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

RA78K0S Ver. 2.00

汇编包

语言篇

目标设备

78K0S 微控制器

文档编号. U17390CA2V0UM00 (第 2 版)
发行日期 2009 年 1 月

[备忘录]

Windows,是 Microsoft Corporation 在美国和其他国家的注册商标或商标。
UNIX 是 X/Open Company Limited 在美国及其他国家的注册商标。
SunOS 和 Solaris Sun 是 Microsystems, Inc.的商标。

- **本文档所刊登的内容有效期截至 2009 年 1 月。将来可能未经预先通知而更改。在实际进行生产设计时，请参阅各产品最新的数据表或数据手册等相关资料以获取本公司产品的最新规格。**
- 并非所有的产品和/或型号都向每个国家供应。请向本公司销售代表查询产品供应及其他信息。
- 未经本公司事先书面许可，禁止复制或转载本文件中的内容。否则因本文档所登载内容引发的错误，本公司概不负责。
- 本公司对于因使用本文件中列明的本公司产品而引起的，对第三者的专利、版权以及其它知识产权的侵权行为概不负责。本文件登载的内容不应视为本公司对本公司或其他人所有的专利、版权以及其它知识产权作出任何明示或默示的许可及授权。
- 本文件中的电路、软件以及相关信息仅用以说明半导体产品的运作和应用实例。用户如在设备设计中应用本文件中的电路、软件以及相关信息，应自行负责。对于用户或其他人因使用了上述电路、软件以及相关信息而引起的任何损失，本公司概不负责。
- 虽然本公司致力于提高半导体产品的质量及可靠性，但用户应同意并知晓，我们仍然无法完全消除出现产品缺陷的可能。为了最大限度地减少因本公司半导体产品故障而引起的对人身、财产造成损害（包括死亡）的危险，用户务必在其设计中采用必要的安全措施，如冗余度、防火和防故障等安全设计。
- 本公司产品质量分为：

  “标准等级”、“专业等级”以及“特殊等级”三种质量等级。

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

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

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

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

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

（注）

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

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

前言

本手册面向于用户正确理解 **RA78K0S 汇编包（**以下称 **RA78K0S）**中每个程序的基本功能，以及描述源程序的方法。

本手册不包括如何操作 RA78K0S 的各个程序。因此，理解了该手册的内容后，请阅读 **RA78K0S 汇编包用户操作手册（U17391E）**（以下称**操作**），以运行汇编包内的每个程序。

本手册中与 RA78K0S 有关的描述适用于 1.50 版本。

[目标读者]

该手册面向理解用于开发微控制器（78K0S 系列）的功能和指令的用户。

[组织]

该手册包括下面 6 个章节和若干附录：

第一章　概述

概述 RA78K0S 的所有基本功能

第二章　如何描述源程序

概述如何描述源程序，并解释汇编器的操作。

第三章　伪指令

解释如何编写和使用伪指令，并辅以应用实例进行说明

第四章　控制指令

解释如何编写和使用控制指令，并辅以应用实例进行说明

第五章　宏

解释所有宏的功能，包括宏定义、宏引用和宏展开。

宏伪指令在**第三章 伪指令**中解释。

第六章　产品应用

推荐一些的描述源程序的方法。

附录

包括保留字列表、伪指令列表、最佳性能表、特征和索引。

本手册不涉及指令组的细节。有关这些指令的更多细节，请参考正对其进行软件开发的微控制器（指令版）用户手册。

对于指令的结构，也请参考正对其进行软件开发的微控制器（指令版）用户手册。

[如何阅读本手册]

对于第一次使用汇编器的读者，建议从手册的**第一章 概述**读起。 至于那些对汇编程序有总体概念的读者可跳过手册的**第一章概述**。但是，还是请读者阅读 **1.2 程序开发前的提示**和**第二章 如何描述源程序**。

希望了解汇编器伪指令和控制指令的读者可分别阅读**第三章 伪指令**和**第四章 控制指令**。每一章都详细描述了各伪指令或指令的格式、功能、用途和应用实例。

[规则]

下列符号和缩写适用于整个手册：

:	重复同一格式
[ ]:	方括号内的字符可省略。
{ }:	选择在{}内的某一项
“ ”:	括在双引号””内的字符是字符串。
‘ ’:	括在单引号’’内的字符是字符串。
( ):	圆括号之间的字符是字符串
< >:	尖括号之间的字符（主要用于标题）是字符串
__:	下划线用于表示一个要点或输入字符串。
␣:	表示一个空格
∆:	表示一个或多个空白字符或 TAB 键。
∇:	表示没有或多个空格或 TAB 键 (如，空格可以省略).
/:	字符分隔符
~:	连续性
[↵]:	表示回车键

注: 文档中条目的脚注以**注**来表示

注意事项: 特别需要注意的信息

备注: 补充信息

[相关文档]

与该手册有关的文档（用户手册）列于下表。

在该手册中伪指令的相关文档包括基础版本。

但是，基础版本不按如下方式标记。

开发工具的相关文档（用户手册）

文档名称		文档编号
CC78K0S Ver. 2.00 C 编译器	操作篇	U17416E
	语言篇	U17415E
RA78K0S Ver. 2.00 汇编包	操作篇	U17391E
	语言篇	本手册
	结构化汇编语言	U17389E
SM+ 系统仿真器	操作篇	U18601E
	用户开放接口	U18602E
SM78K 系列 Ver.2.52 系统仿真器	操作篇	U16768E
ID78K0S-NS Ver. 2.52 集成调试器	操作篇	U16584E
ID78K0S-QB Ver. 3.00 集成调试器	操作篇	U17287E
PM+ Ver. 6.30 项目管理器		U18416E

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

[备忘录]

目录

第一章　概述 ... 14
1.1 汇编器简介 ... 14
什么是汇编器？15
可重新定位汇编器 17
1.2 程序开发前的提示 ... 19
RA78K0S 的最高性能特性 19
1.3 RA78K0S 的特性 ... 21

第二章　如何描述源程序 ... 22
2.1 基本配置 ... 22
模块头 23
模块体 24
模块尾 24
源程序的整体配置 25
描述示例 26
2.2 描述方法 ... 29
配置 29
字符集 30
符号字段 32
助记忆字段 36
操作字段 36
注释字段 40
2.3 表达式和运算符 ... 41
2.3.1 运算符 ... 42
算术运算符 ... 43
逻辑运算符 ... 46
关系运算符 ... 48
移位运算符 ... 51
字节分割运算符 ... 53
特殊运算符 ... 54
其它运算符 ... 56
2.4 运算约束 ... 57
运算符和重定位属性 57
运算符和符号属性 60
如何检查对运算的约束 62
2.5 位位置说明符 ... 63
位位置说明符 ... 64
2.6 操作数的特征 ... 66
操作数的值的大小和地址范围 66
指令所需的操作数的大小 67
操作数的符号属性和重定位属性 68

第三章　伪指令 ... 72
3.1 概述 ... 72
3.2 段定义伪指令 ... 73
CSEG ... 75
DSEG ... 79
BSEG ... 83
ORG ... 87
3.3 符号定义伪指令 ... 89
EQU ... 90
SET ... 93
3.4 内存初始化和区域保留伪指令 ... 95
DB ... 96
DW ... 98

DS ... 100
DBIT ... 102
3.5 链接伪指令 ... 103
EXTRN ... 104
EXTBIT ... 106
PUBLIC ... 108
3.6 目标模块名称声明伪指令 ... 110
NAME ... 111
3.7 自动分支指令选择伪指令 ... 112
BR ... 113
3.8 宏伪指令 ... 115
MACRO ... 116
LOCAL ... 118
REPT ... 121
IRP ... 123
EXITM ... 125
ENDM ... 128
3.9 汇编终止伪指令 ... 130
END ... 131

第四章 控制指令 ... 132
4.1 概述 ... 132
4.2 处理器类型定义控制指令 ... 134
PROCESSOR ... 135
4.3 调试信息输出控制指令 ... 136
DEBUG / NODEBUG ... 137
DEBUGA / NODEBUGA ... 138
4.4 交叉引用列表输出定义控制指令 ... 139
XREF / NOXREF ... 140
SYMLIST / NOSYMLIST ... 141
4.5 包含控制指令 ... 142
INCLUDE ... 143
4.6 汇编列表控制指令 ... 145
EJECT ... 146
LIST / NOLIST ... 148
GEN / NOGEN ... 150
COND / NOCOND ... 152
TITLE ... 154
SUBTITLE ... 156
FORMFEED / NOFORMFEED ... 159
WIDTH ... 160
LENGTH ... 161
TAB ... 162
4.7 条件汇编控制指令 ... 163
IF / _IF / ELSEIF / _ELSEIF / ELSE / ENDIF ... 164
SET / RESET ... 169
4.8 其它控制指令 ... 171

第五章 宏 ... 172
5.1 概述 ... 172
5.2 宏的使用 ... 173
宏定义 173
宏引用 174
宏展开 175
应用示例 175
5.3 宏内的符号 ... 176
5.4 宏操作符 ... 178

第六章 产品应用 ... 180
6.1 启动汇编器时节省时间并减少故障 ... 180
6.2 如何开发具有高内存利用率的程序 ... 181

附录 A 保留字列表 ... 182

附录 B　伪指令列表 S ... 184

附录 C　索引 ... 186

图形的列表

图形编号 标题 页码

1-1 RA78K0S 汇编器包 ... 14
1-2 汇编器流程 ... 15
1-3 应用微控制器的产品的开发过程 ... 16
1-4 重汇编调试 ... 18
1-5 使用现有模块进行程序开发 ... 18
2-1 源模块的配置 ... 22
2-2 源模块的整体配置 ... 25
2-3 源模块配置示例 ... 25
2-4 举例程序的配置 ... 26
2-5 组成语句的字段 ... 29
3-1 段的内存定位 ... 74
3-2 代码段的重置 ... 75
3-3 数据段的重置 ... 79
3-4 位段的重置 ... 83
3-5 绝对段的定位 ... 87
3-6 两模块间符号的关系 ... 103

图表的列表

表编号 标题 页码

1-1 汇编器的最佳性能特征 ... 19
1-2 连接器的最佳性能特征 ... 20
2-1 可在模块头中描述的指令 ... 23
2-2 字母数字字符 ... 30
2-3 特殊字符 ... 30
2-4 符号类型 ... 32
2-5 汇编器自动生成的段的名称 ... 34
2-6 符号的属性和值 ... 35
2-7 表示数字常量的方法 ... 37
2-8 可在操作字段中描述的特殊字符 ... 38
2-9 操作符类型 ... 41
2-10 操作符优先顺序 ... 42
2-11 重置属性的类型 ... 57
2-12 根据重置属性组合项和操作符 (可重置项) ... 58
2-13 根据重置属性组合项和操作符 (外部引用项) ... 59
2-14 操作中符号属性类型 ... 60
2-15 根据符号属性组合项和操作符 ... 61
2-16 X (第一项) 和 Y (第二项) 的组合 ... 64
2-17 根据重置属性组合第一项和第二项 ... 65
2-18 位符号的值 ... 65
2-19 指令的操作数的值的范围 ... 66
2-20 伪指令的操作数的值的范围 ... 67
2-21 作为操作数被描述的符号的属性 ... 69
2-22 作为伪指令操作数被描述的符号的属性 ... 70
3-1 伪指令列表 ... 72
3-2 段定义方法和内存地址定位 ... 73
3-3 CSEG 的重置属性 ... 76
3-4 CSEG 的默认段名 ... 77
3-5 DSEG 的重置属性 ... 80
3-6 DSEG 的默认段名 ... 81
3-7 BSEG 的重置属性 ... 84
3-8 定位计数器 (绝对) ... 84
3-9 定位计数器 (可重置) ... 84
3-10 符号值和位偏移显示 ... 85
3-11 BSEG 的默认段名 ... 85
3-12 代表了比特值操作符的表示格式 ... 91
4-1 控制指令列表 ... 132
4-2 控制指令和汇编器选项 ... 133
A-1 保留字的类型 ... 182
A-2 保留字的列表 ... 183
B-1 伪指令的列表 ... 184

第一章　概述

本文介绍了 RA78K0S 在微控制器软件开发中的作用以及 RA78K0S 的特性 .

