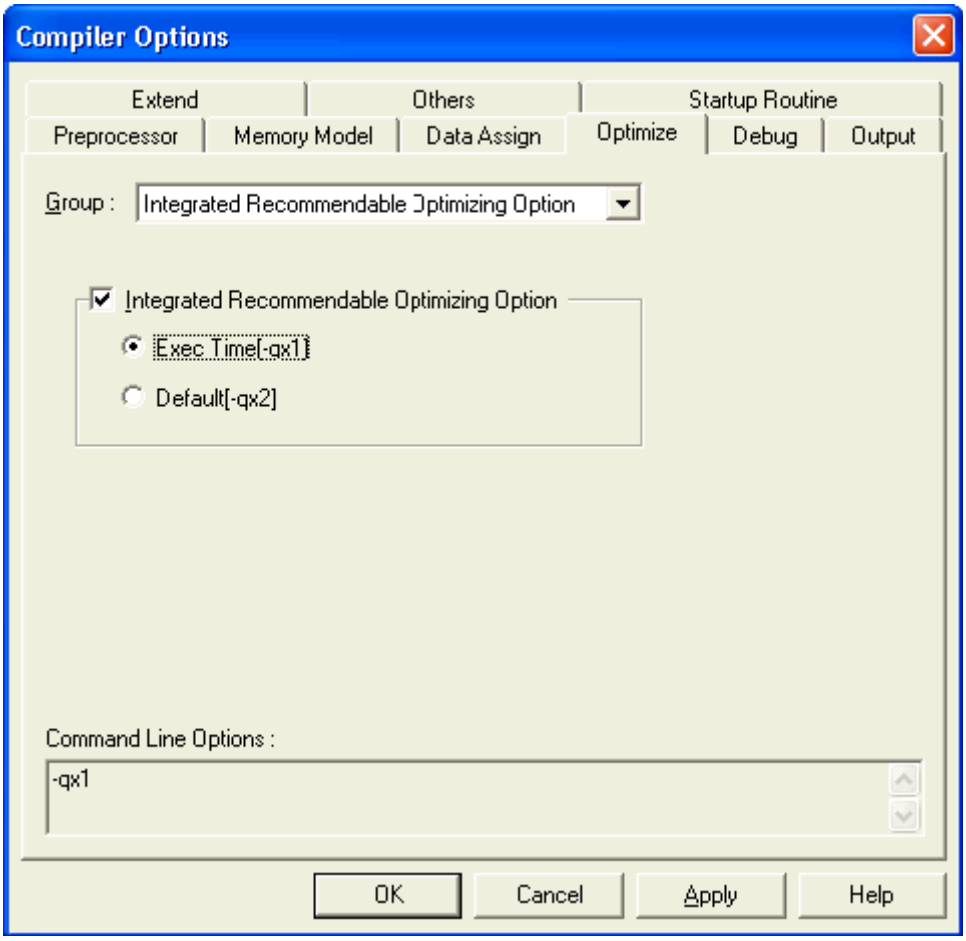
（3） 编译器和连接器的选项设定

为了能够成功建立[Build]，在工程创建之时就已经自动在 MAKE 文件中指定了最基本的必需选项。工程特定的选项在在[Tools]工具菜单中指定。

如果[Tools] 菜单下的[Compiler Options…]被选中，会出现<Compiler Options>对话框。

下面是一个将优化选项从默认的[-QCJLW] 改为“执行时间[-qx1]”的例子。

图 3-22. 优化选项的选择



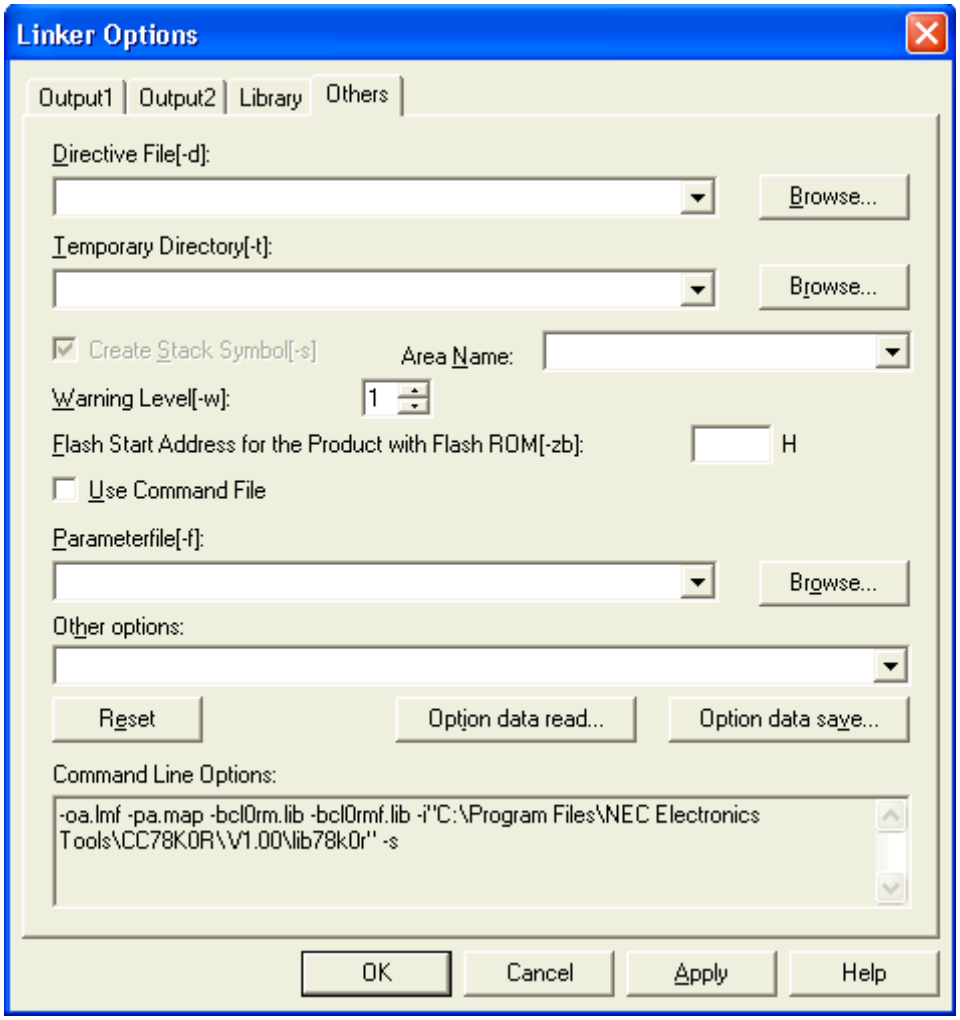
如果在<Compiler Options>对话框的<<Startup Routine>>标签页中选择了“Using Startup Routine”复选框，编译器的标准启动例程在所有源文件之前进行连接。（在<Linker Options>对话框中无对应显示）。

“使用库”被选中的情况下，编译器标准库的连接操作在所有库之后进行。

如果源文件中有 C 语言源程序设置，连接器会自动指定-S 堆栈符号自动生成选项。

启动例程文件的名称不会影响装载模块文件的名称。

图 3-23. 连接器选项对话框



（4） 建立[BUILD]工程

工程的建立要在设定好选项条件下进行。

选择[Build]菜单下的[Build]项就可以完成整个工程的建立，或者点击工具栏上的[Build]按钮。PM +的MAKE 过程会由自动生成的 MAKE 文件启动。

建立完成后，会出现一个信息对话框。查看此对话框就可以知道建立过程是否正常完成。

警告 建立时显示在<Output>窗口中的内容被保存到工程目录下，存储形式为“工程文件名称 + .plg”。

3.2.2 使用命令行来编译连接（命令提示符）

(1) 没有使用参数文件时

下列指令用来在命令行中启动 CC78K0R，汇编器和连接器。

如果 C 源文件中没有汇编语句，则无需进行汇编。在这种情况下，连接 C 编译器产生的目标模块文件。(Δ: 空格)。

```
>[path name]CC78K0R[ Δ option] ΔC 源文件名称 [Δoption]
>[path name]RA78K0R[ Δ option] Δ汇编源文件名称[Δoption]
>[path name]LK78K0R[ Δ option] Δ目标模块名称 [ Δ option]
```

注意 为了连接用户创建的库，一定要指定该库连接到 CC78K0R，并在库列表的最后加入编译器的附属库和浮点库。

要让 sprintf, sscanf, printf, scanf, vprintf 和 vsprintf 支持浮点功能，按顺序指定浮点库加入 CC78K0R 和编译器附属库。

要让 sprintf, sscanf, printf, scanf, vprintf 和 vsprintf 不支持浮点功能。按顺序指定附属库加入 CC78K0R 和编译器附带的浮点库。

在用户程序之前，指定 C 编译器附带的启动例程。在连接过程中的库文件和目标模块文件指定顺序如下所示。

(库文件的说明次序)

当 sprintf, sscanf, printf, scanf, vprintf 和 vsprintf 不支持浮点功能时

1. 用户程序库文件（用-B 选项指定）
2. C 编译器附属的库文件（用-B 选项指定）
3. C 编译器附带的浮点库文件（用-B 选项指定）

当 sprintf, sscanf, printf, scanf, vprintf, 和 vsprin 支持浮点功能时

1. 用户程序库文件（用-B 选项指定）
2. C 编译器附带的浮点库文件（用-B 选项指定）
3. C 编译器附属的库文件（用-B 选项指定）

(其他文件的说明次序)

1. CC78K0R 附带的启动例程的目标文件
2. 用户程序的目标模块文件

下面是一个连接 C 源程序 s1.c 和汇编程序源文件 s2.asm 的例子。

```
C>CC78K0R -cf1166a0 s1.c -e -a
-i"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\inc78k0r"
–y"C:\Program Files\NEC Electronics Tools\dev"
C>ra78K0R -cf1166a0 s2.asm -e
-y"C:\Program Files\NEC Electronics Tools\dev"C>lk78K0R s01.rel s1.rel s2.rel
-b"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r\cl0rxm.lib"
-b"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r\cl0rm.lib" -s
-osample.lmf -y"C:\Program Files\NEC Electronics Tools\dev"
```

备注 指定多个编译选项时，用空格来分隔。描述时使用大写小写均可（大小写不敏感）。具体细节请参见说明书**第五章编译选项**。

-i 选项，-b 选项指定的路径和 -y 选项可以根据条件进行省略。具体细节请参见**第 5 章编译器选项**和 **RA78K0R 汇编程序包用户手册 操作篇**。

(2) 使用参数文件时

在启动编译器、汇编器或连接器时输入了多个选项，如果在命令行中没有为启动提供充分的信息，相同的规格说明可能会重复多次。这种情况下，应该使用参数文件。

当使用参数文件，在命令行指定参数文件规格选项[-f]。

下面是通过参数文件来启动编译、汇编程和连接的方法。

```
>[路径名称]CC78K0R -F 参数文件名称
>[路径名称]RA78K0R -F 参数文件名称
>[路径名称]LK78K0R -F 参数文件名称
```

下面是一个使用例程。

```
例          C>cc78K0R     -Fpara.pcc
            C>ra78K0R     -Fpara.pra
            C>lk78K0R     -Fpara.plk
```

从编辑器创建参数文件。所有应该在命令行中指定的选项和输出文件名称都可以写入参数文件。

下面是程序员在编辑器中创建参数文件的一个实例。

< para.pcc 的内容>

```
-cf1166a0 s1.c -e -a
-i"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\inc78k0r"
-y"C:\Program Files\NEC Electronics Tools\dev"
```

< para.pra 的内容>

```
-cf1166a0 s2.asm -e   -y"C:\Program Files\NEC Electronics Tools\dev"
```

< para.plk 的内容>

```
s0rll.rel   s1.rel   s2.rel
-b"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r\cl0rxm.lib"
-b"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r\cl0rm.lib" -s
-osample.lmf -y"C:\Program Files\NEC Electronics Tools\dev"
```

-i选项说明， -b选项路径说明和 –y选项说明可以根据条件进行省略。具体细节请参见**第5章 编译器选项**和**RA78K0R 汇编程序包用户手册 操作篇**。

3.3 从编译到连接（当使用自写入模式时）

本功能只对那些支持 flash 存储器自编程功能的设备有效。

3.3.1 在PM +中的编译到连接

在 PM +上展示 MAKE 的技巧。
请确保按照下列顺序执行编译到连接的过程。

(1) boot 区域的编译到连接

（a） 创建一个工程

创建一个用于 boot 区域的工程，并将源文件加入到工程中。

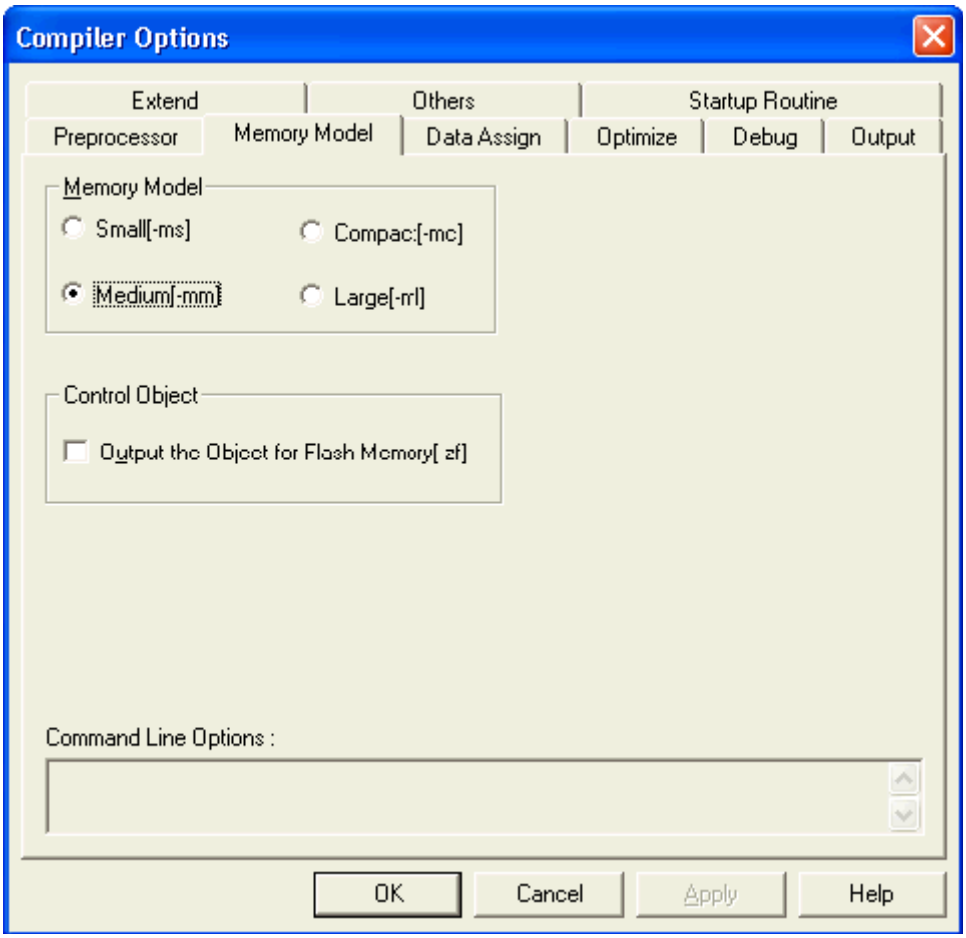
（b） 编译器、连接器和目标转换选项的设置

在工程创建结束时就已经自动在 MAKE 文件中指定了最基本的必需选项。工程特定的选项在在[Tools]工具菜单中指定。

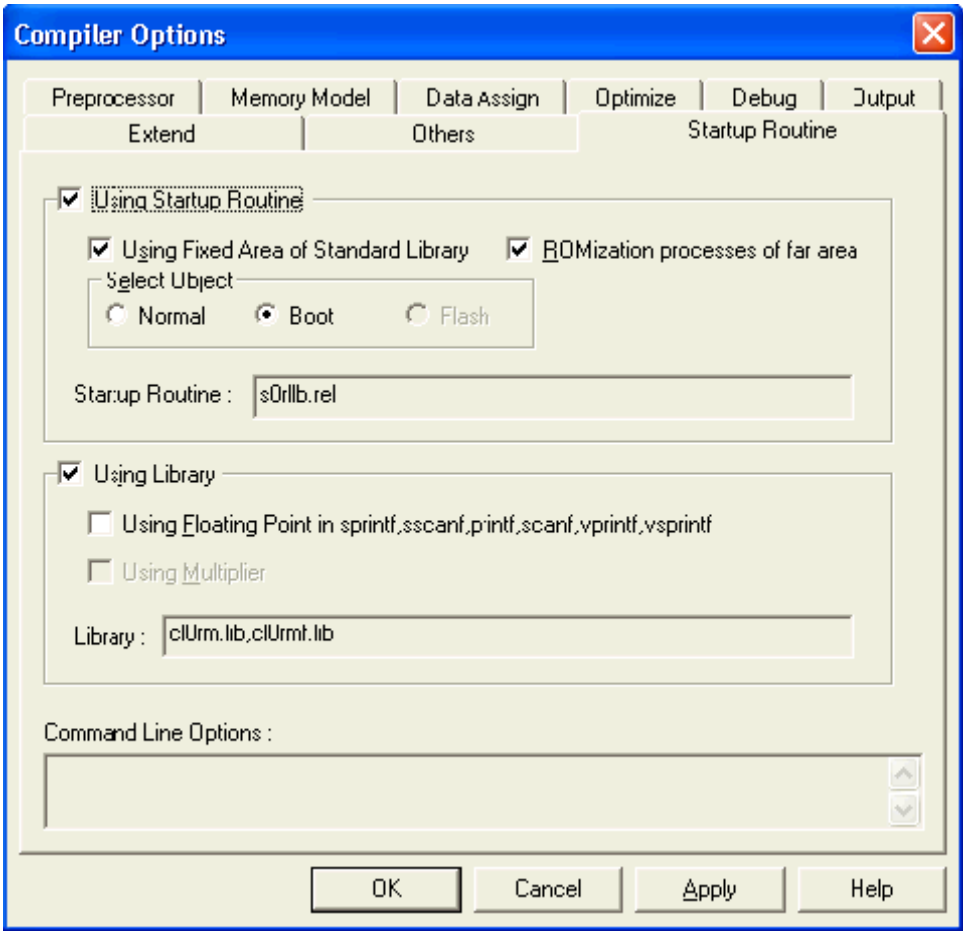
如果[Tools] 菜单下的[Compiler Options…]被选中，会出现<Compiler Options>对话框。

<i> 设置编译选项

不要指定<Memory Model>标签页下的[Output the Object for Flash Memory[-zf]]复选框。



选择<<Startup Routine>>标签页下[Select Object]中的“Boot”单选按钮。



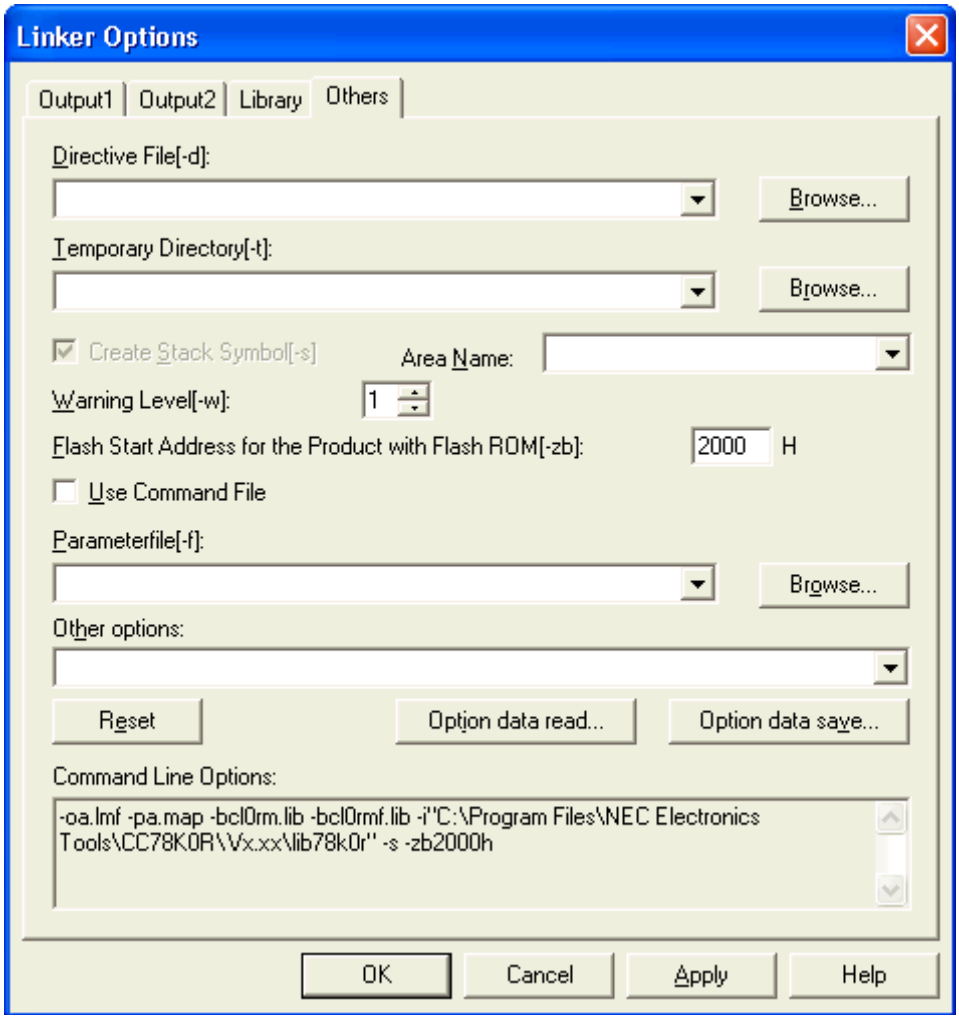
<ii> 设置连接选项

指定“内嵌 Flash ROM 的产品中 flash 启动地址[-ZB]”选项，然后点击[OK]按钮。

由于<<Startup Routine>>标签页下的“Using Startup Routine”和“Using Library”复选框已经选中，所以无需在<Linker Options>对话框中指定启动例程和库。

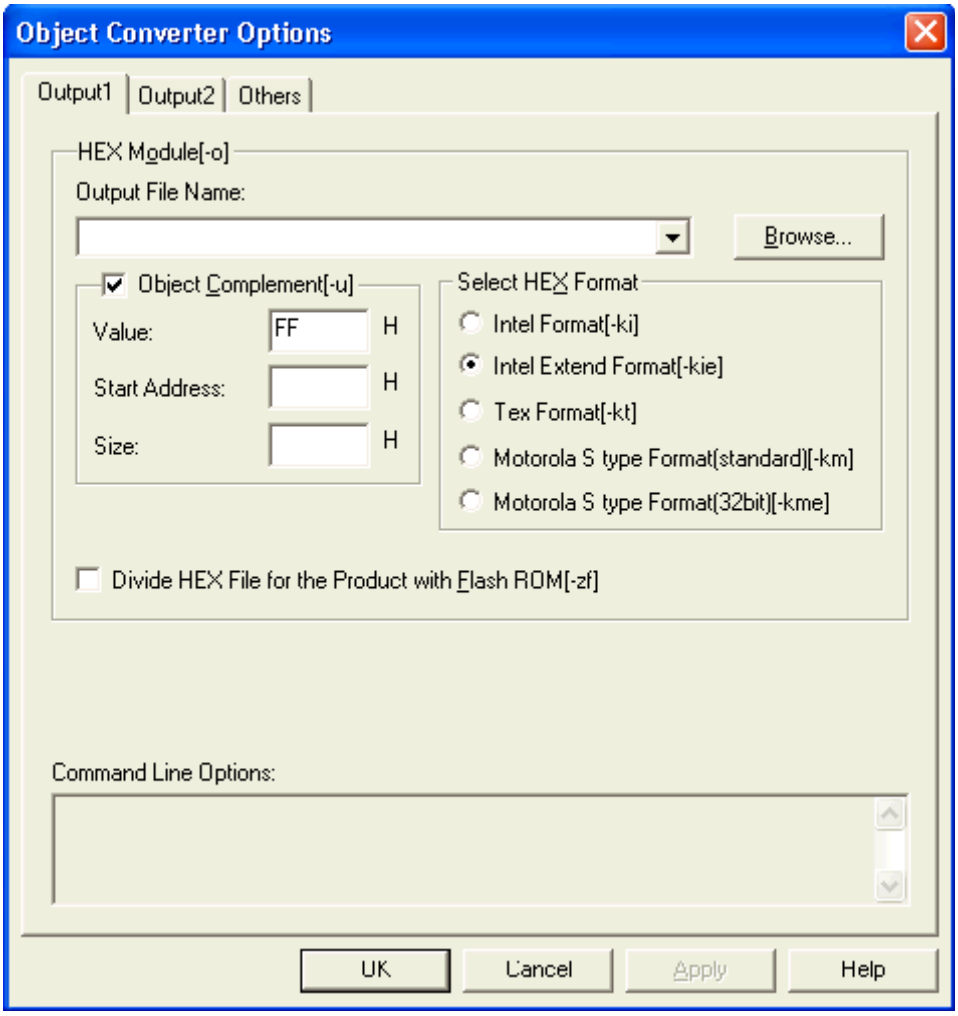
同样，由于在源文件中包括有 C 源文件（boot.c），所以“创建堆栈符号[-S]”会默认被设置。

备注 关于连接选项的信息，请参阅 **RA78K0R 汇编程序包用户手册 操作篇。**



<iii> 设置目标转换选项

不要指定[Divide HEX File for Product with Flash ROM[-zf]]复选框。



注 在 boot 区域的程序进行编译后并目标转换之后，用 flash 编程器写入 hex 文件（比如 boot.hex）。在写入后，请确保对上述过程中产生的装载模块文件（比如 boot.lmf）和 Hex 文件进行保存。不要再次建立（Build）boot 区域的程序，即不要再次生成相关文件。

（c）建立（build）工程

工程的建立是在设定的选项条件下进行的。

完成工程的完整建立之需要选中[Build]菜单下的[Build]项即可，或者点击工具栏上的[Build]按钮。PM + 的 MAKE 过程是自动创建 MAKE 文件时完成的。

在 build 完成后，会弹出一个信息框，查看信息框的信息可以知道 build 过程是否正常结束。

注意 建立时显示在<Output>窗口中的内容被保存到工程目录下，存储形式为 “工程文件名称+ .plg”。

(2) flash 区域的编译到连接

（a）创建一个工程

创建一个用于 flash 区域的工程，并将源文件加入到工程中。

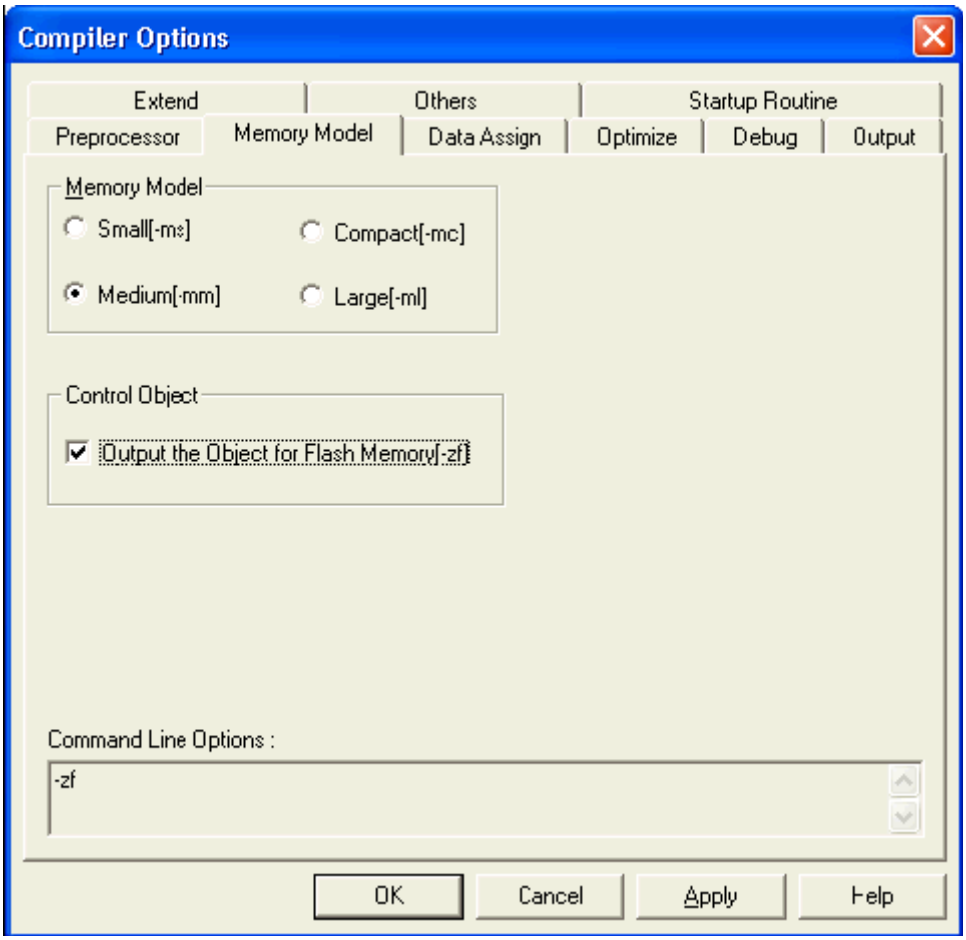
（b）编译器，连接器，和目标转换选项的设置

在工程创建完成之时就已经自动在 MAKE 文件中指定了最基本的必需选项。工程特定的选项在在[Tools]工具菜单中指定。

如果[Tools] 菜单下的[Compiler Options…]被选中，会出现<Compiler Options>对话框。

<i> 设置编译选项

请指定<Memory Model>标签页下的[Output the Object for Flash Memory[-zf]]复选框。



[Startup Routine]标签页下[Select Object]中的“Flash”单选按钮被自动选中。

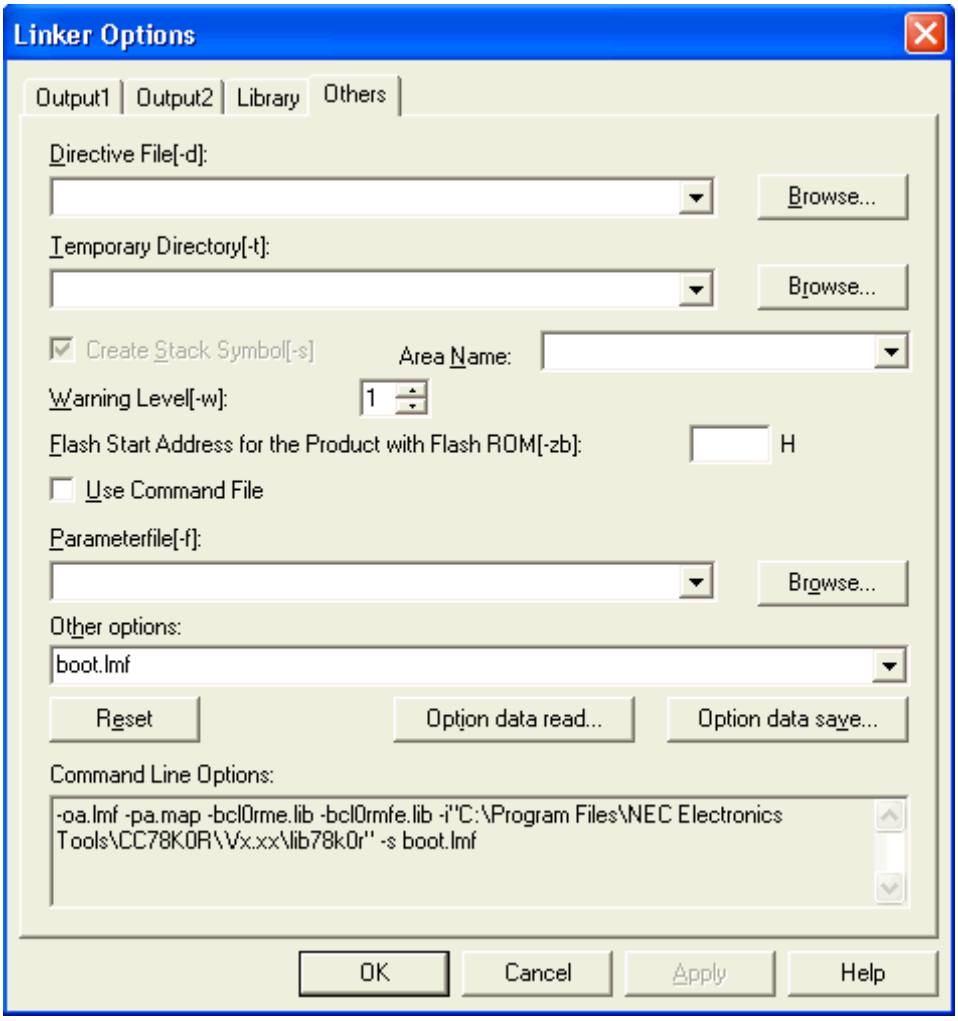
<ii> 设置连接选项

指定 boot 区域的装载模块文件，该文件由“Other options”组合框创建。

由于[Startup Routine]标签页下的“Using Startup Routine”和“Using Library”复选框已经选中，所以无需在<Linker Options>对话框中指定启动例程和库。

