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

C 编译器 1.50 或更高版本

操作

目标设备

78K0S 系列

文档编号　U16654CA1V0UM00 (第一版)
发行日期　2007 年 7 月　CP(K)

© NEC Electronics Corporation 2003

日本印刷

[备忘录]

MS-DOS, 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。
UNIX 是 X/Open Company Limited 在美国及其他国家的注册商标。
SPARCstation 是 SPARC International, Inc 公司的一个注册商标。
SunOS 和 Solaris Sun Microsystems, Inc.的商标。
HP9000 Series 700 和 HP-UX 是 Hewlett-Packard 公司的商标。

- **本文档所刊登的内容有效期截至 2007 年 7 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。

区域信息

本文档中的某些信息可能因国家不同而有所差异。用户在使用任何一种 NEC 产品之前，请与当地的 NEC 办事处联系，以获取权威的代理商和发行商信息。请验证以下内容：

- 设备的可用性
- 定货信息
- 产品发布进度表
- 相关技术资料的可用性
- 开发环境要求（例如：要求第三方工具和组件，主计算机，电源插头，AC 供电电源等）
- 网络要求

此外，对于商标、注册商标、出口限制条款和其他法律规定，不同的国家也有不同的要求。

详细信息请联系：

（中国区）

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

[北京]

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

[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

[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

[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

前言

本手册的目的是使您能够完整的理解 CC78K0S (78K0S 系列 C 汇编程序)功能和的操作。

本手册没有解释如何编写 CC78K0S 源程序。因此，在阅读手册之前，请先阅读**“CC78K0S C 汇编语言用户手册(U16655E)”**(在后面会称作”语言手册”)。

[目标设备]

通过使用 CC78K0S 可以开发 78K0S 系列微控制器软件。为了使用这个软件，需要先安装 RA78K0S(78K0S 系列汇编程序包)(分开销售)和目标模块设备文件。

[目标读者]

本手册为通过阅读设备使用手册获得相关知识和有软件开发经验的人所写。然而，关于 C 编译器和 C 语言的知识并不是必需的，所以第一次使用 C 编译器的用户也可以使用该手册。

[结构]

本手册的结构如下。

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。

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**。

[相关资料]

下表为与本手册相关的资料(如用户手册)。在出版物中说明的相关资料包括初级版本。然而，初级版本并没有标注为此类。

开发工具资料(用户手册)

资料名		资料编号
1.50版本或更新的CC78K0S C 编译器	操作	本文
	语言	U16655E
1.40版本或更新的 RA78K0S 汇编程序包	操作	U16656E
	汇编语言	U16657E
	结构化汇编语言	U11623E
SM78K0S系统模拟器	操作	准备中
2.51版本或最新的ID78K0S-NS集成调试器	操作	U16584E
5.10版本附加项目管理器		准备中

注意　上述列出的相关资料可能会有更新，请务必使用最新版本的资料进行设计开发。

[约定]

以下为本手册使用符号的说明。

…:	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

[文件名约定]

在命令行中指定输入文件名的约定如下所示。

(1) 指定磁盘文件名

```
[驱动器名]   [\]   [[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。

```
实例:   C:\nectools32\smp78k0s\CC78k0s\prime.C
```

注意 **1.** 在‘:’, ‘.’和 ‘\’之前或之后不能有空格。

2. 不区分大写和小写 (大小写不敏感)。

(2) 指定设备文件名

可以使用下列逻辑设备。

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

目录

第 1 章 概述

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 能够识别的机器语言。

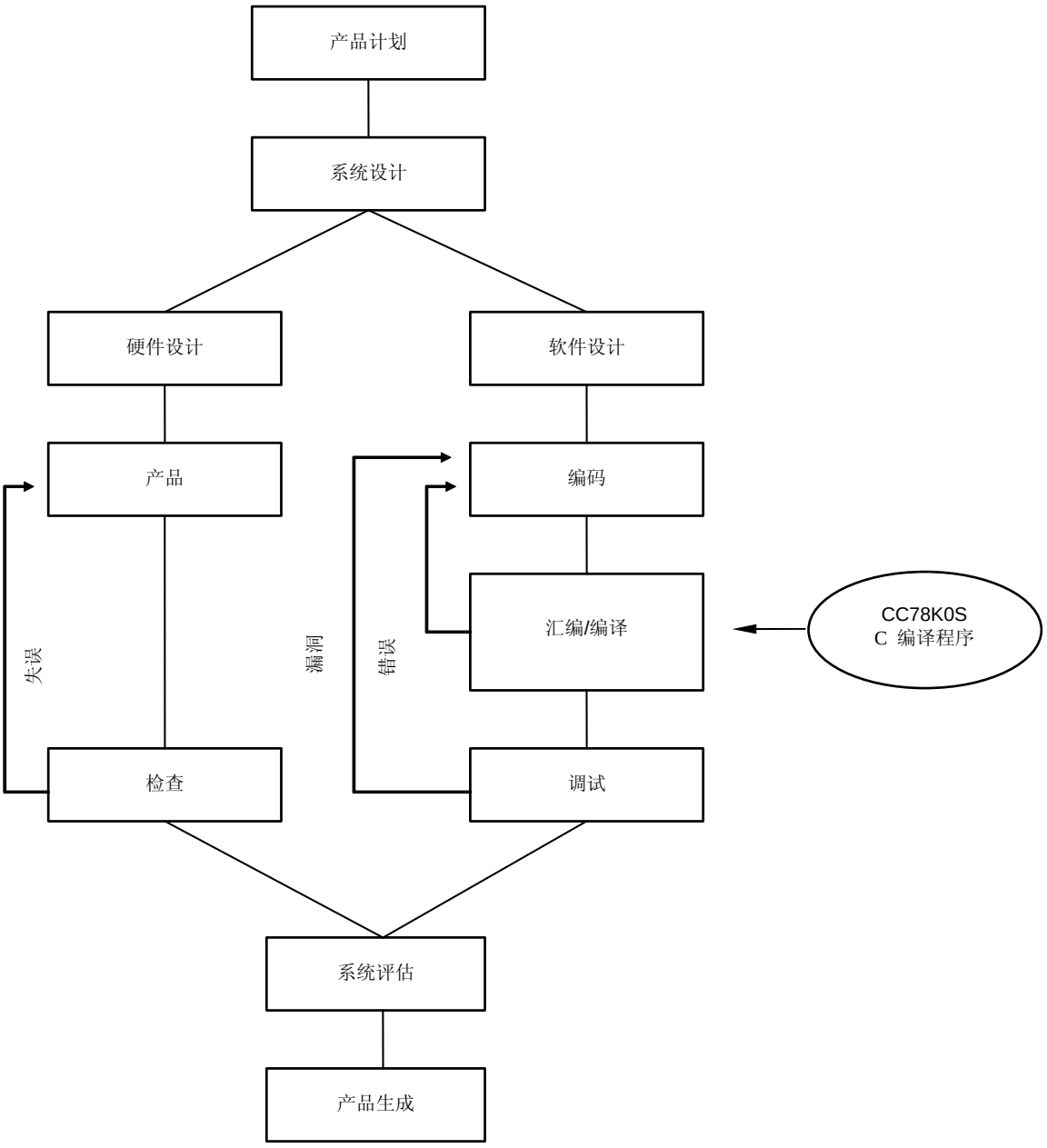
CC78K0S 支持 Windows™ 98/Me/2000/XP 或 Windows NT™ 4.0 系统，必须在 PM+（项目管理器）中使用，PM+工具包含在 CC78K0S 系列的汇编程序安装包中。如果没有使用 PM+，编译程序可以从 DOS 提示符下（Windows 98/Me）或者命令提示符下（Windows NT 4.0/2000/XP）运行（对 Windows 版本而言）。

注 ANSI-C 是美国国家标准局所制定的 C 语言标准。

1.1 微控制器应用产品开发和 CC78K0S 的作用

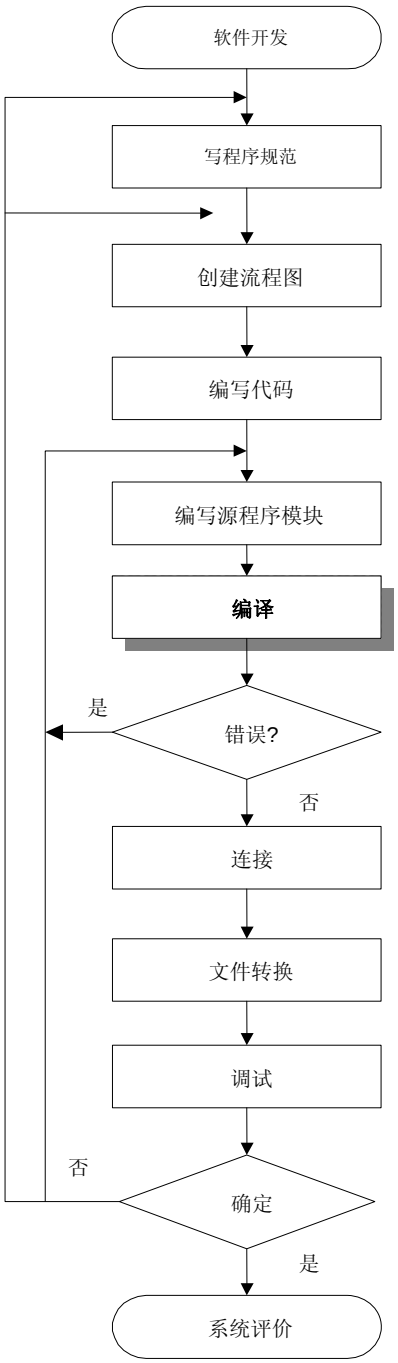
CC78K0S 在产品开发中的位置如下图所示。

图 1-1. 微控制器应用产品开发过程



软件开发过程如下图所示。

图 1-2. 软件开发过程



… 使用 78K 系列 C 语言或 ANSI-C。

… 使用编辑器创件建 C 源程序模块文件。

… 连接到引用库和函数库。

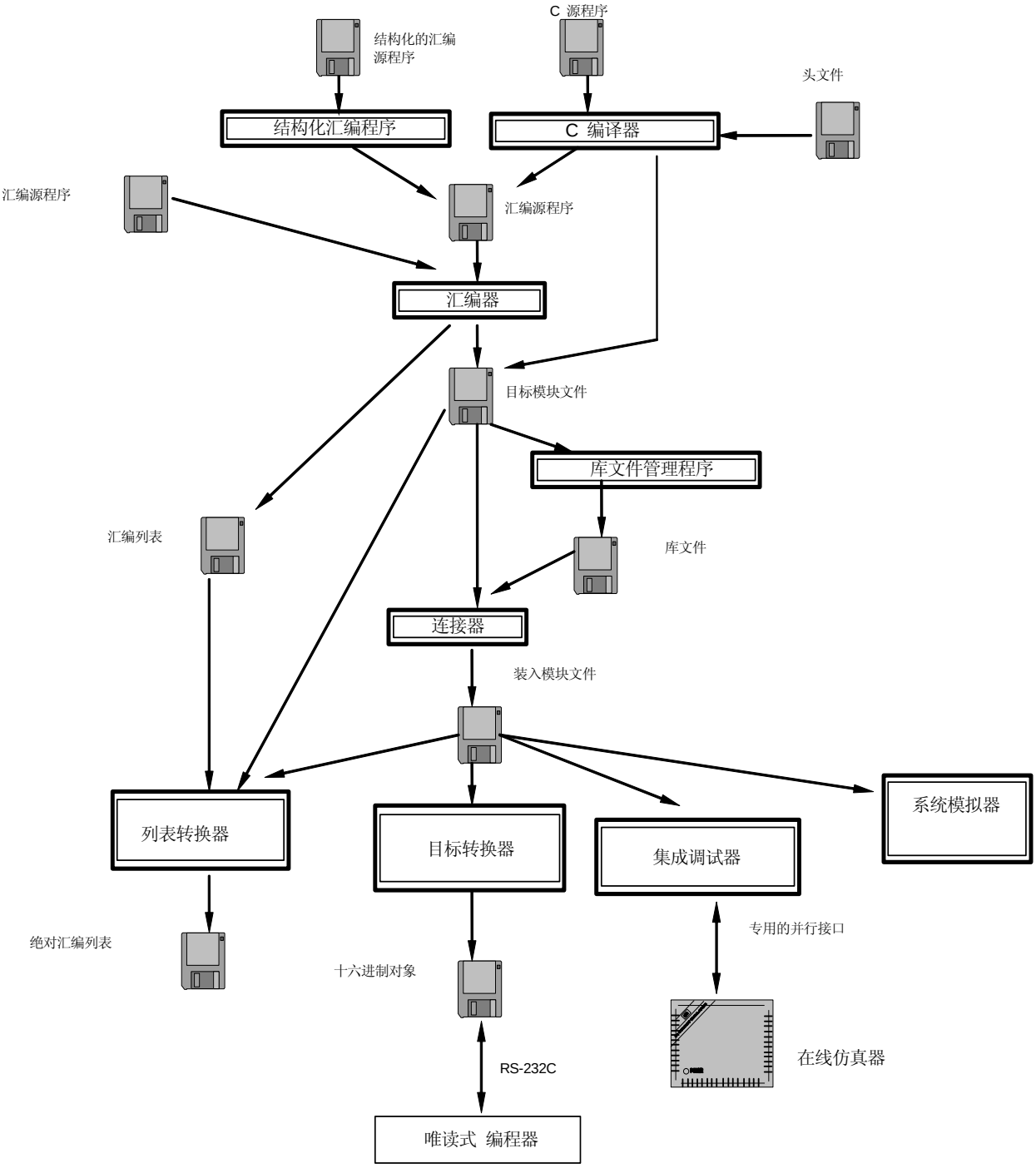
… 将文件转换成十六进制格式。

… 使用硬件调试器（在线仿真器等）来检验系统的运行。

1.2 使用 CC78K0S 的开发过程

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

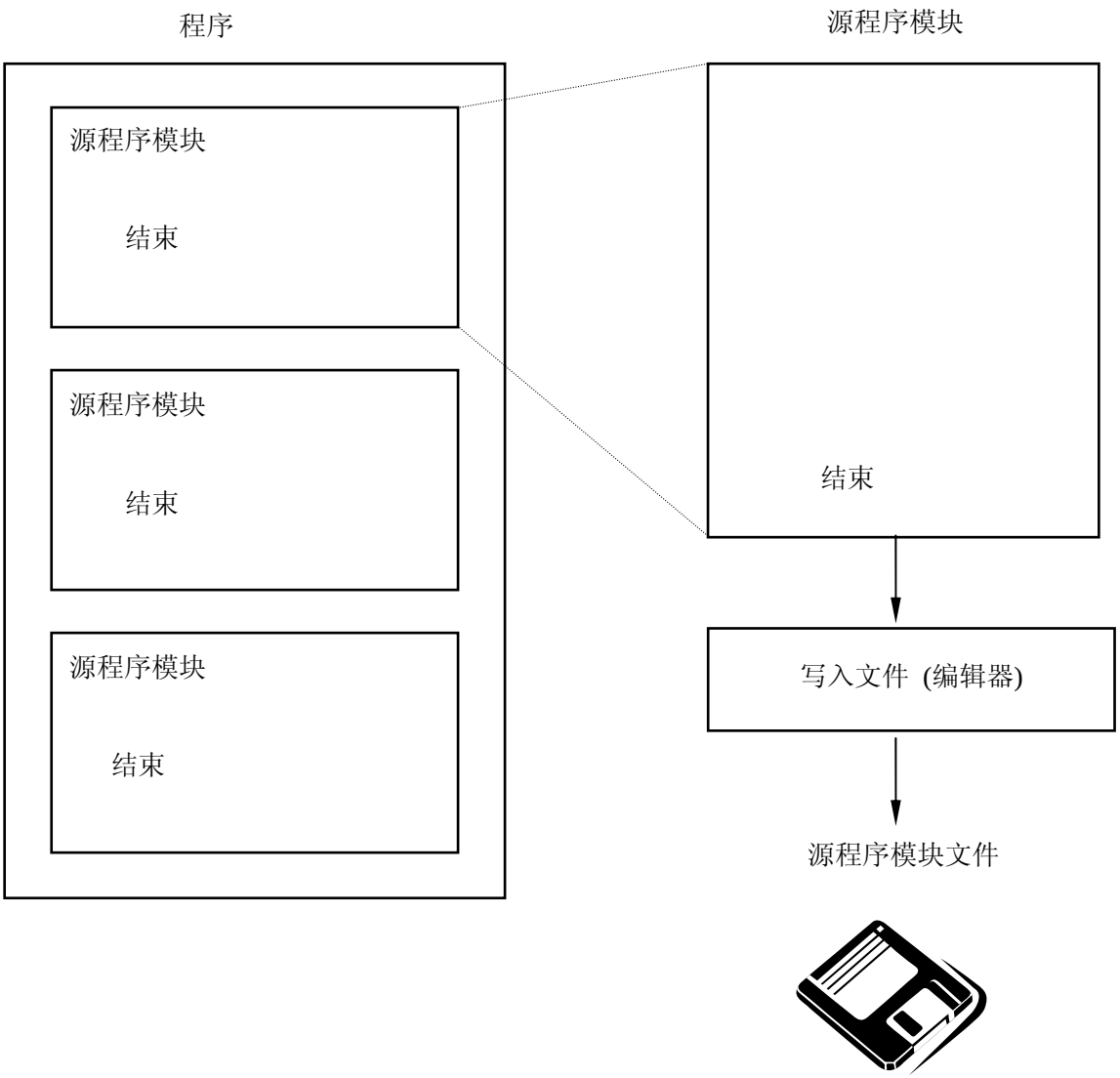
一个程序被划分为若干个功能模块。

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 源程序模块文件就是 CC78K0S 的输入文件。

图 1-4. 创建源程序模块文件

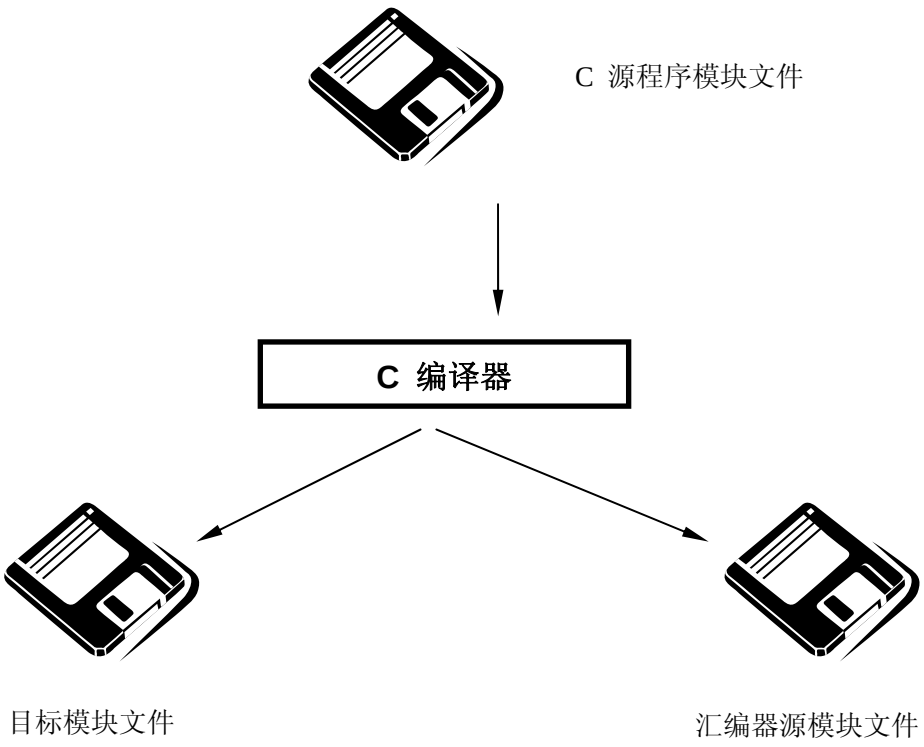


1.2.2 C 编译器

C 编译器读入 C 源程序模块后，将 C 语言转换成机器语言。如果在 C 源程序模块中发现描述错误，就会输出编译错误信息。

如果没有编译错误，就会生成目标模块文件。为了在汇编语言中对程序进行校正和检查，需要生成汇编源程序模块文件。如果要输出汇编源程序模块文件，当编译时具体编译选项中选中确定-A 或-SA 选项，就可以创建汇编源程序模块源文件。（要查看选项的相关信息，可参阅第 5 章 编译选项）

图 1-5. C 编译器功能



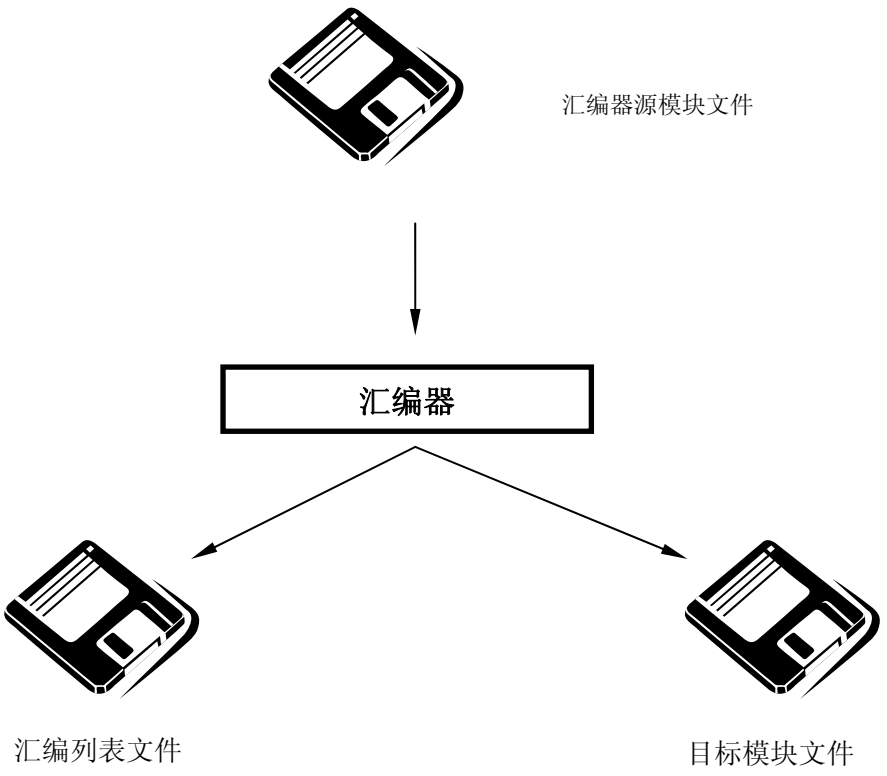
1.2.3 汇编器

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。

汇编器是可以读入汇编源程序模块文件并将汇编语言转化成机器语言的一种程序。如果在源程序模块中发现描述错误，就会输出汇编错误。

如果没有汇编错误，则会生成目标模块文件，这个目标文件模块种包括机器语言信息和位置分配信息，例如每条机器语言代码被放在哪个内存单元地址。除此以外，在汇编过程中的信息会以汇编表文件的形式输出。

图 1-6. 汇编器功能



1.2.4 连接器

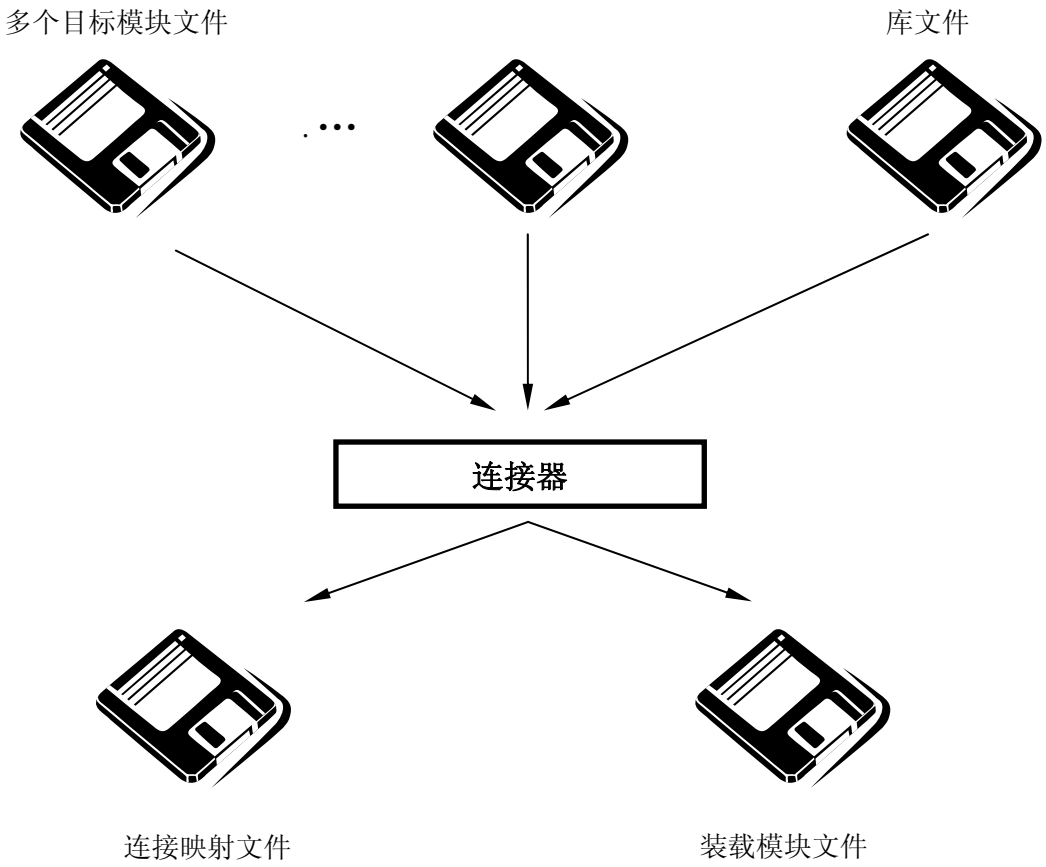
连接操作是通过使用 RA78K0S 汇编程序包（分开销售）中的连接器来执行的。

连接器的输入文件有编译器输出的目标模块文件，也有汇编器输出的目标模块文件，同时将它们和库文件连接起来(即使只有一个目标模块，也必须执行连接操作)。会输出一个装载模块文件。

在这种情况下，连接器决定输入模块中的重定位段的地址。同时也决定了重定位符号的值和外部引用符号的值，并将正确的值嵌入到装载模块文件中。

连接器将连接信息输出到连接映射文件（link map）。

图 1-7. 连接器功能



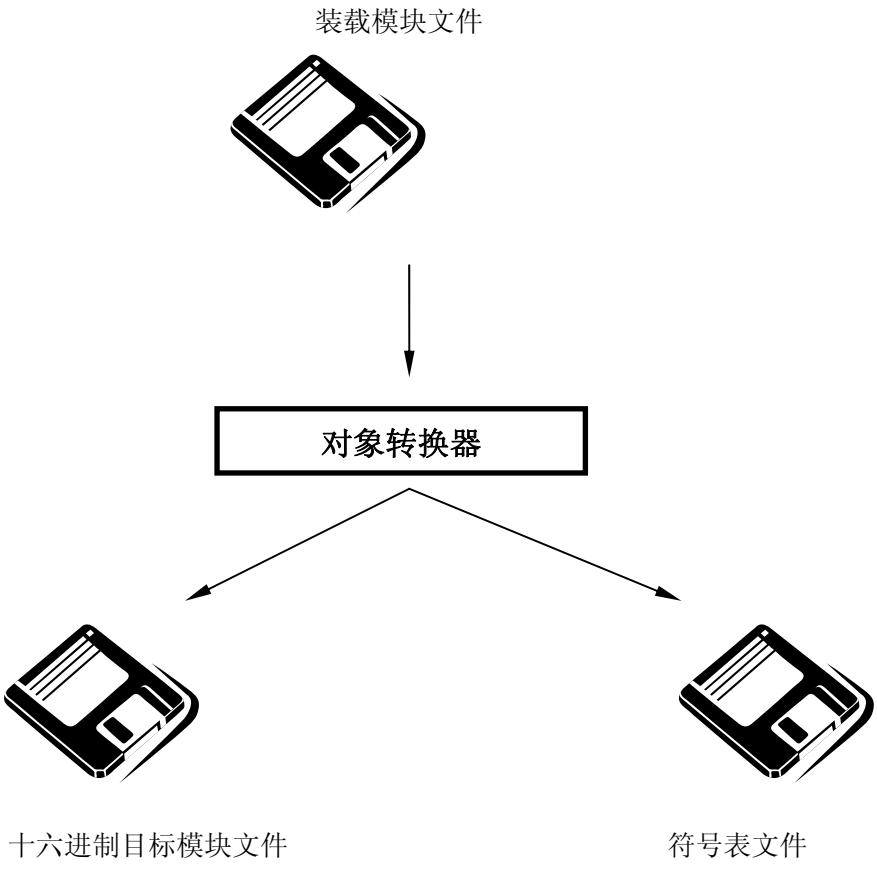
1.2.5 目标转换器

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。

目标转换器读入连接器产生的装载模块文件，转换文件的格式，产生 Intel 标准格式的十六进制模块文件。

符号信息输出符号表文件中。

图 1-8. 目标转换器功能



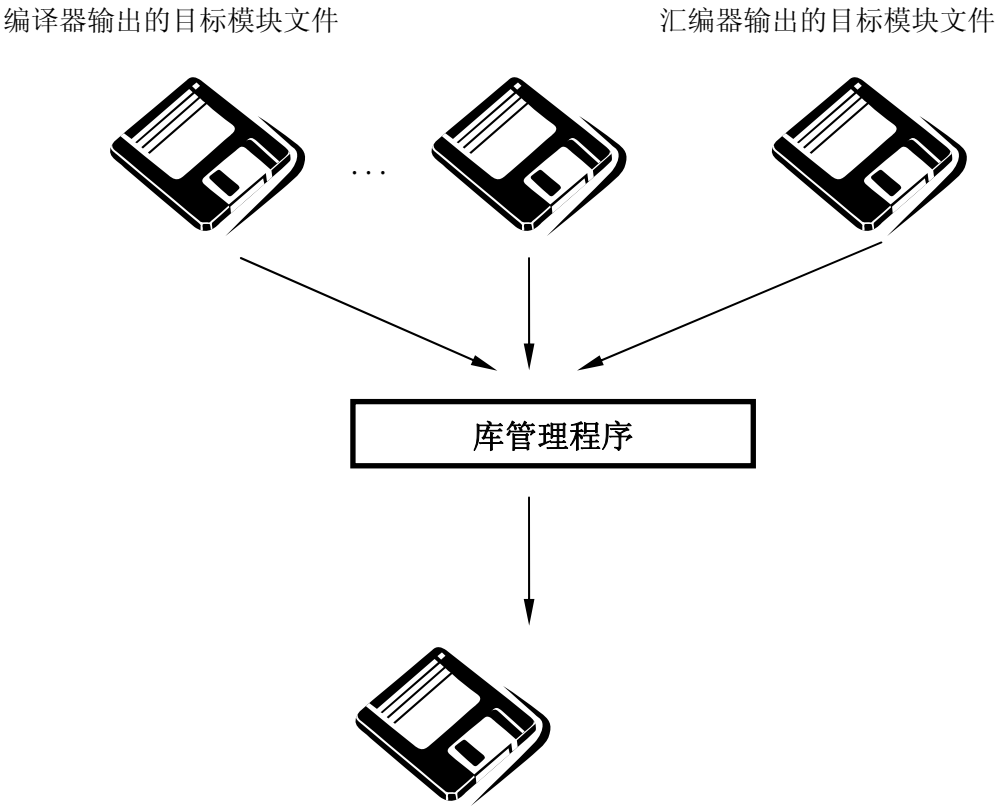
1.2.6 库管理程序

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。

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

库管理程序用来创建和更新库文件。库管理功能通过 RA78K0S 汇编程序包(分开销售)中的库管理程序来执行。

图 1-9. 库管理程序功能

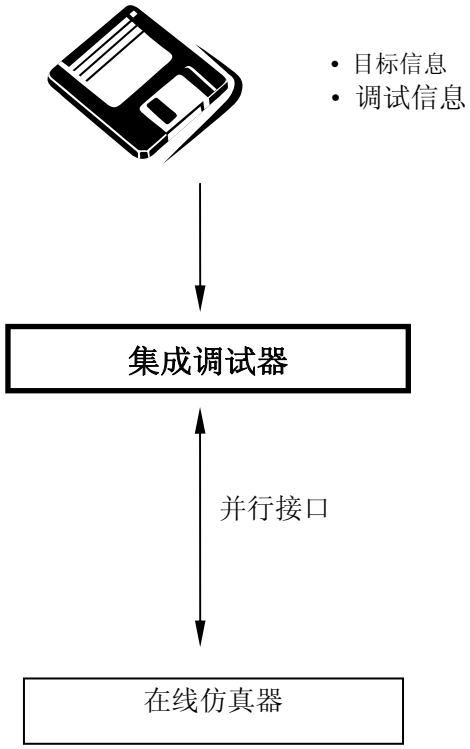


1.2.7 调试器

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

图 1-10. 调试器的功能

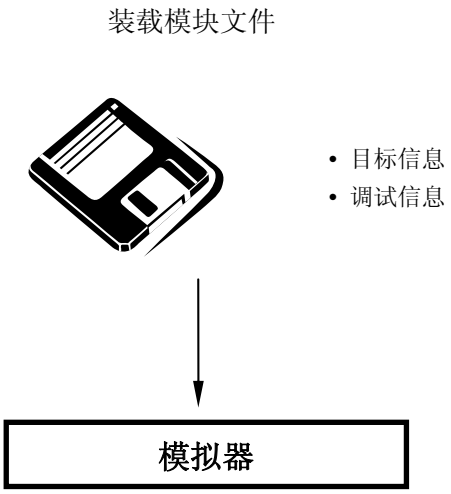


1.2.8 系统模拟器

使用 SM78K0S (78K0S 系列系统模拟器)将连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 在主机上执行模拟的软件。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. 模拟器功能

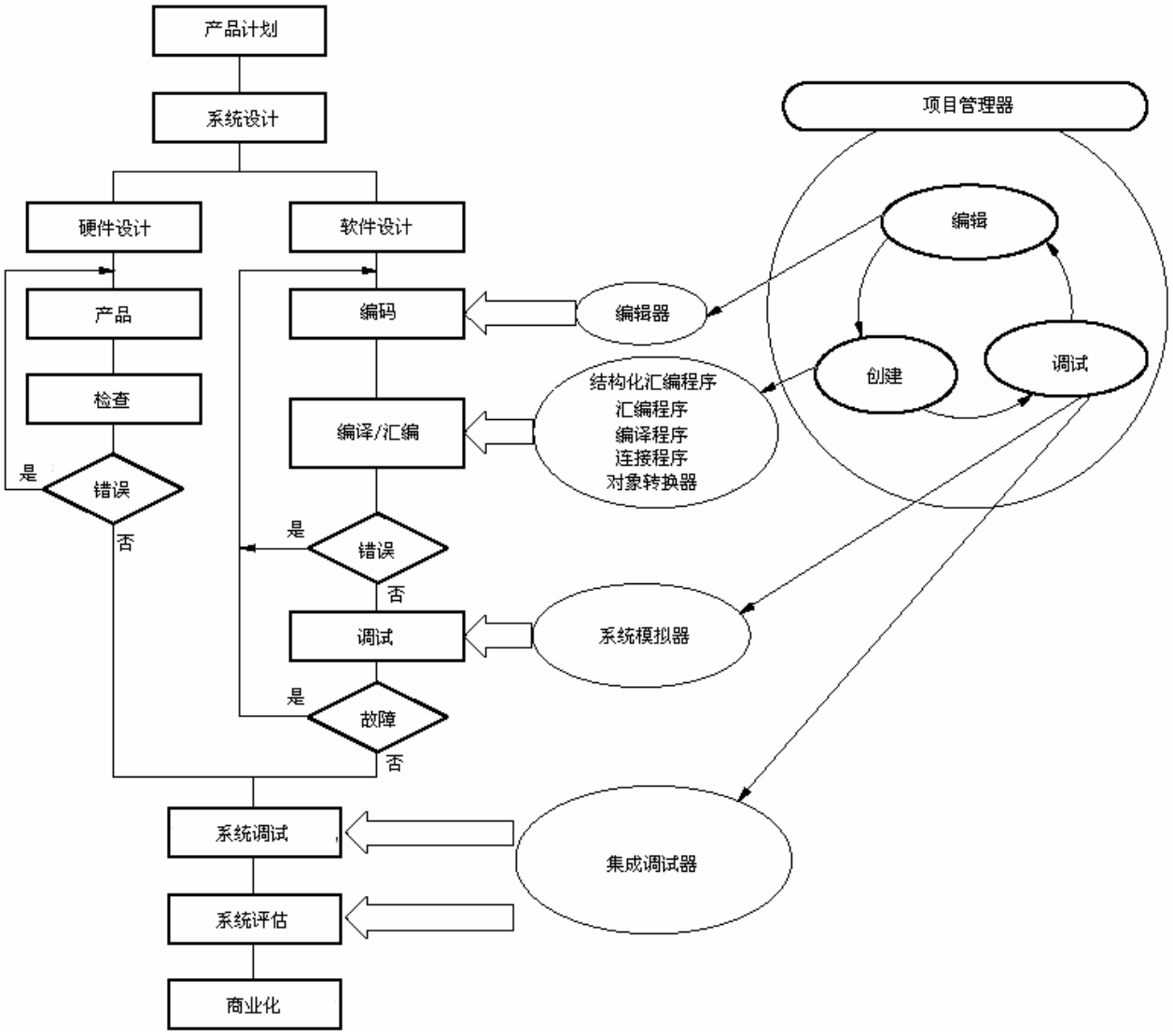


1.2.9 PM+

PM+（项目管理器）使用 DLL 文件加载 CC78K0S，它可以在 Windows 98/Me/2000/XP 或 Windows NT 4.0 系统上启动 CC78K0S。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 汇编程序安装包中。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



注意 Build 阶段会分析并执行 makefile 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. C编译器的供应媒介和记录格式

主机	操作系统	提供媒介	记录格式
PC-9800 系列	日文 Windows (98/Me/2000/XP/NT 4.0)注	CD-ROM	支持 Windows 标准安装方式
IBM PC/AT™ 及兼容机	日文 Windows (98/Me/2000/XP/NT 4.0)注 英文 Windows (98/Me/2000/XP/NT 4.0)注		
HP9000 系列 700™	HP-UX™ (Rel. 10.10 and later)	CD-ROM	cp 命令
SPARCstation™ 家庭版	SunOS™ (Rel. 4.1.4 and later) Solaris™ (Rel. 2.5.1 and later)		

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 NT 4.0/2000/XP)下启动 C 编译器。

2.2 安装

2.2.1 Windows 版本的安装

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。

警告 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) 文件确认

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。

每个文件夹的详细信息，参考 2.4.1 Windows 版本目录结构。

2.2.2 UNIX 版本的安装

安装 UNIX 版本的过程如下。假设安装目录为/nectools。

(1) 登录

登录主机。

(2) 目录选择

进入安装目录。

