To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

用户手册

CC78K0S Ver. 2.00

C 编译器

操作篇

目标设备

78K0S 微控制器

文档编号. U17416CA2V0UM00 (第 2 版)
发行日期 2009 年 1 月

日本印刷

[备注]

Windows, 和 Windows NT 是 Microsoft Corporation 在美国和其他国家的注册商标或商标。
PC/AT 是国际商业机器公司的一个商标。
i386 是 Intel Corporation 的一个商标。

- **本文档所刊登的内容有效期截至 2009 年 1 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。

引言

本手册的目的在于使用户完全理解 CC78K0S (78K0S 微控制器 C 编译器)的功能及操作

本手册不介绍如何操作本 C 编译器。因此，在您掌握了本手册的内容后，请阅读 **CC78K0 C 编译器 操作篇 ((U17415E)**。

[目标设备]

可以使用本 C 编译器开发 78K0S 系列微控制器的软件。

请注意，开发时需要安装与目标设备对应的设备文件。

[目标读者]

尽管本手册适用于那些已经阅读过微控制器用户手册并且具有软件开发经验的读者。但是，关于 C 编译器和 C 语言的知识并不是一定需要的，本手册中的内容假设读者熟悉软件术语。

[组织]

这本手册的结构组织如下描述。

第 1 章 概述

本章描述了在微控制器开发中 CC78K0S 的作用和角色。

第 2 章 产品概述和安装

本章描述了如何安装 CC78K0S，所提供程序的文件名，和程序的运行环境。

第 3 章 编译到连接的过程

本章使用实例程序来说明如何操作 CC78K0S 及显示从编译到连接的处理过程的实例。

第 4 章 CC78K0S 函数

本章描述了在 CC78K0S 中的优化方法和 ROMization 函数。

第 5 章 编译选项

本章描述了编译选项，具体的方法和优先级。

第 6 章 C 编译程序输出文件

本章描述了由 CC78K0S 输出的不同列表文件的输出结果。

第 7 章 C 编译器的使用方法

本章介绍了高效使用 CC78K0S 的技巧。

概括介绍该 CC78K0S 的一般功能、性能指标及特色。

第 8 章 初始例行程序

CC78K0S 提供了初始例行程序作为实例。这章描述了初始例行程序的使用和提供了关于如何改进它们的建议。

第 9 章 错误信息

本章描述了由 CC78K0S 输出的错误信息。

附录

附录提供了实例程序，使用时的注意事项，关于 CC78K0S 的限制，以及索引。

[如何阅读这本手册]

首先，需要了解实际如何操作 CC78K0S 的用户，请先阅读**第 3 章从编译到连接的过程**。

有 C 编译程序知识的用户或者已经阅读了语言手册的用户可以跳过**第 1 章概述**。

[相关文档]

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）

文档名称		文档编号.
CC78K0S Ver. 2.00 C编译器	操作篇	本手册
	语言篇	U17415E
RA78K0S Ver. 2.00 汇编包	操作篇	U17391E
	语言篇	U17390E
	结构化汇编语言	U17389E
SM+ 系统仿真器	操作篇	U18601E
	用户开放接口	U18212E
SM78K 系列 Ver. 2.52系统仿真器	操作篇	U16768E
PM+ Ver. 6.30 项目管理器		U18416E
ID78K0S-NS Ver. 2.52集成调试器	操作篇	U16584E
ID78K0S-QB Ver. 3.00集成调试器	操作篇	U17287E

[规则]

本手册中使用了以下符号及缩写。

RTOS:	78K0 微控制器 RX78K0 实时操作系统
…:	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)。
粗体:	如粗体字符所示(字符串)。
_ :	在重要的位置或实例中的下划线为输入字符的序列。
Δ :	至少一个空间。
⋮ :	在程序中代表省略。
( ) :	如圆括号中之间的字符所示(字符串)。
/ :	定界符
\:	反斜杠

[文件名规则]

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

```
[驱动器名]  [\]  [[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器名(A: 到 Z:)。

<2>指定根目录名。

<3> 指定子目录名。

指定操作系统允许长度的字符串。

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(-)不能当作路径的第一个字符。

<4> 主文件名

指定操作系统允许长度的字符串。

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(-)不能当作路径的第一个字符。

<5> 文件类型

指定操作系统允许长度的字符串。

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。

```
举例:  C:\nectools32\smp78k0S\CC78k0S\prime.C
```

备注 **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)

[备注]

目录

插图列表

插图编码 标题 页码

表格列表

表格编号 标题 页码

第 1 章　概要

CC78K0S C 编译器能够把符合 ANSI-C [注]规范或符合 78K0S 系列规范的 C 语言源程序转变成 78K0S 系列能够识别的机器语言。

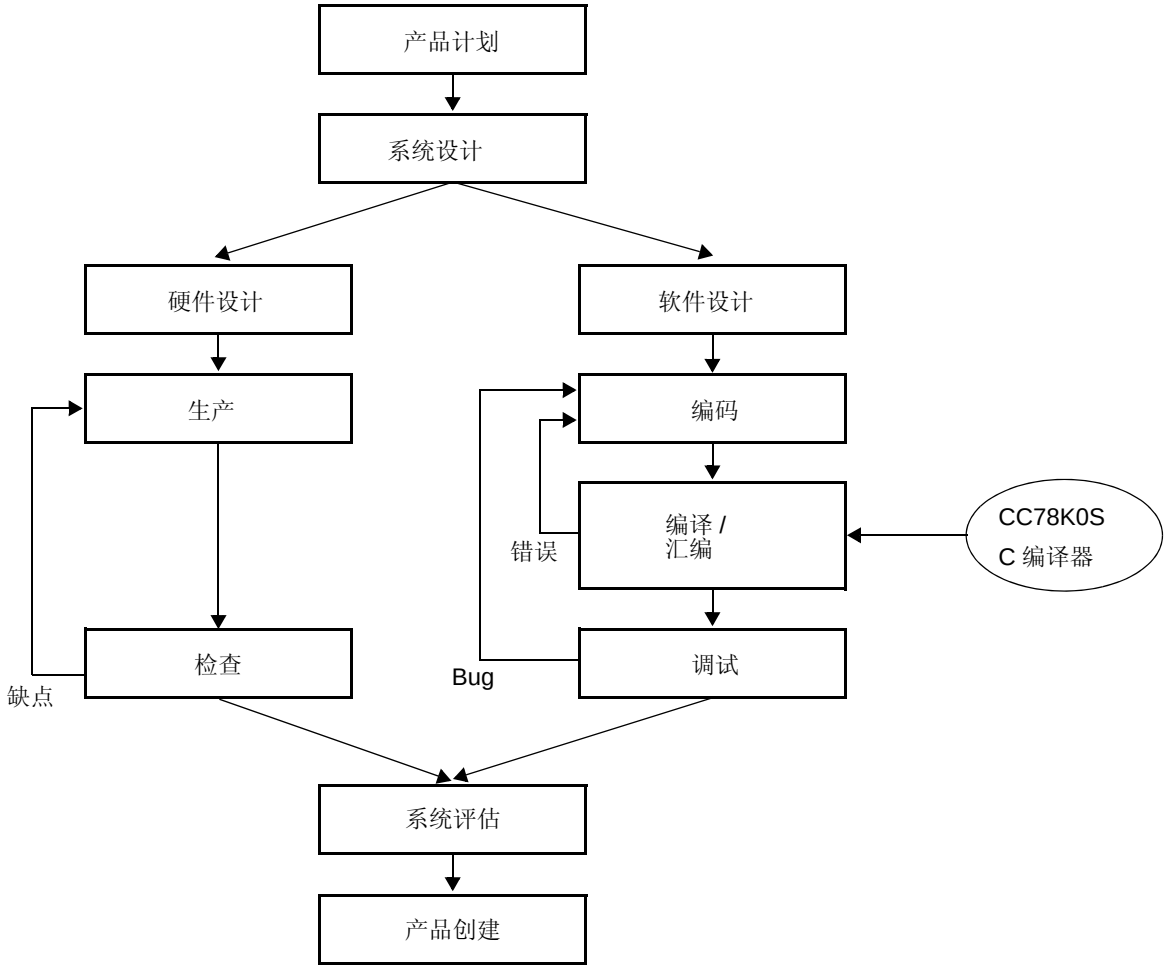
CC78K0S 可通过 Windows 提供的带有汇编程序包的 PM+ 启动，该汇编程序包用于 78K0S 系列。 当未使用 PM+ 时，编译器从 DOS 启动。

注　ANSI-C 是美国国家标准局所制定的 C 语言标准。

1.1　CC78K0S 的作用

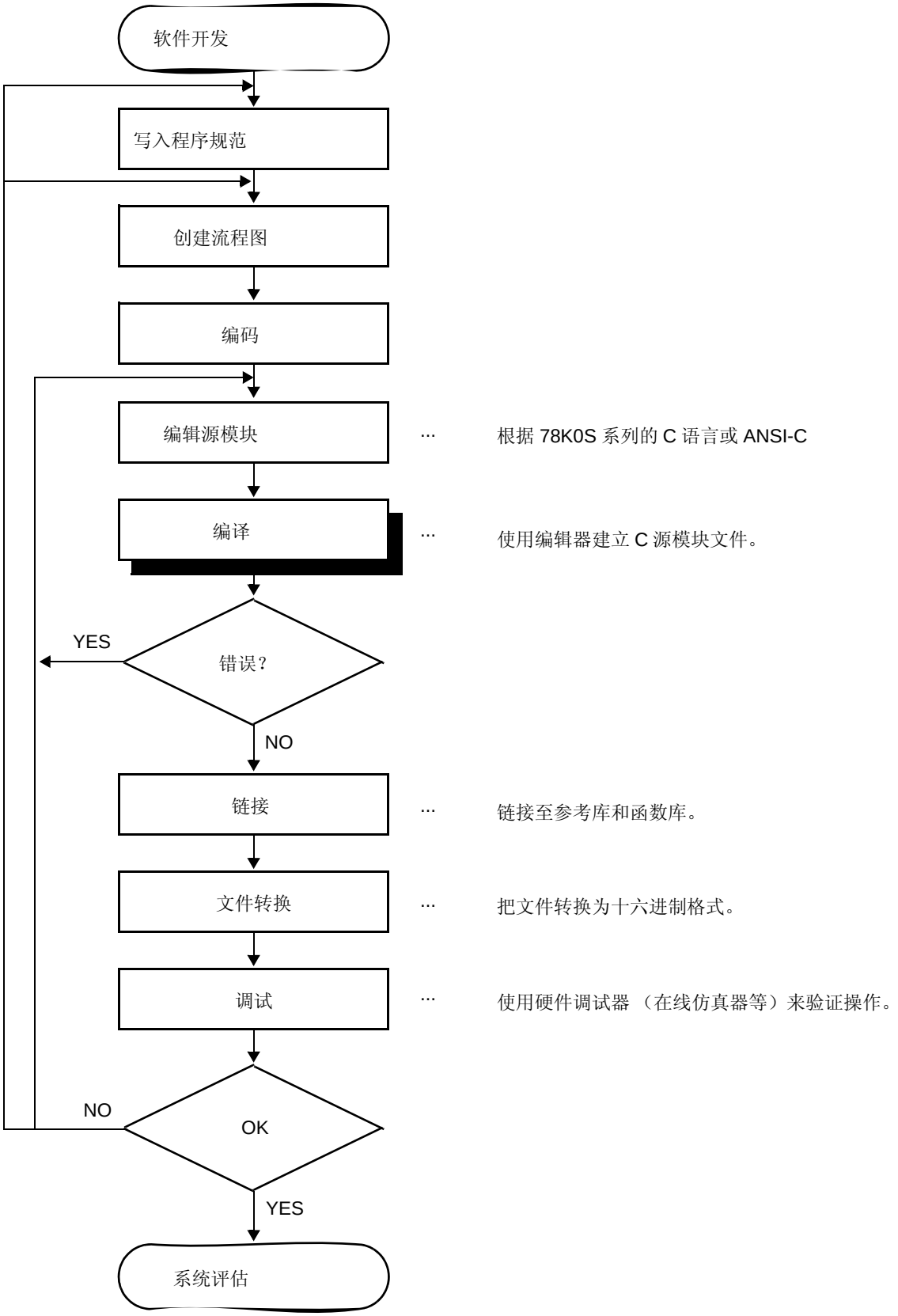
CC78K0S 在产品开发中的位置如下图所示。

图 1-1 开发过程



软件开发过程如下图所示。

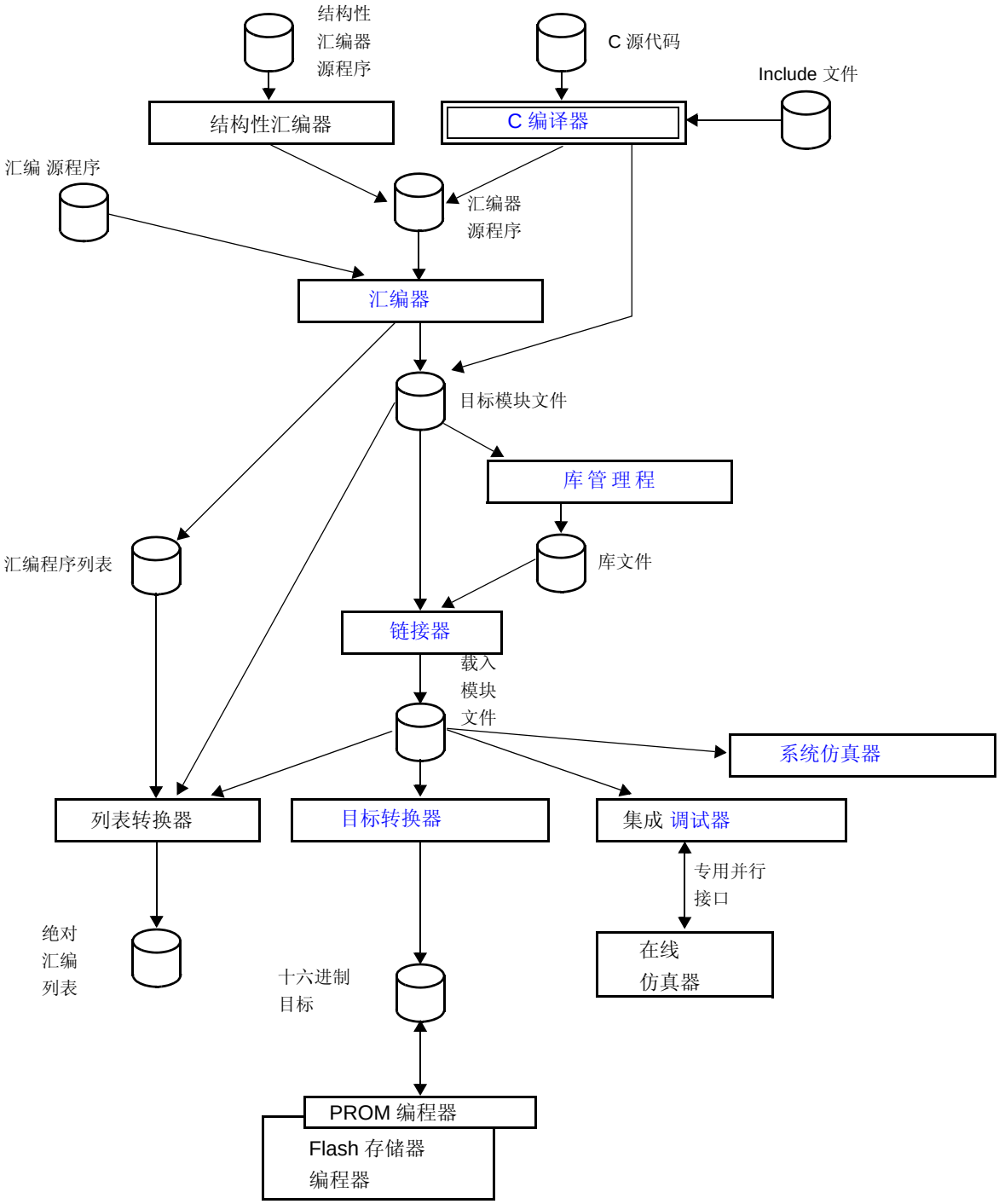
图 1-2 软件开发过程



1.2 使用 CC78K0S 的开发过程 78K0S

使用 CC78K0S 的开发过程如下图所示。

图 1-3 使用 CC78K0S 的程序开发过程



1.2.1 使用编辑器创建源程序模块

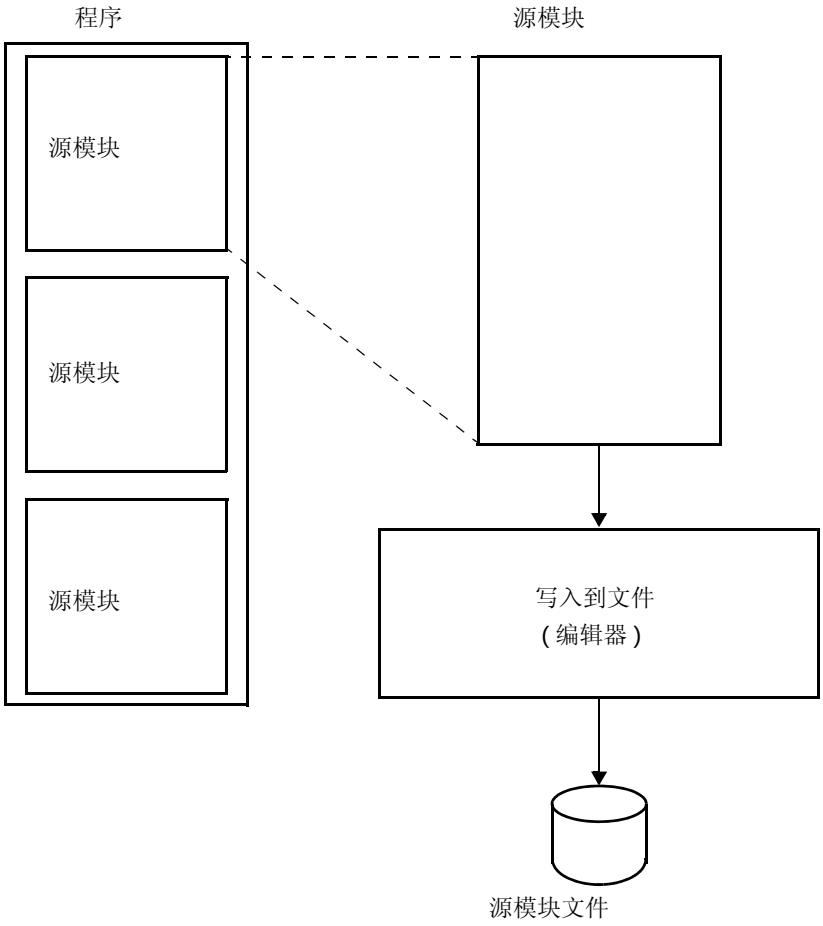
一个程序被划分为若干个功能模块。

其中一个模块是代码编写单元，也是编译器得一个输入单元。 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 源程序模块文件成为 CC78K0S 输入文件。

图 1-4 创建源模块文件

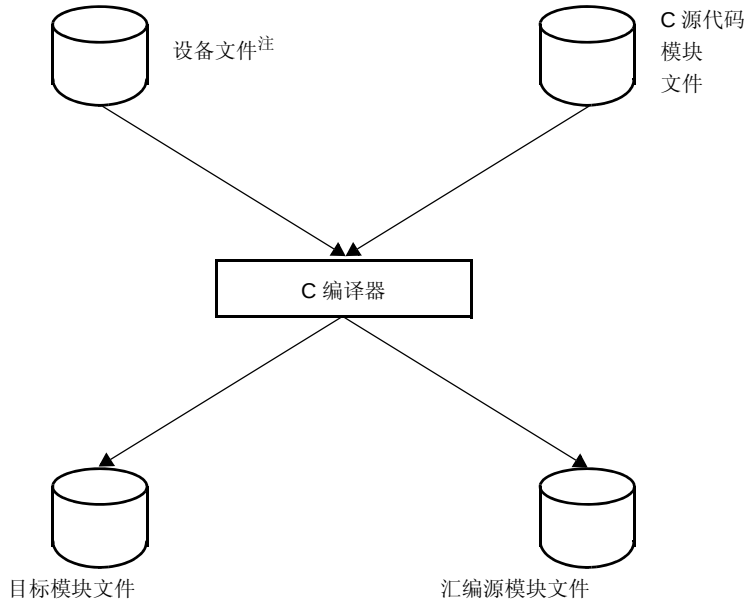


1.2.2 C 编译器

C 编译器把 C 语言转换为机器语言，把 C 源模块文件作为输入。 如果在 C 源模块文件中发现描述错误时， C 编译器将输出编译错误。

如果没有编译错误产生，将输出目标模块文件。 此外，也可输出汇编源模块文件，这样程序可在汇编语言等级上进行修改和检查。 为了输出汇编源模块文件，可在编译时设定 -a 选项或 -sa 选项 （关于选项的详细信息，请参考“第 5 章 编译器选项”。).

图 1-5 C 编译器的功能



注 进入以下网站，通过从在线传送服务 (ODS) 上下载获得设备文件。

http://www.necel.com/micro/ods/eng/tool/DeviceFile/list.html

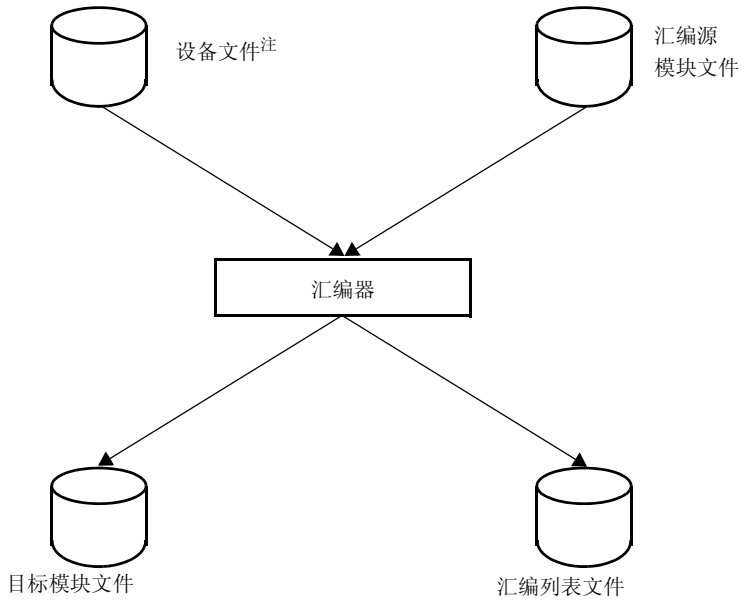
1.2.3 汇编器

汇编工作是通过使用 RA78K0S 汇编程序包 （单独销售）中的汇编程序来执行的。

汇编程序把汇编源模块文件输入并把源模块文件从汇编语言转换为机器语言。 如果在源模块中发现描述错误，则输出汇编错误。

如果未发生汇编错误，包含机器语言信息和位置信息的目标模块文件说明输出需定位的每个机器语言代码的存储器地址。 除此以外，在汇编过程中的信息会以汇编表文件的形式输出。

图 1-6 汇编器功能



注　进入以下网站，通过从在线传送服务 (ODS) 上下载获得设备文件。

http://www.necel.com/micro/ods/eng/tool/DeviceFile/list.html

1.2.4 链接器

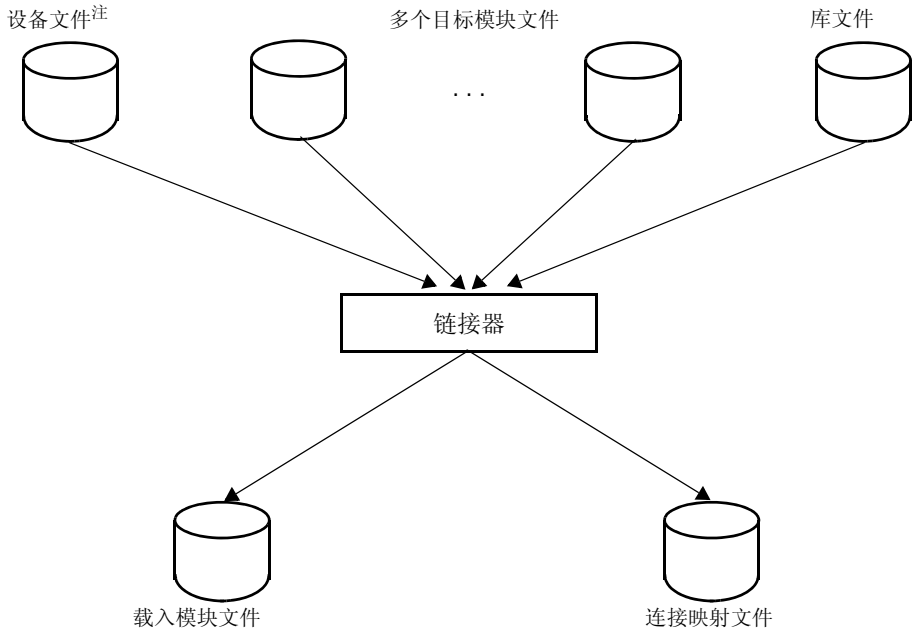
连接操作是通过使用 RA78K0S 汇编程序包 （分开销售）中的连接器来执行的。

链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链接起来 (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，链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。

链接器将链接信息输出到链接映射文件 （link map）。

图 1-7 链接器功能



注　　进入以下网站，通过从在线传送服务 (ODS) 上下载获得设备文件。

http://www.necel.com/micro/ods/eng/tool/DeviceFile/list.html

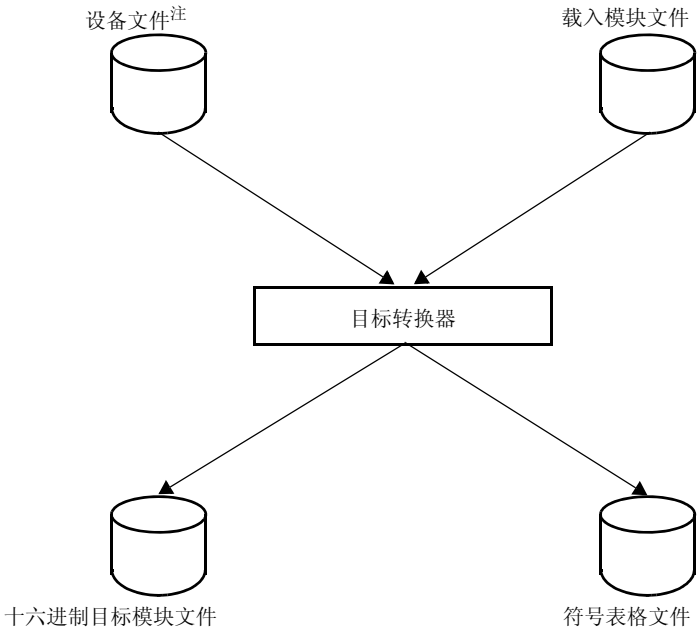
1.2.5 目标转换器

目标转换操作是通过 RA78K0S 汇编程序包 (分开销售) 中的转换器来执行的。

目标转换器输入由链接器输出的负载模块文件并转换器文件格式。 结果以 intel 标准的十六进制目标模块文件作为输出。

符号信息输出符号表文件中。

图 1-8 目标转换器功能



注 进入以下网站，通过从在线传送服务 (ODS) 上下载获得设备文件。

http://www.necel.com/micro/ods/eng/tool/DeviceFile/list.html

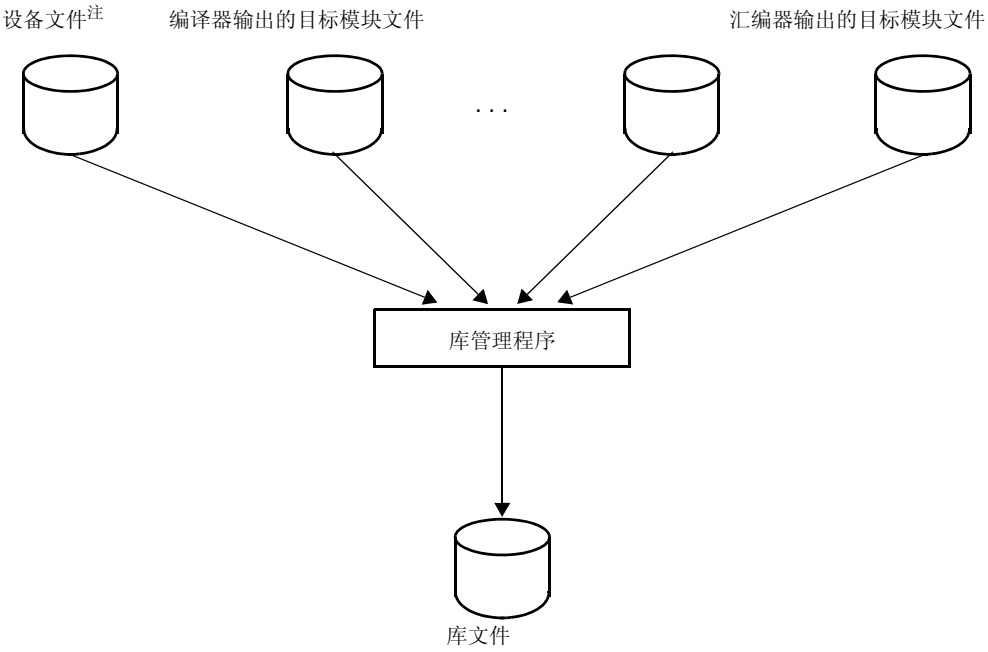
1.2.6 库管理程序

为了方便起见，拥有通用接口并被明确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。

链接器可以从库文件中提取出需要的模块并将它们链接起来。 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 RA78K0S 汇编程序包 (分开销售) 中的库管理程序来执行。

图 1-9 管理员功能



注　进入以下网站，通过从在线传送服务 (ODS) 上下载获得设备文件。

http://www.necel.com/micro/ods/eng/tool/DeviceFile/list.html

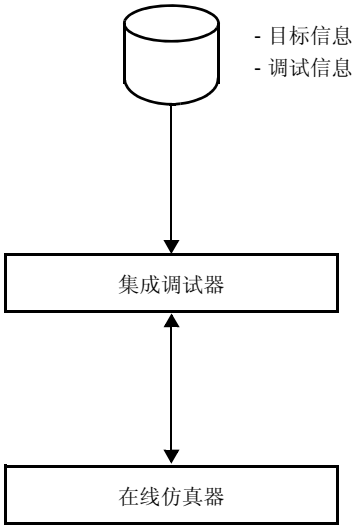
1.2.7 调试器

将链接器输出的装载模块文件通过 ID78K0S-NS (78K0S 系列集成调试器) 下载到 IE(内部电路仿真器) 中，就可以使用图形用户接口对源程序进行调试。

为了实现调试，当编译目标源程序时 (-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加入到目标模块中。 关于编译器选项的信息，请参阅 " 第 5 章 编译器选项 "。

调试器和 IE 分别包装 （分别销售）。

图 1-10 调试器功能

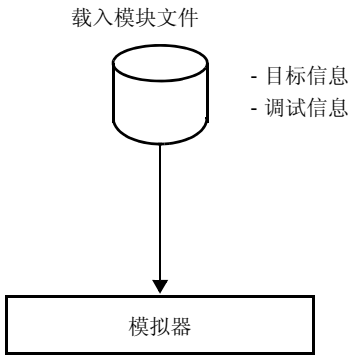


1.2.8 系统仿真器

使用 SM78K0S (78K0S 系列系统仿真器) 将链接器输出的装载模块文件下载，就可以使用图形用户接口对源程序进行调试。

SM78K0S 和 ID78K0S-NS 有同样的操作界面， SM78K0S 在主机上执行仿真的软件。 除了在 SM78K0S 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进行。

图 1-11 仿真器功能

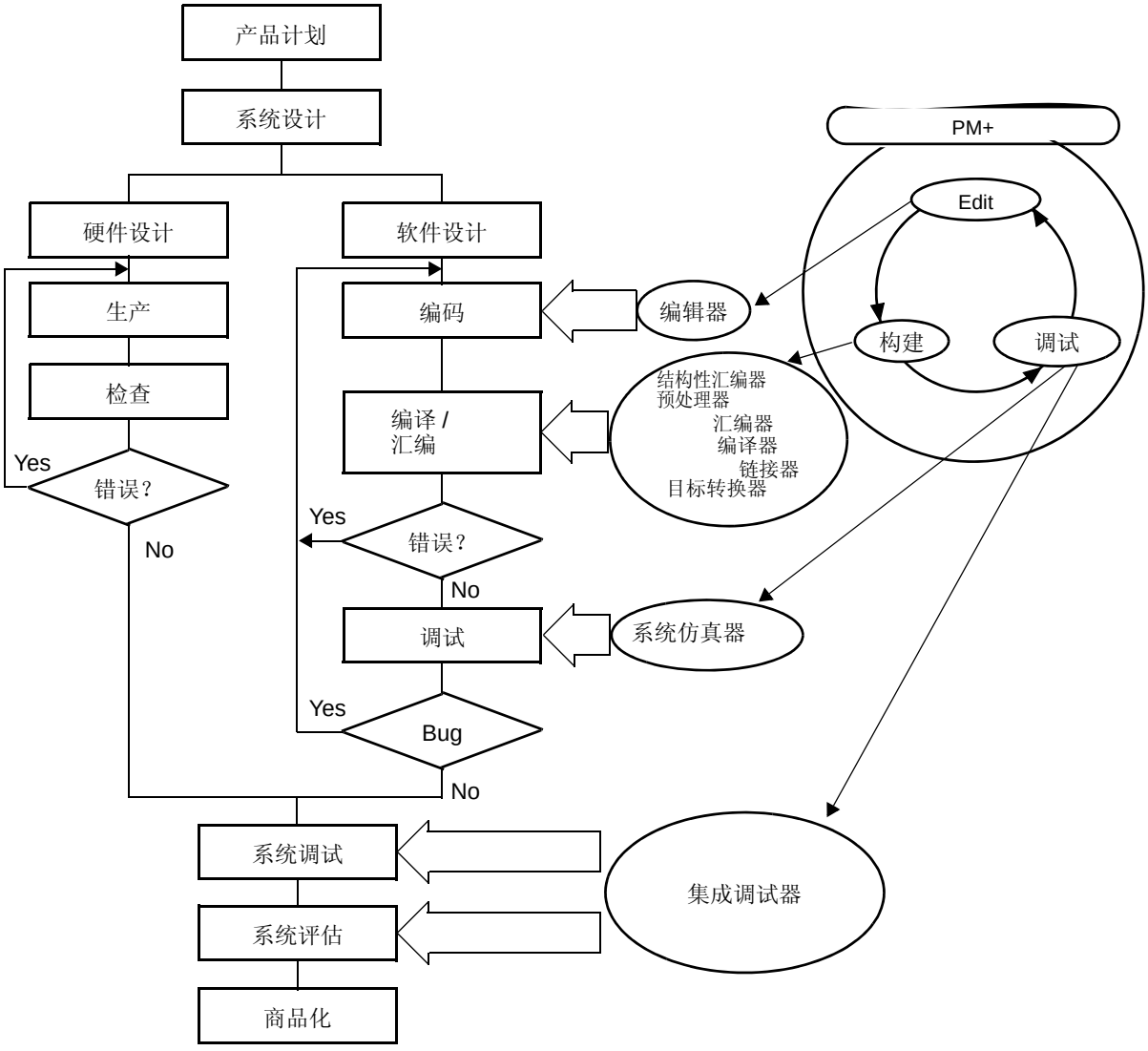


1.2.9 PM+

PM+ 是把 DLL 文件添加到 CC78K0S 并可在 Windows 下启动 CC78K0S 的软件。 编辑源程序，自动创建 MAKEFILE，从编译到连接都可以在 PM+ 的界面中执行。 因此，可以使用 PM+ 的图形用户界面来进行编辑至调试的开发过程。

PM+ 包含在 RA78K0S 汇编程序包内 (分别销售)。 运行 RA78K0S 汇编程序包可以安装程序并进行设置。 如果需要 PM+ 中启动 CC78K0S，请在安装编译器之前先安装 RA78K0S 汇编程序包。

图 1-12 PM+ 功能



备注 Build 阶段会分析并执行 make file 来建立可执行文件。 在 makefile 中描述的关联关系基本上去掉了那些无法使用的汇编、编译和连接功能，同时可以创建高效的可执行文件。

第 2 章　产品概述和安装

本章介绍了将 CC78K0S 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　编译器的供应媒介和记录格式

主机	操作系统	提供媒介
IBM PC/AT™ 兼容机	Windows (98/Me/2000/XP/NT 4.0) 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 DOS 提示符 (Windows 98/Me) 下启动，或命令提示符 (Windows 2000/XP/NT 4.0) 下启动 C 编译器。

2.2 安装

将供应媒体中的 CC78K0S 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 CC78K0S 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。

备注　　如果安装程序没有自动启动，请执行 CC78K0S 文件夹中的 SETUP.EXE 文件。

(3) 确认文件

使用 Windows 资源管理器等，检查 CC78K0S 供应媒体中的文件是否已经安装到主机上。

关于各文件夹的细节，请参阅 "2.4 文件夹配置”。

2.3 设备文件的安装

进入以下网站，通过从在线传送服务 (ODS) 上下载获得设备文件。

http://www.necel.com/micro/ods/eng/tool/DeviceFile/list.html

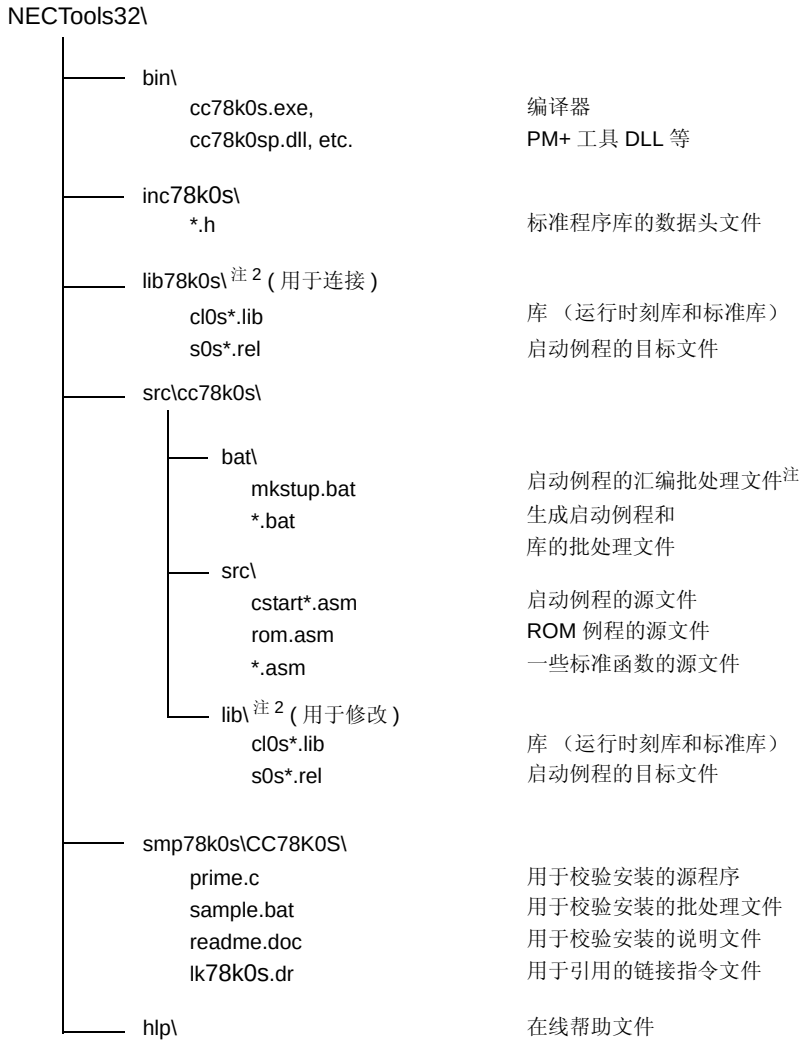
使用安装文件来进行设备文件的安装。 设备安装文件和 CC78K0S 同时安装。

2.4 文件夹配置

Windows 系统安装过程中的标准文件夹是 "NECTools32”。 安装文件夹中的文档结构如下。 注意在安装过程中可以改变驱动器和安装文件夹。 当使用 PM+ 执行编译操作时，相关的工具 (CC78K0S， RA78K0S) 也被安装到这个驱动器和文件夹。

本手册中假定的标准文件夹就是 “NECTools32”，这是默认的程序名称，也是安装的默认路径。

图 2-1 文件夹配置



注 1。 这个批处理文件不能在 PM+ 中使用。 批处理文件只能在 DOS 提示符 (Window 98/Me) 或命令提示符 (Windows 2000/XP/NT 4.0) 中运行。

注 2。 如要修改启动例程，则要修改 src\cc78k0s\lib 文件夹下的源文件 . 由批处理文件得到的汇编文件存放在 src\cc78k0s\lib，复制此文件到 lib78k0s 文件夹执行链接。

2.5 卸载过程

主机中文件的卸载过程说明如下。

(1) 启动 Windows

为主机和外部设备供电并启动 Windows。

(2) 打开 [Control Panel] 窗口

按 [Start] 按钮，选择 [Settings]-[Control Panel] 打开 [Control Panel] 窗口。

(3) 打开 [Add/Remove Programs] 窗口

在 [Control Panel] 窗口中双击 [Add/Remove Programs] 图标，打开 [Add/Remove Programs] 窗口。

备注　在 Windows XP 下 , [Add/Remove Programs] 显示为 [Add or Remove Programs]。

(4) 删除 CC78K0S

从 [Add/Remove Programs] 窗口的 [Install/Uninstall] 标签显示的软件安装列表选择 "NEC CC78K0S 78K/0S C Compiler Vx.xx"，单击 [Add/Remove] 按钮。

当 [System Setting Change] 窗口打开后，单击 [Yes] 按钮。

(5) 确认文件

使用 Windows 资源管理器等，确认安装到主机的文件已经被删除。 关于各文件夹的细节，请参阅 "2.4 文件夹配置”。

2.6 环境设置

2.6.1 主机

CC78K0S 可以处理 32 位，并且在配置 i386TMCPU 或更高版本的模型上运行。

由于使用 DOS 扩展器完成 32 位的处理，所以也适合在以下操作系统上运行。

- Windows98/Me/2000/XP/NT4.0
- Windows98/Me 下的 DOS 提示符
- Windows2000/XP/NT4.0 下的 DOS 提示符

2.6.2 环境变量

为 DOS 提示符 （Windows 98/Me）或命令提示符 （Windows 2000/XP/NT 4.0）操作设置以下环境变量。

表 2-2 环境变量

环境变量	说明
PATH	指定编译器所在的文件夹。
TMP	指定创建的临时文件的文件夹。
LANG78K	在原文件中指定汉字代码 （2 字节代码）。 sjis : Shift JIS (默认) euc : EUC none : 非双字节编码
INC78K0S	指定编译器的标准头所在的文件夹。
LIB78K0S	指定编译器的程序库所在的文件夹。

示例说明