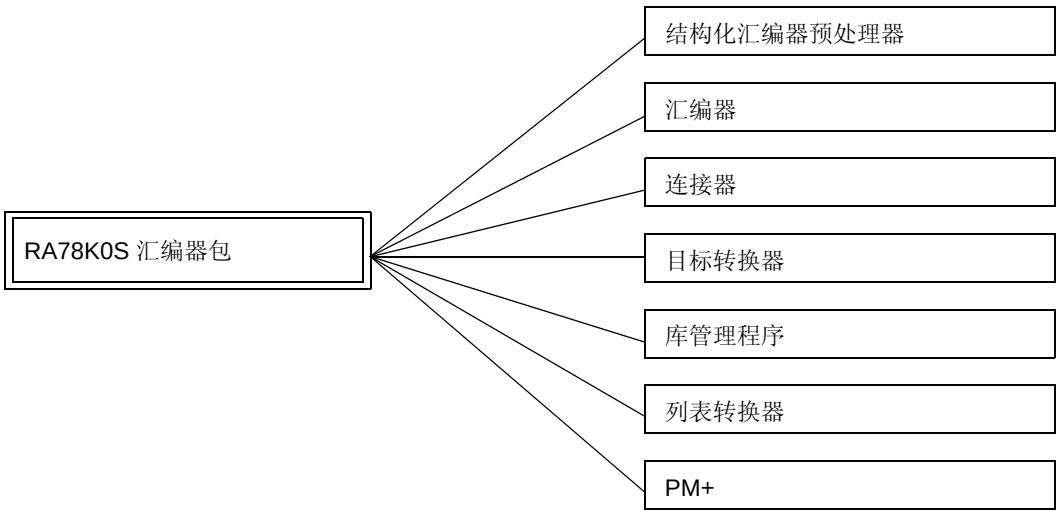
1.1　汇编器简介

RA78K0S 汇编器包（后面称为“RA78K0S”）是一个综合术语，指把采用汇编语言为 78K0S 系列微控制器编写的源程序翻译成机器码的一系列程序 .

RA78K0S 包括 6 个程序：结构化汇编器预处理器，汇编器，连接器，目标转换器，库管理程序和列表转换器 .

另外 , RA78K0S 还提供了一个 PM+, 对在 Windows 上执行程序编辑，编译 / 汇编，连接和调试等一系列操作起辅助作用 .

图 1-1　RA78K0S 汇编器包



1.1.1 什么是汇编器？

(1) 汇编语言和机器语言

汇编语言是针对微控制器的最基础的编程语言．

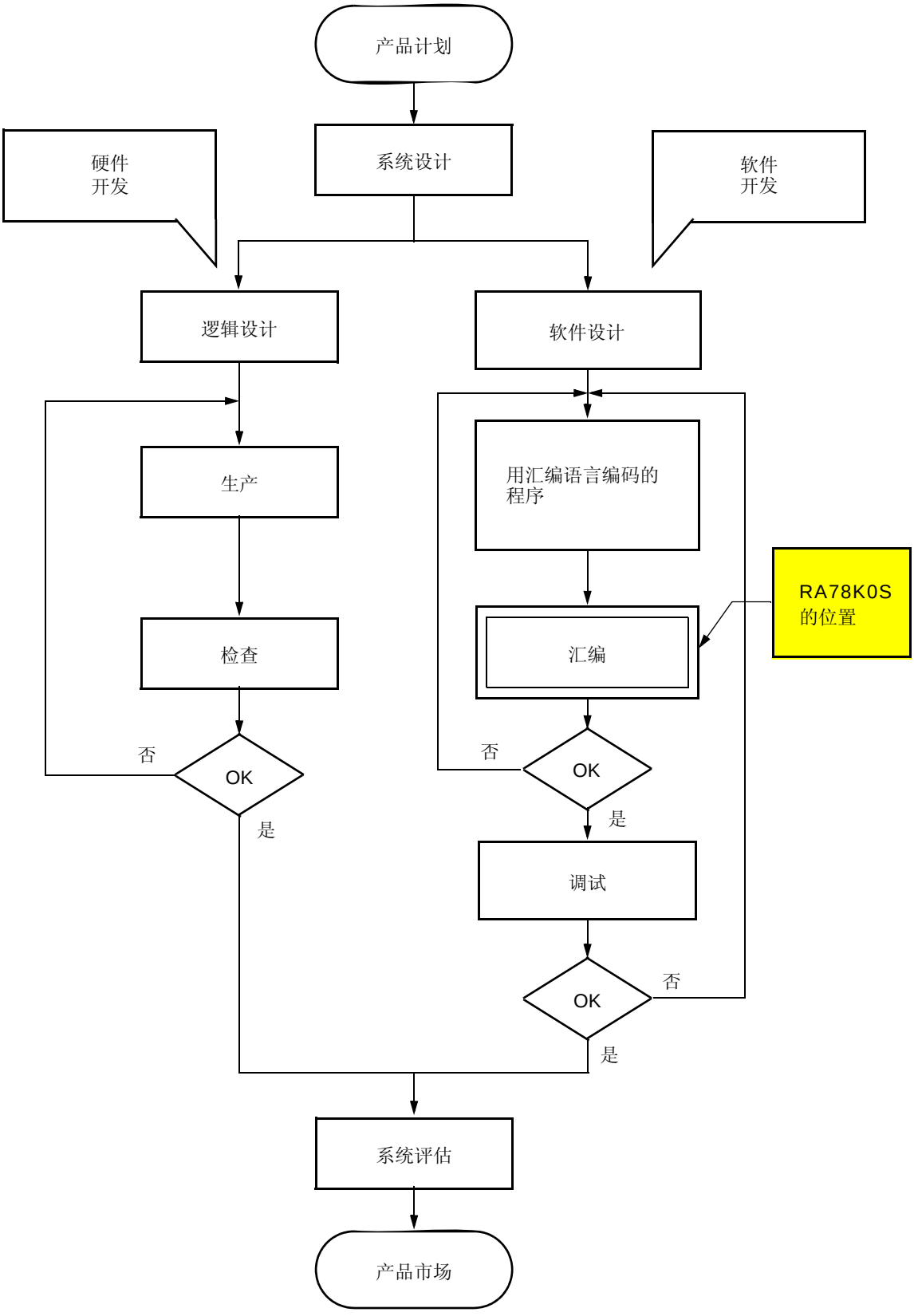
对于微控制器中的微处理器来说，为了完成它的工作，需要一些程序和数据．这些程序和数据必须由用户写入微控制器的内存．由微控制器处理的程序和数据是二进制数据的集合，被称为机器语言．然而，对于用户来说，机器语言代码很难记忆，经常引起错误．幸运的是，存在这样一些方法，即采用英语缩写词或记忆术来表示原始机器代码的含义，从而使用户易于理解．使用这种符号编码的基础编程语言系统被称为汇编语言．然而，由于机器语言是微控制器能处理的程序中仅有的编程语言，所以需要其他程序将汇编语言编写的程序翻译成机器语言．这个程序就是汇编器．

图 1-2 汇编器流程

(2)　应用微控制器的产品的开发和 RA78K0S 的作用

图 1-3 举例说明了 “产品开发过程中的汇编”的位置 .

图 1-3　应用微控制器的产品的开发过程



1.1.2 可重新定位汇编器

由汇编器从源语言翻译生成的机器语言在使用前写入到微控制器的存储器中 . 为此需写入的各机器语言指令在存储器中的位置必须已经确定 .

因此 , 要对汇编器汇编的机器语言增加一些信息 , 说明各机器语言指令在存储器中的位置 .

根据给机器语言指令定位地址的方法 , 汇编器可粗略的分为 “绝对汇编器” 和 “可重新定位汇编器” .

- 绝对汇编器

  绝对汇编器是把由汇编语言汇编生成的机器语言指令定位至绝对地址 .

- 汇编可重新定位汇编器

  在可重新定位汇编器中 , 汇编语言汇编生成的机器语言指令确定临时地址 . 随后 , 由连接器确定绝对地址 .

过去 , 当程序用绝对汇编器创建时 , 通常程序员必须同时完成编程 . 然而 , 如果一个大型程序的所有组成部分作为单个实体被创建 , 则程序会变复杂 , 使程序的分析和维护都很困难 . 为避免这种情况 , 开发这类大型程序时 , 将其分成一些子程序 , 称之为模块 , 其中每个都是一个功能单元 . 这种编程技术称为模块化编程 .