同样，由于在源文件中包括有 C 源文件（flash.c），所以“创建堆栈符号[-S]”会默认被设置。

备注 关于连接选项的信息，请参阅 **RA78K0R 汇编程序包用户手册 操作篇。**

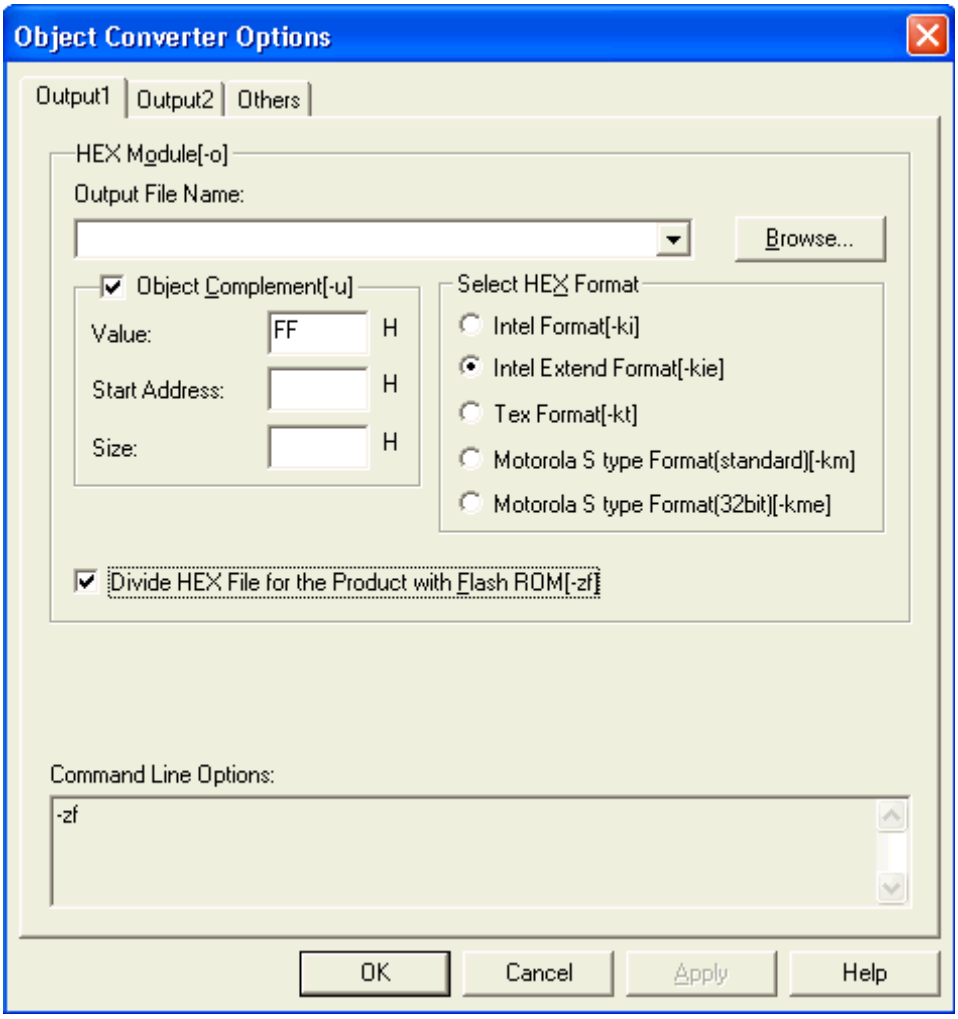


<iii> 设置目标转换选项（flash 区域）

请确保指定[内嵌 Flash ROM 的产品拆分 HEX 文件[-zf]]复选框。

因为指定了目标转换选项 –ZF，会输出 boot 区域的 HEX 文件（比如 flash.hxb）和 flash 区域的 HEX 文件（比如 flash.hxf）。

在 boot 区域的程序编译和目标转换过程中产生的 boot.hex 和 flash.hxb 具有相同的内容。但是当 boot 区域专有的 HEX 文件被写入后，并且 flash 区域专用的程序被再次建立时，推荐用户自行确认已经保存的 boot.hex 和刚生成的 flash.hxb 内容完全一致。



（c）建立（build）工程

工程的建立是在设定的选项条件下进行的。

完成工程的完整建立之需要选中[Build]菜单下的[Build]项即可，或者点击工具栏上的[Build]按钮。PM+的 MAKE 过程是自动创建 MAKE 文件时完成的。

在 build 完成后，会弹出一个信息框，查看信息框的信息可以知道 build 过程是否正常结束。

注意 建立时显示在<Output>窗口中的内容被保存到工程目录下，存储形式为 “工程文件名称+ .plg”。

3.3.2 命令行中的编译到连接（命令提示符）

(1) 没有使用参数文件时

下列指令用来启动 CC78K0R，汇编和连接都在命令行中完成。

如果 C 源文件中没有汇编语句，则无需进行汇编。在这种情况下，连接 C 编译器产生的目标模块文件。(Δ: 空格)。

```
>[path name]CC78K0R[ Δ option] Δ C 源文件名称 [Δoption]
>[path name]RA78K0R[ Δ option] Δ 汇编源文件名称 [Δoption]
>[path name]LK78K0R [ Δ option] Δ 目标模块名称，等 [ Δ option]
```

下面是 boot 区域的 C 源程序和 flash 区域的 C 源程序编译到连接的例子。

（a） boot 区域的程序编译到连接，以及目标转换过程

例<1> boot 区域程序的编译过程

```
C>cc78k0r -cf1166a0 boot.c
-i"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\inc78k0r"
-y"C:\Program Files\NEC Electronics Tools\dev"
```

例<2> boot 区域程序的连接过程

```
C>lk78k0r s0rllb.rel boot.rel
-b"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r\cl0rxm.lib"
-b"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r\cl0rm.lib"
-s -oboot.lmf -zb2000h -y"C:\Program Files\NEC Electronics Tools\dev"
```

例<3> boot 区域程序的目标转换过程

```
C>oc78k0r boot.lmf -oboot.lmf
-y"C:\Program Files\NEC Electronics Tools\dev"
```

注 在 boot 区域的程序编译并进行目标转换后，用 flash 编程器写入 hex 文件（比如 boot.hex）。在写入后，请确保对上述过程中产生的装载模块文件（比如 boot.lmf）和 Hex 文件进行保存。不要再次建立（Build）boot 区域的程序，即不要再次生成相关文件。

（b） flash 区域的程序从编译到连接过程

例<1> flash 区域程序的编译

```
C>cc78k0r -cf1166a0 flash.c -zf
-i"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\inc78k0r"
-y"C:\Program Files\NEC Electronics Tools\dev"
```

例<2>flash 区域程序的连接

```
C>lk78k0r boot.lmf s0lle.rel flash.rel
-b"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r\cl0rxm.lib"
-b"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r\cl0rm.lib"
-s -oflash.lmf -y"C:\Program Files\NEC Electronics Tools\dev"
```

例<3>flash 区域程序的目标转换

```
C>oc78k0r flash.lmf -oflash.lmf -y"C:\Program Files\NEC Electronics Tools\dev"
```

注意 在目标转换过程中指定-ZF 选项，会输出 boot 区域的 HEX 文件（比如 flash.hxb）和 flash 区域的 HEX 文件（比如 flash.hxf）。在 boot 区域的程序编译和目标转换过程中产生的 boot.hex 和 flash.hxb 具有相同的内容。但是当 boot 区域专有的 HEX 文件被写入后，并且用于 flash 区域的程序被再次建立的话，用户务必要确认保存的 boot.hex 和生成的 flash.hxb 内容完全一致。

备注 指定多个编译选项时，用空格来分隔。描述时使用大写小写均可（大小写不敏感）。具体细节请参见说明书**第五章编译选项**。

-i 选项，-b 选项指定的路径和 -y 选项可以根据条件进行省略。具体细节请参见**第 5 章编译器选项**和**RA78K0R 汇编程序包用户手册 操作篇**。

注意 为了连接用户创建的库或浮点库，一定要在库列表的最后加入 CC78K0R 编译器的附属库。当 flash 区域的程序和 boot 区域的程序相互连接时，一定要首先指定 boot 区域的装载模块文件，并在用户程序执行之前指定 flash 区域的启动例程。
下面列出连接时，库和目标模块文件的指定顺序。

- （库文件的说明次序）
  当 sprintf, sscanf, printf, scanf, vprintf 和 vsprintf 不支持浮点功能时
  (i) 用户程序库文件（用-B 选项指定）
  (ii) C 编译器附属的库文件（用-B 选项指定）
  (iii) C 编译器附带的浮点库文件（用-B 选项指定）
- 当 sprintf, sscanf, printf, scanf, vprintf, 和 vsprin 支持浮点功能
  (i) 用户程序库文件（用-B 选项指定）
  (ii) C 编译器附属的浮点库文件（用-B 选项指定）
  (iii) C 编译器附带的库文件（用-B 选项指定）

  注意 当连接 boot 区域的程序时，指定 boot 区域的库；当连接 flash 区域的程序时，需要指定 flash 区域的库

  （其他文件的说明次序）
  (i) 用户程序 boot 区域的装载模块文件。
  (ii) CC78K0R 附带的 flash 区域启动例程目标文件。
  (iii) 用户程序 flash 区域的目标模块文件。

(2) 使用参数文件时

在启动编译器、汇编器或连接器时输入了多个选项，如果在命令行中没有为启动提供充分的信息，相同的规格说明可能会重复多次。这种情况下，应该使用参数文件。

当使用参数文件，在命令行指定参数文件规格选项[-f]。

下面是通过参数文件来启动编译、汇编程和连接的方法。

```
>[path name]CC78K0R -F  参数文件名称
>[path name]RA78K0R -F 参数文件名称
>[path name]LK78K0R -F 参数文件名称
```

下面是一个使用例程。

```
C>CC78K0R       -Fpara.pcc
C>lk78K0R       -Fpara.plk
```

从编辑器创建参数文件。所有应该在命令行中指定的选项和输出文件名称都可以写入参数文件。

下面是程序员在编辑器中创建参数文件的一个实例。

< para.pcc 的内容>

```
-cf1166a0 boot.c
-i"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\inc78k0r"
-y"C:\Program Files\NEC Electronics Tools\dev"
```

< para.pra 的内容>

```
s0rllb.rel boot.rel
-b"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r\cl0rxm.lib"
-b"C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r\cl0rm.lib"
-s -oboot.lmf -zb2000h
-y"C:\Program Files\NEC Electronics Tools\dev"
```

备注 -i选项说明， -b选项路径说明和 –y选项说明可以根据条件进行省略。具体细节请参见**第5章 编译器选项**和**RA78K0R 汇编程序包用户手册 操作篇**。

3.4 C编译器的输入/输出文件

CC78K0R 的输入文件是 C 语言编写的模块文件，这些文件被转换为机器语言，再输出为目标模块文件。

编译后会输出汇编源模块文件，用户可以对汇编语言内容进行检查和修改。根据所选的编译选项，会输出对应的列表文件比如预处理文件，交叉引用文件和错误列表文件。

如果有编译错误，这个错误信息会显示在控制台，并输出到错误列表文件中。如果发生错误，那么除错误列表文件之外不会输出别的文件。

CC78K0R 输入/输出文件显示如下。

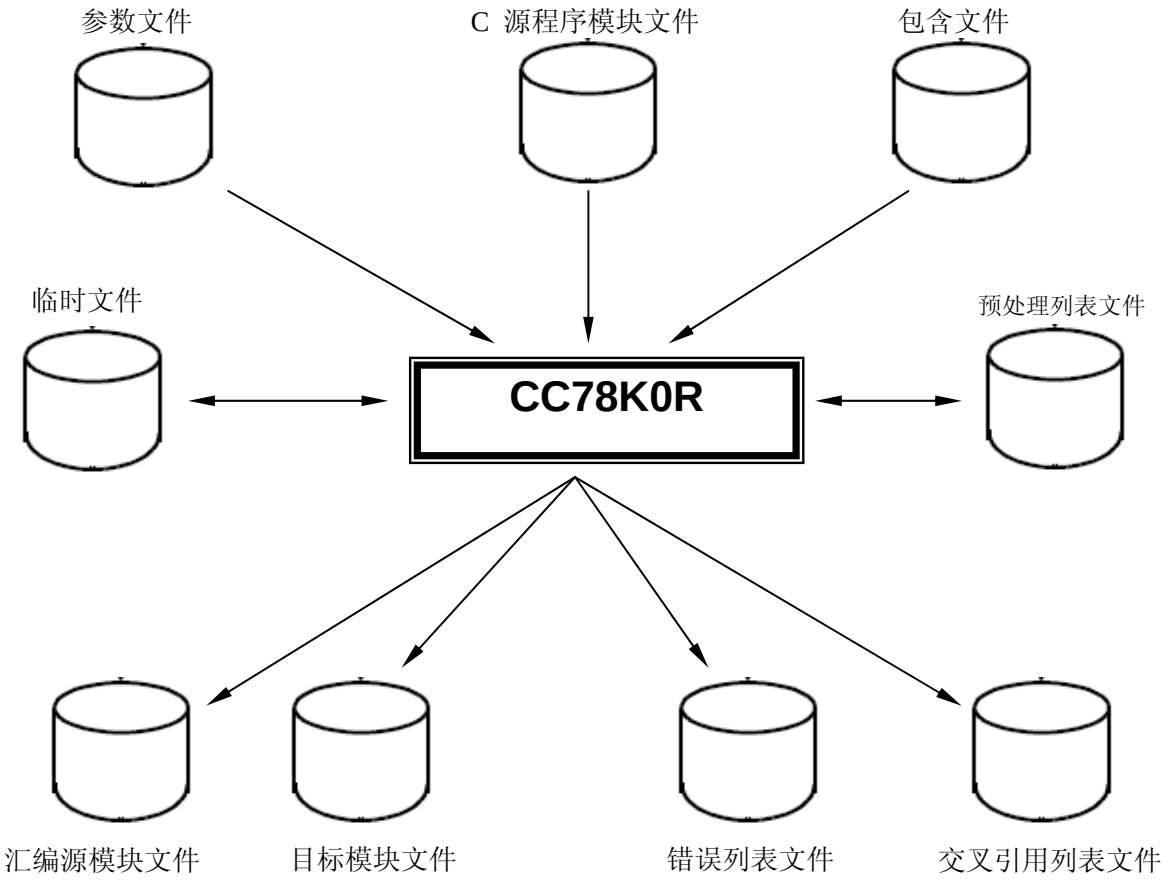
表 3-1 C 编译器 I/O 文件

类型	文件名	描述	默认文件类型
输入文件	C 源程序模块文件	• 用 C 语言编写的源文件(由用户创建的文件)	C
	包含文件	• 由 C 源模块文件引用的文件(用 C 语言编写的文件) • 由用户创建的文件	H
	参数文件	• 当用户需要指定的多条命令在C编译器运行时无法从命令行输入，自行创建的文件	PCC
输出文件	目标模块文件	• 二进制映像文件包含机器语言信息、机器语言分配地址的重定位信息，以及符号信息	REL
	汇编源模块文件	• 由编译器输出的目标代码 ASCII 映像文件	ASM
	预处理列表文件	• 像#include 文件一样的预处理指令产生的列表文件 • ASCII 映像文件	PPL
	交叉引用列表文件	• 包括C源模块文件中使用的函数名称和变量名称信息的列表文件	XRF
	错误列表文件	• 包括源文件和编译错误信息的列表文件	ECC CER HER ER [注]
I/O 文件	临时文件	• 编译产生的中间文件 • 当编译正常结束时，此文件改名为适当的名称。编译有错误，则删除此文件	$nn (固定文件名)

注 错误列表文件有以下四种文件类型。

文件类型	描述
CER	对应于*.C 文件，并带有 C 源程序的错误列表文件（指定-SE 选项输出）
HER	对应于*.H 文件，并带有 C 源程序的错误列表文件（指定-SE 选项输出）
ER	对应于上述两类文件之外的文件，并带有 C 源程序的错误列表文件（指定-SE 选项输出）
ECC	对应于所有源文件的错误列表文件，但不含有 C 源程序（指定-SE 选项输出）

图 3-34 C 编译器的输入/输出文件



备注 如果有编译错误，那么只能输出错误列表文件和交叉引用文件。
当编译正常结束没有发生错误时，临时文件就会重新改名为适当的名字。如果编译出错，那么这个临时文件就会被删除。

3.5 执行开始和结束信息

3.5.1 执行开始信息

当 CC78K0R 启动时，开始信息会显示在控制台上。

```
78K/0R Series    C Compiler Vx.xx    [xx xxx xxxx]
        Copyright (C) NEC Electronics Corporation xxxx,xxxx
```

3.5.2 执行结束信息

如果在编译过程中未发现错误，那么编译器就在控制台输出以下的信息，并将控制权交回操作系统。

```
Target chip : uPD78F1166_A0
Device file : Vx.xx

Compilation complete,       0 error(s) and 0 warning(s) found.
```

如果在编译过程中发现了错误，那么编译器就在控制台输出以下的错误信息和错误数量，并将控制权交回操作系统。

```
PRIME.C(18) : CC78K0R  警告  W0745 Expected function prototype
PRIME.C(20) : CC78K0R  警告  W0745 Expected function prototype
PRIME.C(26) : CC78K0R  警告  W0622 No return value
PRIME.C(37) : CC78K0R  警告  W0622 No return value
PRIME.C(44) : CC78K0R  警告  W0622 No return value

Target chip : uPD78F1166_A0
Device file : Vx.xx

Compilation complete,    0 error(s) and 5 warning(s) found.
```

如果在编译过程中发现一个严重的错误，无法继续编译，那么编译器就输出一个信息到控制台，停止编译并将控制权交回操作系统。

下面是一个输出错误信息的例子：

```
78K/0R Series C Compiler Vx.xx    [xx xxx xxxx]
        Copyright (C) NEC Electronics Corporation xxxx,xxxx

CC78K0R error F0018 : Option is not recognized '-s'
Please enter ‘ CC78K0R    -- ‘ , if you want help messages.
Program aborted.
```

在这个例子中，因为输入了一个不存在的编译选项(-s)，所以导致错误并停止编译。

如果编译器输出了错误信息并停止了编译，那么在**第 9 章错误信息**中可以找到这些错误提示并进行改正。

第 4 章 CC78K0R函数

4.1 优化方法

在 CC78K0R 中优化就是为了创建高效的目标模块文件。下表列出了可以支持的优化方法。

表 4-1 优化方法

编号	内容	举例
语法分析器		
<1>	常量计算在编译过程中执行	a=3*5; → a=15;
<2>	基于逻辑表达式的部分内容就可以评估真假	0 && (a || b) → 0 1 || (a && b) → 1
<3>	指针，数组等的偏移量计算	在编译过程中计算偏移量。
代码生成器		
<4>	寄存器管理	有效使用寄存器。
<5>	使用目标 CPU 中的特殊指令	a=a+1; →使用 inc 指令。 使用 move 指令来替换数组元素的访问。
<6>	使用短指令	如果在程序中有指令可以完成同样的操作，则就使用更短的指令。 mov a, #0 → clrb a
<7>	将长跳转指令换为短跳转指令	重新处理输出的中间代码。
优化器		
<8>	删除表达式的公共部分	a=b+c; → a=b+c; d=b+c+e; d=a+e;

<9>	将无关语句转移到循环体外	for (i=0; i<10; i++)
		{
		...
		a=b+c;
		...
		}
		↓
		a=b+c;
		for (i=0; i<10; i++)
		{
		...
		}

<10>	删除未使用的指令	a=a; → 删除 在 a=b;语句之后，a 没有被引用过 → 删除 (a 是一个自动变量)

表 4-1 优化方法

编号	内容	举例
<11>	删除拷贝	a=b; c=a+d; → c=b+d; a 不再被引用（a 是一个自动变量 ）。
<12>	改变表达式中的计算顺序	在其他计算执行之前，保留在寄存器中的计算结果仍有效。
<13>	存储器设备的分配(临时变量)	局部变量分配给寄存器。
<14>	窥孔优化（peephole optimizer）	代替特殊模式 例如 a*1 → a, a+0 → a
<15>	降低计算强度	例如 a*2 → a+a, a<< 1
<16>	存储器设备的分配(寄存器变量)	数据分配到高速存取的存储器。 例如：寄存器，saddr 区域 (仅当指定了-QR 选项时)
<17>	跳转优化 (-QJ 选项)	将连续跳转指令组合成一条指令。
<18>	寄存器的分配(-QV，-QR，-RD，-RS 选项)	变量自动分配给寄存器。

注意 无论优化选项如何设置，<1> 到 <7>和<14><15>项都会执行。
<8> 到<13>以及<17> 和<18>项只有在指定了对应的优化选项时才会执行。
只有在 C 源程序中有寄存器声明的情况下才会执行<16>项。但是，只有在指定了-QR 选项时才能使用 saddr 区域。
关于优化选项的信息，参见**第 5 章 编译程序选项**。

4.2 ROM化功能

ROM 化意味着初始值存放在 ROM 中，比如说带初始值的外部变量。在系统运行时这些初始值被拷贝到 RAM 中。

CC78K0R 提供了启动例程的范例，可以处理存储在 ROM 中的程序。对于 ROM 化来说，在 ROM 中使用启动例程可以忽略自行描述启动过程中的 ROM 化处理难题。

关于启动例程的信息，敬请参阅 **8.3 启动例程**。

下面描述如何将程序存储在 ROM 中。

4.2.1 连接

在连接过程中，启动例程、目标模块文件和各种库都要进行连接。启动例程会对目标程序进行初始化处理。

- s0*.rel　　启动例程（当存储在 ROM 中）
  包括数据初始化的拷贝过程，并指示初始数据的起始地址。
  标签_@cstart (符号)被当作是开始地址。

- cl0*.lib　　CC78K0R 附属库。
  CC78K0R 库文件包括以下两种。

  <1> 运行时刻（Runtime）库
  在运行时刻库名称的符号最前面加上@@。对于特别的库比如 cstart，符号最前面会加上_@来标记。

  <2> 标准库
  _（下划线）添加到标准库名称的符号最前面。

- *.lib　　用户创建的库。
  在标准库名称前添加”_”（下划线）标记。

注意　　CC78K0R 提供了各种启动例程和库。关于启动例程的细节，请参阅第 8 章 启动例程。关于库的细节，请参阅 2.5.1 库文件。

第 5 章 编译选项

当启动 C 编译器时，可以指定编译选项。指定的编译选项为编译器操作提供指令，并在程序执行前指示必需的信息。

编译选项不但可以单独指定，也可以同时指定多个选项。用户可以根据实际需要选择匹配的编译选项，并且适当的编译选项可以更有效的执行任务。

5.1 编译选项的指定

编译选项可以通过以下几种方法进行指定。

- 当 CC78K0R 编译器启动时在命令行中指定。
- 在 PM +的< Compiler Options>对话框中指定。
- 在参数文件中指定。

关于以上所描述的编译选项的指定方法，请参阅**第 3 章 编译到连接的过程**。

在编译选项之后可以紧跟着指定次级选项或文件名，之间必须没有间隔，比如说空格等。多个编译选项之间必须用空格来分隔。

对于编译选项来说，大写字母和小写字母没有区别。

<例>

```
CC78K0RΔ-cf1166a0Δprime.cΔ-aprime.asmΔ-qx2
```

备注 Δ: 空白，例如空格

5.2 编译选项的优先级

在下表所列的编译选项中，优先性体现在同时指定了垂直方向和水平方向的两个以上选项。

表 5-1 编译选项的优先级

	-NO	-G	-P	-NP	-D	-U	-A	-E	-X	-SA
-R	NG									
-Q	NG									
-G	NG									
-K			Δ	NG						
-D						OK				
-U					OK					
-SA							NG			
-LW			Δ				Δ	Δ	Δ	
-LL			Δ				Δ	Δ	Δ	
-LT			Δ				Δ	Δ	Δ	
-LF			Δ				Δ	Δ	Δ	
-LI										Δ

[标记 NG 的位置]

如果水平方向的选项被指定，那么垂直方向的选项无效。

<例>

```
C>cc78k0r -cf1166a0 -e sample.c -no -rd -g
```

选项-rd 和-g 无效。

[标记Δ的位置]

如果水平方向的选项没被指定，那么垂直方向的选项无效。

<例>

```
C>cc78k0r -cf1166a0 -e sample.c -p -k
```

由于指定了-p 选项，-k 选项无效。

[标记 OK 的位置]

对于水平方向的选项和垂直方向的选项，最后指定的那个选项优先。

<例>

```
C>cc78k0r -cf1166a0 -e sample.c -utest -dtest=1
```

由于-d 选项是最后指定的，-u 选项无效，-d 选项优先。

比如-O 和-NO 选项，即使 N 字母可以加在选项名称前，最后指定的选项仍然具有优先级。

<例>

```
C>cc78k0r -cf1166a0 -e sample.c -o -no
```

因为-NO 选项是最后指定，所以-O 选项无效，-NO 选项优先。

在**表 5-1 编译选项的优先级**中没有描述的选项，并不受其他选项的影响。但是，如果指定了帮助选项（--/-?-H），则所有的指定项无效。

帮助选项（--/-?-H）在 PM +中无法指定。为了在 PM +中使用帮助，请点击每个对话框中的[帮助]按钮。

5.3 类型

该编译器中的选项可以分类如下。

表 5-2 编译选项列表

类型	参数	描述
设备类型说明	-C	指定目标设备的类型
目标模块文件创建说明	-O	指定目标模件文件的输出
	-NO	
存储器分配说明	-R	指定存储器中分配的方法
	-NR	
	-RD	指定外部变量/外部静态变量在 saddr 区域中自动分配（除常量类型外）
	-NR	
	-RS	指定静态自动变量在 saddr 区域中自动分配
	-NR	
优化说明	-Q	指定优化类型
	-NQ	
调试信息输出说明	-G	指定输出 C 语言级别的调试信息
	-NG	
预处理列表文件创建说明	-P	指定预处理列表文件的输出
	-K	指定预处理列表的处理过程
预处理说明	-D	进行宏定义
	-U	取消宏定义
	-I	从指定目录读出包含文件
汇编源模块文件创建说明	-A	指定汇编源模块文件的输出
	-SA	
错误列表文件创建说明	-E	指定了错误列表文件的输出
	-SE	
交叉引用列表文件创建说明	-X	指定交叉引用列表文件的输出

表 5-2 编译选项列表

类型	参数	描述
列表格式说明	-LW	指定所有列表文件中每一行的字符数量
	-LL	指定了所有列表文件中每一页的行数
	-LT	改变在源模块文件中制表符(tab)的基本跨度
	-LF	在每个列表文件的末尾添加新的分页符
	-LI	将包含文件中的 C 源程序以注释形式添加到汇编源模块文件中
警告输出说明	-W	指定输出到控制台的警告信息等级
执行状态显示说明	-V	将当前编译的执行状态输出到控制台
	-NV	
参数文件说明	-F	从指定文件中读入设定选项或文件名
临时文件创建目录说明	-T	指定创建临时文件的驱动器和目录
帮助说明	--	将帮助信息输出到控制台
	-?	
	-H	
函数扩展说明	-Z	使能函数扩展的处理
	-NZ	
设备文件搜索路径	-Y	指定为设备文件搜寻路径的目录
存储器模式说明	-M	指定编译时所使用的存储器模式

5.4 编译选项的描述

这一部分详细介绍编译选项。

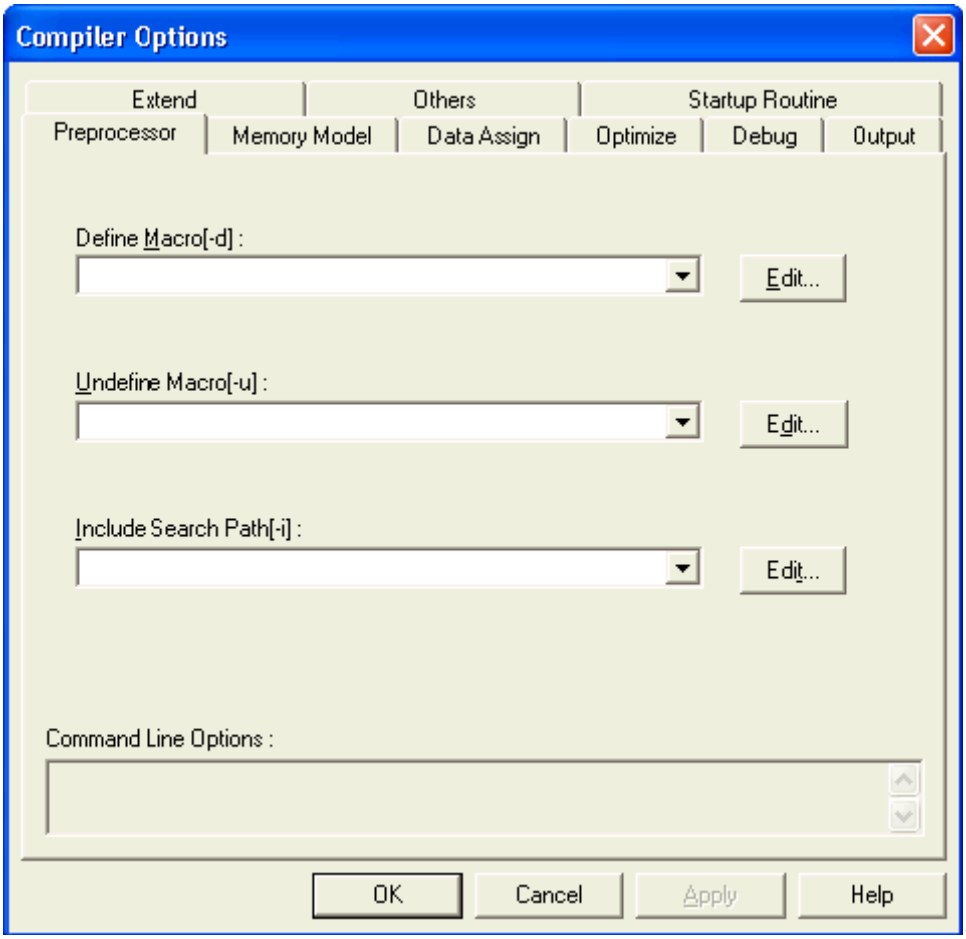
这个例子展示了如何从命令行中启动 CC78K0R。为了在 PM +中启动 CC78K0R，需要指定命令，指定设备类型，并在<Compiler Options>对话框中指定 C 源程序遗漏的选项。

[示例：（在命令行状态时）**]**

```
C>CC78K0R  -cf1166a0  prime.c  -g
```

[示例：（在使用 PM +时）**]**

图 5-1. 编译选项对话框



设备类型说明

(1) -C

[描述格式]

```
-C 设备类型
```

- 当省略时的解释
  本选项说明不能被省略。

[功能]

- -C 选项为编译过程指定了目标设备。

[应用]