```
%cd /nectools
```

(3) 打开供应媒体

将 CD-ROM 放入 CD-ROM 驱动器并关闭托盘。

(4) 对文件执行 cp 命令，从 CD-ROM 中复制文件(请先确认 CD-ROM 已经放到 CD-ROM 驱动器中)。

将/nectools/bin 添加到环境变量路径。

2.3 设备文件的安装

2.3.1 安装 Windows 版本

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。

2.3.2 安装 UNIX 版本

使用-y 选项（例：-y/nectools/dev）指定设备文件的目录，或者以编译器执行格式把文件复制到某个目录中（例：/nectools/bin）。

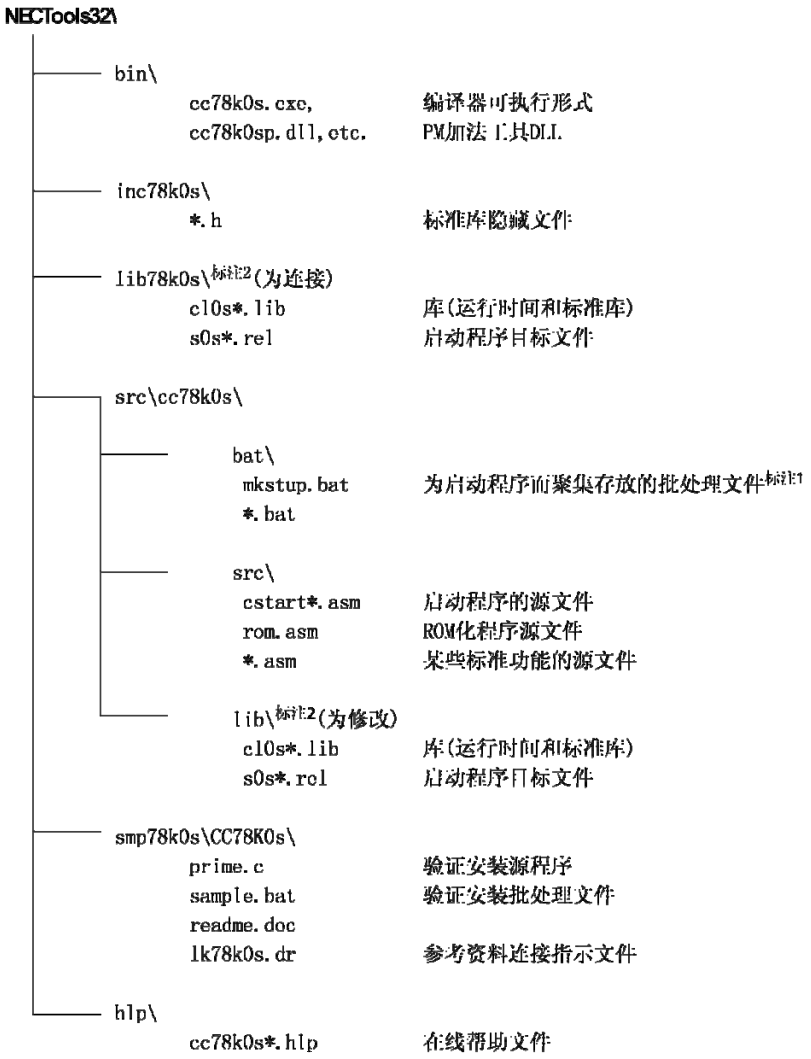
2.4 目录结构

2.4.1 Windows 版本目录结构

Windows 系统安装过程中的标准目录是“NECTools32”。安装目录中的文档结构如下，注意在安装过程中可以改变驱动器和安装目录。当使用 PM+执行 MAKE 操作时，相关的工具(CC78K0S，RA78K0S)也被安装到这个驱动器和目录。

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

图 2-1. 目录结构

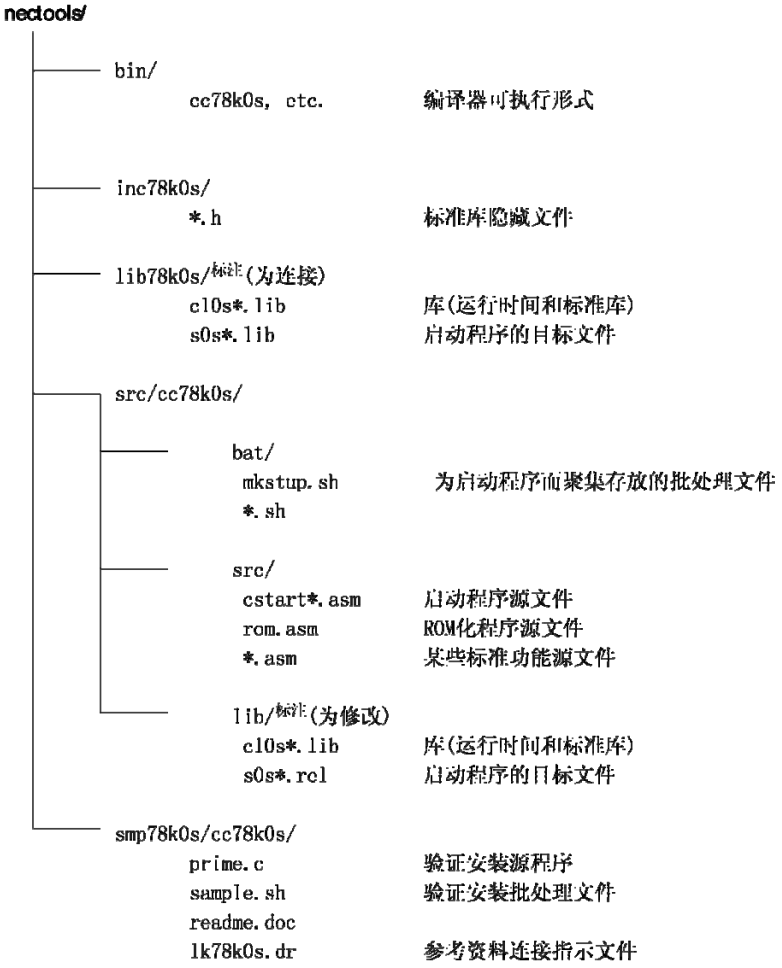


注 **1.** 这个批处理文件不能在 PM+中使用。批处理文件只能在 DOS 提示符(Window 98/Me)或命令提示符(WindowsNT 4.0/2000/XP)中运行。

2. Lib78k0s 目录中的启动例程和库和 src/cc78k0s 目录中的启动例程和库是完全相同的。如果启动例程被修改了，则改变 src/cc78k0s 目录中的源程序。由于批处理文件得到的汇编文件存放在 src/cc78k0s，复制 lib78k0s 目录进行链接。

2.4.2 UNIX 版本目录结构

使用 cp 命令安装到/nectools 的文件结构显示如下。



注 Lib78k0s 目录中的启动例程和库和 src/cc78k0s 目录中的启动例程和库是完全相同的。如果启动例程被修改了，则改变 src/cc78k0s 目录中的源程序。由于批处理文件得到的汇编文件存放在 src/cc78k0s，复制 lib78k0s 目录进行链接

2.5 卸载过程

2.5.1 卸载 Windows 版本

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<控制面板> 窗口

按 开始 按钮，选择 [设置]-[控制面板]，打开<控制面板>窗口。

(3) 打开 <添加或删除程序属性>窗口

在<控制面板>窗口中双击[添加或删除程序]图标，打开<添加或删除程序属性>窗口。

(4) 删除 CC78K0S

从<添加或删除程序属性>窗口的<<当前安装的程序>>显示的软件安装列表选择“NEC CC78K0S/78K/0S C Compiler Vx.xx”，单击[更改/删除]按钮。当<系统设置改变>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。文件夹的详细说明，请参考 2.4.1 Windows 版本目录结构。

2.5.2 卸载 UNIX 版本

使用 rm 命令删除在 **2.3.2 安装 UNIX 版本**中复制的文件。

2.6 环境设置

2.6.1 主机（PC-9800 系列和 IBM PC/AT 兼容机）

CC78K0S 可以处理 32 位，并且在配置 i386™CPU 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。

Windows 98/Me/2000/XP/NT 4.0
Windows 98/Me 下的 DOS 提示符
Windows 2000/XP/NT 4.0 下的命令提示符

2.6.2 环境变量

为 EWS 和 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	指定编译器的可执行文件所在目录。
TMP	指定创建临时文件的目录 (只对 PC-9800 系列和 IBM PC/AT 兼容机有效)。
LANG78K	在原文件中指定汉字代码（2 字节代码）。 sjis Shift JIS（默认 PC-9800 系列，IBM PC/AT 兼容，和 HP9000 系列 700） euc EUC（默认 SPARCstation） none 非双字符编码
INC78K0S	指定编译器的标准头文件目录（只有对 EWS 是必需的）。
LIB78K0S	指定编译器的程序库的目录（只有对 EWS 是必需的）。

实例说明

针对 PC-9800 系列和 IBM PC/AT 兼容

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

针对 HP9000 系列 700 和 SPARCstation

使用 csh 的例子

```
set path = ($path /nectools/bin)
setenv LANG78K    euc
setenv INC78K0S     /nectools/inc78k0s
setenv LIB78K0S     /nectools/lib78k0s
```

使用 sh 的例子

```
PATH = $PATH:/nectools/bin
LANG78K = euc
INC78K0S = /nectools/inc78k0s
LIB78K0S = /nectools/lib78k0s
export PATH LANG78K INC78K0S LIB78K0S
```

2.6.3 文档结构

下面的表格列出了每个目录的内容。说明了针对 PC-9800 系列和 IBM PC/AT 兼容机的文件。安装时就已经决定了目录结构和文档组织。

注意 一些文件扩展名和 UNIX 的不同。

表2-3. 文档组织结构(* = 字母数字符号)

目录名	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 rom.asm *.asm 注 5	启动例程的源文件 ROMization 例程的源文件 标准函数的源文件（部分的）
HLP	*.hlp	在线帮助文件

注 **1.** 见**语言**篇的 **10.2 头文件(U16655E)**。

2. 目录中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。

3. 参考**表 8-1 BAT 目录内容**。

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

5. 参考**表 8-2 SRC 目录内容**。

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 列出了目录内容。

表2-4. 程序库文件

目录名	文件名	文件作用
LIB78K0S¥	cl0s.lib cl0sr.lib cl0ss.lib cl0sf.lib	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<float><pascal><model>.lib
```

<float>

None 标准库和运行时刻库 (移动指针程序库没有使用)
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 普通模型
s 静态模型

注 **2.** 命名启动例程的规则如下。

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

<model>

None 普通模型
sm 静态模型

<lib>

None 未使用标准库函数
l 使用标准库函数

第 3 章 编译到连接的过程

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

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 样例程序**的相关资料）。

描述了在 PC 机 9800 系列和 IBM PC/AT 系列兼容机上如何执行 PM+。至于其他型号的机器，描述了如何从命令行来执行编译到连接(安装信息参见 **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 RA78KOS 汇编程序安装包中。如果在 PM+中启动 CC78K0S，会引用 CC78K0S 中的 CC78K0SP.DLL 动态连接文件。

3.1.1 CC78K0SP.DLL 的位置(工具动态连接文件)

这些工具相关的动态连接文件，比如 CC78K0SP.DLL，需要从 WINDOWS 98/Me/2000/XP 或 Windows NT 4.0 系统下的 PM+中运行 78K0S 系列 C 编程器（CC78K0S）。

3.1.2 执行的环境

这个环境由 PM+决定。

显示的模式有两种，根据操作系统分别是日语模式和英语模式。(英文版/日文版 WINDOWS)。

3.1.3 CC78K0S 选项设置菜单

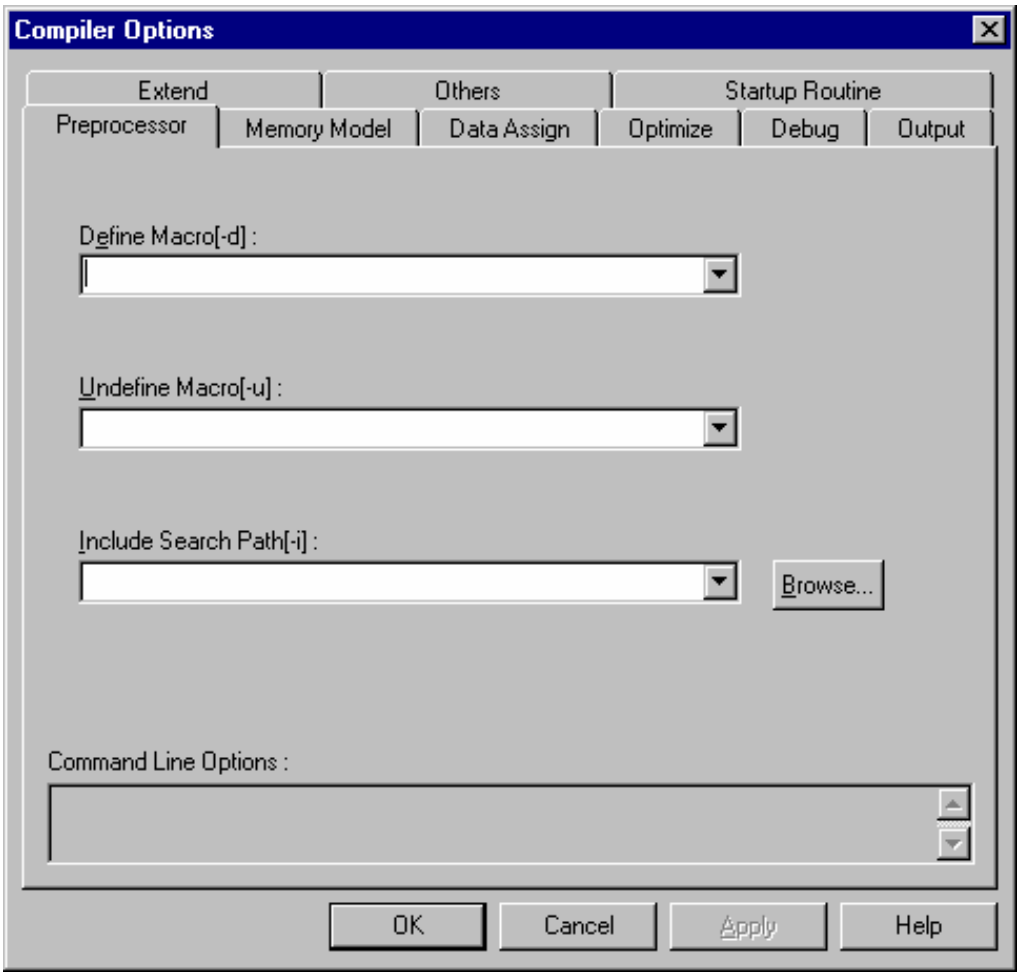
(1) 选项菜单条目

CC78K0S C 编译器安装包中的工具动态连接文件 DLL 文件将在 PM+中的[Tools]菜单中添加“Compiler Options…”项。

(2) < Compiler Options > 对话框

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

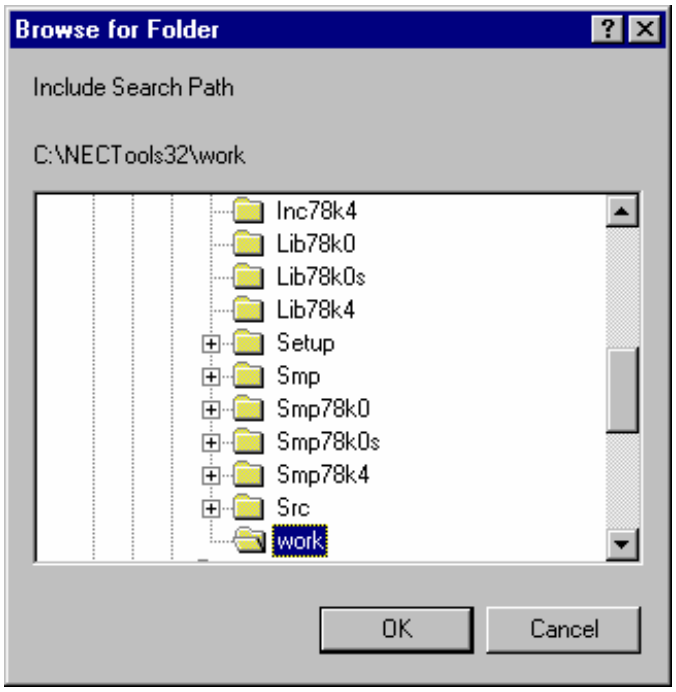
< Compiler Options >对话框如下所示。



(3) < Browse for Folder >对话框

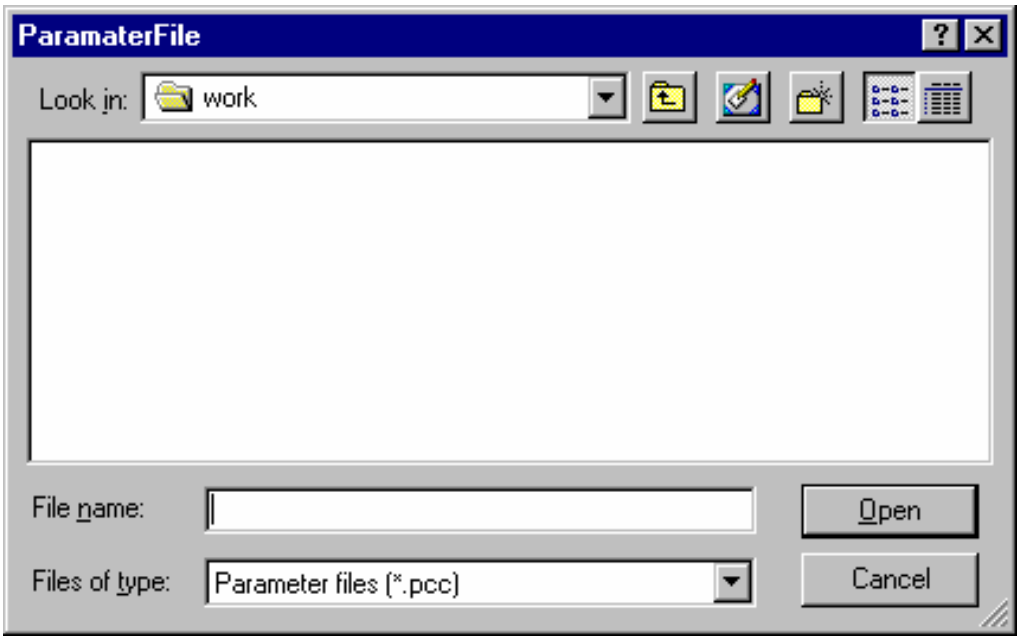
在< Compiler Options > 对话框中，当单击 Browse… 按钮设置路径时，会显示下列对话框。在本对话框中只能选文件夹。

- 包含文件路径
- 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



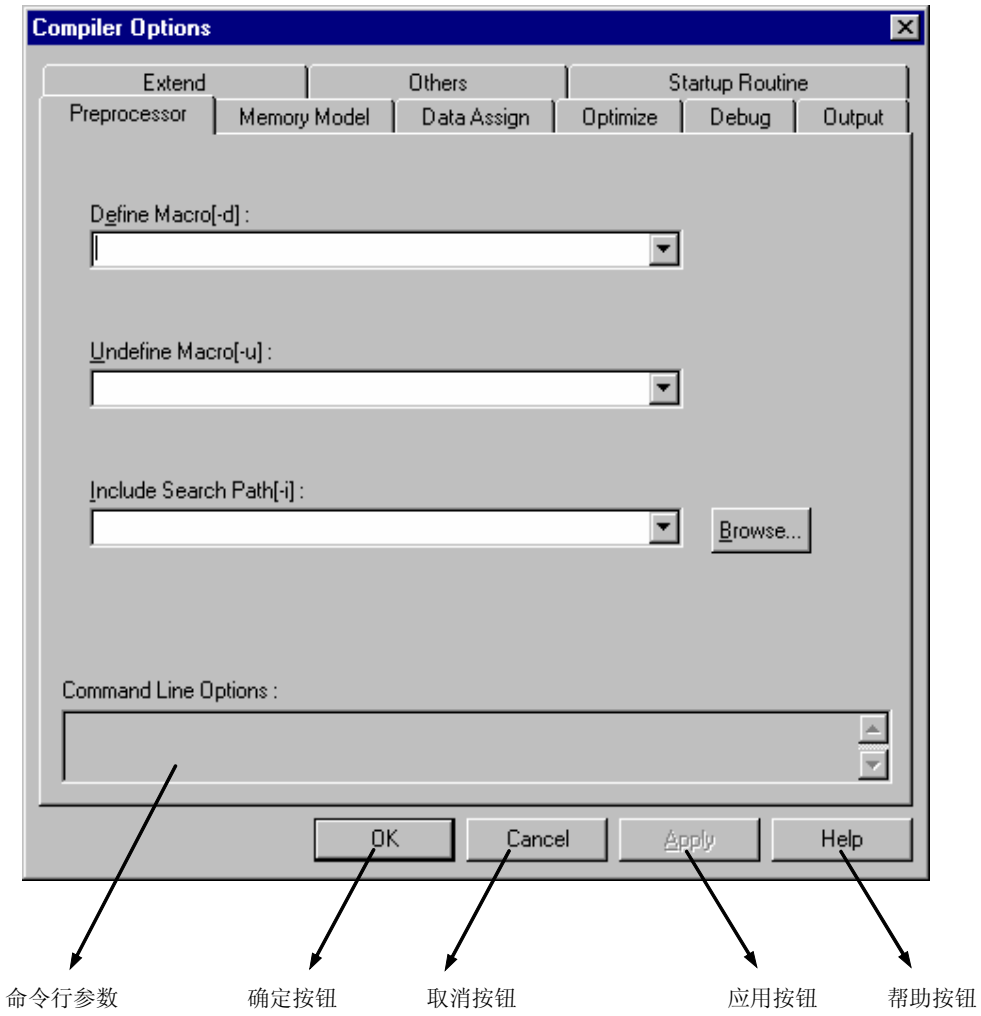
在指定参数文件时单击 Browse… 按钮，会出现下列对话框。
对话框的情况如下：