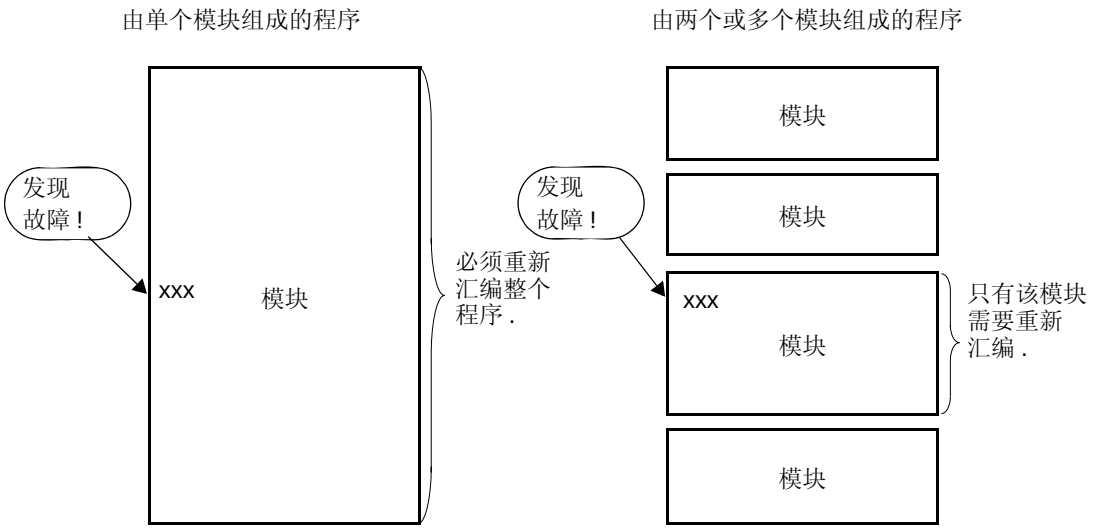
可重新定位汇编器是适用于模块化编程的汇编器 , 它具有以下优点 :

(1) 提高开发效率

同时写一个大型程序是很困难的 . 在这种情况下 , 将程序分成具有单个功能的模块 , 且允许两个或更多的程序员并行开发子程序从而提高开发效率 .

此外 , 如果在程序中发现任何故障 , 则没有必要汇编整个程序而是更正程序的一部分 , 只需要汇编必须更正的那个模块 . 这就缩短了调试时间 .

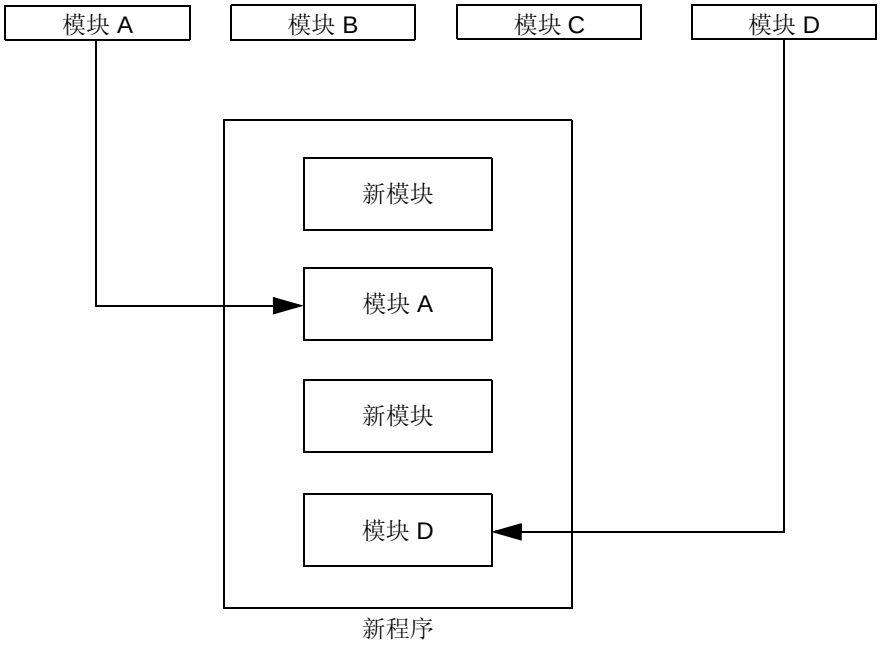
图 1-4 重汇编调试



(2) 资源的利用

已经预先创建好的具有高可靠性和多用途的模块可以在创建其他程序时再次使用 . 如果积累这些具有多用途的模块作为软件资源 , 则在开发一个新程序时就能节省时间和劳力 .

图 1-5　使用已存在模块进行程序开发



1.2 程序开发前的提示

在开始开发程序前，参看下列要点。

1.2.1 RA78K0S 的最高性能特性

(1) 汇编器的最高性能特性

表 1-1 汇编器的最高性能

项目	最高性能特性	
	Windows 版本	UNIX 版本
符号的数量 (局部 + 全局)	65535 符号	65535 符号
交叉引用列表能够输出的符号数量	65534 符号	65534 符号
一个宏引用的宏程序体的最大容量	1 M 字节	1 M 字节
所有宏程序体的总容量	10 M 字节	10 M 字节
一个文件内程序段的数量	256 段	256 段
一个文件内宏和 include 的指定	10000	10000
一个包含文件内宏和 include 的指定	10000	10000
重定位数据注 1	65535 项	65535 项
行数量数据	65535 项	65535 项
一个文件中 BR 指令的数量	32767 指令	32767 指令
每行中的字符数	2048 字符注 2	2048 字符注 2
符号长度	256 字符	256 字符
定义分支名称的数量注 3	1000	1000
分支名称的字符长度注 3	31 字符	31 字符
一个文件内包含文件的嵌套级数	8 级	8 级

注 1. “重定位数据”是指当汇编器不能确定符号的值时传输至连接器的数据。
例如，当用 MOV 指令引用一个外部引用符号时，在 .rel 文件中生成两项重定位数据。

注 2. 这包括回车和分页符。如果一行中描述了 2049 或更多个字符，则输出警告信息并忽略第 2049 个字符或超出部分。

注 3. 通过 SET / RESET 指令并使用 $IF 等来设置分支名称为真或假。

(2) 连接器的最高性能特性

表 1-2 连接器的最高性能

项目	最高性能特性	
	Windows 版本	UNIX 版本
符号的数量 (局部 + 全局)	65535 符号	65535 符号
同一段的行数量数据	65535 项	65535 项
程序段的数量	65535 段	65535 段
输入模块的数量	1024 个模块	1024 个模块

1.3 RA78K0S 的特性

RA78K0S 具有如下特性：

(1) 宏功能

当同一指令组必须在一个源程序中反复出现时，可以通过为指令组赋予一个宏名称来定义宏．

通过使用这种宏功能，可以提高编码效率和程序的可读性．

(2) 分支指令的优化功能

RA78K0S 具有一个自动选择分支指令的伪指令 "BR （分支） ".

为了创建一个具有高存储效率的程序，必须根据分支指令的分支目的范围描述一个字节分支指令．但是，对于程序员来说，通过注意每个分支的分支目的范围来描述分支指令是很棘手的．通过描述 BR 伪指令，汇编器根据分支目的范围生成适当的分支指令．这就是分支指令的优化功能．

(3) 条件汇编功能

通过该功能，一部分源程序可根据预先确定的条件指定为汇编或非汇编．

如果在源程序中描述调试语句，则可通过设置条件汇编分支来选择是否将调试语句翻译成机器语言．当不再需要调试语句时，汇编源程序无需对源程序进行较大的修改．

第二章　　如何描述源程序

本章介绍了源程序的描述方法，表达式和操作符．

2.1　基本配置

当把源程序分成几个模块进行描述时，每个模块变成输入到汇编器的单元，称之为源模块（如果是由单个模块组成的源程序，”源程序”就是”源模块”).

每个成为输入到汇编器的单元的源模块主要由以下三部分组成：

(1) 模块头
(2) 模块体
(3) 模块尾

图 2-1　源模块的配置

2.1.1　模块头

在模块头中，可描述下表 2-1 中显示的控制指令 . 注意，这些控制指令仅能在模块头中描述 .

注意，模块头可以忽略 .

表 2-1　可在模块头中描述的指令

可描述的项目	解释说明	在本手册中的章 / 节
与汇编器选项具有相同功能的控制指令	与汇编器选项具有相同功能的控制指令如下： - PROCESSOR - XREF / NOXREF - DEBUG / NODEBUG, DEBUGA / NODEBUGA - TITLE - SYMLIST / NOSYMLIST - FORMFEED / NOFORMFEED - WIDTH - LENGTH - TAB	第四章　控制指令
由诸如 C 编译器和结构化汇编器预处理器等高级程序输出的特殊控制指令	由诸如 C 编译器和结构化汇编器预处理器等高级程序输出的特殊控制指令如下： - TOL_INF - DGS - DGL	

2.1.2　模块体

在模块体中，不能描述下列指令：

- 与汇编器选项具有相同功能的控制指令

所有其他伪指令，控制指令和指令可在模块体中描述．

必须将模块体分成单元进行描述，这些单元称为”段”.

根据与每个段对应的伪指令，用户可定义以下四种段：

(1) 代码段
　　必须采用 CSEG 伪指令定义．

(2) 数据段
　　必须采用 DSEG 伪指令定义．

(3) 位段
　　必须采用 BSEG 伪指令定义．

(4) 绝对段
　　必须采用 CSEG, DSEG 或 BSEG 伪指令，为重定位属性（AT 定位地址）指定一个定位地址来定义．该段也可用 ORG 伪指令定义．

模块体可以由任何段的组合来构成．

但是，应在代码段前定义数据段和位段．

2.1.3　模块尾

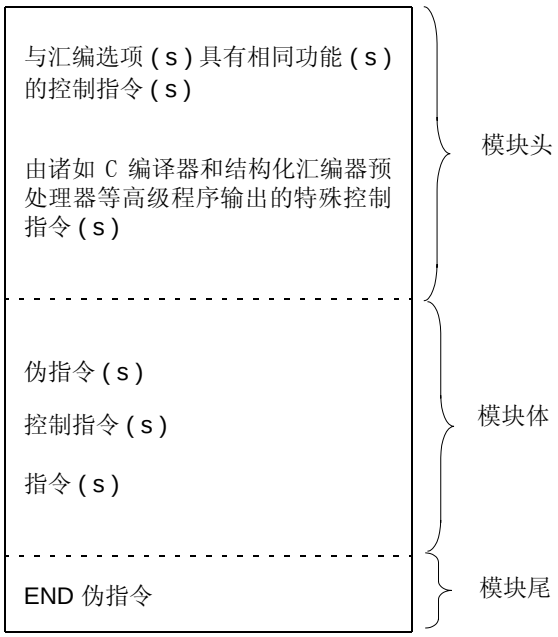
模块尾表明源模块的结束．END 伪指令必须在这部分中描述．

如果在 END 伪指令后描述除注释，空格，制表符或换行符之外的任何字符，那么汇编器将输出警告信息并忽略 END 伪指令后描述的字符．

2.1.4　源程序的整体配置

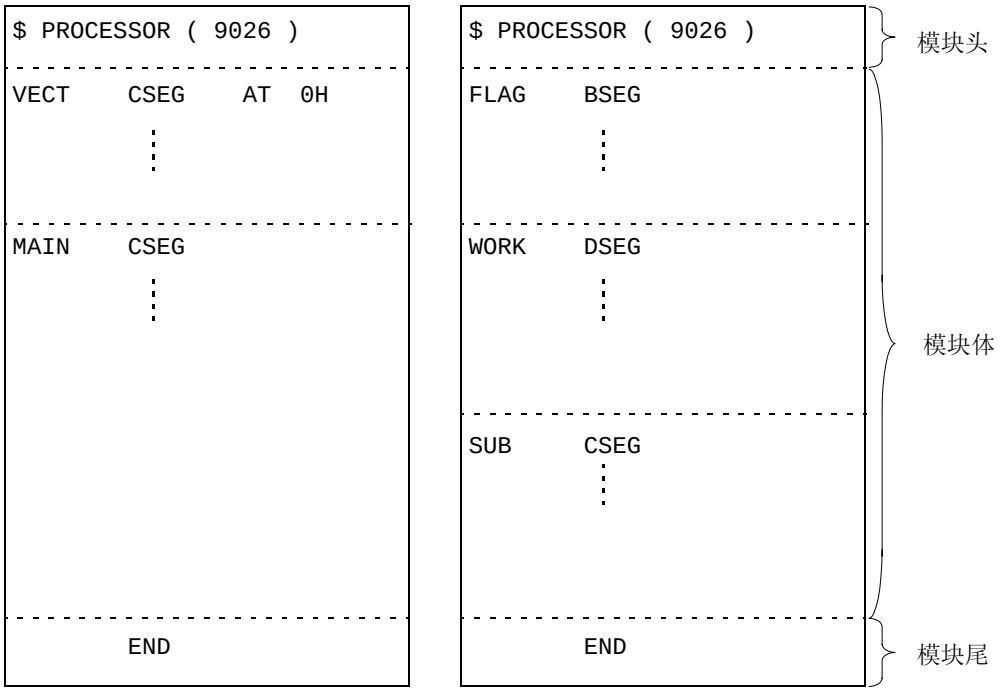
源模块（源程序）的整体配置如下所示.