- 请务必确保要指定这个选项。CC78K0R 编译器针对指定的目标设备进行编译，并为其产生目标代码。

[描述]

- 用-C 选项加相应的设备类型来说明目标设备文件的补充产品信息，请参阅所使用设备的用户手册，或“设备文件操作注意事项”。
- 当使用 CC78K0R 时，必须先安装设备文件。

[注意]

- -C 选项不能被省略，但是，如果在 C 源文件中有如下描述，那么命令行中的说明可以被忽略。

```
#pragma pc (设备类型)
```

- 如果在 C 源文件和命令行中选定了不同的设备，则命令行中选定的设备具有更高优先级。
- 当使用 PM +时，并不一定要用编译选项来设置这个选项。因为这个选项早在创建工程时就已经被默认设置。

[使用范例]

- 在命令行中指定。目标设备是μPD78f1166_a0，描述如下。

```
C>CC78K0R -cf1166a0 prime.c
```

在 C 源程序中进行说明，并启动编译器。

```
#pragma          pc(f1166a0)
#define TRUE     1
#define FALSE    0
#define SIZE     200

char   mark[SIZE+1];

main() {
        int      i,   prime,   k,   count;
        ⋮
```

因此，在命令行中也可以省略目标设备说明。

```
C>CC78K0R   prime.c
```

- 在 C 源文件和命令行中指定了不同的设备，并启动编译器。

<C 源程序>

```
#pragma          pc(f1166a0)
#define TRUE     1
#define FALSE    0
#define SIZE     200

char   mark[SIZE+1];

main() {
        int      i, prime, k, count;
```

<命令行>

```
C>CC78K0R   -cf1176   prime.c
```

在命令行执行后，编译器的执行过程如下。

```
78K/0R Series C Compiler Vx.xx   [xx xxx xxxx]
        Copyright (C) NEC Electronics Corporation xxxx,xxxx

sample\prime.c(1) :  CC78K0R 警告信息  W0832 Duplicated chip specifier
sample\prime.c(18):  CC78K0R 警告信息  W0745 Expected function prototype
sample\prime.c(20):  CC78K0R 警告信息  W0745 Expected function prototype
sample\prime.c(26):  CC78K0R 警告信息  W0622 No return value
sample\prime.c(37):  CC78K0R 警告信息  W0622 No return value
sample\prime.c(44):  CC78K0R 警告信息  W0622 No return value

Target chip : uPD78F1176
Device file : Vx.xx

Compilation complete, 0 error(s) and 6 warning(s) found.
```

目标模块文件创建说明

(1) -O/-NO

[描述格式]

```
-O [输出文件名]
-NO
```

- 当省略时的解释

  -o 输入文件名成.rel

[功能]

- -O 选项指定输出目标模件文件。此外，还能指定输出文件的目录或者输出文件名。
- -NO 选项指定不输出目标模件文件。

[应用]

- 如果要改变目标模件文件的输出文件的目录或者输出文件名，可以指定-O 选项。
- 如果编译的目标只是输出汇编源模块文件，可以指定-NO 选项，从而减少编译时间。

[描述]

- 如果指定了-O 选项却省略了输出文件名称，目标模块文件名称为“输入文件名称.rel”。
- 当-O 选项被选定时，如果没有指定输出文件扩展名，输出的目标模块文件为“输出文件名称.rel”。
- 如果出现编译错误，即使-O 选项已经被选定，目标模件文件也仍然无法输出。
- 当-O 选项被选定时，如果没有指定驱动器名称，目标模件文件会输出到当前驱动器。
- 如果-O 选项与-NO 选项二者同时被选中，那么最后选择的选项有效。

[注意]

- 当使用 PM +时，要想改变输出文件目录，需要在< Compiler Options >对话框的< Output >标签页中的< Object Module File >区域的< Output Path >组合框里选择新的输出文件目录。
- 当指定了单独选项，输出文件名也可以改变。
- 在< Output >标签下的< Output File >组合框中选定文件名或输出文件目录。

[使用范例]

- 本实例里，-NO 和-O 选项都被选中，后指定的-O 选项有效，所以会输出目标模块文件（prime.o）。

```
C>CC78K0R  -cf1166a0  prime.c  -no  -o
```

存储器分配说明

(1) (-R/-NR)

[描述格式]

-R [处理类型] (可以指定多种说明) -NR

- 当省略时的解释

  -nr

[功能]

- -R 选项指定了如何将一个程序分配到存储器中。
- -NR 选项表示-R 选项无效。

[应用]

- 如果想要指定一个程序在存储器中如何分配，选择-R 选项即可。

[描述]

- 下面列出能够用-R 选项指定的处理类型。
- 处理类型说明不能被忽略。否则，会出现异常中断错误(F0012)。

处理类型	功能
A	以字节为单位间接引用。
B	分配一个位域从最高有效位开始存放 (MSB)。
D[n][m] (n = 1, 2, 4)	将一个外部变量/外部静态变量（除了常数类型变量）自动分配到 saddr 寄存器区域，无论是否有 sreg 声明。 关于细节，请参阅“(2) – rd/-nr”
S[n][m] (n = 1, 2, 4)	将静态自动变量自动分配到 saddr 区域中，无论是否有 sreg 声明。 关于细节，请参阅“(3) – rs/-nr”
C	以字节为单位间接引用。 对结构体成员进行打包压缩，按字节对齐。

备注 可以指定多种处理类型。

- 当指定-NR 选项时，处理类型的含义如下。

处理类型	功能
A	不执行以字节为单位的间接引用。
B	分配一个位域从最低有效位开始存放 (LSB)。
D	不会将任何变量自动分配到 saddr 区域。
S	不会将任何变量自动分配到 saddr 区域。
C	不执行以字节为单位的间接引用。 不要对任何结构体成员打包压缩。

[使用范例]

- 不论是否有 sreg 声明，将外部变量或外部静态变量和静态自动变量都自动分配到 saddr 区域。

```
C>CC78K0R   -cf1166a0 -rds
```

(2) (-RD/-NR)

[内存分配说明]

格式说明 -RD[n][m] (n = 1, 2, 4) -NR

- 当省略时的解释

  -nr

[功能]

- -RD 选项能够将外部变量/外部静态变量（除常数类型变量外）自动分配到 saddr 区域。
- -NR 选项使-RD 选项无效。

[应用]

- 如果想要将外部变量/外部静态变量（除常数类型变量外）自动分配到 saddr 区域，选定-RD 选项即可，这与是否有 sreg 声明无关。

[描述]

- 待分配的变量类型会根据 n 的值而改变，是否指定了 m 也会有很大影响。

n，m 的值	待分配的变量类型
n=1	字符型，无符号字符型
n=2	字符型，无符号字符型，短型，无符号短型，整型，无符号整型，指针型
n=4	字符型，无符号字符型短型，无符号短型，整型，无符号整型，枚举型，指针型，长型，无符号长型
M	结构体，共用体和数组
忽略	所有变量

- 用 sreg 进行声明的变量总是能够自动被分配到 saddr 区域，而不管是否指定了-RD 选项。
- 通过外部声明来引用的变量将被分配到 saddr 区域。
- 通过指定该选项被分配到 saddr 区域的变量和 sreg 变量的处理方法类似。

[使用范例]

- 不论是否有 sreg 声明，将 char 或 unsigned char 类型的外部变量或外部静态变量都自动分配到 saddr 区域。

```
C>CC78K0R  -cf1166a0 –rd1
```

(3) (-RS/-NR)

[内存分配说明]

```
格式说明  -RS[n][m] (n = 1, 2, 4)
          -NR
```

- 当省略时的解释

  -nr

[功能]

- -RS 选项能够自动将静态自动变量分配到 saddr 区域。
- -NR 选项使－RS 选项无效。

[应用]

- 如果想要将静态自动变量自动分配到 saddr 区域，只需选定－RS 选项即可，这与是否有 sreg 声明无关。

[描述]

- 待分配的变量类型会根据 n 的值而改变，是否指定了 M 也会有很大影响。

n, m 的值	待分配的变量类型
n=1	字符型，无符号字符型
n=2	字符型，无符号字符型，短型，无符号短型，整型，无符号整型，指针型
n=4	字符型，无符号字符型短型，无符号短型，整型，无符号整型，列举型，指针型，长型，无符号长型
M	结构体，共用体和数组
忽略	所有变量

- 用 sreg 进行声明的变量总是能够自动被分配到 saddr 区域，而不管是否指定了-RS 选项。
- 通过指定这个选项而分配到 saddr 区域的静态自动变量，与 sreg 声明的静态自动变量按照同样的方法来处理。

[使用范例]

- 不论是否有 sreg 声明，将 char 或 unsigned char 类型的静态自动变量都自动分配到 saddr 区域。

```
C>CC78K0R   -cf1166a0 –rs1
```

优化说明

(1) (-Q/-NQ)

[描述格式]

```
-Q[优化类型] (如果需要指定多种选项，连续指定即可)
-NQ
```

- 当省略时的解释

  -QCJLVW

[功能]

- 指定-Q 选项会调用最优化方法来生成高效的目标代码。
- -NQ 选项使 -Q 选项无效。

[应用]

- 如果想要改善目标的执行速度并减少代码大小，请指定-Q 选项。
- 如果已经指定了 -Q 选项，又想要同时执行多种优化，就可以连续地指定多种优化类型。具体细节请参阅[描述]。

[描述]

- 下表列出-Q 选项能够设定的优化类型。

优化类型	过程描述
无说明	默认为 -QCJLVW。
U	将没有用修饰符定义的字符型当作无符号字符型来处理，以改善代码效率。
C	直接进行字符型计算而无需提升为整型。

计算目标	计算结果
无符号字符类型变量和无符号字符类型变量	无符号字符类型
无符号字符类型变量和有符号字符类型变量	无符号字符类型
有符号字符类型变量和有符号字符类型变量	有符号字符类型
介于-128 和 255 之间的常量和无符号字符类型变量	无符号字符类型
介于-128 和 127 之间的常量和有符号字符类型变量	有符号字符类型
带有后缀 U 介于 0 和 255 之间的常量和有符号字符类型变量	无符号字符类型

优化类型	过程描述
R[n](n = 1, 2)	向寄存器中添加一个寄存器变量，并将它分配到 saddr 区域。 用于分配寄存器变量的范围会根据 n 的值有所变化，如果 n 值被忽略，将默认为 n = 2。 1: 将 norec 参数和自动变量分配到 saddr 区域 2: 将 norec 参数和自动变量和寄存器变量分配到 saddr 区域
J	优化转移指令。
X[n] (n = 1 到 2)	根据执行速度/代码量的优先级来自动设置优化选项。 根据 n 值的不同选择不同的选项，具体如下。如果 n 被忽略,将默认 n = 2。 1: 速度优先。 认为指定了-QCJVW 选项。 2: 默认值。 认为指定了-QCJLVW 选项。
W	执行积极的优化策略。 改变表达式的执行次序来生成高效代码并提高寄存器的使用效率。
V	将参数和自动变量自动分配到寄存器或 saddr 区域。
L[n] (n = 1,2)	基于代码量优先的优化，用库来代替标准编码模式。如果未指定该选项，代码的优化就基于速度优先策略。 根据 n 值的不同来选择不同的作用域，具体如下。如果 n 被忽略,则默认 n = 1。 1: 无替代。 2: 只在处理函数之前/之后才会执行。

- 可以指定多种优化类型。
- 如果-Q 选项或优化类型被忽略，优化效果完全等同于指定-QCJLVW 选项的情况。
- 根据实际情况删除部分不需要的缺省选项来准确指定自己的选项，而无需重新指定。(例如需要指定-QR 选项→请删除-QCJLVW)。
- 如果目标模块文件和汇编源模块文件都没有输出，那么-QU 之外的选项都会无效。
- 如果-Q 和-NQ 选项同时被指定，最后指定的选项有效。
- 如果有多个-Q 选项同时被指定，最后指定的-Q 选项有效。

[使用范例]

- 因为进行了优化，所以没有用修饰符定义的字符（char）型被当作无符号（unsigned）型。

```
C>CC78K0R  -cf1166a0  prime.c  -qu
```

- 如果按照如下的方法指定-QC 和-QR 选项，-QC 选项就会无效，同时-QR 选项有效。norec 的参数，自动变量和寄存器变量都分配到 saddr 区域。

```
C>CC78K0R  -cf1166a0  prime.c  -qc  -qr
```

- 如果你想让-QC 和-QR 选项同时有效，输入如下命令。

```
C>CC78K0R  -cf1166a0  prime.c  -qcr
```

调试信息输出说明

(1) (-G/-NG)

[描述格式]

```
-G[n]   (n = 1, 2)
-NG
```

- 当省略时的解释

  -G2

[功能]

- 指定-G 选项会将调试信息添加到目标模块文件中。
- -NG 选项使-G 选项无效。

[应用]

- 如果没有指定-G 选项，目标模块文件中所需的行号和符号信息就不会被输出，目标模块文件是要被装入调试器的。因此，如果需要使用源程序来完成调试，指定-G 选项就可以保证在编译时可以连接所有的模块。

[描述]

- 因为 n 值的不同操作会有所改变。

n 值	功能
省略	默认为 n = 2.
1	仅把调试信息(以$DGS 或$DGL 开始的信息)加到目标模块文件。没有调试信息被加入到汇编源模块文件中。 使用该选项，则可以很方便的引用汇编文件。 因为调试信息被加载到其中，所以可以对目标文件中进行源程序调试。
2	将调试信息加载到目标模块文件和汇编源模块文件。

- 如果-G 和-NG 选项同时被指定，最后被指定的选项有效。
- 如果目标模块文件和汇编源模块文件都没有输出，-G 选项无效。

[使用范例]

- 将调试信息加入到目标模块文件中（prime.o）,描述如下：

```
C>CC78K0R   -cf1166a0   prime.c   -g
```

预处理列表文件创建说明

(1) (-P)

[描述格式]

```
-P [输出文件名]
```

- 当省略时的解释
  -None（没有文件被输出）

[功能]

- -P 选项指定预处理列表文件的输出。此外，指定输出目录或输出文件名称。如果-P 选项被忽略，则没有预处理列表文件输出。

[应用]

- 如果你想在执行预处理过程之后根据-K 选项的处理类型来输出源文件，或改变输出目录或预处理列表文件的输出文件名，那么要指定-P 选项。

[描述]

- 如果指定了-P 选项，输出文件名被忽略，预处理列表文件名会变成“输入文件名.ppl”。
- 如果指定了-P 选项，输出文件名后缀被忽略，预处理列表文件名会变成“输出文件名.ppl”。
- 如果指定了-P 选项时忽略了驱动器名称，预处理列表文件被输出到当前驱动器中。

[注意]

- 当使用 PM +时，要改变输出目录，需要在<Output>标签页下的<Create Preprocess List File>区域中的[Output Path]组合框里指定新的输出目录。
- 单独指定这个编译器选项，输出文件名同样可以改变。
- 在<Output>标签页下的<Output File>组合框中指定文件名或输出目录。

[使用范例]

预处理列表文件 sample.ppl 被输出。

```
C>CC78K0R  -cf1166a0  prime.c  -psample.ppl
```

(2) (-K)

[描述格式]

格式说明	-K[处理类型] (可以指定多种规格)

- 当省略时的解释

  -kfln

[功能]

- -K 选项指定预处理列表的处理过程。

[应用]

- 当注释被删除或引用了定义扩展，此时需要输出预处理列表文件，则请指定该选项。

[描述]

- 下表中列出了-K 选项指定的处理类型。

处理类型	描述
省略	同指定了－kfln 一样
C	删除注释
D	#define 扩展
F	#if, #ifdef, and #ifndef 的条件编译
I	#include 扩展
L	#line 处理
N	和指定了 FLN 等效

注意 可以同时指定多种处理类型。

- 如果没有指定-P 选项，那么-K 选项无效。
- 如果多个-K 选项同时被指定，最后指定的选项有效。

[使用范例]

- 从预处理列表文件 prime.ppl 中删除注释，执行行号处理及分页处理。

```
C>CC78K0R   -cf1166a0   prime.c   -p   -kcn
```

<输出文件示例>

```
/*
78K/0R Series C Compiler VX.XX Preprocess List
                    Date: XX XXX XXXX Page:     1

Command       : -cf1166a0 prime.c –p -kcn
In-file       : prime.c
PPL-file      : prime.ppl
Para-file     :
*/

      1 : #define TRUE     1
      2 : #define FALSE    0
      3 : #define SIZE     200
      4 :
      5 : char   mark[SIZE+1];
      6 :
      7 : main()
      8 : {
              ⋮
/*
Target chip : uPD78f1166_A0
Device file : VX.XX
*/
```

预处理说明

(1) (-D)

[描述格式]

-D 宏名[=定义名] [，宏名[=定义名]]...(可以指定多项说明)

- 当省略时的解释

  -只能在 C 源文件模块中进行宏定义。

[功能]

- 用-D 选项指定的内容和 C 源程序中#define 语句具有同样的效果，都是宏定义。

[应用]

- 如果需要用指定常量替换全部的宏名时，指定该选项。

[描述]

- 用逗号‘,’分隔每个定义内容，一次最多可以完成 30 个宏定义。
- 在紧邻‘=’ 和‘,’的前后不允许出现空格。
- 如果定义名称被忽视，编译器认为定义了“宏名称=1”。
- 如果在-D 和-U 选项中指定了相同的宏名，最后指定的选项有效。

[使用范例]

- 下例所示为 C 源文件中定义如下代码。

```
#define TEST 1
#define TIME 10
```

```
C>CC78K0R  -cf1166a0  prime.c  -dTEST,TIME=10
```

(2) (-U)

[描述格式]

-U 宏名 [，宏名]...(可以指定多个宏名)

- 当省略时的解释
  -用-D 选项指定的宏定义有效。

[功能]

- -U 选项取消宏定义，效果同 C 源程序中的#undef 语句类似。

[应用]

- 指定了该选项，则用-D 选项定义的宏名会无效。

[描述]

- 用逗号‘,’分隔每个定义内容，一次可以取消 30 个宏定义。在紧邻逗号‘,’的前后不允许出现空格。
- 通过-U 选项取消的宏定义必须是已经用-D 选项定义过的。在 C 源程序模块文件用#define 语句定义宏名或编译器的系统宏名不能用-U 选项来取消。
- 如果在-D 和-U 选项中指定了相同的宏名，最后指定的选项有效。

[使用范例]

- 通过-D 和-U 选项指定相同的宏名。在这个例子中，TEST 宏定义失效。

```
C>CC78K0R  -cf1166a0  prime.c  -dTEST  -uTEST
```

(3) (-I)

[描述格式]

-I 目录[，目录]... (可以指定多项说明)

- 当省略时的解释

  -编译器认为指定了如下目录。

  (i) 源文件目录[注1]

  (ii) 环境变量 INC78K0R 指定的目录

  (iii) C:\Program Files\NEC Electronics Tools\CC78K0R\V*x.xx*\inc78k0r [注2]

[功能]

- -I 选项的功能是从指定目录查找 C 源程序中**#include** 语句指定的包含文件。

[应用]

- 需要从某个确定目录查找包含文件时，请指定该选项。

[描述]

- 用逗号‘,’分隔每个定义内容，一次可以指定 64 个目录。
- 在紧邻逗号‘,’的前后不允许出现空格。
- 如果用-I 选项指定了多个目录，或多次使用-I 选项来指定，查找**#include** 指定的文件会按照指定的顺序来进行。
- 查找顺序如下。

  (i) 源文件目录[注1]

  (ii) 用-I 选项指定的目录

  (iii) 环境变量 INC78K0R 指定的目录

  (iv) C:\Program Files\NEC Electronics Tools\CC78K0R\V*x.xx*\inc78k0r [注2]

注 1 如果包含文件名在**#include** 语句中用“ ”(双引号)指定，首先在源文件目录中查找。如果包含文件名用**<>**指定，则不会在源文件目录中进行查找。

注2 这个实例中的CC78K0R安装目录是C:\Program Files\NEC Electronics Tools\ CC78K0R\V*x.xx*。

[使用实例]

- 为了从 D 盘根目录和 D:\sample 目录中查找 C 源文件（prime.c）中**#include** 语句指定的包含文件。

```
C>CC78K0R  -cf1166a0  prime.c  -id:, D:\sample
```

汇编源模块文件创建说明

(1) (-A)

[描述格式]

```
-A [输出文件名]
```

- 当省略时的解释
  -没有汇编源模块文件输出

[功能]

- -A 选项指定汇编源模块文件的输出。另外，还可以指定输出目录和输出文件名。

[应用]

- 如果需要改变输出目录或改变输出汇编源模块文件名，可以通过指定-A 选项来实现。

[描述]

- 磁盘文件名或者设备文件名都可以像指定文件名称一样来设定。
- 当指定了-A 选项时，如果输出文件名被忽略，则汇编源模块文件名称将变成“输入文件名.asm”。
- 当指定了-A 选项时，如果输出文件名的后缀被忽略，则汇编源模块文件名称将变成“输出文件名.asm”。
- 当指定了-A 选项时，如果驱动器名被忽略，汇编源模块文件将输出到当前驱动器。
- 如果-A 选项和 –SA 选项同时被选定，则忽略-SA 选项。

[注意]

- 在 PM +中要改变输出目录，可以在<<Output>>标签页下<< Create Assembler Source Module File >>区域的<<Output Path>>组合框中指定新的输出目录，并选择“without C Source[-a]”。
- 单独指定此选项，输出文件名也会改变。
- 在<<Output>> 标签页下的[Output File]组合框中指定文件名或者输出目录。指定的文件名扩展名为“asm”。

[使用范例]

- 创建汇编源模块文件 sample.asm 的实例。

```
C>CC78K0R  -cf1166a0  prime.c  -asample.asm
```

(2) (-SA)

[描述格式]

```
-SA [输出文件名]
```

- 当省略时的解释
  -没有汇编源模块文件输出

[功能]

- -SA 选项将 C 源文件以注释形式添加到汇编源模块文件中。另外，还指定了输出目录或者输出文件名。

[应用]

- 如果汇编源模块文件和 C 源程序模块文件都需要输出，则请指定-SA 选项。

[描述]

- 磁盘文件名或者设备文件名都可以像指定文件名称一样来设定。
- 当指定了-SA 选项时，如果输出文件名被忽略，则汇编源模块文件名称将变成“输入文件名.asm”。
- 当指定了-SA 选项时，如果输出文件名的后缀被忽略，则输出的汇编源模块文件名称将变成“输出文件名.asm”。
- 当指定了-SA 选项时，如果驱动器名被忽略，汇编源模块文件将输出到当前驱动器。
- 如果-A 选项和-SA 选项同时被选定，则忽略-SA 选项。
- 包含文件中的 C 源程序不会没有被加入到输出汇编源模块文件的注释中，但是如果指定了-LI 选项，则 C 源程序也会被添加到注释中。

[注意]

- 要在 PM +中改变输出目录，可以在<<Output>>标签下的<<Create Assembler Source Module File>>区域的<<Output Path>>组合框中选定输出目录，并选择“with C Source[without Include][-sa]”或者“with C Source[with Include][-sa -li]”。
- 单独指定此选项，输出文件名也会改变。
- 在<<Output>>标签页下的<< Output File >>组合框中指定文件名或者输出目录。指定的文件名扩展名为“asm”。

[使用范例]

- 将 C 源文件（prime.c）作为注释添加到汇编文模块文件（prime.asm）中，描述如下：

```
C>CC78K0R  -cf1166a0  prime.c  -sa
```

<输出文件示例>

```
; 78K0R Series C Compiler Vx.xx Assembler Source
;                                Date : xx xxx xxxx Time : xx : xx : xx
; Command       : -cf1166a0 prime.c -sa
; In-file       : prime.c
; Asm-file      : prime.asm
; Para-file :
$PROCESSOR ( f1166a0 )
$DEBUG
$NODEBUGA
$KANJICODE    SJIS
$TOL_INF      03FH ,      100H ,      00H ,      00H ,      00H
$DGS      FIL_NAM , .file ,   037H ,      0FFFEH ,  03FH ,     067H ,     01H ,      00H
$DGS      AUX_FIL , prime.c
$DGS      MOD_NAM , prime , 00H ,     0FFFEH ,  00H ,      077H ,     00H ,      00H
:
          EXTRN       _@RTARG0
          EXTRN       @@isrem
          PUBLIC       _printf
          PUBLIC      _putchar
          PUBLIC      _mark
          PUBLIC      _main
:
@@CODEL     CSEG
_main :
$DGL      1 , 19
          push        hl                          ; [ INF ] 1 , 1
          subw        sp , #08H                   ; [ INF ] 2 , 1
          movw        hl , sp                     ; [ INF ] 3 , 1
??bf_main :
; line 9 :   int i , prime , k , count ;
; line 10 :
; line 11 :   count = 0 ;
$DGL      0 , 4
          clrw        ax                          ; [ INF ] 1 , 1
          movw        [ hl ] , ax ; count         ; [ INF ] 1 , 1
; line 12 :
; line 13 :   for ( i = 0 ; i <= SIZE ; i++ )
$DGL      0 , 6
          movw        [ hl + 6 ] , ax ; i  ; [ INF ] 2 , 1
?L0003 :
          movw        ax , [ hl + 6 ] ; i  ; [ INF ] 2 , 1
          cmpw        ax , #0C8H ; 200            ; [ INF ] 3 , 1
          or1         CY , a.7                    ; [ INF ] 2 , 1
          skc                                     ; [ INF ] 2 , 1
          bnz   $?L0004                                   ; [ INF ] 2 , 4
        :
```

```
; *** Code Information ***
;;
; $FILE C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\smp78k0r\cc78k0r\prime.c
;
; $FUNC    main ( 8 )
;             bc = ( void )
;             CODE SIZE = 117 bytes , CLOCK_SIZE = 86 clocks , STACK_SIZE = 16 bytes
;
; $CALL    printf ( 18 )
;             bc = ( pointer : ax , int : [ sp + 2 ] )
;
; $CALL    putchar ( 18 )
;             bc = ( pointer : ax , int : [ sp + 2 ] )
;
; $CALL    putchar( 20 )
;             bc = ( int : ax )
;
; $CALL    printf ( 25 )
;             bc = ( pointer : ax , int : [ sp + 2 ] )
;
; $FUNC    printf ( 31 )
;             bc = ( pointer s : ax , int i : [ sp + 4 ] )
;             CODE SIZE = 22 bytes , CLOCK_SIZE = 20 clocks , STACK_SIZE = 14 bytes
;
; $FUNC    putchar ( 41 )
;             bc = ( char c : x )
;             CODE SIZE = 16 bytes , CLOCK_SIZE = 16 clocks , STACK_SIZE = 6 bytes

; Target chip : uPD78F1166_A0
; Device file : Vx.xx
```

错误列表文件创建说明

(1) (-E)

[描述格式]

-E [输出文件名]

- 当省略时的解释
  -没有错误列表文件被输出

[功能]

- -E 选项指定了错误列表文件的输出。此外，还指定了输出目录或输出文件名。

[应用]

- 通过-E 选项，可以更改错误列表文件的输出目录和输出文件名。

[描述]

- 磁盘文件名或者设备文件名都可以像指定文件名称一样来设定。
- 当指定了-E 选项时，如果输出文件名被忽略，则文件名称将变成“输入文件名.ecc”。
- 当指定了-E 选项时，如果输出文件名的后缀被忽略，则输出的错误列表文件名称将变成“输出文件名.ecc”。
- 当指定了-E 选项时，如果驱动器名被忽略，错误列表文件将输出到当前驱动器。
- 如果指定了-W0 选项，则不会输出警告信息。

[注意]

- 要在 PM +中改变输出目录，在<<Output>>标签页下的<<Create Error List File>>区域的<<Output Path>>组合框中指定新的输出目录，并选择“without C Source[-e]”。
- 单独指定此选项，输出文件名也会改变。
- 在<<Output>>标签页下的[Output File]组合框中指定文件名或者输出目录。

[使用范例]

- 为了输出错误列表文件（prime.ecc），描述如下：

```
C>CC78K0R   -cf1166a0   prime.c   -e
```

错误列表文件内容如下。

```
prime.c(18) : CC78K0R 警告信息  W0745 Expected function prototype
prime.c(20) : CC78K0R 警告信息  W0745 Expected function prototype
prime.c(26) : CC78K0R 警告信息  W0622 No return value
prime.c(37) : CC78K0R 警告信息  W0622 No return value
prime.c(44) : CC78K0R 警告信息  W0622 No return value

Target chip : uPD78f1166_A0
Device file : Vx.xx

Compilation complete, 0 error(s) and 5 warning(s) found.
```

(2) (-SE)

[描述格式]

-SE [输出文件名]

- 当省略时的解释
  -没有错误列表文件被输出

[功能]

- -SE 选项能够将 C 源程序添加到错误列表文件，此外还可以指定输出目录或输出文件名。

[应用]

- 如果错误列表文件和 C 源程序都需要输出，请指定-SE 选项。

[描述]

- 磁盘文件名或者设备文件名都可以像指定文件名称一样来设定。
- 当指定了-SE 选项时，如果输出文件名被忽略，则错误列表文件名称将变成“输入文件名.cer”。
- 当指定了-SE 选项时，如果输出文件名的后缀被忽略，则输出的错误列表文件名称将变成“输出文件名.cer”。
- 当指定了-SE 选项时，如果驱动器名被忽略，错误列表文件将输出到当前驱动器。
- 为包含文件指定目录和文件名时有如下限制。
  如果包含文件的文件类型是“H”，那么文件类型为“her”的错误列表文件会输出到当前驱动器。
  如果包含文件的文件类型是“C”，那么文件类型为“cer”的错误列表文件会输出到当前驱动器。
  其余所有情况，都会输出文件类型为“er”的错误列表文件。
- 如果没有任何错误，那么 C 源程序不会被添加。在这种情况下，不会为包含文件创建错误列表文件。
- 如果指定了-W0 选项，则不会输出警告信息。

[注意]

- 要在 PM +中改变输出目录，需要在[Output]标签页下的<< Create Error List File >>中的[Output Path]组合框里指定新的输出目录，并选择“with C Source[-se]”。
- 单独指定此选项，输出文件名也会改变。
- 在<< Output >>标签页下的<< Output File >>组合框中指定文件名或者输出目录。

[使用范例]

- 为了将 C 源模块文件（prime.c）添加到输出错误列表文件（prime.ecc），描述如下：

```
C>CC78K0R   -cf1166a0   prime.c   -se
```

<输出文件示例>

```
/*
78K0R Series C Compiler VX.XX Error List
                                Date : XX XXX XXXX Time : XX : XX : XX
Command         : -cf1166a0 prime.c -se
In-file         : prime.c
Err-file        : prime.cer
Para-file       :
*/
#define TRUE    1
#define FALSE   0
#define SIZE    200
char mark [ SIZE + 1 ] ;
main ( )
{
                :
        prime = i + i + 3 ;
        printf ( "%6d" , prime ) ;
*** CC78K0R warning W0745 : Expected function prototype
        count++ ;
        if ( ( count%8 ) == 0 ) putchar ( '\n' ) ;
*** CC78K0R warning W0745 : Expected function prototype
        for ( k = i + prime ; k <= SIZE ; k += prime )
                :
}
```

交叉引用列表文件创建说明

(1) (-X)

[描述格式]

-X [输出文件名]

- 当省略时的解释
  -没有交叉引用列表文件被输出

[功能]

- -X 选项指定交叉引用列表文件的输出。此外，还可以指定输出目录或输出文件名。交叉列表文件对于检查非常重要，可以检查符号引用频率、符号的定义和符号的被引用位置。

[应用]

- 如果需要输出交叉引用列表文件，或者需要改变交叉引用列表文件的输出目录/输出文件名，请指定-X 选项。

[描述]

- 磁盘文件名或者设备文件名都可以像指定文件名称一样来设定。
- 当指定了-X 选项时，如果输出文件名被忽略，则交叉引用列表文件名称将变成“输入文件名.xrf”。
- 当指定了-X 选项时，如果输出文件名的后缀被忽略，则输出的交叉引用列表文件名称将变成“输出文件名.xrf”。
- 即使有除 C101 之外的内部错误，或者发生编号为 F0024 的编译错误，或者发生以 E 开头编号的严重错误，交叉引用列表文件仍然能够被创建。但是在这种情况下无法保证文件内容的正确性。