```
PATH = %PATH% ; C:\NECTools32\bin
set     TMP = C:\
set     LANG78K = sjis
```

2.6.3 文件结构

下面的表格列出了每个文件夹的内容。 安装时就已经决定了文件夹结构和文件结构。

表 2-3 文件结构 (* = 字母数字符号)

文件夹名称	文件名	说明
bin\	cc78k0s.exe	编译器
	cc78k0s.msg	信息文件
	*.hlp	帮助文件
	*.dll	DLL 文件
inc78k0s\	*.h 注 1	标准程序库的数据头文件
src\cc78k0s\bat\ 注 2	mkstup.bat	启动例程的汇编批处理文件
	reprom.bat	更新 rom.asm
	*.bat 注 3	更新标准函数的批处理文件 （部分的）
src\cc78k0s\src	cstart*.asm 注 4	启动例程的源文件
	rom.asm	ROM 化例程的源文件
	*.asm 注 5	标准函数的源文件 （部分的）
hlp	*.chm	在线帮助文件

注 1. 参考 CC78K0S C 编译器 语言篇 用户手册。

注 2 . 文件夹的批处理文件不能在 PM+ 中使用。 只有在源文件必须被修改时才使用这些文件。

注 3. 参考表 8-1 内容。

注 4. * = N (N : 未使用标准程序库)

注 5. 参考表 8-2 内容。

2.6.4 库文件

这些文件由标准库、运行时刻库和启动例程组成。

表 2-4 列出了文件夹内容。

表 2-4 库文件

文件夹名称	文件名	文件作用
lib78k0s\	cl0s.lib cl0sr.lib cl0ss.lib cl0sf.lib cl0sx.lib 注 3 cl0sxr.lib 注 3 cl0sxs.lib 注 3	库 （运行时刻库和标准库）注 1
	s0s.rel s0sl.rel s0ss.rel s0ssl.rel	启动例程的目标文件注 2

注 1. 命名程序库的规则如下。

```
lib78k0s\cl0s< mul >< float >< pascal >< model >.lib
```

< mul >

None 未使用乘法器

x 使用乘法器

< float >

None 标准库和运行时刻库 (未使用浮点指针库)

f 浮点指针库

< pascal >

None 使用普通函数接口

r 使用 pascal 函数接口 (指定编译选项 -zr)

< model >

None

s 静态模式

注 2. 命名启动例程的规则如下。