图 2-2　源模块的整体配置



简易源模块配置的示例如图 2-3 所示.

图 2-3　源模块配置示例

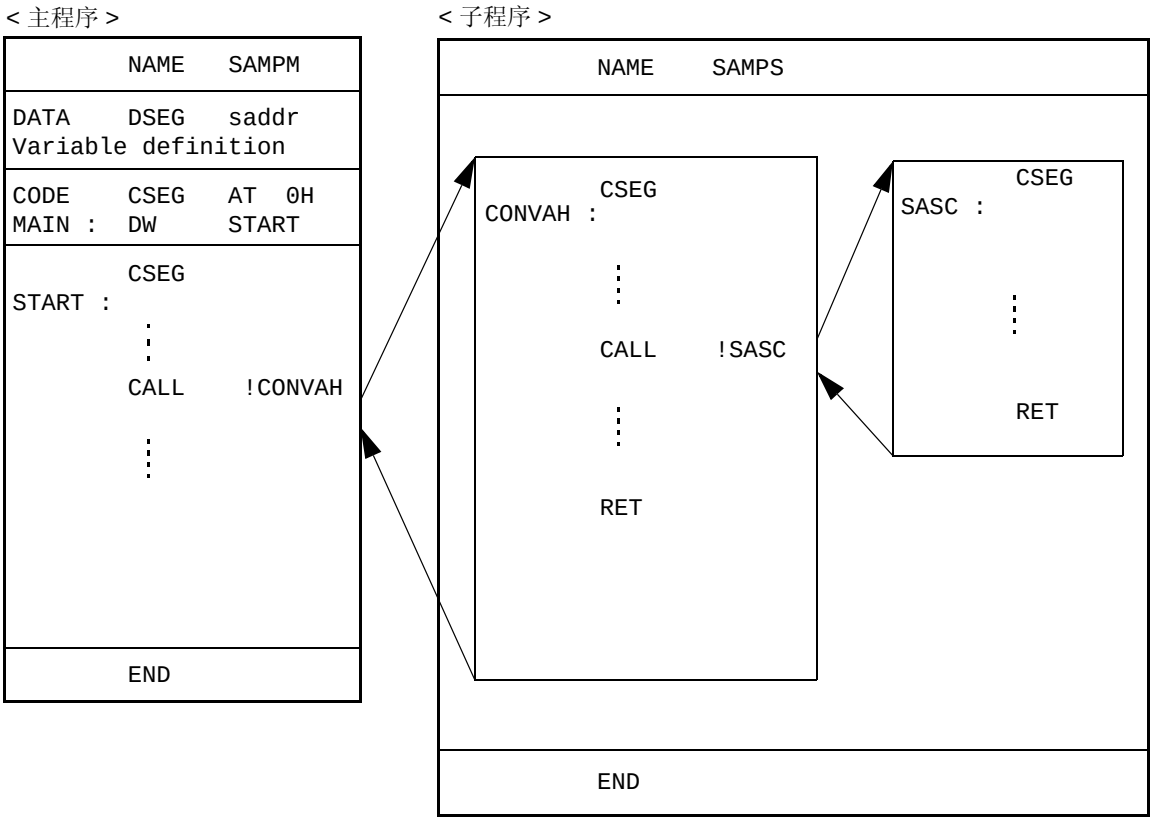


2.1.5　描述示例

本小节中，描述了源模块（源程序）作为样本程序的实例．

样本程序的配置简单说明如下．

图 2-4　样本程序的配置



< 主程序 >

```
        NAME    SAMPM           ; (1)
; ***************************************
;       HEX -> ASCII Conversion Program
;       main-routine
; ***************************************

        PUBLIC  MAIN , START    ; (2)
        EXTRN   CONVAH          ; (3)
        EXTRN   _@STBEG         ; (4)

DATA    DSEG    saddr           ; (5)
HDTSA : DS      1
STASC : DS      2

CODE    CSEG    AT 0H           ; (6)
MAIN :  DW      START

        CSEG                    ; (7)
START :
                                ; 芯片初始化
        MOVW    AX , #_@STBEG
        MOVW    SP , AX

        MOV     HDTSA , #1AH
        MOVW    HL , #HDTSA     ; 在 HL 寄存器中设置 hex 2 代码数据
        CALL    !CONVAH         ; 转换 ASCII <- HEX
                                ; 输出 BC- 寄存器 <- ASCII 码

        MOVW    DE , #STASC     ; 设置 DE <- 存储 ASCII 码表
        MOV     A , B
        MOV     [ DE ] , A
        INCW    DE
        MOV     A , C
        MOV     [ DE ] , A
        BR      $$

        END                     ; (8)
```

(1) 声明模块名

(2) 声明从其他模块引用的符号作为一个外部引用符号

(3) 声明在其他模块中定义的符号作为一个外部引用符号

(4) 声明从连接器的 "-s" 选项中生成的栈解决符号，作为一个外部引用符号（如果链接时没有指定 "-s" 选项，则会发生错误）

(5) 声明一个数据段的开始 （位于 saddr 区）

(6) 声明代码段的开始 （位于由地址 0H 开始的绝对段）

(7) 声明代码段的开始 （指绝对段的结尾）

(8) 声明模块结束

< 子程序 >

```
        NAME    SAMPS           ; (1)
; ************************************************
;       HEX -> ASCII 转换程序
;       子程序
;
;   输入条件    : ( HL ) <- hex 2 码
;   输出条件    : BC- 寄存器 <- ASCII 2 码
; ************************************************

        PUBLIC  CONVAH          ; (2)

        CSEG                    ; (3)
CONVAH :
        MOV     A , [ HL ]
        ROR     A , 1
        ROR     A , 1
        ROR     A , 1
        ROR     A , 1
        AND     A , #0FH        ; hex 高位码
        CALL    !SASC
        MOV     B , A           ; 存储结果

        XOR     A , A
        XCH     A , [ HL ]
        AND     A , #0FH        ; hex 低位码
        CALL    !SASC
        MOV     C , A           ; 存储结果
        RET

 ; *********************************************
 ;       子程序    转换 ASCII 码
 ;
 ;   输入 Acc ( 低 4 位 )  <- hex 码
 ;   输出 Acc                <- ASCII 码
 ; *********************************************

        CSEG
SASC :
        CMP     A , #0AH        ; 检测 hex 码 > 9
        BC      $SASC1
        ADD     A , #07H        ; 偏移 ( +7H )
SASC1 :
        ADD     A , #30H        ; 偏移 ( +30H )
        RET

        END                     ; (4)
```

(1) 声明模块名

(2) 声明从其他模块引用的符号作为一个外部定义符号

(3) 声明代码段的开始

(4) 声明模块结束

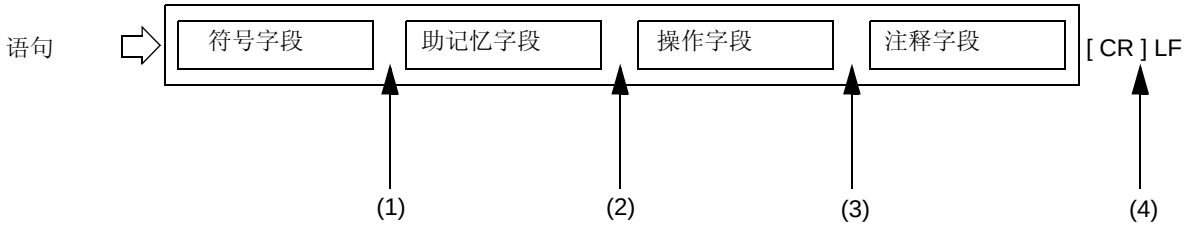
2.2　描述方法

2.2.1　配置

源程序由语句构成 .

每个语句由四个字段组成，如图 2-5 所示 .

图 2-5　组成语句的字段



(1) 符号字段和助记忆字段之间必须用冒号 （：）或一个或多个空格或制表符分开 （使用冒号或是空格取决于在助记词字段中描述的指令） .

(2) 助记忆字段和操作字段之间必须用一个或多个空格或制表符隔开 . 根据助记忆字段中描述的指令，可能不需要操作字段 .

(3) 如果使用注释字段，则必须在前面加分号 （；） .

(4) 每行之间必须用 LF 码界定开 （一个 CR 码后可能紧跟着一个 LF 码） .

一个语句必须在一行内进行描述 . 每行最多可以描述 2048 个字符 （包括 CR 和 LF） .

每个 TAB 或者独立的 CR 按一个独立字符计数 . 如果描述了 2049 或更多个字符，则输出警告信息并忽略第 2049 个字符及之后的任何字符 . 但是，第 2049 个及后续更多的字符将输出到汇编列表中 .

一个独立的 CR 将不会输出至汇编列表 .

也有可能描述如下代码行：

- 伪行 （没有语句描述的行）
- 仅由符号字段构成的行
- 仅由注释字段构成的行

2.2.2　字符集

可以在源文件中描述的字符分为如下三类：

- 语言字符
- 字符数据
- 注释字符

(1) 语言字符

语言字符是用于在源程序中描述指令的字符 . 语言字符集包括字母字符、数字字符和特殊字符 .