[注意]

- 要在 PM +中改变输出目录，需要在<< Output >>标签页下的<< Create Cross Reference List File[-x]>>中的<< Output Path>>组合框里指定新的输出目录。
- 单独指定此选项，输出文件名也会改变。
- 在<< Output >>标签页下的[Output File]组合框里指定文件名或输出目录。

[使用范例]

- 指定了-X 选项。

```
C> CC78K0R  -cf1166a0  prime.c  -x
```

<输出文件示例>

```
78K/0R Series C Compiler Vx.xx Cross reference List
                                Date : XX XXX XXXX Page : 1
Command      : -cf1166a0 prime.c -x
In-file      : prime.c
Xref-file    : prime.xrf
Para-file    :
ATTRIB   MODIFY   TYPE      SYMBOL    DEFINE    REFERENCE
EXTERN   NEAR     array     mark      5         29      31      37
EXTERN   FAR      func      printf    7         33      40
REG1              pointer   s         7         13
PARAM
REG1              int       i         7         12
PARAM
REG1              int       j         9         12
REG1              pointer   ss        10        13
EXTERN   FAR      func      putchar   16        35
REG1              char      c         16        19
PARAM
REG1              char      d         18        19
EXTERN   FAR      func      main      22
REG1              int       i         24        28      28      28      29
        30      30      30      31      32      32      36
REG1              int       prime     24        32      33      36      36
REG1              int       k         24        36      36      36      37
REG1              int       count     24        26      34      35      40
                  #define   TRUE      1         29
                  #define   FALSE     2         37
                  #define   SIZE      3         5       28      30      36
Target chip : uPD78F1166_A0
Device file : Vx.xx
```

列表格式说明

(1) (-LW)

[描述格式]

```
-LW [字符数量]
```

- 当省略时的解释
  -LW132 (对于控制台的输出，这就是 80 个字符)

[功能]

- -LW 选项可以指定所有列表文件中每一行的字符数量。

[应用]

- 如果需要改变列表文件中每一行的字符数量，可以指定-LW 选项。

[描述]

- 用-LW 选项可以指定字符数量的范围，但是不包括结束符(CR，LF)，具体如下。
  72 ≤ 每行可以打印的字符数量 ≤ 132
- 如果没有指定字符数量，那么每行的字符数量会变为 132 个（如果输出到控制台，那么每行最多输出 80 个字符）。
- 如果列表文件规格没有作任何说明，那么-LW 选项无效。

[使用范例]

- 为了设置交叉引用列表文件（prime.xrf）每行的字符数量为 72 个字符，描述如下：

```
C>cc78k0r -cf1166a0 prime.c -x -lw72
```

(2) (-LL)

[描述格式]

-LL[行数量]

- 当省略时的解释
  -没有分页符

[功能]

- -LL 选项指定了所有列表文件中每一页的行数。

[应用]

- 如果需要改变列表文件中每页的行数，请指定-LL 选项。

[描述]

- 通过-LL 选项可以指定的行数范围如下。
  20 ≤ 每页能打印的行数 ≤ 65535
- 如果指定了-LL0 选项，则不会有分页符出现。
- 如果未指定行数，那么不会有分页符。
- 如果列表文件规格没有作任何说明，那么-LL 选项无效。

[使用范例]

- 为了设置交叉引用列表文件（prime.xrf）每页的行数为 20 行，描述如下：

```
C> CC78K0R  -cf1166a0  prime.c  -x  -ll20
```

(3) (-LT)

[描述格式]

```
-LT [字符数量]
```

- 当省略时的解释

  -LT8

[功能]

- -LT 选项指出在源模块文件中输出水平制表符(HT 即 tab)的基本跨度，并在列表文件中用一些空白(空格)来代替。

[应用]

- 如果每个文件中用-LW 选项指定更少的字符跨度，那么 HT 编码就会产生更少的空白，所以可以指定-LT 选项来减少字符数量。

[描述]

- -LT 选项可以指定的字符跨度范围如下。

  0 ≤ 指定的字符数量 ≤ 8
- 如果指定了-LT0，那么不再对表格符号进行处理，并且输出 tab 代码。
- 如果字符数量被忽略，那么 tab 扩展字符的跨度会变为 8 个空格。
- 如果列表文件没有作任何说明，那么-LT 选项无效。

[使用范例]

- 如果-LT 选项被忽略，编译器默认指定了-LT8 选项，因为 HT 编码而输出的空白数量为 8。

```
C> CC78K0R  -cf1166a0  prime.c  -p
```

- 因为 HT 编码而输出的空白数量被设置为 1。

```
C> CC78K0R  -cf1166a0  prime.c  -p  -lt1
```

(4) (-LF)

[描述格式]

```
-LF
```

- 当省略时的解释
  -不会添加分页符

[功能]

- 指定-LF 选项将会在每个列表文件的末尾添加新的分页符。

[描述]

- 如果列表文件没有作任何说明，那么-LF 选项无效。

[使用范例]

- 为了在汇编源模块文件的末尾增加分页符，描述如下：

```
C> CC78K0R  -cf1166a0  prime.c  -a  -lf
```

(5) (-LI)

[描述格式]

```
-LI
```

- 当省略时的解释
  -不会添加包含文件中的 C 源文件

[功能]

- -LI 选项将包含文件中的 C 源程序添加到汇编源模块文件中，其中 C 源程序以注释形式出现。

[描述]

- 如果没有指定-SA 选项，则该选项被忽略。

[使用范例]

- 为了将包含文件中的 C 源程序添加到汇编源模块文件（prime.asm）中，描述如下：

```
C> CC78K0R  -cf1166a0  prime.c  -sa  -li
```

警告输出说明

(1) (-W)

[描述格式]

-W [等级]

- 当省略时的解释

  -W1

[功能]

- 指定-W 选项会将警告信息输出到控制台。

[应用]

- 该选项不仅能够决定是否将警告信息输出到控制台，同时输出的还有具体的细节信息。

[描述]

- 下面给出警告信息的等级。

等级	描述
0	不输出警告信息。
1	输出普通等级的警告信息。
2	输出详细的警告信息。

- 如果指定了-E 或-SE 选项，那么警告信息将被输出到错误列表文件。
- 等级 0 说明不需要向控制台和错误列表文件输出警告信息（当-E 或-SE 选项被选定时）。

[使用范例]

- 当-W 选项被忽略时，编译器默认为指定了-W1 选项，并输出普通警告信息。

```
C> CC78K0R   -cf1166a0   prime.c
```

执行状态显示说明

(1) (-V/-NV)

[描述格式]

```
-V
-NV
```

- 当省略时的解释

  -NV

[功能]

- -V 选项将当前编译的执行状态输出到控制台。
- -NV 选项使-V 选项失效。

[应用]

- 指定这个选项可以在执行编译的同时，持续将当前的执行状态输出到控制台。

[描述]

- 输出此过程中的阶段名称和函数名称。
- 如果-V 选项和-NV 选项同时都被选定，那么最后的选项有效。

[使用范例]

- 要将当前的执行状态输出到控制台，描述如下：

```
C> CC78K0R  -cf1166a0  prime.c  -v
```

参数文件说明

(1) (-F)

[描述格式]

```
-F 文件名
```

- 当省略时的解释
  -选项和输入文件名称只能由命令行输入

[功能]

- 指定-F 选项可以从指定的具体文件中读入设定选项或输入文件名。

[应用]

- 当从命令行无法提供足够的信息来启动编译器时，请选定-F 选项，因为编译时可以输入多个选项。
- 当编译过程需要重复指定选项时，在参数文件中描述该选项，并且指定-F 选项。

[描述]

- 参数文件中不允许嵌套。
- 参数文件中用于描述的字符数量没有限制。
- 空格或者制表符可以用来分隔选项或者输入文件名称。
- 在参数文件中描述的选项或者输入文件名会进行扩展，当参数文件的说明被装入命令行时才会扩展。
- 扩展选项的优先级以顺序为准，即最后指定的选项有效。
- 在‘；’和 ‘#’之后直到行尾的字符都被当作注释。

[使用范例]

- 参数文件 prime.pcc 的内容。

```
; parameter file
prime.c   -cf1166a0   -aprime.asm   -e   -x
```

在编译中使用参数文件 prime.pcc。

```
C> CC78K0R   -fprime.pcc
```

临时文件创建目录说明

(1) (-T)

[描述格式]

```
-T 目录
```

- 当省略时的解释
  -文件被创建在环境变量 TMP 指定的目录路径中。如果未指定，则文件会被创建到当前驱动器的当前目录中。

[功能]

- 临时文件创建在-T 选项指定的驱动器和目录中。

[应用]

- 创建临时文件的位置可以用-T 选项指定。

[描述]

- 即使在指定目录中存在有以前创建的临时文件，如果文件没有被保护，则在下次创建会被直接覆盖。
- 在存储器中扩展临时文件需要一定的存储空间。
  如果所需的存储空间无法满足，那么临时文件会被创建到指定目录中，并且将存储器的内容写入这个文件。对于后续临时文件的访问，是对文件的存储而不是直接访问内存。
- 当编译结束时，临时文件会被删除。如果按下 CTRL-C 时，编译暂停，那么临时文件也会被删除。

[使用范例]

- 为了将临时文件的输出到命令指定的 TMP 目录，描述如下：

```
C> CC78K0R  -cf1166a0  prime.c  -ttmp
```

帮助说明

(1) --/-?/-H

[描述格式]

```
--
-?
-H
```

- 当省略时的解释
  -无屏幕显示

[功能]

- 选项--，-?，和-H 能够显示对应选项的简要说明，或者显示诸如控制台默认选项等的帮助信息(只在命令行有效[注])。

  注 不要在 PM +中指定这个选项，若在 PM +中需要参考帮助，请在<Compiler Options>对话框中按下[help]按钮。

[应用]

- 显示对应选项和它的描述，在 C 编译器运行时可以引用这个选项。

[描述]

- 当--，-?，或-H 选项被选中时，所有其它的编译选项均不可用。
- 当需要查阅所显示的帮助信息的延续内容，可以按下[enter]键。要在结束之前需要退出显示，请按下除[enter]键的任意字符，然后按下[enter]键。

[使用范例]

- 为了在控制台上显示帮助信息，描述如下：

```
C> CC78K0R  -H
```

函数扩展说明

(1) (-Z/-NZ)

[描述格式]

-Z[类型] (可以指定多种类型) -NZ

- 当省略时的解释

  -NZ

[功能]

- -Z 选项使能多种函数扩展类型。
- -NZ 选项使-Z 选项无效。
- 类型不能被忽略，否则会发生异常错误(F0012)。

[应用]

- 以下类型说明的函数处理过程同样适用于 78K0R 系列扩展函数。

[描述]

- -Z 选项的类型说明如下。

类型说明	描述
P	“//” 后直到行末的字符都被认为是注释。
C	允许嵌套注释 “/* */”。
S 注	注释中的日本汉字类型说明被解释为 SJIS 编码。
E 注	注释中的日本汉字类型说明被解释为 EUC 编码。
N 注	注释说明中不包含日本汉字类型。
B	字符/无符号字符型参数和返回值不进行整型扩展。
A	不符合 ANSI 标准的函数都认为是非法的。只有和 ANSI 标准兼容的函数才有效。 具体的说，会执行下列任务。 • 下列内容都不再预留为关键字。 callt，norec，sreg，bit，boolean，#asm，#endasm • 三字母序列 (3 个字母表示法)有效。 • 编辑器定义宏 __STDC__ 为 1。 • 通过 far 指针的三个字节执行关系表达式，fData 可以被分配到 64KB 边界区域的最后一个字节。 • 针对字符型位域会输出下列警告。 (CC78K0R 警告信息 W0787：位域类型是字符型) • 如果指定了 -W2 选项，-QC，-ZP，-ZC，-ZI 和-ZL 选项会输出下列警告。 (CC78K0R 警告信息 W0029：‘-QC’ 选项不可移植) (CC78K0R 警告信息 W0031：‘-ZP’ 选项不可移植) (CC78K0R 警告信息 W0032：‘-ZC’ 选项不可移植) • 如果指定了 -W2 选项，针对每个 #pragma 语句输出下列警告。 (CC78K0R 警告信息 W0849：#pragma 语句不可移植) • 如果指定了 -W2 选项，针对__asm 语句和汇编输出会有下列警告。 (CC78K0R 警告信息 W0850：Asm 语句不可移植) • 如果指定了 -W2 选项，针对#asm 到#endasm 块输出下列错误。 (CC78K0R 错误信息 E0801 未定义控制，等等)
F	从 flash 输出目标

注 S, E, 和 N 不能同时指定。

[使用范例]

- “//” 后直到行末的字符都被认为是注释，同样，还允许嵌套注释 “/* */”。

```
C> CC78K0R  -cf1166a0  prime.c  -zpc
```

设备文件搜索路径

(1) (-Y)

[描述格式]

-Y 目录

- 当省略时的解释

  -只有正常搜索路径

  备注 正常搜索路径如下

  (i) <..\..\..\dev>(从该路径启动 CC78K0R.exe)

  (ii) CC78K0R.exe 的启动路径

  (iii) 当前目录

  (iv) 环境变量 PATH 指定的路径

[功能]

- -Y 选项首先寻找指定为设备文件搜寻路径的目录。如果该路径不存在，则会搜索正常路径。

[应用]

- 如果设备文件没有安装在正常路径，而是安装在特殊路径，那么用这个选项来指定路径。

[注意]

- 当使用 PM +时，当把设备文件注册到<Project Setup>对话框中的“Device Name:”时，就选定了一个目录，因此当使用这个编译器设置选项时没有必要再另外指定这个选项。

[使用范例]

- 首先搜索“C:\tmp\dev”来读取设备文件，描述如下：

```
C> CC78K0R  -cf1166a0  -yC:\tmp\dev
```

存储器模式规格

(1) (-M)

[描述格式]

-M 类型

- 当省略时的解释

  -mm

[功能]

- 选项-m 指定编译使用的存储器模式。
- 不能同时指定多个存储器模式。
- 该类型不能忽略，否则，会发生严重错误（F0012）。

[应用]

- 通过指定某种存储器模式，也就指定了每个函数和变量被分配在 near 区域或 far 区域。
- 如果在 C 源文件中出现__near 或__far 修饰符，分配到 near 区域或 far 区域会优先考虑__near 或__far 修饰符。

[描述]

- 用-M 选项可以指定的存储器模式有下列几种。

类型说明	存储器模式	描述
S	小型模式	认为存储器包括 64KB 的代码部分（max），64KB 的数据部分（max）；总共 128KB，可以指定到 near 区域或 far 区域。
M	中型模式	认为存储器包括 1 MB 的代码部分（max），64KB 的数据部分（max）；总共 1 MB，可以指定到 near 区域或 far 区域。
C	紧凑模式	认为存储器包括 64KB 的代码部分（max），1 MB 的数据部分（max）；总共 1 MB，可以指定到 near 区域或 far 区域。
L	大型模式	认为存储器包括 1 MB 的代码部分（max），1 MB 的数据部分（max）；总共 1 MB，可以指定到 near 区域或 far 区域。

备注 即使指定了存储器模式包括最大 64KB 的代码部分或数据部分，指定了__far 修饰符的函数和变量可以被分配到 1 MB（max）的空间去。

存储器模式规格指定了未用__far 修饰符的函数或变量的分配位置。

[使用范例]

- 为了指定编译时的存储器模式为小型模式，描述如下：

```
C> CC78K0R    -cf1166a0   prime.c   -ms
```

第 6 章 C编译器输出文件

本章介绍 CC78K0R 的输出文件。
CC78K0R 会产生下列文件：

- 目标模块文件
- 汇编源模块文件
- 预处理列表文件
- 交叉引用列表文件
- 错误列表文件

6.1 目标模块文件

目标模块文件是一种包含 C 源程序编译结果的二进制映象文件。
如果指定了调试数据输出选项(-G)，目标模块文件将包含调试数据。

6.2 汇编源模块文件

汇编源模块文件是 C 源程序编译结果的 ASCII 映象列表。它也是和目标 C 源程序相对应的汇编语言源程序模块文件。

也可将 C 源程序以注释的形式包含进汇编源模块文件中，这需要设置汇编源模块文件生成选项(-SA)。

[输出格式]

```
    ;78K0R Series C Compiler V(1)x.xx Assembler Source
    ;                                   Date: (2)xxxxx Time: (3)xxxxx
    ; Command       : (4)-cf1166a0 prime.c –sa
    ; In-file       : (5)prime.c
    ; Asm-file      : (6)prime.asm
    ; Para-file     : (7)
    $PROCESSOR ( (8)f1166a0 )
(9) $DEBUG
(10)$NODEBUGA
(11)$KANJICODE    SJIS
(12)$TOL_INF      03FH ,    100H ,    00H ,     00H ,     00H
(13)$DGS      FIL_NAM ,     .file ,     034H ,    0FFFEH , 03FH ,    067H ,    01H ,     00H
      :
(14)      EXTRN   _@RTARG0
      :
      ; line (15)1     : (16)#define   TRUE    1
      ; line (15)2     : (16)#define   FALSE   0
      ; line (15)3     : (16)#define   SIZE    200
      :
(14)_main :
(17)$DGL      1 ,   14
(14)      push    hl                          ; (21) [ INF ] 1 , 1
(14)      subw    sp , #08H                   ; (21) [ INF ] 2 , 1
(14)      movw    ax , sp                     ; (21) [ INF ] 2 , 1
(14)      movw    hl , ax                     ; (21) [ INF ] 1 , 1
      :
(18)??bf_main :
      :
      ; (22)*** Code Information ***
      ;
      ; (23)$FILE C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\CC78K0R\prime.c
      ;
      ; (24)$FUNC main ( 8 )
      ; (25)      bc = ( void )
      ; (26) CODE SIZE = 116 bytes , CLOCK_SIZE = 86 clocks , STACK_SIZE = 16 bytes
      ;
      ; (27)CALL printf ( 18 )
      ; (28)      bc = ( pointer:ax , int : [ sp + 2 ] )
      ;
      ; (27)$CALL putchar ( 20 )
      ; (28)      bc = ( int : ax ) ;
      ;
      ; (27)$CALL printf ( 25 )
```

```
; (28)        bc = ( pointer : ax , int : [ sp + 2 ] )
;
; (24)$FUNC printf ( 31 )
; (25)        bc = ( pointer s : ax , int i : [ sp + 4 ] )
; (26) CODE SIZE = 23 bytes , CLOCK_SIZE = 22 clocks , STACK_SIZE = 14 bytes
;
; (24)$FUNC putchar ( 41 )
; (25)        bc = ( char c : x )
; (26) CODE SIZE = 16 bytes , CLOCK_SIZE = 18 clocks , STACK_SIZE = 6 bytes
; Target chip      : (19)uPD78F1166_A0
; Device file      : (20)Vx.xx
```

[输出项的说明]

编号	说明	列宽	格式
(1)	版本号	4 (固定长度)	显示格式为“x.yz”
(2)	日期	11 (固定长度)	系统日期 (显示格式为“DD Mmm YYYY”)
(3)	时间	8 (固定长度)	系统时间 (显示格式为“HH：MM：SS”)
(4)	命令行	—	在“CC78K0R”之后输出命令行内容。 80 列后的内容从下一行的第 15 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(5)	C 源模块文件名称	操作系统允许的字符数	输出指定文件名。如果文件类型被忽略，那它的默认文件类型（扩展名）就是“.C”。80 列后的内容从下一行的 15 列开始输出。第一列输出为分号(;)。
(6)	汇编源模块文件名称	操作系统允许的字符数	输出指定文件名。 如果文件类型被忽略，那它的文件类型（扩展名）默认为“.asm”。 80 列后的内容从下一行的第 15 列开始输出。第一列输出为分号(;)。
(7)	参数文件内容	—	输出参数文件目录。 80 列后的内容从下一行的第 15 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(8)	设备类型	最大值 6 (可变长度)	这个字符串由-C 选项指定，见有关设备文件相关文档。
(9)	调试数据	最大值 8 (可变长度)	输出调试控制。输出内容为$DEBUG 或$NODEBUG。
(10)	调试信息汇编器控制	9 (固定长度)	输出 NODEBUGA 控制。输出为$NODEBUGA.

编号	说明	列宽	格式
(11)	日文汉字类型信息	最大值 15 (可变长度)	输出日文汉字编码（2 字节编码）类型。输出的是 $KANJICODE SJIS，$KANJICODE EUC，或 $KANJICODE NONE.
(12)	工具信息	37 (固定长度)	输出工具信息，版本号，错误信息，指定参数等。(信息由 $TOL_INF 开始).
(13)	符号信息	—	输出符号信息 (信息开始标志为$DGS)。只有指定了调试数据输出选项时才会输出这个信息。即使指定了–G1 选项也不会输出这项信息。
(14)	汇编源程序	—	输出的汇编源程序中包括编译结果。
(15)	行号	4 (固定长度)	输出 C 源程序模块文件的行号，计算结果用零抑制的十进制表示，且用向右对准的格式。
(16)	C 源程序	—	输入 C 源程序映象。80 列后的内容从下一行的 15 列开始输出。第一列输出为分号(;)。
(17)	行号信息	—	行号就是行号入口（信息以$DGL 开始）。只有指定了调试数据输出选项时才会输出这个信息。即使指定了–G1 选项也不会输出这项信息。
(18)	符号信息创建对应的标签	最大值 34 (可变长度)	输出函数标签信息（信息以？？开始） 只有指定了调试数据输出选项时才会输出这个信息。
(19)	本次编译中使用的目标设备	最大值 15 (可变长度)	目标设备在命令行中通过选项-C 指定，或在源程序文件中指定。
(20)	设备文件版本	6 (固定长度)	显示输入设备文件的版本号
(21)	大小，周期	—	输出指令所占的空间大小和执行时间（信息以;[INF]开始)。 不是存取数据而是存取内部 RAM 区域或 SFR 区域，则会输出时钟周期的数量。 对于条件跳转指令，会输出坚实条件所需的时钟周期数量。 未考虑意外情况。
(22)	函数信息 (起始)	—	显示函数信息的起始
(23)	函数信息（文件名）	—	输出带有绝对路径的目标源程序文件名（信息以;$FILE 开始).
(24)	函数信息（定义函数）	—	输出函数名称，用十进制码表示行号（信息以;$FUNC 开始).
(25)	函数信息（返回值，函数定义时的参数）	—	输出函数的返回值寄存器和参数信息（寄存器和堆栈）

编号	说明	列宽	格式
(26)	函数信息（定义函数的大小，时钟，堆栈）	—	输出容量大小，执行时间和统计出的函数堆栈最大值。 此处显示为本函数自己使用的堆栈大小。 如果该函数调用了另外的函数，被调用函数所使用的堆栈大小不会加到发起调用的函数堆栈大小。 CLOCK_SIZE 是第（21）项时钟周期数量总和的结果。
(27)	函数信息（调用函数）	—	输出函数名和函数调用行号，用十进制码表示（信息以;$CALL 开始）。
(28)	函数信息（调用函数的返回值，参数）	—	输出在函数调用期间的返回值寄存器和参数信息（寄存器和堆栈位置）

6.3 错误列表文件

错误列表文件包括编译中发生的所有错误信息和警告信息。

通过指定编程选项，可以将 C 源程序加入错误列表文件。通过修改 C 源程序和删除注释如列表头，含有 C 源程序的错误列表文件可以用 C 源程序模块文件。

6.3.1 关于C语言的错误列表文件

[输出格式]

```
/*
78K0R Series C Compiler V(1)x.xx Error List          Date : (2)xxxxx Time : (3)xxxxx
Command       : (4)-cf1166a0 prime.c -se
C-file        : (5)prime.c
Err-file      : (6)prime.cer
Para-file     : (7)
*/
(8)#define      TRUE     1
(8)#define      FALSE    0
(8)#define      SIZE     200
(8)char   mark [ SIZE + 1 ] ;
(8)main ( )
(8){
(8)       int            i , prime , k , count ;
(8)       cont = 0 ;
*** CC78K0R error (9)E0711 : (10)Undeclared 'cont' ; function 'main'
(8)       for ( i = 0 ; i <= SIZE ; i++ )
(8)            mark [ i ] = TRUE ;
(8)       for ( i = 0 ; i <= SIZE ; i++ ) {
(8)            if ( mark [ i ] ) {
                    prime = i + i + 3 ;
                    printf ( "%6d" , prime ) ;
*** CC78K0R warning (9)W0745 : (10)Expected function prototype
:
/*
(11)Target chip : uPD78F1166_A0
(12)Device file : Vx.xx
Compilation complete, (13)1 error(s) and (14)5 warning(s) found.
*/
```

[输出项目的说明]

编号	说明	列宽	格式
(1)	版本号	4 (固定长度)	显示格式为“x.yz”
(2)	日期	11 (固定长度)	系统日期 (显示格式为“DD Mmm YYYY”)
(3)	时间	8 (固定长度)	系统时间 (显示格式为“HH：MM：SS”)
(4)	命令行	—	在“CC78K0R”之后输出命令行内容。 80 列后的内容从下一行的第 13 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(5)	C 源模块文件名称	操作系统允许的字符数(可变长度)	输出指定文件名。 如果文件类型被忽略，那它的默认文件类型(扩展名)为“.C”。80 列后的内容从下一行的第 13 列开始输出。第一列输出为分号(;)。
(6)	错误列表文件	操作系统允许的字符数(可变长度)	输出指定文件名。 如果文件类型被忽略了， 那它的默认文件类型（扩展名）就是‘.cer’。80 列后的内容在第 13 列开始的下一行输出。
(7)	参数文件目录	—	输出参数文件目录。 80 列后的内容从下一行的 15 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(8)	C 源程序	—	这是输入 C 源程序映象，80 列之后的内容不会被换到下一行。
(9)	错误信息编号	5 (固定长度)	用 “#nnn” 这种格式输出错误编号。 如果有错误，“#”的位置输出“F”。，如果有警告，“#”的位置输出“W”。 "nnn" (错误编号)显示为 3 位十进制数(无需零抑制)。
(10)	错误信息	—	见第 9 章 错误信息。80 栏后的内容不会换到下一行
(11)	编译器使用的目标设备	最大值 15 (可变长度)	目标设备在命令行中通过选项-C 指定，或在源程序文件中指定。
(12)	设备文件版本	6 (固定长度)	显示输入设备文件的版本号。
(13)	错误次数	4 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。
(14)	警告次数	4 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。

6.3.2 只有错误信息的错误列表文件

[输出格式]

```
(1)prime.c ( (2)18 ) : CC78K0R warning (3)W0745 : (4)Expected function prototype
(1)prime.c ( (2)20 ) : CC78K0R warning (3)W0745 : (4)Expected function prototype
(1)prime.c ( (2)26 ) : CC78K0R warning (3)W0622 : (4)No return value
(1)prime.c ( (2)37 ) : CC78K0R warning (3)W0622 : (4)No return value
(1)prime.c ( (2)44 ) : CC78K0R warning (3)W0622 : (4)No return value

Target chip    : (7)uPD78F1166_A0
Device file    : (8)Vx.xx
Compilation complete, (5)0 error(s) and (6)5 warning(s) found.
```

[输出项说明]

编号	说明	列宽	格式
(1)	C 源程序模块文件名称	操作系统允许的字符数	输出指定文件名称。如果忽略了文件类型，那它的默认文件类型（扩展名）为“.C”.
(2)	行号	5 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。
(3)	错误信息编号	4 (固定长度)	用 “#nnn” 这种格式输出错误编号。 如果有错误，“#”的位置输出“F”。，如果有警告，“#”的位置输出“W”。 "nnn" (错误编号)显示为 3 位十进制数(无需零抑制)。
(4)	错误信息	—	见**第 9 章 错误信息**
(5)	错误次数	4 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。
(6)	警告次数	4 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。
(7)	编译器中的目标设备	最大值 15 (可变长度)	目标设备在命令行中通过选项-C 指定，或在源程序文件中指定。
(8)	设备文件版本	6 (固定长度)	显示输入设备文件的版本号

6.4 预处理列表文件

预处理列表文件是一种 ASCII 映象文件，仅包括 C 源程序预处理结果。

当指定了-K 选项且“N”没有被指定为处理类型时，预处理列表文件的作用和 C 源程序模块文件相同。当指定-KD 选项时，输出带有#define 扩展的列表。

[输出格式]

<当页面宽度 = 80 时>

```
/*
78K0R Series C Compiler V(1)x.xx Preprocess List
                                Date : (2)xxxxx Page : (3)xxx
Command       : (4)-cf1166a0 prime.c -p -lw80
In-file       : (5)prime.c
PPL-file      : (6)prime.ppl
Para-file     : (7)
*/
(8)1 : (9)#define      TRUE     1
(8)2 : (9)#define      FALSE    0
(8)3 : (9)#define      SIZE     200
(8)4 : (9)
(8)5 : (9)char    mark [ SIZE + 1 ] ;
(8)6 : (9)
/*
(10)Target chip       : uPD78F1166_A0
(11)Device file       : Vx.xx
*/
```

[输出项说明]

编号	说明	列宽	格式
(1)	版本号	4 (固定长度)	显示格式为“x.yz”
(2)	日期	11 (固定长度)	系统日期 (显示格式为“DD Mmm YYYY”)
(3)	页数	4 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。
(4)	命令行	—	在“CC78K0R”之后输出命令行内容。 80 列后的内容从下一行的第 13 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(5)	C 源模块文件名称	操作系统允许的字符数	输出指定文件名。 如果文件类型被忽略，那它的默认文件类型（扩展名）为“.C”。 80 列后的内容从下一行的第 13 列开始输出。

编号	说明	列宽	格式
(6)	预处理列表文件名	操作系统允许的字符数	输出指定文件名。 如果文件类型被忽略，那它的默认文件类型（扩展名）为“.ppl”。 80 列后的内容从下一行的第 13 列开始输出。
(7)	参数文件内容	—	输出参数文件内容。 80 列后的内容从下一行的第 13 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(8)	行号	5 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。
(9)	C 源程序	—	这是输入的 C 源程序。 超过 80 行长度的内容在下一行第 9 列开始输出。
(10)	编译器的目标设备	最大值 15 (可变长度)	通过在命令行中选项指定目标设备，或在源程序文件中指定。
(11)	设备文件版本	6 (固定长度)	显示输入设备文件的版本号

6.5 交叉引用列表文件

交叉引用列表文件包括标识符的列表比如声明、定义、引用函数和变量。同时也包括其他的信息，如属性和行号。输出这些信息是为了方便查看。

[输出格式]

<当页面宽度为 80 时>

```
78K0R Series C Compiler V(1)x.xx Cross reference List
                                    Date: (2)xxxxx Page: (3)xxx
Command       : (4) -cf1166a0 prime.c -x -lw80
In-file       : (5)prime.c
Xref-file     : (6)prime.xrf
Para-file     : (7)
Inc-file      : [ n ] (8)

(9)ATTRIB     (10)MODIFY    (11)TYPE   (12)SYMBOL  (13)DEFINE    (14)REFERENCE
EXTERN        NEAR          array      mark        5             14   16   22
EXTERN        FAR           func       main        7
AUTO1                       int        i           9             13   13   13   14
                                                                 15   15   15   16
                                                                 17   17   21
AUTO1                       int        prime       9             17   18   21   21
AUTO1                       int        k           9             21   21   21   22
AUTO1                       int        count       9             11   19   20   25
:
/*(15)Target chip : uPD78F1166_A0
(16)Device file : Vx.xx */
```

[输出项目的说明]

编号	说明	列宽	格式
(1)	版本号	4	显示格式为“x.yz”
(2)	日期	11 (固定长度)	系统日期 (显示格式为“DD Mmm YYYY”)
(3)	页数	4 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。
(4)	命令行	—	在“CC78K0R”之后输出命令行内容。 80 列后的内容从下一行的第 13 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(5)	C 源模块文件名称	操作系统允许的字符数	输出指定文件名。 如果文件类型被忽略，那它的默认文件类型（扩展名）为“.C”。80 列后的内容从下一行的第 13 列开始输出。
(6)	交叉引用数据清单文件名	操作系统允许的字符数	输出指定文件名。 如果文件类型被忽略，那它的默认文件类型（扩展名）为“.xrf”。80 列后的内容从下一行的第 13 列开始输出。