```
lib78k0s\s0s< model >< lib >.rel
```

< model >

None

s 静态模式

< lib >

None 未使用标准库函数

l 使用标准库函数

注 3. CC78K0S 库与以下的乘法器设备兼容。

然而，如果当运算进行时发生中断，部分运算结果因为被中断而失效，因此未被错用。

关于库功能和中断失效时间，请参考 CC78K0SC 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
16- 位 乘法结果存储寄存器 0	MUL0	FF10H/FF12H/ FF14H	16 位
乘法数据寄存器 A0	MRA0	FFD0H	8 位
乘法数据寄存器 B0	MRB0	FFD1H	8 位
乘法器控制寄存器 0	MULC0	FFD2H	8 位

当采用器件的 16 位乘法结果存储寄存器 （MUL0）不是位于地址 FF10H 时，使用批处理文件 repcmul.bat 来更新库函数。 （关于批处理文件的详细信息，请参考 CC78K0S C 编译器语言的用户手册）。

在 CC78K0S 中，只包含支持乘法器的运行时刻库 @@cumul （无符号 char 型的乘法）和 @@csmul （带符号 char 型的乘法）。

不包含用于 int 和 long 型 （带乘法器）参数乘法运算的运行时刻库，因为虽然这种类型的乘法可减少执行时钟数，但是会增加代码的长度。

使用批处理文件 repimul.bat 和 replmul.bat 来更新库函数，必要的话，添加支持乘法器的运行时刻库，参考下面列出的代码大小和执行时钟数。 （关于批处理文件的详细信息，请参考 CC78K0S C 编译器语言的用户手册）。

[@@csmul/@@cumul]

Model	不支持乘法器的库	支持乘法器的库
Normal	28 bytes, 236 ~ 380 clocks	21 bytes, 60 clocks
静态	28 bytes, 236 ~ 380 clocks	21 bytes, 60 clocks

[@@ismul/@@iumul]

Model	不支持乘法器的库	支持乘法器的库
Normal	47 bytes, 874 ~ 1290 clocks	86 bytes, 236 clocks
静态	51 bytes, 892 ~ 1308 clocks	85 bytes, 236 clocks

[@@lsmul/@@lumul]

Model	不支持乘法器的库	支持乘法器的库
Normal	96 bytes, 3034 ~ 4634 clocks	320 bytes, 818 clocks

第 3 章　由编译至链接过程

本章利用 CC78K0S 和 RA78K0S 汇编程序包的来描述编译到链接这个过程。

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 示例程序”，以获取采用程序的信息）。

描述如何在 PM+ 上执行，以及如何从命令行执行 (关于安装的信息，请参考 "2.2 安装”)。

3.1　PM+

本节描述了在 PM+ 中 CC78K0S 的用户接口， PM+ 包括在 RA78K0S 汇编程序安装包中。 如果 CC78K0S 从 PM+ 启动，将引用 CC78K0S 中包含的 cc78k0sp.dll。

3.1.1　cc78k0sp.dll 的位置 (tools DLL)

tools DLL 文件，如 cc78k0sp.dll 文件，需要用来运行由 PM+ 启动的 Windows 版的 78K0S 系列 C 编译器 (CC78K0S)。

3.1.2　执行的环境

该环境适用于 PM+

3.1.3 CC78K0S 选项设置菜单

(1) 选项菜单条目

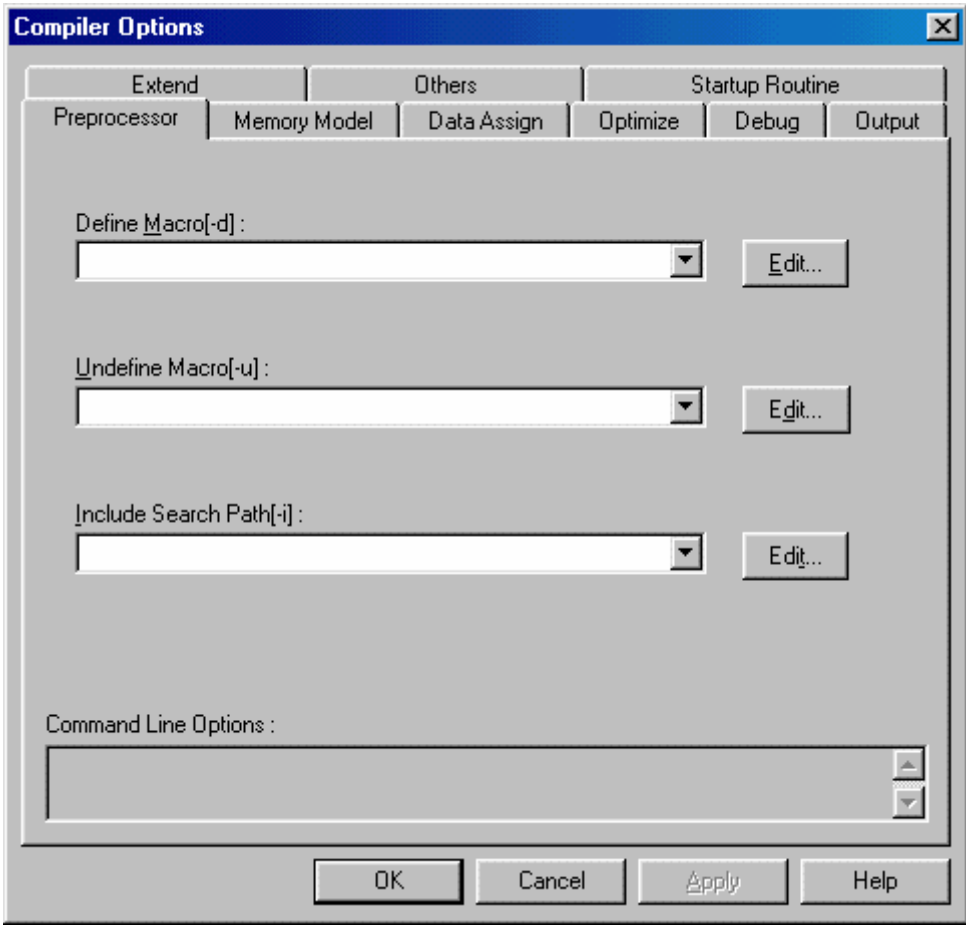
通过 DLL 工具文件把 [Compiler Options] 添加到 PM+ 中的 [Tools] 菜单中，该 DLL 工具文件包含在 CC78K0S C 编译器包中。

(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



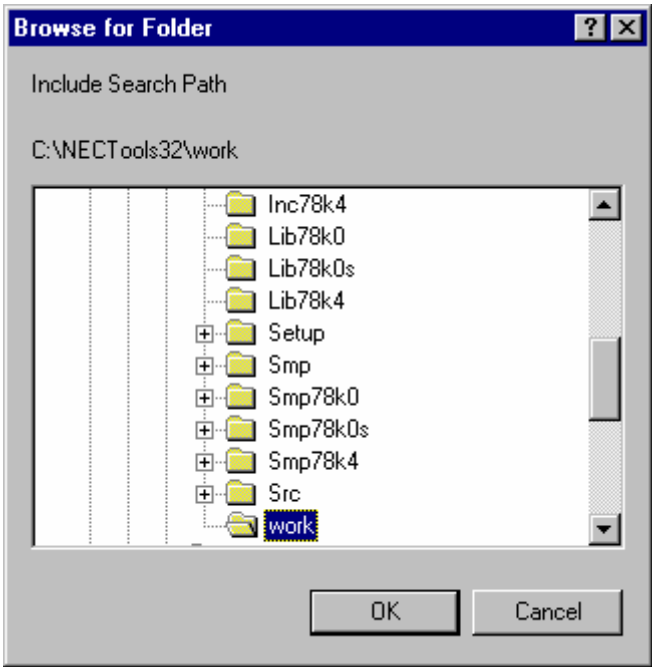
(3) Browse for Folder 对话框

在 < Compiler Options > 对话框中，当单击 [Browse] ° 设置路径时，会出现下列对话框。 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



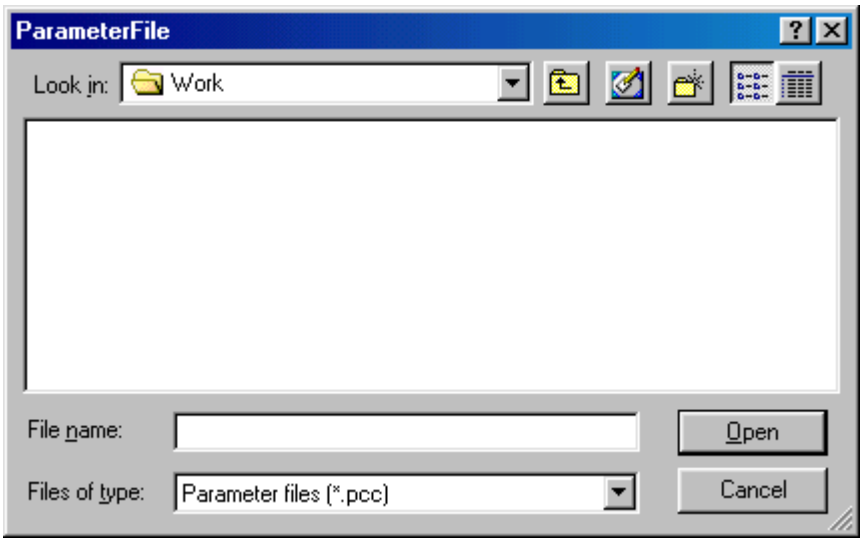
在指定参数文件时单击 [Browse] 按钮，会出现下列对话框。 对话框如下。

此对话框如下所示。

当前文件夹： 项目文件文件夹

文件类型： 参数文件 (*.pcc)

图 3-3 < ParameterFile > 对话框



(4) < Edit Option > 对话框

在 < Edit Option > 对话框中，以列表形式编辑各项。

< Edit Option > 对话框描述如下。

图 3-4 < Edit Option > 对话框

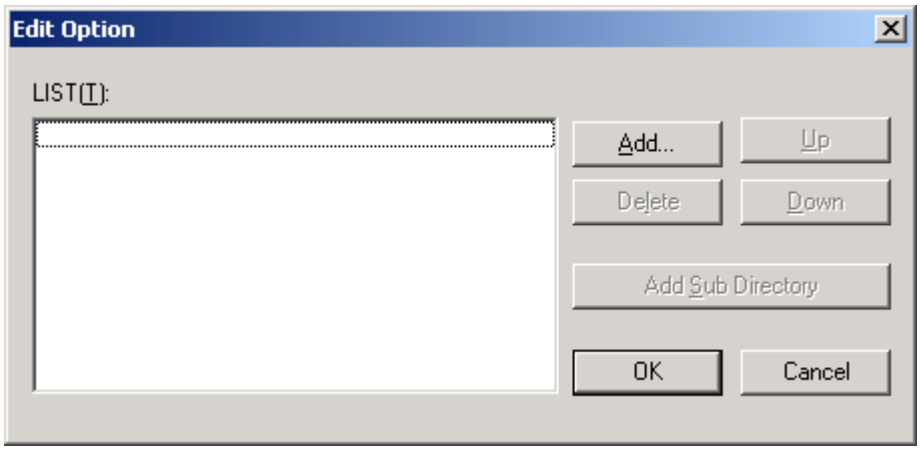
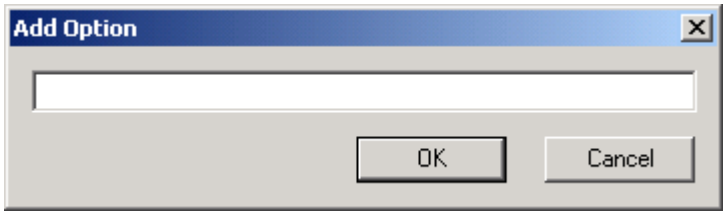


图 3-5 < Add Option > 对话框



- [Add] 按钮

  添加一个列表项。

  如果添加项为文件或文件夹时，会打开相应的 < Browse for Folder > 对话框。

  其他情况下，会打开 < Add Option > 对话框。 设定添加到该对话框中项目的详细信息。

- [Delete] 按钮

  删除已选的列表项。

- [Up] 按钮

  向上移动选中的列表项。

- [Down] 按钮

  向下移动选中的列表项。

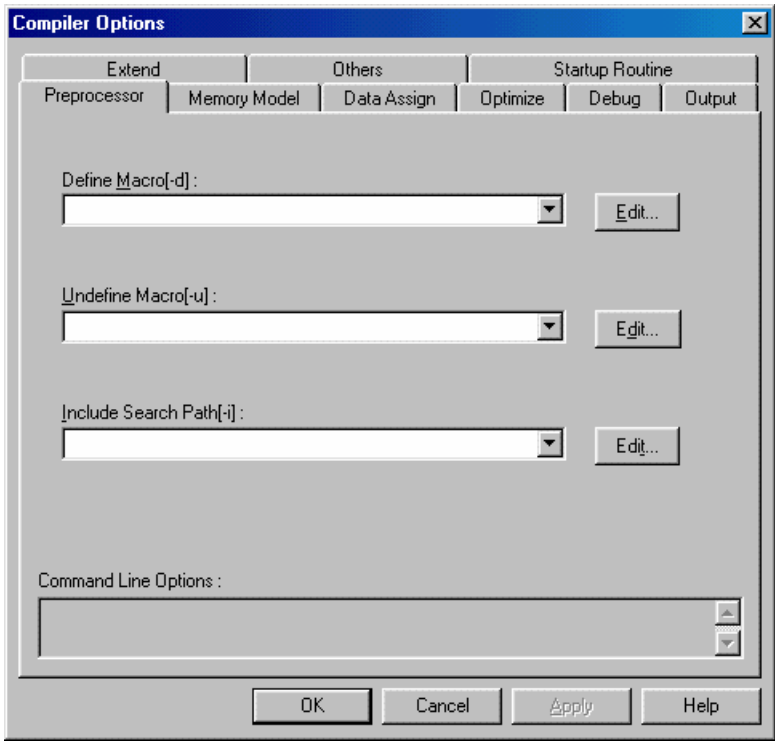
- [Add Sub Directory] 按钮

  当设定的项在 << Others >> 标签下作为 Include Search Path[-i](I) 时，可在已选的列表项中添加子目录。

3.1.4 < Compiler Options > 对话框的具体描述

下面讲解 < 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。 单击对话框上面相应的标签页可以显示每一个设置界面。

  << Preprocessor >> 标签 (默认值)

  << Memory Model >> 标签

  << Data Assign >> 标签

  << Optimize >> 标签

  << Debug >> 标签

  << Output >> 标签

  << Extend >> 标签

  << Others >> 标签

  << Startup Routine >> 标签

- 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 CC78K0S 的默认选项为 " 已指定 ” 状态 （如：复选框已选中），该区域默认为无任何显示。

  选项不在选项属性显示区域显示时，可以通过滚动 [ScrollBar] 来选择。

- [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 Project Window 中选择 [Special Compiler Options] 的话，将对该文件设定选项。如果在 [Tools] 菜单中选择 [Compiler Options] 的话，将对所有源文件设定选项。

- [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。

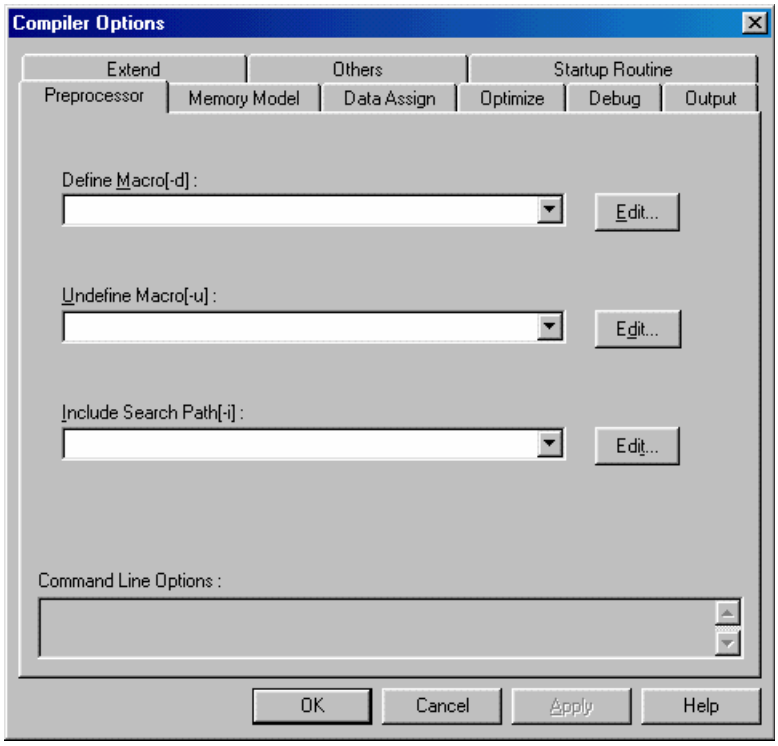
  应用对话框中编辑的内容， < Compiler Options > 对话框继续显示。

- [Help] 按钮

  打开关于本对话框的帮助文件。

(1) << Preprocessor >> 标签

图 3-7 < 编译器选项 > 对话窗 (当 << Memory Model >> 选择标签)



- 定义宏 [-d]

  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，可以一次定义多个名称，多个宏名称可以用 "," 来区分。

  用 [Edit] 按钮就能指定。

- 未定义宏 [-u]

  用 -U 选项指定的宏名字输入进组合框中。

  对于宏名称来说，一旦用 ",” 来限制，多个宏定义无效。

  用 [Edit] 按钮就能指定。

- 包含查找路径 [-i]

  由 [-i] 选项指定的包含文件内容的文件夹输入进这个组合框中。

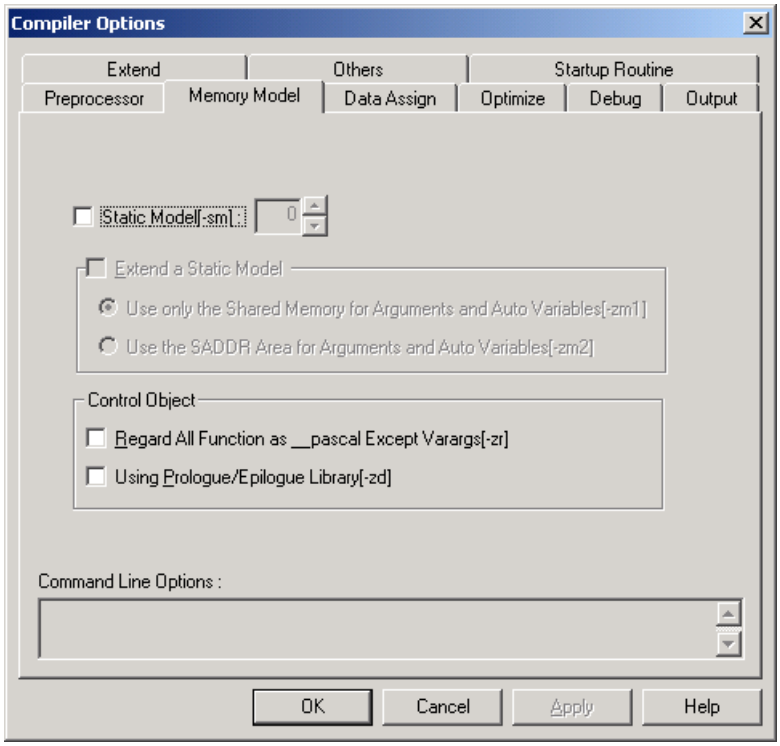
  一旦用 ",” 限制，就能指定多目录。

  用 [Edit] 按钮就能指定。

  不能指定不存在的路径。

(2) << Memory Model >> 标签

图 3-8 < 编译器 选项 > 对话窗 (当 << Memory Model >> 选择标签)



- 静态模式 [-sm]

  选中复选框用于静态模式，并且指定公共区域的字节数。

- 扩展静态模式

  如果指定了 -sm 选项，并希望扩展静态模式，请选中这个选择框。

  点击适当的单选按钮，即可以选择参数和自动变量的存储区域。

  尽管复选框已经是未选中状态，单选按钮的信息仍然可以保存。

- 控制对象

  把所有函数都当作 _ _pascal，除了 Varargs[-zr]

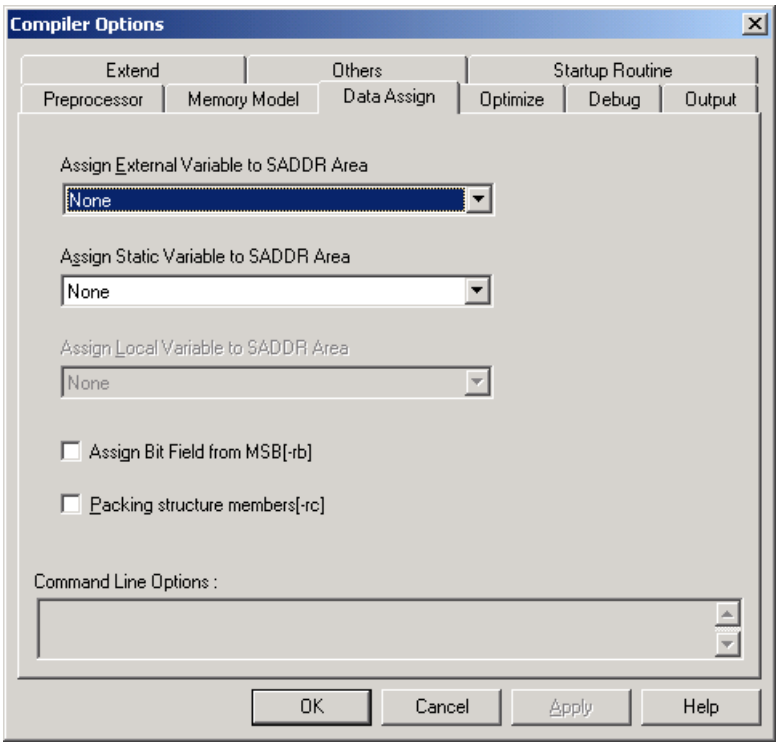
  选中该复选框来生效 -zr 选项。

  使用序言 / 尾声库 [-zd]

  选中该复选框来生效 -zd 选项。

(3) << Data Assign >> 标签

图 3-9 < 编译器选项 > 对话窗 (当 << Data Assign >> 标签选中时)



- 将外部变量分配到 SADDR 区域

  在下拉列表窗中，外部变量类型可以选择分配到 saddr 区域。

- 分配静态变量到 SADDR 区域

  在下拉列表窗中，静态变量类型可以选择分配到 saddr 区域。

- 局部变量分配到 SADDR 区域

  在下拉列表窗中，自动变量类型可以选择分配到 saddr 区域。

- 从 MSB[-rb] 开始分配位域

  选中选择框来生效 -RB 选项。

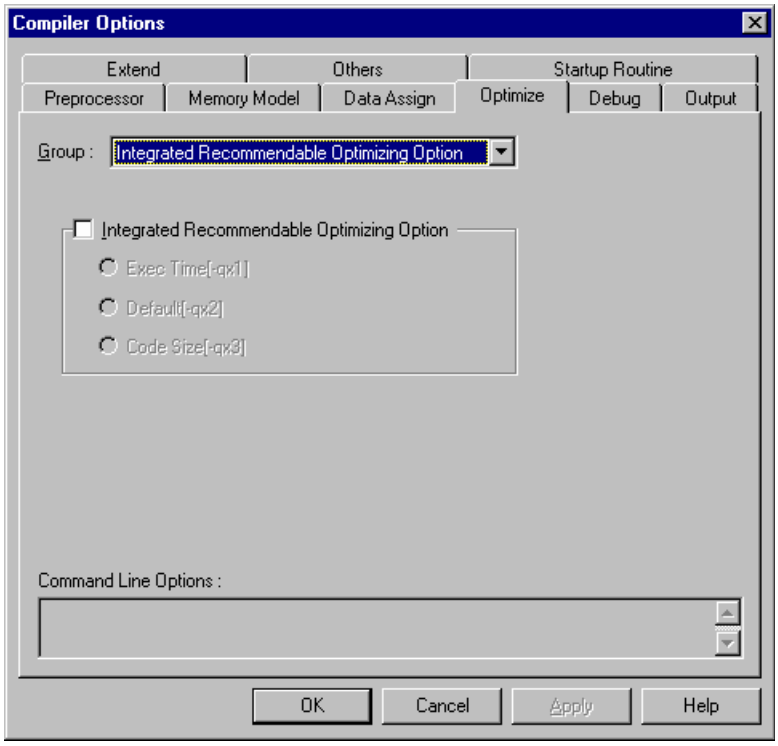
- 结构元素打包 [-rc]

  选中选择框来生效 -RC 选项。

(4) << Optimize >> 标签

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

图 3-10 < 编译选项 > 对话框 (当选择 << 综合推荐优化选项 >>)



- 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。

  共有三种设定："Exec Time [-qx1]”、 "Default [-qx2]”和 "Code Size [-qx3]”。 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]

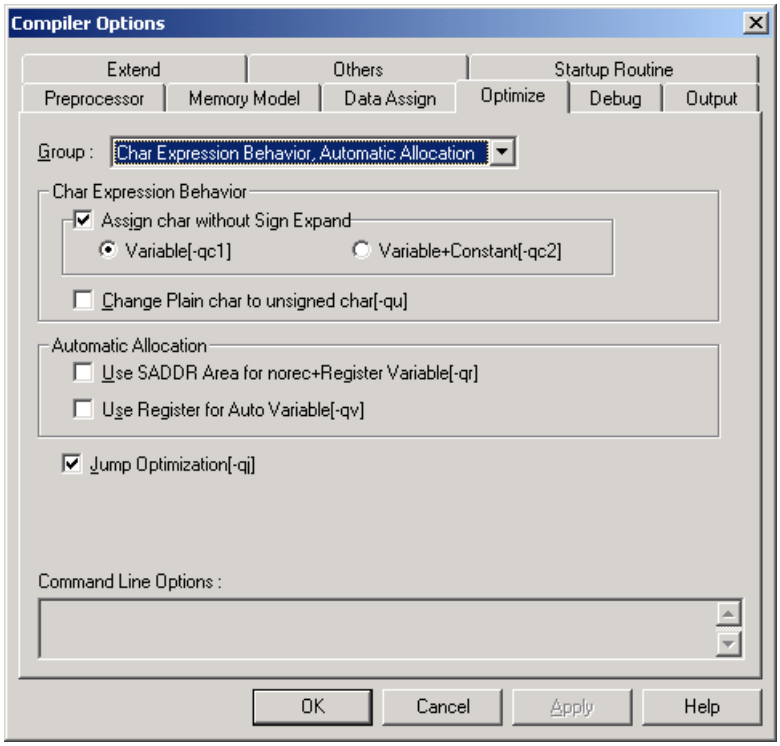
  -qx2 选项。 当执行效率和对象编码速率都很重要时，请选择这个选项。

  Code Size[-qx3]

  -qx3 选项。 看重对象编码效率，请选择该项。

(b) 在 [Group] 下拉菜单中选择 “Char Expression Behavior, Automatic Allocation”

图 3-11 < Compiler Options > 对话框 (在选择 << Char Expression Behavior, Automatic Allocation >> 时)



- 字符表达式动作

  指定字符为无符号扩展

  选中复选框使 -qc 选项有效 （不执行整体提升时)。

  通过选中单选按钮选择不进行字符扩展操作的字符类型。

  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，并且通过选中单选按钮来选择被分配的变量。

  对自动变量使用寄存器 [-qv]

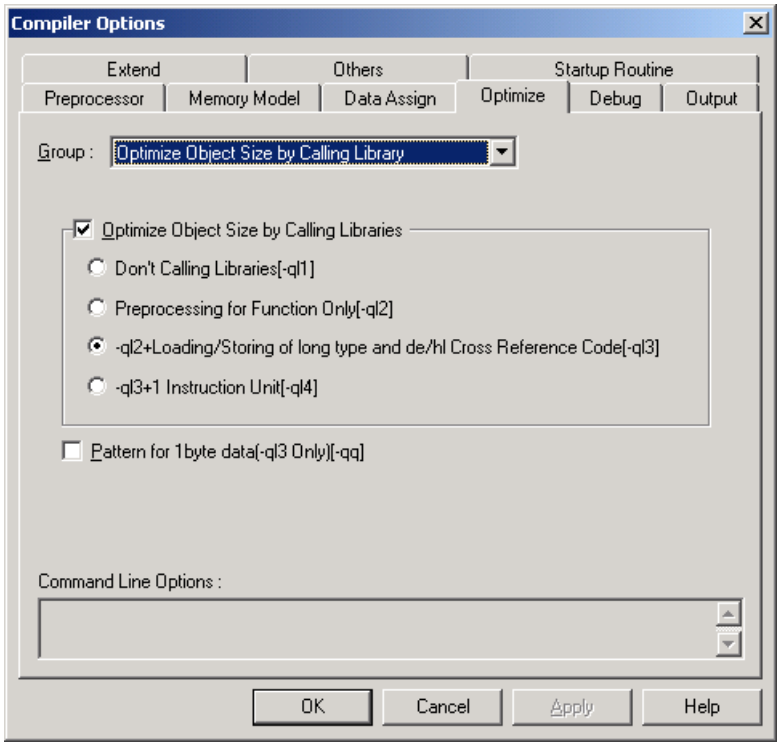
  选中该复选框来指定 -qv 选项。

- 跳转优化 [-qj]

  选中该复选框来指定 -qj 选项。

(c) 在 [Group] 下拉菜单中选择 "Optimize Object Size by Calling Library"

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



- 调用库来对目标程序大小进行优化

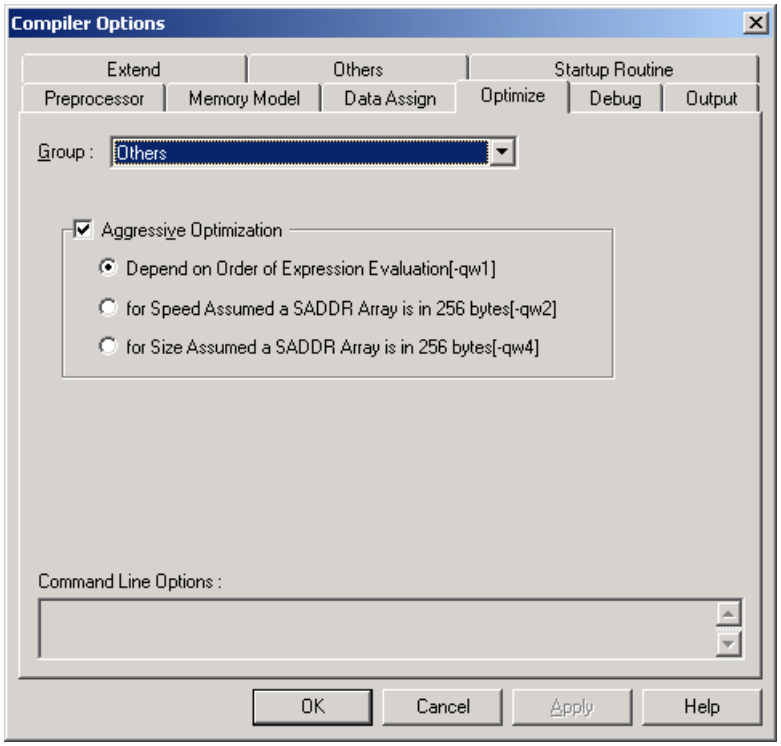
  选中该复选框来指定 -ql 选项，并点击单选按钮来指定目标优化的优先级别。 当 -qln 中的数字 n 越大，目标程序代码就越小，同时执行速度也会越慢。

- 用于 1 个字节数据的模式 (仅 -ql3)[-qq]

  选中该复选框来指定 -qq 选项。

(d) 在 [Group] 下拉菜单中选择 “Others”

图 3-13 < Compiler Options > 对话框 (当选择 << Others >> 时)

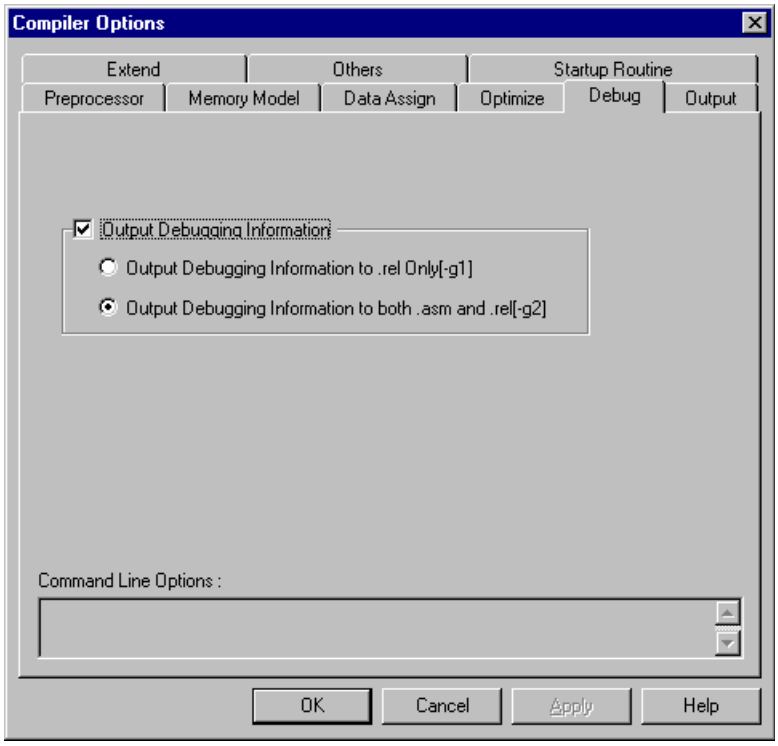


- 积极的优化

  选中该复选框来指定 -qw 选项。

(5) << Debug >> 标签

图 3-14 < 编译器选项 > 对话框 (当选择 << Debug >> 标签时)



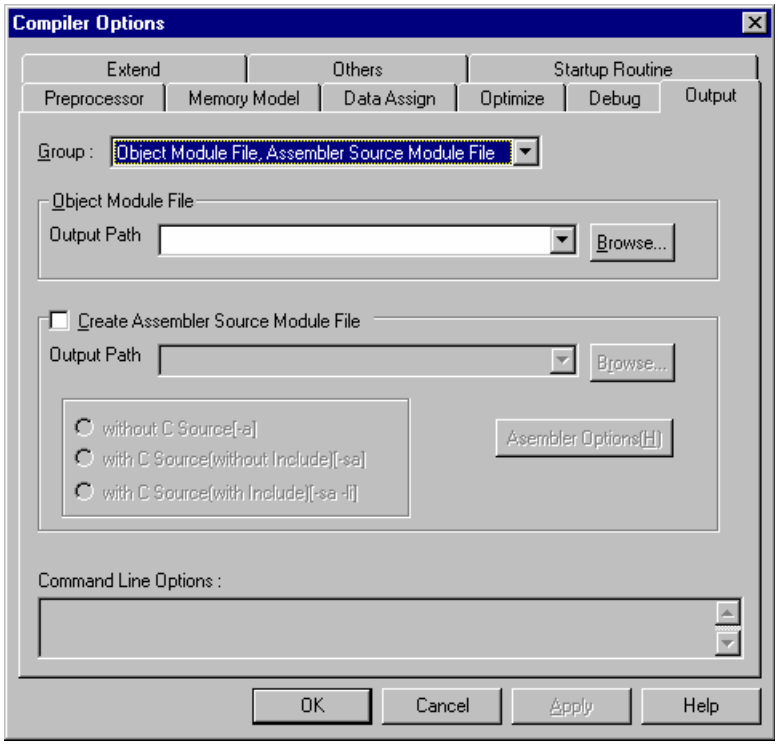
- 输出调试信息

  选中该选择框来生效 -g 选项，并且通过选中单选按钮选择输出调试信息的文件。 如果由 PM+ 选项使 [Debug] 无效 , 那么就不可能在 < Debug > 对话窗中设置，并且也不会输出调试信息。

(6) << Output >> 标签

(a) 在 [Group] 下拉菜单中选择 "Object Module File, Assembler Source Module File"

图 3-15 < 编译器选项 > 对话框 (当选择 << Object Module File, Assembler Source Module File >> 时)



- 目标模块文件

  为了指定目标模块文件输出路径，在组合框里输入具体路径。 也可以使用 [Browse] 按钮来指定。

  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。

- 汇编源模块文件

  选择这个复选框来使能 -a/-sa/-li 选项。可以选择要不要在汇编程序源文件模块文件中包含 / 不包含 C 源程序，也可以选择 C 源程序是否包含 / 不包含头文件，这三种情况都有对应的单选按钮。

  要指定汇编源模块文件的输出路径，需在组合框中输入路径名称。也可以使用 [Browse] 按钮来指定。

  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。

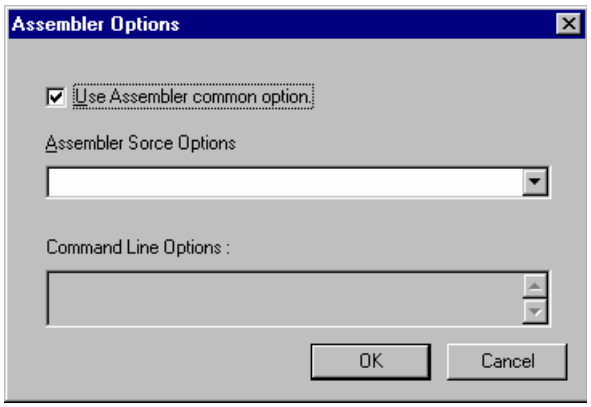
  如果没有指定任何选项，则认为指定了所有的汇编器选项来进行处理。

- < Assembler Options > 对话框

  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 se Assembler common option

  选择这个复选框，来使能所有 < Assembler Options > 对话框中的设置。

- 汇编源程序选项

  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。

  可以点击组合框右边的 [DropDownList] 来查看以前曾经输入的信息。

  备注　　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。

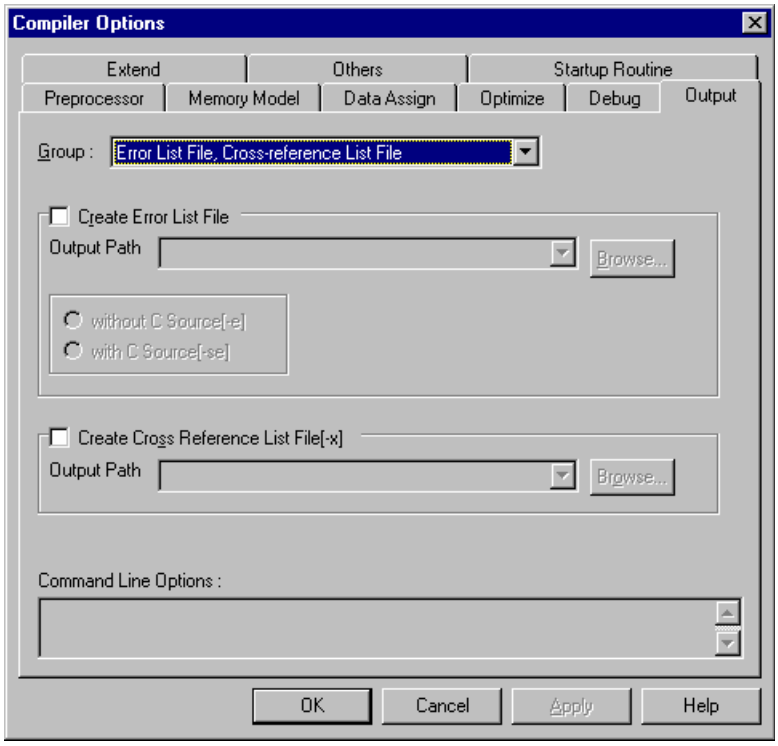
  如果字符串太长，此编辑对话框无法容纳，那么可以通过滚动 [ScrollBar] 来查看。

  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。

  共用汇编选项和输出汇编选项和其他选项一样以字符串形式显示。

(b) 在 [Group] 下拉菜单中选择 "Error List File, Cross-reference List File"

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



- 创建错误列表文件

  选中该复选框来指定 -e/-se 选项。 可以通过选择当前的单选按钮来选择是否将 C 源程序加入错误列表。

  为了指定错误列表文件输出路径，在组合框里输入具体路径。 也可以使用 [Browse] 按钮来指定。

  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]

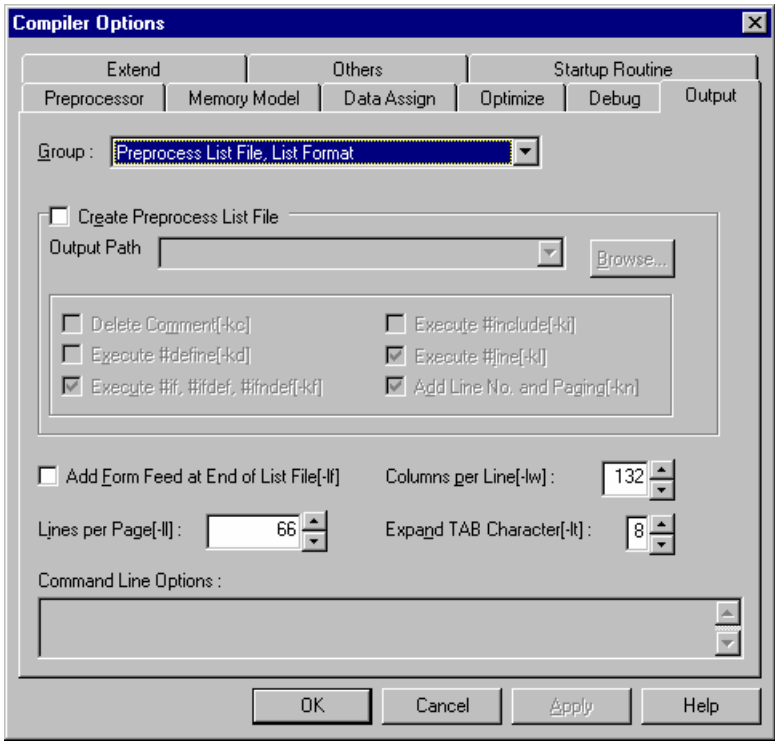
  选中该复选框来指定 -x 选项。 为了指定交叉引用列表文件输出路径，在组合框里输入具体路径。也可以使用 [Browse] 按钮来指定。

  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。

(c) 在 [Group] 下拉菜单中选择 “Preprocess List File, List Format”

图 3-18 < 编译器选项 > 对话框 (当选择 << Preprocess List File, List Format >> 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。

为了指定预处理列表文件输出路径，在组合框里输入具体路径。 也可以使用 [Browse] 按钮来指定。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。 为了增加 / 删除对话框里字符的数目，单击 [UpDown] 按钮。

- 每页的行数 [-ll]

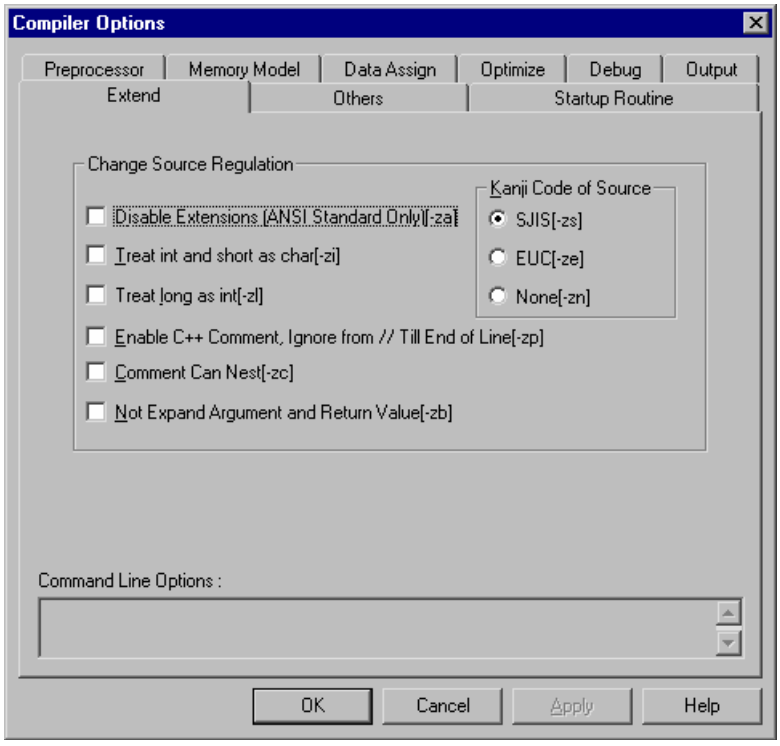
  使用 -ll 选项来指定一页中的行数。 为了增加 / 删除对话框里字符的数目，单击 [UpDown] 按钮。

- 扩展 TAB 字符 [-lt]

  使用 -lt 选项指定 tab 字符的跨度。 为了增加 / 删除对话框里字符的数目，单击 [UpDown] 按钮。

(7) << Extend >> 标签

图 3-19 < Compiler Options > 对话窗 (当 << Extend >> 标签选择时)



- 改变源文件的规则

  禁止扩展 (仅符合 ANSI 标准)[-za]

  选中该复选框来生效 -ZA 选项。

  把 int 和 short 处理为 char[-zi]

  选中该复选框来生效 -zi 选项。

  把 long 处理为 int[-zl]

  选中该复选框来生效 -zl 选项。

  这个选项在静态模式中是默认选项。

  启用 C++ 注释方式，从 // 开始直到行末的内容都被当作注释 [-zp]

  选中该复选框来生效 -ZP 选项。

  注释可以嵌套 [-zc]

  选中该复选框来生效 -ZC 选项。

  没有扩展参数和返回值 [-zb]

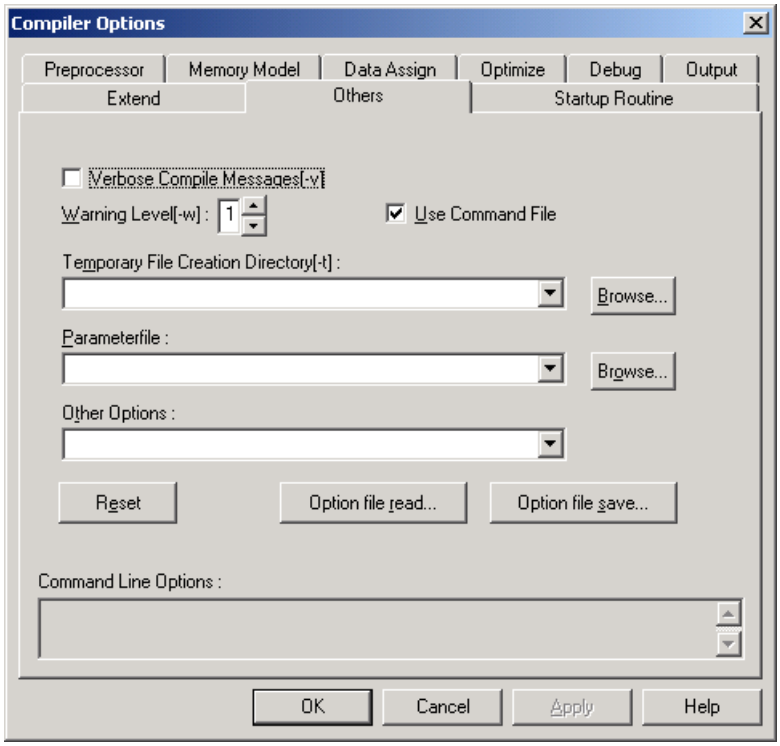
  选中该复选框来生效 -ZB 选项。

  源文件中的 K 编码

  选择相应的单选按钮来指定在源文件的注释中使用的 Kanji 编码类型。

(8) << Others >> 标签

图 3-20 < 编译器选项 > 对话窗 (当 << Others >> 标签选择时)



- 冗余编译信息 [-v]

  选中该复选框来生效 -V 选项。

- 警告级别 [-w]

  用 [UpDown] 按钮改变 -W 选项的等级。

- 使用命令文件

  通过选择这选择框，字符串的选项就输出到命令文件，所以无需关心选项字符串的长度。 本复选框默认为选中状态。

- 临时文件存放目录 [-t]

  用 -T 选项，在组合框中输入存储临时文件的目录。

- 参数文件

  用 -F 选项在组合框中输入参数文件名称。

  通过点击组合框右边的 [DropDownList] 来查看以前曾经输入的信息。

- Other 选择

  如果需要指定某些规范条目之外的编译器选项，请将选项输入组合框中。

  通过点击组合框右边的 [DropDownList] 来查看以前曾经输入的信息。

- [Reset] 按钮

  点击这个按钮设置默认选项。

- [Option file read] 按钮

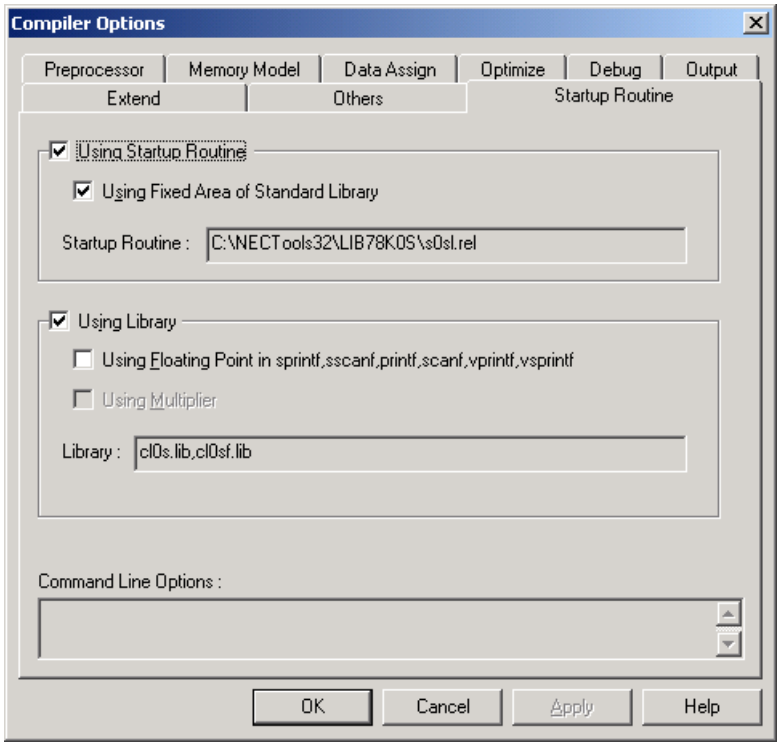
  点击这个按钮读入包含选项设置的选项信息文件。

- [Option file read] 按钮

  只有点击已经为 [OK] 或 [Apply] 的按钮之后这个按钮才有效。 选项的设置被保存在选项信息文件中。

(9) << Startup Routine >> 标签

图 3-21 < Compiler Options > 对话窗 (当 << Startup Routine >> 标签选择时)



< 时当指定源文件 Startup Routine 时 > 对话窗设置就不能完成。

- 使用启动路径

  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使用的固定区域。

  启动路径

  显示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 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。

如果指定 [Static Model[-sm]], 或 [Regard All Function as __pascal Except Varargs[-zr]] 选项 ,sprintf, sscanf, printf, scanf, vprintf, 和 vsprintf 函数不能用来支持浮点 。

使用乘法器

当使用具有乘法器产品的乘法器时选择这个选择框。

标出没有乘法器的产品不要选择。

库

显示使用的库文件名。

3.2 过程

3.2.1 使用 PM+

使用 PM+ 的编译方法描述如下。

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，点击 [Start] 按钮可在程序文件夹内建立 [NECTools32] 菜单，并且 PM+ 和其他程序也在该菜单中注册。

在菜单中单击 [PM plus] 就会启动 PM+。

(2) 创建工程

注册一个项目首先要用 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，请参考 PM+ 的用户手册。

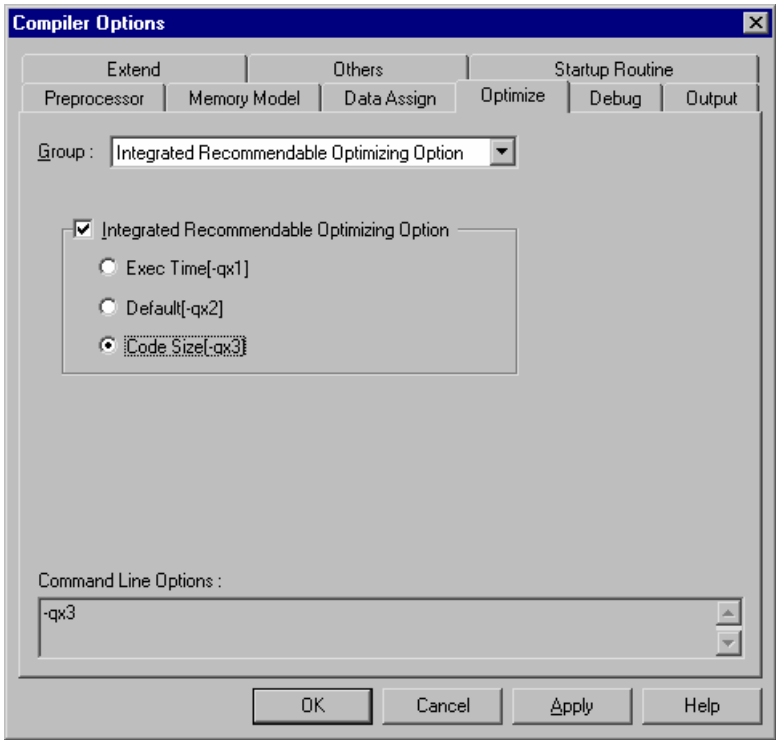
(3) 编译器和连接器的选项设定

为了能够成功建立 [Build]，在工程创建时就已经自动在编译文件中指定了最基本的必需选项。 项目特定的选项在 [Tools] 菜单中指定。

如果选择了 [Tools] 菜单中的 [Compiler Options]，将出现 < Compiler Options > 对话框。

下面是一个将优化选项从默认的 [-qcjlvw] 改为代码长度 [-qx3] 的例子。

图 3-22 < Compiler Options > 对话框 (当选择 << Optimize >> 标签)



如果在 < Compiler Options > 对话框中选择 << Startup Routine >> 标签下的 "Using Startup Routine” 的话，

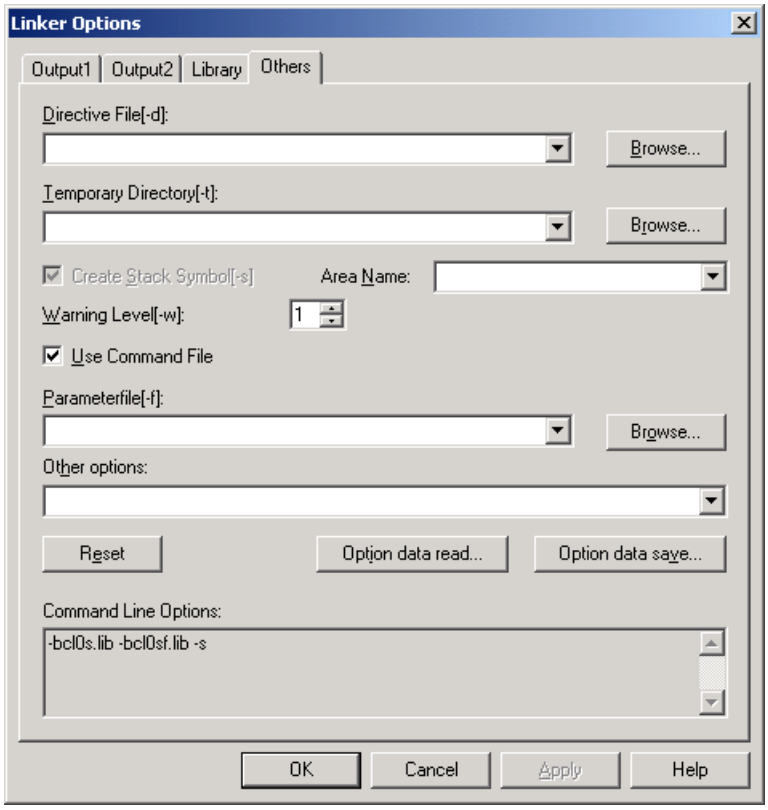
该编译器的标准启动例程将在所有源程序之前获取链接 （未显示在 < Linker Options 对话框中）。

当选择 "Using Library”时，该编译器的标准库将在所有其他库之后获取链接。

如果源文件中有 C 语言源程序，连接器会自动选择 -s 选项，栈符号自动生成。

启动例程文件的名称不会影响加载模块文件的名称。

图 3-23 < Linker Options > 对话框



(4) 建立项目

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] 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+ 的编译过程会由自动生成的编译文件启动。

建立完成后，会出现一个信息对话框。 检查此对话框可以知道建立过程是否正常完成。

备注　　建立时显示在 < OutPut > 窗口中的目录被保存到工程目录下，存储形式为”项目文件名 + .plg”。

3.2.2 使用命令行来编译连接 （对 DOS 提示符）

(1) 没有使用参数文件时

下列指令用来启动 CC78K0S，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 ] cc78k0s [ Δ option ] Δ C source name [ Δ option ]
>[ path name ] ra78k0s [ Δ option ] Δ assembler source name [ Δ option ]
>[ path name ] lk78k0s [ Δ option ] Δ object module name [ Δ option ]
```