表 2-2　字母数字字符

名称		字符
数字字符		0 1 2 3 4 5 6 7 8 9
字母字符	大写字母	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
	小写字母	a b c d e f g h i j k l m n o p q r s t u v w x y z

表 2-3　特殊字符

字符	名称	主要应用	
? @ _	问号 约号 下划线	相当于字母字符的符号 相当于字母字符的符号 相当于字母字符的符号	
Blank HT (09H) : ; CR (0DH) LF (0AH)	 制表符代码 逗号 冒号 分号 回车代码 换行代码	分隔符号	各字段的定界符 相当于空格的字符 操作数的定界符 标签定界符 表示注释字段开始的符号 表示一行结束的符号 （在汇编器中被忽略） 表示一行结束的符号
+ - * / . (,) < , > =	加号 减号 星号 斜线 句点 左括号和 右括号 不等号 等号	汇编器 运算符	加法运算符或正号 减法运算符或负号 乘法运算符 除法运算符 位位置指定符 指定将要执行的算术运算顺序的符号 关系运算符 关系运算符
'	单引号	- 表示字符常量开始或结束的符号 - 表示完整宏参数的符号	
$ & # ! []	美元符号 〞与〞符号 # 号 感叹号 方括号	- 表示位置计数器的符号 - 表示相当于汇编器选项的控制指令开始的符号 - 指定相对寻址的符号 串联符号 （用于宏程序体） 指定立即寻址的符号 指定绝对寻址的符号 指定间接寻址的符号	

(1) 字符数据

“字符数据”指用来描述串常量、字符串和控制指令（TITLE, SUBTITLE, INCLUDE）的字符.

备注 1　除“00H”外的所有字符均可使用（包括 kanji（日本汉字）；根据操作系统的不同，代码也可能不同）. 如果已经描述了“00H”，那么会导致结果出错，并且在结束单引号标志（'）之前的后续字符都将被忽略.

备注 2　如果已描述了任何非法字符，则为了输出至汇编列表汇编器将用“!”替换这些非法字符（独立的 CR（0DH）码将不会输出至汇编列表）.

备注 3　对于 Windows 系统，汇编器把代码“1AH”解释为文件的结束（EOF），因此，该代码不能作为输入数据的一部分.

(2) 注释字符

“注释字符”是指用来描述注释语句的字符.

备注　可以在注释语句中使用的字符和用于字符数据的字符集相同. 但是，即使已经描述了代码“00H”，也不会导致结果出错. 相反，汇编器用“!”来代替非法字符并将其输出至汇编列表.

2.2.3　符号字段

符号在符号字段中描述 . 术语 “ 符号 ” 是指赋予数字数据或地址的名称 .

通过使用符号，可以使源程序的内容更容易理解 .

(1) 符号类型

根据用途和定义方法，符号可分为图 2-4 中所示的类型 .

表 2-4　符号类型

符号类型	用途	定义方法
名称	在源程序中用作数字数据或地址 .	在 EQU、 SET 或 DBIT 伪指令的符号字段中描述该类型 .
标签	在源程序中用作地址数据 .	在一个符号后加后缀冒号 （：） 来定义该类型 .
外部引用名称	由一个模块定义而由另外的模块用作引用符号 .	在 EXTRN 或 EXTBIT 伪指令的操作符字段中描述该类型 .
段名	在链接操作期间使用的符号	在 CSEG、 DSEG、 BSEG 或 ORG 伪指令的符号字段中描述该类型 .
模块名	在符号调试期间使用	在 NAME 伪指令的操作符字段中描述该类型 .
宏名	在源程序中用作宏引用 .	在 MACRO 伪指令的符号字段中描述该类型 .

(2) 符号描述惯例

所有符号必须遵循如下规则进行描述：

(a) 符号必须由字母数字字符和可用作等效字母字符的特殊字符 （?, @, and _）构成 .

数字字符 0 ～ 9 中的任何一个都不能用作符号的第一个字符 .

(b) 符号最多由 255 个字符组成 . 超出最大符号长度的字符将被忽略 .

(c) 预留字不能作为符号 . 表 A-2 中指出了预留字 .

(d) 同一符号不能定义多次 （但是，由 SET 伪指令定义的名称可以由 SET 伪指令重新定义） .

(e) 汇编器区分大小写字符 .

(f) 当在符号字段中描述一个标签时，在标签后面必须立即描述 “：” （冒号） .

< 正确的符号描述示例 >

```
CODE01   CSEG              ; “CODE01” 是段名 .
VAR01    EQU     10H       ; “VAR01” 是名称 .
LAB01 :  DW      0         ; “LAB01” 是标签 .
         NAME    SAMPLE    ; “SAMPLE” 是模块名 .
MAC1     MACRO             ; “MAC1” 是宏名 .
```

< 错误的符号描述示例 >

```
1ABC     EQU     3         ; 数字字符不能用作符号的第一个字符 .
LAB      MOV     A , R0    ; ″LAB″ 是一个标签 , 必须用冒号 (:) 从助记忆字段中隔开 .
FLAG :   EQU     10H       ; 名称中不需要加冒号 (:).
```

< 过长的符号示例 >

```
A123456789B12 至 Y1234567890123456      EQU      70H
                  256     ; 字符 ″6″ 超出了最大字符长度 (255 字符 ), 将被忽略 .
                          ; 符号将被定义为 ″A123456789B12 至 Y123456789012345″.
```

< 仅由一个符号组成的语句举例 >

```
ABCD :                    ; ″ABCD″ 将被定义为一个标签 .
```

(3) 有关符号的一些注意事项

符号 “??RAnnnn (n = 0000 to FFFF)”是指每次宏程序体内部开发本地符号时由汇编器自动替代的符号 . 注意不要定义两次该符号 .

当段名不是由段定义伪指令指定时，汇编器自动生成段名 . 这些段如表 2-5 所示 .

重复定义段名会导致错误发生 .

表 2-5　汇编器自动生成的段名

段名称	伪指令	重定位属性
?An (n = 0000 ~ FFFF)	ORG 伪指令	(无)
?CSEG	CSEG 伪指令	UNIT
?CSEGUP		UNITP
?CSEGT0		CALLT0
?CSEGFX		FIXED
?CSEGIX		IXRAM
?DSEG	DSEG 伪指令	UNIT
?DSEGUP		UNITP
?DSEGS		SADDR
?DSEGSP		SADDRP
?DSEGIH		IHRAM
?DSEGL		LRAM
?DSEGDSP		DSPRAM
?DSEGIX		IXRAM
?BSEG	BSEG 伪指令	UNIT

(4) 符号属性

所有名称和标签都有一个值和一个属性 .

值表示定义的数字数据或地址数据本身的值 .

段名、模块名和宏名没有值 .

一个符号具有的属性称为符号属性，必须是下面表 2-6 中的 8 种类型之一 .

表 2-6　符号的属性和值

属性类型	分类	值
NUMBER	– 分配给数字常量的名称 - 由 EXTRN 伪指令定义的符号 – 数字常量	十进制表示法 : 0 至 65535 十六进制表示法 : 0H 至 FFFFH
ADDRESS	– 作为标签定义的符号 - 由 EQU 和 SET 伪指令作为标签定义的名称	十进制表示法 : 0 至 1048575 十六进制表示法 : 0H 至 FFFFH
BIT	- 作为位值定义的名称 - BSEG 中的名称 - 由 EXTBIT 伪指令定义的符号	saddr 区域
CSEG	由 CSEG 伪指令定义的段名	这些属性类型没有值 .
DSEG	由 DSEG 伪指令定义的段名	
BSEG	由 BSEG 伪指令定义的段名	
MODULE	由 NAME 伪指令定义的模块名（如果模块名没有定义，而是从输入的源文件名的原有名称中创建）	
MACRO	由 MACRO 伪指令定义宏名	

< 示例 >

```
TEN      EQU     10H          ; 名称 "TEN" 具有属性 "NUMBER" 和值 "10H".
         ORG     80H
START :  MOV     A , #10H     ; 标签 "START" 具有属性 "ADDRESS" 和值 "80H".
BIT1     EQU     0FE20H.0     ; 名称 "BIT1" 具有属性 "BIT" 和值 "0FE20H.0" .
```

2.2.4　助记忆字段

在助记忆字段中，描述了助记忆指令、伪指令或宏引用 .

由于指令或伪指令需要一个或多个操作数，助记忆字段必须与操作字段用一个或多个空格或制表符隔开 .

但是，若伪指令的第一个操作数以 "#"、 "$"、 "!" 或 "[" 开头，那么即使助记忆字段和第一个操作字段之间没有符号做间隔，汇编也将会正确执行 .

< 正确的描述示例 >

```
MOV     A , #0H
CALL    !CONVAH
RET
```

< 错误的描述示例 >

```
MOVA    #0H         ; 在助记忆和操作字段之间无空格 .
CALL    !CONVAH     ; 在助记忆字段内有空格 .
ZZZ                 ; 78K0S 系列没有如 "ZZZ" 之类的指令 .
```

2.2.5　操作字段

在操作字段中，描述了执行指令、伪指令或宏引用所需的数据 （操作数）.

根据指令或伪指令的不同，在操作字段中不需要操作数，或者在操作字段中必须描述两个或更多的操作数 .

当描述两个或更多的操作数时，用逗号 （, ）把每个操作数定界 .

在操作字段中可以描述下列数据类型：

- 常量 (数字常量和串常量)
- 字符串
- 寄存器名
- 特殊字符 ($, #, !, and [ ])
- 段定义伪指令的重定位属性
- 符号
- 表达式
- 位术语

所需操作数的大小和属性依据指令或伪指令的不同可能有所不同 . 有关操作数的大小和属性，参考 “2.6 操作数的特征”.

有关在指令集中操作数的表示格式和描述方法，参见目前正对其进行软件开发的微控制器用户手册 .

在操作字段中可以描述的各种数据类型详列如下 .

(1) 常量

常量是一个固定值或数据项，也称作立即数 .

常量分为数字常量和字符串常量 .

(a) 数字常量

二进制、八进制、十进制或十六进制数都可描述为数字常量 .

各种数字常量类型的表示方法如下表 2-7 所示 .

数字常量将作为无符号的 16 位数据来处理

值的范围：0 ≤ n ≤ 65535 （0FFFFH）

当描述一个负值时，使用减号操作符

表 2-7　数字常量的表示法

常量	表示法	示例
二进制常量	在数值后加 ″B″ 或 ″Y″ 作为后缀 .	1101B 1101Y
八进制常量	在数值后加 ″O″ 或 ″Q″ 作为后缀 .	74O 74Q
十进制常量	数值描述为它本身，或在数值后加字符 "D" 或 "T" 作为后缀 .	128 128D 128T
十六进制常量	- 在数字值后加字符 ″H″ 作为后缀 . - 如果首字符以 "A", "B", "C", "D", "E", 或 "F" 开始 , 该常量前必须加前缀 "0".	8CH 0A6H

(b) 字符串常量

字符串常量由包含在一对单引号 （'）内的一串字符表示，字符已在 “2.2.2 字符集”中说明 .

作为一个汇编处理的结果，字符串常量被转换成 7 位 ASCII 码，优先级位 （MSB）设置为 “0”.