编号	说明	列宽	格式
(7)	参数文件内容	—	输出参数文件内容。 80 列后的内容从下一行的第 13 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(8)	包含文件	操作系统允许的字符数	输出在 C 源程序中指定的文件名称。从 1 开始编号的数字“n”表示所包含的文件号。超过 80 行长度的内容在下一行 13 列开始输出。没有包含文件则不输出此行内容。
(9)	符号属性	6 (固定长度)	显示符号属性。 外部变量用 EXTERN 表示，外部静态变量用 EXSTC 表示，内部变量用 INSTC 表示，自动变量用 AUTOnn 表示，寄存器变量用 REGnn 表示（这里的 nn 表示范围，编号从 1 开始累加)，外部 typedef 声明用 EXTYP 表示，内部 typedef 声明用 INTYP 表示，标签用 LABEL 表示，结构体或共用体标签用 TAG 表示，成员用 MEMBER 表示，函数参量用 PARAM 表示。
(10)	符号限定词属性	6（固定长度）	显示符号限定词属性（左对齐）。 常变量用 CONST 表示， volatile 变量用 VLT 表示，callt 函数用 CALLT 表示，norec 函数用 NOREC 表示，sreg-bit 变量用 SREG 表示，特殊功能寄存器变量用 RWSFR 表示，只读 sfr 变量用 ROSFR 表示，只写 sfr 变量用 WOSFR 表示，中断函数用 VECT 表示，分配在 near 区域的函数和变量用 NEAR，分配在 far 区域的函数和变量用 FAR。
(11)	符号类型	7（固定长度）	显示符号类型。类型包括字符型、整型、短整型、长整型、字段。以“u”开始的是无符号型。其他的类型还有空类型、浮点型、双精度、长双精度(long double)、函数、数组、指针、结构、联合、枚举、比特、中断和#define 宏定义.
(12)	符号名称	15 (固定长度)	如果符号名超过了 15 个字符并排为一行，那就按照符号名称原样输出。如果符号名超过 15 个字符且超出了一行的长度，超过的部分从下一行的第 23 列输出，13 项和 14 项从下一行 39 列输出。
(13)	符号定义行号	7 (固定长度)	输出定义符号的文件名和行号，显示格式如下： 行号(5 位数)：包含文件数。
(14)	符号引用行号	7 (固定长度)	输出引用符号的文件名和行号，显示格式如下： 行号(5 位数)：包含文件数。如果所列内容超过行的长度，剩余内容从下一行 47 列开始输出。
(15)	本次编译中的目标设备	最大值 15 (可变长度)	目标设备在命令行中通过选项-C 指定，或在源程序文件中指定。
(16)	设备文件版本	6 (固定长度)	显示输入设备文件的版本号

第 7 章 C编译器的使用方法

本章介绍 CC78K0R 的高效使用方法。

7.1 高效操作（EXIT状态函数）

当编译结束时，CC78K0R 向操作系统返回编译过程中最严重的错误等级，作为 EXIT 状态。

EXIT 状态如下：

表 7-1 EXIT 状态

正常结束	0
警告	0
严重错误	1
中止	2

如果没有使用 PM +，当 CC78K0R 运行于命令行形式时，可使用批处理文件中的状态进一步提高操作效率。

[使用实例]

```
CC78K0R  –cf1166a0  %1
IF ERRORLEVEL 1 GOTO ERR
CC78K0R  –cf1166a0  %2
IF ERRORLEVEL 1 GOTO ERR
GOTO EXIT
ERR
echo Some error found.
EXIT
```

[说明]

如果编译到 1%后的 C 源程序时产生了一个严重错误，从本质上说，在输出错误信息后还会继续处理。但是因为用到了 EXIT 状态返回的值 1，可以停止执行，而无需继续处理后面 2%部分的 C 源程序。

7.2 建立开发环境（环境变量）

CC78K0R 支持下列环境变量。

表 7-2 环境变量

环境变量	描述
路径	搜索可执行工程文件的路径
INC78K0R	搜索包含文件的路径
TMP	搜索临时文件路径
LANG78K	汉字编码的类型（可通过-ZE, -ZS, 或-ZN 选项来指定） （euc: EUC 编码, sjis:shift JIS 码, none：没有 2 字节码）
LIB78K0R	搜索库的路径

[使用实例]

<当使用 DOS 提示符时>

```
; AUTOEXEC.BAT
PATH C:\Program Files\NEC Electronics
Tools\CC78K0R\Vx.xx\bin;C:\bat;C:\cc78k0r;C:\tool
VERIFY ON
BREAK ON
SET INC78K0R=C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\inc78k0r
SET LIB78K0R=C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r
SET TMP=C:\tmp
SET LANG78K=sjis
```

[说明]

- 按 C:\Program Files\NEC Electronics Tools\CC78K0R\V*x.xx*\bin, C:\bat, C:\cc78k0r, C:\tool 的路径顺序来搜索可执行文件。
- 从 C:\Program Files\NEC Electronics Tools\CC78K0R\V*x.xx*\inc78k0r 目录下搜索包含文件
  如果没有设置，将搜索 C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\inc78k0r 目录（假设 CC78K0R 安装目录为 C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx）
- 在连接时从 C:\Program Files\NEC Electronics Tools\CC78K0R\V*x.xx*\lib78k0r 搜索库文件。
  如果没有设置，将搜索 C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx\lib78k0r 目录（假设 CC78K0R 安装目录为 C:\Program Files\NEC Electronics Tools\CC78K0R\Vx.xx）
- 临时文件存放在 c:\tmp 目录下。
- shift JIS 码的使用和日文汉字码相同。

[警告]

当使用 PM +时不要设置环境变量。

7.3 中断编译

如果是从命令行状态下进行编译，输入命令键(CTRL-C)会打断编译。如果指定‘break on’，给操作系统的控制返回值和命令键的输入时间无关。如果指定‘break off,’只有当屏幕有显示时才会向操作系统输出控制返回值。然后所有打开的临时文件和输出文件将被删除。

如果你需要在 PM +中停止建立（MAKE），选择 PM +窗口的[Run]菜单下的“Stop build”，或单击工具栏上的[Stop Build]按钮。当在 PM +中建立时，命令键的输入无效。

第 8 章 启动例程

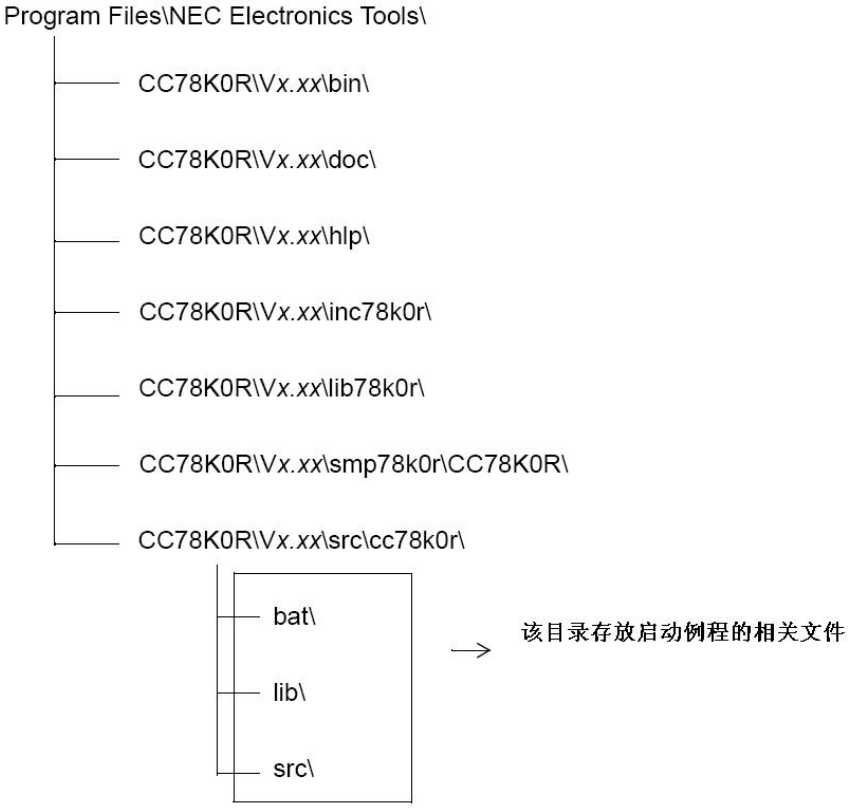
为了执行 C 源程序，需要有一个程序来激活系统和用户程序（主函数）中的 ROM 化过程（ROMization）。这个程序叫做启动例程。

为了执行用户编写的程序，必须为程序创建一个启动例程。CC78K0R 提供了启动例程的目标文件，其中包括程序执行前必需的处理和启动例程的源文件（汇编源程序），用户可以修改启动例程的源文件来满足具体的系统需求。将启动例程的目标文件连接到用户程序，就可以创建一个可执行的程序。即使用户没有描述预处理执行过程，也同样可以成功创建。

本章叙述了启动例程的内容、使用方法和修改办法。

8.1 文件结构

有关启动例程的文件都存放在编译器程序包的 SRC\CC78K0R 目录中。



8.1.1 BAT目录内容

本目录中的批处理文件不能在 PM +中使用。

只有必须修改源程序（例如启动例程）时才使用这些批处理文件。

表 8-1 BAT 目录内容

批处理文件名	说明
mkstup.bat	启动例程的汇编批处理文件
reprom.bat	用来更新 rom.asm 注1 的批处理文件
repgetc.bat	用来更新 getchar.asm 的批处理文件
repputc.bat	用来更新 putchar.asm 的批处理文件
repputcs.bat	用来更新_putchar.asm 的批处理文件
repselo.bat	用来更新 setjmp.asm 和 longjmp.asm 的批处理文件 (保存了编译器程预留区域)注2
repselon.bat	用来更新 setjmp.asm 和 longjmp.asm 的批处理文件 (未保存编译器程预留区域) 注2
repvect.bat	用来更新 vect*.asm 的批处理文件

注 1 ROMization 例程在库中，所以库也会被批处理文件更新。

注 2 保存了编译器预留区域的 setjmp 和 longjmp（为 KREG××保留 saddr 区域，等等），以及未保存编译器预留区域的 setjmp 和 longjmp（只保存寄存器）都会被创建。

8.1.2 SRC目录内容

SRC 目录包括启动例程的汇编源程序、ROM 例程和标准库函数（部分）。如果应用系统要求源程序必须修改，可以对这个源程序修改，并使用 BAT 目录中的一个批处理文件进行汇编，由此创建连接所需的目标文件。

表 8-2 SRC 目录内容

启动例程源程序文件名	说明
cstart.asm 注	启动例程的源程序文件 (使用标准库时)
cstartn.asm 注	启动例程的源程序文件 (未使用标准库时)
rom.asm	ROMization 例程的源文件
_putchar.asm	_putchar 函数
putchar.asm	putchar 函数
getchar.asm	getchar 函数
longjmp.asm	longjmp 函数
setjmp.asm	setjmp 函数
def.inc	根据类型来设置库
macro.inc	每个典型模式的宏定义
vect.inc	Flash 存储区域分支表的起始地址
library.inc	明确选择分配到 boot 区域的库

注 文件名带有 n 的启动例程不包含标准库处理。只有在未使用标准库时才使用文件名带 n 的例程。cstartb*.asm 是 boot 区域的启动例程，cstarte*.asm 是 flash 区域的启动例程。

8.1.3 “Lib”目录内容

Lib 目录包括从启动例程和库的源文件汇编而来的目标文件。该目标文件可以和任何 78K0R 系列目标设备的程序相连接。如果不特别需要修改代码，就可以直接连接系统提供的未经修改的目标文件。当执行 CC78K0R 附属的批处理文件 mkstup.bat 时，该目标文件被覆盖重写。

表 8-3 “Lib”目录内容

文件名称			文件作用
正常	Boot 区域	Flash 区域	
cl0rm.lib cl0rl.lib cl0rmf.lib cl0rlf.lib cl0rxm.lib cl0rxl.lib	cl0rm.lib cl0rl.lib cl0rmf.lib cl0rlf.lib cl0rxm.lib cl0rxl.lib	cl0rme.lib cl0rle.lib cl0rmfe.lib cl0rlfe.lib cl0rxme.lib cl0rxle.lib	库（运行时刻库和标准库）
s0rm.rel s0rml.rel s0rl.rel s0rll.rel	s0rmb.rel s0rmlb.rel s0rlb.rel s0rllb.rel	s0rme.rel s0rmle.rel s0rle.rel s0rlle.rel	启动例程使用的目标文件

详细的文件内容，请参阅“**2.5.1 库文件**”。

8.2 批处理文件说明

8.2.1 生成启动例程的批处理文件

BAT 目录中的 mkstup.bat 用来创建启动例程的目标文件。

RA78K0R 汇编程序包中的汇编器对 mkstup.bat 来说是必需的。因此，如果没有指定路径，就需要指定路径并来运行。

以下详细解释这个文件的使用方法。

[如何使用]

- 在包含 mkstup.bat (mkstup.sh)的 src\CC78K0R\bat 目录中执行如下命令行。

```
mkstup    设备类型注
```

注 请参阅设备文件用户手册或“设备文件操作注意事项”。

[使用实例]

- 将要创建的启动例程使用的目标设备是μPD78f1166_A0。

```
mkstup  f1166a0
```

mkstup.bat 批处理文件存储在 LIB 目录下，它将会覆盖重写启动例程的目标文件。LIB 目录和下面显示的 BAT 目录在同一级目录中。

需要连接到目标文件的启动例程会输出到每一个目录。

在 LIB 中创建的目标文件的名称如下。

```
——— LIB ———————  s0rm.rel
                 s0rmb.rel
                 s0rme.rel
                 s0rml.rel
                 s0rmlb.rel
                 s0rmle.rel
                 s0rl.rel
                 s0rlb.rel
                 s0rle.rel
                 s0rll.rel
                 s0rllb.rel
                 s0rlle.rel
```

8.3 启动例程

8.3.1 启动例程概述

启动例程的作用是为了执行用户编写的 C 源程序而做的准备工作。通过连接用户程序，可以创建装载模块文件，此文件可以完成目标。

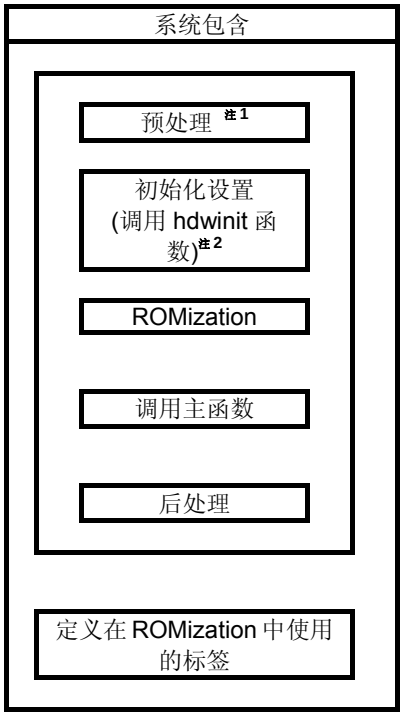
（1）功能

存储器初始化，包含在系统中的 ROMization 和 C 源程序的开始进程和终止进程都会进行。

ROMization： 在 C 源程序中定义的外部变量、静态变量和 sreg 变量的初始值都被存放在 ROM 中。然而，ROM 中的变量值无法重写，只能在 ROM 中保持原值不变。因此，定位到 ROM 中的初值必须复制到 RAM 中去运行。这个过程叫做 ROMization。当程序被写入 ROM 后，可以由微处理器来调用执行。

（2）配置

下图显示了与启动例程相关的程序以及它们的配置情况。



注 1 如果使用标准库，就首先进行库的相关处理。启动例程源文件中名字后面没有加“n”的文件在处理时与标准库有关。文件名末尾加“n”的文件不需要处理标准库。

注 2 hdwinit 函数是当用户需要对外围设备（sfr）初始化时创建的函数。通过创建 hdwinit 函数，初始设置的时间可以加快（初始设置也可以在主函数中完成）。如果用户未创建 hdwinit 函数，将不做任何处理就直接返回。

cstart.asm 和 cstartn.asm 的内容几乎完全相同。

下表展示了 cstart.asm 和 cstartn.asm 的不同之处。

启动例程的类型	使用库处理
cstart.asm	是
cstartn.asm	否

（3）启动例程的使用

下表列举了 CC78K0R 为启动例程提供的目标文件名称。

文件类型	源文件	目标文件
启动例程	cstart*.asm 注 1,2	s0*.rel 注 2,3,4
ROM 文件	rom.asm	包含在库中

注 1 *：如果没有使用标准库，就在*的位置上换“n”。如果使用了，就不需加 n。

注 2 ‘b’表示 boot 区域使用的启动例程，‘e’表示 flash 区域使用的启动例程，

注 3 *：如果使用了标准程序库中的一个固定区域，需要加上“l”。

注 4 *：如果指定了小型模式或紧凑模式，加上“m”。如果指定了紧凑模式或大型模式，需要加上“l”。如果指定了小型模式或紧凑模式，当变量被分配到 far 区域，使用加上“l”的启动例程。

备注 rom.asm 中定义了标签，用来指示 ROMization 过程中数据复制的结束地址。rom.asm 的目标包含在库中。

8.3.2 样例程序的说明（cstart.asm）

本节使用 cstart.asm 和 rom.asm 作为范例来说明启动例程的内容。启动例程包括预处理、初始化设置、ROMization 处理、启动主函数和后处理等部分。

注 调用 cstart 时需要在前面加_@，即格式为_@cstart。

（1）预处理

cstart.asm 中的预处理在<1>到<6>项中说明（如下）。

<cstart.asm 预处理>

```
        NAME    @cstart
$INCLUDE ( def.inc )                                                    ; (1)
$INCLUDE ( macro.inc )                                                  ; (2)

BRKSW           EQU     1       ; brk , sbrk , calloc , free , malloc , realloc function use
EXITSW          EQU     1       ; exit , atexit function use
RANDSW          EQU     1       ; rand , srand function use
DIVSW           EQU     1       ; div function use
LDIVSW          EQU     1       ; 1div function use
FLOATSW         EQU     1       ; floating point variables use
STRTOKSW        EQU     1       ; strtok function use
                PUBLIC  _@cstart , _@cend                               ; (3)
$_IF ( BRKSW )
                PUBLIC  _@BRKADR , _@MEMTOP , _@MEMBTM
:
$ENDIF
                EXTRN   _main , _@STBEG , _hdwinit , _@MAA              ; (4)
$_IF ( EXITSW )
                EXTRN   _exit
$ENDIF
                                                                        ; (5)
                EXTRN   _?R_INIT , _?RLINIT , _?R_INIS , _?DATA , _?DATAL , _?DATS
@@DATA  DSEG    BASEP ; near                                            ; (6)
$_IF ( EXITSW )
_@FNCTBL :      DS   4 * 32
_@FNCENT :      DS   2
:
_@MEMTOP :      DS   32
_@MEMBTM :
$ENDIF
```

<1>包含头文件

def.inc → 根据类型设置库。

macro.inc → 每个典型模式的宏定义。

<2>库切换

如果没有使用注释中的标准库，如果把 EQU 的定义改为 0，会保留未使用的库处理所需空间，库使用所需的空间也同样会保留。默认设置是全都使用（如果启动例程中无需库处理，则不进行这个过程）。

<3>符号定义

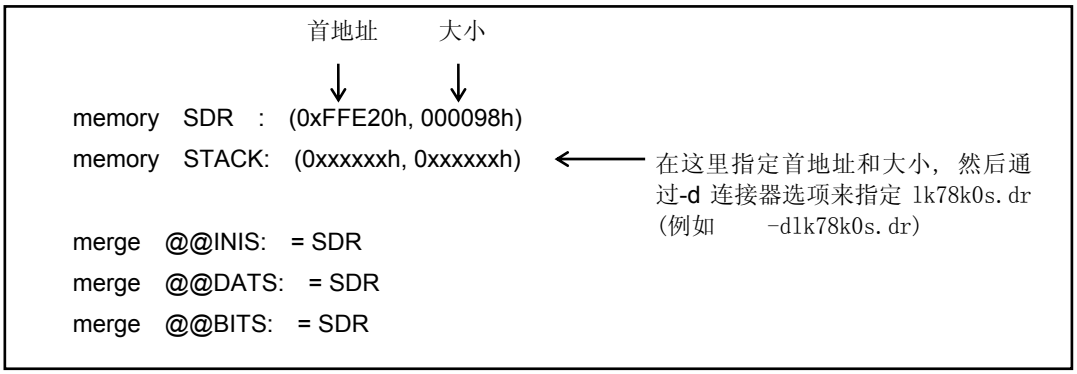
定义使用标准库所需的符号。

<4>堆栈分析所需的符号外部引用声明

- 用于堆栈分析的这个公共符号（_@STBEG）是个外部引用声明。
  _@STBEG 的值是堆栈区域的最终地址+1。
- 在连接器中指定符号生成选项（-S）即可自动生成_@STBEG 以方便堆栈分析。因此，当连接时都会指定-S 选项。这种情况下，指定堆栈中使用的区域名称。如果区域的名称被省略，就使用 RAM 区域。但是创建一个连接指令文件（link directive file），可以将堆栈区域定位到任何地方。关于存储器映射，敬请参阅目标设备的用户手册。
  下面是一个连接指令文件的例子。连接指令文件是一个文本文件，可以由用户在普通的编辑器中创建（关于描述方法的细节，请参阅 RA78K0R 汇编器程序包操作篇 用户手册）。

[在连接中指定–sSTACK 的例子]

创建 lk78K0R.dr（连接指令文件）。由于 ROM 和 RAM 的分配都是通过引用目标设备中的存储器映射进行默认操作，所以不需要指定 ROM 和 RAM 的分配，除非必须要改变。连接指令参阅 smp78K0R\CC78K0R 目录中的 lk78K0R.dr 文件。



<5> ROMization 处理中的标签外部引用声明

ROMization 处理所需的标签在后处理部分中定义。

<6>为标准库保留区域

对标准库使用所需的区域进行保留。

（2）初始化设置

cstart.asm 中的初始化设置描述如下。

<cstart.asm 中的初始化设置>

```
@@VECT00    CSEG    AT      0                                   ; (1)
            DW      _@cstart
@@LCODE     CSEG    BASE
_@cstart :
            SEL     RB0                                         ; (3)
            MOV     A , #_@MAA                                  ; (2)
            MOV1    CY , A.0
            MOV1    MAA , CY
            MOVW    SP , #LOWW _@STBEG ; SP <-stack begin address   ; (4)
            CALL    !!_hdwinit                                  ; (5)
:
$_IF ( BRKSW  OR  EXITSW  OR  RANDSW  OR  FLOATSW )
            CLRW    AX
$ENDIF
:
```

<1>复位向量设置

复位向量表区段的定义如下。设置启动例程的起始地址。

```
@@VECT00    CSEG    AT      0000H
            DW      _@cstart
```

<2>Mirror 区域设置

设置 mirror 区域。

关于 mirror 区域，请参阅目标设备的用户手册。

<3>设置寄存器 bank 组

将寄存器 bank 第 0 组 RB0 设置为当前工作寄存器。

<4>堆栈指针(SP)设置

将_@STBEG 存入堆栈指针。

通过在连接器中为堆栈归总指定符号生成选项（-S）自动生成_@STBEG。

<5>硬件初始化函数调用

当用户需要一个函数来初始化外部设备（SRF）时，创建 Hdwinit 函数。通过创建这个函数，初始化设置就可以和用户目标相匹配。

如果用户没有创建 hdwinit 函数，不做任何处理就直接返回。

（3）ROMization 处理

下面描述 cstart.asm 中的 ROMization。

<ROMization 处理>

```
; copy external variables having initial value
$_IF ( _ESCOPY )
        MOV     ES , #HIGHW _@R_INIT
$ENDIF
        MOVW    HL , #LOWW _@R_INIT
        MOVW    DE , #LOWW _@INIT
        BR      $LINIT2
LINIT1 :
$_IF ( _ESCOPY )
        MOV     A , ES : [ HL ]
$ELSE
        MOV     A , [ HL ]
$ENDIF
        MOV     [ DE ] , A
        INCW    HL
        INCW    DE
LINIT2 :
        MOVW    AX , HL
        CMPW    AX , #LOWW _?R_INIT
        BNZ     $LINIT1
```

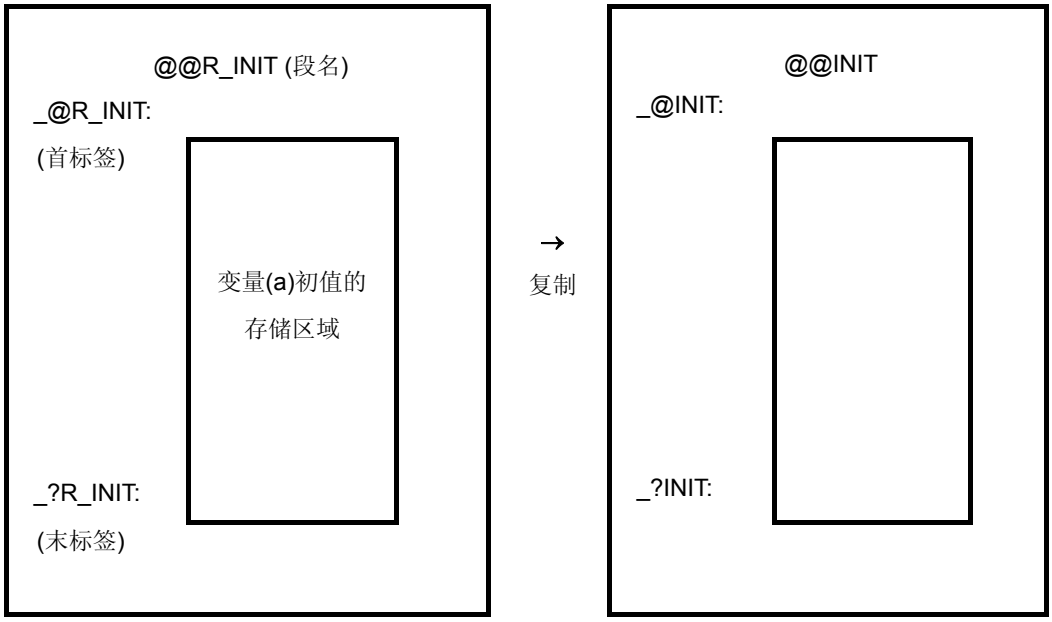
在 ROMization 处理过程中，储存在 ROM 中的外部变量初始值和 sreg 变量初始值都被复制到 RAM 中。涉及的变量有如下（a）到（d）四种类型：

<示例>

```
char    c = 1;              (a) 有初值的外部变量
int     i;                  (b) 没有初值的外部变量注
_ _sreg int     si = 0;     (c) 有初值的 sreg 变量
_ _sreg char    sc;         (d) 没有初值的 sreg 变量注
main ()
{
        ⋮
}
```

注 没有初值的外部变量和没有初值的 sreg 变量无需复制，对应的 RAM 直接清零。

- 下图显示了(a)有初值的外部变量 的 ROMization 处理过程。
  变量（a）的初值被 CC78K0R 编译器放在 ROM 中的@@R_INIT 段。ROMization 处理将这些值复制到 RAM 中的@@INIT 段（变量（c）的处理过程与此相同）。



- @@R_INIT 中的首标签和末标签分别由_@R_INIT 和 _?R_INIT 定义。@@INIT 段的首标签和末标签分别由_@INIT 和 _?INIT 定义。

- 变量（b）和（d）无需复制，RAM 中对应的段直接清零。下表显示了变量（a）和（d）放置的 ROM 段名称和 RAM 段名称，以及每个段中初值的首末标签。

<初值在 ROM 中的区域>

变量类型	区段	首标签	首标签
有初值的外部变量 (a) （被分配在 near 区域）	@@R_INIT	_@R_INIT	_?R_INIT
有初值的外部变量 (a) （被分配在 far 区域）	@@RLINIT	_@RLINIT	_?RLINIT
有初值的 sreg 变量 (c)	@@R_INIS	_@R_INIS	_?R_INIS

<初值在 RAM 中的区域（复制的目标地址）>

变量类型	区段	首标签	首标签
有初值的外部变量 (a) （被分配在 near 区域）	@@INIT	_@INIT	_?INIT
有初值的外部变量 (a) （被分配在 far 区域）	@@INITL	_@INITL	_?INITL
无初值的外部变量 (b) （被分配在 near 区域）	@@DATA	_@DATA	_?DATA
无初值的外部变量 (b) （被分配在 far 区域）	@@DATAL	_@DATAL	_?DATAL
有初值的 sreg 变量 (c)	@@INIS	_@INIS	_?INIS
无初值的 sreg 变量 (d)	@@DATS	_@DATS	_?DATS

（4）启动主函数和后处理

cstart.asm 中启动主函数和后处理的描述如下。

<启动主函数和后处理>

```
        CALL    !!_main    ; main ( ) ;                    ; (1)
$_IF ( EXITSW )
        CLRW    AX
        CALL    !!_exit    ; exit ( 0 ) ;                  ; (2)
$ENDIF
        BR      $$
;
_@cend :
                                                           ; (3)
@@R_INIT        CSEG    UNIT64KP
_@R_INIT :
@@RLINIT        CSEG    UNIT64KP
_@RLINIT :
@@R_INIS        CSEG    UNIT64KP
_@R_INIS :
@@INIT          DSEG    BASEP
_@INIT :
@@INITL         DSEG    UNIT64KP
_@INITL :
@@DATA          DSEG    BASEP
_@DATA :
@@DATAL         DSEG    UNIT64KP
_@DATAL :
@@INIS          DSEG    SADDRP
_@INIS :
@@DATS          DSEG    SADDRP
_@DATS :
@@CALT          CSEG    CALLT0
@@CNST          CSEG    MIRRORP
@@CNSTL         CSEG    PAGE64KP
@@BITS BSEG
;
                END
```

<1>启动主函数

主函数被调用。

<2>启动 EXIT 函数

如果需要，调用 EXIT 函数。

<3>对 ROMization 处理中使用的段和标签进行定义

定义了（a）到（d）每个变量在 ROMization 处理中使用的段和标签（见**(3) ROMization 处理**）。区段指示了每个变量初值的存储区域，标签指示每个段的首地址。

下面对 ROMization 处理文件 rom.asm 说明。rom.asm 的可重定位目标文件在库中。

```
        NAME    @rom
;
        PUBLIC  _?R_INIT , _?RLINIT , _?R_INIS
        PUBLIC  _?INIT , _?INITL , _?DATA , _?DATAL , _?INIS , _?DATS
;
@@R_INIT        CSEG    UNIT64KP                                    ; (1)
_?R_INIT :
@@RLINIT        CSEG    UNIT64KP
_?RLINIT :
@@R_INIS        CSEG    UNIT64KP
_?R_INIS :
@@INIT          DSEG    BASEP
_?INIT :
@@INITL         DSEG    UNIT64KP
_?INITL :
@@DATA          DSEG    BASEP
_?DATA :
@@DATAL         DSEG    UNIT64KP
_?DATAL :
@@INIS          DSEG    SADDRP
_?INIS :
@@DATS          DSEG    SADDRP
_?DATS :
;
END
```

<1>定义 ROMization 处理中使用的标签

定义了在 ROMization 处理中（a）到（d）所有变量使用的标签（见**(3) ROMization 处理**）。这些标签指示了存储每个变量初值的段的末地址。

8.3.3 修改启动例程

可以对 CC78K0R 提供的启动例程进行修改以满足具体目标系统的需求。本节将介绍修改这些文件的基本方法。

（1）修改启动例程时

下面对启动例程源程序文件的基本修改要点进行说明。修改后，使用 src\CC78K0R\bat 目录下的 mkstup.bat (mkstup.sh)对修改的源程序文件（cstart*.asm）(*：字母数字的符号)重新汇编。