注意事项 为了连接用户创建的库，一定要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，按顺序指定编译器附带的浮点库和编译器附属库。

要让 sprintf, sscanf, printf, scanf, vprintf 和 vsprintf 不支持浮点功能。按顺序指定编译器附属库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

(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 using 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 编译器附带的浮点库文件 （用 -b 选项指定）

(iii) 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)

(i) CC78K0S 附带的启动例程的目标文件

(ii) 2. 用户程序的目标模块文件

下面显示了链接 C 源程序 s1.c 和汇编程序 s2.asm 的示例。

```
C>cc78k0s -c9024 s1.c -e -a -iC:\NECTools32\inc78k0s -yC:\NECTools32\dev -
sm16
C>ra78k0s -c9024 s2.asm -e -yC:\NECTools32\dev
C>lk78k0s s0ss1.rel s1.rel s2.rel -bC:\NECTools32\lib78k0s\cl0ss.lib -s
-osample.lmf -yC:\NECTools32\dev
```

备注 指定多个编译选项时，用空格来分隔。 大写小写都无影响 （大小写不敏感）。 详情请参阅 " 第 5 章 编译器选项”。

-i 选项说明，-b 选项路径说明和 杜选项说明可以根据条件进行省略。 关于详情，请参考 " 第 5 章 编译器选项” 和 RA78K0S 汇编程序包的操作用户手册。

(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，应使用参数文件。

当使用参数文件，在命令行指定参数文件的设定选项。

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 ] cc78k0s Δ -f parameter file name
>[ path name ] ra78k0s Δ -f parameter file name
>[ path name ] lk78k0s Δ -f parameter file name
```

下面是一个使用的例程。

```
C>cc78k0s -fpara.pcc
C>ra78k0s -fpara.pra
C>lk78k0s -fpara.plk
```

程序员自己创建参数文件。 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。

下面是程序员创建参数文件的一个示例。

< para.pcc 的内容 >

```
-c9024 sl.c -e -a -iC:\NECTools32\inc78k0s -yC:\NECTools32\dev -sm16
```

< para.pra 的内容 >

```
-c9024 s2.asm -e -yC:\NECTools32\dev
```

< para.plk 的内容 >

```
s0ss1.rel s1.rel s2.rel -bC:\NECTools32\lib78k0s\cl0ss.lib -s -osample.lmf
-yC:\NECTools32\dev
```

-i 选项说明，-b 选项路径说明和 杜选项说明可以根据条件进行省略。 关于详情，请参考 " 第 5 章 编译器选项 ” 和 RA78K0S 汇编程序包的操作用户手册。

3.3 C 编译器的 I/O 文件

CC78K0S 输入 C 源模块文件写入 C 语言。 这些将转换为机器语言并作为目标模块文件输出。

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。

如果存在编译器错误，则输出错误信息到控制台和错误列表文件中。 如果产生错误，不能输出除错误列表文件外的其余各类文件。

CC78K0S 的输入 / 输出文件显示如下。

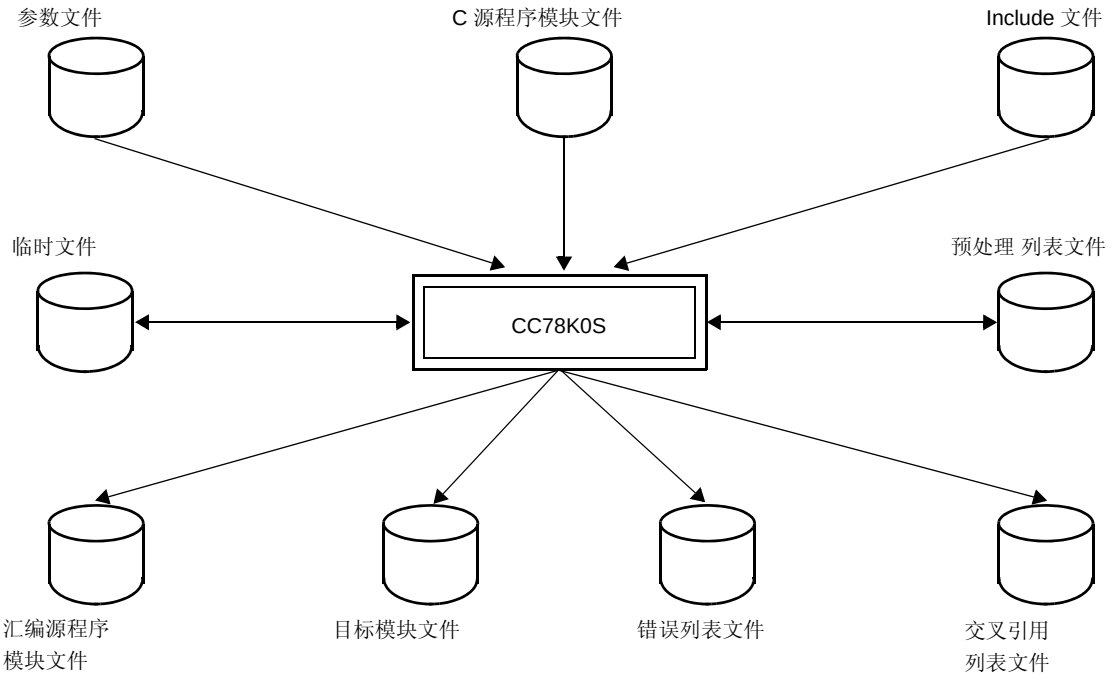
表 3-1 C 编译器的 I/O 文件

类型	文件名	说明	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
	Include 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	- 当用户想设定多重命令时，用户创建的文件在 C 编译器运行时不能在命令行中设定。	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，以及与机器语言地址定位相关的重定位信息和符号信息。	rel
	汇编源模块文件	- 由编译器输出目标代码的ASCII映像文件	asm
	预处理列表文件	- 通过预处理指令例如#include输出列表文件 - 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
	出错列表文件	- 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	- 编译时的中间文件 - 编译结束后没有产生错误时，用适当的名称重新命名文件，而编译结束后产生错误时则会删除文件。	$nn （固定的文件名）

注　　下列 4 种文件类型可用于错误列表文件。

- cer :　　与 *.C 文件一致并带有 C 源程序的错误列表文件 （由指定的 -se 选项输出）
- her :　　与 *.H 文件一致并带有 C 源程序的错误列表文件 （由指定的 -se 选项输出）
- er :　　与 CER 和 HER 之外的其他文件一致并带有 C 源程序的错误列表文件 （由指定的 -se 选项输出）
- ecc :　　错误列表文件不带有任何源文件对应的 C 源程序的 （由指定的 -se 选项输出）

图 3-24 C 编译器的 I/O 文件



备注　　如果存在编译错误，无法输出除了错误列表文件和交叉引用文件外的各类文件。

编译结束没有错误产生时，给临时文件重新命名为适当的名称。 如果编译结束有错误产生，则会删除临时文件。

3.4 执行开始和结束信息

3.4.1 执行开始消息

当 CC78K0S 启动时，开始信息会显示在控制台上。

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

3.4.2 执行结束消息

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 : uPD789xxx
Device file : Vx.xx

编译完成，没有发现错误和警告。
```

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) : CC78K0S warning W0745 : Expected function prototype
PRIME.C(20) : CC78K0S warning W0745 : Expected function prototype
PRIME.C(26) : CC78K0S warning W0622 : No return value
PRIME.C(37) : CC78K0S warning W0622 : No return value
PRIME.C(44) : CC78K0S warning W0622 : No return value

Target chip : uPD789xxx
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。

以下是一个输出错误信息的例子：

```
   C>cc78k0s -c9024 -e prime.c 杕

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

CC78K0S error F0018 : Option is not recognized ' -m '
Please enter ' CC78K0S  -- ' , if you want helps.
程序终止。
        :
```

在这个例子中，因为输入了一个不存在的编译选项，所以导致错误并停止编译器。

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　CC78K0S 函数

4.1　优化方法

在 CC78K0S 中，实现创建有效的目标模块文件的优化方法。表 4-1 列出了可以支持的优化方法。

表 4-1 优化方法

相位	内容	示例
语法		
(1)	常量计算编译过程中执行	`a = 3 * 5 ;  --> a = 15 ;`
(2)	基于逻辑表达式部分求值的真假判定	`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	有效地利用未使用的寄存器。
(5)	使用目标 CPU 中的特殊指令	`a = a + 1 ;  -->` 使用 `inc` 指令。 使用移动指令代替数组元素。
(6)	使用短指令。	如果存在相同操作的指令，使用较少字节的指令。 mov a, #0 或 xor a, a (根据设备的不同而不同)
(7)	将长跳转指令换为短跳转指令	中间代码在预处理时输出。
优化程序		
(8)	删除公用的部分表达式。	`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 ;  --> Delete`
		`After a = b ; , a is not referenced --> Delete`
		（a 是自动变量）

(11)	删除副本。	`a = b ;`
		`c = a + d ;  --> c = b + d ;`
		a 不再引用 （a 是自动变量）。

(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)	配制存储设备 （临时变量）	局部使用的变量分配到寄存器。

表 4-1 优化方法

相位	内容	示例
(14)	窥孔优化	特定模式的替换 例如 `a * 1 --> a , a + 0 --> a`
(15)	降低计算强度	例如 `a * 2 --> a + a , a << 1`
(16)	分配存储器设备 （寄存器变量）	快速分配数据到可使用的存储器。 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, -rk, -rs 选项)	变量自动分配到寄存器。

备注 无论优化选项如何设置，(1) 到 (7) 项都会执行。

(8) 到 (13)、 (17) 和 (18) 项只有在指定了对应的优化选项时才会执行。 (8) 至 (13) 的优化选项在将来会继续改进。

无论优化选项如何设置 (14) 和 (15) 项都会执行。

只有在 C 源程序中有寄存器声明的情况下才会执行 (16) 项。 然而，当 - 设定 -qr 选项时，仅分配 saddr 区。

关于优化选项的信息，请参阅 " 第 5 章 编译器选项 ”。

4.2 ROM 化函数

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写入 RAM 中。

CC78K0S 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，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

在链接期间，链接启动程序、目标模块文件和库。 启动程序初始化目标程序。

(1) s0s*.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。

在起始地址上添加标号 “_@cstart”（符号）。

(2) cl0s*.lib

CC78K0S 附属库。 CC78K0S 库文件包括以下两种。

运行时间库

"@@ 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。

标准库

"_"（下划线）添加到标准库名称的符号最前面。

(3) *.lib

用户创建的库。 添加 “_” 到符号头部。

备注 CC78K0S 提供不同种类的启动程序和库。 关于启动例程的细节，请参阅 " 第 8 章 启动程序”。 关于库的细节，请参阅 "2.6.4 库文件”。

第 5 章　编译器选项

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。

(1)　当 C 编译器启动时再命令行中指定。

(2)　指定 < 编译器选项中的 >PM+ 对话框。

(3)　在参数文件中指定。

关于上述编译器选项的指定方法，请参考 “第 3 章 由编译至链接过程”。

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。

编译器选项中不区分大、小写字符。

示例

```
cc78k0s Δ -c9024 Δ prime.c Δ -aprime.asm Δ -qx3
```

备注　　Δ : blanks such as spaces

5.2 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	-	-	-	-	-	-	-	-	NG	-
-q	NG	-	-	-	-	-	-	-	-	NG	-
-g	NG	-	-	-	-	-	-	-	-	NG	-
-k	-	-	Δ	NG	-	-	-	-	-	NG	-
-d	-	-	-	-	-	OK	-	-	-	NG	-
-u	-	-	-	-	OK	-	-	-	-	NG	-
-sa	-	-	-	-	-	-	NG	-	-	NG	-
-lw	-	-	Δ	-	-	-	Δ	Δ	Δ	NG	-
-ll	-	-	Δ	-	-	-	Δ	Δ	Δ	NG	-
-lt	-	-	Δ	-	-	-	Δ	Δ	Δ	NG	-
-lf	-	-	Δ	-	-	-	Δ	Δ	Δ	NG	-
-li	-	-	-	-	-	-	-	-	-	NG	Δ

[由 NG 标记的位置]

如果水平方向的选项被指定，那么垂直方向的选项无效。

[由 Δ 标记的位置]

如果水平方向的选项没被指定，那么垂直方向的选项无效。

[由 OK 标记的位置]

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 1

```
C>cc78k0s -c9024 -e sample.c -no -rd -g
```

备注　　-rd 和 -g 选项变为无效。

例 2

```
C>cc78k0s -c9024 -e sample.c -p -k
```

备注　　由于 -p 选项被指定， -k 选项有效。

例 3

```
C>cc78k0s -c9024 -e sample.c -utest -dtest = 1
```

备注　　因为 -d 选项是最后被指定的，所以 -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 4

```
C>cc78k0s -c9024 -e sample.c -o -no
```

备注　　因为 -no 选项是最后指定，所以 -o 选项无效， -no 选项优先。

没有在 表 5-1 描述的选项不受其它选项的影响。 然而，如果设定了帮助说明选项 （--/-?-h），则所有其他选项说明变成无效。 帮助选项 (--/-?-h) 在 PM 中无法指定。 为了在 PM+ 中使用帮助，请按下每个对话框中的 [帮助] 按钮。

5.3 类型

编译器选项分为下列 19 中类型。

表 5-2 编译选项列表

类型	选项	说明
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	指定目标模块文件的输出。
	-no	
存储分配说明	-r	指定存储器分配的方法
	-nr	
	-rd	设定外部变量 / 外部静态变量 (除了常数类型变量) 自动分配到 saddr 区域。
	-nr	
	-rk	设定函数参数和自动变量 (除了静态自动变量) 自动分配到 saddr 区域。
	-nr	
	-rs	设定静态自动变量自动分配到 saddr 区域。
	-nr	
优化说明	-q	设定优化类型。
	-nq	
调试信息输出说明	-g	设定输出 C 源代码级别的调试信息。
	-ng	
预处理列表文件创建说明	-p	设定预处理列表文件的输出。
	-k	设定预处理列表的处理。
预处理说明	-d	执行宏定义。
	-u	使宏定义作废。
	-i	从设定为包含文件的文件夹读取。
汇编源模块文件创建说明	-a	设定汇编源模块文件的输出。
	-sa	
错误列表文件创建说明	-e	设定错误列表文件的输出。
	-se	
交叉引用列表文件创建说明	-x	设定交叉参考列表文件的输出。
列表格式说明	-lw	在每个列表文件中设定一行的字符数。
	-ll	在每个列表文件中设定每页的行数。
	-lt	在每个列表文件的标签上更改字符的扩展数。
	-lf	在列表文件最后添加页中断码。
	-li	添加 include 文件中的 C 源代码到带 C 源代码注释的汇编源文件中。

表 5-2 编译选项列表

类型	选项	说明
警告输出说明	-w	指定输出至操作台的警告信息等级。
执行状态显示设定	-v	设定输出至控制台的编译执行状态。
	-nv	
参数文件说明	-f	从指定文件中导入输入文件名和选项。
临时文件创建文件夹说明	-t	指定创建临时文件的驱动和文件夹。
帮助说明	--	输出帮助信息至控制台。
	-?	
	-h	
函数扩展说明	-z	允许函数扩展进程。
	-nz	
驱动器文件搜索路径	-y	设定查找设备文件的路径。
静态模型说明	-sm	设定目标静态模型或常规模型。

5.4 说明

这一部分详细描述编译选项。

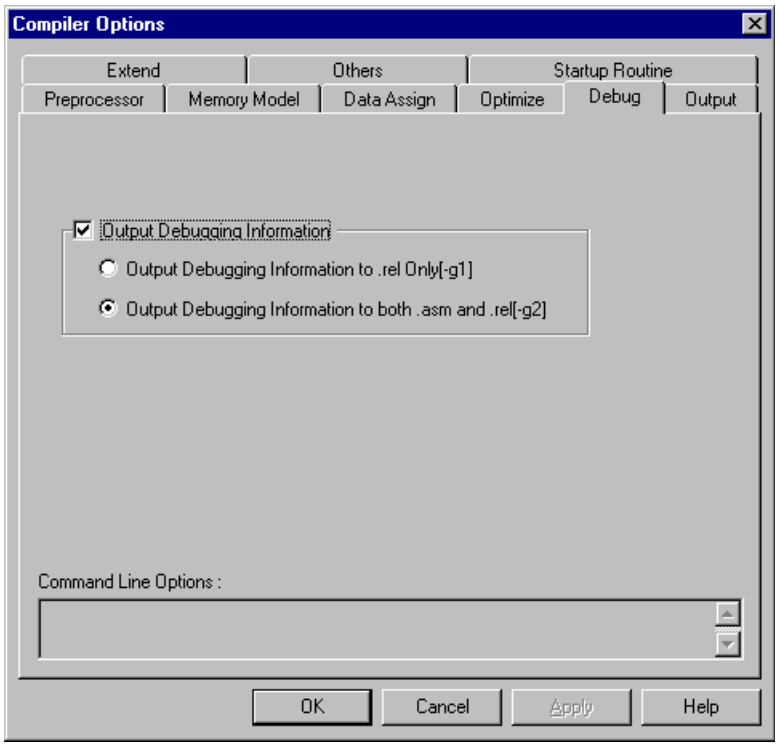
这个例子展示了如何从命令行中启动 CC78K0S。 为了在 PM+ 中启动 CC78K0S，需要指定命令，指定设备类型，并在 < Compiler Options > 对话框中指定 C 源程序遗漏的选项。

示例 (在命令行状态时)

```
C>cc78k0s -c9024 prime.c -g
```

示例 (当使用 PM+ 时)

图 5-1 < Compiler Options > 对话框



(1) 设备类型说明

设备类型说明 (-c)

[格式描述]

```
-c device-type
```

- 省略时解释

  None

[功能]

- -c 选项为执行编译设定目标器件。

[应用]

- 请务必确保指定这个选项。 C 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。

[说明]

- 用 -c 选项 + 相应的设备类型来说明目标设备文件的补充产品信息。
- 当使用 CC78K0S 时，需要设备文件。

[注意事项]

- -c 选项不能省略。 然而，如果在 C 源文件中有下列说明，则可以省略命令行中的说明。

```
#pragma pc (device type)
```

- 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设置了。

[使用示例]

- 在命令行中指定。 目标器件为 uPD789024。

```
C>cc78k0s -c9024 prime.c
```

- 在 C 源程序中进行说明，并启动编译器。

```
#pragma pc ( 9024 )
#define TRUE    1
#define FALSE   0
#define SIZE    200

char    mark [ SIZE + 1 ] ;

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

因此，在命令行中的目标设备的设定可以省略。

```
C>cc78k0s 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 源代码 >

```
#pragma pc ( 9024 )
#define TRUE    1
#define FALSE   0
#define SIZE    200

char    mark [ SIZE + 1 ] ;

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

< 命令行 >

```
C>cc78k0s -c9014 prime.c
```

在命令行执行后，编译器的执行过程如下。

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

sample\prime.c ( 1 )  : CC78K0S warning W0832 : Duplicated chip specifier
sample\prime.c ( 18 ) : CC78K0S warning W0745 : Expected function prototype
sample\prime.c ( 20 ) : CC78K0S warning W0745 : Expected function prototype
sample\prime.c ( 26 ) : CC78K0S warning W0622 : No return value
sample\prime.c ( 37 ) : CC78K0S warning W0622 : No return value
sample\prime.c ( 44 ) : CC78K0S warning W0622 : No return value

Target chip : uPD789014
Device file : Vx.xx

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

命令行中指定的目标设备具有较高的优先级。

(2) 目标模块文件创建说明

目标模块文件创建说明 (-o/-no)

[格式描述]

```
-o [ 输出文件名 ]
-no
```

- 省略时解释

  -o input-file-name.rel

[功能]

- -o 选项指定输出目标模块文件。 此外，还可以指定输出目录或输出文件名。
- -no 选项设定不输出目标模块文件。

[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。 从而缩短了编译时间。

[说明]

- 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。

[注意事项]

- 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。

[使用示例]

- 在这个示例里， -no 和 -o 选项都被选中。

```
C>cc78k0s -c9024 prime.c -no -o
```

(3) 存储分配说明

存储分配说明 (-r/-nr, -rd/-nr, -rk/-nr, -rs/-nr)

(a) -r/-nr

[格式描述]

```
-r [ 处理类型 ] ( 多种可能的说明 )
-nr
```

- 省略时解释

  -nr

[功能]

- -r 选项设定如何分配程序到存储器。
- -nr 选项使 -r 选项失效。

[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。

[说明]

- 通过 -r 选项设定处理的类型，如下所示： 不能省略处理类型设定。 否则，会出现严重错误 (F0012)。

表 5-3 -r 选项可设定的处理类型

处理类型	功能
b	从最高有效位 （MSB）分配位字段。
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。
k [n] [m] (n = 1, 2, 4)	在静态模式中，将函数函数和自动变量 (除了静态自动变量) 自动分配到 saddr 区域中，无论是否有 sreg 声明。
s [n] [m] (n = 1, 2, 4)	将自动变量自动分配到 saddr 区域中，无论是否有 sreg 声明。 m
c	不用为了将 2 字节或更长的结构成员分配到偶地址上而插入任何边界对齐数据，总之，会对结构体成员进行打包压缩。

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。

表 5-4 设定 NR 时的解释

处理类型	功能
b	从最低有效位 （LSB）分配位字段。
d	不自动分配任何变量到 saddr 区。
k	不自动分配任何变量到 saddr 区。
s	不自动分配任何变量到 saddr 区。
c	不要对任何结构体成员打包压缩。

[使用示例]

```
C>cc78k0s -c9024 -rds
```

(b) -rd/-nr

[格式描述]

```
-rd [ n ] [ m ] ( n = 1, 2, 4 )
-nr
```

- 省略时解释

  -nr

[功能]

- -rd 选项能够将外部变量 / 外部静态变量 (除了常数类型变量) 自动分配到 saddr 区域。
- -nr 选项使 -rd 选项失效。

[应用]

- 如果想要将外部变量 / 外部静态变量 (除了常数类型变量) 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。

[说明]

- 分配变量，并依据 n 的值和 m 的设定来做改变。

表 5-5 变量的最大范围 (-rd)

n÷μ	待分配的变量类型
1	字符型，无符号字符型
2	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
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，联合体，数组
Omitted	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。

(c) -rk/-nr

[格式描述]

```
-rk [ n ] [ m ] ( n = 1, 2, 4 )
-nr
```

- 省略时解释

  -nr

[功能]

- -rk 选项将自动把函数参数和自动变量 (除了静态自动变量) 分配到 saddr 区域。
- -nr 选项使 -rk 选项失效。

[应用]

- 在静态模式中，如果想把函数参数和自动变量 (除了静态自动变量) 自动分配到 saddr 区域，只需选定 -rk 选型即可，这与是否有 sreg 声明无关。

[说明]

- 分配变量，并依据 n 的值和 m 的设定来做改变。

表 5-6 分配变量的最大范围 (-rk)

n÷μ	待分配的变量类型
1	字符型，无符号字符型
2	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
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、联合体、数组
Omitted	所有变量

- 用 register 关键字声明过的变量不会被分配。
- 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 -rk 选项。
- 通过指定这个选项而分配到 saddr 区域的函数参数和自动变量，与 sreg 声明的函数参数和自动变量按照同样的方法来处理。

[注意事项]

- 该选项仅当 -sm 选项设定时有效。 如果 -sm 选项没有设定，可忽略 -rk 选项。

(d) -rs/-nr

[格式描述]