当前目录： 项目文件目录
文件类型： 参数文件 (*.pcc)



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。



·[OK] 按钮

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

·[Cancel]按钮

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

·[Apply] 按钮

只有在设置的选项发生改变时这个按钮才有效。

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

·[Help]按钮

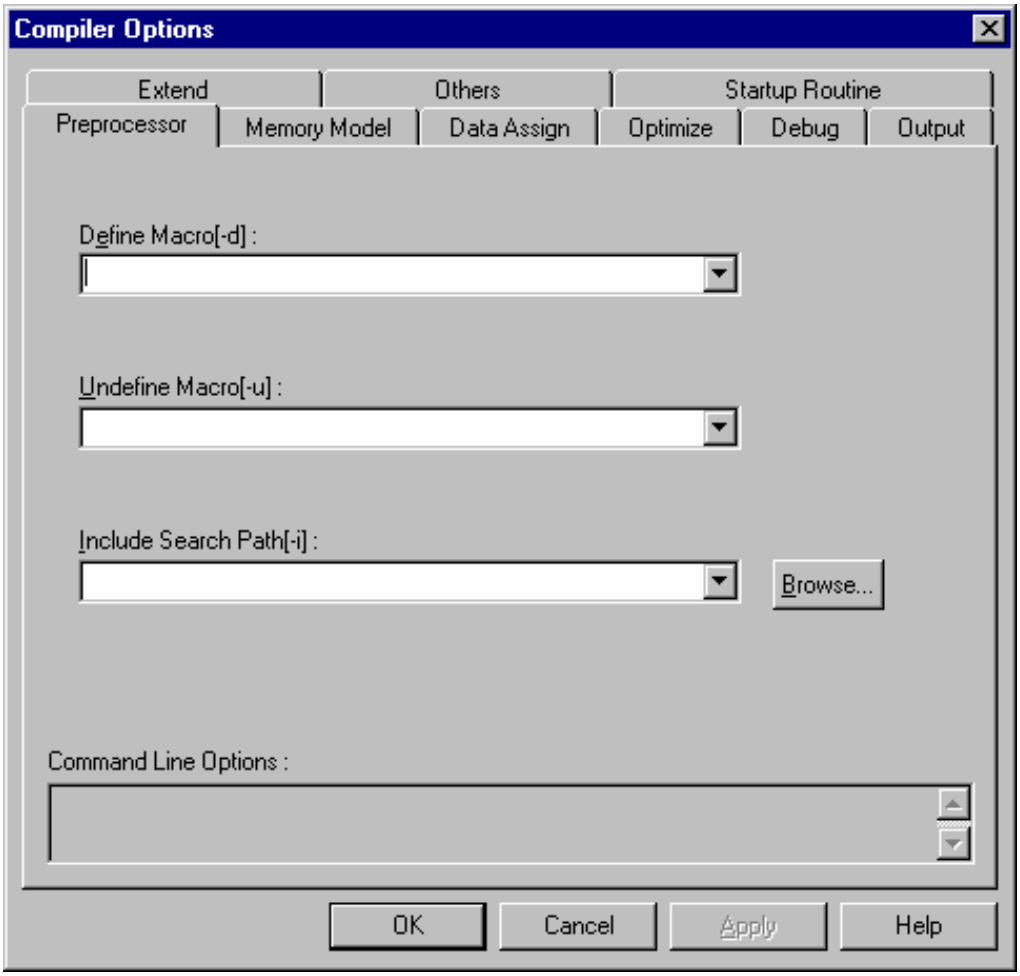
打开关于本对话框的帮助文件。

- 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 CC78KOS 的默认选项是“已指定的”状态 (比如，选择框是选中的，等等)，在默认的情况下也不会显示任何东西。
  选项不在选项属性显示区域显示时，可以通过滚动着这个  按钮来选择。

- 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。

  预处理器 (默认)
  存储器模块
  数据的分配
  优化
  调试
  输出
  扩展
  其他
  启动例程

(1) 选择“Preprocessor”时的界面

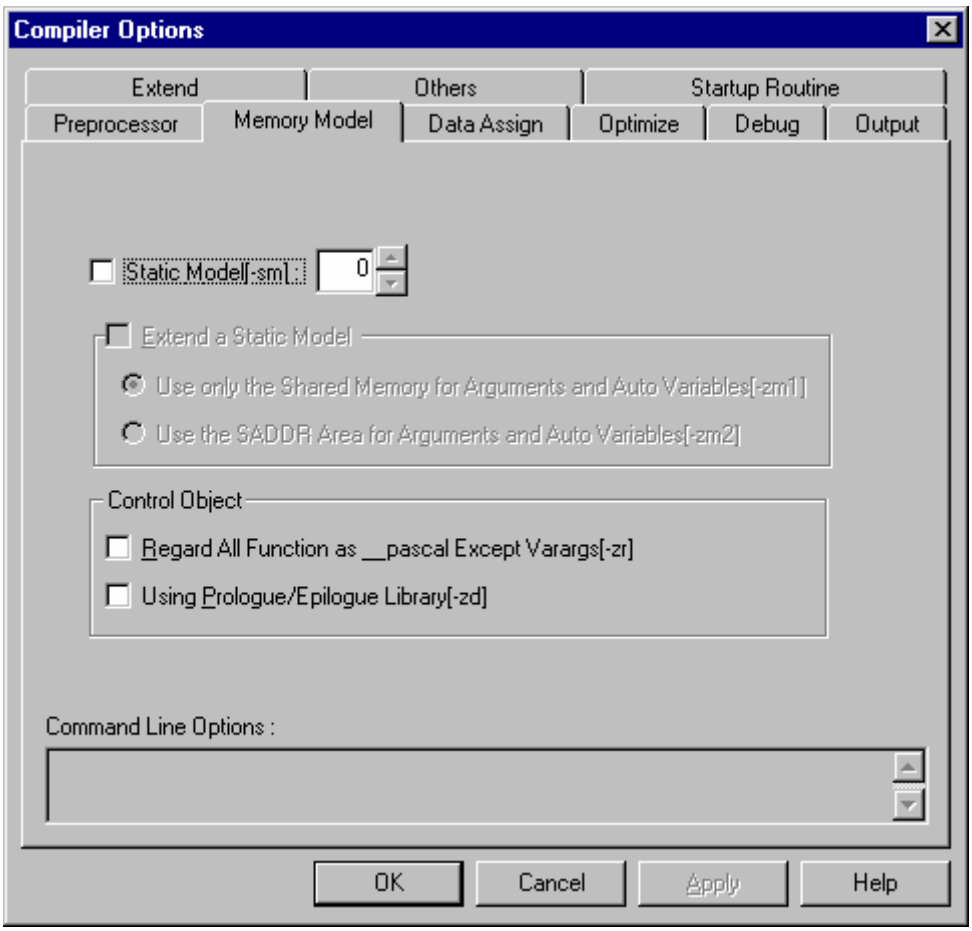


- 定义宏[-d]:
  需要用-D 选项来指定的宏名称和定义名，输入组合框即可。
  对于宏名称来说，可以用“，”来区分多个宏定义。

- 未定义宏[-u]:
  输入组合框中的宏定义都用-U 选项来指定。
  对于宏名称来说，可以用“，”来区分多个宏定义。

- 包含查找路径[-i]:
  在这个组合框中输入的内容将被[-i]选项指定为包含文件的目录。
  可以用“，”来指定多个目录。
  也可以通过[浏览]按钮来指定目录。
  不能指定不存在的路径。

(2) 选择“Memory Model”时的界面

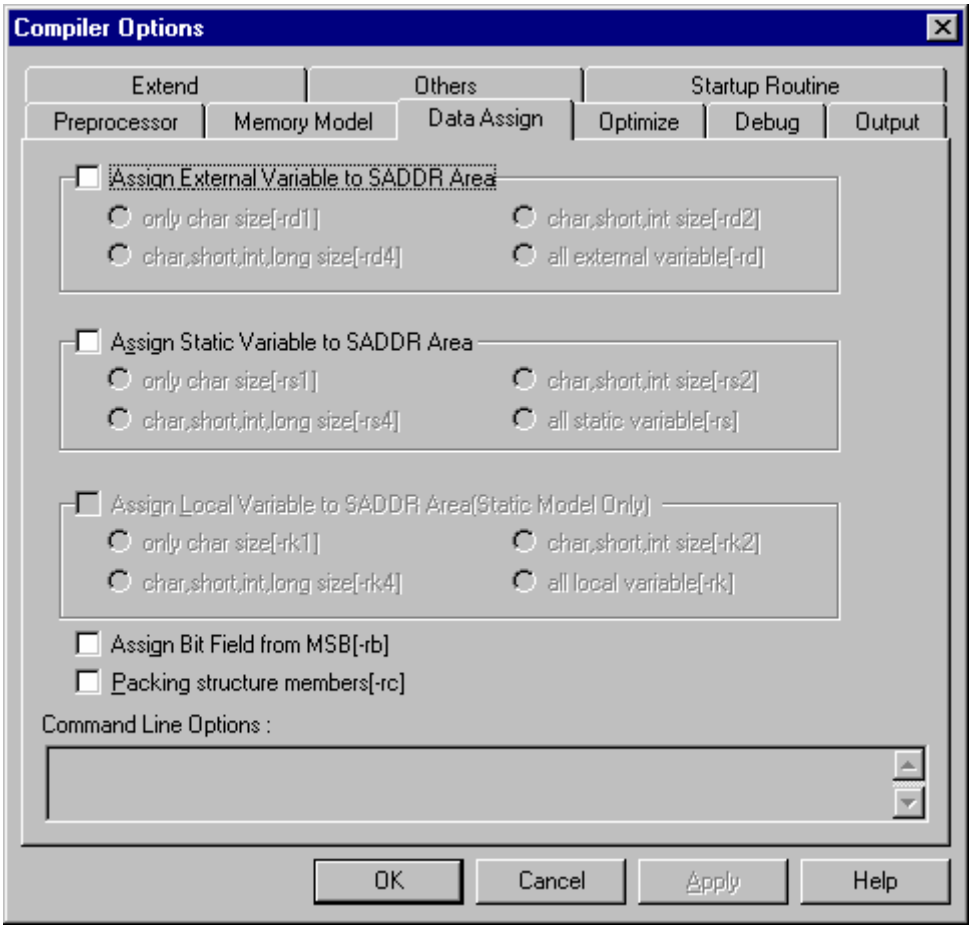


- 静态模式[-sm]:
  选中复选框并为公共区域指定一个数字就可以使用静态模式。

- 扩展静态模式
  如果指定了-SM 选项，并希望扩展静态模式，请选择这个复选框。
  点击适当的单选按钮，即可以选择参数和变量的存储区域。
  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 形式，除了 Varargs[-zr]
  选中该复选框来指定-ZR 选项。
  使用序言/尾声库 [-zd]
  选中该复选框来指定-ZD 选项。

(3) 选择“Data Assign”时的界面



- 将外部变量分配到 SADDR 区域
  选中该复选框来指定-RD 选项。
  用单选按钮可以选择分配到 saddr 区域的外部变量类型。

- 将静态变量分配到 SADDR 区域
  选中该复选框来指定-RS 选项。
  用单选按钮可以选择分配到 saddr 区域的静态变量类型。

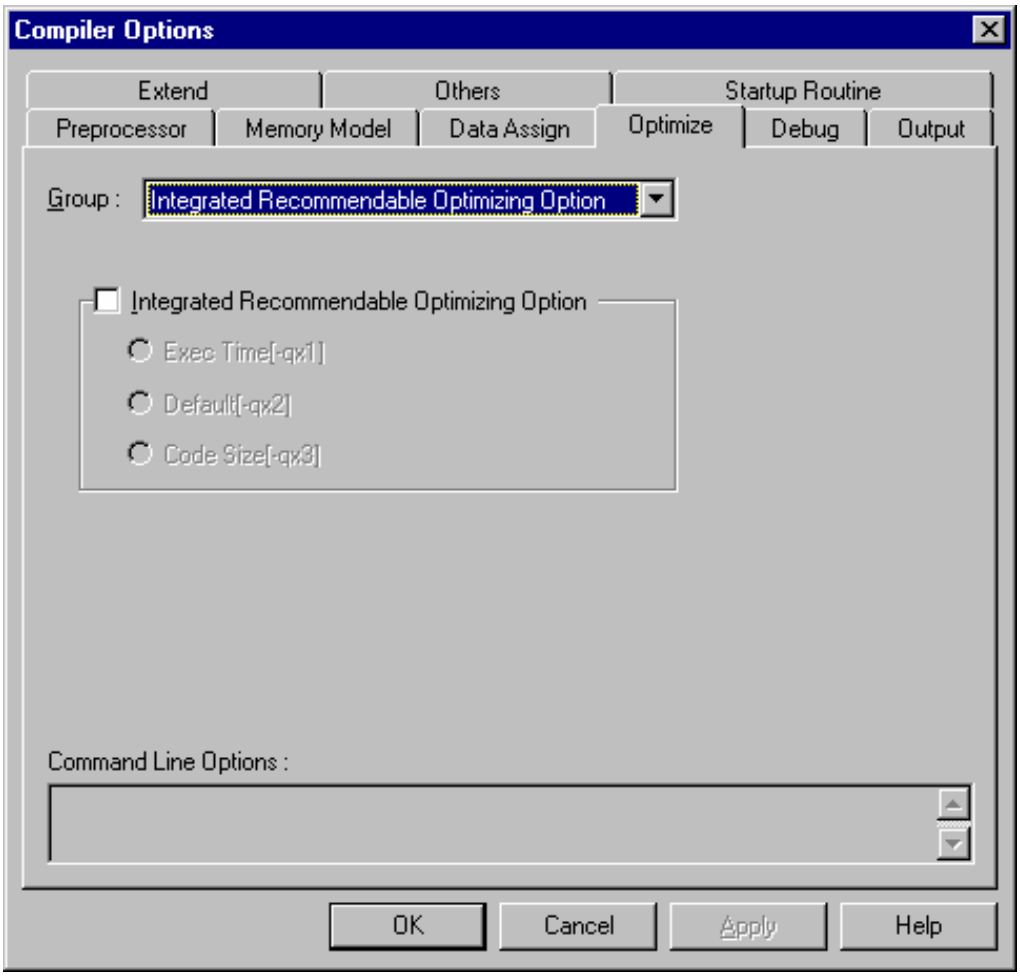
- 将局部变量分配到 SADDR 区域[仅静态模式有效]
  选中该复选框来指定-RK 选项。
  用单选按钮可以选择分配到 saddr 区域的局部变量类型。

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

- 结构成员打包[-rc]
  选中该复选框来指定-RC 选项。

(4) 选择“Optimize”时的界面

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

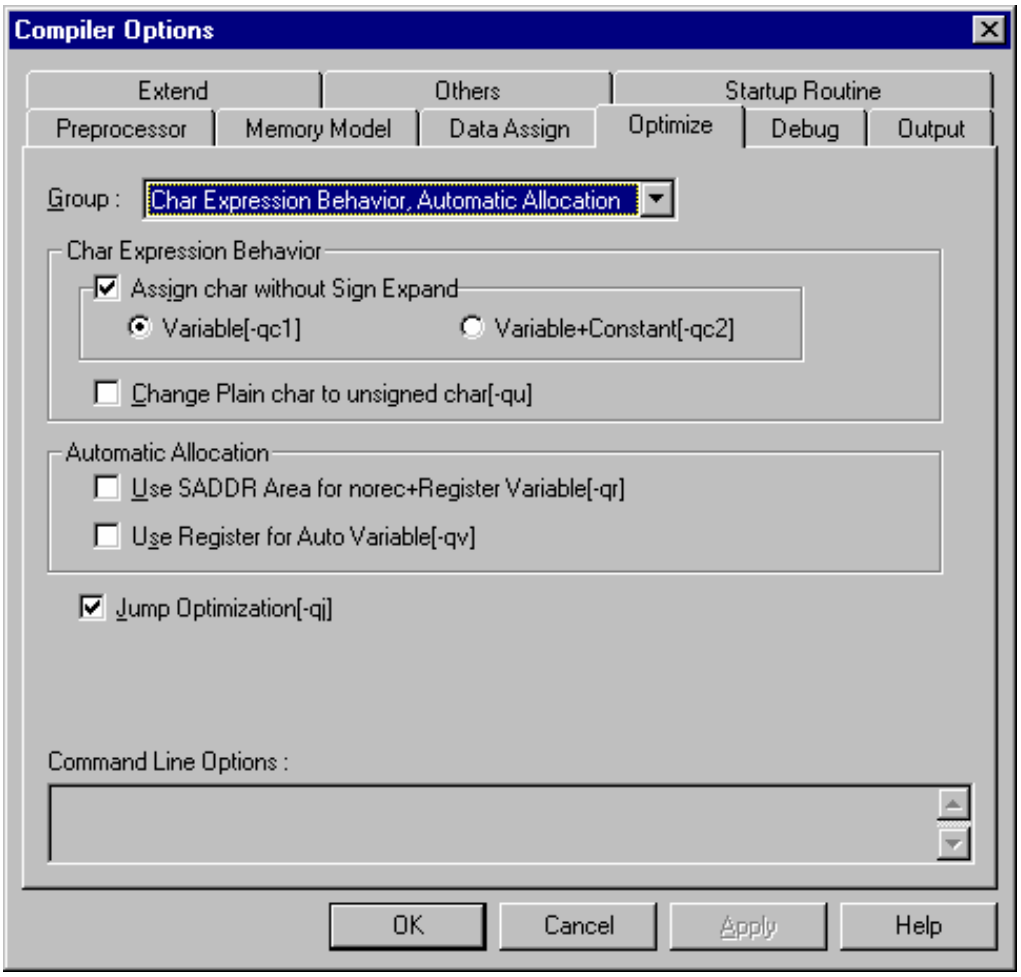


- 综合推荐优化选项

  “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。共有三个设置项：“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”时的界面如下：

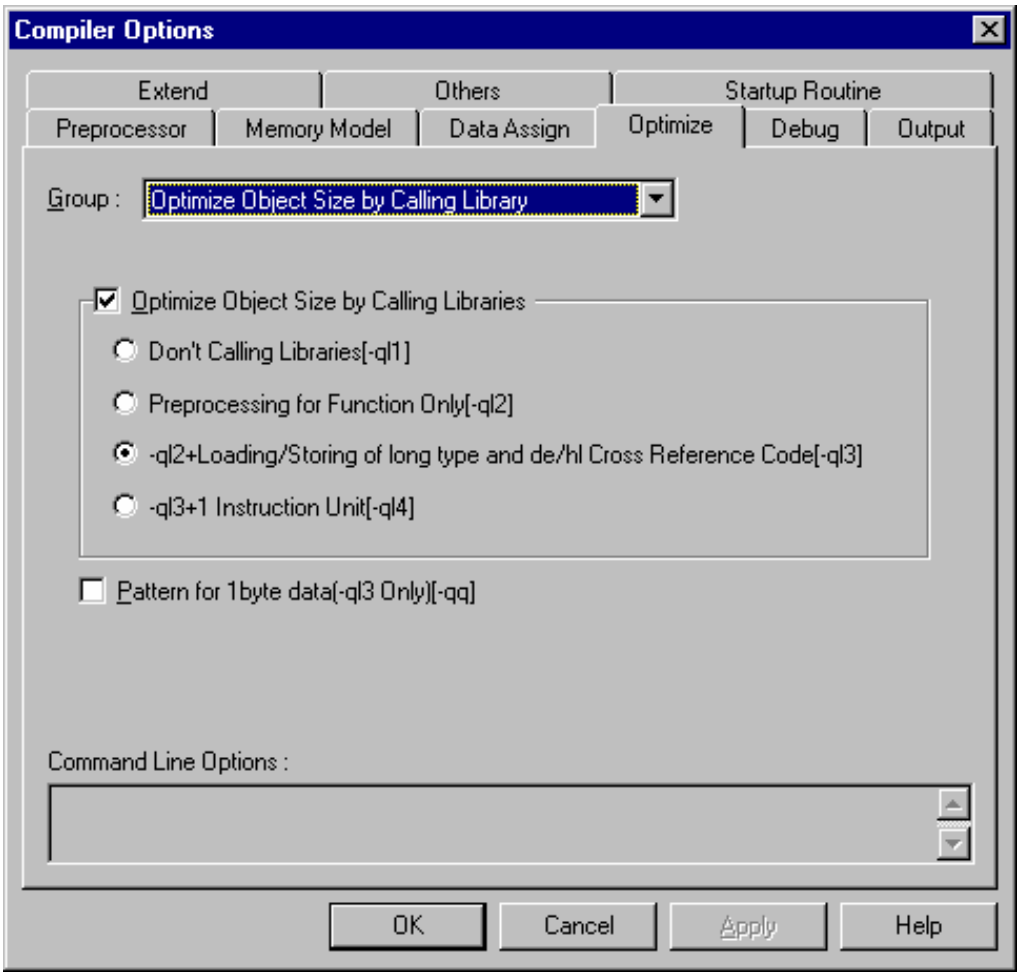


- 字符表达式动作
  指定字符为无符号扩展
  选中该复选框来指定-QC 选项。（不执行整体提升时）。
  通过点击单选按钮来选择不进行字符扩展操作的字符类型。
  将无格式字符型变换为无符号字符型[-qu]
  选中该复选框来指定-QU 选项。

- 自动分配
  Norec 函数+寄存器变量可以使用 SADDR 区域
  选中该复选框来指定-QR 选项。
  自动变量使用寄存器传递[-qv]
  选中该复选框来指定-QV 选项。

- 跳转优化[-qj]
  选中该复选框来指定-QJ 选项。

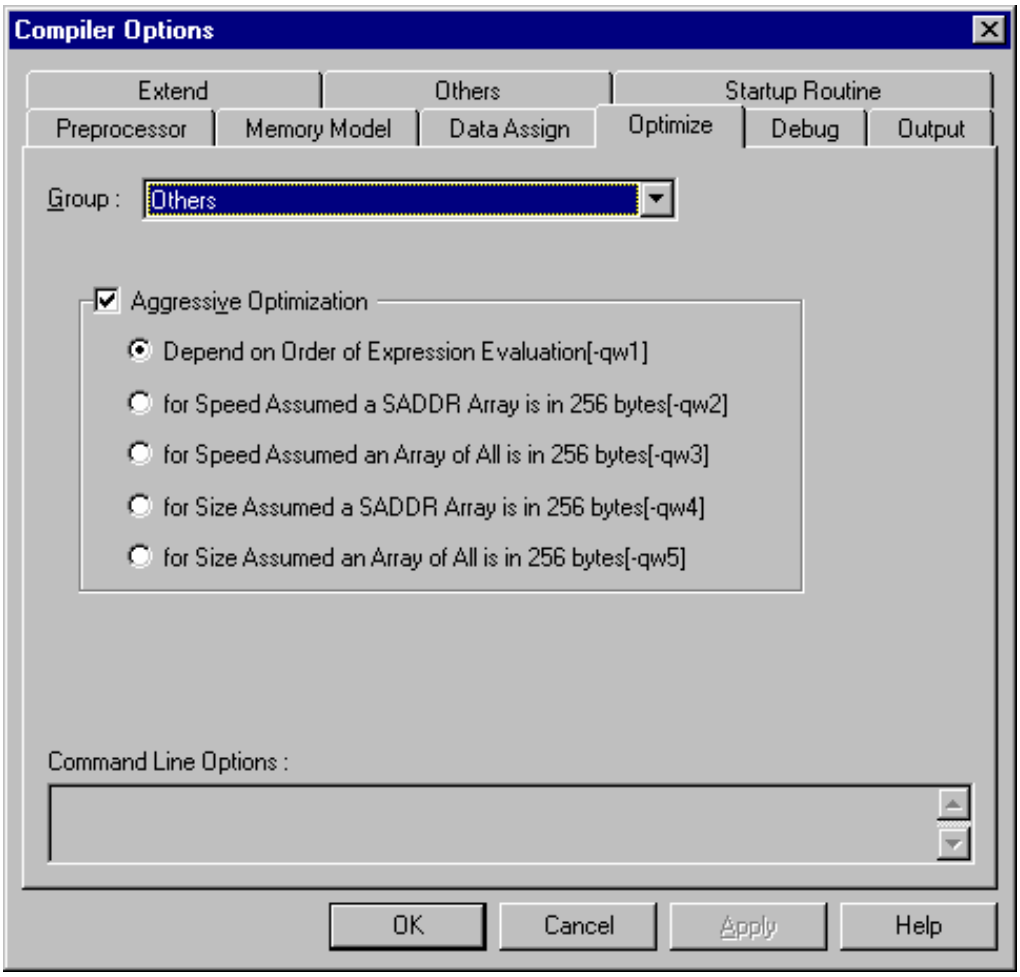
(c) 在[Group:]下拉菜单中选择“Optimize Object Size by Calling Library”时的界面如下：



- 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。

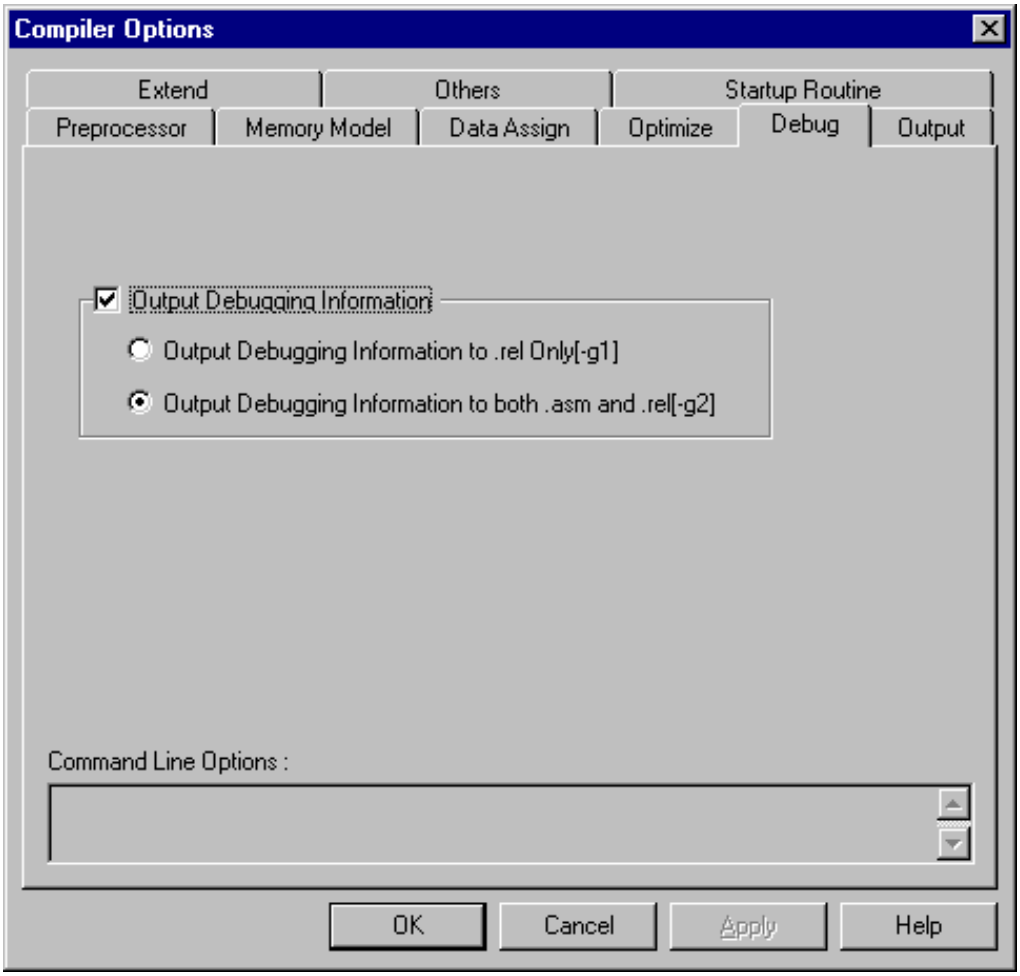
- 单字节数据的模式(-ql3 Only) (-qq)
  选中该复选框来指定-QQ 选项。