- 标准库函数中使用的符号

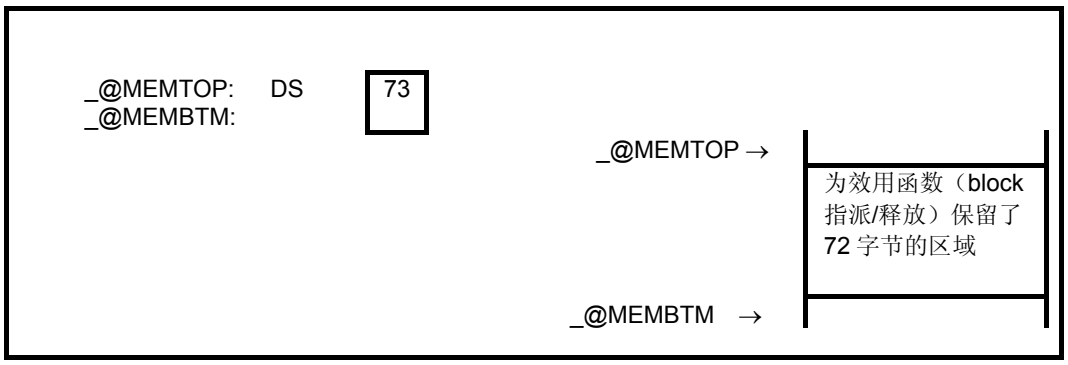
  如果没有使用下表中列出的库函数，在启动例程（cstart.asm）中这些函数对应的所有符号都可以删除。然而，由于启动例程中使用了 EXIT 函数，_@FNCTBL 和 _@FNCENT 不能删除（如果删除了 EXIT 函数，则这些符号也可以删除）。通过库的切换可以删除未使用的库函数对应符号。

库函数名	使用的符号
brk sbrk malloc calloc realloc free	_errno _@MEMTOP _@MEMBTM _@BRKADR
exit	_@FNCTBL _@FNCENT
rand srand	_@SEED
div	_@DIVR
ldiv	_@LDIVR
strtok	_@TOKPTR
atof strtod 数学函数 浮点运行库	_errno

- 效用函数中使用的区域（block 指派/释放）
  如果用户定义了效用函数（block 指派/释放）使用区域的大小，在下例中对此说明。

[例]

如果你想为效用函数（block 指派/释放）保留 72 字节空间，需对启动例程的初始设置做如下改动。

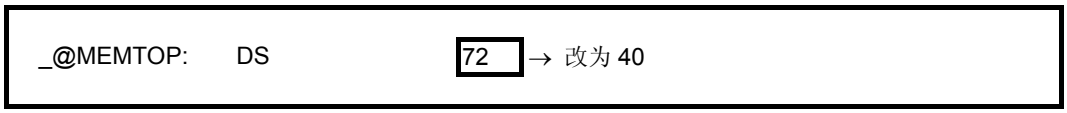


给区域的大小增加 1 个字节是为了保证安全，然后在启动例程中指定该值。在上述例子中，在启动例程中确保为 73 字节，但是效用函数最多能真正用到 72 字节。

如果指定的区域过大，无法放在 RAM 中，在连接时会发生错误。

因此，按照下列方法来减小指定空间，或通过修改连接指令文件来避免这种溢出错误，如下所示。对连接指令文件的修改请参阅 **(2) 连接指令文件**。

例）减少指定区域的大小



（2）连接指令文件

本节介绍如何创建连接指令文件。当连接到具体的目标系统时，使用-D 选项来指定一个现有的文件。创建文件时请注意下面的警告（连接指令的详细使用方法敬请参阅 RA78K0R 汇编程序包用户操作手册)）。

- CC78K0R 有时会使用部分短立即寻址区域（saddr 区域）来处理下面的编译器指定对象。
  特别是普通模型的 FFED8H 到 FFEDFH 的 44 字节。

(a) 当指定-QR 选项时的寄存器变量 [FFEB4H 到 FFEC3H]
(b) norec 函数的自动变量或参数[FFEC4H 到 FFED3H]
(c) 区段信息[FFED4H to FFED7H]
(d) 运行时刻库的参数[FFED8H to FFEDFH]
(e) 标准库任务（区域（b）和（c）的一部分）

如果用户不使用标准库，则不会涉及区域（e）的使用。

下面是使用连接指令文件(lk78K0R.dr)改变 RAM 空间大小的例子。当改变存储器空间大小时，不要与另外的区域重叠。当改变存储器大小时请参阅具体目标设备的内存映射情况。

<lk78K0R.dr>

```
                    首地址          大小
memory RAM:         (0FCF00h,   002F20h) →  将此空间扩大
memory SDR:         (0FFE20h,   000098h)    (如果需要也可以修改首地址)
merge @@INIS: =SDR                          → 指定区段的存储位置
merge @@DATS:=SDR                           → 指定区段的存储位置
merge @@BITS: =SDR                          → 指定区段的存储位置
```

如果你想改变段的存储位置，需要添加合并（Merge）语句。如果使用了修改编译器输出区块名称的函数，这个段可以独立定位（敬请参阅 **CC78K0R 语言篇 用户手册的第 11 章**）。

如果段的位置修改结果不能为定位内容提供足够的存储空间，则需改变相应的存储器声明语句。

（3）使用实时操作系统（RTOS）时

分别为 RX78K0R 和 CC78K0R 提供了初始化例程作为样例（汇编格式）。于是当同时使用 RX78K0R 和 CC78K0R 时，必须对各自的初始化例程作某些改变。

关于修改初始化例程的方法，敬请参阅 RX78K0R 功能用户手册。

8.4 flash区域中启动模块的ROM化处理

flash 区域中启动模块和普通启动模块的区别主要在以下几点。

表 8-4 初始化数据的 ROM 区域块

变量类型	区段	首标签	终止标签
带初值的外部变量 (a) （被分配在 near 区域）	@ER_INIT CSEG UNITP	E@R_INIT	E?R_INIT
带初值的外部变量 (a) （被分配在 far 区域）	@ERLINIT CSEG UNITP	E@RLINIT	E?RLINIT
带初值的 sreg 变量 (c)	@ER_INIS CSEG UNITP	E@R_INIS	E?R_INIS

表 8-5 拷贝目的地的 RAM 区域块

变量类型	段	首标签	终止标签
带初值的外部变量 (a) （被分配在 near 区域）	@EINIT DSEG UNITP	E@INIT	E?INIT
带初值的外部变量 (a) （被分配在 far 区域）	@EINITL DSEG UNITP	E@INITL	E?INITL
不带初值的外部变量 (b) （被分配在 near 区域）	@EDATA DSEG UNITP	E@DATA	E?DATA
不带初值的外部变量 (b) （被分配在 far 区域）	@EDATAL DSEG UNITP	E@DATAL	E?DATAL
带初值的 sreg 变量 (c)	@EINIS DSEG SADDRP	E@INIS	E?INIS
不带初值的 sreg 变量 (d)	@EDATS DSEG SADDRP	E@DATS	E?DATS

- 在启动模块中，下列标签被加到 ROM 区域和 RAM 区域中每个段的开始处。
  E@R_INIT, E@R_INIS, E@INIT, E@DATA, E@INIS, E@DATS, E@INITL, E@DATAL
  并且，如果指定为紧凑模式或大型模式，或者变量被分配在 far 区域，需要添加下列标签
  E@RLINIT, E@INITL, E@DATAL

- 在终止模块中，下列标签被加到 ROM 区域和 RAM 区域中每个段的终止位置处。
  E?R_INIT, E?R_INIS, E?INIT, E?DATA, E?INIS, E?DATS, E?RLINIT, E?INITL, E?DATAL

- 启动模块进行拷贝，将 ROM 区域中每个段的首标签位置开始到终止标签地址-1 位置的所有内容拷贝到 RAM 区域中，拷贝的目标地址为 RAM 区域每个段的首标签位置。

- 从 E@DATA 到 E?DATA，从 E@DATS 到 E?DATS 的内容都清零。

- 并且，如果指定为紧凑模式或大型模式，或者变量被分配在 far 区域，E@DATAL 至 E?DATAL 之间都被清零。

第 9 章 错误信息

9.1 错误信息格式

错误信息格式如下。

```
源文件名 (行号) ：错误信息
```

例

```
prime.c(8) :   CC78K0R 错误信息 E0712 Declaration syntax
prime.c(8) :   CC78K0R 错误信息 F0301 Syntax error
prime.c(8) :   CC78K0R 错误信息 E0701 External definition syntax
prime.c(19) : CC78K0R 警告信息 W0745 Expected function prototype
```

但是，内部错误 C0101，C0103 和 C0104 会使用下列格式输出。

```
[xxx.c <yyy> zzz] CC78K0R 错误信息 C0101 Internal error
[xxx.c <yyy> zzz] CC78K0R 错误信息 C0103 Intermediate file error
[xxx.c <yyy> zzz] CC78K0R 错误信息 C0104 Illegal use of register
```

备注　xxx.c: 源文件名
　　yyy: 行号
　　zzz: 信息

9.2 错误信息类型

编译器可能输出的错误信息有下列十种。

- 命令行错误信息
- 内部错误或存储器错误信息
- 字符错误信息
- 配置元素错误信息
- 转换的错误信息
- 表达式的错误信息
- 语句的错误信息
- 声明或函数定义的错误信息
- 预处理指令的错误信息
- 重要文件 I/O 和运行于非法操作系统的错误信息

9.3 错误信息列表

在使用错误信息列表之前，需要对错误编号的格式有所了解。错误编号说明了错误信息的类型，也说明了编译器对该错误的处理。

错误编号格式如下所示。

F/E/C/Wnnn

（1） 异常中止（Fxxxx）

如果发生此类错误，编译处理总是会停止。目标模块文件和汇编源程序模块文件都不会被输出。

（2） 严重错误（Exxxx）

如果此类错误发生的数量超过了某个特定值，编译处理会停止进行。目标模块文件和汇编源程序模块文件都不会被输出。

（3） 内部错误（Cxxxx）

在错误信息输出之后，编译器总是立即停止处理。目标模块文件和汇编源程序模块文件都不会被输出。

（4） 警告（Wxxxx）

在错误信息输出之后，编译处理继续进行。

备注 nnn (4-位数字)

类型	描述
从 0001 开始的编号	命令行错误信息
从 0101 开始的编号	内部错误或存储器错误信息
从 0201 开始的编号	字符错误信息
从 0301 开始的编号	配置元素错误信息
从 0401 开始的编号	转换的错误信息
从 0501 开始的编号	表达式的错误信息
从 0601 开始的编号	语句的错误信息
从 0701 开始的编号	声明或函数定义的错误信息
从 0801 开始的编号	预处理指令的错误信息
从 0901 开始的编号	重要文件 I/O 和运行于非法操作系统的错误信息

注意 如果文件名有错误语法，文件名会加入到错误信息中。

错误信息的添加、修改和删除要符合所使用的 C 编译器具体语言规格。

9.3.1 命令行错误信息

表 9-1 命令行错误信息

错误编号	错误信息	
F0001	信息	Missing input file
	原因	输入源文件名没有被指定。
	对策	输出“Please enter ‘CC78K0R--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的文件名。
F0002	信息	Too many input files
	原因	指定了多个输入源文件名。
	对策	输出“Please enter ‘CC78K0R--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的文件名。
F0003	信息	Unrecognized string
	原因	在交互命令行中指定的内容不是能够正确识别的选项。
F0004	信息	Illegal file name
	原因	在指定文件名时使用了不正确的格式、字符或数字。
F0005	信息	Illegal file specification
	原因	指定了一个非法的文件名。
F0006	信息	File not found
	原因	指定的输入文件不存在。
F0007	信息	Input file specification overlapped file name
	原因	指定的输入文件名有重名。
F0008	信息	File specification conflicted file name
	原因	指定的 I/O 文件名有重名。
F0009	信息	Unable to make file file name
	原因	因为指定的输出文件已经存在，并且是只读文件，所以不能被创建。
F0010	信息	Directory not found
	原因	输出文件名中指定的驱动或目录不存在。
F0011	信息	Illegal path
	原因	在路径参数设置选项中指定了非法路径名。
F0012	信息	Missing parameter ‘*option*’
	原因	某个必需的参数没有被指定。
	对策	输出“Please enter ‘CC78K0R--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的参数。

表 9-1 命令行错误信息

错误编号	错误信息	
F0013	信息	Parameter not needed ‘option’
	原因	指定了一个不必要的选项参数。
	对策	输出“Please enter ‘CC78K0R--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的参数。
F0014	信息	Out of range ‘option’
	原因	对选项指定的参数值超出了允许范围。
	对策	输出“Please enter ‘CC78K0R--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的参数值。
F0015	信息	Parameter is too long
	原因	在选项参数中的字符数目超出了限制范围。
F0016	信息	Illegal parameter ‘*option*’
	原因	在选项参数中有语法错误。
	对策	输出“Please enter ‘CC78K0R--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的选项。
F0017	信息	Too many parameters
	原因	选项参数的总数超出了限制。
F0018	信息	Option is not recognized ‘option’
	原因	指定了一个不正确的选项。
	对策	输出“Please enter ‘CC78K0R--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的选项。
F0019	信息	Parameter file nested
	原因	在参数文件中指定了-F 选项。
	对策	不能在参数文件中指定参数文件，对其进行修正，去除参数文件的嵌套。
F0020	信息	Parameter file read error
	原因	参数文件读取失败。
F0021	信息	Memory allocation failed
	原因	存储器分配失败
W0022	信息	Same category option specified – ignored ‘option’
	原因	指定了相互冲突的选项
	对策	最后指定的选项有效，并继续进行处理。
W0023	信息	Incompatible chip name
	原因	在命令行中指定的设备类型和源程序中指定的设备类型有冲突。
	编译	以命令行中指定的设备类型为准。
F0024	信息	Illegal chip specifier on command line
	原因	在命令行中指定的设备类型是不正确的。

表 9-1 命令行错误信息

错误编号		错误信息
W0029	信息	'-QC' option is not portable
	原因	-QC 选项不符合 ANSI 标准规格 (关于-QC 的细节，敬请参阅**第 5 章 编译选项**)。
W0031	信息	'-ZP' option is not portable
	原因	-ZP 选项不符合 ANSI 标准规格 (关于-ZP 的细节，敬请参阅**第 5 章 编译选项**)。
W0032	信息	'-ZC' option is not portable
	原因	-ZC 选项不符合 ANSI 标准规格 (关于-ZC 的细节， 敬请参阅**第 5 章 编译选项**)。
F0033	信息	Same category option specified 'option'
	原因	同时指定了多个相同冲突的选项。
	对策	输出"Please enter 'CC78K0R--' if you want help message"信息。 使用 --，-? 或 –H 选项来访问帮助文件，并对输入进行改正。
W0046	信息	'-ZF' option specified - regarded as '-QL1'
	原因	因为指定了 flash 区域目标创建选项-ZF，在库中的-QL2 代替常数代码模式，-QL 选项被当作-QL1 处理。
W0067	信息	'Option' option deleted - ignored
	原因	指定的选项已经被删除，'option' 被忽略。
W0068	信息	'Option 1' option deleted - regarded as 'option 2'
	原因	'Option 1' 被删除，并启用'option 2'

9.3.2 内部错误和内存错误信息

表 9-2 内部错误和内存错误信息

错误编号	错误信息	
C0101	信息	Internal error
	原因	内部错误发生。
	对策	请联系技术支持。
E0102	信息	Too many errors
	原因	严重错误的数量超过了 30。
	编译	处理继续进行，但后续的错误信息不被输出。先前的错误可能会引起更多的错误。所以首先请修改之前发生的错误。
C0103	信息	Intermediate file error
	原因	媒介文件有误。
	对策	请联系技术支持。
C0104	信息	Illegal use of register
	原因	使用寄存器的方法不正确。
E0105	信息	Register overflow : simplify expression
	原因	表达式过于复杂，导致没有空余的寄存器可用。
	对策	对引起错误的复杂表达式进行简化。
C0106	信息	Stack overflow ‘overflow cause’
	原因	堆栈溢出。可能是栈或堆发生溢出。
	对策	请联系技术支持。
E0108	信息	Compiler limit : too much automatic data in function
	原因	为函数中的自动变量分配的区域超过了 64 KB 的限制。
	对策	减少变量，使其所占的区域不超过 64 KB。
E0109	信息	Compiler limit : too much parameter of function
	原因	为函数中的参数文件分配的区域超过了 64 KB 的限制。
	对策	减少参数的使用，使其所占的区域不超过 64 KB。
E0110	信息	Compiler limit : too much code defined in file
	原因	为文件中的代码分配的区域超过了 64 KB 的限制。
E0111	信息	Compiler limit : too much global data defined in file
	原因	为文件中的全局变量分配的区域超过了 64 KB 的限制。

表 9-2 内部错误和内存错误信息

错误编号	错误信息	
E0113	信息	Compiler limit: too many local labels
	原因	在单个函数中定义的本地标签数量超过了处理限制。
	对策	函数本身过大，建议将其拆分。

9.3.3 字符错误信息

表 9-3 字符错误信息

错误编号	错误信息	
E0201	信息	Unknown character ‘hexadecimal number’
	原因	具有特定内部编码的字符无法被识别。
E0202	信息	Unexpected EOF
	原因	文件正在操作时，文件结束。
W0203	信息	Trigraph encountered
	原因	出现了三字符序列(3 字符显示)
	对策	如果指定了-ZA 选项，由于三字符序列有效，所以不输出警告。

9.3.4 配置元素错误信息

表 9-4 配置元素错误信息

错误编号	错误信息	
E0301	信息	Syntax error
	原因	语法错误发生。
	对策	请确认在源程序中不要出现描述错误。
E0303	信息	Expected identifier
	原因	goto 语句需要标识符。
	对策	正确描述 goto 语句使用的标识符。
W0304	信息	Identifier truncate to ‘identifier’
	原因	指定的标识符太长，标识符（包括下划线“_”）的字符数目超出了 250。
	对策	删减标识符的长度。
E0305	信息	Compiler limit : too many identifiers with block scope
	原因	在一个块中有过多的块范围指定符号。
E0306	信息	Illegal index , indirection not allowed
	原因	在表达式中使用索引，没有指针指向此索引。
E0307	信息	Call of non-function ‘variable name’
	原因	变量名作为函数名使用。
E0308	信息	Improper use of a typedef name
	原因	typedef 定义的名称使用方法不正确。
W0309	信息	Unused ‘variable name’
	原因	在源文件中声明的变量从未使用过。
W0310	信息	‘Variable name’ is assigned a value which is never used
	原因	指定的变量仅用在在赋值语句中，在其他地方从未使用过。
E0311	信息	Number syntax
	原因	常量表达式非法。
E0312	信息	Illegal octal digit
	原因	非法的八进制数。
E0313	信息	Illegal hexadecimal digit
	原因	非法的十六进制数。

表 9-4 配置元素错误信息

错误编号	错误信息	
E0314	信息	Too big constant
	原因	常量太大超出了表示范围，无法显示。
E0315	信息	Too small constant
	原因	常量过小无法表示。
E0316	信息	Too many character constants
	原因	字符常量超过了两个字符。
E0317	信息	Empty character constant
	原因	字符常量‘ ’为空 。
E0318	信息	No terminated string literal
	原因	在字符串最后缺少双引号“”。
E0319	信息	Changing string literal
	原因	字符串文字被重写。
W0320	信息	No null terminator in string literal
	原因	字符串文字的末尾缺少 null 字符。
E0321	信息	Compiler limit : too many characters in string literal
	原因	在字符串文字的字符数目超过了 509 个。
E0322	信息	Ellipsis requires three periods
	原因	编译器检测到“..”，但省略号需要三个点“...”。
E0323	信息	Missing ‘delimiter’
	原因	分界符不正确。
E0324	信息	Too many }’s
	原因	‘{’ 和 ‘}’ 没有正确配对。
E0325	信息	No terminated comment
	原因	注释的最后没有使用“*/”来终止。
E0326	信息	Illegal binary digit
	原因	非法的二进制数。
E0327	信息	Hex constants must have at least one hex digit
	原因	在十六进制常量表达式中至少需要一个十六进制数。
W0328	信息	Unrecognized character escape sequence ‘character’
	原因	转义序列符号无法正确识别。
E0329	信息	Compiler limit : too many comment nesting
	原因	注释嵌套的层次超过了 255 层的限制。

表 9-4 配置元素错误信息

错误编号	错误信息	
E0332	信息	Non-supported keyword found-ignored 'function attributes' in this file
	原因	检测到不支持的关键字，该文件中的函数属性被忽略
W0340	信息	Unreferenced label 'label name'
	原因	已经定义的标签从来没有被引用过。
E0342	信息	function qualifier' keyword is not allowed
	原因	无法使用函数修饰符。

9.3.5 转换错误信息

表 9-5 转换错误信息

错误编号	错误信息	
W0401	信息	Conversion may lose significant digits
	原因	Long 型被转换成 int.型。请注意部分数值可能会丢失。
E0402	信息	Incompatible type conversion
	原因	在赋值语句中发生了非法的类型转换。
E0403	信息	Illegal indirection
	原因	在整型表达式中使用了 * 操作符。
E0404	信息	Incompatible structure type conversion
	原因	对于结构体赋值的语句左右两边类型不同。
E0405	信息	Illegal lvalue
	原因	非法的左值。
E0406	信息	Cannot modify a const object ‘variable name’
	原因	const 属性的变量被重写。
E0407	信息	Cannot write for read / only sfr ‘SFR name’
	原因	尝试向只读属性的 sfr 中写入数值。
E0408	信息	Cannot read for write/only sfr ‘SFR name’
	原因	尝试从只写属性的 sfr 中读取数值。
E0409	信息	Illegal SFR access ‘sfr name’
	原因	对 sfr 进行非法操作，比如从 sfr 中强行读取数据，或向 sfr 写入非法数据。
W0410	信息	Illegal pointer conversion
	原因	指针和非指针类型的目标进行转换。
W0411	信息	Illegal pointer combination
	原因	在相同类型的指针组合中，混入了不同类型。
W0412	信息	Illegal pointer combination in conditional expression
	原因	在条件表达式中使用不同类型的指针组合。
W0413	信息	Illegal structure pointer combination
	原因	指向结构体的指针被混合使用，结构体包含多种不同的数据类型。
E0414	信息	Expected pointer
	原因	需要一个指针。

错误编号	错误信息	
W0415	信息	Conversion may lose significant digits for far pointer
	原因	试图将 far 指针转换为 near 指针或者 int 型。注意其值可能会丢失。
W0416	信息	Illegal type and size (far/near) pointer combination
	原因	在同一种指针合并时，使用了不同类型或不同长度（far 或 near 指针）。
W0417	信息	Illegal type and size (far/near) pointer combination in conditional expression
	原因	在指针之间的条件表达式中，使用了不同类型或不同长度（far 或 near 指针）。
W0418	信息	Illegal structure and size (far/near) pointer combination
	原因	指向结构体的指针使用了不同类型或不同长度（far 或 near 指针）。

9.3.6 表达式错误信息

表 9-6 表达式错误信息

错误编号	错误信息	
E0501	信息	Expression syntax
	原因	表达式包含了一个语法错误。
E0502	信息	Compiler limit : too many parentheses
	原因	在表达式中的括号嵌套超过 32 层。
W0503	信息	Possible use of ‘variable name’ before definition
	原因	在变量被赋值之前使用了该变量。
W0504	信息	Possibly incorrect assignment
	原因	在条件表达式例如 if，while 和 do 语句中的主要运算符是赋值运算符。
W0505	信息	Operator ‘operator’ has no effect
	原因	在程序中，运算符无效。这可能是由于语法描述错误。
E0507	信息	Expected integral index
	原因	数组的索引中只允许使用整型表达式。
W0508	信息	Too many actual arguments
	原因	在函数调用时指定的参数数量大于函数定义时参数类型列表中指定的参数数量。
W0509	信息	Too few actual arguments
	原因	在函数调用时指定的参数数量小于函数定义时参数类型列表中指定的参数数量。
W0510	信息	Pointer mismatch in function ‘function name’
	原因	给定的参数指针类型和函数定义时参数类型列表中指定的参数类型不符。
W0511	信息	Different argument types in function ‘function name’
	原因	函数调用时给出的参数类型和参数类型列表不符，或者和函数定义时指定的参数类型不符。
E0512	信息	Cannot call function in norec function
	原因	在 norec 函数中发生了函数调用。在 norec 函数中调用函数是被禁止的。
E0513	信息	Illegal structure / union member ‘member name’
	原因	被引用的结构体成员找不到对应的定义成员。
E0514	信息	Expected structure / union pointer
	原因	在‘→’运算符之前的表达式不是一个指向结构体或共用体的指针，但这个表达式是一个结构体或共用体的名称。
	对策	确保出现在‘→’运算符之前的表达式是指向结构体或共用体的指针。

表 9-6 表达式错误信息

错误编号	错误信息	
E0515	信息	Expected structure / union name
	原因	在‘.’运算符之前的表达式不是一个结构体或共用体的名称，但这个表达式是一个指向结构体或共用体的指针。
	对策	确保出现在‘→’运算符之前的表达式是结构体或共用体的名称。
E0516	信息	Zero sized structure ‘structure name’
	原因	结构体的容量大小是 0。
E0517	信息	Illegal structure operation
	原因	使用了某个在结构体中无法支持的操作符。
E0518	信息	Illegal structure / union comparison
	原因	两个结构体或共用体无法进行比较。
E0519	信息	Illegal bit field operation
	原因	对位域的非法描述。
E0520	信息	Illegal use of pointer
	原因	指针支持的操作符有加号、减号、赋值号、取值符号(*)和成员访问符号(->)。
E0521	信息	Illegal use of floating
	原因	使用了某个不能对浮点类型变量使用的操作符。
W0522	信息	Ambiguous operators need parentheses
	原因	在没有使用括号的情况下连续使用两个以上的移位操作符、关系操作符和位操作符。
E0523	信息	Illegal bit, boolean type operation
	原因	执行了位变量或布尔型变量的非法操作。
E0524	信息	‘&’ requires lvalue
	原因	A 的常量地址没有被获得。
E0525	信息	&’ requires lvalue
	原因	‘&’ 操作符只能用于向左侧的子表达式赋值的表达式中。
E0526	信息	‘&’ on register variable
	原因	某寄存器变量的地址无法获取。
E0527	信息	‘&’ on bit, boolean ignored
	原因	某个位字段地址或 bit 型或布尔类型变量的地址无法获取。
W0528	信息	‘&’ is not allowed array / function, ignored
	原因	&操作符不能应用于数组名或函数名。
E0529	信息	Sizeof returns zero
	原因	Sizeof 表达式的值变成 0。
E0530	信息	Illegal sizeof operand
	原因	Sizeof 表达式的操作数必须是标识符或类型名。

表 9-6 表达式错误信息

错误编号	错误信息	
E0531	信息	Disallowed conversion
	原因	非法指向发生。
	对策	检查非法指向。 当常量被用作指针或地址超出了内存模块的范围时，会有这条错误发生。
E0532	信息	Pointer on left, needs integral right : ‘operator’
	原因	因为左操作数是指针，要求右操作数一定是整数。
E0533	信息	Invalid left-or-right operand : ‘operator’
	原因	左操作数或右操作数和运算符不匹配。
E0534	信息	Divide check
	原因	/运算符或%运算符的除数是零。
E0535	信息	Invalid pointer addition
	原因	两个指针不能相加。
E0536	信息	Must be integral value addition
	原因	只有整数可以和指针相加。
E0537	信息	Illegal pointer subtraction
	原因	两个指针之间的减法要求两个指针必须具有相同的类型。
E0538	信息	Illegal conditional operator
	原因	条件运算符的描述不正确。
E0539	信息	Expected constant expression
	原因	需要常量表达式。
W0540	信息	Constant out of range in comparison
	原因	用来和常量部分表达式比较的值超过了其他部分表达式的类型允许范围。
E0541	信息	Function argument has void type
	原因	函数的参数类型是 void 型。
W0543	信息	Undeclared parameter in noauto or norec function prototype
	原因	该参数的声明没有出现在 noauto 或 norec 函数的原型声明中。
E0544	信息	Illegal type for parameter in noauto or norec function prototype
	原因	在 noauto 或 norec 函数的原型声明中指定的参数类型是非法的。

表 9-6 表达式错误信息

错误编号	错误信息	
E0546	信息	Too few actual argument for inline function ‘function name’
	原因	内联（Inline）函数展开时，函数调用所指定的参数数目少于定义时指定的参数数目。

9.3.7 语句错误信息

表 9-7 语句错误信息

错误编号	错误信息	
E0602	信息	Compiler limit : too many characters in logical source line
	原因	在源程序的一个逻辑行中出现的字符数量超过了 2048。
E0603	信息	Compiler limit : too many labels
	原因	标签的数目超过了 33 个。
E0604	信息	Case not in switch
	原因	Case 语句出现的位置不正确。
E0605	信息	Duplicate case ‘label name’
	原因	在一个 switch 语句中出现了两个以上同样的 case 标签。
E0606	信息	Non constant case expression
	原因	在 case 语句中指定了整数常量之外的数据。
E0607	信息	Compiler limit : too many case labels
	原因	在 switch 语句中出现的 case 标记数目超过了 257 个。
E0608	信息	Default not in switch
	原因	default 语句出现的位置不正确。
E0609	信息	More than one ‘default’
	原因	在 switch 语句中出现多个 default 语句。
E0610	信息	Compiler limit : block nest level too deep
	原因	块嵌套超过了 45 层。
E0611	信息	Inappropriate ‘else’
	原因	If 和 else 语句不匹配。
W0613	信息	Loop entered at top of switch
	原因	在 switch 语句之后紧跟有 while，do 或者 for 语句。
W0615	信息	Statement not reached
	原因	此语句永远执行不到。
E0617	信息	Do statement must have ‘while’
	原因	do 语句之后一定要加上 While 语句来对应。
E0620	信息	Break / continue error
	原因	break 和 continue 语句的位置不正确。
E0621	信息	Void function ‘function name’ cannot return value
	原因	声明为 void 的函数却收到一个返回值。

表 9-7 语句错误信息

错误编号	错误信息	
W0622	信息	No return value
	原因	应该返回一个值的函数没有返回任何值。
	对策	如果某个值必须要返回，添加 return 语句。如果没有必要向函数返回值，将函数定义为 void 类型。
E0623	信息	No effective code and data, cannot create output file
	原因	因为代码和数据无效，输出文件无法创建。

9.3.8 声明和函数定义的错误信息

表 9-8 声明和函数定义的错误信息

错误编号	错误信息	
E0701	信息	External definition syntax
	原因	函数没有被正确定义。
E0702	信息	Too many callt functions
	原因	声明为 callt 型的函数数量太多，最多可以定义 32 个 callt 函数。
	对策	减少 callt 函数的声明数量。
E0703	信息	Function has illegal storage class
	原因	函数被指定的存储类是非法的。
E0704	信息	Function returns illegal type
	原因	函数返回值的类型是非法的。
E0705	信息	Too many parameters in noauto or norec function
	原因	noauto 或 norec 函数的参数太多。
	对策	减少参数数目。
E0706	信息	Parameter list error
	原因	函数参数列表有误。
E0707	信息	Not parameter ‘character string’
	原因	在函数定义时声明的参数不正确。
E0710	信息	Illegal storage class
	原因	自动的寄存器声明在函数外部，或布尔变量被定义在函数内部。
E0711	信息	Undeclared ‘variable name’; function ‘function name’
	原因	使用的变量未进行声明。
E0712	信息	Declaration　syntax
	原因	声明语句不符合语法规格。
E0713	信息	Redefined ‘variable name’
	原因	定义了两个以上的同名变量。
	对策	变量定义进行一次即可。