```
-rs [ n ] [ m ] ( n = 1, 2, 4 )
-nr
```

- 省略时解释

  -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 选项失效。

[应用]

- 如果想要将静态自动变量自动分配到 saddr 区域，只需选定一 rs 选项即可，这与是否有 sreg 声明无关。

[说明]

- 分配变量，并依据 n 的值和 m 的设定来做改变。

表 5-7 分配变量的最大范围 (-rs)

n÷μ	待分配的变量类型
1	字符型，无符号字符型
2	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
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，联合体，数组
Omitted	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。

(4) 优化说明

优化说明 (-q/-nq)

[格式描述]

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

- 省略时解释

  -qcjlw

[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 选项失效。

[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。 详情请参阅表 5-8。

[说明]

- 表 5-8 列出 -q 选项能够设定的优化类型。

表 5-8 优化类型

优化类型	过程说明
无说明	默认为 -qcjlw。
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 [n] (n = 1, 2)	直接进行字符型计算而无需提升为整型，这样会使编码更高效。 整型提升需要符合 ANSI-C 规定，在小于整形的类型 （char， short）计算时会转换为整型[注]。 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 : 只有变量不进行整型提升 2 : 不管变量还是常量都不进行整型提升
q	切换到专门为单字节库。
r	添加寄存器变量到寄存器并且分配该变量到 saddr 区。
j	优化转移指令。
x [n] (n = 1-3)	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 , 则默认 n = 2。 1 : 速度优先。 认为指定了 -qcjw 选项。 2 : 默认值。认为指定了 -q 选项。 3 : 代码大小优先。 认为指定了 -qcjl4w 选项。

表 5-8 优化类型

优化类型	过程说明
w [n] (n = 1, 2, 4)	通过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 (例如，交换某表达式的右侧子表达式和左侧子表达式执行次序，要求此表达式两侧都是运算项)。 然而，如果不包括选项 (尽管也符合标准，但由于 ANSI-C 标准忽略了一些操作符，也没有设置执行次序)，执行结果有时候会有些出入。 根据 ANSI-C 标准，这在正常的写入源中不是问题。 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 : 改变表达式内部的执行次序 2 : 改变表达式内部的执行次序。 当字符、无符号字符、短型、无符号短型，整数型或者无符号整数型数组分配到 saddr 空间且被无符号字符变量引用，假设数组的大小不会超出 256 个字节数，则计算地址时无需进位并给予速度较高优先级。 3 : 改变表达式内部的执行次序。 当字符、无符号字符、短型、无符号短型，整数型或者无符号整数型数组分配到 saddr 空间且被无符号字符变量引用，假设数组的大小不会超出 256 个字节数，则计算地址时无需进位并给予代码大小较高优先级。
v	将一个自动变量自动分配到寄存器或 saddr 区域。
l [n] (n = 1-4)	用库来代替常数编码模式。 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 3 : 在处理函数之前 / 之后才会执行，加载 / 保存一个长整型变量，DE/HL 间接引用代码 4 : 在一个函数和一条指令之前 / 之后才会执行处理

注　在 CC78K0S 中指定了－ QC 选项时，常数类型和字符常数类型将会按照以下方式来处理。

```
0 to 127 , 0x00 to 0x7F , 00 to 0177          char type
128 to 255 , 0x80 to 0xFF , 0200 to 0377      unsigned char type
0U to 255U                                    unsigned char type
' \0 ' to ' \377 '                            char type
```

然而，当设定 -qu 选项时，从 ‘\200’ 到 ‘\377’ 范围内的字符常量的处理当成无符号的 char 类型常量并获取从 +128 到 +255 的值。

带 ñ(负号) 的常量处理方法如下。

```
ñ0 to 128       char type
From ñ129       int type
```

如果常量或变量计算溢出，将常量或变量转化为可以代表计算结果的类型。 通过暂代或指定 -qI 选项，数据类型的改变可以避免。 当设定 -qc1 选项，常量计算为符号扩展类型。

- 可以设定多重优化类型。
- 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 -QJ 选项 -> 删除 -qcjlw)。
- 如果没有输出目标模块文件和汇编源模块文件，那么 -qu 选项和 -q 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。
- 如果 -qr 和 -sm 同时被指定，会输出警告信息，并且 -qr 选项失效。

[使用示例]

- 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>cc78k0s -c9024 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。

```
C>cc78k0s -c9024 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>cc78k0s -c9024 prime.c -qcr
```

(5) 调试信息输出说明

调试信息输出说明 (-g/-ng)

[格式描述]

```
-g [ n ] ( n = 1, 2 )
-nq
```

- 省略时解释

  -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 选项禁止 -g 选项。

[应用]

- 如果没有设定 -g 选项，输入到调试器中的目标模块文件所需要的行号和符号信息将不输出。 因此，在源代码级别的调试，所有的模块通过设定 -g 选项进行链接编译。

[说明]

- 因为 n 值的不同操作会有所改变。

表 5-9 n 值不同引起的操作改变

n 值	功能
Omitted	默认为 n = 2.
1	仅将调试信息 （从 $DGS 或 $DGL 开始的信息）添加到目标模块文件。 没有调试信息被加入到汇编源模块文件中。 该选项使引用汇编文件更为简单。 因为调试信息添加到可用的目标文件的源代码调试。
2	添加调试信息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。
- 如果没有输出目标模块文件和汇编源模块文件， -g 选项无效。

[使用示例]

- 指定了 -g 选项。

```
C>cc78k0s -c9024 prime.c -g
```

(6) 预处理列表文件创建说明

预处理列表文件创建说明 (-p, -k)

(a) -p

[格式描述]

```
-p [ 输出文件名 ]
```

- 省略时解释

  无 (无文件输出)

[功能]

- -p 选项设定预处理列表文件的输出。 此外，还可以指定输出目录或输出文件名。 如果省略 -p 选项，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。

[说明]

- 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。

[注意事项]

- 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。

[使用示例]

- 预处理列表文件 sample.ppl 被输出。

```
C>cc78k0s -c9024 prime.c -psample.ppl
```

(b) -k

[格式描述]

-k[处理类型] (可以指定多种规范)

- 省略时解释

  -kfln

[功能]

- -k 选项设定预处理列表的处理。

[应用]

- 当注释被删除或使用了宏定义扩展，此时输出预处理列表文件需要指定该选项。

[说明]

- 下表中列出了 -k 选项指定的处理类型。

表 5-10 -k 选项的处理类型

处理类型	说明
Omitted	与指定 -fln 一致
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	同指定了 FLN 一样

备注 可以指定多种处理类型。

- 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。

[使用示例]

- 从预处理列表文件 prime.ppl 中删除注释，加入行号和分页处理。

```
C>cc78k0s -c9024 prime.c -p -kcn
```

引用 prime.ppl。

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

Command         : -c9024 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     : uPD789024
Device file     : Vx.xx
*/
```

(7) 预处理说明

预处理说明 (-d, -u, -i)

(a) -d

[格式描述]

```
-d 宏名 [= 定义名 ] [， 宏名 [= 定义名 ]]...
( 可以指定多项说明 )
```

- 省略时解释

  只有在 C 源程序模块文件中的宏定义才有效。

[功能]

- -d 选项设定了与 C 源文件内 #define 语句相同的宏定义。

[应用]

- 如果需要用指定常量替换全部的宏名时，指定该选项。

[说明]

- 用逗号 “,” 分隔每个定义内容，一次可以完成多个宏的定义。
- 在紧邻 ‘=’ 和 “,” 的前后不允许出现空格。
- 如果定义名被忽视，该名字被定义为 “1”。
- 如果在 -d 和 -u 选项中指定了相同的宏名，最后指定的那个宏名有效。

[使用示例]

```
C>cc78k0s -c9024 prime.c -dTEST , TIME = 10
```

(b) -u

[格式描述]

```
-u 宏名 [ , 宏名 ]... ( 可以指定多项说明 )
```

- 省略时解释
  使用 -d 设定的宏定义有效。

[功能]

- -u 选项使与 C 源文件中 #undef 语句相同的宏定义无效。

[应用]

- 指定了该选项，则用 -d 选项定义的宏名会无效。

[说明]

- 用逗号 “,” 分隔每个定义内容，一次可以完成多个宏的取消。 在紧邻逗号 “,” 的前后不允许出现空格。
- 可以通过 -u 选项设定失效的宏定义是由 -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，最后指定的那个宏名有效。

[使用示例]

- 通过 -d 和 -u 选项指定相同的宏名。 在这个例子中， TEXT 宏被禁止。

```
C>cc78k0s -c9024 prime.c -dTEST -uTEST
```

(c) -i

[格式描述]

```
-i 文件夹 [ , 文件夹 ] ... ( 可以指定多项说明 )
```

- 省略时解释

  (i) 包含源文件的文件夹[注 1]

  (ii) 通过环境变量 INC78K0S 设定的文件夹

  (iii) C:\NECTools32\inc78k0s[Note 2]

[功能]

- -i 选项的功能是从指定文件夹查找输入 C 源程序中 #include 语句指定的包含文件。

[应用]

- 需要从某个确定文件夹查找包含文件时，请指定该选项。

[说明]

- 用逗号 “,” 分隔每个定义内容，一次可以指定多个目录。
- 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) 包含源文件的文件夹[注 1]

  (ii) 用 -i 选项指定的文件夹

  (iii) 通过环境变量 INC78K0S 设定的文件夹

  (iv) C:\NECTools32\inc78k0s [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 include 文件名 < > 设定，则不执行查找。

  注 2。 这个示例需要提前将 CC78K0S 安装到 C:\NECTools32 目录。

[使用示例]

- 指定了 -i 选项。

```
C>cc78k0s -c9024 prime.c -ib: , b:\sample
```

(8) 汇编源模块文件创建说明

汇编源模块文件创建说明 (-a, -sa)

(a) -a

[格式描述]

```
-a[ 输出文件名 ]
```

- 省略时解释

  没有汇编源程序输出。

- 输出文件

  .asm (: alphanumeric symbols)

[功能]

- -a 选项指定汇编源模块文件的输出。 此外，还可以指定输出目录或输出文件名。

[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。

[说明]

- 磁盘文件名或者设备文件名都可以指定为文件名称。
- 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 枹 a 选项同时被选定，则忽略 -sa 选项。

[注意事项]

- 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” 组合框里指定文件名或输出目录。

[使用示例]

- 创建汇编源模块文件 sample.asm 的示例。

```
C>cc78k0s -c9024 prime.c -asample.asm
```

(b) -sa

[格式描述]

```
-sa [ 输出文件名 ]
```

- 省略时解释
  没有汇编源程序输出。
- 输出文件
  .asm (: alphanumeric symbols)

[功能]

- -sa 选项添加 C 源作为汇编源模块文件的注释。 此外，还可以指定输出目录或输出文件名。

[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。

[说明]

- 磁盘文件名或者设备文件名都可以指定为文件名称。
- 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 源没有添加到输出汇编源模块的注释。 然而，如果设定 -li 选项， include 文件中的 C 源代码添加到注释。

[注意事项]

- 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 “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” 组合框里指定文件名或输出目录。

[使用示例]

- 指定 -sa 选项。

```
C>cc78k0s -c9024 prime.c -sa
```

prime.asm 的内容如下：

```
; 78K/0S Series C Compiler Vx.xx Assembler Source
;                                                Date:xx xxx xxxx Time:xx:xx:xx

; Command       : -c9024 prime.c -sa
; In-file   : prime.c
; Asm-file  : prime.asm
; Para-file :

$PROCESSOR(9024)
$DEBUG
$NODEBUGA
$KANJICODE SJIS
$TOL_INF        03FH , 0130H , 02H , 00H

$DGS    FIL_NAM ,       .file , 033H , 0FFFEH , 03FH , 067H , 01H , 00H
$DGS    AUX_FIL ,       prime.c
$DGS    MOD_NAM ,       prime , 00H , 0FFFEH , 00H , 077H , 00H , 00H
          F
        EXTRN   _@RTARG0
        EXTRN   @@isrem
        PUBLIC  _mark
        PUBLIC  _main
        PUBLIC  _printf
        PUBLIC  _putchar
          F
@@CODE CSEG
_main :
$DGL   1 , 13
        push    hl                              ; [ INF ] 1 , 4
        movw    ax , #08H                       ; [ INF ] 3 , 6
        callt   [ _@cprep ]                     ; [ INF ] 1 , 8
??bf_main :
; line   9 :    int i , prime , k , count ;
; line  10 :
; line  11 :    count = 0 ;
$DGL    0 , 4
        xor     a , a                           ; [ INF ] 2 , 4
        mov     [ hl ] , a      ; count         ; [ INF ] 1 , 6
        mov     [ hl + 1 ] , a  ; count         ; [ INF ] 2 , 6
; line  12 :
; line  13 :    for ( i = 0 ; i <= SIZE ; i++ )
$DGL    0 , 6
        mov     [ hl + 6 ] , a  ; i             ; [ INF ] 2 , 6
        mov     [ hl + 7 ] , a  ; i             ; [ INF ] 2 , 6
?L0003 :
        mov     a , [ hl + 6 ]  ; i             ; [ INF ] 2 , 6
        xch     a , x                           ; [ INF ] 1 , 4
        mov     a , [ hl + 7 ]  ; i             ; [ INF ] 2 , 6
        xor     a , #080H       ; 128           ; [ INF ] 2 , 4
        cmpw    ax , #080C8H    ; -32568        ; [ INF ] 3 , 6
        bc      $$ + 4                          ; [ INF ] 2 , 6
        bnz     $?L0004                         ; [ INF ] 2 , 6
          :
        END
```

```
; *** Code Information ***
;
; $FILE H : \um\prime.c
;
; $FUNC main ( 8 )
;       bc = ( void )
;       CODE SIZE = 222 bytes , CLOCK_SIZE = 654 clocks , STACK_SIZE = 14 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 ( 31 )
;       bc = ( pointer s : ax , int i : [ sp + 2 ] )
;       CODE SIZE = 28 bytes , CLOCK_SIZE = 108 clocks , STACK_SIZE = 10 bytes
;
; $FUNC printf ( 41 )
;       bc = ( char c : x )
;       CODE SIZE = 14 bytes , CLOCK_SIZE = 58 clocks , STACK_SIZE = 8 bytes

; Target chip : uPD789024
; Device file : Vx.xx
```

(9) 错误列表文件创建说明

错误列表文件创建说明 (-e, -se)

(a) -e

[格式描述]

```
-e [ 输出文件名 ]
```

- 省略时解释

  没有错误列表文件输出。

- 输出文件

  .ecc (: alphanumeric symbols)

[功能]

- -e 选项指定错误表文件的输出。 此外，还可以指定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。

[说明]

- 磁盘文件名或者设备文件名都可以指定为文件名称。
- 当指定了 -e 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 .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。

[注意事项]

- 要在 PM+ 中改变输出目录，需要在 << Output >> 标签和选择 “without C Source[-e]” 下的 “Create Error List File” 区域中的 << 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。

[使用示例]

- 指定了 -e 选项。

```
C>cc78k0s -c9024 prime.c -e
```

错误列表文件内容如下。

```
prime.c ( 18 ) : CC78K0S warning W0745 : Expected function prototype
prime.c ( 20 ) : CC78K0S warning W0745 : Expected function prototype
prime.c ( 26 ) : CC78K0S warning W0622 : No return value
prime.c ( 37 ) : CC78K0S warning W0622 : No return value
prime.c ( 44 ) : CC78K0S warning W0622 : No return value

Target chip : uPD789024
Device file : Vx.xx

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

(b) -se

[格式描述]

```
-se [ 输出文件名 ]
```

- 省略时解释

  没有错误列表文件输出。

- 输出文件

  *.cer : Error list for *.c files (* : alphanumeric symbols)

  *.her : Error list for *.h files

  *.er : Error list for files other than *.c and *.h files

[功能]

- -se 选项添加 C 源模块文件到错误列表文件。 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。

[说明]

- 磁盘文件名或者设备文件名都可以指定为文件名称。
- 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，如果驱动器名被忽略，错误列表文件将输出到当前驱动器。
- 不能给 include 文件设定文件夹和文件名。 如果 include 文件的文件类型是 “H” , 类型为 “her”的错误列表文件被输出到当前驱动器。 如果 include 文件的文件类型是 ‘C’，类型为 “cer”的错误列表文件输出。 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，不为 include 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。

[注意事项]

- 在 PM+ 中要改变输出目录，可以在 << Output >> 标签页下 “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[-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”组合框里指定文件名或输出目录。

[使用示例]

- 指定了 -se 选项。

```
C>cc78k0s -c9024 prime.c -se
```

prime.cer 文件内容如下。

```
/*
78K/0S Series C Compiler VX.XX Error List Date : XX XXX XXXX Time : XX : XX : XX

Command         : -c9024  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 ) ;
*** CC78K0S warning W0745 : Expected function prototype
        count++ ;
        if ( ( count%8 ) == 0 ) putchar ( '\n' ) ;
*** CC78K0S warning W0745 : Expected function prototype
        for ( k = i + prime ; k <= SIZE ; k += prime )
            :
}
```

(10) 交叉引用列表文件创建说明

交叉引用列表文件创建说明 （-x）

[格式描述]

```
-x [ 输出文件名 ]
```

- 省略时解释
  没有交叉引用列表文件输出。
- 输出文件
  .xrf (: alphanumeric symbols)

[功能]

- -x 选项设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。

[Application]

- 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。

[说明]

- 磁盘文件名或者设备文件名都可以指定为文件名称。
- 当指定了 -x 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 .xrf”。
- 即使除 CO101 以外的内部错误或错误编号 F0024 （或从 E 开始编号）的编译错误发生，交叉引用列表文件仍然创建。 然而，文件的内容不能保证。

[注意事项]

- 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。

[使用示例]

- 指定了 -x 选项。

```
C>cc78k0s -c9024 prime.c -x
```

prime.xrf 文件内容如下。

```
78K/0S Series C Compiler Vx.xx Cross reference List  Date : XX XXX XXXX Page : 1

Command         : -c9024 prime -x
In-file         : prime.c
Xref-file       : prime.xrf
Para-file       :

ATTRIB MODIFY TYPE SYMBOL   DEFIN   REFERENCE

EXTERN    array   mark       5      14      16      22
EXTERN    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    func    printf    28      18      25
EXTERN    func    putchar   39      20
PARAM     pointer s         29      36
PARAM     int     i         30      35
AUTO1     int     j         32      35
AUTO1     pointer ss        33      36
PARAM     char    c         40      43
AUTO1     char    d         42      43
          #define TRUE       1      14
          #define FALSE      2      22
          #define SIZE       3       5       13     15      21


Target chip     : uPD789024
Device file     : Vx.xx
```

(11) 列表格式说明

列表格式说明 (-lw, -ll, -lt, -lf, -li)

(a) -lw

[格式描述]

```
-lw [ 字符数量 ]
```

- 省略时解释

  -lw132 (对于控制台的输出，这就是 80 个字符)

[功能]

- -lw 选项设定每个类型列表文件每行的字符数。

[Application]

- 指定 -lw 选项可以改变列表文件中每一行的字符数量。

[说明]

- 用 -lw 选项可以指定字符数量的范围如下，但是不包括结束符 (CR， LF)，具体如下。

  72 ≤ 每行能够打印的字符数量 ≤ 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)。
- 如果列表文件没有作任何说明，那么 -lw 选项无效。

[使用示例]

- 当没有指定 -lw 选项时，交叉引用列表输出为 “文件名 .xrf”。

```
C>cc78k0s -c9024 prime.c -x
```

(b) -ll

[格式描述]

```
-LL [ 行数量 ]
```

- 省略时解释

  -ll66 (对于控制台的输出，这就是 65,535 个字符)

[功能]

- -ll 选项指定了所有列表文件中每一页的行数。

[Application]

- 如果需要改变列表文件中每页的行数，请指定 -ll 选项。

[说明]

- 通过 -ll 选项可以指定的行数范围如下。

  20 ≤ 每页能够打印的行数 ≤ 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，那么每页的行数会默认为 66 行 (如果输出到控制台，每页的行数就会变为 65535 行)。
- 如果列表文件没有作任何说明，那么 -ll 选项无效。

[使用示例]

- 交叉引用文件每页的行数被设定为 20 行。

```
C>cc78k0s -c9024 prime.c -x -ll20
```

(c) -lt

[格式描述]

```
-lt [ 字符数量 ]
```

- 省略时解释
  -lt8

[功能]

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

[Application]

- 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。

[说明]

- -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。

[使用示例]

- -lt 选项被忽略。

```
C>cc78k0s -c9024 prime.c -p
```

- 基于 HT 编码的空白数量被设置为 1。

```
C>cc78k0s -c9024 prime.c -p -lt1
```

(d) -lf

[格式描述]

```
-lf
```

- 省略时解释

  None

[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。

[说明]

- 如果列表文件没有作任何说明，那么 -lf 选项无效。

[使用示例]

- 指定 -lf 选项。

```
C>cc78k0s -c9024 prime.c -a -lf
```

(e) -li

[格式描述]

```
-li
```

- 省略时解释

  None

[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。

[说明]

- 如果没有指定 -sa 选项，则该选项被忽略。

[使用示例]

- 指定 -li 选项。

```
C>cc78k0s -c9024 prime.c -sa -li
```

(12) 警告输出说明

警告输出说明 (-w)

[格式描述]

```
-w [ level ]
```

- 省略时解释

  -w1

[功能]

- 指定 -w 选项会将警告信息输出到控制台。

[应用]

- 该选项指定了是否输出警告信息到控制台。 详细消息也能输出。

[说明]

- 下面给出警告信息的等级。

表 5-11 警告信息等级

级别	说明
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）。

[使用示例]

- 当 -w 选项被忽略时会引用警告信息。

```
C>cc78k0s -c9024 prime.c
```

(13) 执行状态显示设定

执行状态显示说明 (-v/-nv)

[格式描述]

```
-v
-nv
```

- 省略时解释

  -nv

[功能]

- -v 选项输出当前编译的执行状态到控制台。
- -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。

[说明]

- 处理中的阶段名和函数名输出。
- 如果 -v 选项和 -nv 选项同时都被选定，那么最后的选项有效。

[使用示例]

- -v 选项被选定。

```
C>cc78k0s -c9024 prime.c -v
```

(14) 参数文件说明

参数文件说明 (-f)

[格式描述]

```
-f 文件名
```

- 省略时解释

  只能从命令行输入选项和输入文件名。

[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，因为编译时输入了多个选项。
- 当编译过程需要重复指定选项时，在参数文件中描述选项并且指定 -f 选项。

[说明]

- 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。
- 在参量文件中描述的选项或者输入文件名会进行扩展，当参量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“#”后直到行尾的字符都被当作注释。

[使用示例]

- 参数文件 prime.pcc 的内容。

```
; parameter file
prime.c -c9024 -aprime.asm -e -x
```

prime.pcc 在编译中的使用。

```
C>cc78k0s -fprime.pcc
```

(15) 临时文件创建文件夹说明

临时文件创建文件夹说明 (-t)

[格式描述]

```
-t 文件夹
```

- 省略时解释

  文件被创建在环境变量 TMP 指定的驱动和文件夹中。 若不指定，文件将在当前驱动和当前文件夹内创建。

[功能]

- -t 选项指定创建临时文件的驱动和文件夹。

[应用]

- 创建临时文件的位置可以用 -T 选项指定。

[说明]

- 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。
- 临时文件在存储器中占去所需的存储大小。 如果需要的存储大小不再可用，在指定文件夹创建的临时文件以及存储内容写入文件。 访问后来的临时文件就是访问不在存储器中的文件。
- 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。

[使用示例]

- 下列命令指定了 TMP 文件夹作为临时文件的输出位置。

```
C>cc78k0s -c9024 prime.c -ttmp
```

(16) 帮助说明

帮助说明 (--/-?/-h)

[格式描述]

```
--
-?
-h
```

- 省略时解释

  无屏幕显示

[功能]

- --， -?，和 -h 选项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+ 中指定该选项。 为了在 PM+ 中使用帮助，请按下 < Compiler Options > 对话框中的 [Help] 按钮。

[应用]

- 显示选项及其说明。 运行 C 编译器时参考。

[说明]

- 当 --， -?，或 -h 选项被选中时，所有其它的编译选项均不可用。
- 当需要查阅正在显示的帮助信息的延续内容，可以按下返回键。 要在结束之前需要退出显示，请按下除返回键的任意字符，然后按下返回键。

[使用示例]

- 指定 -h 选项。

```
C>cc78k0s -h
```

(17) 函数扩展说明

函数扩展说明 (-z/-nz)

[格式描述]

```
-z[ 类型 ] ( 如果需要指定多种类型，连续指定即可 )
-nz
```

- 省略时解释

  -nz

[功能]

- -z 选项能够指定多种函数扩展类型的处理。
- -nz 选项使 -z 选项失效。
- 类型不能被忽略，否则会发生严重错误 (F0012)。

[应用]

- 以下类型说明的函数处理过程同样适用于 78K0S 系列扩展函数。

[说明]

- -z 选项的类型说明如下。

表 5-12 -z 选项的类型说明

类型说明	说明
Omitted	默认指定了 -nz 选项。
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	说明注释不包含 kanji 代码。
b	char-/unsigned char 类型自变量和返回值不是整型扩展的。

表 5-12 -z 选项的类型说明

类型说明	说明
a	ANSI 标准之外的函数都认为是非法的。 只有和 ANSI 标准兼容的函数才有效。 下述任务逐一地执行。 - 下述不再是保留字。 callt, noauto, norec, sreg, bit, boolean, #asm, #endasm - 三字母序列 (3 个字母表示法) 有效。 - 编译器定义的宏 __STDC__ 是 1。 - 针对字符型位域会输出下列警告。 (CC78K0S warning W0787: Bit field type is char) - 如果指定了 -w2 选项， -qc， -zp， -zc， -zi 和 -zl 选项会输出下列警告。 (CC78K0S warning W0029: '-QC' option is not portable) (CC78K0S warning W0031: '-ZP' option is not portable) (CC78K0S warning W0032: '-ZC' option is not portable) (CC78K0S warning W0036: '-ZI' option is not portable) (CC78K0S warning W0037: '-ZL' option is not portable) - 如果指定了 -w2 选项，针对每个 #pragma 语句输出下列警告。 (CC78K0S warning W0849: #pragma statement is not portable) - 如果设定 -w2 针对 __asm 语句，会输出下述警告，同时执行汇编输出。 (CC78K0S warning W0850: Asm statement is not portable) - 如果指定了 -w2 选项，针对 #asm 到 #endasm 块输出下列错误。 (CC78K0S error E0801: Undefined control, etc.)
m [n] (n = 1, 2)	在静态模式中可以使用扩展说明。 多达 6 个的语句可以用整数大小描述，多达 9 个的语句可以用字符大小描述。 允许针对 1-， 2- 字节结构 / 联合体自变量和函数返回值的结构 / 联合体描述。 根据 n 值的不同 _@KREGxx 的使用方法会发生改变。如果 n 被忽略，则默认 n 值为 1。 1 : 只在 leaf 函数时将 _@KREGxx 用作共享区域。 2 : 用 _@KREGxx 执行保存 / 恢复，并将参数和自动变量分配到 _@KREGxx。
d	在函数进出库之前或之后加入例程处理。 同时指定了 -ql4 则会产生警告，然后被当作 -ql3 选项来进行处理。
r	自动添加 pascal 函数变址数。
i	将 int 和 short 描述作为 char 处理。 编译器定义的宏 _FROM_INT_TO_CHAR_ 当作是 1。
l	将长型描述当作整型。 编译器定义的宏 _FROM_LONG_TO_INT 当作是 1。

注 s、 e 和 n 不能同时设定。

[使用示例]

- 指定了 -zc 和 -zp 选项。

```
C>cc78k0s -c9024 prime.c -zpc
```

(18) 驱动器文件搜索路径

驱动器文件搜索路径 (-y)

[格式描述]

```
-y 文件夹
```

- 省略时解释

  仅一般的查找路径

[功能]

- -y 选项首先查找设定为读取设备文件的查找路径。 如果其不存在，则查找一般路径。

  一般查找路径如下。

  (i) < ..\dev > (cc78k0s.exe 的启动路径)

  (ii) CC78K0S 的启动路径

  (iii) 当前文件夹

  (iv) 环境变量 PATH 指定的路径

[应用]

- 如果设备文件没有安装在标准查找路径，而在指定的文件夹，则通过该选项设定路径。

[注意事项]

- 当使用 PM+ 时 , 把器件文件注册到 < Project Setup > 对话框中的 “Device Name:”时将定义一个文件夹。

  因此，在设定编译器选项时无需指定该选项。

[使用示例]

- 指定了 -y 选项。

```
C>cc78k0s -c9024 -ya:\tmp\dev
```

(19) 静态模型说明

静态模型说明 (-sm)

[格式描述]

```
-sm [ n ] ( n = 1-16 )
```

- 省略时解释

  普通模型 (n = 0)

[功能]

- 编译时设定 -sm 选项。 设定 -sm 选项时，目标被称为静态模型，未设定 -sm 选项时，目标被称为标准模型。
- 通常，访问静态模型的指令为 shorter 与访问栈片断的指令相比执行速度要快。 因此，可以减少目标代码并提高执行速度。
- 指定 -sm 选项，可以加快中断服务速度。 因为在静态模下并不需要使用 saddr 区域 （比如，中断函数中的寄存器变量， norec 函数中的参数 / 自动变量，运行时刻库的参数等）进行参数和变量的保存 / 返回工作，而在普通模型下才会这样执行。
- 由于多个 leaf 函数的数据共享，可以节省内存容量。

[应用]

- 如果想要提高目标执行速度，或使中断服务更快，指定 -sm 选项把普通模型切换为静态模式。

[说明]

- 通过寄存器给出所有的函数自变量，同时通过函数分配函数自变量和自动变量到静态区。
- leaf 函数分配函数自变量和自动变量依照所说明的次序从高地址到 FEFFH 和 saddr 的低地址区。 因为这个区域被所有模型的 leaf 函数共享，所以这个 saddr 区域被称为 “共用区域”。
- n 值表示共用区域的空间大小。
- 当 n = 0 或 n 省略时，公用区不存在。
- 编译器定义的宏 __STATIC__MODEL 认为其值为 1。
- sreg/__sreg 关键字可以添加到函数自变量和自动变量。 添加有 sreg/__sreg 关键字的函数自变量和自动变量分配到 saddr 区，并可以用 1 位单元操作。
- 指定 -rk 选项来将各种类型的函数参数和自动变量 (除了函数中的静态变量) 分配到 saddr 区域中，并能够支持位操作。

[注意事项]

- 由于自变量和自动变量静态锁定， recursive 函数自变量和自动变量的内容可能损坏。 当递归函数调用自身时，就会发生错误。 当一个函数调用到另一个已经被调用的函数，就没有错误发生，是因为编译器没有检测到参数和自动变量的问题。
- 如果在中断时调用的函数以中断服务的方式被调用 (中断函数或者被中断函数调用的函数)，它的参数和自动变量可能被损坏。
- 即使在中断修复期间处理的函数使用公用区，存储、返回到公用区或从公用区返回不执行。

[使用示例]

```
C>cc78k0s -c9024 test.c -sm16
```

第 6 章　C 编译器输出文件

本章描述了 CC78K0S 输出的文件。

CC78K0S 会输出下列文件。

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

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]

```
    ; 78K/0S Series C Compiler V(1)x.xx Assembler Source
    ;                                       Date: (2)xxxxx  Time: (3)xxxxx

    ; Command       : (4)-c9024 prime.c 柂 a
    ; In-file       : (5)prime.c
    ; Asm-file      : (6)prime.asm
    ; Para-file     : (7)