(d) 在[Group:]下拉菜单中选择“Others”时的界面如下：



- 积极的优化
  选中该复选框来指定-QW 选项，并且在四个单选按钮中任选一个来指定速度或代码大小的优先级。

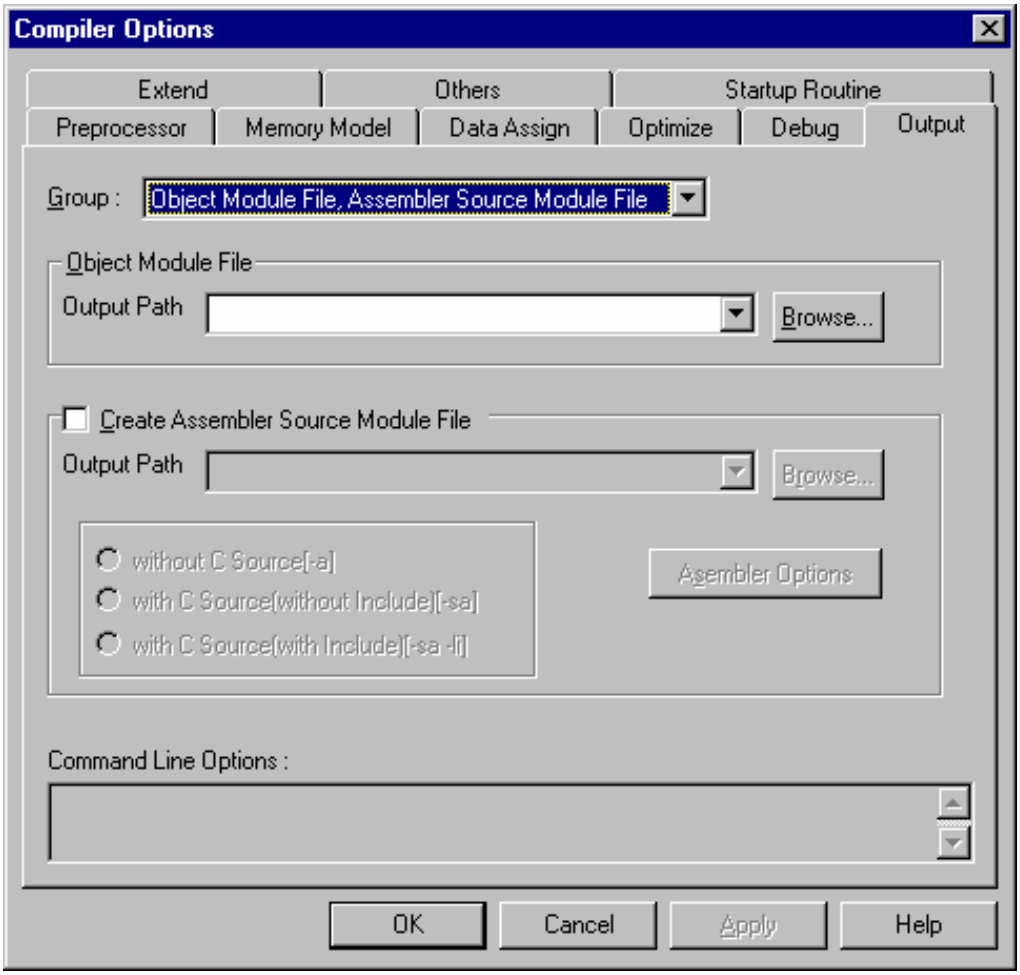
(5) 选择“Debug”时的界面如下：



- 输出调试信息
  选中该复选框来指定-G 选项，并且通过点击单选按钮选择一个存放调试输出信息的文件。如果 PM+中的[Debug]失效，那就不可能在<Debug>对话框里进行设置，并且也不会输出调试信息。

(6) 选择“Output”时的界面

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

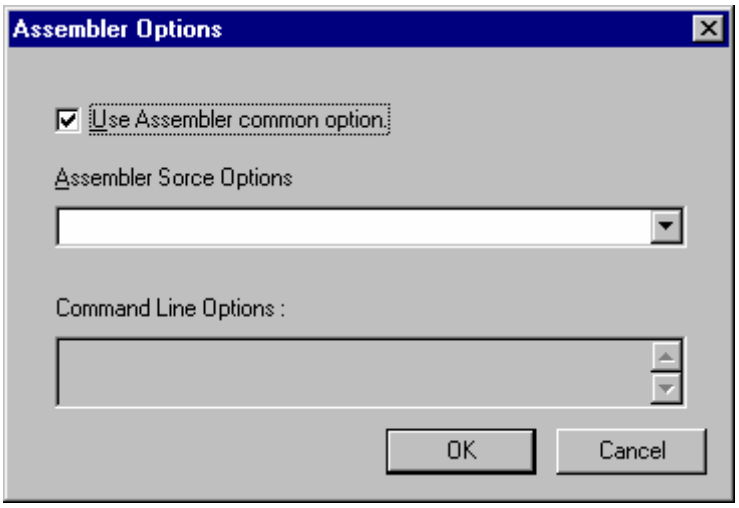


- 目标模块文件
  为了指定目标模块文件输出路径，在组合框里输入具体路径。也可以使用[Browse...]按钮来指定。
  在 PM+里指定了通用选项时，总是默认假定路径名已经指定。
  指定了源文件后，如果这个路径确实存在，则按照这个路径名来进行处理；如果路径不存在，则被当做文件名来进行处理。

- 创建汇编源模块文件
  选择这个复选框来使能-A/-SA/-LI 选项。可以选择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。在组合框里输入汇编程序源模块文件的输出路径，也可以使用[Browse…]按钮来指定路径的名称。
  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。



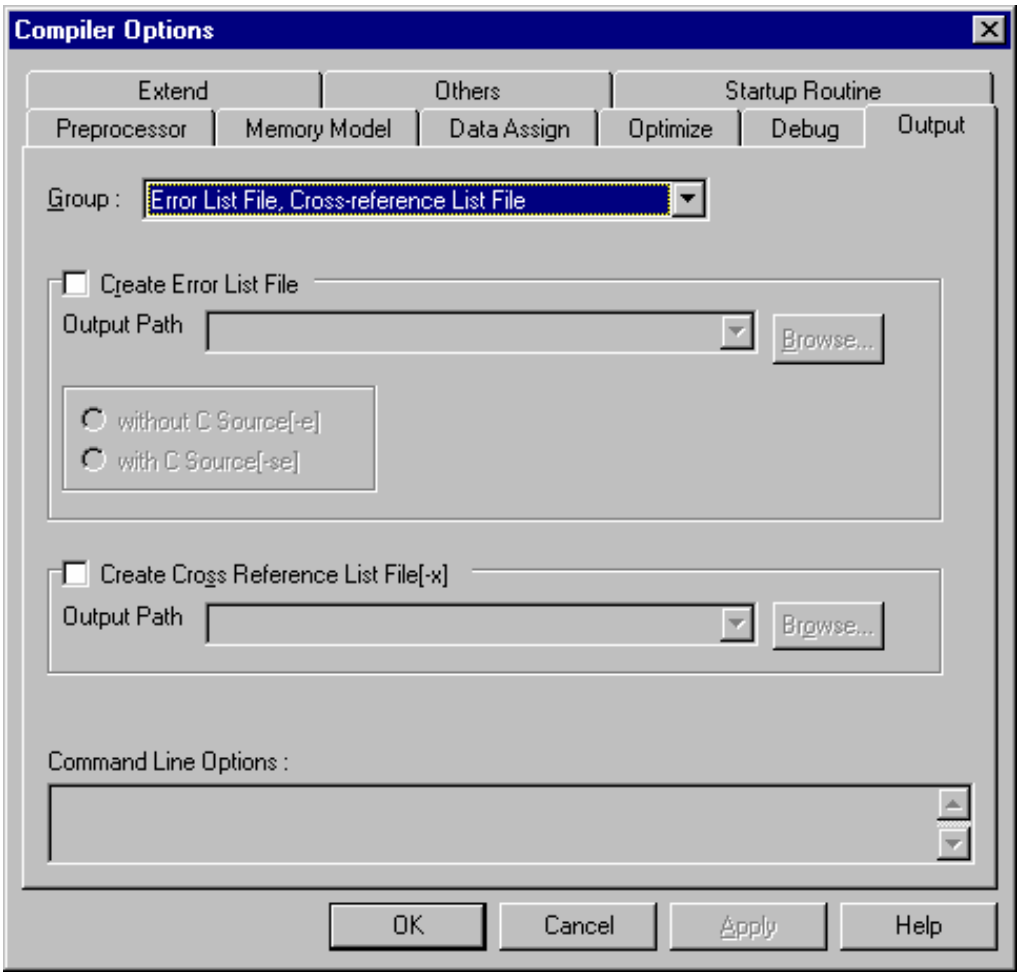
- 使用共用汇编器选项
  选择这个复选框，来使能所有<Assembler Options>对话框中的设置。

- 汇编器源文件选项
  使能编译器的输出汇编源文件选项，在组合框里输入的字符串必须包括选项名称。
  可以点击组合框右边的 按钮来选择以前曾经输入过的信息。

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

- 命令行选项：
  此编辑对话框是只读对话框。
  此编辑对话框中的字符串显示当前的选项。
  如果字符串太长，此编辑对话框无法容纳，，那么可以通过滚动按钮 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。

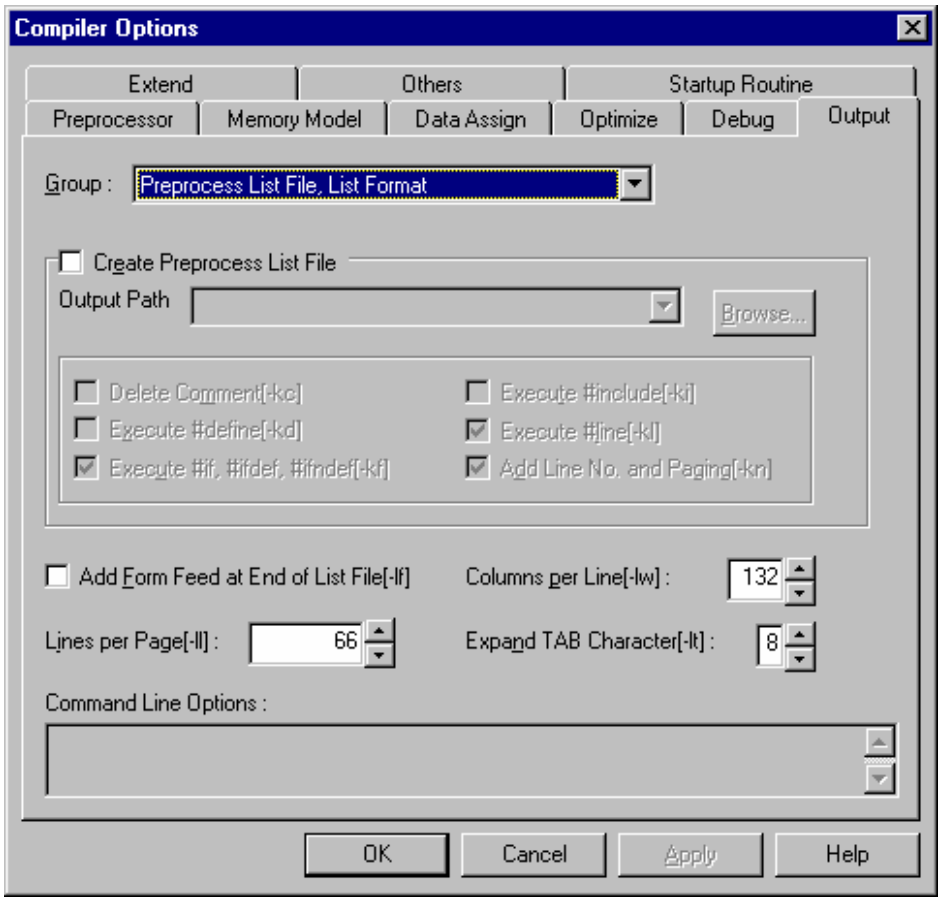
(b) 在[Group:]下拉菜单中选择“ Error List File, Cross-reference List File”时的界面如下：



- 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。
  在组合框里输入具体路径来指定错误列表文件的输出路径，也可以通过[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。在组合框里输入具体路径来指定交叉引用列表文件的输出路径，也可以通过[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”时的界面如下：



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

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

  在组合框里输入具体路径来指定预处理列表文件的输出路径，也可以通过[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。

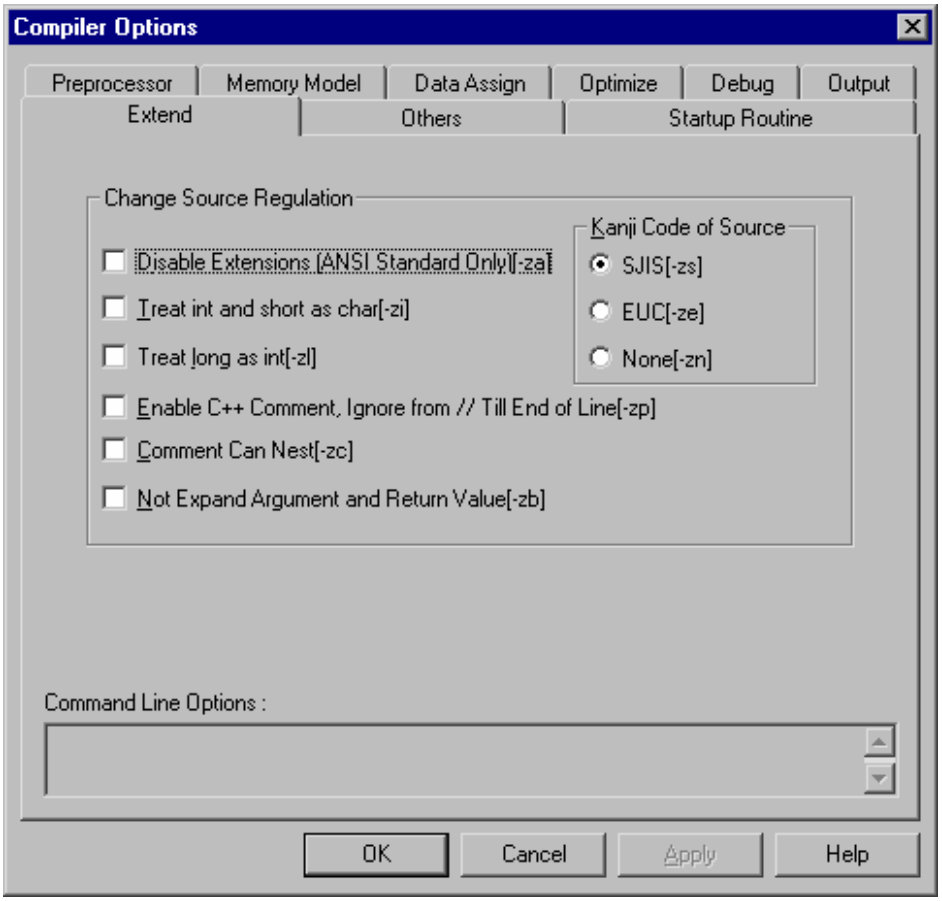
- 在列表文件的最后添加换页符[-lf]
  选中该复选框来指定-LF 选项。
- 列表设置
  当每一个列表选项设置好后，按照下列指定格式输出列表。

  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  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  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  按钮。

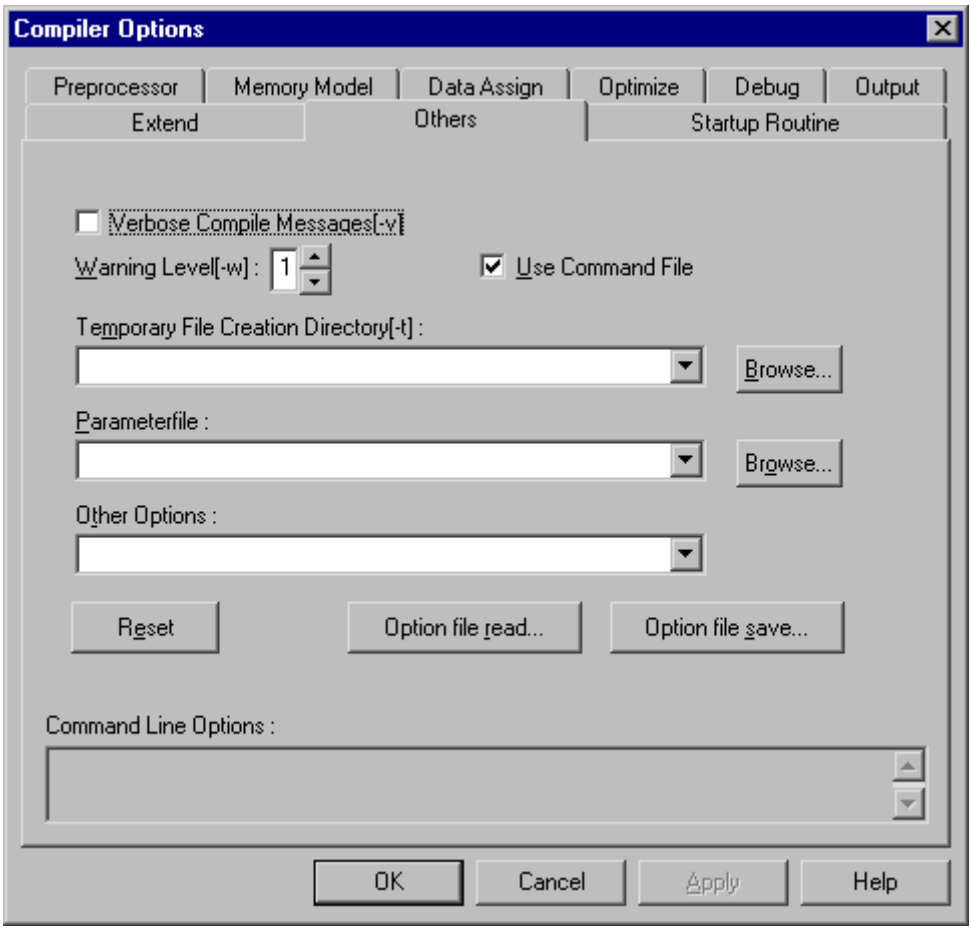
(7) 选择 “Extend”时的界面



- 改变源文件的规则

禁止扩展(仅符合 ANSI 标准)[-za]
选中该复选框来指定-ZA 选项。
把整型和短整型当作字符型处理[-zi]
选中该复选框来指定-ZI 选项。
把长整型看作整型来处理[-zl]
选中该复选框来指定-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]
选中该复选框来指定-ZP 选项。
注释可以嵌套[-zc]
选中该复选框来指定-ZC 选项。
没有扩展参数和返回值[-zb]
选中该复选框来指定-ZB 选项。
源文件中的汉字
选择对应的按钮来指定在源文件的注释中使用的汉字编码类型。

(8) 选择“Others”时的界面



- 冗余编译信息[-v]
  选中该复选框来指定-V 选项。

- 警告级别[-w]：
  用 按钮改变-W 选项的等级。

- 临时文件存放目录[-t]:
  用 -T 选项将组合框中输入的目录指定为临时文件的存储目录。

- 参数文件：
  用 -F 选项将组合框中输入的名称指定为参数文件名称。
  单击对话框的右边的 按钮来查看以前曾经输入的信息。

- 其他选项：
  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。
  可以点击组合框右边的 按钮来查看以前曾经输入的信息。

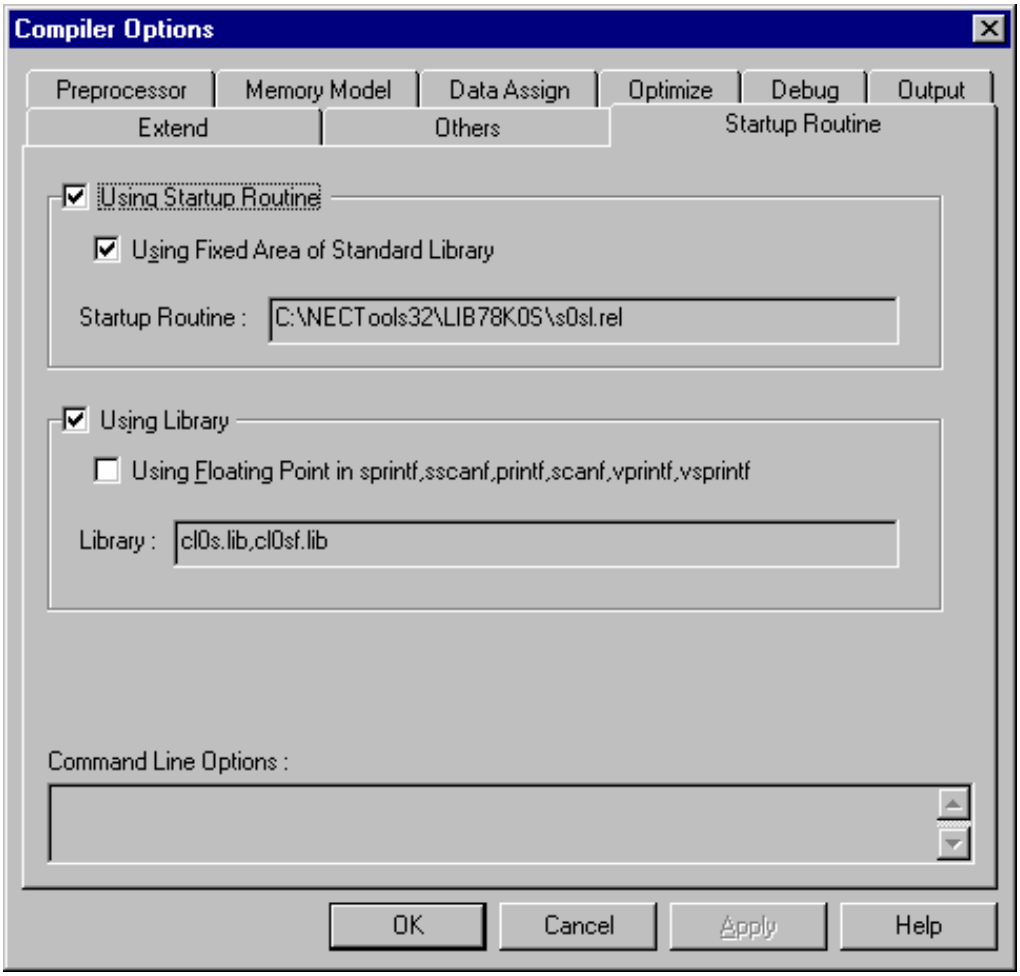
- [Reset] 按钮
  单击这个按钮设置默认选项。

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

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

- 使用指令文件
  选中这个复选框，选项字符串就会输出到命令文件中，所以无需关心选项字符串的长度。本复选框默认为选中状态。

(9) 选择 “Startup Routine”时的界面



针对单个文件时指定编译参数时，<Startup Routine>对话框无法设置。

- 使用启动例程

选择这个复选框来启用 C 编译器提供的标准启动例程。

- 使用标准库的固定区域
  选定复选框来确定标准库使用的固定区域。

- 启动例程：
  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 中使用浮点

  选择这个复选框使 sprintf, sscanf, printf, scanf, vprintf 和 vsprintf 函数支持浮点。
  如果指定了[静态模式[-sm]:] 或 [Regard All Function as _ _pascal Except Varargs[-zr]] 选项，那么 sprintf, sscanf, printf, scanf, vprintf 和 vsprintf 函数不能使用浮点支持功能。

- 库：

  显示使用的库文件名。

3.2 从编译到连接的过程

3.2.1 从 PM+中 MAKE

在 PC-9800 或 IBM PC/AT 兼容机上运行的 PM+中进行 MAKE 的方法描述如下。

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

3.2.2 启动 PM+

开发的工具包正确安装之后，在 开始 按钮的“所有程序（P）”中会创建[NECTools32]菜单，并且 PM+和其他程序会注册到[NECTools32]菜单下。

在菜单中单击[PM plus]就会启动 PM+。

3.2.3 创建工程

注册一个工程首先要用 PM+进行一系列的开发操作。

注册一个工程，首先创建工程管理的工作区。关于创建工作区的过程，敬请参考 **PM+ 5.10 版本 用户手册 (U16569E)**。

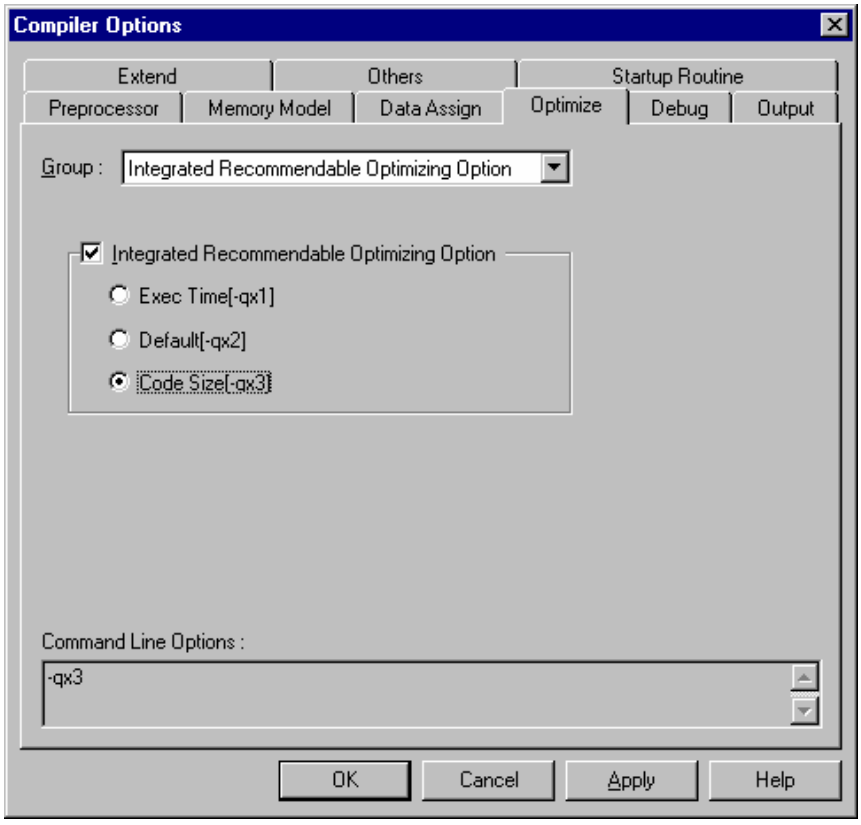
3.2.4 编译器和连接器的选项设定

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

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

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

图 3-1. 优化选项的选择



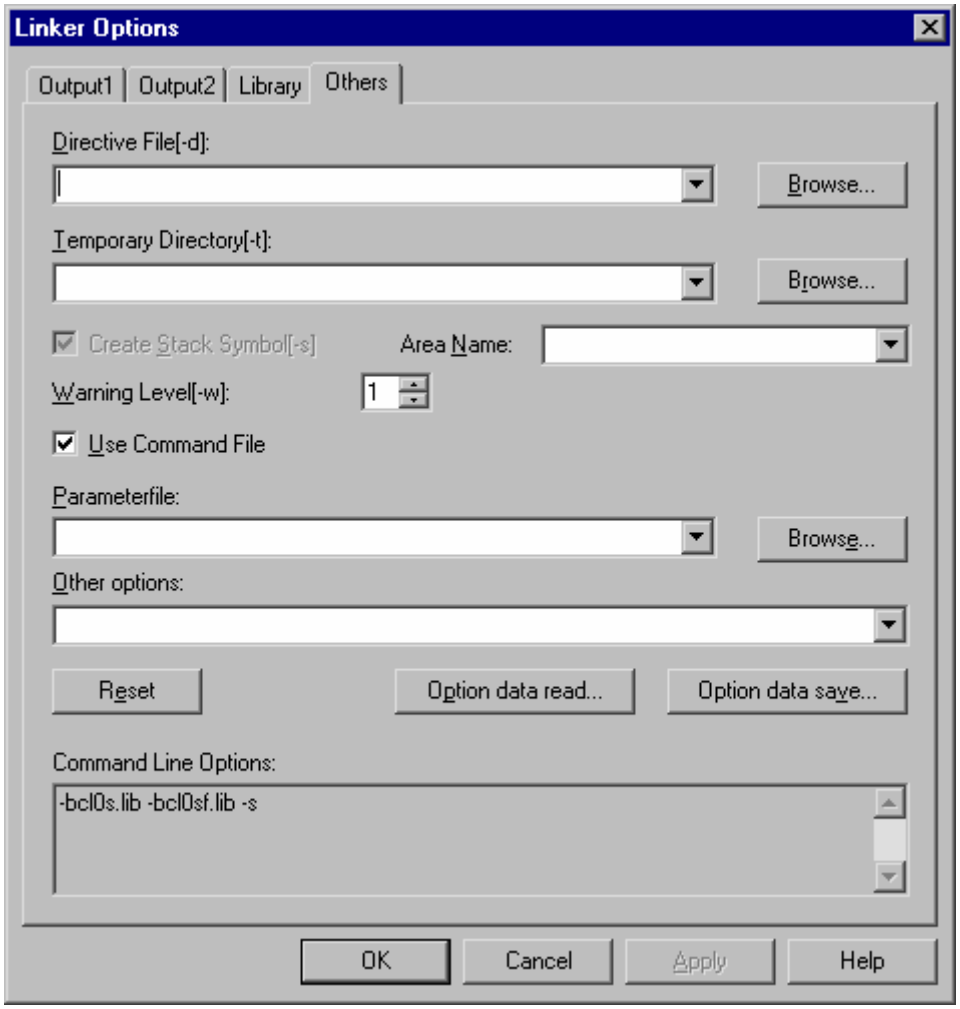
如果在<Compiler Options>对话框的<<Startup Routine>>标签页中选择了“Using Startup Routine”复选框，编译器的标准启动例程的连接在所有源文件之前进行。（不显示到 <Linker Options>对话框中）。

“使用库”是选中的情况下，编译器的标准库连接次序在所有库之后。

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 3-2. 连接器选项对话框



3.2.5 建立[BUILD]工程

工程的建立要在设定好选项条件下进行。

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

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

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

3.2.6 使用命令行来编译连接（对 DOS 提示符和 EWS）

(1) 没有使用参数文件时

下列指令用来启动 CC78KOS，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] ΔC    source name[Δoption]
>[path name]RA78K0S[Δoption] Δassembler source name[Δoption]
>[path name]LK78K0S object module name[ Δ option]
```

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

当 using sprintf, sscanf, printf, scanf, vprintf, 和 vsprin 支持浮点功能
1. **用户程序库文件（用-B 选项指定）**
2. **C 编译器附带的浮点库文件（用-B 选项指定）**
3. **C 编译器附属的库文件（用-B 选项指定）**

(其他文件的说明次序)
1. **CC78K0S 附带的启动例程的目标文件**
2. **用户程序的目标模块文件**

下面是一个连接 C 源程序 s1.c 和汇编程序源文件 s2.asm 的例子(指定为静态模式时)。

```
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 s0ssl.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）。详细信息请参见说明书**第五章编译选项**。

-i 选项，-b 选项指定的路径和 -y 选项可以根据条件进行省略。详细信息请参见**第 5 章编译器选项**和 **RA78K0S 汇编程序包操作用户手册 (U16656E)。**

(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。

当使用参数文件，在命令行指定参数文件的设定选项。

注意 参数文件不能通过 PM+的选项设定来指定。

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。

```
例子        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实例。

(para.pcc 的内容)

```
-c9024 s1.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 –y选项说明可以根据条件进行省略。详细信息请参见**第5章 编译器选项**和**RA78K0S 汇编程序包用户手册 操作篇(U16656E)。**