表 9-8 声明和函数定义的错误信息

错误编号	错误信息	
W0714	信息	Too many register variables
	原因	寄存器变量的声明太多。
	对策	减少寄存器变量的数量。至于可用的寄存器数量，敬请查阅 CC78K0R 编译器语言篇用户手册。
E0715	信息	Too many sreg variables
	原因	sreg 变量的声明太多。
E0717	信息	Too many automatic data in noauto or norec function
	原因	在 noauto 或 norec 函数中有太多自动变量。
	对策	减少 noauto 或 norec 函数中自动变量的数量。至于允许的使用数量，敬请查阅 CC78K0R 编译器语言篇用户手册。
E0718	信息	Too many bit, boolean type variables
	原因	位变量和布尔型变量太多。
	对策	减少位变量、布尔型和__boolean 型变量的数量。至于允许的使用数量，敬请查阅 CC78K0R 编译器语言篇用户手册。
E0719	信息	Illegal use of type
	原因	使用非法的类型名称。
E0720	信息	Illegal void type for ‘identifier’
	原因	标识符用 void 进行声明。
W0721	信息	Illegal type for register declaration
	原因	寄存器声明时被指定了非法类型。
	编译器	寄存器声明被忽略，处理过程继续进行。
E0723	信息	Illegal type for parameter in noauto or norec function
	原因	在 noauto 或 norec 函数中的参数类型太大。
E0724	信息	Structure redefinition
	原因	定义了两个以上的同名结构体。
W0725	信息	Illegal zero sized structure member
	原因	用来存放结构体成员的区域无法保证。
	对策	当数组被用作结构体成员，索引需要用常量计算给出，使用-QC2 选项有时候会产生溢出，并且成员的区域也无法保证。在这种情况下，在-QC 的位置指定-QC1 选项。-QC 选项本身是默认的选项之一。
E0726	信息	Function cannot be structure / union member
	原因	函数不能作为结构体或共用体的成员。
E0727	信息	Unknown size structure / union ‘name’
	原因	结构体或共用体的大小容量未定义。

表 9-8 声明和函数定义的错误信息

错误编号		错误信息
E0728	信息	Compiler limit : too many structure / union members
	原因	结构体或共用体中的成员数量超过 256。
E0729	信息	Compiler limit : structure / union nesting
	原因	结构体或共用体的嵌套层数超过 15。
E0730	信息	Bit field outside of structure
	原因	位域的声明放在结构体之外。
E0731	信息	Illegal bit field type
	原因	在位域类型中指定的类型不是整型的。
E0732	信息	Too long bit field size
	原因	位域声明中指定的位变量数量超过了类型允许的位数容量。
E0733	信息	Negative bit field size
	原因	在位域声明中的位变量数量是负值。
E0734	信息	Illegal enumeration
	原因	枚举类型声明不符合语法规格。
E0735	信息	Illegal enumeration constant
	原因	非法的枚举常量。
E0736	信息	Compiler limit : too many enumeration constants
	原因	枚举常量的数量超过 255。
E0737	信息	Undeclared structure / union / enum tag
	原因	某个未声明的标记（tag）。
E0738	信息	Compiler limit : too many pointer modifying
	原因	在指针定义中，间接运算符(*)的数量超过 12 个。
E0739	信息	Expected constant
	原因	在用于数组声明的索引中使用了变量。
E0740	信息	Negative subscript
	原因	数组的容量大小被指定为负值。
E0741	信息	Unknown size array ‘array name’
	原因	数组的容量大小未做定义。
	对策	指定数组的大小。
E0742	信息	Compiler limit : too many array modifying
	原因	数组的声明超过了 12 维。
E0743	信息	Array element type cannot be function
	原因	函数数组是不被允许的。

表 9-8 声明和函数定义的错误信息

错误编号	错误信息	
W0744	信息	Zero sized array ‘array name’
	原因	定义的数组中，元素数量为 0。
W0745	信息	Expected function prototype
	原因	函数原型未做声明。
E0747	信息	Function prototype mismatch
	原因	函数原型的声明有误。
	对策	检查函数的参数和返回值的类型是否匹配。
W0748	信息	A function is declared as a parameter
	原因	在声明中将函数用作参数。
W0749	信息	Unused parameter ‘parameter name’
	原因	参数没有被使用。
E0750	信息	Initializer syntax
	原因	初始化不符合语法规格。
E0751	信息	Illegal initialization
	原因	为变量设置的初始值使用了类型不匹配的常量。
W0752	信息	Undeclared initializer name ‘name’
	原因	初始名称未做声明。
E0753	信息	Cannot initialize static with automatic
	原因	静态变量不能用自动变量来进行初始化。
E0756	信息	Too many initializers ‘array name’
	原因	初值的数量多于数组声明中的元素数量。
E0757	信息	Too many structure initializers
	原因	初值的数量多于结构体声明中的成员数量。
E0758	信息	Cannot initialize a function ‘function name’
	原因	该函数无法完成初始化。
E0759	信息	Compiler limit : initializers too deeply nested
	原因	需要初始化的元素嵌套深度超过限制。
W0760	信息	Double and long double are treated as IEEE 754 single format
	原因	双精度和长双精度按照 IEEE 754 的规定，当作单精度格式来处理。
W0761	信息	Cannot declare sreg with const or function
	原因	不能将常数或函数声明为 sreg 型。
	编译器	sreg 声明被忽略。

表 9-8 声明和函数定义的错误信息

错误编号	错误信息	
W0762	信息	Overlapped memory area ‘variable name 1’ and ‘variable name 2’
	原因	变量名 1 和变量名 2 区域的绝对地址分配说明被重叠执行。
W0763	信息	Cannot declare const with bit, boolean
	原因	位变量和布尔类型变量在声明不能用常量赋值。
	编译器	常量声明被忽略。
W0764	信息	‘Variable name’ initialized and declared extern-ignored extern
	原因	定义为外部变量的变量被初始化，在别的文件中却找不到被引用变量的对应定义。
	编译器	extern 声明被忽略。
E0765	信息	Undefined static function ‘function name’
	原因	引用了一个静态函数，在该文件的找不到对应的定义，或者该函数定义了却不是静态的。
E0766	信息	Illegal type for automatic data in noauto or norec function
	原因	noauto 或 norec 函数中的自动变量类型太大。
E0769	信息	__far is not allowed for callt/interrupt function
	原因	对于 callt 和中断函数，不能使用__far 修饰符。
E0770	信息	Parameters are not allowed for interrupt function
	原因	中断函数不能带参数使用。
E0771	信息	Interrupt function must be void type
	原因	中断函数必须是 void 类型。
E0772	信息	Callt / noauto / norec / _ _pascal are not allowed for interrupt function
	原因	中断函数不能被声明 callt, noauto, norec, 或 _ _pascal 类型。
E0773	信息	Cannot call interrupt function
	原因	中断函数不能被调用。
E0774	信息	Interrupt function can’t use with the other kind interrupts
	原因	中断函数不能在其他类型的中断服务函数中使用。
E0775	信息	Cannot call rtos_task function
	原因	无法调用 RTOS 任务
E0776	信息	Cannot call ret_int / ret_wup except in rtos_interrupt_handler
	原因	ret_int / ret_wup 系统调用无法完成，除非在 RTOS_interrupt 句柄中。
E0777	信息	Not call ret_int / ret_wup in rtos_interrupt_handler
	原因	ret_int / ret_wup 系统调用不是在 RTOS_interrupt 句柄中完成调用。
E0778	信息	Cannot call ext_tsk in interrupt function
	原因	ext_tsk 系统调用无法在中断函数/ 中断句柄中完成。

表 9-8 声明和函数定义的错误信息

错误编号	错误信息	
W0779	信息	Not call ext_tsk in rtos_task
	原因	ext_tsk 系统调用无法在 RTOS 任务中完成。
E0780	信息	Zero width for bit field ‘member name’
	原因	成员的位域声明中占用空间为 0，这种成员名无法指定。
W0787	信息	Bit field type is char
	原因	字符类型被指定为位域类型。
E0788	信息	Cannot allocate a __flash function '*function name*'
	原因	其中一个__flash 函数无法分配。
E0789	信息	'-ZF' option did not specify - cannot allocate an EXT_FUNC function '*function name*'
	原因	指定了 Flash 存储器区域目标创建选项-zf。 在#pragma EXT_FUNC 语句指定的函数不能被分配到该区域。
E0790	信息	callt/__interrupt are not allowed for EXT_FUNC function '*function name*'
	原因	在#pragma EXT_FUNC 语句指定的函数不能使用 callt/__interrupt 进行声明。
E0791	信息	'-ZF' option specified - cannot allocate a callt function 'function name'
	原因	指定了 Flash 存储器区域目标创建选项-zf。 不能用于存放 callt 函数。
E0799	信息	Cannot allocate ‘variable name’ out of ‘address range’
	原因	为变量名分配的绝对地址超过了指定的地址范围。

9.3.9 预处理命令的错误信息

表 9-9 预处理命令的错误信息

错误编号	错误信息	
E0801	信息	Undefined control
	原因	以 # 开始的符号不能被识别为关键字。
E0802	信息	Illegal preprocess directive
	原因	非法的预处理命令。
	对策	检查预处理命令(如 #pragma)是否写在头文件前面，并且检查是否有错误。
E0803	信息	Unexpected non-whitespace before preprocess directive
	原因	在预处理命令之前有纯空格之外的字符出现。
W0804	信息	Unexpected characters following ‘preprocess directive’ directive - newline expected
	原因	预处理命令之后，在同一行有无法识别为预处理命令的额外字符。
E0805	信息	Misplaced else or elif
	原因	#if，#ifdef，和#ifndef 与#else 和#elif 的配对不相符。
E0806	信息	Misplaced endif
	原因	#if，#ifdef，和 #ifndef 与#endif 的配对不相符。
E0807	信息	Compiler limit:too many conditional inclusion nesting
	原因	条件编译的嵌套层数超过 255。
E0810	信息	Cannot find include file ‘file name’
	原因	找不到包含文件。
	对策	重新指定存在包含文件的路径，或使用环境变量 INC78K0R 的 -i 选项来指定一个路径。
E0811	信息	Too long file name ‘file name’
	原因	文件名太长。
E0812	信息	Include directive syntax
	原因	#include 语句中的文件名没有被正确的包含在“ ”或< >符号中。
E0813	信息	Compiler limit : too many include nesting
	原因	包含文件的嵌套层数超过 8。
E0814	信息	Illegal macro name
	原因	非法的宏名。
E0815	信息	Compiler limit: too many macro nesting
	原因	宏的嵌套层数超过 200。

表 9-9 预处理命令的错误信息

错误编号	错误信息	
W0816	信息	Redefined macro name ‘macro name’
	原因	宏名称有重定义。
W0817	信息	Redefined system macro name ‘macro name’
	原因	系统宏名称有重定义。
E0818	信息	Redeclared parameter in macro ‘macro name’
	原因	宏定义的参数列表中出现相同的标识符。
W0819	信息	Mismatch number of parameter ‘macro name’
	原因	引用时的参数数量和#define 命令定义的参数数量不符。
E0821	信息	Illegal macro parameter ‘macro name’
	原因	在函数格式的宏定义中，圆括号( )中的描述内容是非法的。
E0822	信息	Missing) ‘macro name’
	原因	在函数格式的宏定义中，在#define 关键字的同一行中没有被找到对应的右括号‘)’。
E0823	信息	Too long macro expansion ‘macro name’
	原因	宏展开式时使用的实际参数太长。
W0824	信息	Identifier truncate to ‘macro name’
	原因	宏名称太长。
	编译器	‘宏名称’被截短显示。
W0825	信息	Macro recursion ‘macro name’
	原因	#define 定义的宏名称是递归的。
E0826	信息	Compiler limit : too many macro defines
	原因	宏定义的数量超过 10,000。
E0827	信息	Compiler limit : too many macro parameters
	原因	单个宏定义的调用参数超过 31 个。
E0828	信息	Not allowed #undef for system macro name
	原因	系统宏指令名被#undef 指定停止。
W0829	信息	Unrecognized pragma ‘character string’
	原因	#pragma 后的字符串无法识别。
	对策	检查关键字的正确性。 如果在#pragma 中指定了不正确的区段，会出现此警告。
E0830	信息	No chip specifier : #pragma pc ( )
	原因	没有设备说明符。
E0831	信息	Illegal chip specifier : #pragma pc (device type)
	原因	非法的设备说明符。
W0832	信息	Duplicated chip specifier
	原因	多个相同的设备说明符。

表 9-9 预处理命令的错误信息

错误编号	错误信息	
E0833	信息	Expected #asm
	原因	没有 #asm。
E0834	信息	Expected #endasm
	原因	没有#endasm。
W0835	信息	Too many characters in assembler source line
	原因	汇编源程序中的某行太长。
W0836	信息	Expected assembler source
	原因	在 #asm 和 #endasm 之间没有汇编源程序。
W0837	信息	Output assembler source file, not object file
	原因	C 源程序中有一个 #asm 块或_ _asm 语句。输出汇编源程序文件，而不会输出目标文件。
	对策	指定-a 或-sa 编译选项来将#asm 和__asm 语句输出到目标文件，然后对输出的 asm 文件进行汇编。
E0838	信息	Duplicated pragma VECT or INTERRUPT ‘character string’
	原因	有多个相同的#pragma VECT‘字符串’或#pragma INTERRUPT‘字符串’声明。
E0839	信息	Unrecognized pragma VECT or INTERRUPT ‘character string’
	原因	有一个无法识别的#pragma VECT‘字符串’或 INTERRUPT‘字符串’。
W0840	信息	Undefined interrupt function ‘function name’ -ignored NOBANK or LEAFWORK specified
	原因	为一个未定义的中断函数指定了存储目录。
	编译器	NOBANK 说明或 LEAFWORK 说明被忽略。
E0842	信息	Unrecognized pragma SECTION ‘character string’
	原因	有一个无法识别的#pragma SECTION‘字符串’。
E0843	信息	Unspecified start address of ‘section name’
	原因	在#pragma 部分中 AT 后没有指定正确的起始地址。
E0845	信息	Cannot allocate ‘section name’ out of ‘address range’
	原因	指定的区块无法放入指定的起始地址。
W0846	信息	Rechanged section name ‘section name’
	原因	定义了相同的节名，但是说明部分却被更改。
	编译器	最后指定的节名是有效的，并且处理过程继续进行。
E0847	信息	Different NOBANK or LEAFWORK specified on same interrupt function ‘function name’
	原因	不同的 NOBANK 说明或 LEAFWORK 说明被指定给同一个中断函数。

表 9-9 预处理命令的错误信息

错误编号	错误信息	
W0849	信息	#pragma statement is not portable
	原因	#pragma 语句不符合 ANSI 标准规格。
W0850	信息	Asm statement is not portable
	原因	ASM 语句不符合 ANSI 标准规格。
W0851	信息	Data aligned in ‘area name’
	原因	段区域或结构体标记是数据对齐的。区域名称是段名称或结构体标记。
W0852	信息	Module name truncate to ‘module name’
	原因	指定的模块名太长。
	编译器	‘模块名’被截短显示。
E0853	信息	Unrecognized pragma NAME ‘module name’
	原因	‘模块名’中有无法识别的字符。
E0854	信息	Undefined rtos_task ‘character string’
	原因	RTOS 任务体没有定义。
E0855	信息	Cannot assign rtos_interrupt_handler to non-maskable and software interrupt
	原因	不能在 RTOS_INTERRUPT 句柄中指定不可屏蔽中断和软件中断。
W0856	信息	Rechanged module name ‘module name’
	原因	指定了多个相同的模块名。
W0857	信息	Section name truncate to ‘section name’
	原因	指定的节名（section name）太长。
	编译器	‘节名’被截短显示。节名被截为 8 个或更少字符。
E0858	信息	Unrecognized pragma 'pragma character string' 'illegal character string'
	原因	存在无法识别的#pragma 'pragma 字符串'，非法的字符串
E0859	信息	Cannot allocate EXT_TABLE out of 0x80-0xff80
	原因	Flash 存储器区域分支表的起始地址必须处于 0x80-0xff80 之间。
E0860	信息	Redefined #pragma EXT_TABLE
	原因	#pragma EXT_TABLE 有重复定义。

表 9-9 预处理命令的错误信息

错误编号	错误信息	
E0861	信息	No EXIT_TABLE specifier
	原因	没有指定 flash 区域分支表起始地址。
E0862	信息	Illegal EXT_FUNC id specifier : out of 0x0-0xff
	原因	在 Flash 存储器区域分支表中，通过#pragma EXT_FUNC 指定的函数 ID 值必须为 0x80-0xff80
E0863	信息	Redefined #pragma EXT_FUNC name 'function name'
	原因	用#pragma EXT_FUNC name 指定的函数名称有重复。
E0864	信息	Redefined #pragma EXT_FUNC id 'ID value'
	原因	用#pragma EXT_FUNC id 指定的函数 ID 值有重复。通过#pragma EXT_FUNC 指定的函数无法正常分配。
E0865	信息	Out of range - cannot allocate an EXT_FUNC function 'function name'
	原因	Flash 存储器区域分支表的地址超出了指定的地址范围。
E0866	信息	#pragma section found after C body. cannot include file containing #pragma section and without C body at the line
	原因	在 C 程序体描述后有#pragma 语句。包含的头文件中不能只有#pragma 语句而没有 C 程序体（包含变量和函数的外部引用声明）。
E0867	信息	#pragma section found after C body.　cannot specify #include after #pragma section in this file
	原因	在 C 程序体描述后有#pragma 语句。于是，#include 语句不能出现。
E0868	信息	#include found after C body.　cannot specify #pragma section after #include directive
	原因	在 C 程序体描述后有#include 语句。于是，#pragma 语句不能出现。
W0869	信息	‘section name’ section cannot change after C body
	原因	指定的节名在 C 程序体之后不能被改变。
W0870	信息	Data aligned before ‘variable name’ in ‘section name’
	原因	在变量名称分配到节名称之前，先执行数据对齐。
W0871	信息	Data aligned after ‘variable name’ in ‘section name’
	原因	在变量名称分配到节名称之后，再执行数据对齐。
E0899	信息	Character string specified by #error is output
	原因	指定了#error 字符串。

9.3.10 致命的文件I/O和运行非法操作系统的错误信息

表 9-10 致命的文件 I/O 和运行非法操作系统的错误信息

错误编号	错误信息	
F0901	信息	File I/O error
	原因	在文件输入/输出过程中产生物理 I/O 错误。
	对策	如果是中间文件引起了这个错误，请增加常规存储器，或者使用 EMS 或 XMS 存储器。
F0902	信息	Cannot open ‘file name’
	原因	文件无法打开。
	对策	检查设备文件是否被安装在常规查询路径。这个路径可以使用 –Y 选项来指定。参照 **5.4 设备文件搜索路径**中的查询路径的相关描述。
F0903	信息	Cannot open overlay file ‘file name’
	原因	覆盖文件无法打开。
F0904	信息	Cannot open temp
	原因	输入的临时文件无法打开。
F0905	信息	Cannot create ‘file name’
	原因	产生了一个文件创建错误。
F0906	信息	Cannot create temp
	原因	在输出临时文件时产生了一个创建错误。
	对策	检查是否指定了环境变量 TMP。
F0907	信息	No available data block
	原因	临时文件无法创建，因为驱动文件没有足够的存储容量。
F0908	信息	o available directory space
	原因	临时文件无法创建，因为驱动器上没有足够的目录区。
F0909	信息	R/O : read / only disk
	原因	临时文件无法创建，因为驱动器的属性是只读的。
F0910	信息	R/O file : read / only , file opened read / only mode
	原因	由于以下原因，临时文件会产生写错误。 1. 有相同名字的临时文件已经存在于驱动器上，并且是只读属性。 2. 由于内部的冲突，输出临时文件以只读属性被打开。
F0911	信息	Reading unwritten data, no available directory space
	原因	由于以下原因会产生 I/O 错误。 在 EOF 之后输入继续。 临时文件无法创建，因为驱动器上没有足够的目录区。

表 9-10 致命的文件 I/O 和运行非法操作系统的错误信息

错误编号	错误信息	
F0912	信息	Write error on temp
	原因	在输出临时文件时产生了一个写入错误。
	对策	可能是因为源程序表达式过于复杂 (如非常深的嵌套)而引起的，请联系技术支持。
F0914	信息	Insufficient memory in hostmachine
	原因	因为内存不足，编译器无法启动。
	对策	增加常规存储器的可用区域。
W0915	信息	Asm statement found. skip to jump optimize this function ‘function name’
	原因	检测到#asm 块或__asm 声明语句。这个函数没有进行跳转优化。执行 W0837 对策。
E0922	信息	Heap overflow : please retry compile without -QJ
	原因	在转移优化时产生存储器溢出错误。取消-QJ 选项，并重新编译。
F0923	信息	Illegal device file format
	原因	引用了旧格式的设备文件。

9.4 PM+错误信息列表

表 9-11 PM+错误信息

错误类型	错误信息	
!	信息	Out of range. The range of columns is from 72 to 132.
	原因	每行出现的字符数量不在规格的范围内。
	对策	指定一个在规格允许范围内的值，然后重新执行。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Out of range. The range of lines is from 0, and 20 to 32,767.
	原因	每页出现的行数超过了规格的范围。
	对策	指定一个在规格允许范围内的值，然后重新执行。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Out of range. The range of TAB character is from 0 to 8.
	原因	每个制表符的长度超过了规格的范围。
	对策	指定一个在规格允许范围内的值，然后重新执行。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Out of range. The range of warning level is from 0 to 2.
	原因	出现的警告级别不在规格的范围。
	对策	指定一个在规格允许范围内的值，然后重新执行。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Can not find folder. Will you create?
	原因	目录根本不存在。
	对策	点击[OK] 按钮来创建一个新目录。点击[Cancel]按钮来取消目录的创建。
	按钮	[OK]，创建一个新目录并关闭信息对话框。 [Cancel]，关闭信息对话框而不创建新目录。
!	信息	Can not find drive. Make sure the drive.
	原因	未找到指定的驱动器名称。
	对策	指定正确的驱动器。
	按钮	[Retry]，重试对该驱动器的访问。 [Cancel]，该访问被取消。

错误类型	错误信息	
!	信息	Illegal File name.
	原因	文件名中包括的某些字符不能被操作系统或 CC78K0R 接受。
	对策	不要使用操作系统无法处理的字符，或者"#"或"，"等，对它们 CC78K0R 不能接受。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Unable to create the folder.
	原因	目录创建被操作系统拒绝，因为可用空间不足，操作目录是只读的，等原因。
	对策	检测可用的磁盘剩余空间，以及是否允许写入。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Ignored Options.
	原因	项目文件中的选项信息包括有多个选项的合并，会导致 CC78K0R 的警告或错误。
	对策	检查是否有相互冲突的选项。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Can't read Option Information.
	原因	选项信息读取规格指定的文件中未包含有效的选项信息。
	对策	检查指定的选项信息文件是否为 78K0R 系列，或者指定的选项信息文件是否已损坏。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Can not find path or file. Make sure the path or filename.
	原因	必须指定某个实际存在的路径或文件名称时，比如对于包含文件，指定了不存在的路径或文件。
	对策	检查目标文件名称和其路径，然后指定正确的文件名称和路径。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Cannot find %s shown in environment variable PATH.
	原因	未找到 cc78k0r.exe
	对策	检查 CC78K0R 的正常安装路径，或者是否正确设置了 PATH 环境变量。
	按钮	点击[OK]，关闭信息对话框。

错误类型	错误信息	
!	信息	Multiple Include Search Path definition.
	原因	同一个包含文件路径被指定了两次。
	对策	不要对同一个包含文件路径指定两次。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Too many characters for Include Search Path.
	原因	指定的包含文件路径，其长度超过了可以允许的范围。
	对策	指定正确的路径名称。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Too many Include Search Path. Up to 64 can be specified for Include Search Path.
	原因	指定的包含文件路径数量超过了可以允许的范围。
	对策	保证指定路径的数量不超过 64。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Multiple define definition.
	原因	同一个宏定义被指定了两次。
	对策	不要对同一个宏定义指定两次。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Too many characters for macro Definition. Up to 256 characters can be described for a macro name.
	原因	宏定义中的字符长度超过了允许范围。
	对策	保证宏定义中的字符长度不超过 256 个。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Too many macro for macro Definition. macro Definition and macro Undefinition can be specified to 30 in all.
	原因	定义的宏和未定义（undefined）的宏数量超过了允许范围。
	对策	保证宏定义和宏的反定义数量总共不超过 30 个。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Multiple undefine definition.
	原因	同一个宏的反定义被指定了两次。
	对策	不要对同一个宏的反定义指定两次。
	按钮	点击[OK]，关闭信息对话框。

错误类型	错误信息	
!	信息	Too many characters for undefine Definition. Up to 256 characters can be described for a macro name.
	原因	未定义中的宏中字符长度超过了允许范围。
	对策	保证宏的未定义中的字符长度不超过 256 个。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Too many macro for macro Undefinition. macro Definition and macro Undefinition can be specified to 30 in all.
	原因	定义的宏和未定义（undefined）的宏数量超过了允许范围。
	对策	保证宏定义和宏的未定义数量总共不超过 30 个。
	按钮	点击[OK]，关闭信息对话框。
!	信息	Can't set options correctly to (source name)
	原因	当试图将公共选项反映到特殊选项，在规格中发现有冲突，或者指定的规格超过了数量限制。
	对策	不能反映的选项规格将被忽略。 如果需要，检查特殊选项设置。
	按钮	点击[OK]，关闭信息对话框。

附录A 样例程序

本章介绍 CC78K0R 使用的样例程序。

A.1 C 源程序模块文件

```
#define TRUE    1
#define FALSE   0
#define SIZE    200

char   mark[SIZE+1];

main()
{
        int     i, prime, k, count;

        count = 0;

        for ( i = 0 ; i <= SIZE ; i++)
                mark[i] = TRUE;
        for ( i = 0 ; i <= SIZE ; i++) {
                if (mark[i]) {
                        prime = i + i + 3;
                        printf("%6d",prime);
                        count++;
                        if((count%8) == 0)     putchar('\n');
                        for ( k = i + prime ; k <= SIZE ; k += prime)
                                mark[k] = FALSE;
                }
        }
        printf("\n%d primes found.",count);
}

printf ( char *s , int i )
{
        int     j ;
        char   *ss ;
        j = i ;
        ss = s ;
}
putchar ( char c )
{
        char   d ;
        d = c ;
}
```

A.2 执行例程

```
C>CC78K0R –cf1166a0 prime.c -a -p -x -e -ng
```

```
78K0R Series C Compiler Vx.xx [xx xxx xxxx]
Copyright (C) NEC Electronics Corporation xxxx, xxxx

prime.c ( 18 ) : CC78K0R warning W0745 : Expected function prototype
prime.c ( 20 ) : CC78K0R warning W0745 : Expected function prototype
prime.c ( 26 ) : CC78K0R warning W0622 : No return value
prime.c ( 35 ) : CC78K0R warning W0622 : No return value
prime.c ( 41 ) : CC78K0R warning W0622 : No return value

Target chip : uPD78F1166_A0
Device file : Vx.xx
Compilation complete, 0 error(s) and 5 warning(s) found.
```

A.3 输出列表

A.3.1 汇编源程序模块文件

```
; 78K0R Series C Compiler Vx.xx Assembler Source
;                                                Date : xx xxx xxxx Time : xx : xx : xx
; Command       : -cf1166a0 prime.c -a -p -x -e -ng
; In-file       : prime.c
; Asm-file      : prime.asm
; Para-file     :

$PROCESSOR ( f1166a0 )
$NODEBUG
$NODEBUGA
$KANJICODE EUC
$TOL_INF        03FH ,   100H ,          02H ,       00H ,       00H
                EXTRN       _@RTARG0
                EXTRN       @@isrem
                PUBLIC       _mark
                PUBLIC      _main
                PUBLIC       _printf
                PUBLIC       _putchar
@@CNST CSEG     MIRRORP
L0011 :         DB          '%6d'
                DB          00H
L0017 :         DB          ' '
                DB          0AH
                DB          '%d primes found.'
                DB          00H

@@DATA DSEG     BASEP
_mark :         DS    ( 201 )
                DS    ( 1 )

; line 5
; line 8

@@CODE CSEG     BASE
_main :
                push        hl                                  ; [ INF ] 1 , 1
                subw        sp , #08H                           ; [ INF ] 2 , 1
                movw        hl , sp                             ; [ INF ] 3 , 1
; line 11
                clrw        ax                                  ; [ INF ] 1 , 1
                movw        [ hl ] , a                ; count   ; [ INF ] 1 , 1
; line 13
                movw        [ hl + 6 ] , ax           ; i       ; [ INF ] 2 , 1
L0003 :
                movw        ax , [ hl + 6 ]           ; i       ; [ INF ] 2 , 1
                cmpw        ax , #0C8H                ; 200     ; [ INF ] 3 , 1
                orl         CY , a.7                            ; [ INF ] 2 , 1
                skc                                             ; [ INF ] 2 , 1
                bnz     $L0004                                  ; [ INF ] 2 , 4
; line 14
                movw        ax , [ hl + 6 ]           ; i       ; [ INF ] 2 , 1
                movw        bc , ax                             ; [ INF ] 1 , 1
                mov         _mark [ bc ] , #01H       ; 1       ; [ INF ] 4 , 1
                movw        ax , [ hl + 6 ]           ; i       ; [ INF ] 2 , 1
```

```
            incw        ax                          ; [ INF ] 1 , 1
            movw        [ hl + 6 ] , ax     ; i     ; [ INF ] 2 , 1
            br      $L0003                          ; [ INF ] 2 , 4
L0004 :
; line 15
            clrw    ax                              ; [ INF ] 1 , 1
            movw        [ hl + 6 ] , ax     ; i     ; [ INF ] 2 , 1
L0006 :
            movw        ax , [ hl + 6 ]     ; i     ; [ INF ] 2 , 1
            cmpw        ax , #0C8H          ; 200   ; [ INF ] 3 , 1
            or1         CY , a.7                    ; [ INF ] 2 , 1
            skc                                     ; [ INF ] 2 , 1
            bnz     !L0007 ; [ INF ] 2 , 4
; line 16
            movw        ax , [ hl + 6 ]     ; i     ; [ INF ] 2 , 1
            addw        ax , #loww ( _mark )        ; [ INF ] 3 , 1
            movw        de , ax                     ; [ INF ] 1 , 1
            mov         a , [ de ]                  ; [ INF ] 2 , 2
            cmp0        a                           ; [ INF ] 1 , 1
            bz          $L0015                      ; [ INF ] 2 , 4
; line 17
            movw        ax , [ hl + 6 ]     ; i     ; [ INF ] 2 , 1
            addw        ax , ax                     ; [ INF ] 1 , 1
            addw        ax , #03H           ; 3     ; [ INF ] 3 , 1
            movw        [ hl + 4 ] , ax     ; prime ; [ INF ] 2 , 1
; line 18
            push        ax                          ; [ INF ] 1 , 1
            movw        ax , #loww ( L0011 ) ; 0    ; [ INF ] 3 , 1
            call        !!_printf                   ; [ INF ] 4 , 3
            pop         ax                          ; [ INF ] 1 , 1
; line 19
            movw        ax , [ hl ]         ; count ; [ INF ] 1 , 1
            incw        ax                          ; [ INF ] 1 , 1
            movw        [ hl ] , ax         ; count ; [ INF ] 1 , 1
; line 20
            movw        _@RTARG0 , ax               ; [ INF ] 2 , 1
            movw        ax , #08H           ; 8     ; [ INF ] 3 , 1
            call        !!@@isrem                   ; [ INF ] 3 , 3
            or          a , x                       ; [ INF ] 2 , 1
            bnz         $L0012                      ; [ INF ] 2 , 4
            movw        ax , #0AH           ; 10    ; [ INF ] 3 , 1
            call        !!_putchar                  ; [ INF ] 4 , 3
L0012 :
; line 21
            movw        ax , [ hl + 6 ]     ; i     ; [ INF ] 2 , 1
            xchw        ax , bc                     ; [ INF ] 1 , 1
            movw        ax , [ hl + 4 ]     ; prime ; [ INF ] 2 , 1
            addw        ax , bc                     ; [ INF ] 1 , 1
            movw        [ hl + 2 ] , ax     ; k     ; [ INF ] 2 , 1
L0014 :
            movw        ax , [ hl + 2 ]     ; k     ; [ INF ] 2 , 1
            cmpw        ax , #0C8H          ; 200   ; [ INF ] 3 , 1
            or1          CY , a.7                   ; [ INF ] 2 , 1
            skc                                     ; [ INF ] 2 , 1
            bnz         $L0015                      ; [ INF ] 2 , 4
; line 22
            movw        ax , [ hl + 2 ]     ; k     ; [ INF ] 2 , 1
            movw        bc , ax                     ; [ INF ] 1 , 1
            mov         _mark [ bc ] , #00H ; 0     ; [ INF ] 4 , 1
            movw        ax , [ hl + 2 ]     ; k     ; [ INF ] 2 , 1
            xchw        ax , bc                     ; [ INF ] 1 , 1
```

```
                movw    ax , [ hl + 4 ]     ; prime     ; [ INF ] 2 , 1
                addw    ax , bc                         ; [ INF ] 1 , 1
                movw    [ hl + 2 ] , ax     ; k         ; [ INF ] 2 , 1
                br      $L0014                          ; [ INF ] 2 , 4
L0015 :
; line 24
                movw    ax , [ hl + 6 ]     ; i         ; [ INF ] 2 , 1
                incw    ax                              ; [ INF ] 1 , 1
                movw    [ hl + 6 ] , ax      ; i        ; [ INF ] 2 , 1
                br      $L0006                          ; [ INF ] 2 , 4
L0007 :
; line 25
                movw    ax , [ hl ]         ; count     ; [ INF ] 1 , 1
                push    ax                              ; [ INF ] 1 , 1
                movw    ax , #loww ( L0017 ) ; 0        ; [ INF ] 3 , 1
                call    !!_printf                       ; [ INF ] 4 , 3
                pop     ax                              ; [ INF ] 1 , 1
; line 26
                addw    sp , #08H                       ; [ INF ] 2 , 1
                pop     hl                              ; [ INF ] 1 , 1
                ret                                     ; [ INF ] 1 , 6
; line 29
_printf :
                push    hl                              ; [ INF ] 1 , 1
                push    ax                              ; [ INF ] 1 , 1
                subw    sp , #04H                       ; [ INF ] 2 , 1
                movw    hl , sp                         ; [ INF ] 3 , 1
; line 33
                movw    ax , [ hl + 12 ]    ; i         ; [ INF ] 2 , 1
                movw    [ hl + 2 ] , ax     ; j         ; [ INF ] 2 , 1
; line 34
                movw    ax , [ hl + 4 ]     ; s         ; [ INF ] 2 , 1
                movw    [ hl ] , ax         ; ss        ; [ INF ] 1 , 1
; line 35
                addw    sp , #04H                       ; [ INF ] 2 , 1
                pop     hl                              ; [ INF ] 1 , 1
                ret                                     ; [ INF ] 1 , 6
; line 38
_putchar :
                push    hl                              ; [ INF ] 1 , 1
                movw    hl , ax                         ; [ INF ] 1 , 1
; line 40
                mov     a , l                           ; [ INF ] 1 , 1
                mov     h , a                           ; [ INF ] 1 , 1
; line 41
                pop     hl                              ; [ INF ] 1 , 1
                ret                                     ; [ INF ] 1 , 6
                END

; *** Code Information ***
;
; $FILE /auto/proj/cmp/cc.new/work/egashira/cc78sk0r/src/test/prime2.c
;
; $FUNC main ( 8 )
;               bc = ( void )
;               CODE SIZE = 155 bytes , CLOCK_SIZE = 117 clocks , STACK_SIZE = 16 bytes
;
; $CALL printf ( 18 )
;               bc = ( pointer : ax , int : [ sp + 2 ] )
;
```

```
; $CALL putchar ( 20 )
;               bc = ( int : ax )
;
; $CALL printf ( 25 )
;               bc = ( pointer : ax , int : [ sp + 2 ] )
;
; $FUNC printf ( 29 )
;               bc = ( pointer s : ax , int i : [ sp + 4 ] )
;               CODE SIZE = 18 bytes , CLOCK_SIZE = 16 clocks , STACK_SIZE = 8 bytes
;
; $FUNC putchar ( 38 )
;               bc = ( char c : x )
;               CODE SIZE = 6 bytes , CLOCK_SIZE = 11 clocks , STACK_SIZE = 2 bytes

; Target chip : uPD78F1166_A0
; Device file : Vx.xx
```