    $PROCESSOR ((8) 9024 )
(9)  $DEBUG
(10)$NODEBUGA
(11)$KANJICODE SJIS
(12)$TOL_INF 03FH , 0130H , 02H , 00H

(13)$DGS    FIL_NAM , .file , 033H , 0FFFEH , 03FH , 067H , 01H , 00H
                :
(14)        EXTRN   _@cprep
                :
    ; line (15)1    : (16)#define   TRUE    1
    ; line (15)2    : (16)#define   FALSE   0
    ; line (15)3    : (16)#define   SIZE    200
                :
(14)_main :
(17)$DGL    1.13
(14)        push    hl                          ; (21) [ INF ] 1 , 4
(14)        movw    ax , #08H                   ; (21) [ INF ] 3 , 6
(14)        callt   [ _@cprep ]                 ; (21) [ INF ] 1 , 8
                :
(18)??bf_main :
                :
    ; (22)*** Code Information ***
    ;
    ; (23)$FILE C:\NECTools32\Smp78k0s\CC78K0S\prime.c
    ; (24)$FUNC main ( 8 )
    ; (25)bc = ( void )
    ; (26)CODE SIZE = 222 bytes , CLOCK_SIZE = 654 clocks , STACK_SIZE = 14
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 + 2 ] )
    ; (26)CODE SIZE = 28 bytes , CLOCK_SIZE = 108 clocks , STACK_SIZE = 10
bytes
    ;
    ; (24)$FUNC putchar ( 41 )
    ; (25)      bc = ( char c : x )
    ; (26)CODE SIZE = 14 bytes , CLOCK_SIZE = 58 clocks , STACK_SIZE = 8 bytes

    ; Target chip   : (19)uPD789024
    ; Device file   : (20)Vx.xx
```

[输出项说明]

表 6-1 输出项说明 (汇编源模块文件)

条目号	说明	列宽	格式
(1)	版本号	4 (固定长度)	以 “x.yz” 格式显示
(2)	Date	11 (固定长度)	系统日期 (以 "DD Mmm YYYY" 格式显示)
(3)	时间	8 (固定长度)	系统时间 (显示格式为 "HH:MM:SS")
(4)	命令行	-	由 "CC78K0S" 输出命令行内容。 将 80 列以后的内容输出到下一行第 15 列的起始位置。 输出分号 (;) 到第 1 列。将一个以上的空字符或制表符替换成单个空字符。
(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。 如果省略文件类型，则会附加 ".c" 作为文件类型 (扩展名)。将 80 列以后的内容输出到下一行第 15 列的起始位置。 输出分号 (;) 到第 1 列。
(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。 如果省略文件类型 , 则会附加 ".asm" 作为文件类型 (扩展名)。将 80 列以后的内容输出到下一行第 15 列的起始位置。 输出分号 (;) 到第 1 列。
(7)	参数文件内容	-	输出参数文件内容。 将 80 列以后的内容输出到下一行第 15 列的起始位置。 输出分号 (;) 到第 1 列。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)	通过 -c 选项设定字符串。 参考描述器件文件的文档。
(9)	调试数据	最大值 8 (可变)	输出 DEBUG 控制。 输出 $DEBUG 或 $NODEBUG。
(10)	汇编程序的调试信息控制	9 (固定长度)	输出 NODEBUGA 控制。 输出 $NODEBUGA。
(11)	kanji 类型信息	最大值 15 (可变)	输出 Kanji 编码类型。 输出 $KANJICODE SJIS, $KANJICODE EUC, 或 $KANJICODE NONE。
(12)	工具信息	37 (固定长度)	输出工具信息、版本信息、出错信息、特定选项等 (信息以 $TOL_INF 开始)。
(13)	符号信息	-	输出符号信息 (信息以 $DGS 开始)。 只有指定了调试数据输出选项时才会输出这个信息。 即使设定 -g1 选项，也不会输出。
(14)	汇编源程序	-	输出包含编译结果的汇编程序源文件。
(15)	行号	4 (固定长度)	输出 C 源模块文件的行号采用右对齐的非零十进制数值表示。
(16)	C 源代码	-	这是输入 C 源程序映像 将 80 列以后的内容输出到下一行第 16 列的起始位置。 输出分号 (;) 到第 1 列。
(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。 即使设定 -g1 选项，也不会输出。

表 6-1 输出项说明 (汇编源模块文件)

条目号	说明	列宽	格式
(18)	符号信息创建标签	最大值 34 (可变)	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)	编译器的目标设备	最大值 15 (可变)	通过命令行选项 -c 或源文件显示指定的目标设备。
(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，时钟	-	为输出指令输出大小和时钟。(信息以 ;[INF] 开始)。
(22)	函数信息 (开始)	-	表明函数信息开始。
(23)	函数信息 (文件名)	-	含完整路径的输出目标源文件名。(信息以 ;$FILE 开始)。
(24)	函数信息 (定义函数)	-	输出函数名和用十进制码定义的行号。(信息以 ;$FUNC 开始)。
(25)	函数信息 (返回值、定义函数的参数)	-	输出定义函数的返回值寄存器和参数信息 (寄存器或堆栈位置)。
(26)	函数信息 (定义函数大小、时钟、堆栈)	-	输出定义函数所需的大小、时钟和最大堆栈使用量。 这里只显示函数本身所使用的堆栈大小。 若一个函数调用另一函数，被调用函数的堆栈大小不会添加到进行调用函数的堆栈大小。 CLOCK_SIZE 是项 (21) 中的时钟数增加的结果。
(27)	函数信息 (调用函数)	-	输出函数名和以十进制码定义的函数调用行号。(信息以 ;$CALL 开始)。

表 6-1 输出项说明 (汇编源模块文件)

条目号	说明	列宽	格式
(28)	函数信息 (调用函数返回值，自变量)	-	在函数调用过程中，输出返回值寄存器和自变量信息 (寄存器或堆栈位置)。

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]

```
/*
78K/0S Series C Compiler V (1) x.xx Error List Date : (2) xxxxx Time : (3)
xxxxx

Command         : (4) -c9024 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 ;
  *** CC78K0S 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 ) ;
  *** CC78K0S warning (9) W0745 : (10) Expected function prototype
          :
/*
 (11) Target chip : uPD789024
 (12)Device file : Vx.xx
Compilation complete, (13)1 error(s) and (14)5 warning(s) found.
*/
```

[[输出项说明]

表 6-2 输出项说明 (C 源代码的错误列表文件)

条目号	说明	列宽	格式
(1)	版本号	4 (固定长度)	以 “x.yz” 格式显示
(2)	Date	11 (固定长度)	系统日期 (以 "DD Mmm YYYY" 格式显示)
(3)	时间	8 (固定长度)	系统时间 (显示格式为 "HH:MM:SS")
(4)	命令行	-	由 "CC78K0S" 输出命令行内容。 80 列后的内容从下一行的 15 列开始输出。 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。 如果省略文件类型，则会附加 ".c" 作为文件类型 (扩展名)。 将 80 列以后的内容输出到下一行第 13 列的起始位置。
(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。 如果省略文件类型 , 则会附加 ".cer" 作为文件类型。 80 列后的内容从下一行的 15 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 15 列开始输出。 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 源代码	-	这是输入 C 源程序映像 80 列以后的内容不接在下一行。
(9)	错误信息编号	5 (固定长度)	以 "#nnnn" 格式输出出错编号。 内部错误则输出 "C"，严重错误则输出 "F"，由语法错误或编译器限制产生的错误则输出 "E"，并且若是警告则输出 "W"。 "nnnn"(出错编号) 以 4 位十进制数字表示 (不允许为 0)。
(10)	错误信息	-	参见 " 第 9 章 错误信息 "。 80 列以后的内容不接在下一行。
(11)	编译器的目标设备	最大值 15 (可变)	通过命令行选项 -c 或源文件显示指定的目标设备。
(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)	输出右对齐的非零十进制数。
(14)	警告次数	4 (固定长度)	输出右对齐的非零十进制数。

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 ) : CC78K0S warning (3) W0745 : (4) Expected function
prototype
(1) prime.c ( (2) 20 ) : CC78K0S warning (3) W0745 : (4) Expected function
prototype
(1) prime.c ( (2) 26 ) : CC78K0S warning (3) W0622 : (4) No return value
(1) prime.c ( (2) 37 ) : CC78K0S warning (3) W0622 : (4) No return value
(1) prime.c ( (2) 44 ) : CC78K0S warning (3) W0622 : (4) No return value

Target chip : (7) uPD789024
Device file : (8)Vx.xx

Compilation complete , (5) 0 error ( s ) and (6) 5 warning ( s ) found
```

[输出项说明]

表 6-3 输出项说明 (仅带有错误信息的错误列表文件)

条目号	说明	列宽	格式
(1)	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。 如果省略文件类型，则会附加 ".c" 作为文件类型 (扩展名)。
(2)	行号	5 (固定长度)	输出右对齐的非零十进制数。
(3)	错误信息编号	5 (固定长度)	以 "#nnnn" 格式输出错误信息数。 内部错误则输出 "C"，严重错误则输出 "F"，由语法错误或编译器限制产生的错误则输出 "E"，若是警告则输出 "W"。 "nnnn"(出错编号) 以 4 位十进制数字表示 (不允许为 0)。
(4)	错误信息	-	参见 " 第 9 章 错误信息 "。
(5)	错误次数	4 (固定长度)	输出右对齐的非零十进制数。
(6)	警告次数	4 (固定长度)	输出右对齐的非零十进制数。
(7)	编译器的目标设备	最大值 15 (可变)	通过命令行选项 -c 或源文件显示指定的目标设备。
(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

预处理列表文件是只包含 C 源程序预处理结果的 ASCII 映像文件。

当指定 -k 选项，可把预处理列表文件作为 C 源模块文件使用，除非 "n" 被指定为处理类型。设定 -kd 选项后，则输出 #define 扩展的列表。

[输出格式]

< 当页面宽度为 80 时 >

```
/*
78K/0S Series C Compiler V (1) x.xx Preprocess List    Date : (2) xxxxx Page :
(3) xxx

Command :       (4) -c9024  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 : uPD789024
    (11)Device file : Vx.xx
*/
```

[[输出项说明]

表 6-4 输出项说明 (预处理列表文件)

条目号	说明	列宽	格式
(1)	版本号	4 (固定长度)	以 “x.yz” 格式显示
(2)	Date	11 (固定长度)	系统日期 (以 "DD Mmm YYYY" 格式显示)
(3)	页数	4 (固定长度)	输出右对齐的非零十进制数。
(4)	命令行	-	由 "CC78K0S" 输出命令行内容。 超出该行长度的内容输出到下一行 13 列的起始位置。 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。 如果省略文件类型，则会附加 ".c" 作为文件类型 (扩展名)。超出该行长度的内容输出到下一行 13 列的起始位置。
(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。 如果省略文件类型，则附加 ".ppl" 作为文件类型 (扩展名)。 超出该行长度的内容输出到下一行 13 列的起始位置。
(7)	参数文件内容	-	输出参数文件内容。 超出该行长度的内容输出到下一行 13 列的起始位置。 输出分号 “;” 到第 1 列。将一个以上的空字符或制表符替换成单个空字符。

表 6-4 输出项说明 (预处理列表文件)

条目号	说明	列宽	格式
(8)	行号	5 (固定长度)	输出右对齐的非零十进制数。
(9)	C 源代码	-	这输入的是 C 源代码。 超出该行长度的内容输出到下一行 9 列的起始位置。
(10)	编译器的目标设备	最大值 15 (可变)	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。
(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

交叉引用列表文件中包含标识符列表，例如声明、定义、引用函数和变量。 其中也包含其它信息，例如属性和行数。 它们以编译器编译的顺序输出。

[输出格式]

< 当页面宽度为 80 时 >

```
78K/0S Series C Compiler V (1) x.xx Cross reference List
                                                Date : (2) xxxxx Page : (3) xxx

Command         : (4) -c9024  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                array    mark          5            14      16      22
   EXTERN                 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 : uPD789024
   (16) Device file : Vx.xx */
```

[[输出项说明]

表 6-5 输出项说明 (交叉引用列表文件)

条目号	说明	列宽	格式
(1)	版本号	4	以 “x.yz” 格式显示
(2)	Date	11 (固定长度)	系统日期 (以 "DD Mmm YYYY" 格式显示)
(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	-	由 "CC78K0S" 输出命令行内容。 超出该行长度的内容输出到下一行 13 列的起始位置。 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。 如果省略文件类型，则会附加 ".c" 作为文件类型 (扩展名)。超出该行长度的内容输出到下一行 13 列的起始位置。
(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。 如果省略文件类型，则附加 ".xrf" 作为文件类型。 超出该行长度的内容输出到下一行 13 列的起始位置。
(7)	参数文件内容	-	输出参数文件内容。 超出该行长度的内容输出到下一行 13 列的起始位置。 一个以上的空白字符或制表符都用一个空白字符表示。
(8)	Include 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。 "n" 表示起始于 "1" 的数字 , 以此来标示包含文件号。 超出该行长度的内容输出到下一行 13 列的起始位置。 在没有包含文件时，则该行不被输出。
(9)	符号属性	6 (固定长度)	显示符号的属性。 用 EXTERN 来标识外部变量， EXSTC 来标识外部静态变量， INSTC 来标识内部静态变量， AUTOnn 来标识自动变量， REGnn 来标识寄存器变量 (用 nn 来表示数值的范围，其数值从 "1" 开始)，用 EXTYP 来标识外部 typedef 声明， INTYP 来标识内部 typedef 声明，以 LABEL 来标识标签，以 TAG 来标识结构体或共用体，以 MEMBER 标识成员，并以 PARAM 来标识函数参数。
(10)	符号可用属性	6 (固定长度)	显示符号可用属性 (左对齐)。 用 CONST 来标识常量，VLT 来标识变量， CALLT 标识调用函数， CALLF 来标识被调用函数， NOAUTO 来标识非自动函数， NOREC 来标识无记录函数，SREG 标识 sreg- 位变量，RWSFR 标识专用寄存器变量， ROSFR 标识只读的专用寄存器变量， WOSFR 来标识只写的专用寄存器变量， VECT 来标识中断功能和 BANK 作为库功能。
(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、长整型和域型。 把 "u" 加到 unsigned 型的开头。 附加类型包括空类型、浮点型、双精度型、长精度 (long double) 型、函数型、数组型、指针型、结构体型、共用体型、枚举型、位 , inter, 和宏替换指令 #define。
(12)	符号名称	15 (固定长度)	如果符号名称超过 15 个字符并处于一行中，则名称原样输出。 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 23 列输出， (13) 项和 (14) 项从下一行 39 列输出。

表 6-5 输出项说明 (交叉引用列表文件)

条目号	说明	列宽	格式
(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。 如果行内容超过行的长度，余下内容输出到下一行第 47 列的起始位置。
(15)	编译器的目标设备	最大值 15 (可变)	通过命令行选项 -c 或源文件显示指定的目标设备。
(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　有效使用 C 编译器

这章介绍了有效使用 CC78K0S 的方法。

7.1　高效操作 （EXIT Status Function）

当编译结束时， CC78K0S 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。

Ends normally :	0
WARNING :	0
FATAL ERROR :	1
ABORT :	2

如果没有使用 PM+， CC78K0S 在命令行中启动，可使用批处理文件中的状态进一步提高操作效率。

[使用示例]

```
cc78k0s -c9024 %1
IF ERRORLEVEL 1 GOTO ERR
cc78k0s -c9024 %2
IF ERRORLEVEL 1 GOTO ERR
GOTO EXIT
ERR
echo Some error found.
EXIT
```

[说明]

- 当 C 源程序编译通过 %1 时，一个严重错误发生。 实质上，在错误信息输出后处理继续进行。 但是因为到了 EXIT 状态返回了 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）

CC78K0S 支持下面的环境变量。

PATH :	搜索可执行工程文件的路径
INC78K0S :	搜索 include files 路径
TMP :	搜索临时文件路径
LANG78K :	T 汉字编码的类型 （可通过 -ZE, -ZS, 或 -ZN 选项来指定）
	(euc : EUC 码 , sjis : 移位 JIS 码 , none : 无 2 字节码)
LIB78K0S :	搜索库路径

[使用示例]

< 当使用 DOS 提示符时 >

```
;AUTOEXEC.BAT
PATH C:\NECTools32\bin ; C:\bat ; C:\cc78k0s ; C:\tool
VERIFY ON
BREAK ON
SET INC78K0S = C:\NECTools32\inc78k0s
SET LIB78K0S = C:\NECTools32\lib78k0s
SET TMP = C:\tmp
SET LANG78K = sjis
```

[说明]

- 按 C:\NECTools32\bin, C:\bat, C:\cc78k0s, C:\tool 的路径顺序来搜索可执行文件。
- 从 C:\NECTools32\inc78k0s 目录下搜索包含文件。

  如果没有设置 , 搜索从 C:\NECTools32\inc78k0s 完成 (如果 CC78K0S 是安装在 C:\NECTools32)。
- 在连接时从 C:\NECTools32\lib78k0s 搜索库文件。

  如果没有设置 , 搜索从 C:\NECTools32\lib78k0s 完成 (如果 CC78K0S 是安装在 C:\NECTools32)。
- 临时文件存放在 c:\tmp 目录下。
- 移动 JIS 码的使用和 Kanji 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 [Run] 菜单下的 [Stop build] 或点击工具栏上的 [Stop Build] 按钮。 当在 PM+ 中建立时，命令键的输入无效。

第 8 章　启动程序

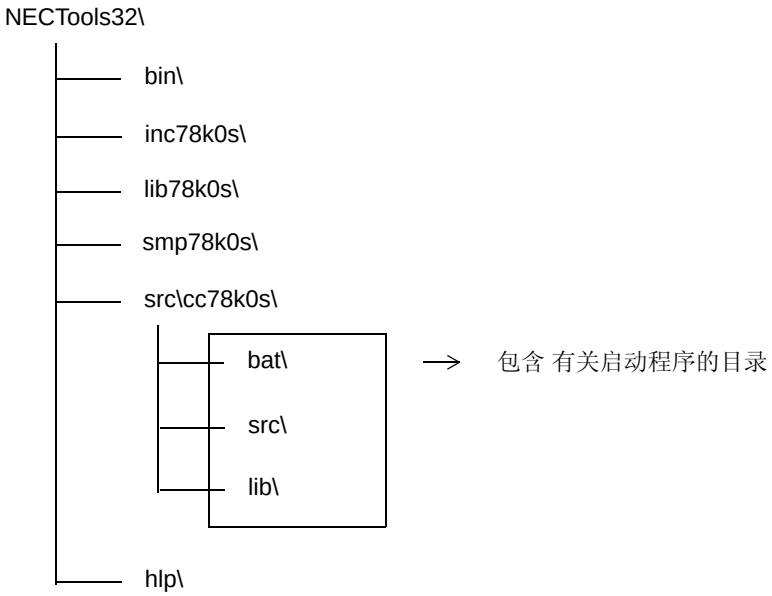
为了执行 C 源程序，在系统和用户程序 （主函数）中，程序需要 ROM 化过程 来激活。 这个程序叫做启动例程。

为了执行用户编写的程序，必须为程序创建一个启动例程。 CC78K0S 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。即使用户没有对预处理执行过程进行描述，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\cc78k0s 文件夹中。



在 src\cc78k0s 目录下的内容显示如下。

lib 目录包括启动例程的目标文件和汇编后的源程序库。 此目标文件可以和任何使用 78K0S 系列目标设备的程序相连接。 如果不需要特别的修正，可以链接系统提供的未经修改的样例目标文件。 如果执行了 CC78K0S 提供的 mkstup.bat ，这个目标文件可以重写。

文件内容见 "2.6.4 库文件 "。

8.1.1　bat 文件夹内容

这个文件夹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。

bat 文件夹中的器件文件 (d9026.78k) 不是用来开发的，是为了更新库等启动批处理文件时使用。 因此，实际开发时需要其它对应的器件文件。

表 8-1 bat 文件夹内容

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	用于 updating _putchar.asm 的批处理文件
repselo.bat	用于更新 updating setjmp.asm 和 longjmp.asm 的批处理文件 (保存到编译保留区) 注 2
repselon.bat	用于更新 updating setjmp.asm 和 longjmp.asm 的批处理文件 (不保存到编译保留区) 注 2
repcmul.bat	用于更新运行时间库和 char 类型乘法 (带有乘法器的器件使用) 的批文件
repimul.bat	用于增加或删除运行时间库和 char 类型乘法 (带有乘法器的器件使用) 的批文件
replmul.bat	用于增加或删除运行时间库和 long 类型乘法 (带有乘法器的器件使用) 的批文件

注 1。 由于 ROM 化例程在库中，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），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 文件夹内容

src 文件夹包括启动例程的汇编源程序、 ROM 例程，错误处理例程和标准库函数 （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 文件夹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 文件夹内容

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	ROM 化例程的源文件
_putchar.asm	_putchar 功能
putchar.asm	putchar 功能
getchar.asm	getchar 功能
longjmp.asm	longjmp 功能
setjmp.asm	setjmp 功能
xcmul.asm	@@csmul, @@cumul function (使用乘法器)
ximul.asm	@@ismul, @@iumul function (使用乘法器)
xlmul.asm	@@lsmul, @@lumul function (使用乘法器)
def.inc	根据类型来设置库
macro.inc	每个典型模式的宏定义

注 带有 "n" 文件名是不包含标准库处理的启动例程。 只有在未使用标准库时才使用文件名带 n 的例程。

8.2 批处理文件说明

8.2.1 生成启动例程的批处理文件

在 bat 目录中的 mkstup.bat 是用来创建启动例程的目标文件。

在 RA78K0S 汇编包中的编译器是 mkstup.bat 所需要的。 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 的 src\cc78k0s\bat 目录中执行如下命令行。

```
mkstup 设备类型注
```

注　　请参阅器件设备文件相关文档。

[使用示例]

- 将要创建的启动例程使用的目标设备是 uPD789024。

```
mkstup 9024
```

mkstup.bat 批处理文件以改写的启动例程目标文件的格式存储在 lib 目录中，如显示下面的 bat 同一级目录 LIB 目录下。

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\ ——— s0s.rel
         s0sl.rel
         s0ss.rel
         s0ssl.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 源程序而作的准备工作。 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

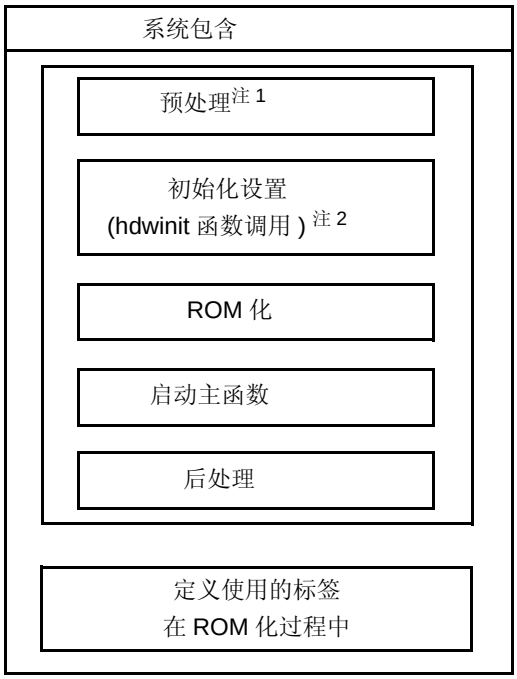
存储器初始化，包含在系统中的 ROM 化和 C 源程序的开始进程和终止进程都会进行。

ROM 化： 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 ROM 化。 当程序被写入 ROM 后，可以由微处理器来调用执行。

(2) 配置

图 8-1 显示了与启动例程相关序和它们的配置情况。

图 8-1 启动例程简介



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’ 的文件进行标准库相关处理。 文件名末尾加 ‘n’ 的文件不进行处理。

注 2。 hdwinit 函数是当用户需要对外围设备 （src）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）。 如果用户没有创建 hdwinit 函数，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。

表 8-3 显示了 cstart.asm 和 cstartn.asm 之间的区别。

表 8-3 启动例程源代码之间的区别

启动例程的类型	使用库处理
cstart.asm	Yes
cstartn.asm	No

(3) （3）启动例程的使用

表 8-4 列举了 CC78K0S 为启动例程提供的目标文件名称。

表 8-4 源文件和目标文件之间的相似处

文件类型	源文件	目标文件
启动程序	cstart*.asm 注 1	s0s*.rel 注 2
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. *：如果使用了标准程序库中的一个固定区域，需要加上 “l” }

rom.asm 中定义了标签，用来指示 ROM 化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。

备注　　调用 cstart 时需要在前面加 _@，即格式为 _@cstart。

(1) （1）预处理

cstart.asm 中的预处理在 (1) 至 (6) 中说明 (如下所示)。

[cstart.asm preprocessing]

```
        NAME    @cstart

$INCLUDE ( def.inc )                                          ; (1)
$INCLUDE ( macro.inc )
                                                              ; (2)
                                                              ; (3)
BRKSW     EQU 1 ; brk , sbrk , calloc , free , malloc , realloc function use
EXITSW    EQU 1   ; exit , atexitfunction use
$_IF (_STATIC)
RANDSW    EQU 0   ; rand , srandfunction use
DIVSW     EQU 0   ; divfunction use
LDIVSW    EQU 0   ; 1divfunction use
FLOATSW   EQU   0     ; floating point variables use
$ELSE
RANDSW    EQU 1   ; rand , srandfunction use
DIVSW     EQU 1   ; divfunction use
LDIVSW    EQU 1   ; 1divfunction use
FLOATSW   EQU   1     ; floating point variables use
$ENDIF
STRTOKSW  EQU 1   ; strtokfunction use

PUBLIC  _@cstart , _@cend

$_IF ( BRKSW )
PUBLIC  _@BRKADR, _@MEMTOP, _@MEMBTM
   :
$ENDIF
EXTRN   _main , _@STBEG , _hdwinit                            ; (4)
$_IF ( EXITSW )
EXTRN   _exit
$ENDIF
                                                              ; (5)
EXTRN   _?R_INIT , _?R_INIS , _?DATA , _?DATS
                                                              ; (6)
@@DATA  DSEG    UNITP
$_IF ( EXITSW )
_@FNCTBL :      DS      2 * 32
_@FNCENT :      DS    2
                  :
_@MEMTOP :      DS    32
_@MEMBTM :
$ENDIF
```

(1) 包含 include 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

- 用于堆栈分析的公共符号 (_@STBEG) 是外部引用声明。 _@STBEG 的值是堆栈区域的最终地址 +1。
- _@STBEG 通过在连接程序中为堆栈图形分辨率指定符号生成选项 （-s）自动生成。 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。但是创建一个链接命令文件 （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。
- 下述是一个链接命令文件的示例。连接命令文件是一个文本文件，可以由用户在普通编辑器中创建（关于描述方法的细节，请参阅 RA78K0S 汇编器程序包操作用户手册）。

[在连接中指定栈 STACK 的例子]

创建 lk78k0s.dr (链接命令文件)。 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。 关于链接命令，请参考 smp78k0s\cc78k0s 文件夹下的 lk78k0s.dr 文件。

```
             First address   Size
                        ↓       ↓
memory SDR      : ( 0FE20h , 00098h )
memory  STACK   : ( xxxxh , xxxh )  <-- Specify the first address and size
                                        here, then specify lk78k0s.dr by
                                        the -d linker option.
                                        (Example: -dlk78k0s.dr)
merge @@INIS    : = SDR
merge @@DATS    : = SDR
merge @@BITS    : = SDR
```

(5) ROM 化处理标签的外部引用声明

ROMization 处理所需的标号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) （2）初始化设置

cstart.asm 中的初始设置在 (1) 至 (4) 中说明。

[cstart.asm 中的初始化设置]

```
                                                                    ; (1)
@@VECT00        CSEG    AT      0
                DW      _@cstart

@LCODE          CSEG
_@cstart :
                                                                    ; (2)
                MOVW    AX , #_@STBEG   ; SP <-stack begin address
                MOVW    SP , AX
                CALL    !_hdwinit                                   ; (3)
$ENDIF
                  :
$_IF ( BRKSW OR EXITSW OR RANDSW OR FLOATSW )
                MOVW    AX , #0
$ENDIF
                  :
```

(1) 复位向量设定

复位向量表段的定义如下。 设置启动例程的起始地址

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

(2) 堆栈指针 (SP) 设置

将 _@STBEG 存入堆栈指针。

_@STBEG 通过在连接程序中为堆栈图形分辨率指定符号生成选项 （-s）自动生成。

(3) 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，初始化设置就有可能达成用户目标。

如果用户没有创建 hdwinit 函数，不做任何处理就直接返回。

(3) ROM 化处理

下面描述 cstart.asm 中的 ROM 化。

[ROM 化处理]

```
; ************************************
;ROM 数据复制
; ************************************
; copy external variables having initial value
        MOVW    HL , #_@R_INIT
        MOVW    DE , #_@INIT
LINIT1 :
        MOVW    AX, HL
        CMPW    AX , #_?R_INIT
        BZ      $LINIT2
        MOV     A , [ HL ]
        MOV     [DE], A
        INCW    HL
        INCW    DE
        BR      $LINIT1
LINIT2 :
        MOVW    HL , #_@DATA
; copy external variables which doesn't have initial value
LDATA1:
            :
```

在 ROMization 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。 涉及的变量有四种类型 （a）到 （d），示例如下：

示例

```
char    c = 1 ;             (a) External variable with initial value
int     i ;                 (b) External variable without initial value[Note]
__sreg  int     si = 0 ;    (c) sreg variable with initial value
__sreg  char    sc ;        (d) sreg variable without initial value[Note]

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。

- 图 8-2 显示了 (a) 有初值的外部变量的 ROM 化处理过程。
  变量 （a）的初值被编译器放在 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定义为 _@R_INIT 和 _?R_INIT。 @@INIT 段的首标签和末标签分别定义为 _@INIT 和 _?INIT。
- 变量 (b) 和 (d) 未被复制，但零直接由 RAM 放置到段中 (见表 8-6)。表 8-5 和表 8-6 显示了 ROM 和 RAM 中放置变量 (a) 至 (d) 的段名称以及每个段中第一个和最后一个初始值标签。

图 8-2 ROM 化处理

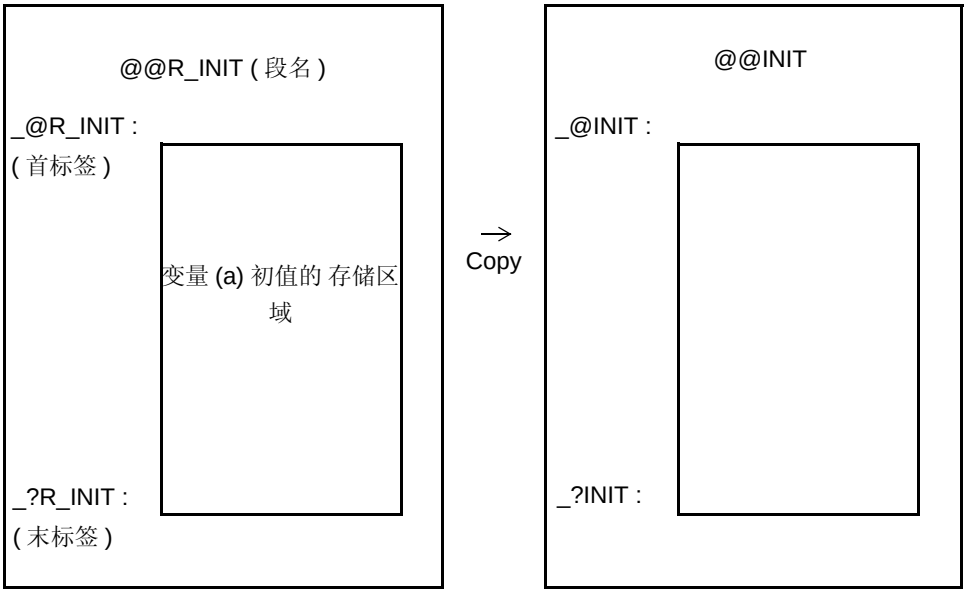


表 8-5 初始值的 ROM 区域

变量类型	段	首标签	首标签
有初值的外部变量 (a)	@@R_INIT	_@R_INIT	_?R_INIT
有初值的 sreg 变量 （c)	@@R_INIS	_@R_INIS	_?R_INIS

表 8-6 初始值在 RAM 中的区域 （复制的目标地址）

变量类型	段	首标签	首标签
有初值的外部变量 (a)	@@INIT	_@INIT	_?INIT
无初值的外部变量 (b)	@@DATA	_@DATA	_?DATA
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

在 cstart.asm 中开始主函数和后处理的内容 (1) 至 (3) 中说明。

[启动主函数和后处理]