3.3 C 编译器的输入/输出文件

CC78K0S 的输入文件是 C 语言编写的模块文件，这些文件被转换为机器语言，再输出为目标模块文件。

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

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

CC78K0S 输入/输出文件显示如下。

表3-1. C编译器I/O文件

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

注 以下四种文件类型对于错误列表文件来说是有效的。

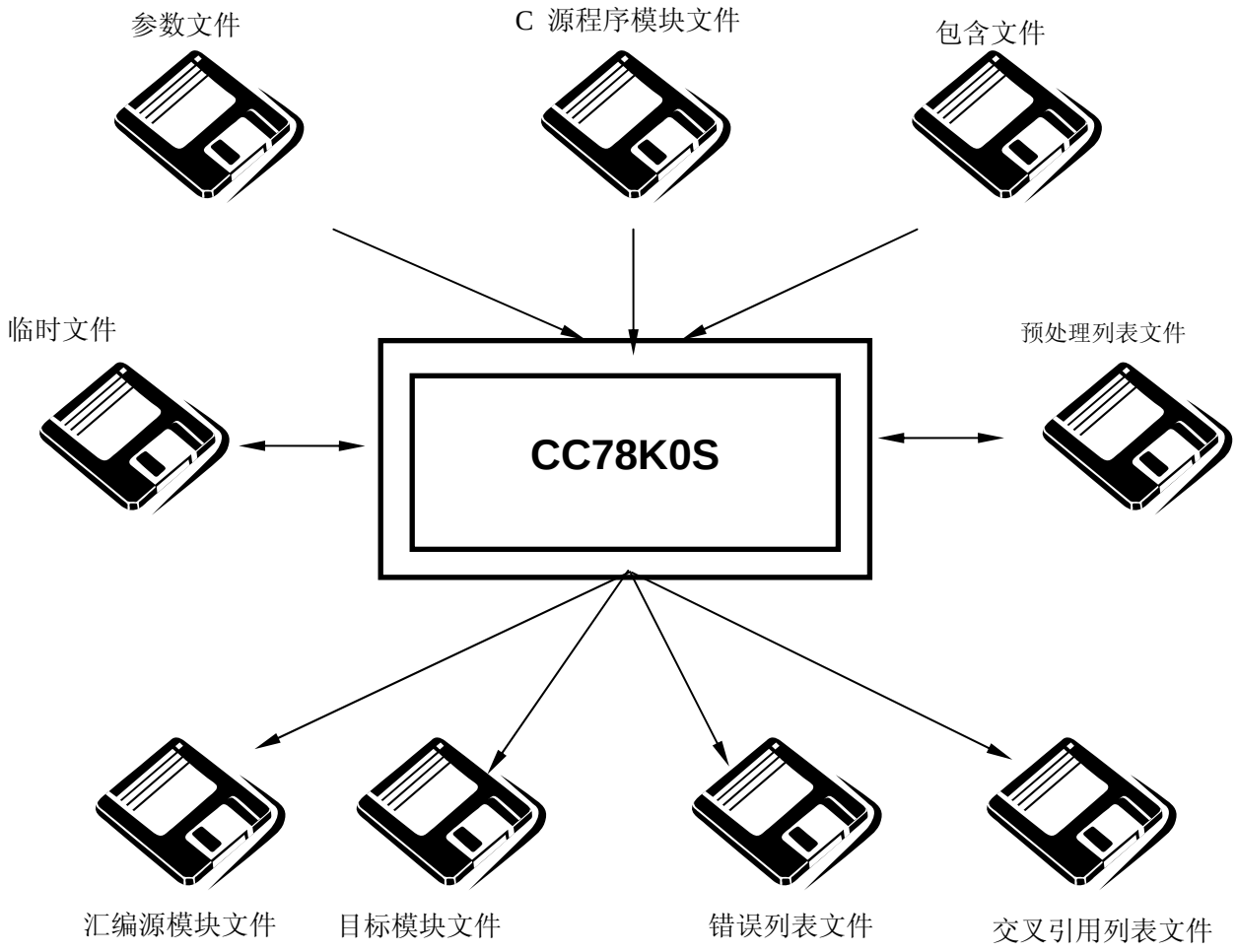
CER: 与*.C 文件一致并带有 C 源程序的错误列表文件（由指定的-SE 选项输出）

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 3-3. C 编译器的输入/输出文件



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

3.4 执行开始和结束信息

(1) 执行开始信息

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
```

(2) 执行结束信息

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

```
Target chip : uPD789xxx
Device file : Vx.xx

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

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

```
PRIME.C(18) : W745 Expected function prototype
PRIME.C(20) : W745 Expected function prototype
PRIME.C(26) : W622 No return value
PRIME.C(37) : W622 No return value
PRIME.C(44) : W622 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 -m

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

A018 Option is not recognized '-m'
Please enter ‘ CC78K0S   -- ‘ , if you want help messages.
Program aborted.
          . .
           ..
           ..
```

在这个例子中，因为输入了一个不存在的编译选项，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 中优化就是为了创建高效的目标模块文件。**表 4-1 优化方法**列出了可以支持的优化方法。

表 4-1. 优化方法(1/2)

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

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

	<10>	删除未使用的指令	a=a; → 删除 在 a=b;语句之后，a 没有被引用过 → 删除 (a 是一个自动变量)
	<11>	删除拷贝	a=b; c=a+d; → c=b+d; a 不再被引用（a 是一个自动变量 ）。
	<12>	改变表达式中的计算顺序	在其他计算执行之前，保留在寄存器中的计算结果仍有效。

表 4-1. 优化方法(2/2)

阶段		内容	例子
优化器	<13>	存储器设备的分配(临时变量)	局部变量分配给寄存器。
	<14>	优化控制	代替特殊模式 例如 a*1 → a, a+0 → a
	<15>	降低计算强度	例如 a*2 → a+a, a<< 1
	<16>	存储器设备的分配(寄存器变量)	数据分配到高速存取的存储器。 例如：寄存器，saddr 区域 (仅当指定了-QR 选项时)
	<17>	跳转优化 (-QJ 选项)	将连续跳转指令组合成一条指令。
	<18>	寄存器的分配(-QV/-QR/-RD/-RK/-RS 选项)	变量自动分配给寄存器。

注意 无论优化选项如何设置，<1> 到 <7>项都会执行。

<8> 到<13>以及<17> 和<18>项只有在指定了对应的优化选项时才会执行。

<8> 到<13>项的优化选项在未来会继续改进。

无论优化选项如何设置，<14> 到 <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。但是，只有在指定了-QR 选项时才能使用 saddr 区域。

关于优化选项的信息，参见**第 5 章 编译程序选项。**

4.2 ROM 化函数

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，敬请参阅 **8.3 启动例程。**

下面描述如何将程序存储在 ROM 中。

启动例程是使用 s0ss.rel 的例子来描述的(当需要 ROM 化处理过程时，就会使用这个 rel 文件；如果指定了静态模式则需要使用标准库)。

4.2.1 连接

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

(1) s0sl.rel: 启动例程(当存储在 ROM 中)
包括数据初始化的拷贝过程，并指示初始数据的起始地址。
标签_@cstart (符号)被当作是开始地址。

(2) cl0s*.lib: CC78K0S 附属库。CC78K0S 库文件包括以下两种。

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

<2> 标准库
_（下划线）添加到标准库名称的符号最前面。

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

注意 CC78K0S 提供了各种启动例程和库。关于启动例程的细节，请参阅第 8 章 启动例程。关于库的细节，请参阅 2.6.4 库文件。

第 5 章 编译选项

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

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

5.1 编译选项的指定

编译选项可以通过以下几种方法进行指定。

(1) 当 C 编译器启动时再命令行中指定。
(2) 在 PM+的< Compiler Options>对话框中指定。
(3) 在参数文件中指定。

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

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

例 (Δ: 空白，例如空格)

```
CC78K0SΔ-c9024Δprime.cΔ-aΔ-qx3
```

5.2 编译选项的优先级

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

表5-1. 编译选项的优先级

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

←水平方向

↑

垂直方向

[×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。

[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。

[O 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 -g

-G 选项无效。

例 2	C>cc78k0s –c9024 -e sample.c -p -k

由于-P 选项被指定，-K 选项有效。

例 3	C>cc78k0s –c9024 -e sample.c -utest -dtest=1

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

因为-NO 选项是最后指定，所以-O 选项无效，-NO 选项优先。

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

这一部分详细描述编译选项。

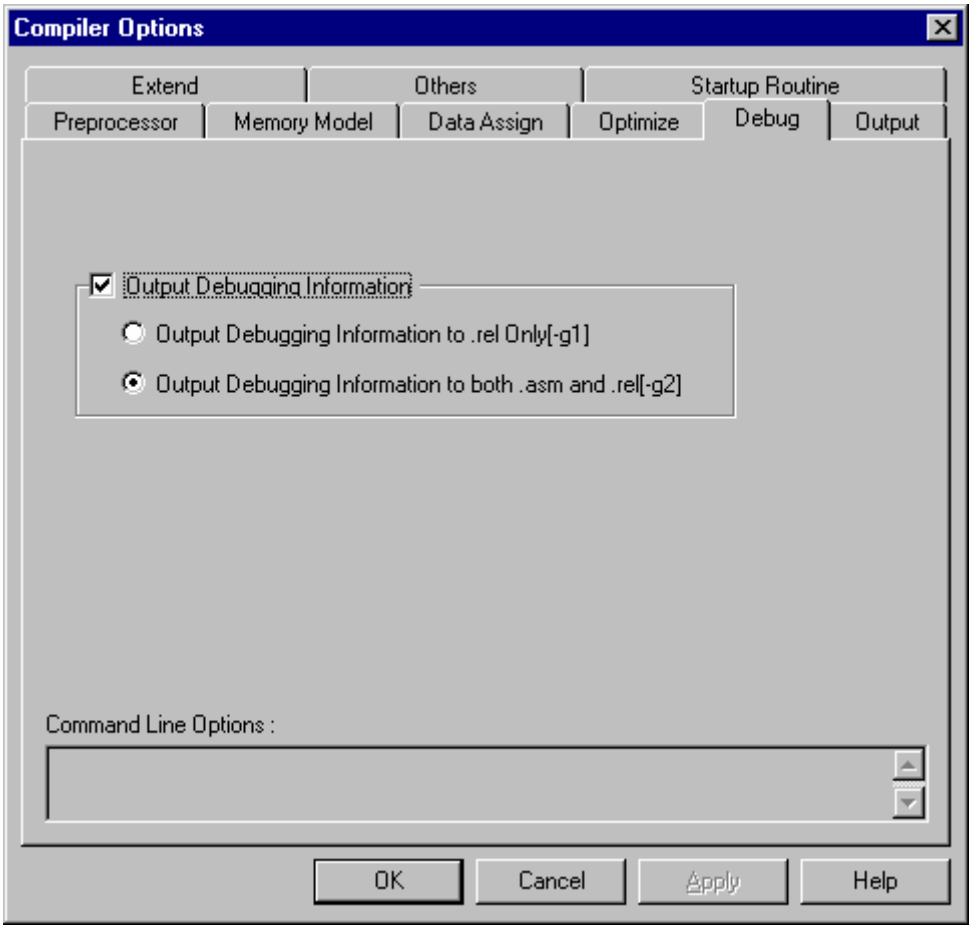
这个例子展示了如何从命令行中启动 CC78K0S。为了在 PM+中启动 CC78K0S，需要指定命令，指定设备类型，并在<Compiler Options>对话框中指定 C 源程序遗漏的选项。

范例 在命令行状态时

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

范例 使用 PM+时

图 5-1. 编译选项对话框



(1) 设备类型说明(-C)

-C	设备类型说明

格式描述　-C 设备类型 默认解释　无

[功能]
-C 选项为编译过程指定了目标设备。

[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。

[描述]
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，必须先安装设备文件。需要复制设备文件到 BIN 目录中或者 DEV 目录。

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

```
#pragma    pc (设备类型)
```

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

-C	设备类型说明

[使用范例]

在命令行中指定。目标设备是μPD789024。

```
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;
            M
```

因此，在命令行中也可以省略目标设备说明。

```
C>cc78k0s   prime.c
```

-C 设备类型说明

在 C 源文件和命令行中指定了不同的设备，并启动编译器。

C 源程序

```
#pragma          pc(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) :    W832 Duplicated chip specifier
 sample\prime.c(18):    W745 Expected function pro 至 type
 sample\prime.c(20):    W745 Expected function pro 至 type
 sample\prime.c(26):    W622 No return value
 sample\prime.c(37):    W622 No return value
 sample\prime.c(44):    W622 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 (-O/-NO)

-O/-NO	目标模块文件创建说明

```
描述格式    -O [输出文件名]
            -NO
默认解释    -O [输入文件名.rel]
```

[功能]

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

[应用]

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

[描述]

如果出现编译错误，即使-O 选项已经被选定，目标模件文件也仍然无法输出。
当-O 选项被选定时，如果没有指定驱动器名称，目标模件文件会输出到当前驱动器。
如果-O 选项与-NO 选项二者同时被选中，那么最后选择的选项有效。

[注意]

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

[使用范例]

在这个实例里，-NO 和-O 选项都被选中。

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

(3) 存储器分配说明 (-R/-NR, -RD/-NR, -RK/-NR, -RS/-NR, -RC/-NR)

-R/-NR 存储器分配说明

格式描述 -R [处理类型] (多种可能的说明)
-NR
默认说明 -NR

[功能]
-R 选项指定了如何将一个程序指派到存储器中。
-NR 选项表示-R 选项无效。

[应用]
如果想要指定一个程序在存储器中如何分配，选择-R 选项即可。

[描述]
下面列出能够用-R 选项指定的处理类型。处理类型说明不能被忽略。否则，会出现异常中断错误(A012)。

处理类型	功能
B	分配一个位域从最高有效位开始存放 (MSB)。
D[n] (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] (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] (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。
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。

注意 可以指定多种处理类型。

当指定-NR 选项时，处理类型的含义如下。

处理类型	功能
B	分配一个位域从最低有效位开始存放 (LSB)。
D	不会将任何变量自动分配到 saddr 区域。
K	不会将任何变量自动分配到 saddr 区域。
S	不会将任何变量自动分配到 saddr 区域。
C	不要对任何结构体成员打包压缩。

[使用范例]

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

-RD/-NR 内存分配说明

格式说明	-RD[n] (n = 1, 2, 4) -NR
默认说明	-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 选项无效。

[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。

[描述]

待分配的变量类型会根据 n 的值而改变。

N 的值	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，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，枚举型，指针型，长型，无符号长型
忽略	所有变量 (包括结构体，共用体和数组)

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。

-RK/-NR	内容分配说明

格式描述	-RK[n] (n = 1, 2, 4) -NR
默认说明	-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 选项无效。

[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。

[描述]
待分配的变量类型会根据 n 的值而改变。

N 的值	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，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
忽略	所有变量(包括结构体，共用体和数组)

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]
这个选项只在-SM 选项被选中后有效，如果没有指定-SM 选项，那么会输出警告信息且-RK 选项被忽略。

-RS/-NR 内存分配说明

格式描述	-RS[n] (n = 1, 2, 4) -NR
默认说明	-NR

[功能]

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

[应用]

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

[描述]

待分配的变量类型会根据 n 的值而改变。

N 的值	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，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
忽略	所有变量(包括结构体，共用体和数组)

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(-Q/-NQ)

-Q/-NQ	优化说明

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
默认说明 -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 选项无效。

[应用]

如果想要改善目标的执行速度并减少代码大小，请指定-Q 选项。如果已经指定了 -Q 选项，又想要同时执行多种优化，就可以连续地指定多种优化类型。具体细节请参阅表 5-2 优化类型。

[描述]

表 5-2 列出-Q 选项能够设定的优化类型。

表5-2. 优化类型(1/2)

优化类型	过程描述
无说明	默认为 -QCJLW。
U(-QU 选项)	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) (-QC 选项)	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]。 范围的改变依靠 n 的值来控制，具体如下。如果 n 值被忽略，将默认为 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
J(-QJ 选项)	优化转移指令。
X[n](n = 1 到 3) (-QX 选项)	根据执行速度/代码量的优先级来自动设置优化选项。 根据 n 值的不同选择不同的选项，具体如下。如果 n 被忽略,将默认 n = 2。 1: 速度优先。 认为指定了-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。
Q	切换到专门为单字节库。
R	添加寄存器变量到寄存器，并将寄存器变量分配到 saddr 区域。

-Q/-NQ	优化说明

表 5-2. 优化类型(2/2)

优化类型	过程描述
W[n] (n = 1 到 5)	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，在源程序符合要求的情况下这个问题可以避免。 根据 n 值的不同来选择不同的作用域，具体如下。如果 n 值被忽略，将默认为 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。假定在使用无符号字符变量引用 saddr 边界对齐数据时不产生进位，则可以通过计算低位字节的地址来实现速度优化。saddr 边界对齐数据包括字符型/无符号字符型/短型/无符号短型/整型/无符号整形数组型等类型。 3: 可以申请将第 2 项存储到 saddr 区域之外的其他位置。 4: 改变表达式内部的执行次序。假定在使用无符号字符变量引用 saddr 边界对齐数据时不产生进位，则可以通过计算低位字节的地址来实现代码量的优化。saddr 边界对齐数据包括字符型/无符号字符型/短型/无符号短型/整型/无符号整形数组型等类型。 5: 可以申请将第 4 项存储到 saddr 区域之外的其他位置。
V (-QV 选项)	将一个自动变量自动分配到寄存器或 saddr 区域。
L[n] (n = 1 到 4) (-QL 选项)	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 = 3。 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.:在第 3 项基础上，增加了单字节指令。

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 至 127，0x00 至 0x7F，00 至 0177 字符类型
128 至 255, 0x80 至 0xFF, 0200 至 0377 无符号字符类型
0U 至 255U 无符号字符类型
‘\0’ 至 ‘\377’ 字符类型

-Q/-NQ	优化说明

但是，当-QU 选项被指定时，从'\200'到'\377'范围内的字符常量都被当作无符号字符型常数来处理，同时值的范围从+128 到+255。

带–(负号)的常量处理方法如下。

```
–0 至 128  字符型
从–129          整型
```

如果常量或变量的计算结果溢出，用某个可以表示计算结果的类型来暂代常量或变量。通过暂代或指定-QI 选项，数据类型的改变可以避免。当指定-QC1 选项时，常量计算支持符号扩展。

```
(例)  当-QC2 选项被指定

  int  i;

  i = (int)20 * 20;  /* 400 */
```

可以指定多种优化类型。
如果-Q 选项或优化类型被忽略，优化效果完全等同于指定-QCJLW 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→请删除-QCLW)。
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。

[使用范例]

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

```
C>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(-G/-NG)

-G/-NG	调试信息输出说明

格式说明	-G[n] (n = 1, 2) -NG
系统设定值解释	-G2

[功能]

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

[应用]

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

[描述]

因为 n 值的不同操作会有所改变。

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

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

[使用范例]

指定了-G 选项。

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

(6) 预处理列表文件创建说明(-P, -K)

-P	预处理列表文件创建说明

格式说明	-P [输出文件名]
默认说明	无 (无文件输出)

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

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

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

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

[使用范例]
预处理列表文件 sample.ppl 被输出。

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

-K 预处理列表文件创建说明

格式说明	-K[处理类型] (可以指定多种规范)
默认说明	-KFLN

[功能]
-K 选项指定预处理列表的处理过程。

[应用]
当注释被删除或使用了宏定义扩展，此时输出预处理列表文件需要指定该选项。

[描述]
下表中列出了-K 选项指定的处理类型。

表5-3. –K选项处理类型

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

注意 可以同时指定多种处理类型。

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

-K	预处理列表文件创建说明

[使用范例]

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 : {
                          M
/*
  Target chip : uPD789024
  Device file : VX.XX
*/
```

(7) 预处理说明(-D，-U，-I)

-D	预处理说明

格式描述	-D 宏名[=定义名] [，宏名[=定义名]]... (可以指定多项说明)
默认说明	只有在 C 源程序模块文件中的宏定义才有效。

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

[应用]
如果需要用指定常量替换全部的宏名时，指定该选项。

[描述]
用逗号‘,’分隔每个定义内容，一次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。

[使用范例]

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

-U 预处理说明

格式描述	-U 宏名 [，宏名]...(可以指定多个宏名)
默认说明	用-D 选项指定的宏定义有效。

[功能]
-U 选项取消宏定义，效果同 C 源程序中的#undef 语句类似。

[应用]
指定了该选项，则用-D 选项定义的宏名会无效。

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

[使用范例]
通过-D 和-U 选项指定相同的宏名。在这个例子中，TEXT 宏被禁止。

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

-I	预处理说明

格式描述	-I 目录[，目录]... (可以指定多项说明)
默认说明	源文件目录注 1 环境变量 INC78K0S 指定的目录 C:\NECTools32\INC78K0S 注 2

[功能]

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

[应用]

需要从某个确定目录查找包含文件时，请指定该选项。

[描述]

用逗号‘,’分隔每个定义内容，一次可以指定多个目录。
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。

- 源文件目录注 1
- 用-I 选项指定的目录
- 用环境变量 INC78K0S 指定的目录
- 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。如果包含文件名用<>指定，无需进行查找。
2. 这个实例需要提前将 CC78K0S 安装到 C:\NECTools32 目录(Windows 版本)。

[使用实例]

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(-A, -SA)

-A	汇编源模块文件创建说明

格式描述	-A[输出文件名]
默认说明	没有汇编源模块文件输出
输出文件	*.asm (*: 字母数字符号)

[功能]

-A 选项指定汇编源模块文件的输出。另外，还可以指定输出目录和输出文件名。

[应用]

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

[描述]

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

[注意]

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

[使用范例]

创建汇编源模块文件 sample.asm 的实例。

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

汇编源模块文件输出到打印机。

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

-SA 汇编源模块文件创建说明

格式描述	-SA [输出文件名]
默认说明	没有汇编源程序输出
输出文件	*.asm (*: 字母数字符号)

[功能]

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

[应用]

如果汇编源模块文件和 C 源程序模块文件都需要输出，则请指定-SA 选项。

[描述]

磁盘文件名或者设备文件名都可以指定为文件名称。

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 源程序不会没有被加入到输出汇编源模块文件的注释中，但是如果指定了-LI 选项，则 C 源程序也会被添加到注释中。

[注意]

要在 PM+中改变输出目录，可以在<<Output>>标签下的<<Create Assembler Source Module File>>区域的<<Output Path>>组合框中选定输出目录，并选择“with C Source[without Include][-sa]”或者“with C Source[with Include][-sa -li]”。

单独指定此选项，输出文件名也会改变。

在<<Output>>标签页下的<< Output File >>组合框中指定文件名或者输出目录。

-SA	汇编源模块文件创建说明

[使用范例]

指定-SA 选项时

```
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_INF03FH, 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
                          ⋮
      EXTRN   _@cprep
      EXTRN   _@RTARG0
      EXTRN   @@isrem
      EXTRN   _@cdisp
      PUBLIC  _mark
      PUBLIC  _main
      PUBLIC  _printf
      PUBLIC  _putchar
                          ⋮
@@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++)
```

-SA 汇编源模块文件创建说明

```
$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 C:\NECTools32\SMP78K0S\CC78K0S\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 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
```

C 源程序作为注释被添加。

(9) 错误列表文件创建说明 (-E, -SE)

-E	错误列表文件创建说明

格式描述	-E [输出文件名]
默认说明	没有错误列表文件输出
输出文件	*.ecc (*: 字母数字符号)

[功能]

-E 选项指定了错误列表文件的输出。此外，还指定了输出目录或输出文件名。

[应用]

通过-E 选项，可以更改错误列表文件的输出目录和输出文件名。

[描述]

磁盘文件名或者设备文件名都可以指定为文件名称。
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]

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

-E 错误列表文件创建说明

[使用范例]

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) : W745 Expected function prototype
prime.c(    20) : W745 Expected function prototype
prime.c(    26) : W622 No return value
prime.c(    37) : W622 No return value
prime.c(    44) : W622 No return value

Target chip : uPD789024
Device file : Vx.xx

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

-SE 错误列表文件创建说明

描述格式	-SE [输出文件名]
默认说明	无错误列表文件输出
输出文件	*.cer : *.C 文件的错误列表(*: a 包含文字与数字符号) *.her : *.H 文件的错误列表 *.er : 除*.C 和 *.H 文件外的错误列表

[功能]

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

[应用]

如果错误列表文件和 C 源程序都需要输出，请指定-SE 选项。

[描述]

磁盘文件名或者设备文件名都可以指定为文件名称。

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。

不能为包含文件指定目录和文件名，如果包含文件的文件类型是“H”，那么文件类型为“her”的错误列表文件会输出到当前驱动器。如果包含文件的文件类型是“C”，那么文件类型为“cer”的错误列表文件会输出到当前驱动器。其余所有情况，都会输出文件类型为“er”的错误列表文件。如果没有任何错误，那么 C 源程序不会被添加。在这种情况下，不会为包含文件创建错误列表文件。

如果指定了-W0 选项，则不会输出警告信息。

[注意]

要在 PM+中改变输出目录，需要在<< Output >>标签页下的<< Create Error List File >>中的<< Output Path >>组合框里指定新的输出目录，并选择“with C Source[-se]”。

单独指定此选项，输出文件名也会改变。

在<< Output >>标签页下的<< Output File >>组合框中指定文件名或者输出目录。

-SE 错误列表创建说明

[使用范例]

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 :
*/

#defineTRUE      1
#defineFALSE     0
#defineSIZE      200

char    mark[SIZE+1];
main()
{
                M
                        prime = i + i + 3;
                        printf("%6d",prime);
*** WARNING W745 Expected function prototype
                        count++;
                                if((count%8) == 0) putchar('\n');
*** WARNING W745 Expected function prototype
                        for ( k = i + prime ; k <= SIZE ; k += prime)
                M
```

(10) 交叉引用列表文件创建说明 (-X)

-X	交叉引用列表文件创建说明

格式描述	-X [输出文件名]
默认说明	没有交叉引用列表文件输出
输出文件	*.xrf (*: 字母数字符号)

[功能]

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

[应用]

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

[描述]

磁盘文件名或者设备文件名都可以指定为文件名称。

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”。

除非发生严重错误(F101，除 A024 之外的异常中断)，否则即使有编译错误，交叉引用列表文件仍然能够被创建。但是在这种情况下无法保证文件内容的正确性。

[注意]

要在 PM+中改变输出目录，需要在<< Output >>标签页下的<< Create Cross Reference List File[-x]>>中的<< Output Path>>组合框里指定新的输出目录。

单独指定此选项，输出文件名也会改变。

在<< Output >>标签页下的<< Output File >>组合框里指定文件名或输出目录。

[使用范例]

指定了-X 选项。

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

-X 交叉引用列表文件创建说明

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      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  nt      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(-LW，-LL，-LT，-LF，-LI)

-LW	列表格式说明

格式描述	-LW [字符数量]	
默认说明	-LW132	(对于控制台的输出，这就是 80 个字符)

[功能]

-LW 选项可以指定所有列表文件中每一行的字符数量。

[应用]

指定-LW 选项可以改变列表文件中每一行的字符数量。

[描述]

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。

[使用范例]

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
```

-LL 列表格式说明

格式描述	-LL [行数量]
默认说明	-LL66 (对于控制台输出，则为 65535 行)

[功能]

-LL 选项指定了所有列表文件中每一页的行数。

[应用]

如果需要改变列表文件中每页的行数，请指定-LL 选项。

[描述]

通过-LL 选项可以指定的行数范围如下。

20 ≤ 每页能打印的行数 ≤ 65535

如果指定了-LL0 选项，则不会有分页符出现。

如果未指定行数，那么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。

[使用范例]

交叉引用文件每页的行数被设定为 20 行。

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

-LT 列表格式说明

描述格式	-LT [字符数量]
默认说明	-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。

[应用]

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

[描述]

-LT 选项可以指定的字符跨度范围如下。

0 ≤ 指定的字符数量 ≤ 8

如果指定了-LT0，那么不再对表格符号进行处理，并且输出 tab 代码。

如果字符数量被忽略，那么 tab 扩展字符的跨度会变为 8 个空格。

如果列表文件没有作任何说明，那么-LT 选项无效。

[使用范例]

-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 (1)。

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

-LF 列表格式说明

格式描述	-LF
默认说明	无

[功能]

指定-LF 选项将会在每个列表文件的末尾添加新的分页符。

[描述]

如果列表文件没有作任何说明，那么-LF 选项无效。

[使用范例]

指定-LF 选项。

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

-LI	列表格式说明

格式描述	-LI
默认说明	无

[功能]

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

[描述]

如果没有指定-SA 选项，则该选项被忽略。

[使用范例]

指定-LI 选项。

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

(12) 警告输出说明(-W)

-W	警告输出说明

格式描述	-W [等级]
默认说明	-W1

[功能]

指定-W 选项会将警告信息输出到控制台。

[应用]

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

[描述]

下面给出警告信息的等级。

表5-4. 警告信息等级

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

如果指定了-E 或-SE 选项，那么警告信息将被输出到错误列表文件。

等级 0 说明不需要向控制台和错误列表文件输出警告信息（当-E 或-SE 选项被选定时）。

[使用范例]

当-W 选项被忽略时会引用警告信息。

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

(13) 执行状态显示说明 (-V/-NV)

-V/-NV	执行状态显示说明

格式描述	-V -NV
默认说明	-NV

[功能]

-V 选项将当前编译的执行状态输出到控制台。

-NV 选项使-V 选项失效。

[应用]

指定这个选项可以在执行编译的同时，持续将当前的执行状态输出到控制台。

[描述]

输出此过程中的阶段名称和功能名称。

如果-V 选项和-NV 选项同时都被选定，那么最后的选项有效。

[使用范例]

-V 选项被选定。

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

(14) 参数文件说明(-F)

-F	参数文件说明

格式描述	-F 文件名
默认说明	选项和输入文件名称只能由命令行输入

[功能]

指定-F 选项可以从指定的具体文件中读入设定选项或输入文件名。

[应用]

当从命令行无法提供足够的信息来启动编译器时，请选定-F 选项，因为编译时输入了多个选项。
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。

[描述]

参数文件中不允许嵌套。
参数文件中用于描述的字符数量没有限制。
空格或者制表符可以用来分隔选项或者输入文件名称。
在参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。

[注意]

当使用 PM+时，这个选项无法使用(会发生错误)。

[使用范例]

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) 临时文件创建目录说明 (-T)

-T	临时文件创建目录说明

格式描述	-T 目录
默认说明	文件被创建在环境变量 TMP 指定的驱动器和目录中。如果不在基于 Windows 的系统中，则文件会被创建到当前驱动器的当前目录中。如果是基于 UNIX 的系统，那么文件会被创建到 in /tmp 中。

[功能]

-T 选项指定创建临时文件的驱动器和目录。

[应用]

创建临时文件的位置可以用-T 选项指定。

[描述]

即使在指定目录中存在有以前创建的临时文件，如果文件没有被保护，则在下次创建会被直接覆盖。

在存储器中扩展临时文件需要一定的存储空间，如果所需的存储空间无法满足，那么临时文件会被创建到指定目录中，并且将存储器的内容写入这个文件。对于后续临时文件的访问，是对文件的存储而不是直接访问内存。

当编译结束时，临时文件会被删除。如果按下 CTRL-C 时，编译暂停，那么临时文件也会被删除。

[使用范例]

下列命令指定了 TMP 目录作为临时文件的输出位置。

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

(16) 帮助说明(--/-?/-H)

--/-?/-H	帮助说明

格式描述	--, -?, -H
默认说明	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+中指定这个选项，若在 PM+中需要参考帮助，请在<Compiler Options>对话框中按下帮助按钮。

[应用]

显示对应选项和它的描述，在 C 编译器运行时可以引用这个选项。

[描述]

当--，-?，或-H 选项被选中时，所有其它的编译选项均不可用。

当需要查阅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。

[使用范例]

指定-H 选项。