A.3.2 预处理列表文件

```
/*
78K0R Series C Compiler Vx.xx Preprocess List                    Date : xx xxx xxxx Page : 1
Command         : -cf1166a0 prime.c -a -p -x -e -ng
In-file         : prime.c
PPL-file        : prime.ppl
Para-file       :
*/

1 :     #define     TRUE        1
2 :     #define     FALSE       0
3 :     #define     SIZE        200
4 :
5 :     __far char mark [ SIZE + 1 ] ;
6 :
7 :     main ( )
8 :     {
9 :                 int i , prime , k , count ;
10 :
11 :                count = 0 ;
12 :
13 :                for ( i = 0 ; i <= SIZE ; i++ )
14 :                        mark [ i ] = TRUE ;
15 :                for ( i = 0 ; i <= SIZE ; i++ ) {
16 :                        if ( mark [ i ] ) {
17 :                                prime = i + i + 3 ;
18 :                                printf ( "%6d", prime ) ;
19 :                                count++ ;
20 :                                if ( ( count%8 ) == 0 ) putchar ( '\n' ) ;
21 :                                        for ( k = i + prime ; k <= SIZE ; k += prime )
22 :                                        mark [ k ] = FALSE ;
23 :                                }
24 :                 }
25 :                printf ( "\n%d primes found." , count ) ;
26 :    }
27 :
28 :    printf ( char *s , int i )
29 :    {
30 :                int j ;
31 :                char *ss ;
32 :
33 :                j = i ;
34 :                ss = s ;
35 :    }
36 :
37 :     putchar ( char c )
38 :    {
39 :                char d ;
40 :                d = c ;
41 :    }

/*
Target chip : uPD78F1166_A0
Device file : Vx.xx
*/
```

A.3.3 交叉引用列表文件

```
78K0R Series C Compiler Vx.xx Cross reference List
                                Date : XX XXX XXXX Page : 1
Command         : -cf1166a0 prime.c -a -p -x -e –ng
In-file         : prime.c
Xref-file       : prime.xrf
Para-file       :

ATTRIB   MODIFY   TYPE     SYMBOL   DEFINE  REFERENCE
EXTERN   FAR      array    mark     5       14    16    22
EXTERN   FAR      func     main     7
AUTO1    int        i     9        13      13    13    14    15    15    15    16
                                    17      17    21
AUTO1             int      prime    9       17    18    21    21
AUTO1             int      k        9       21    21    21    22
AUTO1             int      count    9       11    19    20    25
EXTERN   FAR      func     printf   28      18    25
EXTERN   FAR      func     putchar  37      20
PARAM             pointer  s        28      34
PARAM             int      i        28      33
AUTO1             int      j        30      33
AUTO1             pointer  ss       31      34
REG1              char     c        37      40
PARAM
REG1              char     d        39      40
         #define  TRUE     1        14
         #define  FALSE    2        22
         #define  SIZE     3        5       13    15    21

Target chip : uPD78F1166_A0
Device file : Vx.xx
```

A.3.4 错误列表文件

```
prime.c ( 18 ) : CC78K0R warning W0745 : Expected function prototype
prime.c ( 20 ) : CC78K0R warning W0745 : Expected function prototype
prime.c ( 26 ) : CC78K0R warning W0622 : No return value
prime.c ( 35 ) : CC78K0R warning W0622 : No return value
prime.c ( 41 ) : CC78K0R warning W0622 : No return value


Target chip : uPD78F1166_A0
Device file : Vx.xx
Compilation complete, 0 error(s) and 5 warning(s) found.
```

附录B 注意事项列表

本章介绍使用 CC78K0R 的注意事项。

表 B-1 相关注意事项列表

编号	注意事项
1	**[关于指定选项的注意事项]** (a) 当某选项不允许指定多个参数却被赋予了多个参数时，最后一个指定参数优先（有效）。 (b) 所有在-C 选项后的类型规格都不能忽略。如果被忽视，会产生一个异常中止错误。 如果没有指定-C 选项，请确保在 C 源程序模块文件中输入#pragma pc(类型)。 在编译期间，如果具体指定的选项和 C 源程序中的选项不同，则指定的选项优先。同时会输出一个警告信息。 (c) 如果帮助选项被指定，所有其它选项可以被忽视。
2	**[关于文件输出目标的注意事项]** 对于目标模块文件，只能输出为磁盘类型的文件。
3	**[关于错误信息的注意事项]** 当文件中发现一个语法错误，就会有一条出错信息关联到这个文件名。 如果设备文件被指定到禁止位置，就会自动输出指定的字符串。在其它所有情况下，必须加上驱动器、路径和文件扩展名。
4	**[源文件名的注意事项]** 在 CC78K0R 中，除去文件名后缀的部分（主文件名）默认被用作模块名。因此，使用的源文件名会有一些限制。. (a) 关于文件名的长度，在操作系统允许的范围内给文件名配置一个基本名和扩展名，基本名和扩展名之间用点(.)来间隔。 (b) 文件基本名和后缀名的字符由系统允许的字符组成，除了圆括号(())，分号(;)和逗号(,)。应该注意的是连字符(-)可以被作为文件名或者路径的首字符。指定的文件名或者路径名绝对不能包括 2 字节字符。 (c) 在文件连接时，如果有基本名前 8 个字符相同的文件，系统将会报错。 (d) (#)符号不能出现在参数文件的文件名和路径名中。
5	**[关于包含文件的注意事项]** 在包含文件中定义函数（除了函数声明），然后在 C 源程序中展开该文件是不可能的。 当在包含文件中出现定义，那么在源程序调试过程中会出现某些问题，比如定义行的错误显示等。

编号	注意事项
6	**[输出汇编源程序的使用注意事项]** 当 C 源程序包括了#asm 块或 __asm 语句的汇编语言时，装载模块文件产生的过程顺序是编译、汇编，然后连接。 对下列几种使用情况要留心，当使用汇编器输出汇编源程序来进行汇编操作，而不是通过 CC78K0R 编译器输出直接目标时，例如，当使用汇编语言进行描述时。 (a) 需要在#asm 块（在#asm 和#endasm 语句之间的部分）和__asm 语句中定义符号时，要使用字符串应该少于 8 个字符，并以?L@开始（例如，?L@01，?L@sym，等）。然而，这些符号不能在被定义为外部的（PUBLIC 声明）。在#asm 块中和__asm 语句中不能定义区段。如果定义的符号没有以?L@开始，那么在汇编过程中会产生 F2114 异常中止信息。 (b) 在 C 源程序#asm 块中使用的变量是外部变量时，如果在 C 程序中其它位置没有对此变量引用，就不会有 EXTRN 声明，同时产生一个连接错误。因此如果 C 中没有引用，那么在#asm 块中要进行 EXTRN 声明。 (c) 如果 C 源程序中包含#asm 块和__asm 语句，指定-A 或–SA 编译选项来开启汇编描述，并且对输出的汇编源程序进行汇编。 当使用 PM +时，或者对源程序单独指定-A/-SA 选项，或者在通用选项中指定-A/-SA 选项，单独指定的话只输出汇编源文件。 (d) 当使用 PM +时，如果指定了-A 或-SA 编译选项，那么不论-O/-NO 选项如何，都会启动 RA78K0R。 (e) 当使用#pragma 命令改变段名称时，段名称不要和源程序基本名相同。否则汇编时会产生异常中止错误 F2106。

编号	注意事项
7	**[创建连接指示文件]（link directive file）** 当连接编译器生成的目标文件时，如果超出了目标设备的 ROM 或 RAM 区域范围；或者需要把代码或数据放在任意指定的地址时，可以创建一个连接定向文件，并且在连接的时候指定-D 选项。 关于创建连接定向文件的详细信息，敬请参阅 **RA78K0R 汇编程序包用户手册操作篇**以及编译器自带的 lk78K0R.dr（在 SMP78K0R 目录下）文件。 **例)** 要把一个没有初始值的外部变量从某个特定 C 源程序文件放到外部存储器中 **(1).** 在 C 源程序的开始，为这个没有初始值的外部变量改变段名称。 #pragma section @@DATA EXTDATA ⋮ **注意 通过改变启动例程来对更名的段进行初始化，ROMization 也同样要在启动例程中描述。** **(2).** 创建一个连接指示文件。 <lk78K0R.dr> memory EXTRAM : (040000h, 1000h) merge EXTDATA := EXTRAM 创建连接定向文件时注意下面几点。 • 当连接时使用-S 自动生成选项来生成堆栈符号，推荐通过连接定向文件的存储器定位来保留堆栈区域，并指定唯一的堆栈名称。如果省略了对该区域的命名，保留出的区域就被用作 RAM 中的堆栈区（SFR 区域除外）。 **例)** 向连接定向文件 lk78K0R.dr 中添加 memory EXTRAM : (040000h, 1000h) memory STK : (0FB000H, 100H) merge EXTDATA := EXTRAM (命令行) > lk78K0R s0rml.rel prime.rel -bcl0rm.lib -SSTK –Dlk78K0R.dr • 在自己定义的存储器区域进行连接时可能会遇到以下错误。 “*** ERROR E3206 Segment ‘xxx’ can’t allocate to memory-ignored.” **[原因]** 由于已定义的存储器区域空间不足，无法存放指定段。 **[对策]** 应对步骤大致分为以下三步。 1. 检查无法分配的程序段所需空间 (参阅.map 文件)。 2. 在第一步检查到的段大小的基础上，增加指示文件中对应段的预留空间。 3. 对定向文件指定-D 选项并连接。 然而，因为在步骤 1 中发生错误的区段类型不同，用来检查段空间的方法也有如下几种方法。 (1) 当程序段在编译过程中被自动生成时，借助连接过程中创建的 map 文件来检查段的空间大小。 (2) 如果这个段是用户创建的，借助汇编列表文件检查未分配的段空间大小(.prn)。

编号	注意事项
8	**[使用 va_start 宏时的注意事项]** 在 stdarg.h 中定义的 va_start 宏操作是不确定的（因为第一个参数的偏移量根据函数而不同） • 指定了第一个参数后，使用 va_starttop • 指定了第二个及以后的参数后，使用 va_start 宏。
9	**[启动例程和库]** (a) 使用系统提供的启动例程和库，版本要和可执行程序中的文件保持一致。(CC78K0R.exe 或 CC78K0R)。 (b) 对于支持浮点的函数比如 sprintf，vprintf 和 vsprintf，如果已经指定了转换说明符“%f”, “%e”, “%E”, “%g”或“%G”的转换结果比指定精确度还准确，那么转换值就向下舍去。即使指定为“%g”“%G”的转换结果值大于精确度，仍然会执行“%f”转换。 对于 sscanf 和 scanf 函数，指定转换说明符为“%f”, “%e”, “%E”, “%g”，或“%G”，如果在转换期间没有读入有效的字符，那么转换的结果就是+0。如果转换结果是“±”, 那么±0 就被认为是转换结果。 **[预防方法]** 无

编号	注意事项
10	**[执行 ROMization]** ROMization 的内容包括配放初始值，例如那些在 ROM 中有初始值的外部变量，然后在系统运行期间将这些值拷贝到 RAM 中。在 CC78K0R 中，默认会生成 ROMization 相关代码。因此，在连接时执行含有 ROMization 的启动例程是很有必要的。 小型模式和中等模式所用的启动例程都不包括对 far 区域的 ROMization 处理。当变量因为使用了__far 修饰符或类似声明为被分配在 far 区域内时，需要使用紧凑模式和大型模式的启动例程。 下列是 C 编译器提供的启动例程，都包括有 ROMization 处理。 如果使用了 flash 存储器的自写入模式，请参考“8.3.3 使用 RTOS 时”。 启动例程： (1) 当不使用 C 标准库区域时：S0RM.REL，S0RL.REL (2) 当使用 C 标准库区域时：S0RML.REL，S0RLL.REL **[使用实例]** C>lk78k0r s0rl.rel sample.rel -s -bcl0rxm.lib -bcl0rm.lib -osample.lmf SAMPLE.REL: 用户程序的目标模块文件 S0RL.REL: 启动例程 CL0RXM.LIB: 使用乘法器的库 CL0RM.LIB: 运行时刻库，标准库 **-S 选项是堆栈符号 (_@STBEG, _@STEND)自动生成选项。** **注意 1：确保在开始时连接启动例程。** **注意 2：创建库时，同 CC78K0R 提供的库分开创建，同时在连接过程中指定它的优先级高于编译程序库。** **注意 3：不要向 CC78K0R 库中添加用户函数。** **注意 4：当使用浮点库(CL0R*F.LIB)时，包含 ROMization 处理的启动例程必须连接到标准库和浮点库。** 当使用支持浮点的 sprintf，sscanf，printf，scanf，vprintf，和 vsprintf 函数时 **例)** -BMYLIB.LIB -BCL0RMF.LIB -BCL0RM.LIB 当使用不支持浮点的 sprintf，sscanf，printf，scanf，vprintf，和 vsprintf 函数时 **例)** -BMYLIB.LIB -BCL0RM.LIB -BCL0RMF.LIB
11	**[生成堆栈区域符号(-S)]** 在 CC78K0R 中，用户无法为堆栈区保留空间。为了保留堆栈区的空间，在连接时需要指定-S 选项。 在 PM +中，当源文件中包含 C 源程序时，-S 选项会自动指定。

编号	注意事项
12	**[ROM 代码]** 当使用 ROM 代码时，指定目标转换选项为-R 或-U，例如–r，-u0FFH（不取消规格） **（例）** -R -U0FFH -R： 根据地址顺序排列十六进制文件目录 -U 填充值：向 ROM 区的空白处填充指定值。
13	**[帮助说明选项]** 在 PM +中，编译器选项--，-?和-H 都会被忽略，它们用来对选项作简单解释。 在 Tool 菜单下<Option Setup>对话框中单击“帮助”按钮，就可以看到对应的帮助文档。
14	**[-LL 选项说明]** 使用 PM +时，-LL 选项能够接受的最大数是 32767。如果指定的数字大于 32767，那么-LL 要和其他选项联合使用。

编号	注意事项
15	**[使用 PM +时注意]** (a) 用户创建的参数文件 当 PM +指定使用由用户创建的参数文件时，那些内容被装载到 PM +创建的参数文件中。 当建立参数文件时，注意以下几点，否则在建立（build）时会发生错误。 指定的文件名要和 PM +创建的参数文件同名。 设备类型描述选项(-c)，设备文件搜索路径描述选项(-y)和源文件都不要指定。 并不对用户创建的参数文件中所描述的选项进行有效性验证。 (b) <汇编选项> 对话框 不要指定-C，-F 和-Y 选项和源文件，否则在编译执行期间会输出一个错误。 对于<汇编选项>对话框中指定的选项并不作有效性验证，因此如果有描述错误，在执行建立（build）时会产生错误。 (c) 包含文件的从属关系 用 PM +创建 MAKE 文件时，会检查包含文件的从属关系，条件语句如 #if 会被忽视。因此有可能把非必需的包含文件误当作必需的文件。如果用注释或字符串形式来描述，那么这些包含文件就能够被正确判定，像无依赖关系的独立文件一样。 （例） #if 0 #include "header1.h" /* Dependence relationship judged to exist */ #else / * ! zero */ #include "header2.h" #endif /* #include "header3.h" */ 在从属关系检查中，header1.h 被判定为必须编译的。如果 header1.h 文件存在，那么 header1.h 就会引入到 PM +的” ProjectWindow”中。 **[预防方法]** 无。但是这对建立（build）过程没有影响 。 (d) 工程相关文件的设置 可以从 PM +的[Project]菜单或工程窗体中的"Add Project-Related File"添加或删除编译器属性的例行程序和标准库。 从<Compiler Options> 对话框的<<Startup Routine>>标签页中进行编译器附带的启动例程和标准库的设置。
16	**[关于原型声明的注意事项]** 如果一个函数原型声明没有包括函数类型描述，就会产生(E0301, E0701)错误。 **例)** f (void) ; /* E0301：语法错误 */ /* E0701：外部定义语法 */ **[预防方法]** 加上函数类型描述符 **例)** int f (void) ;

<table>
<tr><th>编号</th><th>注意事项</th></tr>
<tr><td>17</td><td>[关于错误信息输出的注意事项]

如果关键字中有拼写错误，且位于函数外的行首处，那么报告的错误行号会有偏差，还会输出一个不适当的错误提示。

例)
extren int i ;　　/* extern 拼写错误，不会有错误指向这里*/
/* 注释 */
void f (void) ;
[EOF]　　/* Error such as E0712 */
[预防方法] 无</td></tr>
<tr><td>18</td><td>[关于预处理指令中的注释描述]

在描述预处理命令时，如果有注释和函数类型宏放在同一行，且位于预处理命令之前或夹杂其中，那么会导致错误（E0803, E0814, E0821，等）。
例)
/* com1 */ #pragma sfr /* E0803 */
/* com2 */ #define ONE 1 /* E0803 */
#define /* com3 */ TWO 2 /* E0814 */
#ifdef /* com4 */ ANSI_C /* E0814 */

/* com5 */ #endif
#define SUB(p1, /* com6 */ p2) p2 = p1 /* E0821 */
[预防方法] 将注释内容移至预处理命令之后。
例)
#pragma sfr /* com1 */
#define ONE 1 /* com2 */
#define TWO 2 /* com3 */
#ifdef ANSI_C /* com4 */

#endif /* com5 */
#define SUB(p1, p2) p2 = p1 /* com6 */</td></tr>
</table>

编号	注意事项

19	**[关于使用结构体，共用体或枚举型（enum）特征标记的注意事项]**
	
	如果在函数定义之前的原型声明中使用特征标记(对结构体，共用体或枚举型)，若满足下面条件(1)，会产生警告，如果满足下面条件(2)，会输出错误。
	(1) 如果特征标记出现在函数的参数声明中，并定义了一个指向结构体或共用体的指针，当此函数被调用时，会输出 W0510 警告信息。
	例)
	`void func ( int , struct st ) ;`
	`struct st {`
	`char memb1;`
	`char memb2;`
	`} st [ ] = {`
	`{ 1, 'a' } , { 2, 'b' }`
	`} ;`
	`void caller ( void ) {`
	`/* W0510 Pointer mismatch */`
	`func ( sizeof ( st ) / sizeof ( st[0] ) , st );`
	`}`
	(2) 如果特征标记用于参数声明中的返回值类型说明，并指定为结构体，共用体或枚举型，则会输出 E0737 错误信息。
	例)
	`void func1( int , struct st ) ;` /* E0737 未声明的 structure/union/enum tag */
	`struct st func2 ( int ) ;` /* E0737 未声明的 structure/union/enum tag */
	`struct st {`
	`char memb1;`
	`char memb2;`
	`} ;`
	[预防方法] 预先定义结构体，共用体或枚举型的特征标记。

20	**[关于函数中的数组、结构体或共用体初始化的注意事项]**
	
	只有静态变量地址、常量或字符串可以才能完成数组、结构体或共用体的初始化。
	例)
	`void f ( void ) ;`
	`void f ( void ) {`
	`char *p, *p1, *p2 ;`
	`char *ca[3] = { p , p1 , p2 } ;` /* 错误(E0750) */
	`}`
	[预防方法] 用赋值语句来代替初始化。
	例)
	`void f ( void ) ;`
	`void f ( void ) {`
	`char *ca[3] ;`
	`char *p, *p1, *p2 ;`
	`ca[0] = p ; ca[1] = p1 ; ca[2] = p2 ;`
	`}`

编号	注意事项

21	**[关于外部 callt 函数的注意事项]**
	
	如果在函数表初始化时引用了外部 callt 函数的地址，而且此 callt 函数在这个模块中被调用，那么汇编产生的汇编列表是非法的，还会导致一个错误。
	例)
	callt extern void funca (void) ;
	callt extern void funcb (void) ;
	callt extern void funcc (void) ;
	
	static void (* const func [ ]) ( ) = {
	funca , funcb , funcc
	} ;
	callt void func2 (void) {
	funcc ( ) ;
	funcb ( ) ;
	funca ( ) ;
	}
	[预防方法] 将函数表和函数调用模块分开。

22	**[关于函数返回值为结构体的注意事项]**
	
	当函数返回值是结构体，在返回返回值的过程中产生了一个中断，如果在中断服务程序中又调用了这个函数，那么中断服务结束后得到的返回值是非法的。
	例）
	struct str {
	char c ;
	int i ;
	long l ;
	} st ;
	
	struct str func ( ) {
	/* 中断发生 */
	:
	}
	
	void main ( ) {
	st = func ( ) ; /* 中断发生 */
	}
	在上述服务过程中，如果 func 函数在中断服务程序中被调用，st 可能会被损坏。
	[预防方法] 无

编号	注意事项

23	**[关于共用体初始化的注意事项]**
	
	若共用体的成员有结构体、共用体或数组，初始化过程中的语法嵌套会导致 E0750 错误。
	
	例）
	struct Ss {
	int d1, d2 ;
	} ;
	union Au {
	struct Ss t1;
	} u = { { 1, 2 } } ; /* E0750 Initializer syntax */
	
	[预防方法] 对共用体初始化时不要使用嵌套。
	例）
	struct Ss {
	int d1, d2 ;
	} ;
	union Au {
	struct Ss t1;
	} u = { 1, 2 } ;

24	**[日文汉字代码分类相关的注意事项]** 为了在源程序中使用 EUC 编码，将环境变量 LANG78K 设置为 euc，或者指定-ZE 选项。
25	**[区块起始地址规格的注意事项]** 使用#pragma section 指令指定的区块起始地址，其长度总是偶数。

附录C 编译参数

本章以表格形式对程序的编译参数总结。
当进行程序开发时需要使用这些选项。
可以使用这个参数列表作用参数索引来查找。

表 C-1 编译参数

类型	描述格式	功能	和其他参数的关系	默认值
设备类型参数说明	-C 设备类型	指定目标设备的类型	完全独立	该选项的规格不能为空
目标模块创建参数说明	-O[输出文件名称]	指定目标模块文件的输出	如果同时指定了-O 和-NO 参数，最后指定的参数有效。	-O 输入文件名称.rel
	-NO	指定不输出目标模块文件		
存储器分配参数说明	-R[处理类型]（可以同时指定多个参数选项）	指定存储器的分配办法	如果同时指定了-R 和-NR，或-RD 和-NR，或-RS 和-NR,则最后指定的参数有效。	-NR
	-RD[N][M] (n = 1,2,4)	自动指定外部变量/外部静态变量分配到 saddr 区域。		
	-RS[N][M] (n = 1,2,4)	自动指定静态自动变量分配到 saddr 区域。		
	-NR	-R,-RD,-RK,-RS 参数被禁止。		
优化参数说明	-Q[优化类型]（如果需要指定多个参数选项，请依次列出）	指定调用的优化阶段，来产生不同程度的优化目标代码。	如果同时指定了-Q 和-NQ 参数，最后指定的参数有效。	-QCJLVW
	-NQ	使所有的-Q 参数失效。		
调试信息输出参数说明	-G[n](n =1,2)	指定输出源代码级的调试信息。	如果同时指定了-G 和-NG 参数，最后指定的参数有效。	-G2
	-NG	使所有的-G 参数失效。		

类型	描述格式	功能	和其他参数的关系	默认值
预处理列表文件创建参数说明	-P[输出文件名称]	指定输出预处理列表文件。	如果没有指定-P 参数，则-K 参数无效。	无（不输出文件）
	-K[处理类型]（可以同时指定多个参数选项）	指定预处理列表的处理过程。		-KFLN
预处理参数说明	-D 宏名称[=定义名称][,宏名称[=定义名称]]（可以同时指定多个参数选项）	指定的处理内容和 C 源程序中定义的内容效果相同。	完全独立	只有在 C 源程序模块文件中定义的宏才有效。
	-U 宏名称[宏名称]（可以同时指定多个参数选项）	禁用宏定义，和 C 源程序中#undef 语句效果相同。	完全独立	用-D 参数指定的宏定义有效。
	-I 目录[目录]。（可以同时指定多个参数选项）	从指定的目录读入在 C 源程序由#include 语句指定的头文件。	完全独立	1 源程序所在目录 2 环境变量 INC78K0R 指定的目录 3 C:\NECTools32\inc78K0R
汇编源程序模块文件创建参数说明	-A[输出文件名称]	指定输出汇编源程序模块文件	如果-A 和-SA 同时指定，-SA 参数无效	不输出汇编源程序模块文件
	-SA[输出文件名称]	将 C 源程序以注释方式加入汇编源程序模块文件		
错误列表文件创建参数说明	-E[输出文件名称]	指定输出错误列表文件	完全独立	不输出错误列表文件
	-SE[输出文件名称]	将 C 源程序模块文件加入错误列表文件	完全独立	
交叉引用列表文件创建说明	-X[输出文件名称]	指定输出交叉引用列表文件	完全独立	不输出交叉引用列表文件

类型	描述格式	功能	和其他参数的关系	默认值
列表格式参数说明	-LW[字符数量]	指定各种类型列表文件中每行可以容纳的字符数量。	完全独立	-LW132（对于控制台输出，此参数变为 80 字符每行。）
	-LL[行数量]	指定各种类型列表文件中每页可以容纳的行数。	完全独立	没有分页符
	-LT[字符数量]	-LT 参数指定在源程序模块文件中输出一个水平制表符（HT）代码的基本字符数量，在每个列表（制表处理）中用若干个空白（空格）代替。	完全独立	LT8
	-LF	在每个列表文件的末尾指定增加新的换页代码	完全独立	无法添加新的分页符
	-LI	以注释形式将头文件的 C 源程序加入汇编源程序模块文件。	完全独立	不能添加 C 文件格式的头文件
警告输出参数说明	-W[等级]	将警告信息输出到控制台	完全独立	-W1
执行状态显示参数说明	-V	指定是否将当前编译指定状态输出到控制台	如果同时指定了-V 和-NV 参数，最后指定的参数有效。	-NV
	-NV	使所有的-V 参数失效。		
参数文件说明	-F 文件名称	从指定的文件中输入参数或者输入文件名称。	完全独立	从命令行仅可以输入一个参数和一个输入文件名称。
临时文件创建目录说明	-T 目录	在指定驱动器的指定文件夹目录中创建临时文件	完全独立	临时文件被创建在由环境变量 TMP 指定的目录。如果未指定，临时文件被创建在当前驱动器的当前目录中。

类型	描述格式	功能	和其他参数的关系	默认值
帮助参数说明	--/-？/-H	--/-?/-H 参数显示参数的简短解释和帮助信息，比如在控制台的默认参数（仅在命令行中有效）	所有其它参数都失效	不显示任何内容
功能扩展参数说明	-Z[类型]（如果需要指定多个参数选项，请依次列出）	指定某些类型的处理过程	如果同时指定了-Z 和-NZ 参数，最后指定的参数有效。	-NZ
	-NZ	使所有的-Z 参数失效。		
设备文件搜索路径	-Y 目录	指定可能存在设备文件的搜索路径	完全独立	仅对普通搜索进行路径
存储器模式说明	-M 类型	指定编译时使用的存储器模式。	完全独立	-mm

索引

详细信息请联系：

（中国区）

网址：
http://www.cn.necel.com/
http://www.necel.com/

[北京]

日电电子（中国）有限公司
中国北京市海淀区知春路 27 号
量子芯座 7，8，9，15 层
电话：(+86)10-8235-1155
传真：(+86)10-8235-7679

[上海]

日电电子（中国）有限公司上海分公司
中国上海市浦东新区银城中路 200 号
中银大厦 2409-2412 和 2509-2510 室
电话：(+86)21-5888-5400
传真：(+86)21-5888-5230

上海恩益禧电子国际贸易有限公司
中国上海市浦东新区银城中路 200 号
中银大厦 2511-2512 室
电话：(+86)21-5888-5400
传真：(+86)21-5888-5230

[深圳]

日电电子（中国）有限公司深圳分公司
深圳市福田区益田路卓越时代广场大厦 39 楼
3901，3902，3909 室
电话：(+86)755-8282-9800
传真：(+86)755-8282-9899

[香港]

香港日电电子有限公司
香港九龙旺角太子道西 193 号新世纪广场
第 2 座 16 楼 1601-1613 室
电话： (+852)2886-9318
传真： (+852)2886-9022
2886-9044