```
        CALL    !_main          ; main ( ) ;            ; (1)
$_IF ( EXITSW )
        MOVW    AX , #0
        CALL    !_exit          ; exit ( 0 ) ;          ; (2)
$ENDIF
        BR      $$
;
_@cend :
                                                        ; (3)
@@R_INIT        CSEG    UNITP
_@R_INIT :
@@R_INIS        CSEG
_@R_INIS :
@@INIT          DSEG
_@INIT :
@@DATA          DSEG
_@DATA :
@@INIS          DSEG    SADDRP
_@INIS :
@@DATS          DSEG    SADDRP
_@DATS :
@@CALT          CSEG    CALLT0
@@CNST          CSEG
@@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 ROM 化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 ROM 化处理中使用的段和标签 （见 "(3) ROM 化处理 "）。 段是指存放每个变量的初始值的区域。 标签是每个段中的首地址。

下面对 ROM 化处理文件 rom.asm 说明。 rom.asm 中可重定位的文件在库中。

```
        NAME    @rom
;
        PUBLIC  _?R_INIT , _?R_INIS
        PUBLIC  _?INIT , _?DATA , _?INIS , _?DATS
;
@@R_INIT        CSEG                                    ; (1)
_?R_INIT :
@@R_INIS        CSEG    UNITP
_?R_INIS :
@@INIT          DSEG
_?INIT :
@@DATA          DSEG
_?DATA :
@@INIS          DSEG    SADDRP
_?INIS :
@@DATS          DSEG    SADDRP
_?DATS :
;
        END
```

(1) 定义 ROM 化处理中使用的标签

定义了 （a）到 （d）每个变量在 ROM 化处理中使用的标签 （见 "(3) ROM 化处理 "）。 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 CC78K0S 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) （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\cc78k0s\bat 目录下的 mkstup.bat 对修改的源程序文件 （cstart*.asm） (*: 字母数字符号）。

- 标准库函数中使用的符号

  如果没有使用表 8-7 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。

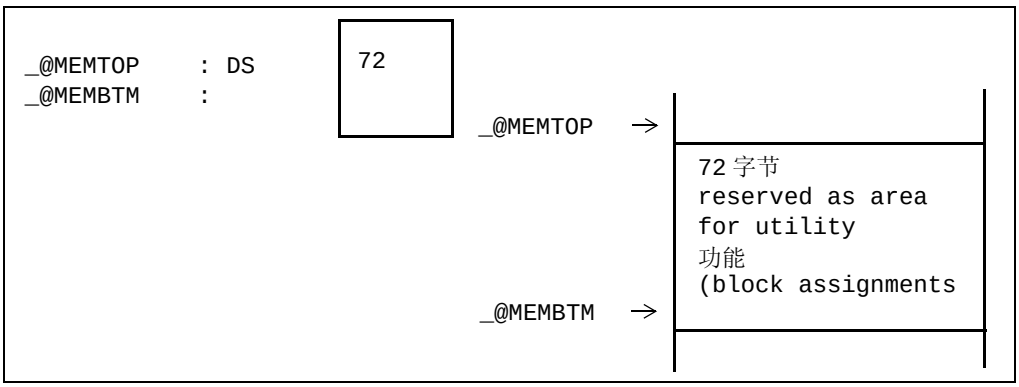
表 8-7 库函数中使用的符号

程序库函数名	使用的符号
brk sbrk strtol strtoul 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 浮点数运行时间库	_errno

- 用于效用函数 (块分配 / 发布) 的区域

  如果用户定义了效用函数 (块分配 / 发布) 使用区域的大小，在下例中对此说明。

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



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，需要减小指定空间，或通过修改连接命令文件来避免错误，如下所示。 关于修复链接命令文件，请参考 "(2) 链接命令文件 "。

例　　减小指定大小

(2) 链接命令文件

本节阐述如何创建连接命令文件。 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 （连接命令的详细使用方法敬请参阅 RA78K0S 汇编程序包用户操作手册）。

- CC78K0S 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。 特别的，该 40 字节的区域（FED8H 至 FEFFH）用于普通模型。 静态模式由 -sm[n] 选项指定， saddr 区域的一部分空间 [FEF0H 至 FEFFH] 用于公共区域。

  （普通模型）

  (a) 运行时间库参数 [FED8H 至 FEFFH]

  (b) norec 函数的参数或自动变量 [FEF8H 至 FEF7H]

  (c) -qr 选项指定时的寄存器变量 [FED8H 至 FEF7H]

  (d) 标准库任务 （区域 （b）和 （c）的一部分）。

- 如果用户没有使用标准库，则不使用此区域。

  （静态模型）

  (a) 公共区域 [FEF0H 至 FEFFH]

下面是使用连接命令文件 (lk78k0s.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。

< lk78k0s.dr >

```
              First address Size
memory RAM : ( 0FB00h , 00320h )  -> Make this size larger.
memory SDR : ( 0FE20h , 00098h )     (also change the first address if
necessary)
merge @@INIS : = SDR              -> Specifies the location of the segment.
merge @@DATS : = SDR              -> Specifies the location of the segment.
merge @@BITS : = SDR              -> Specifies the location of the segment.
```

如果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 CC78K0S C 编译器语言用户手册）。

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。

第 9 章　错误信息

本章解释由 CC78K0S 输出的错误信息的原因。

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

示例

prime.c (8) :　　CC78K0S error E0712 : Declaration syntax
prime.c (8) :　　CC78K0S error E0301 : Syntax error
prime.c (8) :　　CC78K0S error E0701 : External definition syntax
prime.c (19) :　　CC78K0S warning W0745 : Expected function prototype

但是 , 内部错误 C0101、 C0103 和 C0104 使用以下输出格式。

```
[ xxx.c < yyy > zzz ] CC78K0S error C0101 : Internal error
[ xxx.c < yyy > zzz ] CC78K0S error C0103 : Intermediate file error
[ xxx.c < yyy > zzz ] CC78K0S error 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。

(1) 命令行错误信息

(2) 内部错误和存储器错误信息

(3) 字符错误信息

(4) 配置元素错误信息

(5) 转换时的错误信息

(6) 表达式的错误信息

(7) 语句的错误信息

(8) 声明和函数定义的错误信息

(9) 预处理命令的错误信息

(10) 严重错误文件的输入 / 输出和在非法操作系统上运行的错误信息

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。

C/F/E/Wnnnn

C： 内部错误

如果溢出，则编译停止。 不会输出目标模块文件和汇编源程序文件。

F： 致命错误

如果发生这样的错误，则编译停止。 不会输出目标模块文件和汇编源程序文件。

E： 语法错误或编译限制引起的错误。

当错误发生超过指定数字，则编译停止。 不会输出目标模块文件和汇编源程序文件。

W： 警告

继续编辑。

nnnn (4- 位 数字)

编号从 0001 开始命令行错误信息

编号从 0101 开始 内部错误和存储器错误信息

编号从 201 开始 字符错误信息

编号从 0301 开始 配置元素错误信息

编号从 0401 开始 转换错误信息

编号从 0501 开始 表达式的错误信息

编号从 0601 开始 语句错误信息

编号从 0701 开始 声明或函数定义错误信息

编号从 0801 开始 预处理命令的错误信息

编号从 0901 开始 致命的文件 I/O 或运行非法操作系统的错误信息

9.3.1 命令行错误信息

表 9-1 命令行错误信息 < 编号从 0001 开始 >

错误 编号	错误信息	
F0001	消息	Missing input file
	原因	没有指定输入源文件名。
	解决方法	输出 "Please enter 'cc78k0s--' if you want help message"。 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	消息	Too many input files
	原因	指定了多个输入源文件名。
	解决方法	输出 "Please enter 'cc78k0s--' if you want help message"。 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	消息	Unrecognized string
	原因	在交互命令行中指定的对象不是能够正确识别的选项。
F0004	消息	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	消息	Illegal file specification
	原因	指定了非法的文件名。
F0006	消息	File not found
	原因	指定的输入文件不存在。
F0007	消息	Input file specification overlapped file name
	原因	指定的输入文件名有重名。
F0008	消息	File specification conflicted file name
	原因	指定的 I/O 文件名有重名。
F0009	消息	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，所以不能创建。
F0010	消息	Directory not found
	原因	输出文件名中指定的驱动或目录不存在。
F0011	消息	Illegal path
	原因	在路径参数设置选项中指定了非法路径名。
F0012	消息	Missing parameter 'option'
	原因	没有指定所需要的参数文件。
	解决方法	输出 "Please enter 'cc78k0s--' if you want help message"。 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 < 编号从 0001 开始 >

错误 编号	错误信息	
F0013	消息	Parameter not needed 'option'
	原因	指定了不必要的选项参数。
	解决方法	输出 "Please enter 'cc78k0s--' if you want help message"。 使用 --, -?，或 -h 选项来访问帮助文件，同时请输入正确的参数。
F0014	消息	Out of range 'option'
	原因	对选项指定的参数值超出了允许范围。
	解决方法	输出 "Please enter 'cc78k0s--' if you want help message"。 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	消息	Parameter is too long
	原因	在选项参数中的字符数超出了限制范围。
F0016	消息	Illegal parameter 'option'
	原因	在选项参数中有语法错误。
	解决方法	输出 "Please enter 'cc78k0s--' if you want help message"。 使用 --, -?，或 -h 选项来访问帮助文件，同时请输入正确的选项。
F0017	消息	Too many parameters
	原因	选项参数的总数超出限制。
F0018	消息	Option is not recognized 'option'
	原因	指定了不正确的选项。
	解决方法	输出 "Please enter 'cc78k0s--' if you want help message"。 使用 --, -?，或 -h 选项来访问帮助文件，同时请输入正确的选项。
F0019	消息	Parameter file nested
	原因	在参数文件中指定了 -f 选项。
	解决方法	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	消息	Parameter file read error
	原因	参数文件读取失败。
F0021	消息	Memory allocation failed
	原因	存储器分配失败
W0022	消息	Same category option specified ñ ignored 'option'
	原因	指定了多个相同冲突的选项。
	编译器	最后指定的选项是有效的，并继续进行处理。
W0023	消息	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不同。
	编译器	以命令行中指定的设备类型为准。
F0024	消息	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 < 编号从 0001 开始 >

错误 编号	错误信息	
W0029	消息	‘-QC’ option is not portable
	原因	-qc 选项不符合 ANSI 标准规范 (关于 -qc 的细节，敬请参阅 " 第 5 章 编译器选项 ")。
W0031	消息	'-ZP' option is not portable
	原因	-zp 选项不符合 ANSI 标准规范 (关于 -zp 的细节，敬请参阅 " 第 5 章 编译器选项 ")。
W0032	消息	'-ZC' option is not portable
	原因	-zc 选项不符合 ANSI 标准规范 (关于 -zc 的细节，敬请参阅 " 第 5 章 编译器选项 ")。
F0033	消息	Same category option specified 'option'
	原因	指定了多个相同冲突的选项。
	解决方法	输出 "Please enter 'cc78k0s--' if you want help message"。 使用 --， -? 或 -h 枫选项来访问帮助文件，并对输入进行改正。
W0036	消息	'-ZI' option is not portable
	原因	-zi 选项不符合 ANSI 标准规范 (关于 -zi 的细节，敬请参阅 " 第 5 章 编译器选项 ")。
W0037	消息	'-ZL' option is not portable
	原因	-zl 选项不符合 ANSI 标准规范 (关于 -zl 的细节，敬请参阅 " 第 5 章 编译器选项 ")。
W0038	消息	'-ZI' option specified - regarded as '-QC'
	原因	指定的 -zi 选项把整型和短型当作字符型来处理，所以整型扩展控制优化选项 -qc 就有效。
W0039	消息	'-SM' option specified - regarded as '-ZL'
	原因	因为指定了静态模式选项 -sm，所以把长整型当作整型处理的 -zl 选项自动有效。
W0040	消息	'-RK' option required '-SM' - ignored '-RK'
	原因	只有指定了静态模式选项 -sm 时，本地变量优化选项 -rk 才会有效。 忽略 -rk 选项。
W0041	消息	'-SM' option specified - ignored '-QR'
	原因	因为指定了静态模式选项 -sm，忽略寄存器优化选项 -qr。
W0045	消息	'-SM' option specified - ignored '-ZR'
	原因	因为指定了静态模式选项 -sm，将忽略 pasca 函数接口说明选项 -zr。
W0052	消息	'-ZD' option specified - regarded as '-QL3'
	原因	指定了支持序言 / 结尾的 (-zd) 选项，因此 -QL4 选项被当作 -ql3 选项用来处理标准代码模式库转换选项 (-ql)。

表 9-1 命令行错误信息 < 编号从 0001 开始 >

错误 编号	错误信息	
W0055	消息	'-ZM' option required '-SM' - ignored '-ZM'
	原因	只有指定了静态模式选项 -sm 时，静态模式扩展说明选项 (-zm) 才会有效。忽略 -zm 选项。
W0065	消息	'-QW3' option deleted - regarded as '-QW2'
	原因	由于删除了优化设定选项 -qw3， -qw2 变为有效。
W0066	消息	'-QW5' option deleted - regarded as '-QW4'
	原因	由于删除了优化设定选项 -qw， -qw 变为有效。

9.3.2 内部错误和存储器错误信息

表 9-2 内部错误和存储器错误信息 < 编号从 0101 开始 >

错误 编号	错误信息	
C0101	消息	Internal error
	原因	内部错误发生。
	解决方法	请联系技术支持。
E0102	消息	Too many errors
	原因	严重错误的数量超过了 30。
	编译器	处理继续进行，但不输出后续的错误信息。 先前的错误可能会引起更多的错误。 因此首先请修改之前发生的错误。
C0103	消息	Intermediate file error
	原因	媒介文件包含错误。
	解决方法	请联系技术支持。
C0104	消息	Illegal use of register
	原因	使用寄存器的方法不正确。
E0105	消息	Register overflow : simplify expression
	原因	表达式过于复杂，使得没有多余寄存器可用。
	解决方法	对引起错误的复杂表达式进行简化。
C0106	消息	Stack overflow 'overflow cause'
	原因	堆栈溢出。 可能是栈或堆发生溢出。
	解决方法	请联系技术支持。
E0108	消息	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。
	解决方法	减少变量，使其所占的区域不超过 64 KB。
E0109	消息	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。
	解决方法	减少参数的使用，使其所占的区域不超过 64 KB。
E0110	消息	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	消息	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。
E0113	消息	Compiler limit: too many local labels
	原因	在一个函数中定义的本地标签数量超过了处理限制。
	解决方法	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 < 编号从 0201 开始 >

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 < 编号从 0301 开始 >

错误 编号	错误信息	
E0301	信息	Syntax error
	原因	语法错误发生。
	对策	请检查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 字母。
	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。
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。

表 9-4 配置元素错误信息 < 编号从 0301 开始 >

错误 编号	错误信息	
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。
W0330	信息	'-ZI' option specified-int & short are treated as char in this file
	原因	-zi 选项被指定。文件中的 int 和 short 类型被当作 char 类型进行处理。
W0331	信息	'-ZL' option specified-long is treated as int in this file
	原因	'-ZL' 选项被指定。文件中的 Long 类型被当作 int 类型来进行处理。

表 9-4 配置元素错误信息 < 编号从 0301 开始 >

错误 编号	错误信息	
W0332	信息	Non-supported keyword found-ignored 'function attributes' in this file
	原因	发现无法支持的关键词。本文件中的函数属性被忽略。
W0333	信息	'-SM' option specified-ignored 'function attributes' keyword in this file
	原因	指定静态模型说明选项 -sm。 Function attributes in this file are ignored.
E0334	信息	'-SM' option specified-float & double keywords are not allowed
	原因	因为指定了静态模式选项 -sm，不允许使用 Float 和 double 关键字。
W0335	信息	'-SM' option specified-long constant is treated as int constant
	原因	因为指定了静态模式选项 -sm， long 常量被当作 int 常量处理 。
W0339	信息	'__temp' required '-SM -ZM' -ignored '__temp' in this file
	原因	只有静态模式选项 (-sm) 和静态模式扩展说明选项 (-zm) 都被指定时，临时变量关键字 __temp 才可以使用。 本文件中 __temp 关键字被忽略。
W0340	信息	Unreferenced label 'label name'
	原因	已经定义的标签从来没有被引用过。

9.3.5 转换时的错误信息

表 9-5 转换时的错误信息 < 从 0401 开始 >

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	尝试向唯读属性的 sfr 中写入数值。
E0408	信息	Cannot read for write/only sfr 'SFR name'
	原因	尝试从唯写属性的 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。

9.3.6 表达式的错误信息

表 9-6 表达式的错误信息 < 从 0501 开始 >

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	在数组的索引中只允许整型表达式。
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，或者贺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同体的指针，但这个表达式是一个结构体或共同体的名称。
	对策	确保出现在 '->' 运算符之前的表达式是指向结构体或共同体的指针。

表 9-6 表达式的错误信息 < 从 0501 开始 >

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同体的名称，但这个表达式是一个指向结构体或共同体的指针。
	对策	确保出现在 "." 运算符之前的表达式是结构体或共同体的名称。
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	两个结构体或共同体无法进行比较。
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	'&' on constant
	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。

表 9-6 表达式的错误信息 < 从 0501 开始 >

错误 编号	错误信息	
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。
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，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	只有整数可以被和指针相加。
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。
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。

表 9-6 表达式的错误信息 < 从 0501 开始 >

错误 编号	错误信息	
E0549	信息	'-SM' option specified-recursive function is not allowed
	原因	指定静态模型说明选项 -sm。 不允许使用递归函数。

9.3.7 语句的错误信息

表 9-7 语句的错误信息 < 从 0601 开始 >

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 swich 语句中出现多个 default 语句。
E0610	信息	Compiler limit : block nest level too depth
	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 语句的错误信息 < 从 0601 开始 >

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 < 从 0701 开始 >

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	callt 函数的声明太多。 最多可声明 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。
W0708	信息	Already declared symbol 'variable name'
	原因	相同的 “变量名称” 符号已经被声明过。
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范。
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。
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。 关于可用数量，请参考 CC78K0S C 编译器语言用户手册。
E0715	信息	Too many sreg variables
	原因	sreg 变量的声明太多。

表 9-8 声明和函数定义的错误信息 < 从 0701 开始 >

错误 编号	错误信息	
E0716	信息	Not allowed automatic data in noauto function
	原因	在 noauto 函数中不能使用自动变量。
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。 关于可用数量，请参考 CC78K0S C 编译器语言用户手册。
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。 关于可用数量，请参考 CC78K0S C 编译器语言用户手册。
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同体的成员。
E0727	信息	Unknown size structure / union 'name'
	原因	结构体或共同体的大小容量未定义。
E0728	信息	Compiler limit : too many structure / union members
	原因	结构体或共同体中的成员数量超过 256。
E0729	信息	Compiler limit : structure / union nesting
	原因	结构体或共同体的嵌套层数超过 15。

表 9-8 声明和函数定义的错误信息 < 从 0701 开始 >

错误 编号	错误信息	
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数型的。
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范。
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。
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。

表 9-8 声明和函数定义的错误信息 < 从 0701 开始 >

错误 编号	错误信息	
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范。
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。
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。

表 9-8 声明和函数定义的错误信息 < 从 0701 开始 >

错误 编号	错误信息	
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。
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 / callf / noauto / norec / __banked / __pascal are not allowed for interrupt function
	原因	中断函数不能被声明 callf, noauto, norec, __banked 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。
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。
E0781	信息	'-SM' option specified-variable parameters are not allowed
	原因	指定静态模型说明选项 -sm。 不允许使用变量参数。
E0782	信息	'-SM' option specified-structure & union parameter is not allowed
	原因	指定静态模型说明选项 -sm。 不允许使用结构和联合体参数。
E0783	信息	'-SM' option specified-structure & union return value is not allowed
	原因	指定静态模型说明选项 -sm。 不允许使用结构和联合体返回值。
E0784	信息	'-SM' option specified-too many parameters of function
	原因	指定静态模型说明选项 -sm。 函数参数超出了限制的 3 个参数 /6 字节。
E0785	信息	'-SM' option specified-expected function prototype
	原因	指定静态模型说明选项 -sm。 缺少函数原型声明。
W0786	信息	'-SM' option specified-undeclared parameter in function prototype
	原因	指定静态模型说明选项 -sm。 参数在函数原型声明中未做声明。

表 9-8 声明和函数定义的错误信息 < 从 0701 开始 >

错误 编号	错误信息	
W0787	信息	Bit field type is char
	原因	字符类型被指定为位域类型。
W0792	信息	Undeclared parameter in __pascal function definition or prototype
	原因	__pascal 函数定义或原型声明中没有对应的参数声明。如果没有参数，必须定义成 void 类型。
W0793	信息	Variable parameters are not allowed for __pascal function - ignored __pascal
	原因	对于 __pascal 函数不能被指定变量参数。 __pascal 关键字被忽略。
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 < 从 0801 开始 >

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 INC78K0S 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。
W0816	信息	Redefined macro name 'macro name'
	原因	宏名称有重定义。

表 9-9 预处理命令的错误信息 < 从 0801 开始 >

错误 编号	错误信息	
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 < 从 0801 开始 >

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	有未识别的 #pragma VECT ‘字符串’或 #pragma 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 EAFWORK 说明被指定给同一个中断函数。
W0849	信息	#pragma statement is not portable
	原因	#pragma 语句不符合 ANSI 标准规范。

表 9-9 预处理命令的错误信息 < 从 0801 开始 >

错误 编号	错误信息	
W0850	信息	Asm statement is not portable
	原因	ASM 语句不符合 ANSI 标准规范。
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。
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。
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，先执行数据对齐。
E0899	信息	Character string specified by #error is output
	原因	指定了 #error 字符串。

9.3.10 严重错误文件的输入 / 输出和在非法操作系统上运行的错误信息

表 9-10 严重错误文件的输入 / 输出和在非法操作系统上运行的错误信息 < 从 0901 开始 >

错误 编号	错误信息	
F0901	信息	File I/O error
	原因	在文件输入 / 输出过程中被产生物理 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。 这个路径可以使用杜选项来指定。 参照 "5.4 (18) 驱动器文件搜索路径 " 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	在输出临时文件时被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文件夹区域。
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。 1. 在 EOF 之后输入继续。 2. 临时文件无法创建，因为驱动器上没有足够的目录区。
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。 请联系技术支持。

表 9-10 严重错误文件的输入 / 输出和在非法操作系统上运行的错误信息 < 从 0901 开始 >

错误 编号	错误信息	
F0913	信息	Requires MS-DOS V2.11 or greater
	原因	操作系统不是 MS-DOS (2.11 版本 或更新的版本)。
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 block 或 __asm 语句。 这个函数没有进行跳转优化。 执行 W0837 响应。
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。

附录 A　示例程序

本章介绍了用于 CC78K0S 的示例程序。

A.1　C 源模块文件

```
#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 ( s , i )
char    *s ;
int     i ;
{
        int     j ;
        char    *ss ;
        j = i ;
        ss = s ;
}

putchar ( c )
char    c ;
{
        char    d ;
        d = c ;
}
```

A.2 执行示例

```
C>cc78K0S -c9024 prime.c -a 栈 -x -e -ng
```

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

sample\prime.c ( 18 ) : CC78K0S warning W0745 : Expected function prototype
sample\prime.c ( 20 ) : CC78K0S warning W0745 : Expected function prototype
sample\prime.c ( 26 ) : CC78K0S warning W0622 : No return value
sample\prime.c ( 37 ) : CC78K0S warning W0622 : No return value
sample\prime.c ( 44 ) : CC78K0S warning W0622 : No return value

Target chip : uPD789024
Device file : Vx.xx

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

A.3 输出列表

A.3.1 汇编源模块文件

```
; 78K/0S Series C Compiler Vx.xx Assembler Source
;                                    Date : xx xxx xxxx Time : xx : xx : xx
; Command       : -c9024 prime.c -a -p -x -e -ng
; In-file       : prime.c
; Asm-file      : prime.asm
; Para-file     :

$PROCESSOR      ( 9024 )
$NODEBUG
$NODEBUGA
$KANJICODE SJIS
$TOL_INF        03FH , 0130H , 02H , 00H

        EXTRN   _@cprep
        EXTRN   _@RTARG0
        EXTRN   @@isrem
        PUBLIC  _mark
        PUBLIC  _main
        PUBLIC  _printf
        PUBLIC  _putchar

@@CNST    CSEG
L0011 : DB      ' %6d '
        DB      00H
L0017 : DB      0AH
        DB      ' %d primes found. '
        DB      00H

@@DATA          DSEG
_mark : DS      ( 201 )

; line  5
; line  8

@@CODE CSEG
_main :
        push    hl                              ; [ INF ] 1 , 4
        movw    ax , #08H                               ; [ INF ] 3 , 6
        callt   [ _@cprep ]                             ; [ INF ] 1 , 8
; line  11
        mov     a , a                                   ; [ INF ] 2 , 4
        mov     [ hl ] , a              ; count         ; [ INF ] 1 , 6
        mov     [ hl + 1 ] , a          ; count         ; [ INF ] 2 , 6
; line  13
        mov     [ hl + 6 ] , a          ; i             ; [ INF ] 2 , 8
        mov     [ hl + 7 ] , a          ; i             ; [ INF ] 2 , 8
L0003 :
        mov     a , [ hl + 6 ]          ; i             ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 7 ]          ; i             ; [ INF ] 2 , 6
        xor     a , #080H               ; 128           ; [ INF ] 2 , 4
        cmpw    ax , #080C8H            ; -32568        ; [ INF ] 3 , 6
        bc      $$ + 4                  ; [ INF ] 2 , 6
        bnz     $L0004                                  ; [ INF ] 2 , 6
```

```
; line  14
        xor     a , #080H       ; 128           ; [ INF ] 2 , 4
        addw    ax , #_mark                             ; [ INF ] 3 , 6
        movw    de , ax                                 ; [ INF ] 1 , 4
        mov     a , #01H                ; 1             ; [ INF ] 3 , 6
        mov     [ de ] , a                              ; [ INF ] 1 , 6
        mov     a , [ hl + 6 ]          ; i             ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 7 ] , a      ; i             ; [ INF ] 2 , 6
        incw    ax                                      ; [ INF ] 1 , 4
        mov     [ hl + 7 ] , a          ; i             ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     [ hl + 6 ] , a          ; i             ; [ INF ] 2 , 6
        br      $L0003                                  ; [ INF ] 2 , 6
L0004 :
; line  15
        xor     a , a                                   ; [ INF ] 2 , 4
        mov     [ hl + 6 ] , a          ; i             ; [ INF ] 2 , 6
        mov     [ hl + 7 ] , a          ; i             ; [ INF ] 2 , 6
L0006 :
        mov     a , [ hl + 6 ]          ; i             ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 7 ]          ; i             ; [ INF ] 2 , 6
        xor     a , #080H       ; 128           ; [ INF ] 2 , 4
        cmpw    ax , #080C8H            ; 200           ; [ INF ] 3 , 6
        bc      $$ + 7                  ; [ INF ] 2 , 6
        bz      $$ + 5                                  ; [ INF ] 2 , 6
        br      !L0007                                  ; [ INF ] 3 , 6
; line  16
        xor     a , #080H       ; 128           ; [ INF ] 2 , 4
        addw    ax , #_mark                             ; [ INF ] 3 , 6
        movw    de , ax                                 ; [ INF ] 1 , 4
        mov     a , [ de ]                              ; [ INF ] 1 , 6
        cmp     a , #00H                ; 0             ; [ INF ] 2 , 4
        bz      $L0015v                                 ; [ INF ] 2 , 6
; line  17
        mov     a , [ hl + 6 ]          ; i             ; [ INF ] 2 , 8
        rolc    a , 1                                   ; [ INF ] 1 , 2
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 7 ]          ; i             ; [ INF ] 2 , 6
        rolc    a , 1                                   ; [ INF ] 1 , 2
        addw    ax , #03H               ; 3             ; [ INF ] 3 , 6
        mov     [ hl + 5 ] , a          ; prime         ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     [ hl + 4 ] , a          ; prime         ; [ INF ] 2 , 6
; line  18
        xch     a , x                                   ; [ INF ] 1 , 4
        push    ax                      ; [ INF ] 1 , 4
        movw    ax , #L0011                             ; [ INF ] 3 , 6
        cal     !_printf                                ; [ INF ] 3 , 6
        pop     ax                                      ; [ INF ] 1 , 6
; line  19
        mov     a , [ hl ]              ; count         ; [ INF ] 1 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 1 ]          ; count         ; [ INF ] 2 , 6
        incw    ax                                      ; [ INF ] 1 , 4
        mov     [ hl + 1 ] , a          ; count         ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     [ hl ] , a              ; count         ; [ INF ] 1 , 6
```

```
; line  19
        mov     a , [ hl ]              ; count         ; [ INF ] 1 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 1 ]          ; count         ; [ INF ] 2 , 6
        incw    ax                                      ; [ INF ] 1 , 4
        mov     [ hl + 1 ] , a          ; count         ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     [ hl ] , a              ; count         ; [ INF ] 1 , 6
; line  20
        xch     a , x                                   ; [ INF ] 1 , 4
        movw    _@RTARG0 , ax                           ; [ INF ] 2 , 8
        movw    ax , #08H               ; 8             ; [ INF ] 3 , 6
        call    !@@isrem                                ; [ INF ] 3 , 6
        or      a , x                                   ; [ INF ] 2 , 4
        bnz     $L0012                                  ; [ INF ] 2 , 6
        movw    ax , #0AH               ; 10            ; [ INF ] 3 , 6
        call    !_putchar                               ; [ INF ] 3 , 6
L0012 :
; line  21
        mov     a , [ hl + 6 ]          ; i             ; [ INF ] 2 , 6
        add     a , [ hl + 4 ]          ; prime         ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 7 ]          ; i             ; [ INF ] 2 , 6
        addc    a , [ hl + 5 ]          ; prime         ; [ INF ] 2 , 6
        mov     [ hl + 3 ] , a          ; k             ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     [ hl + 2 ] , a          ; k             ; [ INF ] 2 , 6
L0014 :
        mov     a , [ hl + 2 ]          ; k             ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 3 ]          ; k             ; [ INF ] 2 , 6
        xor     a , #080H               ; 128           ; [ INF ] 2 , 4
        cmpw    ax , #080C8H            ; 200           ; [ INF ] 3 , 6
        bc      $$ + 4                  ; [ INF ] 2 , 6
        bnz     $L0015                                  ; [ INF ] 2 , 6
; line  22
        xor     a , #080H               ; 128           ; [ INF ] 2 , 4
        addw    ax , #_mark                             ; [ INF ] 3 , 6
        movw    de , ax                                 ; [ INF ] 1 , 4
        xor     a , a                   ; 128           ; [ INF ] 2 , 4
        mov     [ de ] , a                              ; [ INF ] 1 , 6
        mov     a , [ hl + 2 ]          ; k             ; [ INF ] 2 , 6
        add     a , [ hl + 4 ]          ; prime         ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 3 ]          ; k             ; [ INF ] 2 , 6
        addc    a , [ hl + 5 ]          ; prime         ; [ INF ] 2 , 6
        mov     [ hl + 3 ] , a          ; k             ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     [ hl + 2 ] , a          ; k             ; [ INF ] 2 , 6
        br      $L0014                                  ; [ INF ] 2 , 6
L0015 :
; line  24
        mov     a , [ hl + 6 ]          ; i             ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 7 ]          ; i             ; [ INF ] 2 , 6
        incw    ax                                      ; [ INF ] 1 , 4
        mov     [ hl + 7 ] , a          ; i             ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     [ hl + 6 ] , a          ; i             ; [ INF ] 2 , 6
        br      !L0006                                  ; [ INF ] 3 , 6
```