字符串常量的长度为 0 到 2 个字符 .

对于使用单引号标志本身作为字符串常量的情况，单引号标志必须连续输入两次 .

< 字符串常量描述示例 >

```
' ab '          ; 表示 "6162H"
' A '           ; 表示 "0041H"
' A '''         ; 表示 "4127H"
' '             ; 表示 "0020H" ( 一个空格 )
```

(2) 字符串

字符串常量由包含在一对单引号 （'）内的一串字符表示，字符已在 “2.2.2 字符集”中说明 . 字符串主要用作 DB 伪指令和 TITLE 或 SUBTITLE 控制指令中的操作数 .

< 字符串的应用实例 >>

```
        CSEG
MAS1 :  DB      ' YES ' ; 初始化字符串 "YES".
MAS2 :  DB      ' NO '  ; 初始化字符串 "NO".
```

(3) 寄存器名

在操作字段中可对以下寄存器进行描述：

– 通用寄存器

- 通用寄存器对

- 特殊功能寄存器

通用寄存器和通用寄存器对可以用其绝对名称 （R0 至 R7、 RP0 至 RP3）以及功能名称 （X、 A、 B、 C、 D、 E、 H、 L、 AX、 BC、 DE、 HL）进行描述 .

可在操作字段中描述的寄存器名根据指令类型的不同可能会有所相同 . 有关描述各个寄存器名称描述方法的详细内容，参见正对其进行开发的各设备用户手册 .

(4) 特殊字符

可以在操作字段中描述的特殊字符如下表 2-8 所示 .

表 2-8　可在操作字段中描述的特殊字符

特殊字符	功能
$	- 表示具有该操作符的指令的定位地址 （或者在多字节指令的地址中，表示该地址的第一个字节）. - 表示一个跳转指令的相对寻址模式 .
!	- 表示一个跳转指令的绝对寻址模式 . - 表示 addr16 的规定，即允许由 MOV 指令指定所有的存储空间 .
#	– 表示立即数 .
[]	– 表示间接寻址模式 .

< 特殊字符应用实例 >

```
地址          源程序
100             ADD     A , #10H
102     LOOP :  INC     A
103             BR      $$ - 1          ; (1)
105             BR      !$ + 100H       ; (2)
```

(1) 操作数中第二个 $ 表示地址 103H. 描述 “BR $ - 1”导致相同的运算 .

(2) 操作数中第二个 $ 表示地址 105H. 描述 “BR $ + 100H”导致相同的运算 .

(5) 段定义伪指令的重定位属性

在操作字段中可描述重定位属性 .

有关重定位的详细内容，参考 “3.2 段定义伪指令”.

(6) 符号

如果在操作字段中描述了符号，则分配给符号的地址 （或值）变成了操作数的值 .

< 符号应用实例 >

```
VALUE   EQU     100H
        MOV     A , #VALUE      ; 该描述语句可以写作 ″MOV A , #100H″.
```

(7) 表达式

表达式是由操作符连起来的常量、 $ （指示定位地址）、名称或标签组成的 .

在数字值可被表示成指令操作符的地方可以描述表达式 .

有关表达式和操作符，参见 “2.3 表达式和操作符”.

< 表达式示例 >

```
TEN     EQU     10H
        MOV     A , #TEN - 5H
```

该例中， "TEN - 5H" 是一个表达式 .

该表达式中，名称和数字常量通过一个 “-” 号 （减号）连接 . 表达式的值为 BH.

因此，该描述可被重写为 “MOV A , #0BH”.

(8) 位术语

位术语可通过位位置指定符获取 . 有关位术语的详细内容，参见 2.5 位位置指定符 .

< 位术语示例 >

```
CLR1    A.5
SET1    1 + 0FE30H.3            ; 操作符的值是 0FE31H.3.
CLR1    0FE40H.4 + 2            ; 操作符的值是 0FE40H.6.
```

2.2.6　注释字段

在注释字段中，注释或备注可能在输入分号 （；）以后进行描述．注释字段是从分号到该行的换行代码或 EOF. 通过在注释字段中描述注释语句，可以创建一个易于理解的源程序 ．在注释字段中的注释语句不受汇编器操作的控制 （如，转换成机器语言），但是将会无改变地输出至汇编列表．

可在注释字段中描述的字符在 “2.2.2 字符集”中显示．

< 注释示例 >

```
        NAME    SAMPM
; ***************************************
;       HEX -> ASCII 转换程序                    仅组成注释字段的行
;       主程序
; ***************************************

        PUBLIC  MAIN , START
        EXTRN   CONVAH
        EXTRN   @STBEG

DATA    DSEG    saddr
HDTSA:  DS      1
STASC:  DS      2

CODE    CSEG    AT 0H
MAIN :  DW      START

        CSEG
START :
                                ; 芯片初始化          仅组成注释字段的行
        MOVW    AX , #_@STBEG
        MOVW    SP , AX

        MOV     HDTSA , #1AH
        MOVW    HL , #HDTSA     ; 在 HL 寄存器中设置 hex 2 代码数据
                                                      在注释字段中描述注释
        CALL    !CONVAH         ; 转换 ASCII <- HEX
                                ; 输出 BC- 寄存器 <- ASCII 码

        MOVW    DE , #STASC     ; 设置 DE <- 存储 ASCII 码表
        MOV     A , B
        MOV     [ DE ] , A
        INCW    DE
        MOV     A , C
        MOV     [ DE ] , A
        BR      $$
```

2.3 表达式和运算符

表达式是指符号、常量、定位地址（通过 $ 指示）或位术语，与以上任意一个组合的运算符，或者运算符的组合体 .

表达式中除运算符外的元素称为项，并按它们出现在表达式中的顺序从左到右依次称为第一项、第二项等 .

在表 2-9 中显示的类型中运算符可用，在计算中它们的优先级顺序已经确定 , 如表 2-10 所示 .

圆括号 “（）”用来改变执行计算中的顺序 .

< 示例 >

```
MOV     A , #5 * ( SYM + 1 )    ; ( 1 )
```

在上面中的 （1）中，“5*(SYM+1) ”是一个表达式 . “5”是表达式的第一项，“SYM”和 “1”分别是第二项和第三项 . "*"、 "+"、和 "( )" 是运算符 .

表 2-9 运算符类型

运算符类型	运算符
算术运算符	+, -, *, /, MOD, + 号 , - 号
逻辑运算符	NOT, AND, OR, XOR
关系运算符	EQ (或者 =), NE (或者 < >), GT (或者 >), GE (或者 >=), LT (或者 <), LE (或者 <=)
移位运算符	SHR, SHL
字节分割运算符	HIGH, LOW
特殊运算符	DATAPOS, BITPOS, MASK
其它运算符	( )

上述运算符也可分为一元运算符，特殊一元运算符，二元运算符，N- 元运算符和其它运算符 .

一元运算符	+ 号 , - 号 , NOT, HIGH, LOW
特殊一元运算符	DATAPOS, BITPOS
二元运算符	+, -, *, /, MOD, AND, OR, XOR, EQ (或者 =), NE (或者 < >), GT (或者 >), GE (或者 >=), LT (或者 <), LE (或者 <=), SHR, SHL
N- 元运算符	MASK
其它运算符	( )

表 2-10　运算符的优先级顺序

优先级	优先级等级	运算符
较高	1	+ 号 , - 号 , NOT, HIGH, LOW, DATAPOS, BITPOS, MASK
	2	*, /, MOD, SHR, SHL
	3	+, -
	4	AND
	5	OR, XOR
较低	6	EQ (或者 =), NE (或者 < >), GT (或者 >), GE (或者 >=), LT (或者 <), LE (或者 <=)

依照以下原则执行表达式中的运算：

(1) 根据赋予各个运算符的优先级顺序执行运算 .　如果在一个表达式中存在两个或多个具有同一优先级的运算符，则执行最左边运算符指定的运算 . 在一元运算符的情况下，则按从右向左的顺序执行运算 .

(2) 先执行圆括号内的表达式，再执行圆括号外的表达式 .

(3) 允许两个或多个一元运算符之间的运算 .

示例 :　　1 = --1 == 1

-1 = -+1 = -1

(4) 表达式在 16 位以内计算，无符号 . 在运算中，如果由于表达式超出 16 位而发生上溢出，则忽略上溢出的值 .

(5) 如果常量超出 16 位 （0FFFFH），则会出错，且计算的结果指将记为 0.

(3) 在除法中，舍去结果中的十进制小数部分 . 若除数为 0，则会发生错误，且结果为 0.

2.3.1　运算符

这部分介绍了各个运算符的功能 .

算术运算符

(1) +

[功能]

- 返回表达式中第一项与第二项的和值 .

[应用示例]

```
        ORG     100H
START : BR      !$ + 6          ; ( a )
```

[说明]

- BR 指令使得地址跳转到 “当前定位地址 +6”，即地址 “100H + 6H = 106H”.

  因此，上面例子中的 (a) 也可以描述为 : START : BR !106H

(2) -

[功能]

- 返回从第一项的值中减去第二项值的结果 .

[应用示例]

```
        ORG     100H
BACK :  BR      BACK - 6H       ; ( b )
```

[说明]

- BR 指令使得地址跳转到 “分配给 BACK 的地址 -6”，即地址 “100H -6H = 0FAH”.

  因此，上面例子中的 (b) 也可以描述为 : BACK : BR !0FAH

(3) *

[功能]

- 返回表达式中第一项与第二项的值的乘积的结果 （乘积）.

[应用示例]

```
TEN     EQU     10H
        MOV     A , #TEN * 3    ; ( c )
```

[说明]

- 用 EQU 伪指令，将数值 “10H" 的名称定义为 “TEN”.

  "#" 表示立即数 . 表达式 "TEN * 3" 等同于 "10H * 3"，返回值为 "30H".

  因此，上面例子中的 (c) 也可以描述为 : MOV A , #30H

(4) /

[功能]

- 将表达式中第一项的值除以第二项并且返回结果的整数部分．舍去结果中十进制小数部分．如果除法运算中除数（即第二项）为 0，则会产生错误．

[应用示例]

```
MOV     A , #256 / 50           ; ( d )
```

[说明]

- 除法式 "256 / 50" 的结果是 5，余数是 6.

  运算符返回除式结果的整数部分，即 "5".

  因此，上面例子中的 (d) 也可以描述为 : MOV A , #5

(5) MOD

[功能]

- 得到表达式的第一项值除以第二项值结果的余数．

  如果出书（第二项）为 0，则会出错．

  在 MOD 运算符的前后都需要有空格．

[应用示例]

```
MOV     A , #256 MOD 50         ; ( e )
```

[说明]

- 除式 "256 / 50" 的结果是 5，余数是 6.

  MOD 运算符返回余数 6.

  因此，上述表达式中的（e）也可描述为 : MOV A , #6.