```
C> cc78k0s  -H
```

(17) 函数扩展说明 (-Z/-NZ)

-Z/-NZ	函数扩展说明

格式描述	-Z[类型] (如果需要指定多种类型，连续指定即可) -NZ
默认说明	-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 选项无效。

类型不能被忽略，否则会发生异常错误(A012)。

[应用]

以下类型说明的函数处理过程同样适用于 78K 系列扩展函数。

[描述]

-Z 选项的类型说明如下。

表5-5. -Z选项的类型说明(1/2)

类型说明	描述
忽略	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	注释说明中不包含日本汉字类型。
B	字符/无符号字符型参数和返回值不进行整型扩展。

注 S, E, 和 N 不能同时指定。

-Z/-NZ 函数扩展说明

表 5-5. -Z 选项的类型说明(2/2)

类型 说明	描述
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。 具体的说，会执行下列任务。 下列内容都不再预留为关键字。 callt，noauto，norec，sreg，bit，boolean，#asm，#endasm 三字母序列 (3 个字母表示法)有效。 编辑器定义宏 __STDC__ 为 1。 针对字符型位域会输出下列警告。 (W787 位域类型是字符型) 如果指定了 -W2 选项，-QC，-ZP，-ZC，-ZI 和-ZL 选项会输出下列警告。 (W029 '-QC' 选项不可移植) (W031 '-ZP' 选项不可移植) (W032 '-ZC' 选项不可移植) (W036 '-ZI' 选项不可移植) (W037 '-ZL' 选项不可移植) 如果指定了 -W2 选项，针对每个 #pragma 语句输出下列警告。 (W849 #pragma 语句不可移植) 如果指定了 -W2 选项，针对__asm 语句和汇编输出会有下列警告。 (W850 Asm 语句不可移植) 如果指定了 -W2 选项，针对#asm 到#endasm 块输出下列错误。 (F801 未定义控制，等等)
M[n] (n = 1, 2)	在静态模式中可以使用扩展说明。 可以指定多达 6 个整型参数，或者 9 个字符型参数。 允许使用结构体/共同体作为函数返回值，单字节和双字节的结构体/共同体可以作参数。 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 函数修饰符。
I	将整型和短型当作字符型。编译器定义宏 __FROM_INT_TO_CHAR__ 的值为 1。
L	将长型描述当作整型。编译器定义宏 __FROM_LONG_TO_INT__ 的值为 1。

[使用范例]

指定了-ZC 和-ZP 选项。

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

(18) 驱动器文件搜索路径(-Y)

-Y	驱动器文件搜索路径

描述格式	-Y
默认说明	只有正常搜索路径

[功能]

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

正常搜索路径如下。

(1) <..\dev> (cc78k0s.exe 的启动路径)
(2) CC78K0S 的启动路径
(3) 当前目录
(4) 环境变量 PATH 指定的路径

[应用]

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

[注意]

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

[使用范例]

指定了-Y 选项。

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

(19) 静态模式说明 (-SM)

-SM	静态模式说明

格式描述	-SM [n] (n = 1 到 16)
默认说明	普通模型 (n = 0)

[功能]

在编译时指定-SM 选项。当-SM 被选中时，该目标被称为静态模式，如果未指定-SM 选项，则该目标被称为普通模型。

一般情况下，访问静态区域的指令比较短，并且执行速度比访问堆栈帧的指令更快。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。

[描述]

所有的函数参数的传递都通过寄存器完成，函数将函数参数和自动变量分配到静态区域。

leaf 函数将函数参数和自动变量分配到 saddr 区域内低于 FEFFH 的地址，必须按照指定的顺序来存储。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 被省略时，就没有共用区域。

编译器定义的宏 _ _STATIC_ _MODEL 认为其值为 1。

sreg/_ _sreg 关键字可以用来定义函数参数和自动变量。这些拥有 sreg/_ _sreg 关键字的函数参数和自动变量被分配到 saddr 区域，并能够支持位操作。

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。

-SM	静态模式说明

[注意]

由于参数和自动变量被保护为静态，递归函数的参数和自动变量的可能会被损坏。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。

即使中断服务函数使用的是共用区域，在无法在共用区域中进行保存/返回操作。

[使用范例]

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

第 6 章 C 编译器输出文件

CC78K0S 会产生下列文件：

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

6.1 目标模块文件

目标模块文件是一种包含 C 源程序编译结果的二进制映象文件。

如果指定了调试数据输出选项(-G)，目标模块文件将包含调试数据。

6.2 汇编源模块文件

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

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

[输出格式]

```
    ; 78K/0S 系列 C 编译器 V(1)x.xx 汇编器 源程序
    ;                                                                    日期：(2)xxxxx 时间：
(3)xxxxx

    ; 命令       ：(4)-c9024 prime.c -sa
    ; 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
```

[输出项的说明] (1/2)

编号	说明	列宽	格式
(1)	版本号	4 (固定长度)	显示格式为“x.yz”
(2)	日期	11 (固定长度)	系统日期 (显示格式为“DD Mmm YYYY”)
(3)	时间	8 (固定长度)	系统时间 (显示格式为“HH：MM：SS”)
(4)	命令行	—	在“CC78K0S”之后输出命令行内容，80 列后的内容从下一行的 15 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(5)	C 源模块文件名称	操作系统允许的字符数	输出指定文件名。如果文件类型被忽略，那它的默认文件类型（扩展名）就是“.C”。80 列后的内容从下一行的 15 列开始输出。第一列输出为分号(;)。
(6)	汇编源模块文件名称	操作系统允许的字符数	输出指定文件名。如果文件类型被忽略，那它的文件类型（扩展名）默认为“.asm”。 80 列后的内容从下一行的 15 列开始输出。第一列输出为分号(;)。
(7)	参数文件内容	—	输出参数文件目录。80 列后的内容从下一行的 15 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(8)	设备类型	最大值 6 (可变)	这个字符串由-C 选项指定，见有关设备文件相关文档。
(9)	调试数据	最大值 8 (可变)	输出调试控制。输出内容为$DEBUG 或$NODEBUG。
(10)	调试信息汇编器控制	9 (固定长度)	输出 NODEBUGA 控制。输出为$NODEBUGA.
(11)	汉字类型信息	最大值 15 (可变)	输出汉字编码类型。输出的是$KANJICODE SJIS，$KANJICODE EUC，或 $KANJICODE NONE.
(12)	工具信息	37 (固定长度)	输出工具信息，版本号，错误信息，指定参数等。(信息由$TOL_INF 开始).
(13)	符号信息	—	输出符号信息 (信息开始标志为$DGS)。只有指定了调试数据输出选项时才会输出这个信息。即使指定了–G1 选项也不会输出这项信息。
(14)	汇编源程序	—	输出的汇编源程序中包括编译结果。
(15)	行号	4 (固定长度)	输出 C 源程序模块文件的行号，计算结果用零抑制的十进制表示，且用向右对准的格式。
(16)	C 源程序	—	输入 C 源程序映象。80 列后的内容从下一行的 15 列开始输出。第一列输出为分号(;)。
(17)	行号信息	—	行号就是行号入口（信息以$DGL 开始）。只有指定了调试数据输出选项时才会输出这个信息。即使指定了–G1 选项也不会输出这项信息。

[输出项说明] (2/2)

编号	说明	列宽	格式
(18)	符号信息创建对应的标签	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)	编译器中使用的目标设备	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。
(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	—	输出指令所占的空间大小和执行时间（信息以;[INF]开始).
(22)	函数信息(起始)	—	显示函数信息的起始
(23)	函数信息（文件名）	—	输出带有绝对路径的目标源程序文件名（信息以;$FILE 开始).
(24)	函数信息（定义函数）	—	输出函数名称，用十进制码表示行号（信息以;$FUNC 开始).
(25)	函数信息（返回值，函数定义时的参数）	—	输出函数的返回值寄存器和参数信息（寄存器和堆栈）
(26)	函数信息（定义函数的大小，时钟，堆栈）	—	输出容量大小，执行时间和统计出的函数堆栈最大值。
(27)	函数信息（调用函数）	—	输出函数名和函数调用行号，用十进制码表示（信息以;$CALL 开始）。
(28)	函数信息（调用函数的返回值，参数）	—	输出在函数调用期间的返回值寄存器和参数信息（寄存器和堆栈位置）

6.3 错误列表文件

错误列表文件包括编译中发生的所有错误信息和警告信息。

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

6.3.1 关于 C 语言的错误列表文件

[输出格式]

```
/*
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;
 ***ERROR (9) F711 (10)  未定义‘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);
 ***WARNING (9)W745 (10)  期望函数原形。
                        M
/*
(11) Target chip：  uPD789024
(12) Device file：  Vx.xx
Compilation complete, (13) 1 error(s) and   (14) 5 warning(s) found.
*/
```

[输出项目的说明]

编号	说明	列宽	格式
(1)	版本号	4 (固定长度)	显示格式为“x.yz”
(2)	日期	11 (固定长度)	系统日期 (显示格式为“DD Mmm YYYY”)
(3)	时间	8 (固定长度)	系统时间 (显示格式为“HH：MM：SS”)
(4)	命令行	—	在“CC78K0S”之后输出命令行内容，80 列后的内容从下一行的 13 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(5)	C 源模块文件名称	操作系统允许的字符数(可变长度)	输出指定文件名。如果文件类型被忽略，那它的默认文件类型（扩展名）为“.C”。80 列后的内容从下一行的 13 列开始输出。第一列输出为分号(;)。
(6)	错误列表文件	操作系统允许的字符数(可变长度)	输出指定文件名。 如果文件类型被忽略了， 那它的默认文件类型（扩展名）就是‘.cer’。80 列后的内容在 13 列开始的下一行输出。
(7)	参数文件目录	—	输出参数文件目录。80 列后的内容从下一行的 15 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(8)	C 源程序	—	这是输入 C 源程序映象，80 列之后的内容不会被换到下一行。
(9)	错误信息编号	4 (固定长度)	用 “#nnn” 这种格式输出错误编号。如果有错误，“#”的位置输出“F”。，如果有警告，“#”的位置输出“W”。"nnn" (错误编号)显示为 3 位十进制数(无需零抑制)。
(10)	错误信息	—	见第 9 章 错误信息。80 栏后的内容不会换到下一行
(11)	编译器使用的目标设备	最大值 15 (变量)	目标设备在命令行中通过选项-C 指定，或在源程序文件中指定。
(12)	设备文件版本	6 (固定长度)	显示输入设备文件的版本号。
(13)	错误次数	4 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。
(14)	警告次数	4 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。

6.3.2 只有错误信息的错误列表文件

[输出格式]

```
(1) PRIME.C((2) 18) :    (3) W745   (4) Expected function prototype
(1) prime.c((2) 20) :    (3) W745   (4) Expected function prototype
(1) prime.c((2) 26) :    (3) W622   (4) No return value
(1) prime.c((2) 37) :    (3) W622   (4) No return value
(1) prime.c((2) 44) :    (3) W622   (4) No return value

目标芯片 : (7) uPD789024
设备文件 : (8) Vx.xx

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

[输出项说明]

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

6.4 预处理列表文件

预处理列表文件是一种 ASCII 映象文件，仅包括 C 源程序预处理结果。

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

[输出格式]

当页面宽度为 80 时

```
/*
78K/0S Series C 编译器  V (1) x.xx  预处理列表      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]

编号	说明	列宽	格式
(1)	版本号	4 (固定长度)	显示格式为“x.yz”
(2)	日期	11 (固定长度)	系统日期 (显示格式为“DD Mmm YYYY”)
(3)	页数	4 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。
(4)	命令行	—	在“CC78K0S”之后输出命令行内容，80 列后的内容从下一行的 13 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(5)	C 源模块文件名称	操作系统允许的字符数	输出指定文件名。如果文件类型被忽略，那它的默认文件类型（扩展名）为“.C”。80 列后的内容从下一行的 13 列开始输出。
(6)	预处理列表文件名	操作系统允许的字符数	输出指定文件名。如果文件类型被忽略，那它的默认文件类型（扩展名）为“.ppl”。80 列后的内容从下一行的 13 列开始输出。
(7)	参数文件目录	—	输出参数文件目录。80 列后的内容从下一行的 13 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(8)	行号	5 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。
(9)	C 源程序	—	这是输入的 C 源程序。超过 80 行长度的内容在下一行 9 列开始输出。
(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

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

[输出格式]

当页面宽度为 80 时

```
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)

ATTRIB      MODIFYTYPE          SYMBOL    DEFINE          REFERENCE

(9) EXTERN  (10)    (11) array  (12) mark  (13) 5   (14) 14    (14)16   (14) 22
(9) EXTERN  (10)    (11) func   (12) main  (13) 7
(9) AUTO1   (10)    (11) int    (12) i     (13) 9   (14) 13    (14) 13  (14) 13  (14) 14
                                                    (14) 15    (14) 15  (14) 15  (14) 16
                                                    (14) 17    (14) 17  (14) 21
(9) AUTO1   (10)    (11) int    (12) prime (13) 9   (14) 17    (14) 18  (14) 21  (14) 21
(9) AUTO1   (10)    (11) int    (12) k     (13) 9   (14) 21    (14) 21  (14) 21  (14) 22
(9) AUTO1   (10)    (11) int    (12) count (13) 9   (14) 11    (14) 19  (14) 20  (14) 25

/*
(15) Target chip：  uPD789024                ⋮
(16) Device file：  Vx.xx
*/
```

[输出项目的说明] (1/2)

编号	说明	列宽	格式
(1)	版本号	4	显示格式为“x.yz”
(2)	日期	11 (固定长度)	系统日期 (显示格式为“DD Mmm YYYY”)
(3)	页数	4 (固定长度)	输出一个右对齐的 10 进制数，并作零抑制。
(4)	命令行	—	在“CC78K0S”之后输出命令行内容，80 列后的内容从下一行的 13 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(5)	C 源模块文件名称	操作系统允许的字符数	输出指定文件名。如果文件类型被忽略，那它的默认文件类型（扩展名）为“.C”。80 列后的内容从下一行的 13 列开始输出。
(6)	交叉引用数据清单文件名	操作系统允许的字符数	输出指定文件名。如果文件类型被忽略，那它的默认文件类型（扩展名）为“.xrf”。80 列后的内容从下一行的 13 列开始输出。
(7)	参数文件目录	—	输出参数文件目录。80 列后的内容从下一行的 13 列开始输出。第一列输出为分号(;)。一个以上的空白字符或制表符都用一个空白字符表示。
(8)	包含文件	操作系统允许的字符数	输出在 C 源程序中指定的文件名称。从 1 开始编号的数字“n”表示所包含的文件号。超过 80 行长度的内容在下一行 13 列开始输出。没有包含文件则不输出此行内容。
(9)	符号属性	6 (固定长度)	显示符号属性。 外部变量用 EXTERN 表示，外部静态变量用 EXSTC 表示，内部变量用 INSTC 表示，自动变量用 AUTOnn 表示，寄存器变量用 REGnn 表示（这里的 nn 表示范围，编号从 1 开始累加)，外部 typedef 声明用 EXTYP 表示，内部 typedef 声明用 INTYP 表示，标签用 LABEL 表示，结构体或共用体标签用 TAG 表示，成员用 MEMBER 表示，函数参量用 PARAM 表示。
(10)	符号限定词属性	6（固定长度）	显示符号限定词属性（左对齐）.常变量用 CONST 表示，volatile 变量用 VLT 表示，callt 函数用 CALLT 表示，callf 函数用 CALLF 表示，noauto 函数用 NOAUTO 表示，norec 函数用 NOREC 表示，sreg-bit 变量用 SREG 表示，特殊功能寄存器变量用 RWSFR 表示，唯读 sfr 变量用 ROSFR 表示，唯写 sfr 变量用 WOSFR 表示，中断函数用 VECT 表示。
(11)	符号类型	7（固定长度）	显示符号类型。类型包括字符型、整型、短整型、长整型、字段。 以“u”开始的是无符号型。其他的类型还有空类型、浮点型、双精度、长双精度(long double)、函数、数组、指针、结构、联合、枚举、比特、中断和#define 宏定义.

[输出项说明] (2/2)

编号	说明	列宽	格式
(12)	符号名称	15 (固定长度)	如果符号名超过了 15 个字符并排为一行，那就按照符号名称原样输出。如果符号名超过 15 个字符且超出了一行的长度，超过的部分从下一行的 23 列输出，13 项和 14 项从下一行 39 列输出。
(13)	符号定义行号	7 (固定长度)	输出定义符号的文件名和行号，显示格式如下： 行号(5 位数)：包含文件数。
(14)	符号引用行号	7 (固定长度)	输出引用符号的文件名和行号，显示格式如下： 行号(5 位数)：包含文件数。如果所列内容超过行的长度，剩余内容从下一行 47 列开始输出。
(15)	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。
(16)	设备文件版本	6 (固定长度)	显示输入设备文件的版本号

第 7 章 C 编译器的使用方法

7.1 高效操作（EXIT 状态函数）

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

- 正常结束：0
- 警告：0
- 严重错误：1
- 中止：2

如果没有使用 PM+，CC78K0S 运行于命令行形式，可使用批处理文件中的状态进一步提高操作效率。

[使用实例]

```
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]

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

7.2 建立开发环境（环境变量）

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

[使用实例]（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
在 Windows 系统中，如果没有设置，将搜索 C:\NECTools32\INC78K0S 目录（假设 CC78K0S 被安装在 C:\NECTools32 中）
•在连接时从 c:\nectools32\lib78k0s 搜索库文件。
在 Windows 系统中，如果没有设置，将搜索 C:\NECTools32\LIB78K0S 目录（假设 CC78K0S 被安装在 C:\NECTools32 中）
•临时文件存放在 c:\tmp 目录下。
•移动 JIS 码的使用和汉字码相同。

[警告]

当使用 PM+时不要设置环境变量。

7.3 中断编译

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

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

第 8 章 启动例程

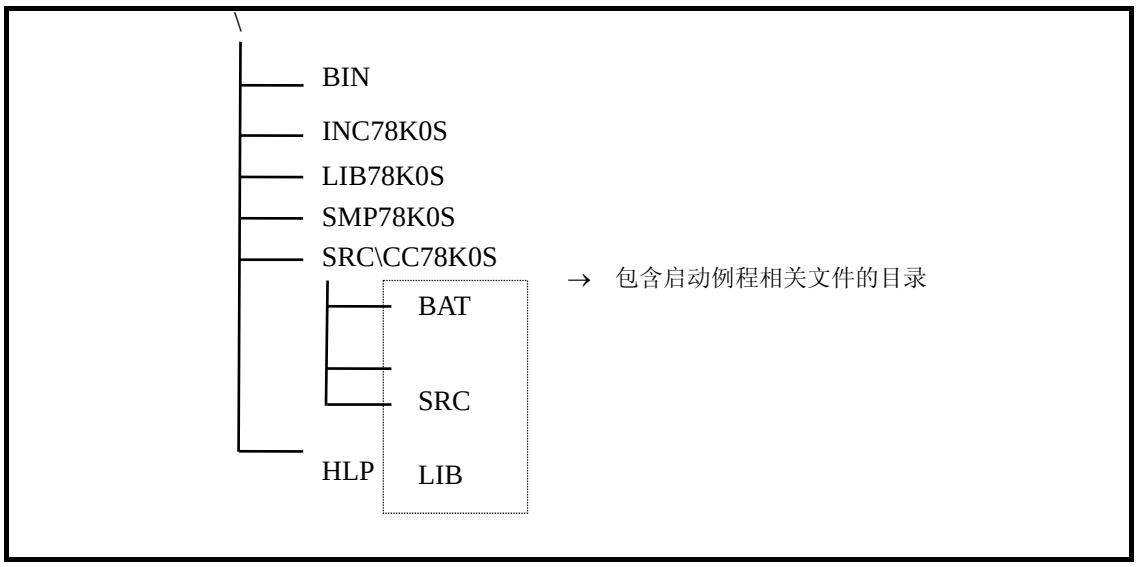
为了执行 C 源程序，需要有一个程序来激活系统和用户程序（主函数）中的 ROM 化过程（ROMization）。这个程序叫做启动例程。

为了执行用户编写的程序，必须为程序创建一个启动例程。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 目录中。



SRC\CC78K0S 目录的内容显示如下：

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，可以连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 (mkstup.sh)，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 目录内容

这个目录中的批处理文件不能在 PM+中使用。

只有必须修改源程序（例如启动例程）时才使用这些批处理文件。

BAT 目录中的设备文件(d9026.78k)不是用来开发的，是为了更新库等动作而启动批处理文件时使用。因此，实际开发时需要其它对应的设备文件。

表 8-1 BAT 目录内容

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

注 1. ROMization 例程在库中，所以库也会被批处理文件更新。

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

8.1.2 SRC 目录内容

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

表 8-2 SRC 目录内容

启动例程源程序文件名	说明
cstart.asm 注	启动例程的源程序文件 (使用标准库时)
cstartn.asm 注	启动例程的源程序文件 (未使用标准库时)
rom.asm	ROMization 例程的源文件
_putchar.asm	_putchar 功能
putchar.asm	putchar 功能
getchar.asm	getchar 功能
longjmp.asm	longjmp 功能
setjmp.asm	setjmp 功能
def.inc	根据类型来设置库
macro.inc	每个典型模式的宏定义
stdio.inc	为 EOF 和 LF 设置符号代码

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

8.2 批处理文件说明

8.2.1 生成启动例程的批处理文件

BAT 目录中的 mkstup.bat (UNIX 中的 mkstup.sh)用来创建启动例程的目标文件。

RA78K0S 汇编程序包中的汇编器对 mkstup.bat (mkstup.sh)来说是必需的。因此，如果没有指定路径，就需要指定路径并来运行。

以下详细解释这个文件的使用方法。

[如何使用]

在包含 mkstup.bat (mkstup.sh)的 src\cc78k0s\bat 目录中执行如下命令行。

```
mkstup    设备类型注
```

注 请参阅设备文件相关文档。

[使用实例]

将要创建的启动例程使用的目标设备是μPD789024，。

```
mkstup  9024
```

mkstup.bat (mkstup.sh)批处理文件存储在 LIB 目录下，它将会改写的启动例程目标文件，LIB 目录和下面显示的 BAT 目录在同一级目录中。

需要连接到目标文件的启动例程会输出到每一个目录。

在 LIB 中创建的目标文件的名称如下。

```
——— LIB ———— 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

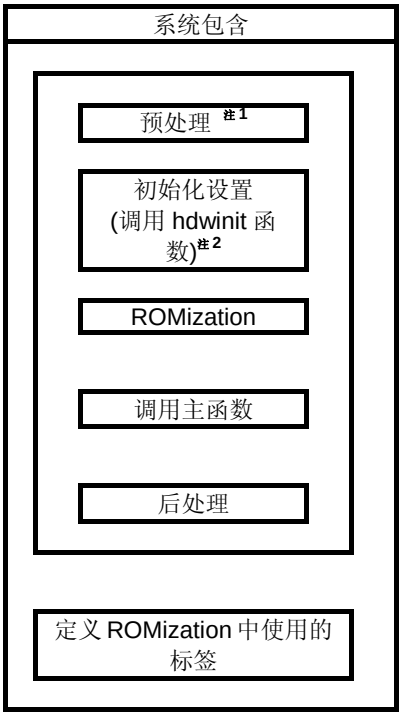
存储器初始化，包含在系统中的 ROMization 和 C 源程序的开始进程和终止进程都会进行。

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

（2）配置

表 8-3 显示了与启动例程相关序和它们的配置情况。

表 8-3 启动例程概述



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

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）。如果用户不创建 hdwinit 函数，将不做任何处理就直接返回。

cstart.asm 和 cstartn.asm 的内容几乎完全相同。

表 8-4 展示了 cstart.asm 和 cstartn.asm 的不同之处。

表 8-4 cstart.asm 和 cstartn.asm 的区别

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

（3）启动例程的使用

表 8-5 列举了 CC78K0S 为启动例程提供的目标文件名称。

表 8-5 源程序文件和目标文件的比较

文件类型	源文件	目标文件
启动例程	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。

2. *：如果使用了标准程序库中的一个固定区域，需要加上“l”。

如果指定为静态模型，则需要加上“s”。

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

8.3.2 样例程序的说明（cstart.asm）

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

注 调用 cstart 时需要在前面加_@，即格式为_@cstart。

（1）预处理

cstart.asm 中的预处理在<1>到<6>项中说明（如下）。

[cstart.asm 预处理]

```
        NAME    @cstart

$INCLUDE (def.inc)                  <1>包含头文件
$INCLUDE (macro.inc)
                                    <2>库切换
                                    <3>符号定义

BRKSW   EQU     1       ;brk,sbrk,calloc,free,malloc,realloc function use
EXITSW  EQU     1       ;exit,atexit function use
$_IF (_STATIC)
RANDSW  EQU     0       ;rand,srand     function use
DIVSW   EQU     0       ;div            function use
LDIVSW  EQU     0       ;ldiv           function use
FLOATSW EQU     0       ;floating point variable use
$ELSE
RANDSW  EQU     1       ;rand,srand     function use
DIVSW   EQU     1       ;div            function use
LDIVSW  EQU     1       ;ldiv           function use
FLOATSW EQU     1       ;floating poin  variable use
$ENDIF
STRTOKSW EQU    1       ;strtok         function use

        PUBLIC  _@cstart, _@cend

$_IF(BRKSW)
        PUBLIC  _@BRKADR, _@MEMTOP, _@MEMBTM
                M
$ENDIF
                                    <4>堆栈分析所需的符号外部引用声明
        EXTRN   _main,_@STBEG,_hdwinit
$_IF(EXITSW)
        EXTRN   _exit
$ENDIF
                                    <5> ROMization 处理标签的外部引用声明
```

```
EXTRN     _?R_INIT,_?R_INIS,_?DATA,_?DATS
                                        <6>为标准库保留区域
@@DATA          DSEG

$_IF(EXITSW)
_@FNCTBL:       DS              2*32
_@FNCENT:       DS              2
                M
_@MEMTOP:       DS              32
_@MEMBTM:
$ENDIF
```

<1>包含头文件

def.inc → 根据类型设置库。

macro.inc → 每个典型模式的宏定义。

<2>库切换

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

<3>符号定义

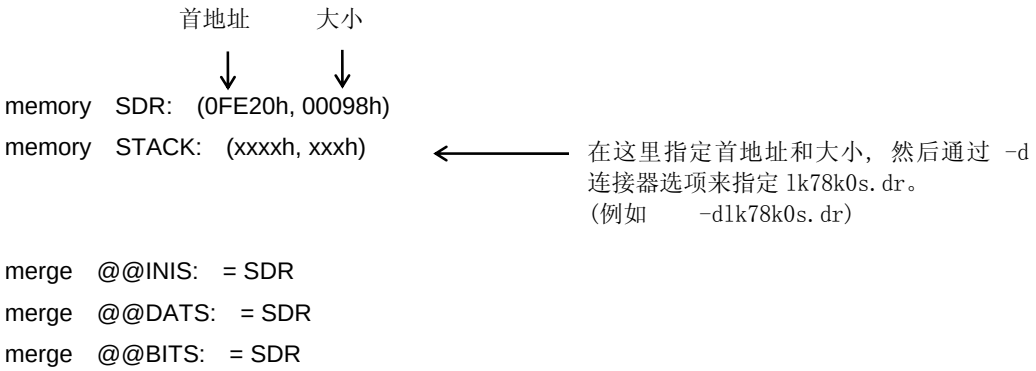
定义使用标准库所需的符号。

<4>堆栈分析所需的符号外部引用声明

- 用于堆栈分析的这个公共符号（_@STBEG）是个外部引用声明。_@STBEG 的值是堆栈区域的最终地址+1。
- 在连接器中指定符号生成选项（-S）即可自动生成_@STBEG 以方便堆栈分析。因此，当连接时都会指定-S 选项。这种情况下，指定堆栈中使用的区域名称。如果区域的名称被省略，就使用 RAM 区域。但是创建一个连接命令文件（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。
- 下面是一个连接命令文件的例子。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 汇编器程序包操作篇用户手册(U16656E)**）。

[在连接中指定–sSTACK 的例子]

创建 lk78k0s.dr（连接命令文件）。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。连接指令参阅 smp78k0s\cc78k0s 目录中的 lk78k0s.dr 文件。



<5> ROMization 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）初始化设置

cstart.asm 中的初始化设置在<7>到<9>项中加以说明。

[cstart.asm 中的初始化设置]

```
                                        <7> 复位向量设置
@@VECT00 CSEG   AT   0
         DW          _@cstart

@LCODE    CSEG
_@cstart:
                                        <8> SP (堆栈指针)设置
         MOVW   AX,#_@STBEG             ;SP <- 堆栈起始地址
         MOVW   SP,AX
         CALL    !_hdwinit              <9> 硬件初始化函数调用
$ENDIF
                M
$_IF(BRKSW OR EXITSW OR RANDSW OR FLOATSW)
         XOR    A,A
$ENDIF
                M
```

<7>复位向量设置

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
```

<8>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。

<9>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）ROMization 处理

下面描述 cstart.asm 中的 ROMization。

[ROMization 处理]

```
;*****************************
;ROM 数据复制
;*****************************
;复制有初值的外部变量
        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
;复制没有初值的外部变量
LDATA1:
                M
```