```
L0007 :
; line  25
        mov     a , [ hl ]              ; count         ; [ INF ] 1 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 1 ]          ; count         ; [ INF ] 2 , 6
        push    ax                      ; [ INF ] 1 , 4
        movw    ax , #L0017                             ; [ INF ] 3 , 6
        call    !_printf                                ; [ INF ] 3 , 6
        pop     ax                                      ; [ INF ] 1 , 6
; line  26
        movw    ax , #08H                               ; [ INF ] 3 , 6
        callt   [ _@cdisp ]                             ; [ INF ] 1 , 8
        pop     hl                                      ; [ INF ] 1 , 6
        ret                                             ; [ INF ] 1 , 6
; line  31
_printf :
        push    hl                      ; [ INF ] 1 , 4
        push    ax                      ; [ INF ] 1 , 4
        movw    ax , #04H                               ; [ INF ] 3 , 6
        callt   [ _@cprep ]                             ; [ INF ] 1 , 8
; line  35
        mov     a , [ hl + 10 ]          ; i            ; [ INF ] 2 , 6
        mov     [ hl + 2 ] , a          ; j             ; [ INF ] 2 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 11 ]          ; i            ; [ INF ] 2 , 6
        mov     [ hl + 3 ] , a          ; j             ; [ INF ] 2 , 6
; line  36
        mov     a , [ hl + 4 ]          ; s             ; [ INF ] 2 , 6
        mov     [ hl ] , a              ; ss            ; [ INF ] 1 , 6
        xch     a , x                                   ; [ INF ] 1 , 4
        mov     a , [ hl + 5 ]          ; s             ; [ INF ] 2 , 6
        mov     [ hl + 1 ] , a          ; ss            ; [ INF ] 2 , 6
; line  37
        pop     ax                                      ; [ INF ] 1 , 6
        pop     ax                                      ; [ INF ] 1 , 6
        pop     ax                                      ; [ INF ] 1 , 6
        pop     hl                                      ; [ INF ] 1 , 6
        ret                                             ; [ INF ] 1 , 6
; line  41
_putchar :
        push    hl                      ; [ INF ] 1 , 4
        push    ax                      ; [ INF ] 1 , 4
        movw    ax , #02H                               ; [ INF ] 3 , 6
        callt   [ _@cprep ]                             ; [ INF ] 1 , 8
; line  43
        mov     a , [ hl + 2 ]          ; c             ; [ INF ] 2 , 6
        mov     [ hl + 1 ] , a          ; d             ; [ INF ] 2 , 6
; line  44
        pop     ax                                      ; [ INF ] 1 , 6
        pop     ax                                      ; [ INF ] 1 , 6

        pop     hl                                      ; [ INF ] 1 , 6
        ret                                             ; [ INF ] 1 , 6
        END
```

```
; *** Code Information ***
;
; $FILE C:\NECTools32\sample\prime.c
;
; $FUNC main ( 8 )
;       bc = ( void )
;   CODE SIZE = 218 bytes , CLOCK_SIZE = 654 clocks , STACK_SIZE = 14 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 ( 31 )
;       bc = ( pointer s : ax , int i : [ sp + 2 ] )
;   CODE SIZE = 28 bytes , CLOCK_SIZE = 108 clocks , STACK_SIZE = 10 bytes
;
; $FUNC putchar(41)
;       bc = ( char c : x )
;       CODE SIZE = 14 bytes , CLOCK_SIZE = 58 clocks , STACK_SIZE = 8 bytes

; Target chip : uPD789024
; Device file : Vx.xx
```

A.3.2 预处理列表文件

```
/*
78K/0S Series C Compiler Vx.xx Preprocess List     Date : xx xxx xxxx Page : 1
Command         : -c9024 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 :   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 ( s , i )
        29 :  char    *s ;
        30 :  int     i ;
        31 :  {
        32 :      int     j ;
        33 :      char    *ss ;
        34 :
        35 :      j = i ;
        36 :      ss = s ;
        37 :  }
        38 :
        39 :  putchar ( c )
        40 :  char    c ;
        41 :  {
        42 :      char    d ;
        43 :      d = c ;
        44 :  }


/*
Target chip : uPD789024
Device file : Vx.xx
*/
```

A.3.3 交叉引用列表文件

```
78K/0S Series C Compiler Vx.xx Cross reference List Date : XX XXX XXXX Page : 1

Command         : -c9024 prime.c -x
In-file   : prime.c
Xref-file : prime.xrf
Para-file:

ATTRIB  MODIFY  TYPE    SYMBOL  DEFINE  REFERENCE

EXTERN          array   mark    5       14  16  22
EXTERN                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          func    printf  28      18  25
EXTERN          func    putchar 39      20
PARAM           pointer s       29      36
PARAM           int     i       30      35
AUTO1           int     j       32      35
AUTO1           pointer ss      33      36
PARAM           char    c       40      43
AUTO1           char    d       42      43
                #define TRUE    1       14
                #define FALSE   2       22
                #define SIZE    3       5   13  15  21


Target chip : uPD789024
Device file : Vx.xx
```

A.3.4 错误列表文件

```
PRIME.C ( 18 ) : CC78K0S warning W0745 : Expected function prototype
PRIME.C ( 20 ) : CC78K0S warning W0745 : Expected function prototype
PRIME.C ( 26 ) : CC78K0S warning W0622 : No return value
PRIME.C ( 37 ) : CC78K0S warning W0622 : No return value
PRIME.C ( 44 ) : CC78K0S warning W0622 : No return value


Target chip : uPD789024
Device file : Vx.xx
Compilation complete, 0 error(s) and 5 warning(s) found.
```

附录 B　相关使用注意事项列表

本章说明了使用 CC78K0S 的相关注意事项。

表 B-1　相关使用注意事项列表

编号	注意事项
1	[关于指定选项的注意事项] - 当某选项不允许重复指定多项参数却被赋予了多个参数时，以最后的设定参数为准 （有效）。 - 不能省略 -c 选项后的类型设定。 如果省略，将产生异常中止错误。 如果没有设定 -c 选项，确保在 C 源文件中输入 “#pragma pc (类型)” 替代。 在编译过程中，如果设定的选项与在 C 源文件中不同，则指定的选项优先。 此时会输出警告消息。 - 如果设定了帮助选项，则忽略所有其它选项。
2	[关于文件输出目录的注意事项] 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] 若在文件中发现语法错误，将在文件名之后附加错误信息。 若在禁止区域指定设备文件，将只输出指定的字符串。 其他情况下，必须附加驱动、路径和文件扩展名。
4	[源文件名的相关注意事项] 在 CC78K0S 中，源文件名中除去扩展名外的部分 （主名）默认为模块名。因此，对源文件名有些限制。 - 关于文件名的长度，在主操作系统允许的范围内配置主文件名中的主名和扩展名，并使用点（.）作为主名和扩展名的分割符。 当使用 PM+ 时，用点（.）把主名和扩展名分开，使用 “.c” 或 “.C” 作为 C 源代码的扩展名。 - 主文件名和扩展名能使用的字符包含主操作系统允许的字符，除圆括号 （ ( ) ）、分号 （；）以及逗号 （，）。 注意连字符 （-）不能用作文件名及文件名的首字符使用。当使用 PM+ 时，指定路径名称时不能用空格或包括诸如中文字符的双字节字符。 - 在参数文件中 , 井字符 (#) 不能用作文件名和路径名。 - 当文件主名的前 8 个字符重名时，链接文件时会输出错误。 - 若使用 ID78K0S-NS 或 SM78K0S 时，可用于文件名的字符为小写字母 （a 至 z）、大写字母 （A 至 Z）、数字 （0 至 9）、下划线 （_）和点 （.）
5	[关于包含文件的注意事项] 在 include 文件中不能定义函数 （不包括声明），则在 C 源代码中展开该函数。 在 include 文件中定义的内容，在源代码调试中可能出现定义行不能正确显示的情况。

表 B-1 相关使用注意事项列表

编号	注意事项
6	[汇编源程序输出的相关注意事项] 当 C 源程序包含使用汇编语言 （例如 #asm 块或 __asm 声明）的描述时，加载模块文件的创建顺序是编译、汇编然后链接。 假设存在汇编语言的说明符，当由 C 编译器首先输出汇编器源程序编译而不直接输出目标文件时，需遵守以下几点注意事项。 - 如果符号需要在 #asm 块 （#asm 和 #endasm 之间的部分）和 __asm 语句中定义时，使用 8 位或更少的字节且以字符串 ?L@ 开头（例如：?L@01、?L@sym 等）。 然而，这些符号不能从外部进行定义（PUBLIC 声明）。 不能在 #asm 块和 __asm 语句中定义段。 如果符号不是以字符串 ?L 开头的话，汇编时将输出严重错误 （F2114）。 - 当使用 C 源代码中 #asm 块以外的变量时，如果在其他 C 描述中不存在引用，那么 EXTRN 不会产生，并且输出链接错误。因此，如果在 C 源代码中无任何引用时，在 #asm 块中执行 EXTRN。 - 如果 C 源文件包含 #asm 块和 __asm 声明的 , 指定 -a 或 -sa 编译器选项来启用汇编描述，并汇编输出汇编源文件。 当使用 PM+ 时，可通过独立选项规范指定 -a/-sa 选项用于仅输出汇编程序文件，或通过通用选项规范指定 -a/-sa 选项。 - 当使用 PM+ 时且指定汇编器选项 -a 或 -sa 时，无论汇编器选项 -o/-no 如何， RA78K0S 将开始。 - 当以 #pragma 片段命令修改段名时，不要把段名称定义为与源文件主名相同的名称。 否则，在汇编中会输出错误 （F2106）。
7	[可用汇编程序包] 由于支持长文件名，因此若 RA78K0S 的版本早于 Ver.1.30 的话，可能会产生错误。
8	[使用网络时的注意事项] 当位于文件系统的用于创建临时文件的文件夹在网络上共享时，根据所使用的网络软件类型，文件争用可能上升并导致异常操作。 通过设定选项和环境变量来避免这种竞争。 在网络环境中，当使用 PM+ 时，不要使用 CC78K0S。

表 B-1 相关使用注意事项列表

<table>
<tr><th>编号</th><th>注意事项</th></tr>
<tr><td>9</td><td>[创建链接命令文件]
当使用目标设备 ROM 或 RAM 之外的区域、当编译器创建链接对象或当希望把代码或数据存放于任意指定地址时，在链接时需建立链接命令文件并定义 -d 选项。
关于建立链接命令文件的信息，请参考 RA78K0S 汇编程序包操作手册并在编译器上安装 lk78k0s.dr（位于 smp78k0s 文件夹内）。

例 若希望把不带初始值的外部变量（sreg 变量除外）从某个 C 源代码文件中放入外部存储器。

(1) 在 C 源代码起始部分为不带初始值的外部变量更片段名。
<pre>#pragma section @@DATA EXTDATA
:</pre>注意 必须通过改变启动程序对改变的段进行初始化和 ROM 化。

(2) 建立链接命令文件。
< lk78k0s.dr >
<pre>memory EXTRAM : (0F000h , 00200h)
merge EXTDATA : = EXTRAM</pre>
当创建链接命令文件时，需注意以下几点。

(1) 在链接时，当堆栈符号使用 -s 自动生成选项时，建议通过链接命令文件中的存储器命令对堆栈区域进行保护，并明确地指定其名称。 若省略区域名称，将作为 RAM 中的堆栈区域使用（SFR 区域除外）。

例 当添加链接命令文件 lk78k0s.dr 时
<pre>memory EXTRAM : (0F000h , 00200h)
memory STK : (0FB00H , 20H)
merge EXTDATA : = EXTRAM</pre>
(Command line)
<pre>> lk78k0s s0sl.rel prime.rel -bcl0s.lib -sstk ñdlk78k0s.dr</pre>
(2) 当在已定义的存储器区域内进行链接时，可能会输出下列错误。
<pre>" *** RA78K0S error E3206: Segment 'xxx' can't allocate to memory-ignored. "</pre>
[原因]
由于已定义的存储器区域内空间不足，描述段无法定位。
[对策]
解决方法大致可分为下列 3 个步骤。

(i) 查询未被存放段的大小（查看 .map 文件）。
(ii) 在步骤 (i) 查询段大小的基础上，在命令文件中增加放置段的区域大小。
(iii) 指定命令文件选项 -d 和链接。

但是，如果在步骤 (i) 内对于段的查询出错的话，根据段类型查询段大小的方法如下所示。
- 当段是在编译时自动生成的
通过已链接和创建的映射文件查询段的大小
- 当段是由用户创建时
查询未列于汇编列表文件（.prn）上的段的大小。</td></tr>
<tr><td>10</td><td>[使用 va_start 宏时的注意事项]
定义在 stdarg.h 中的 va_start 宏无法得到保证（因为根据函数的不同，第一个参数的偏移量也会不同）。
- 当指定首个参数时，使用 va_starttop
- 当指定第二个和子参数时，使用 va_start macro。</td></tr>
</table>

表 B-1 相关使用注意事项列表

编号	注意事项
11	[当引用特殊功能寄存器 （SFR）的常量地址时的注意事项] 若常量地址引用 16 位 SFR 时，则使用 SFR 名称来引用，因为产生了一个访问 8 位单元的非法代码。
12	[启动例程和库] - 使用与执行表中文件 （cc78k0s.exe 或 cc78k0s）相同版本的启动例程和库。 - (b) 针对浮点数支持的函数 sprintf、 vprintf 和 vsprintf，若由格式转换符 "%f"、 "%e"、 "%E"、 "%g" 或 "%G" 转换的结果小于精度时，舍去该值。 即使由 "%g"/"%G" 指定的转换结果大于精度， "%f" 转换也会执行。 对于函数 sscanf 和 scanf，如果在由格式转换符 "%f"、 "%e"、 "%E"、 "%g" 或 "%G" 指定的转换时未读取有效的字符，则转换结果为 +0。 如果转换结果为 "±" 的话，则把 ±0 作为转换结果。 [预防措施] 无
13	[当进行带有 ID78K0S-NS 的源代码调试时的注意事项] 当调用 pascal 函数时， Next 命令与 Step 命令操作方法相同。 通过 Return 命令等返回函数的调用端。当指定编译选项 -zr 时，所有函数均变为 pascal 函数。 因此，不要执行 Next 命令。
14	[当进行带有 SM78K0S 的源代码调试时的注意事项] 当调用 pascal 函数时，不要执行 Next 命令。 否则，可能发生程序跑飞。 当指定编译选项 -zr 时，所有函数均变为 pascal 函数。 因此，在指定 -zr 时绝对不能执行 Next 命令。

表 B-1 相关使用注意事项列表

编号	注意事项

15	[当进行 ROM 化时]
	ROM 化就是放置初始值，诸如那些在 ROM 中有初始值的外部变量，然后在系统运行中把这些值复制到 RAM 中。 在 CC78K0S 中，会默认产生用于 ROM 化的代码。 因此，在链接中有必要在包含 ROM 化的启动例程中进行链接。
	下列的启动例程由 C 编译器提供，且全部带有 ROM 化进程。
	如果使用闪存存储器的自重写模式，请参考表 8-4。
	启动例程：
	(1) 当不使用 C 标准库区域时：s0s.rel
	(2) 当使用 C 标准库区域时：s0sl.rel
	
	[使用示例]
	`C:>lk78k0s s0s.rel sample.rel -s -bcl0s.lib -osample.lmf`
	
	`sample.rel :` 用户程序的目标模块文件
	`s0s.rel :` 启动例程
	`cl0s.lib :` 运行时间库和标准库
	-s 选项为堆栈符号 （_@STBEG 和 _@STEND）自动生成的选项。
	
	注意事项
	- 一定要在开始时链接启动程序。
	- 当创建库时，应当独立于 CC78K0S 提供的库，并且在链接时优先于编译器的库。
	- 不要向 CC78K0S 库添加用户函数。
	- 当使用浮点数库 （cl0s*f.lib）时，有必要把带有 ROM 化进程的启动例程链接至标准库和浮点数库。
	
	当 using sprintf, sscanf, printf, scanf, vprintf, 和 vsprin 支持浮点功能
	示例
	`-bmylib.lib -bcl0sf.lib -bcl0s.lib`
	
	当 sprintf, sscanf, printf, scanf, vprintf 和 vsprintf 不支持浮点功能时
	示例
	`-bmylib.lib -bcl0s.lib -bcl0sf.lib`

16	[产生堆栈区域的符号 （-s）] 在 CC78K0S 中，用户不能保留堆栈区域。 为了获得一个栈区，要在链接过程中设定选项 （-s）。 使用 PM+ 时，在设定的源文件中包含 C 源代码的话，则自动附加 -s 选项。
17	[ROM 代码] 当要求获得 ROM 代码时，指定 -r 或 -u 目标转换器选项，如朽和 -u0FFH （不要取消规范）。 示例 `-r -u0FFH` `-r :` 以地址顺序对 HEX 文件内容进行排序 `-u fill value :` 把指定的值填入 ROM 代码中的空白区域。
18	[帮助说明选项] 在 PM+ 中，用于显示选项说明的编译器选项 --、 -? 和 -h 被忽略。 若需要帮助，请点击每个工具对话框的 < Option Setup > 中的 [Help] 按钮。

表 B-1 相关使用注意事项列表

<table>
<tr><th>编号</th><th>注意事项</th></tr>
<tr><td>19</td><td>[-ll 选项说明]
当使用 PM+ 时， -ll 选项可指定的最大数为 32767。若指定的数超出 32767 的话，需采用其他选项指定 -ll。</td></tr>
<tr><td>20</td><td>[关于符号名称长度时的注意事项]
当使用 ID78K0S-NS Ver.1.01 和 SM78K0S Ver.1.42 或更早期的版本时，不要使用超过 127 字符的符号名称。</td></tr>
<tr><td>21</td><td>[使用 PM+ 时的注意事项]
(a) 用户建立的参数文件
当把 PM+ 指定用于用户创建的参数文件时，载入参数文件的内容由 PM+ 建立。 当创建参数文件时，需注意以下几点。 否则，在执行时会产生错误。
- 指定与 PM+ 创建的参数文件同名的文件。
- 不要描述设备类型说明选项 （-c）、设备文件搜索路径说明选项 （-y）和源文件。
- 用户创建的参数文件中的选项不会进行有效性检查。
(b) < 汇编器选项 > 对话框
不要定义 -c、 -f 和 -y 选项以及源文件，否则在执行时会产生错误。
由于不对 < 汇编器选项 > 对话框中的选项进行有效性检查，因此如果描述出错的话，将在执行时产生错误。
(c) 包含文件相关度
在 include 文件的依赖关系检查时，使用 PM+ 建立编译文件时，诸如 #if 的条件语句将被忽略。因此，建立时不需要的 include 文件会被误认为需要的文件。 若描述为注释或字符串时，它们会被正确判断而不具备依赖关系。
示例
<pre>#if   0
#include   " header1.h "   /* Dependence relationship judged */
                                /* to exist */
#else   / * ! zero */
#include   " header2.h "
#endif
/*
#include   " header3.h "
*/</pre>在依赖关系检查中判断 header1.h 为建立时所需的文件。如果存在 header1.h 文件，则 header1.h 注册到 PM+ 的 "ProjectWindow" 中。

[预防措施] 无。 但是，这对建立进程无效。
(d) 项目相关文件的设定
编译器属性启动例程和标准库可从 PM+ 的 [Project] 菜单或由右键点击 Project 窗口显示的 "Add Project-Related File" 进行添加和删除。
在 << Startup Routine >>tab （位于 < Compiler Options > 对话框）中进行汇编器属性的启动例程和标准库设定。</td></tr>
<tr><td>22</td><td>[原型声明相关的注意事项]
若函数原型声明中不含有函数类型说明的话，将产生错误 （E0301 和 E0701）。
示例
<pre>f (void) ;   /* E0301 : Syntax error */
                /* E0701 : External definition syntax */</pre>[预防措施] 描述函数类型。
示例
<pre>int   f (void) ;</pre></td></tr>
</table>

表 B-1 相关使用注意事项列表

<table>
<tr><th>编号</th><th>注意事项</th></tr>
<tr><td>23</td><td>[错误信息相关的注意事项]
除了函数，如果在命令行的开始出现关键字的拼写错误的话，错误行的显示位置可能偏移或可能输出不正确的错误信息。
示例
<pre>
extren int i ; /* extern spelling error. 这里不会产生错误。 */
/* comment */
void   f (void) ;
[EOF] /* Error such as E0712 */
</pre>[预防措施] 无</td></tr>
<tr><td>24</td><td>[预处理命令中注释描述相关的注意事项]
在描述预处理命令中，当注释作为函数类型宏与预处理命令在同一行中描述时，将产生错误（E0803、 E0814、 E0821 等）。
示例
<pre>
/* 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 */
</pre>[预防措施] 预处理命令之后的注释。
示例
<pre>
#pragma sfr   /* com1 */
#define ONE   1   /* com2 */
#define TWO   2   /* com3 */
#ifdef ANSI_C   /* com4 */

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

表 B-1 相关使用注意事项列表

编号	注意事项

25	[与用于结构、 union 或 enum 的标记相关的注意事项]
	如果标记 （用于 structure、 union 或 enum）在函数原型声明中定义前被使用时，如果满足下列条件（a），将产生警告；如果满足下列条件 （b），将出现错误。
	(a) 若标记用于定义参数声明和指向结构或 union 的指针时，调用函数将产生警告 （W0510）。
	示例
	`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 ) ;`
	`}`
	(b) 若标记用于参数声明的返回值类型声明，并指定结构、 union 或 enum 时，将产生错误（E0737）。
	示例
	`/* E0737 Undeclared structure/union/enum tag */`
	`void func1 ( int , struct st ) ;`
	`/* E0737 Undeclared structure/union/enum tag */`
	`struct st func2 ( int ) ;`
	`struct st {`
	`char memb1 ;`
	`char memb2 ;`
	`} ;`
	
	[预防措施] 预先定义结构、 union 或 enum。

26	[在函数中初始化阵列、结构或 union 的相关注意事项]
	使用静态变量地址、常量或字符串的阵列、结构和 union 不能进行初始化。
	示例
	`void f ( void );`
	`void f ( void ) {`
	`char *p, *p1, *p2 ;`
	`char *ca[3] = { p, p1, p2 }; /* Error( E0750 ) */`
	`}`
	
	[预防措施] 描述并使用分配语句，以取代初始化。
	示例
	`void f ( void );`
	`void f ( void ) {`
	`char *ca[3] ;`
	`char *p, *p1, *p2 ;`
	`ca [ 0 ] = p ; ca [ 1 ] = p1 ; ca [ 2 ] = p2 ;`
	`}`

表 B-1 相关使用注意事项列表

编号	注意事项

27	[外部 callt 函数相关的注意事项]
	如果引用初始函数表中的外部 callt 函数地址并且在相同的模块中调用函数，汇编列表无效和在汇编期间发生出错。
	示例
	`callt extern void funca ( void ) ;`
	`callt extern void funcb ( void ) ;`
	`callt extern void funcc ( void ) ;`
	
	`static void ( * const func [ ] ) ( ) = {`
	`funca, funcb, funcc`
	`} ;`
	`callf void func2 ( void ) {`
	`funcc ( );`
	`funcb ( );`
	`funca ( );`
	`}`
	
	[预防措施] 分离函数表和函数调用模块。

28	[与函数返回结构相关的注意事项]
	当函数返回结构时，在返回一个返回值的过程中将产生中断。如果在中断进程中出现相同函数的调用时，在中断进程结束后返回值为非法值。
	示例
	`struct str {`
	`char c ;`
	`int i ;`
	`long l ;`
	`} st ;`
	
	`struct str func ( ) {`
	`/* Interrupt occurrence */`
	`:`
	`}`
	
	`void main ( ) {`
	`st = func ( ) ; /* Interrupt occurrence */`
	`}`
	在上述服务中，如果在中断目标调用 func 函数， st 可能无法正常运行。
	
	[预防措施] 无

表 B-1 相关使用注意事项列表

编号	注意事项

29	[union 初始化相关的注意事项]
	当以结构、 union 或阵列作为元素的 union 进行初始化时，指定的初始化程序语法中带有嵌套的话，将发生错误 （E0750）。
	示例
	`struct Ss {`
	`int d1 , d2 ;`
	`} ;`
	`union Au {`
	`struct Ss t1 ;`
	`} u = { { 1, 2 } }; /* E0750 Initializer syntax */`
	[预防措施] 不要指定带嵌套的 union 初始化程序。
	示例
	`struct Ss {`
	`int d1 , d2 ;`
	`} ;`
	`union Au {`
	`struct Ss t1 ;`
	`} u = { 1, 2 };`

30	[汉字代码分类相关的注意事项] 要使用包含有 EUC 的源代码 , 需要将环境变量 LANG78K 设置为 euc，或指定 -ze 选项。

附录 C　命令选项

本章节以表格形式总结了程序选项。

开发程序时可参考。

该选项表可用作选项索引。

表 C-1 编译选项

类型	说明 格式	功能	和其他选项的关系	省略时解释
设备类型说明	-c 设备类型	指定目标设备的类型。	独立的	无
目标模块文件创建说明	-o [输出文件名]	指定目标模块文件的输出。	如果同时指定 -o 和 -no，则最后指定的选项有效。	-o 输入文件名 .rel
	-no	指定不输出目标模块文件。		
存储分配说明	-r [处理类型] (多种可能的说明)	指定存储器分配的方法	如果同时指定 -r 和 -nr、 -rd 和 -nr、 -rk 和 -nr 以及 -rs 和 -nr 时，最后指定的那一选项有效。	-nr
	-rd [n] [m] (n = 1, 2, 4)	定义的外部变量 / 外部静态变量将自动分配至 saddr 区域。		
	-rk [n] [m] (n = 1, 2, 4)	自动分配函数参数和自动变量 （除了静态自动变量）到 saddr 区。		
	-rs [n] [m] (n = 1, 2, 4)	自动分配静态自动变量到 saddr 区。		
	-nr	禁止使用 -r、 -rd、 -rk 和 -rs 选项。		
优化说明	-q[优化类型] (如果需要指定多种选项，连续指定即可)	设定调用优化程序生成高效的目标文件。	如果同时指定 -q 和 -nq，则最后指定的选项有效。	-qcjlvw
	-nq	使 -q 选项无效。		
调试信息输出说明	-g [n] (n = 1, 2)	设定输出源代码级别的调试信息。	如果同时指定 -g 和 -ng，则最后指定的选项有效。	-g2
	-ng	使 -g 选项无效。		

表 C-1 编译选项

类型	说明 格式	功能	和其他选项的关系	省略时解释
预处理列表文件创建说明	-p [输出文件名]	设定预处理列表文件的输出。	如果没有指定 -P 选项，那么 -K 选项无效。	无 (无文件输出)
	-k[处理类型] (可以指定多种规范)	设定预处理列表的处理。		-kfln
预处理说明	-d 宏名 [= 定义名] [，宏名 [= 定义名]]... (可以指定多项说明)	设定与 C 源代码中定义的文本相兼容的进程。	独立的	只有在 C 源程序模块文件中的宏定义才有效。
	-u 宏名 [, 宏名]... (可以指定多项说明)	在 C 源程序中，禁止类似 #undef 语句的宏定义。	独立的	使用 -d 设定的宏定义有效。
	-i folder [, folder]...(可以指定多项说明)	从指定目录查找输入 C 源程序中 #include 语句指定的包含文件。	独立的	1. 含有源程序文件的文件夹 2. 通过环境变量 INC78K0S 设定的文件夹 3. C:\NECTools32\inc78k0s
汇编源模块文件创建说明	-a [输出文件名]	设定汇编源模块文件的输出。	若同时设定 -a 和 -sa 时，则禁止使用 -sa。	没有汇编源程序输出
	-sa [输出文件名]	添加 C 源程序作为汇编源模块文件的注释。		
错误列表文件创建说明	-e [输出文件名]	设定错误列表文件的输出。	独立的	没有错误列表文件输出。
	-se [输出文件名]	把 C 源模块文件添加到错误列表文件中。	独立的	
交叉引用列表文件创建说明	-x [输出文件名]	设定交叉参考列表文件的输出。	独立的	没有交叉引用列表文件输出

表 C-1 编译选项

类型	说明 格式	功能	和其他选项的关系	省略时解释
列表格式说明	-lw [字符数量]	指定列表文件每行字符数。	独立的	-lw132 (对于控制台的输出是 80 个字符)
	-LL [行数量]	指定每种类型表文件的每页行数。	独立的	-ll66 (对于控制台输出为 65535 行)
	-lt [字符数量]	-lt 选项指出在源模块文件中输出水平制表符 (HT， tab) 的基本跨度，并在列表文件中用一些空白 (空格) 来代替。	独立的	-lt8
	-lf	指定将会在每个列表文件的末尾添加新的分页符。	独立的	无
	-li	添加 include 文件中的 C 源代码到带 C 源代码注释的汇编源文件中。	独立的	无
警告输出说明	-w [level]	指定警告信息输出至操作台。	独立的	-w1
执行状态显示设定	-v	设定是否输出目前编译的执行状态到控制台。	如果同时指定 -v 和 -nv，则最后指定的选项有效。	-nv
	-nv	使 -v 选项无效。		
参数文件说明	-f 文件名	可以从指定的具体文件中读入设定选项或输入文件名。	独立的	只能从命令行输入选项和输入文件名。
临时文件创建文件夹说明	-t 文件夹	在指定的驱动和目录下建立临时文件。	独立的	通过环境变量 TMP 指定驱动和文件夹
帮助说明	-- / -? / -h	--，-?，和 -h 选项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)。	所有其他选项无效。	无屏幕显示

表 C-1 编译选项

类型	说明 格式	功能	和其他选项的关系	省略时解释
函数扩展说明	-z[类型] (如果需要指定多种类型，连续指定即可)	设定类型说明的进程。	如果同时指定 -z 和 -nz，则最后指定的选项有效。	-nz
	-nz	使 -z 选项无效。		
驱动器文件搜索路径	-y 文件夹	设定查找设备文件的路径。	独立的	只有正常搜索路径
静态模型说明	-sm [n] (n = 1-16)	设定目标的静态模型或常规模型。	独立的	普通模型 (n = 0)

附录 D 索引

详细信息请联系：

中国区

MCU 技术支持热线：
电话：+86-400-700-0606 (普通话)
服务时间：9:00-12:00，13:00-17:00 （不含法定节假日）

网址：
http://www.cn.necel.com/ （中文）
http://www.necel.com/ （英文）

[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

[成都]

日电电子（中国）有限公司成都分公司
成都市二环路南三段 15 号天华大厦 7 楼 703 室
电话：(+86)28-8512-5224
传真：(+86)28-8512-5334

[长春]

日电电子（中国）有限公司长春分公司
吉林省长春市朝阳区
西安大路 727 号中银大厦 A 座 1609 室
电话：(+86)431-8859-7533 / 8859-8533
传真：(+86)431-8680-2944

[大连]

日电电子（中国）有限公司长春分公司
大连市中山路 88 号天安国际大厦 2701 室
电话：(+86)411-8230-8815 / 8230-8825
传真：(+86)411-8230-8835