(6) + sign

[功能]

- 原封不动的返回表达式的项的值．

[应用示例]

```
FIVE    EQU     +5
```

[说明]

- 原封不动的返回项的值“5”.

  值 "5" 由 EQU 伪指令定义为名称 "FIVE" .

(7) - sign

[功能]

- 返回表达式的项以 2 为基数的补码的值 .

[应用示例]

```
NO      EQU     -1
```

[说明]

- -1 为 1 的以 2 为基数的补码 .

  二进制数 0000 0000 0000 0001 以 2 为基数的补码为：

  1111 1111 1111 1111

  因此，用 EQU 伪指令将数值 “0FFFFH” 定义成名称 “NO”.

逻辑运算符

(1) NOT

[功能]

- 按位对表达式项的值取反并返回结果值 .

  NOT 运算符和项之间需要一个空格 .

[应用示例]

```
MOVW    AX , #NOT 3H            ; ( a )
```

[说明]

- 对 “3H”进行逻辑取反运算如下所示：

NOT)	0000	0000	0000	0011
	1111	1111	1111	1100

返回 0FFFCH.

因此，（a）也可描述为：MOVW AX , #0FFFCH

(2) AND

[功能]

- 将表达式第一项的值和第二项的值按位执行 AND （逻辑乘）运算，并返回结果值

  AND 运算符的前后均需要一个空格 .

[应用示例]

```
MOV     A , #6FAH AND 0FH       ; ( b )
```

[说明]

- 在值 "6FAH" 和 "0FH" 之间进行 AND 运算，如下：

	0000	0110	1111	1010
AND)	0000	0000	0000	1111
	0000	0000	0000	1010

返回结果 "0AH". 因此，上述表达式中的 （b）也可描述为：MOV A , #0AH

(3) OR

[功能]

- 将表达式第一项的值和第二项的值按位进行 OR （逻辑和）运算，并且返回结果 .

  OR 运算符的前后均需要一个空格 .

[应用示例]

```
MOV     A , #0AH OR 1101B       ; ( c )
```

[说明]

- 在 "0AH" 和 "1101B" 之间进行 OR 运算，如下：

	0000	0000	0000	1010
OR)	0000	0000	0000	1101
	0000	0000	0000	1111

返回结果 "0FH".

因此，上述表达式中的 （c）也可描述为：MOV A , #0FH

(4) XOR

[功能]

- 将表达式第一项的值和第二项的值按位进行互斥逻辑和运算，并返回结果值 . XOR 运算符的前后均需要一个空格 .

[应用示例]

```
MOV     A , #9AH XOR 9DH        ; ( d )
```

[说明]

- 在 "9AH " 和 "9DH " 之间进行 XOR 运算，如下：

	0000	0000	1001	1010
XOR)	0000	0000	1001	1101
	0000	0000	0000	0111

返回结果 "7H".

因此，以上表达式中的 （d）也可描述为：MOV A , #7H

关系运算符

(1) EQ (or =)

[功能]

- 如果表达式中第一项的值等于第二项的值，则返回 0FFH （真）；如果二者不相等，则返回 00H （假）.

  EQ 运算符的前后均需要一个空格 .

[应用示例]

```
A1      EQU     12C4H
A2      EQU     12C0H

        MOV     A , #A1 EQ ( A2 + 4H )  ; ( a )
        MOV     X , #A1 EQ A2           ; ( b )
```

[说明]

- 在上面的 （a）中，表达式 "A1 EQ (A2 + 4H)" 变成 "12C4H EQ (12C0H + 4H)".

  因为第一项的值等于第二项的值，所以运算符返回 0FFH .

- 在上面的 （b）中，表达式 " A1 EQ A2" 变为 "12C4H EQ 12C0H ".

  因为第一项的值不等于第二项的值，所以运算符返回 00H.

(2) NE (or < >)

[功能]

- 如果表达式中第一项的值不等于第二项的值，则返回 0FFH （真）；如果二者相等，则返回 00H （假）.

  NE 运算符的前后均需要一个空格 .

[应用示例]

```
A1      EQU     5678H
A2      EQU     5670H

        MOV     A , #A1 NE A2           ; ( c )
        MOV     A , #A1 NE ( A2 + 8H )  ; ( d )
```

[说明]

- 在上面的 （c）中，表达式 "A1 NE A2" 变为 "5678H NE 5670H".

  因为第一项的值不等于第二项的值，所以运算符返回 0FFH .

- 在上面的 （d）中，表达式 "A1 NE (A2 + 8H)" 变为 "5678H NE (5670H + 8H)".

  因为第一项的值等于第二项的值，所以运算符返回 00H .

(3) GT (or >)

[功能]

- 如果表达式中第一项的值大于第二项的值，则返回 0FFH （真）；如果第一项的值等于或小于第二项的值，则返回 00H （假）.

  GT 运算符的前后均需要一个空格 .

[应用示例]

```
A1      EQU     1023H
A2      EQU     1013H

        MOV     A , #A1 GT A2           ; ( e )
        MOV     X , #A1 GT ( A2 + 10H ) ; ( f )
```

[说明]

- 在上面的 （e）中，表达式 " A1 GT A2" 变为 "1023H GT 1013H ".

  因为第一项的值大于第二项的值，所以运算符返回 0FFH.

- 在上面的 （f）中，表达式 " A1 GT (A2 + 10H)" 变为 "1023H GT (1013H + 10H)".

  因为第一项的值等于第二项的值，所以运算符返回 00H.

(4) GE (or >=)

[功能]

- 如果表达式中第一项的值大于或等于第二项的值，则返回 0FFH （真）；如果第一项的值小于第二项的值，则返回 00H （假）.
- GE 运算符的前后均需要一个空格 .

[应用示例]

```
A1      EQU     2037H
A2      EQU     2015H

        MOV     A , #A1 GE A2           ; ( g )
        MOV     X , #A1 GE ( A2 + 23H ) ; ( h )
```

[说明]

- 在上面的 （g）中，表达式 " A1 GE A2" 变为 "2037H GE 2015H ".

  因为第一项的值大于第二项的值，所以运算符返回 0FFH.

- 在上面的 （h）中，表达式 " A1 GE (A2 + 23H)" 变为 "2037H GE (2015H + 23H) ".

  因为第一项的值小于第二项的值，所以运算符返回 00H.

(5) LT (or <)

[功能]

- 如果表达式中第一项的值小于第二项的值，则返回 0FFH （真）；如果第一项的值等于或大于第二项的值，则返回 00H （假）.

  LT 运算符前后均需要一个空格 .

[应用示例]

```
A1      EQU     1000H
A2      EQU     1020H

        MOV     A , #A1 LT A2                   ; ( i )
        MOV     X , # ( A1 + 20H ) LT A2        ; ( j )
```

[说明]

- 在上面的 （i）中，表达式 " A1 LT A2" 变为 "1000H LT 1020H ".

  因为第一项的值小于第二项的值，所以运算符返回 0FFH .

- 在上面的 （j）中，表达式 " (A1 + 20H) LT A2" 变为 "(1000H + 20H) LT 1020H ".

  因为第一项的值等于第二项的值，所以运算符返回 00H .

(6) LE (or <=)

[功能]

- 如果表达式中第一项的值小于或等于第二项的值，则返回 0FFH （真）；如果第一项的值大于第二项的值，则返回 00H （假）.

  LE 运算符前后均需要一个空格 .

[应用示例]

```
A1      EQU     103AH
A2      EQU     1040H

        MOV     A , #A1 LE A2                   ; ( k )
        MOV     X , # ( A1 + 7H ) LE A2         ; ( l )
```

[说明]

- 在上面的 （k）中，表达式 " A1 LE A2" 变为 "103AH LE 1040H ".

  因为第一项的值小于第二项的值，所以运算符返回 0FFH .

- 在上面的 （l）中，表达式 "(A1 + 7H) LE A2" 变为 "(103AH + 7H) LE 1040H ".

  因为第一项的值大于第二项的值，所以运算符返回 00H .

移位运算符

(1) SHR

[功能]

- 将表达式第一项的值向右移动由第二项的值所指定的位数，并返回获取的值 . 与指定的移位位数等值的零移入高位 .

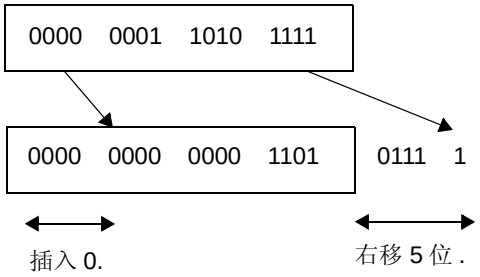
  SHR 运算符前后均需要一个空格 .

[应用示例]

```
MOV     A , #01AFH SHR 5        ; ( a )
```

[说明]

- 该运算符将值 "01AFH" 向右移动 5 位 .



返回结果 "000DH".

因此，上述例子中的 （a）也可描述为：MOV A , #0H

(2) SHL

[功能]

- 将表达式第一项的值向左移动由第二项所指定的位数，并返回获取的值 . 与指定的移位位数等值的零移入低位 .

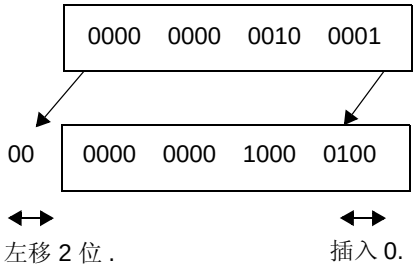
  SHL 运算符的前后均需要一个空格 .

[应用示例]

```
MOV     A , #21H SHL 2          ; ( b )
```

[说明]

- 该运算符将值 "21H" 向左移动 2 位 .



返回结果 "84H".

因此，上述例子中的 （b）也可描述为：MOV A , #84H

字节分割运算符

(1) HIGH

[功能]

- 返回项的高 8 位值 .

  HIGH 运算符和项之间需要一个空格 .

[应用示例]

```
MOV     A , #HIGH 1234H          ; ( a )
```

[说明]

- 通过执行 MOV 指令，该运算符返回表达式 "1234H" 的高 8 位值 "12H".

  因此 , 上面例子中的 (a) 也可描述为 : MOV A , #12H

(2) LOW

[功能]

- 返回项的低 8 位值 .

  LOW 运算符和项之间需要一个空格 .

[应用示例]

```
MOV     A , #LOW 1234H           ; ( b )
```

[说明]

- 通过执行 MOV 指令，该运算符返回表达式 "1234H" 的低 8 位值 "34H".

  因此 , 上面例子中的 (b) 也可描述为 : MOV A , #34H

特殊运算符

(1) DATAPOS

[功能]

- 返回位符号的地址部分 （字节地址）.

[应用示例]

```
SYM     EQU     0FE68H.6

        MOV     A , !DATAPOS SYM        ; ( a )
```

[说明]

- EQU 伪指令定义了名称 "SYM"，值为 0FE68H.6.