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

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 8-1 显示了(a)有初值的外部变量 的 ROMization 处理过程。

变量（a）的初值被编译器放在 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）无需复制，RAM 中对应的段直接清零（见**表 8-7 初值在 RAM 中的区域（复制的目标地址）**。表 8-6 和表 8-7 显示了变量（a）和（d）放置的 ROM 段名称和 RAM 段名称，以及每个段中初值的首末标签。

图 8-1 ROMization 处理过程

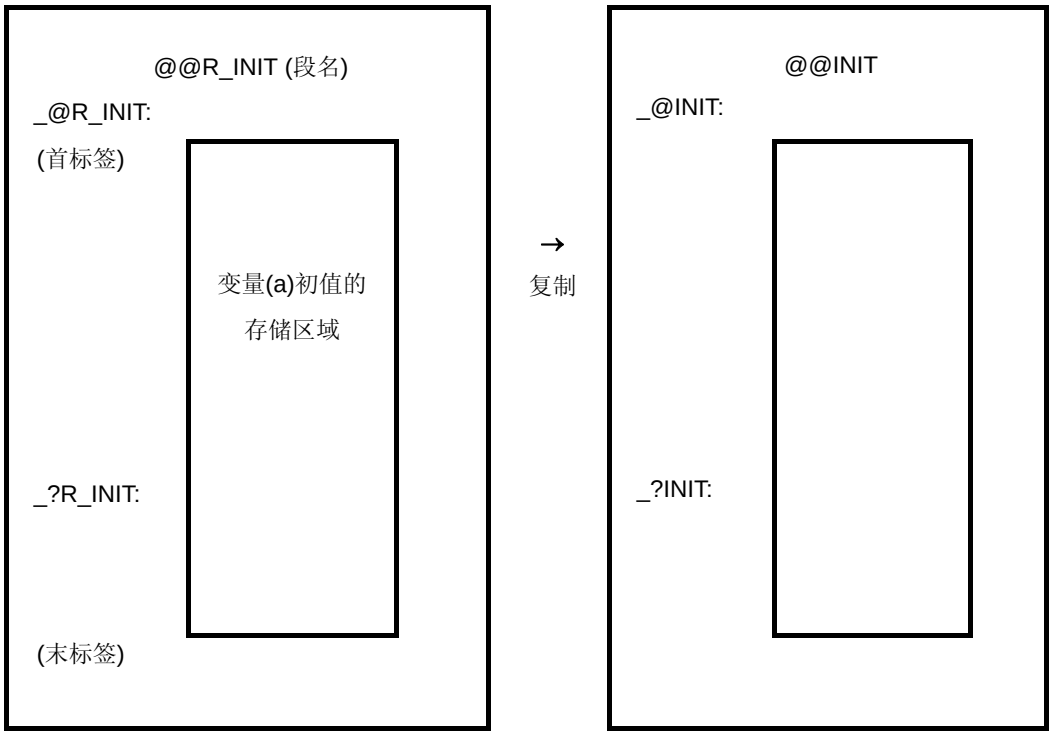


表 8-6 初值在 ROM 中的区域

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 8-7 初值在 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

cstart.asm 中启动主函数和后处理的说明

[启动主函数和后处理]

```
                CALL    !_main          ;main();            ;<10> 启动主函数
$_IF(EXITSW)
                MOVW    AX,#0
                CALL    !_exit          ;exit(0);           ;<11> 启动 EXIT 函数
$ENDIF
          BR        $$
;
_@cend:
                                                            ;<12> 对 ROMization 处理中
@@R_INIT  CSEG                                              ;使用的段和标签进行定义
_@R_INIT:
@@R_INIS  CSEG    UNITP
_@R_INIS:
@@INIT    DSEG
_@INIT:
@@DATA    DSEG
_@DATA:
@@INIS    DSEG    SADDRP
_@INIS:
@@DATS    DSEG    SADDRP
_@DATS:
@@CALT    CSEG    CALLT0
@@CNST    CSEG
@@BITS    BSEG
;
                END
```

<10>启动主函数
主函数被调用。

<11>启动 EXIT 函数
如果需要，调用 EXIT 函数。

<12>对 ROMization 处理中使用的段和标签进行定义
定义了（a）到（d）每个变量在 ROMization 处理中使用的段和标签（见 **8.3.2 (3) ROMization 处理**）。段指示了每个变量初值的存储区域，标签指示每个段的首地址。

下面对 ROMization 处理文件 rom.asm 说明。rom.asm 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>  定义 ROMization 处理中使用的标签
_?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 ROMization 处理中使用的标签

定义了在 ROMization 处理中（a）到（d）所有变量使用的标签（见 **8.3.2 (3) ROMization 处理**）。这些标签指示了存储每个变量初值的段的末地址。

8.3.3 修改启动例程

可以对 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）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 (mkstup.sh)对修改的源程序文件（cstart*.asm）(*: 字母数字的符号)重新汇编。

·标准库函数中使用的符号

如果没有使用表 8-8 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 8-8 程序库函数中使用的符号

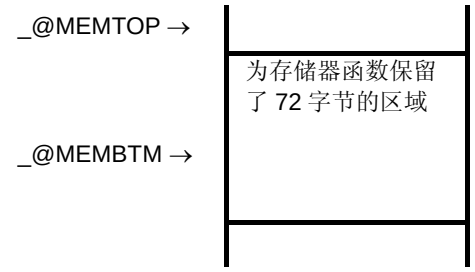
程序库函数名	使用的符号
brk sbrk strtol strtoul malloc calloc realloc free	_errno _@MEMTOP _@MEMBTM _@BRKADR
exit	_@FNCTBL _@FNCENT
rand srand	_@SEED
div	_@DIVR
ldiv	_@LDIVR
strtok	_@TOKPTR
atof strtod 数学函数 浮点运行库	_errno

·存储器函数中使用的区域

如果用户定义了存储器函数使用区域的大小，在下例中对此说明：

例) 如果你想为存储器函数保留 72 字节空间，需对启动例程的初始设置做如下改动

```
_@MEMTOP:    DS    72
_@MEMBTM:
```



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。对连接命令文件的修改请参阅 **(2) 连接命令文件。**

例) 减小指定大小

```
_@MEMTOP:    DS    72    → 改为 40
```

（2）连接命令文件

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(U16656E)**）。

·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。特别是普通模型的 FED8H 到 FEFFH 的 40 字节。当指定了-SM[n]选项进入静态模型时，saddr 的部分区域[FEF0H 到 FEFFH]作为公用区域使用。

（普通模型）
(a) 运行时刻（runtime）库的参数[FEF8H 到 FEFFH]
(b) norec 函数的自动变量或参数[FEE8H 到 FEF7H]
(c) 指定–qr 选项时的寄存器变量[FED8H 到 FEE7H]
(d) 标准库任务（区域（b）和（c）的一部分）

·如果用户不使用标准库，区域（d）不会被使用。

（静态模型）
（a）共用区域[FEF0H to 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>

	首地址	大小	
memory RAM:	(0FB00h,	00320h) →	将此空间小大的值改大.
memory SDR:	(0FE20h,	00098h)	(如果需要也可以修改首地址)

merge @@INIS: =SDR
merge @@DATS:=SDR → 指定段的存储位置.
merge @@BITS: =SDR

如果你想改变段的存储位置，需要添加合并（Merge）语句。如果使用了修改编译器输出区块名称的函数，这个段可以独立定位（敬请参阅 **CC78K0S 语言篇 用户手册(U16655E)中的第 11 章**）。
如果段的位置修改结果不能为定位内容提供足够的存储空间，则需改变相应的存储器声明语句。

第 9 章 错误信息

9.1 错误信息格式

错误信息格式如下。

```
源文件名 (行号) ：错误信息
```

例

```
prime.c(8) : F712 Declaration syntax
prime.c(8) : F301 Syntax error
prime.c(8) : F701 External definition syntax
prime.c(19) : W745 Expected function prototype
```

但是，内部错误 F101，F103 和 F104 会使用下列输出格式。

```
[xxx.c <yyy> zzz] F101 Internal error
[xxx.c <yyy> zzz] F103 Intermediate file error
[xxx.c <yyy> zzz] F104 Illegal use of register
```

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) 内部错误或存储器错误信息
(3) 字符错误信息
(4) 配置元素错误信息
(5) 转换错误信息
(6) 表达式错误信息
(7) 语句错误信息
(8) 声明或函数定义错误信息
(9) 预处理指令错误信息
(10) 重要文件 I/O 和运行于非法操作系统的错误信息

9.3 错误信息列表

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

错误编号格式如下所示。

A/F/Wnnn

A：异常中止

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

F：严重错误

在错误信息输出之后，错误部分被忽略，同时编译处理会继续进行。不会输出目标模块文件和汇编源程序模块文件。

W：警告

在错误信息输出之后，编译处理继续进行。选项指定的文件会正常输出。

nnn (3-位 数字)

From 001	Error message for a command line
From 101	Error message for an internal error or memory
From 201	Error message for a character
From 301	Error message for a configuration element
From 401	Error message for conversion
From 501	Error message for an expression
From 601	Error message for a statement
From 701	Error message for a declaration or a function definition
From 801	Error message for a preprocessing directive
From 901	Error message for fatal file I/O or running on an illegal operating system

注意　如果文件名有错误语法，文件名会加入到错误信息中。错误信息的添加、修改和删除要符合所使用的 C 编译器具体语言规范。

(1) 命令行错误信息<编号从 001 开始> (1/3)

A001	信息	Missing input file
	原因	输入源文件名没有被指定。
	对策	输出“Please enter ‘cc78k0s--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的文件名。
A002	信息	Too many input files
	原因	指定了多个输入源文件名。
	对策	输出“Please enter ‘cc78k0s--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的文件名。
A003	信息	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。
A004	信息	Illegal file name
	原因	在指定文件名时使用了不正确的格式、字符或数字。
A005	信息	Illegal file specification
	原因	指定了一个非法的文件名。
A006	信息	File not found
	原因	指定的输入文件不存在。
A007	信息	Input file specification overlapped file name
	原因	指定的输入文件名有重名。
A008	信息	File specification conflicted file name
	原因	指定的 I/O 文件名有重名。
A009	信息	Unable to make file file name
	原因	因为指定的输出文件已经存在，并且是只读文件，所以不能被创建。
A010	信息	Directory not found
	原因	输出文件名中指定的驱动或目录不存在。
A011	信息	Illegal path
	原因	在路径参数设置选项中指定了非法路径名。
A012	信息	Missing parameter ‘option’
	原因	某个必需的参数没有被指定。
	对策	输出“Please enter ‘cc78k0s--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的参数。
A013	信息	Parameter not needed ‘option’
	原因	指定了一个不必要的选项参数。
	对策	输出“Please enter ‘cc78k0s--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的参数。
A014	信息	Out of range ‘option’
	原因	对选项指定的参数值超出了允许范围。
	对策	输出“Please enter ‘cc78k0s--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的参数值。
A015	信息	Parameter is too long
	原因	在选项参数中的字符数目超出了限制范围。

(1) 命令行错误信息<编号从 001 开始> (2/3)

A016	信息	Illegal parameter ‘option’
	原因	在选项参数中有语法错误。
	对策	输出“Please enter ‘cc78k0s--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的选项。
A017	信息	Too many parameters
	原因	选项参数的总数超出了限制。
A018	信息	Option is not recognized ‘option’
	原因	指定了一个不正确的选项。
	对策	输出“Please enter ‘cc78k0s--’ if you want help message”信息。 使用--, -?，或-H 选项来访问帮助文件，同时请输入正确的选项。
A019	信息	Parameter file nested
	原因	在参数文件中指定了-F 选项。
	对策	不能在参数文件中指定参数文件，对其进行修正，去除参数文件的嵌套。
A020	信息	Parameter file read error
	原因	参数文件读取失败。
A021	信息	Memory allocation failed
	原因	存储器分配失败
W022	信息	Same category option specified – ignored ‘option’
	原因	同时指定了多个相同冲突的选项。
	编译	最后指定的选项是有效的，并继续进行处理。
W023	信息	Incompatible chip name
	原因	在命令行中指定的设备类型和源程序中指定的设备类型有冲突。
	编译	以命令行中指定的设备类型为准。
A024	信息	Illegal chip specifier on command line
	原因	在命令行中指定的设备类型是不正确的。
W029	信息	‘-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 编译选项**)。
W031	信息	‘-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 编译选项**)。
W032	信息	‘-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 编译选项**)。
A033	信息	Same category option specified ‘option’
	原因	同时指定了多个相同冲突的选项。
	对策	输出“Please enter ‘cc78k0s--’ if you want help message”信息。 使用 --，-? 或 –H 选项来访问帮助文件，并对输入进行改正。

(1) 命令行错误信息<编号从 001 开始> (3/3)

W036	信息	'-ZI' option is not portable
	原因	-ZI 选项不符合 ANSI 标准规范 (对于-ZI 细节，敬请参阅**第 5 章 编译选项**)。
W037	信息	'-ZL' option is not portable
	原因	-ZL 选项不符合 ANSI 标准规范(对于-ZL 细节，敬请参阅**第 5 章 编译选项**)。
W038	信息	'-ZI' option specified - regarded as '-QC'
	原因	因为指定的-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。
W039	信息	'-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。
W040	信息	'-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。-RK 选项被忽略。
W041	信息	'-SM' option specified - ignored '-QR'
	原因	因为指定了静态模式选项-SM，寄存器优化选项-QR 被忽略。
W045	信息	'-SM' option specified - ignored '-ZR'
	原因	因为指定了静态模式选项-SM，pasca 函数接口说明选项-ZR 被忽略。
W052	信息	'-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)。
W055	信息	'-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。-ZM 选项被忽略。

(2) 内部错误和内存错误信息<编号从 101 开始>

F101	信息	Internal error
	原因	内部错误发生。
	对策	请联系技术支持。
F102	信息	Too many errors
	原因	严重错误的数量超过了 30。
	编译	处理继续进行，但后续的错误信息不被输出。先前的错误可能会引起更多的错误。所以首先请修改之前发生的错误。
F103	信息	Intermediate file error
	原因	媒介文件有误。
	对策	请联系技术支持。
F104	信息	Illegal use of register
	原因	使用寄存器的方法不正确。
F105	信息	Register overflow : simplify expression
	原因	表达式过于复杂，导致没有空余的寄存器可用。
	对策	对引起错误的复杂表达式进行简化。
A106	信息	Stack overflow ‘overflow cause’
	原因	堆栈溢出。可能是栈或堆发生溢出。
	对策	请联系技术支持。
F108	信息	Compiler limit : too much automatic data in function
	原因	为函数中的自动变量分配的区域超过了 64 KB 的限制。
	对策	减少变量，使其所占的区域不超过 64 KB。
F109	信息	Compiler limit : too much parameter of function
	原因	为函数中的参数文件分配的区域超过了 64 KB 的限制。
	对策	减少参数的使用，使其所占的区域不超过 64 KB。
F110	信息	Compiler limit : too much code defined in file
	原因	为文件中的代码分配的区域超过了 64 KB 的限制。
F111	信息	Compiler limit : too much global data defined in file
	原因	为文件中的全局变量分配的区域超过了 64 KB 的限制。
F113	信息	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。
	对策	函数本身过大，建议将其拆分。

(3) 字符错误信息<编号从 201 开始>

F201	信息	Unknown character ‘hexadecimal number’
	原因	具有特定内部编码的字符无法被识别。
F202	信息	Unexpected EOF
	原因	文件正在操作时，文件结束。
W203	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。

(4) 配置元素错误信息<编号从 301 开始> (1/3)

F301	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。
F303	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。
W304	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。
F305	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。
F306	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。
F307	信息	Call of non-function ‘variable name’
	原因	变量名作为函数名使用。
F308	信息	Improper use of a typedef name
	原因	typedef 定义的名称使用方法不正确。
W309	信息	Unused ‘variable name’
	原因	在源文件中声明的变量从未使用过。
W310	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。
F311	信息	Number syntax
	原因	常量表达式非法。
F312	信息	Illegal octal digit
	原因	非法的八进制数。
F313	信息	Illegal hexadecimal digit
	原因	非法的十六进制数。
F314	信息	Too big constant
	原因	常量太大超出了表示范围，无法显示。

(4) 配置元素错误信息<编号从 301 开始> (2/3)

F315	信息	Too small constant
	原因	常量过小无法表示。
F316	信息	Too many character constants
	原因	字符常量超过了两个字符。
F317	信息	Empty character constant
	原因	字符常量‘ ’为空 。
F318	信息	No terminated string literal
	原因	在字符串最后缺少双引号“”。
F319	信息	Changing string literal
	原因	字符串文字被重写。
W320	信息	No null terminator in string literal
	原因	字符串文字的末尾缺少 null 字符。
F321	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。
F322	信息	Ellipsis requires three periods
	原因	编译器检测到“..”，但省略号需要三个点“...”。
F323	信息	Missing ‘delimiter’
	原因	分界符不正确。
F324	信息	Too many }’s
	原因	‘{’ 和 ‘}’ 没有正确配对。
F325	信息	No terminated comment
	原因	注释的最后没有使用“*/”来终止。
F326	信息	Illegal binary digit
	原因	非法的二进制数。
F327	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。
W328	信息	Unrecognized character escape sequence ‘character’
	原因	转义序列符号无法正确识别。
F329	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。
W330	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。
W331	信息	‘-ZL’ option specified – long is treated as int in this file
	原因	被指定的-ZL 选项。文件中的 Long 类型被当作 int 类型来进行处理。
W332	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。
W333	信息	‘-SM’ option specified – ignored ‘function attributes’ keyword in this file
	原因	指定了静态模式选项-SM，本文件中的函数属性被忽略。

(4) 配置元素错误信息<编号从 301 开始> (3/3)

F334	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。
W335	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 。
W339	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。
W340	信息	Unreferenced label 'label name'
	原因	已经定义的标签从来没有被引用过。

(5) 转换错误信息<编号从 401 开始>

W401	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。
F402	信息	Incompatible type conversion
	原因	在赋值语句中发生了非法的类型转换。
F403	信息	Illegal indirection
	原因	在整型表达式中使用了 * 操作符。
F404	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。
F405	信息	Illegal lvalue
	原因	非法的左值。
F406	信息	Cannot modify a const object 'variable name'
	原因	const 属性的变量被重写。
F407	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。
F408	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。
F409	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。
W410	信息	Illegal pointer conversion
	原因	指针和非指针类型的目标进行转换。
W411	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。
W412	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。
F413	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。
F414	信息	Expected pointer
	原因	需要一个指针。

(6) 表达式错误信息<编号从 501 开始> (1/3)

F501	信息	Expression syntax
	原因	表达式包含了一个语法错误。
F502	信息	Compiler limit : too many parentheses
	原因	在表达式中的括号嵌套超过 32 层。
W503	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。
W504	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。
W505	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。
F507	信息	Expected integral index
	原因	在数组的索引中只允许整型表达式。
W508	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。
W509	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。
W510	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。
W511	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。
F512	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。
F513	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。
F514	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。
F515	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。
F516	信息	Zero sized structure ‘structure name’
	原因	结构体的容量大小是 0。
F517	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。
F518	信息	Illegal structure / union comparison
	原因	两个结构体或共同体无法进行比较。

(6) 表达式错错误信息<编号从 501 开始> (2/3)

F519	信息	Illegal bit field operation
	原因	对位域的非法描述。
F520	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(->)。
F521	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。
W522	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。
F523	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。
F524	信息	‘&’ requires lvalue
	原因	A 的常量地址没有被获得。
F525	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。
F526	信息	‘&’ on register variable
	原因	某寄存器变量的地址无法获取。
F527	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。
W528	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。
F529	信息	Sizeof returns zero
	原因	Sizeof 表达式的值变成 0。
F530	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。
F531	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。
F532	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。
F533	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。
F534	信息	Divide check
	原因	/运算符或%运算符的除数是零。
F535	信息	Invalid pointer addition
	原因	两个指针不能相加。
F536	信息	Must be integral value addition
	原因	只有整数可以被和指针相加。

(6) 表达式的错误信息<编号从 501 开始> (3/3)

F537	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。
F538	信息	Illegal conditional operator
	原因	条件运算符的描述不正确。
F539	信息	Expected constant expression
	原因	需要常量表达式。
W540	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。
F541	信息	Function argument has void type
	原因	函数的参数类型是 void 型。
W543	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。
F544	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。
F546	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。
F549	信息	‘-SM’ option specified - recursive function is not allowed
	原因	因为指定了静态模式选项-SM，所以不允许使用递归函数。

(7) 语句错误信息<编号从 601 开始>

F602	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。
F603	信息	Compiler limit : too many labels
	原因	标签的数目超过了 33 个。
F604	信息	Case not in switch
	原因	Case 语句出现的位置不正确。
F605	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。
F606	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。
F607	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。
F608	信息	Default not in switch
	原因	default 语句出现的位置不正确。
F609	信息	More than one ‘default’
	原因	在 swich 语句中出现多个 default 语句。
F610	信息	Compiler limit : block nest level too deep
	原因	块嵌套超过了 45 层。
F611	信息	Inappropriate ‘else’
	原因	If 和 else 语句不匹配。
W613	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。
W615	信息	Statement not reached
	原因	此语句永远执行不到。
F617	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。
F620	信息	Break / continue error
	原因	break 和 continue 语句的位置不正确。
F621	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。
W622	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。
F623	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。

(8) 声明和函数定义的错误信息 <编号从 701 开始> (1/5)

F701	信息	External definition syntax
	原因	函数没有被正确定义。
F702	信息	Too many callt functions
	原因	声明为 callt 型的函数数量太多，最多可以定义 32 个 callt 函数。
	对策	减少 callt 函数的声明数量。
F703	信息	Function has illegal storage class
	原因	函数被指定的存储类是非法的。
F704	信息	Function returns illegal type
	原因	函数返回值的类型是非法的。
F705	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。
F706	信息	Parameter list error
	原因	函数参数列表有误。
F707	信息	Not parameter ‘character string’
	原因	在函数定义时声明的参数不正确。
W708	信息	Already declared symbol ‘variable name’
	原因	相同的“变量名称”符号已经被声明过。
F710	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。
F711	信息	Undeclared ‘variable name’; function ‘function name’
	原因	使用的变量未进行声明。
F712	信息	Declaration syntax
	原因	声明语句不符合语法规范。
F713	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。
W714	信息	Too many register variables
	原因	寄存器变量的声明太多。
	对策	减少寄存器变量的数量。至于可用的寄存器数量，敬请查阅 **C 编译器语言篇(U16655E)**手册的**第 11 章**。
F715	信息	Too many sreg variables
	原因	sreg 变量的声明太多。
F716	信息	Not allowed automatic data in noauto function
	原因	在 noauto 函数中不能使用自动变量。

(8) 声明和函数定义的错误信息 <编号从 701 开始> (2/5)

F717	信息	Too many automatic data in noauto or norec function
	原因	在 noauto 或 norec 函数中有太多自动变量。
	对策	减少 noauto 或 norec 函数中自动变量的数量。至于允许的使用数量，敬请查阅 **C 编译器语言篇 (U16655E)**手册的**第 11 章**。
F718	信息	Too many bit, boolean type variables
	原因	位变量和布尔型变量太多。
	对策	减少位变量、布尔型和__boolean 型变量的数量。至于允许的使用数量，敬请查阅 **C 编译器语言篇 (U16655E)**手册的**第 11 章**。
F719	信息	Illegal use of type
	原因	使用非法的类型名称。
F720	信息	Illegal void type for ‘identifier’
	原因	标识符用 void 进行声明。
W721	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。
F723	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。
F724	信息	Structure redefinition
	原因	定义了两个以上的同名结构体。
W725	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。
F726	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。
F727	信息	Unknown size structure / union ‘name’
	原因	结构体或共同体的大小容量未定义。
F728	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。
F729	信息	Compiler limit : structure / union nesting
	原因	结构体或共同体的嵌套层数超过 15。
F730	信息	Bit field outside of structure
	原因	位域的声明放在结构体之外。
F731	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。
F732	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。

(8) 声明和函数定义的错误信息 <编号从 701 开始> (3/5)

F733	信息	Negative bit field size
	原因	在位域声明中的位变量数量是负值。
F734	信息	Illegal enumeration
	原因	枚举类型声明不符合语法规范。
F735	信息	Illegal enumeration constant
	原因	非法的枚举常量。
F736	信息	Compiler limit : too many enumeration constants
	原因	枚举常量的数量超过 255。
F737	信息	Undeclared structure / union / enum tag
	原因	某个未声明的标记（tag）。
F738	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。
F739	信息	Expected constant
	原因	在用于数组声明的索引中使用了变量。
F740	信息	Negative subscript
	原因	数组的容量大小被指定为负值。
F741	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。
F742	信息	Compiler limit : too many array modifying
	原因	数组的声明超过了 12 维。
F743	信息	Array element type cannot be function
	原因	函数数组是不被允许的。
W744	信息	Zero sized array ‘array name’
	原因	定义的数组中，元素数量为 0。
W745	信息	Expected function prototype
	原因	函数原型未做声明。
F747	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。
W748	信息	A function is declared as a parameter
	原因	在声明中将函数用作参数。
W749	信息	Unused parameter ‘parameter name’
	原因	参数没有被使用。
F750	信息	Initializer syntax
	原因	初始化不符合语法规范。
F751	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。

(8) 声明和函数定义的错误信息 <编号从 701 开始> (4/5)

W752	信息	Undeclared initializer name ‘name’
	原因	初始名称未做声明。
F753	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。
F756	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。
F757	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。
F758	信息	Cannot initialize a function ‘function name’
	原因	该函数无法完成初始化。
F759	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。
W760	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。
W761	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。
W762	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。
W763	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。
W764	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。
F765	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。
F766	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。
F770	信息	Parameters are not allowed for interrupt function
	原因	中断函数不能带参数使用。
F771	信息	Interrupt function must be void type
	原因	中断函数必须是 void 类型。
F772	信息	Callt / noauto / norec / __pascal are not allowed for interrupt function
	原因	中断函数不能被声明 callt, noauto, norec, 或 __pascal 类型。
F773	信息	Cannot call interrupt function
	原因	中断函数不能被调用。

(8) 声明和函数定义的错误信息 <编号从 701 开始> (5/5)

F774	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。
F780	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。
F781	信息	'-SM' option specified - variable parameters are not allowed
	原因	因为指定了静态模式选项-SM，不允许使用变量参数。
F782	信息	'-SM' option specified – structure & union parameter is not allowed
	原因	因为指定了静态模式选项-SM，不允许使用结构体和共同体作为参数。
F783	信息	'-SM' option specified – structure & union return value is not allowed
	原因	因为指定了静态模式选项-SM，不允许使用结构体和共同体作为返回值。
F784	信息	'-SM' option specified - too many parameters of function
	原因	因为指定了静态模式选项-SM，函数的参数数量超过了 3 个/长度超过 6 字节。
F785	信息	'-SM' option specified - expected function prototype
	原因	因为指定了静态模式选项-SM，未找到函数原型声明。
W786	信息	'-SM' option specified - undeclared parameter in function prototype
	原因	因为指定了静态模式选项-SM，参数在函数原型声明中未做声明。
W787	信息	Bit field type is char
	原因	字符类型被指定为位域类型。
W792	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。
W793	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。
F799	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) 预处理命令的错误信息 <编号从 801 开始> (1/4)

F801	信息	Undefined control
	原因	以 # 开始的符号不能被识别为关键字。
F802	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。
F803	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。
W804	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。
F805	信息	Misplaced else or elif
	原因	#if，#ifdef，和#ifndef 与#else 和#elif 的配对不相符。
F806	信息	Misplaced endif
	原因	#if，#ifdef，和 #ifndef 与#endif 的配对不相符。
F807	信息	Compiler limit:too many conditional inclusion nesting
	原因	条件编译的嵌套层数超过 255。
F810	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。
F811	信息	Too long file name ‘file name’
	原因	文件名太长。
F812	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。
F813	信息	Compiler limit : too many include nesting
	原因	包含文件的嵌套层数超过 8。
F814	信息	Illegal macro name
	原因	非法的宏名。
F815	信息	Compiler limit: too many macro nesting
	原因	宏的嵌套层数超过 200。
W816	信息	Redefined macro name ‘macro name’
	原因	宏名称有重定义。
W817	信息	Redefined system macro name ‘macro name’
	原因	系统宏名称有重定义。
F818	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。
W819	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。

(9) 预处理命令的错误信息 <编号从 801 开始> (2/4)

F821	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。
F822	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‘)’。
F823	信息	Too long macro expansion ‘macro name’
	原因	宏展开式时使用的实际参数太长。
W824	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。
W825	信息	Macro recursion ‘macro name’
	原因	#define 定义的宏名称是递归的。
F826	信息	Compiler limit : too many macro defines
	原因	宏定义的数量超过 10,000。
F827	信息	Compiler limit : too many macro parameters
	原因	单个宏定义的调用参数超过 31 个。
F828	信息	Not allowed #undef for system macro name
	原因	系统宏指令名被#undef 指定停止。
W829	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。
F830	信息	No chip specifier : #pragma pc ( )
	原因	没有设备说明符。
F831	信息	Illegal chip specifier : #pragma pc (device type)
	原因	非法的设备说明符。
W832	信息	Duplicated chip specifier
	原因	多个相同的设备说明符。
F833	信息	Expected #asm
	原因	没有 #asm。
F834	信息	Expected #endasm
	原因	没有#endasm。
W835	信息	Too many characters in assembler source line
	原因	汇编源程序中的某行太长。
W836	信息	Expected assembler source
	原因	在 #asm 和 #endasm 之间没有汇编源程序。

(9) 预处理命令的错误信息 <编号从 801 开始> (3/4)

W837	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。
F838	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。
F839	信息	Unrecognized pragma VECT or INTERRUPT ‘character string’
	原因	有一个无法识别的#pragma VECT‘字符串’或 INTERRUPT‘字符串’。
W840	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。
F842	信息	Unrecognized pragma SECTION ‘character string’
	原因	有一个无法识别的#pragma SECTION‘字符串’。
F843	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。
F845	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。
W846	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。
F847	信息	Different NOBANK or LEAFWORK specified on same interrupt function ‘function name’
	原因	不同的 NOBANK 说明或 LEAFWORK 说明被指定给同一个中断函数。
W849	信息	#pragma statement is not portable
	原因	#pragma 语句不符合 ANSI 标准规范。
W850	信息	Asm statement is not portable
	原因	ASM 语句不符合 ANSI 标准规范。
W851	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。
W852	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。
F853	信息	Unrecognized pragma NAME ‘module name’
	原因	‘模块名’中有无法识别的字符。
W856	信息	Rechanged module name ‘module name’
	原因	指定了多个相同的模块名。

(9) 预处理命令的错误信息 <编号从 801 开始> (4/4)

W857	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。
F866	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）。
F867	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。
F868	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。
W869	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。
W870	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。
W871	信息	Data aligned after ‘variable name’ in ‘section name’
	原因	在变量名称分配到节名称之后，再执行数据对齐。
F899	信息	Character string specified by #error is output
	原因	指定了#error 字符串。

(10) 致命的文件 I/O 和运行非法操作系统的错误信息<从 901 开始> (1/2)

A901	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。
A902	信息	Cannot open ‘file name’
	原因	文件无法打开。
	对策	检查设备文件是否被安装在常规查询路径。这个路径可以使用 –Y 选项来指定。参照 **5.3 (18) 设备文件**中的查询路径的相关描述。
A903	信息	Cannot open overlay file ‘file name’
	原因	覆盖文件无法打开。
A904	信息	Cannot open temp
	原因	输入的临时文件无法打开。
A905	信息	Cannot create ‘file name’
	原因	产生了一个文件创建错误。
A906	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。
A907	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。
A908	信息	o available directory space
	原因	临时文件无法创建，因为驱动器上没有足够的目录区。
A909	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。
A910	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。
A911	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。
A912	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。
A913	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)。

(10) 致命的文件 I/O 和运行非法操作系统的错误信息<编号从 901 开始> (2/2)

A914	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。
W915	信息	Asm statement found. skip to jump optimize this function ‘function name’
	原因	检测到#asm 块或__asm 声明语句。这个函数没有进行跳转优化。执行 W837 对策。
F922	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，。
A923	信息	Illegal device file format
	原因	引用了旧格式的设备文件。

附录 A 样例程序

A.1 C 源程序模块文件

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

char    mark[SIZE+1];

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

            count = 0;

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

printf(s,i)
char *s;
int i;
{
            int j;
            char *ss;

            j = i;
            ss = s;
}
```

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

A.2 执行例程

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

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


sample\prime.c(18) : W745 Expected function prototype
sample\prime.c(20) : W745 Expected function prototype
sample\prime.c(26) : W622 No return value
sample\prime.c(37) : W622 No return value
sample\prime.c(44) : W622 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

(1) 汇编源程序模块文件

```
; 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
      EXTRN  _@cdisp
      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
      xor  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, 6
      mov[hl+7],a         ; i                   ;[INF] 2, 6

L0003:
      mova,[hl+6]  ; i                      ;[INF] 2, 6
      xch  a,x                            ;[INF] 1, 4
      mova,[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
      mova,#01H           ; i               ;[INF] 3, 6
      mov[de],a                             ;[INF] 1, 6
      mova,[hl+6]  ; i                      ;[INF] 2, 6
      xch  a,x                            ;[INF] 1, 4
      mova,[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   $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  ; -32568                ;[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  $L0015                          ;[INF] 2, 6

; line      17
    mov a,[hl+6]        ; i                 ;[INF] 2, 6
    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
    call !_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      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       ; -32568            ;[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                          ;[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= 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 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
```

(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
*/
```

(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 -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
```

(4) 错误列表文件

```
PRIME.C(     18) : W745 Expected function prototype
PRIME.C(     20) : W745 Expected function prototype
PRIME.C(     26) : W622 No return value
PRIME.C(     37) : W622 No return value
PRIME.C(     44) : W622 No return value


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

附录 B 注意事项列表

编号	注意事项
1	**[关于指定选项的注意事项]** (a) 当某选项不允许指定多个参数却被赋予了多个参数时，最后一个指定参数优先（有效）。 (b) 所有在-C 选项后的类型规范都不能忽略。如果被忽视，会产生一个异常中止错误。如果没有指定-C 选项，请确保在 C 源程序模块文件中输入#pragma pc(类型)。在编译期间，如果具体指定的选项和 C 源程序中的选项不同，则指定的选项优先。同时会输出一个警告信息。 (c) 如果帮助选项被指定，所有其它选项可以被忽视。
2	**[关于文件输出目标的注意事项]** 对于目标模块文件，只能输出为磁盘类型的文件。
3	**[关于错误信息的注意事项]** 当文件中发现一个语法错误，就会有一条出错信息关联到这个文件名。如果设备文件被指定到禁止位置，就会自动输出指定的字符串。在其它所有情况下，会附加驱动器、路径和文件后缀名。
4	**[源文件名的注意事项]** 在 CC78K0S 中，除去文件名后缀的部分（主文件名）默认被用作模块名。因此，使用的源文件名会有一些限制。. (a) 关于文件名的长度，在操作系统允许的范围内给文件名配置一个基本名和扩展名，基本名和扩展名之间用点(.)来间隔。当使用 PM+时，用点(.),来分隔基本名和扩展名，同时使用“.c”作为 C 源程序的扩展名。 (b) 文件基本名和后缀名的字符由系统允许的字符组成，除了圆括号(())，分号(;)和逗号(,)。应该注意的是连字符(-)可以被作为文件名或者路径的首字符。当使用 PM+时，指定的文件名或者路径名绝对不能包括空格和方括号（[ ]），也不能指定带有双字节字符的路径名，例如中文汉字。 (c) (#)符号不能出现在参数文件的文件名和路径名中。 (d) 在文件连接时，如果有基本名前 8 个字符相同的文件，系统将会报错。 (e) 如果使用 ID78K0S-NS 或 SM78K0S，文件基本名的字符可以是小写字母(a 到 z), 大写字母(A 到 Z)，数字(0 到 9)，下划线 (_)，和点(.)。
5	**[关于包含文件的注意事项]** 在包含文件中定义函数（除了函数声明），然后在 C 源程序中展开该文件是不可能的。 当在包含文件中出现定义，那么在源程序调试过程中会出现某些问题，比如错误的定义行显示等。

<table>
<tr><th>编号</th><th>注意事项</th></tr>
<tr><td>6</td><td>[输出汇编源程序的使用注意事项]

当 C 源程序包括了#asm 块或 __asm 语句的汇编语言时，装载模块文件产生的过程顺序是编译、汇编，然后连接。对下列几中使用情况要留心，当使用汇编器输出汇编源程序来进行汇编操作而不输出直接目标时，例如，当使用汇编语言进行描述时。

(a) 需要在#asm 块（在#asm 和#endasm 语句之间的部分）和__asm 语句中定义符号时，要使用 8 字符以内的字符串并以?L@开始（例如，?L@01，?L@sym，等）。然而，这些符号不能在被定义为外部的（PUBLIC 声明）。在#asm 块中和__asm 语句中不能定义段。如果定义的符号没有使用 8 字符以内并以?L@开始的字符串，那么在汇编过程中会产生 A114 异常中止信息。
(b) 在 C 源程序#asm 块中的变量是外部变量时，如果在 C 程序中没有对此变量引用，就不会有 EXTRN 声明，同时产生一个连接错误。因此如果 C 中没有引用，那么在#asm 块中要进行 EXTRN 声明。
(c) 如果 C 源程序中包含#asm 块和__asm 语句，指定-A 或–SA 编译选项来开启汇编描述，并且对输出的汇编源程序进行汇编。
当使用 PM+时，或者对源程序单独指定-A/-SA 选项，或者在通用选项中指定-A/-SA 选项。
(d) 当使用 PM+时，如果指定了-A 或-SA 编译选项，那么不论-O/-NO 选项如何，都会启动 RA78K0S。
(e) 当使用#pragma 命令改变段名称时，段名称不要和源程序基本名相同。否则汇编时会产生异常中止错误 A106。</td></tr>
<tr><td>7</td><td>[指定编译器选项-QC2 时的注意事项]

如果在 CC78K0S 中指定了-QC2 选项，常量和字符常量类型的可表示范围如下。

<table>
<tr><td>–128 到 +127</td><td>char 类型</td></tr>
<tr><td>128 到 255</td><td>unsigned char 类型</td></tr>
<tr><td>0U 到 255U</td><td>unsigned char 类型</td></tr>
<tr><td>大于 256</td><td>int 类型</td></tr>
<tr><td>到 –129</td><td>int 类型</td></tr>
<tr><td>‘\0’ 到 ‘\377’</td><td>char 类型</td></tr>
</table>
当指定-QC2 选项时，两个字符型常量的运算结果为字符型，两个无符号字符型常量的计算结果是无符号字符型。一个字符型型常量和一个无符号字符型常量的计算结果按无符号字符型来处理。
如果计算结果溢出，将两个常量都临时转换为某种可以正确表示结果的类型，或者指定-QC1(缺省)选项。临时转换防止数据类型被更改。</td></tr>
</table>

<table>
<tr><th>编号</th><th>注意事项</th></tr>
<tr><td>7</td><td>例) 当指定-QC2 选项时
<pre>int i;
i = 20*20 /*负值*/
i = (int)20*20 /*400*/</pre>
备注 然而，当指定-QU 选项时，所有字符型数据都被当作无符号字符型处理。范围在 ‘\200’ 到 ‘\377’ 的字符常量按无符号字符型来处理，值的范围从+128 到+255。</td></tr>
<tr><td>8</td><td>[可用的汇编程序包]

V1.50 版的 CC78K0S 和 V1.30 版或更新版本的汇编程序包配合使用。
由于支持长文件名，所以使用 RA78K0S V1.30 以前的版本可能会导致错误。</td></tr>
<tr><td>9</td><td>[使用网络时的注意事项]

当存储临时文件的目录被放在网络共享的文件系统中时，就可能会引起文件争用或是不正常的操作，这取决于所用的网络软件。设置选项和环境变量可以避免这种竞争。
使用 PM+时，不要在网络环境中使用 CC78K0S。</td></tr>
<tr><td>10</td><td>[创建连接定向文件]

当连接编译器生成的目标文件时，如果超出了目标设备的 ROM 或 RAM 区域范围之外；或者需要把代码或数据放在任意指定的地址时，可以创建一个连接定向文件，并且在连接的时候指定-D 选项。
关于创建连接定向文件的详细信息，敬请参阅 RA78K0S 汇编包操作篇用户手册 (U16656E) ，编译器自带有 lk78k0s.dr（在 SMP78K0S 目录下）文件。

例) 要把一个没有初始值的外部变量从某个特定 C 源程序文件放到外部存储器中

1. 在 C 源程序的开始，为这个没有初始值的外部变量改变段名称。
<pre>#pragma section @@DATA EXTDATA
M</pre>
注意 通过改变启动例程来对更名的段进行初始化，ROMization 也同样要在启动例程中描述。
2. 创建一个连接定向文件。
<lk78k0s.dr>
<pre>memory EXTRAM : (0F000h, 00200h)
merge EXTDATA := EXTRAM</pre>
创建连接定向文件时注意下面几点。</td></tr>
</table>

编号	注意事项
10	1. 当连接时使用-S 自动生成选项来生成堆栈符号，推荐通过连接定向文件的存储器定位来保留堆栈区域，并指定唯一的堆栈名称。如果省略了对该区域的命名，保留出的区域就被用作 RAM 中的堆栈区（SFR 区域除外）。 **例)** 向连接定向文件 lk78k0s.dr 中添加 memory EXTRAM : (0F000h, 00200h) memory STK : (0FB00H, 20H) merge EXTDATA := EXTRAM (命令行) > lk78k0s s0s.rel prime.rel -bcl0s.lib -SSTK –Dlk78k0s.dr 2. 当自己定义的存储器区域进行连接时可能会遇到以下错误。 “*** ERROR F206 Segment ‘xxx’ can’t allocate to memory-ignored.” **[原因]** 由于已定义的存储器区域空间不足，无法存放指定段。 **[应对]** 应对步骤大致分为以下三步。 1. 检查无法分配的程序段所需空间 (参阅.map 文件)。 2. 在第一步检查到的段大小的基础上，增加定向文件中对应段的预留空间。 3. 对定向文件指定-D 选项并连接。 然而，因为在步骤 1 中发生错误的段类型不同，用来检查段空间的方法也有如下几种方法。 (1) 当程序段在编译过程中被自动生成时， 借助连接过程中创建的 map 文件来检查段的空间大小。 (2) 如果这个段是用户创建的 借助汇编列表文件检查未分配的段空间大小(.prn)。
11	**[使用 va_start 宏时的注意事项]** 在 stdarg.h 中定义的 va_start 宏操作是不确定的（因为函数第一个参数的偏移量不同） ·指定了第一个参数后，使用 va_starttop ·指定了第二个及以后的参数后，使用 va_start 宏。

编号	注意事项
12	**[引用特殊功能寄存器(SFR)常地址时的注意事项]** 如果通过常地址来引用 16-位 SFR，访问 8-位单元会产生非法代码，所以请使用 SFR 名称来访问。
13	**[启动例程和库]** (a) 使用系统提供的启动例程和库，版本要和可执行程序中的文件保持一致。(cc78k0s.exe 或 cc78k0s)。 (b) 对于支持浮点的函数比如 sprintf，vprintf 和 vsprintf，如果已经指定了转换说明符“%f”, “%e”, “%E”, “%g”或“%G”的转换结果比指定精确度还准确，那么转换值就向下舍去。即使指定为“%g”“%G”的转换结果值大于精确度，仍然会执行“%f”转换。 对于 sscanf 和 scanf 函数，指定转换说明符为“%f”, “%e”, “%E”, “%g”，或“%G”，如果在转换期间没有读入有效的字符，那么转换的结果就是+0。如果转换结果是“±”, 那么±0 就被认为是转换结果。 **[预防方法]** 无
14	**[使用 ID78K0S-NS 调试源程序的注意事项]** 当调用 pascal 函数时，Next 命令的操作和单步操作相同，返回（Return）命令会返回发起调用的函数，等。当指定了编译选项-ZR 后，所有函数都变为 pascal 函数。因此，永远不要执行 Next 命令。
15	**[使用 SM78K0S 调试源程序的注意事项]** 当调用 pascal 函数时，不要执行 Next 命令。否则，程序就会失控。当指定编译选项-ZR 时，所有的函数都变为 pascal 函数。因此，当指定-ZR 时，确保永远不要执行 Next 命令。
16	**[执行 ROMization]** ROMization 的内容包括配放初始值，例如那些在 ROM 中有初始值的外部变量，然后在系统运行期间将这些值拷贝到 RAM 中。在 CC78K0S1.50 版本中，默认会生成 ROMization 相关代码。因此，在连接时执行含有 ROMization 的启动例程是很有必要的。 下列是 C 编译器提供的启动例程，都包括有 ROMization 处理。 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

编号	注意事项
16	**[使用实例]** 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: 运行时刻库，标准库 -S 选项是堆栈符号 (_@STBEG, _@STEND)自动生成选项。 **注意** **·确保在开始时连接启动例程。** **·创建库时，同 CC78K0S 提供的库分开创建，同时在连接过程中指定它的优先级高于编译程序库。** **·不要向 CC78K0S 库中添加用户函数。** **·当使用浮点库(CL0SF.LIB)时，包含 ROMization 处理的启动例程必须连接到标准库和浮点库。** 当使用支持浮点的 sprintf， sscanf，printf，scanf，vprintf，和 vsprintf 函数时 **例)** -BMYLIB.LIB -BCL0SF.LIB -BCL0S.LIB 当使用不支持浮点的 sprintf，sscanf，printf，scanf，vprintf，和 vsprintf 函数时 **例)** -BMYLIB.LIB -BCL0S.LIB -BCL0SF.LIB
17	**[生成堆栈区域符号(-S)]** 在 CC78K0S 中，用户无法为堆栈区保留空间。 为了保留堆栈区的空间，在连接时需要指定-S 选项。 在 PM+中，当源文件中包含 C 源程序时，-S 选项会自动指定。
18	**[ROM 代码]** 当使用 ROM 代码时，指定目标转换选项为-R 或-U，例如–r，-u0FFH， -R：根据地址顺序排列十六进制文件目录 -U 填充值：向 ROM 区的空白处填充指定值。
19	**[帮助说明选项]** 在 PM+中，编译器选项--，-?和-H 都会被忽略，它们用来对选项作简单解释。 在 Tool 菜单下各个的< xxx Option >设置对话框中单击“帮助”按钮，就可以看到对应的帮助文档。
20	**[-LL 选项说明]** 使用 PM+时，-LL 选项能够接受的最大数是 32767。如果指定的数字大于 32767，那么-LL 要和其他选项联合使用。
21	**[关于符号名称长度的注意]** 在 ID78K0S-NS V1.01 和 SM78K0S V1.42 或更早的版本中，符号名称的长度不要超过 127 个字符。

编号	注意事项
22	**[使用 PM+的注意]** (a) 用户创建的参数文件 当 PM+指定使用用户创建的参数文件时，那些内容被装载到 PM+创建的参数文件中。当建立参数文件时，注意以下几点，否则在建立（build）时会发生错误。 ·指定的文件名和 PM+创建的参数文件要同名。 ·设备类型描述选项(-c)，设备文件搜索路径描述选项(-y)和源文件都不要指定。 ·并不对用户创建的参数文件中所描述的选项进行有效性验证。 (b) <汇编选项> 对话框 不要指定-C，-F 和-Y 选项和源文件，否则在编译执行期间会输出一个错误。 对于<汇编选项>对话框中指定的选项并不作有效性验证，因此如果有描述错误，在执行建立（build）时会产生错误。 (c) 包含文件的从属关系 用 PM+创建 MAKE 文件时，会检查包含文件的从属关系，条件语句如 #if 会被忽视。因此有可能把非必需的包含文件误当作必需的文件。 如果用注释或字符串形式来描述，那么这些包含文件就会像无依赖关系一样被正确判定。 **例)** #if 0 #include "header1.h" /* Dependence relationship judged to exist */ #else / * ! zero */ #include "header2.h" #endif /* #include "header3.h" */ 在从属关系检查中，header1.h 被判定为必需要被编译的。如果 header1.h 文件存在，那么 header1.h 就会引入到 PM+的” ProjectWindow”中。 **[预防方法]**无。但是,这对建立（build）过程没有影响 。 (d) 项目工程文件 如果在<Compiler Options>对话框下的<<Startup Routine>>标签页中的[Using Startup Routine]和[Using Startup Routine]选择框被选中，但是没有按下<Compiler Options>对话框中的[OK]或[Apply]按钮，那么实际上它们不会起作用。 **[预防方法]** 按照下面的流程来检查 PM+的设置 <1> 打开<Compiler Options>对话框。 <2> 选择<<Startup Routine>> 标签页。 <3> 检查[Using Startup Routine]和 [Using Fixed Area of Standard Library]的设置情况。 <4> 按下[OK] 或 [Apply] 按钮（按下这些按钮就会反映出正确的配置）。 (e) 工程相关文件的设置 可以从 PM+的[Project]菜单或工程窗体中的"Add Project-Related File"添加或删除编译器属性的例行程序和标准库。 从<Compiler Options> 对话框的<<Startup Routine>>标签页中进行编译器属性的启动例程和标准库的设置。 (f) 包含在方括号内([ ])的文件名称和路径名称无法处理。

编号	注意事项
23	**[关于原型声明的注意事项]** 如果一个函数原型声明没有包括函数类型描述，就会产生(F301, F701)错误。 **例)** f (void) ;　　/* F301 : Syntax error */ 　　/* F701 : External definition syntax */ **[预防方法]** 加上函数类型描述 **例)** int　f (void) ;
24	**[关于错误信息输出的注意事项]** 如果关键字中有拼写错误，且位于函数外的行首处，那么报告的错误行号会有偏差，还会输出一个不适当的错误提示。 **例)** extren int i ;　　/* extern 拼写错误，不会有错误指向这里*/ /* comment */ void f (void) ; [EOF]　　/* Error such as F712 */ **[预防方法]** 无
25	**[关于预处理命令中注释的描述]** 在描述预处理命令时，如果有注释和函数类型宏放在同一行，且位于预处理命令之前或夹杂其中，那么会导致错误（F803, F814, F821, 等）。 **例)** /* com1 */ #pragma　sfr　　/* F803 */ /* com2 */ #define ONE 1　　/* F803 */ #define /* com3 */ TWO 2　　/* F814 */ #ifdef /* com4 */ ANSI_C　　/* F814 */ /* com5 */ #endif #define　SUB(p1,　/* com6 */ p2)　p2 = p1　　/* F821 */ **[预防方法]** 将注释内容移至预处理命令之后。 **例)** #pragma　sfr　　/* com1 */ #define ONE 1　　/* com2 */ #define TWO 2　　/* com3 */ #ifdef　ANSI_C　　/* com4 */ #endif　　/* com5 */ #define SUB(p1,　p2)　　p2 = p1　　/* com6 */

编号	注意事项

26	**[关于使用 structure, union,或 enum 特征标记的注意事项]**
	
	如果在函数原型声明中未经定义就使用(structure,union,或 enum)特征标记，若满足下面条件(1)，会产生警告，如果满足下面条件(2)，会输出错误。
	(1) 如果特征标记出现在函数的参数声明中，并定义了一个指向结构体或共同体的指针，当此函数被调用时，会输出 W510 警告信息。
	例)
	void func (int , struct st) ;
	
	struct st {
	char memb1;
	char memb2;
	} st [ ] = {
	{ 1, 'a' } , { 2, 'b' }
	} ;
	
	void caller (void) {
	func (sizeof (st) / sizeof (st[0]) , st); /* W510 Pointer mismatch */
	}
	(2) 如果特征标记用于参数声明中的返回值类型说明，并指定为 structure, union, 或 enum 类型，则会输出 F737 错误信息。
	例)
	void func1(int , struct st) ; /* F737 Undeclared structure/union/enum tag */
	struct st func2 (int) ; /* F737 Undeclared structure/union/enum tag */
	struct st {
	char memb1;
	char memb2;
	} ;
	[预防方法] 预先定义 structure，union，或 enum 的特征标记。

<table>
<tr><th>编号</th><th>注意事项</th></tr>
<tr><td>27</td><td>
[关于函数中的数组、结构体或共同体初始化的注意事项]

只有静态变量地址、常量或字符串可以才能完成数组、结构体或共同体的初始化。
例)
void f (void) ;
void f (void) {
  char *p, *p1, *p2 ;
  char *ca[3] = { p , p1 , p2 } ; /* 错误(F750) */
}
[预防方法] 用赋值语句来代替初始化。
例)
void f (void) ;
void f (void) {
  char *ca[3] ;
  char *p, *p1, *p2 ;
  ca[0] = p ; ca[1] = p1 ; ca[2] = p2 ;
}
</td></tr>
<tr><td>28</td><td>
[关于外部 callt 函数的注意事项]

如果在函数表初始化时引用了外部 callt 函数的地址，而且此 callt 函数在这个模块中被调用，那么汇编产生的汇编列表是非法的，还会导致一个错误。
例)
callt extern void funca (void) ;
callt extern void funcb (void) ;
callt extern void funcc (void) ;

static void (* const func [ ]) ( ) = {
  funca , funcb , funcc
} ;
callf void func2 (void) {
  funcc ( ) ;
  funcb ( ) ;
  funca ( ) ;
}
[预防方法] 将函数表和函数调用模块分开。
</td></tr>
</table>

<table>
<tr><th>编号</th><th>注意事项</th></tr>
<tr><td>29</td><td>

[关于函数返回值为结构体的注意事项]

当函数返回值是结构体，在返回返回值的过程中产生了一个中断，如果在中断服务程序中又调用了这个函数，那么中断服务结束后得到的返回值是非法的。

例）

```
struct str {
  char c ;
  int i ;
  long l ;
} st ;

struct str func ( ) {
  /* Interrupt occurrence */
       :
}

void main ( ) {
  st = func ( ) ;    /* Interrupt occurrence */
}
```

在上述服务过程中，如果 func 函数在中断服务程序中被调用，st 可能会被损坏。

[预防方法] 无
</td></tr>
<tr><td>30</td><td>

[关于共同体初始化的注意事项]

若共同体的成员有结构体、共同体或数组，初始化过程中的语法嵌套会导致 F750 错误。

例）

```
 struct Ss {
       int   d1, d2 ;
 } ;
 union Au {
       struct Ss t1;
  } u = { { 1, 2 } } ;   /* F750   Initializer syntax */
```

[预防方法] 对共同体初始化时不要使用嵌套。

例）

```
struct Ss {
  int   d1, d2 ;
} ;
union Au {
  struct Ss t1;
} u = { 1, 2 } ;
```

</td></tr>
</table>

附录 C CC78K0S 的限制列表

本章详细描述了 CC78K0S 的限制，以及如何避免这些限制。

编号	限制情况概述
1	对块内声明为 extern 的外部变量进行初始化不会发生错误。但是，汇编源程序中的调试信息是不正确的。
2	某变量和块内声明为 extern 的变量同名，对两者的连接有时是非法的。
3	如果有某类型是用 typedef 命令定义的，如果此类型用于函数原形声明或者用于带 const / volatile 类型修饰符的声明，typedef 的展开是非法的，而且会导致一个错误。
4	多维数组未指定长度，则可能有时无法正常运行。
5	若有函数的返回值是带有参数的函数地址，这些参数不能引用。当引用该参数时不会发生错误，但会输出非法代码。
6	有符号型位域被当作无符号型位域进行处理。

C.1 关于限制的细节和预防办法

限制 1

对块内声明为 extern 的外部变量进行初始化不会发生错误。但是，汇编源程序中的调试信息是不正确的。

[描述]

由于和 ANSI C 语言不兼容，对块内声明为 extern 的外部变量进行初始化会产生一个错误，但这样描述并不是一个错误。编译器对定义时有初始值的外部变量进行解释并输出对应代码。

编译器输出的目标中的调试信息是正确的，但是汇编源程序中的调试信息不正确。

[实例再现]

```
int   i;
void   f(void) {
          extern int   i = 2;
}
```

[预防办法] 无

[适用版本] **V1.00 至 V1.50 的所有版本**

限制 2

将一个变量和在块中外部声明的和它同名的变量绑定在一起有时是不合法的。

[描述]

在下列任何一种情况下将一个变量和在块中外部声明的同名变量绑定在一起是非法的。

(1) 当块内声明为 extern 的变量和块外声明为静态的变量有相同的名称
因为没有错误也没有绑定，所以当这个变量被引用时会输出非法代码。

[实例再现]

```
void   f(void) {
          extern int   i;
          i = 1;                    /* Illegal code output */
}
static int   i;
```

(2) 当一个在块内声明为 extern 的变量和块外未声明为静态的变量有相同的名称，非静态变量的位置处于 extern 变量之后。
即使没有绑定，也会输出非法代码。

[实例再现]

```
void   f(void) {
           extern int   i;
           i = 1;                          /* Illegal code output */
       }
int   i;
```

(3) 当在块内声明为 extern 的变量和在块外未用 extern 声明的变量有着相同的名称，非外部变量的位置处于块之前，并且在包含此 extern 变量的块内还有一个自动变量也是同样的名称。
即使在块外的变量和在块内声明为 extern 的变量没有绑定，也会输出非法代码。

[实例再现]

```
int i = 1;
void   f(void) {
           int   i;
           {
                   extern int   i;
                   i = 1;                  /* Illegal code output */
           }
}
```

(4) 在块内声明为 extern 的变量和在另一个块内声明为 extern 的变量有着相同的名称。
即使没有绑定，也会输出非法代码。

[实例再现]

```
void   f1(void) {
           extern int   i;
           i = 2;
}
void   f2(void){
           extern int   i;
           i = 3;
}
```

[预防办法] 无

[适用版本] **V1.00 至 V1.50 的所有版本**

限制 3

如果有某类型是用 typedef 命令定义的，如果此类型用于函数原形声明或者用于带 const / volatile 类型修饰符的声明，typedef 的展开是非法的，而且会导致一个错误。

[描述]

如果有某类型是用 typedef 命令定义的，如果此类型用于函数原形声明或者用于带 const / volatile 类型修饰符的声明，typedef 的展开是非法的，而且会导致一个错误。

[实例再现 1]

```
typedef int    FTYPE();

FTYPE    func;
int    func(void);                          /* F713 Redefined 'func' */
```

[实例再现 2]

```
typedef int    VTYPE[2];
typedef int    *VPTYPE[3];

const VTYPE    *a;
const int    (*a)[2];                       /* F713 Redefined 'a' */
volatile VPTYPE    b[2];
volatile int *volatile    b[2][3];         /* F713 Redefined 'b' */
```

[预防办法] 无

[适用版本] **V1.00 至 V1.50 的所有版本**

限制 4

多维数组未指定长度，可能有时无法正常运行。

[描述]

多维数组未指定长度，可能有时无法正常运行。

[实例再现 1]

```
char    c[][3]={{1},2,3,4,5};              /* Illegal code */
```

[实例再现 2]

```
char    c[][2][3]={"ab","cd","ef"};        /* Error (F756) */
```

[预防办法]

定义多维数组的维度大小。

[适用版本] **V1.00 至 V1.50 的所有版本**

限制 5

若有函数的返回值是带有参数的函数地址，这些参数不能引用。当引用该参数时不会发生错误，但会输出非法代码。

[描述]

若有函数的返回值是带有参数的函数地址，这些参数不能引用。当引用该参数时不会发生错误，但会输出非法代码。

[实例再现]

```
char *c;
int   *i;
void (*f1(int *))(char *);
void (*f2(void))(char *);
void (*f3(int *))(void);

void main() {
        (*f1(i))(c);            /* Correct description (W510) */
        (*f1(i))(i);            /* Incorrect description */
        (*f2( ))(c);            /* Correct description (W509) */
        (*f2( ))( );            /* Incorrect description (W509) */
        (*f3(i))( );            /* Correct description (W509) */
        (*f3(i))(i);            /* Incorrect description */
}
```

W509 或 W510 是正确描述的输出信息。对于产生警告的描述不输出任何值。但是，输出的代码是正常的。

```
void (*f4( ))(int p) {
        p++;                    /* Incorrect description */
}
```

对一个可能产生错误的描述，错误不会被输出，而是产生非法代码。

[预防办法] 无

[适用版本] **V1.00 至 V1.50 的所有版本**

限制 6

有符号型位域被当作无符号型位域进行处理。

[描述]

有符号型位域被当作无符号型位域进行处理。

[预防办法] 无

[适用版本] **V1.00 至 V1.50 的所有版本**

附录 D 索引

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z