  "DATAPOS SYM" 表示 "DATAPOS 0FE68H.6"，并返回 "0FE68H".

  因此 , 上面例子中的 (a) 也可描述为 : MOV A , !0FE68H

(2) BITPOS

[功能]

- 返回位符号的位部分 （位位置）.

[应用示例]

```
SYM     EQU     0FE68H.6

        CLR1    [ HL ] .BITPOS SYM      ; ( b )
```

[说明]

- EQU 伪指令定义了名称 "SYM"，值为 0FE68H.6.

  " BITPOS.SYM " 表示 " BITPOS 0FE68H.6"，并返回 "6"

  CLR1 指令将 [HL].6 清为 0.

(3) MASK

[功能]

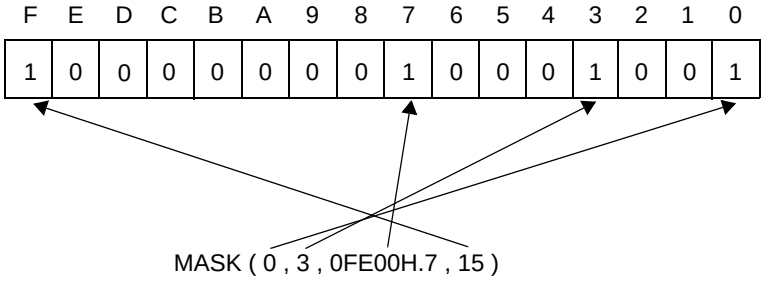
- 返回一个 16 位数值，其中指定位位置为 1 而其它位为 0.

[应用示例]

```
MOVW    AX , #MASK ( 0 , 3 , 0FE00H.7 , 15 )
```

[说明]

- MOVW 指令返回值 "8089H".



其它运算符

(1) ( )

[功能]

- 使圆括号内的运算优先于圆括号外的运算执行 .

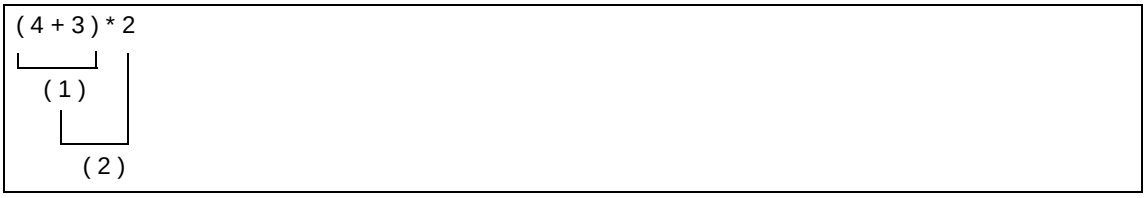
  该运算符用于改变其它运算符的优先级顺序 .

  如果圆括号嵌套多层，则最先计算最内部圆括号中的表达式 .

[应用示例]

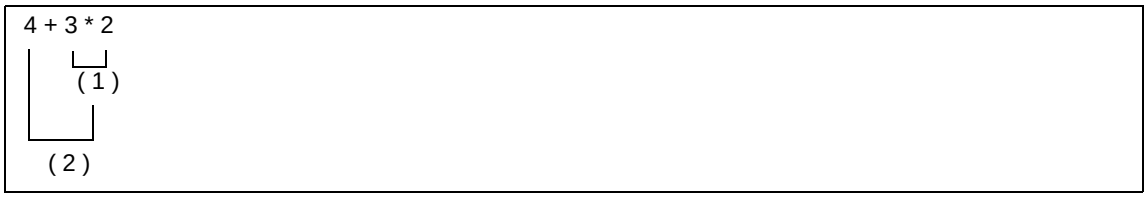
```
MOV     A , # ( 4 + 3 ) * 2
```

[说明]



按以上表达式 (1), (2) 的顺序进行计算，返回结果值 “14”.

若不使用圆括号，



按以上所示 (1), (2) 的顺序进行计算，返回结果值 “10”.

有关运算符的优先级顺序，参见表 2-10.

2.4　运算约束

通过用运算符连接项来执行表达式的运算．可被描述为项的元素包括常量、$、名称和标签．每一项都具有一个重定位属性和一个符号属性．

根据每个项固有的重定位属性和符号属性的类型，限制了可在该项上工作的运算符．因此，当描述一个表达式时，注意构成表达式各项的重定位属性和符号属性是非常重要的．

2.4.1　运算符和重新定位属性

正如前面提到的，构成表达式的每一项都具有重新定位属性和符号属性．

当根据重定位属性进行分类时，项可以分成三种类型：绝对项、可重新定位项和外部引用项．

表 2-11 显示了运算中重新定位属性的类型、各个属性的特点以及可应用于每个属性的项．

表 2-11　重新定位属性的类型

类型	特点	可用项
绝对项	值和常量在汇编时确定的项	- 常量 - 在绝对段内定义的标签 - $ 指示在绝对段内定义的定位地址 - 与常量、上述标签、上述 $ 或绝对值一起定义的名称
可重新定位项	值不在汇编时确定的项	- 在重定位段内定义的标签 - $ 指示在重新定位段内定义的定位地址 – 与可重新定位符号一起定义的名称
外部引用项[注]	外部引用其它模块的符号的项	- 用 EXTRN 伪指令定义的标签 - 用 EXTBIT 伪指令定义的名称

注　下面四个运算符可在外部引用项上工作："+", "-", "HIGH", and "LOW". 在一个表达式中，只能描述一个外部引用符号．在这种情况下，外部引用符号必须用 "+" 运算符进行连接．

表 2-12 显示了运算符的类型和每个可工作于运算符上的项的组合 .

表 2-12　按重定位属性划分的项和运算符的组合 （可重新定位项）

项的重新定位属性 运算符类型	X : ABS Y : ABS	X : ABS Y : REL	X : REL Y : ABS	X : REL Y : REL
X + Y	A	R	R	-
X - Y	A	-	R	A 注 1
X * Y	A	-	-	-
X / Y	A	-	-	-
X MOD Y	A	-	-	-
X SHL Y	A	-	-	-
X SHR Y	A	-	-	-
X EQ Y	A	-	-	A 注 1
X LT Y	A	-	-	A 注 1
X LE Y	A	-	-	A 注 1
X GT Y	A	-	-	A 注 1
X GE Y	A	-	-	A 注 1
X NE Y	A	-	-	A 注 1
X AND Y	A	-	-	-
X OR Y	A	-	-	-
X XOR Y	A	-	-	-
NOT X	A	A	-	-
+ X	A	A	R	R
- X	A	A	-	-
HIGH X	A	A	R 注 2	R 注 2
LOW X	A	A	R 注 2	R 注 2
MASK (X)	A	A	-	-
DATAPOS X.Y	A	-	-	-
BITPOS X.Y	A	-	-	-
MASK (X.Y)	A	-	-	-
DATAPOS X	A	A	R	R
BITPOS X	A	A	A	A

< 标签说明 >

ABS :　　绝对项

REL :　　重新定位项

A :　　运算结果变为绝对项 .

R :　　运算结果变为可重新定位项 .

- :　　运算不能执行 .

注　1.　　如果 X 和 Y 不是 HIGH, LOW, DATAPOS 上操作的可重新定位项，且在同一段内定义 X 和 Y 时，才能执行该运算 .

注　2.　　仅当 X 和 Y 不是 HIGH, LOW,DATAPOS 上操作的可重新定位项时，才能执行该运算 .

下面五个运算符可用于外部引用项："+", "-", "HIGH" 和 "LOW" （但是，需注意，在一个表达式中仅能描述一个外部引用项）.

根据重新定位属性，表 2-13 给出了运算符的类型和可工作在每个运算符上的外部引用项的组合 .

表 2-13　按重新定位属性划分的项和运算符的组合 （外部引用项）

项的重定位属性 运算符类型	X : ABS Y : EXT	X : EXT Y : ABS	X : REL Y : EXT	X : EXT Y : REL	X : EXT Y : EXT
X + Y	E	E	-	-	-
X - Y	-	E	-	-	-
+ X	A	E	R	E	E
HIGH X	A	E 注 1	R 注 2	E 注 1	E 注 1
LOW X	A	E 注 1	R 注 2	E 注 1	E 注 1
MASK (X)	A	-	-	-	-
DATAPOS X.Y	-	-	-	-	-
BITPOS X.Y	-	-	-	-	-
MASK (X.Y)	-	-	-	-	-
DATAPOS X	A	E	R	E	E
BITPOS X	A	E	A	E	E

< 标签说明 >

ABS :　　绝对项

EXT :　　外部引用项

REL :　　可重新定位项

A :　　运算结果变为绝对项 .

E :　　运算结果变为外部引用项 .

R :　　运算结果变为可重新定位项 .

- :　　运算不能执行 .

注　1.　仅当 X 和 Y 不是 HIGH, LOW, DATAPOS, BITPOS 上操作的外部引用项时，才能执行该运算 . .

注　2.　仅当 X 和 Y 不是 HIGH, LOW, DATAPOS 上操作的可重新定位项时，才能执行该运算 . .

2.4.2　运算符和符号属性

正如前面提到的，构成表达式的每一项除了具有重定位属性外，还具有符号属性 . 当根据符号属性进行分类时，项可以分成两类：NUMBER 项和 ADDRESS 项 .

表 2-14 显示了运算中符号属性的类型和可用于每个属性的项 .

表 2-14　运算中符号属性的类型

符号属性的类型	可用项
NUMBER 项	- 具有 NUMBER 属性的符号 - 常量
ADDRESS 项	- 具有 ADDRESS 属性的符号 - $ 表示位置计数器

表 2-15 显示了运算符的类型和根据符号属性类型进行分类时每个可用于运算符上的项的组合 .

表 2-15　按符号属性划分的项和运算符的组合

项的符号属性 运算符的类型	X : ADDRESS Y : ADDRESS	X : ADDRESS Y : NUMBER	X : NUMBER Y : ADDRESS	X : NUMBER Y : NUMBER
X + Y	-	A	A	N
X - Y	N	A	-	N
X * Y	-	-	-	N
X / Y	-	-	-	N
X MOD Y	-	-	-	N
X SHL Y	-	-	-	N
X SHR Y	-	-	-	N
X EQ Y	N	-	-	N
X LT Y	N	-	-	N
X LE Y	N	-	-	N
X GT Y	N	-	-	N
X GE Y	N	-	-	N
X NE Y	N	-	-	N
X AND Y	-	-	-	N
X OR Y	-	-	-	N
X XOR Y	-	-	-	N
NOT X	-	-	N	N
+ X	A	A	N	N
- X	-	-	N	N
HIGH X	A	A	N	N
LOW X	A	A	N	N
DATAPOS X	A	A	N	N
MASK X	N	N	N	N

< 标签说明 >

ADDRESS :　ADDRESS 项

NUMBER :　NUMBER 项

A :　运算结果变为 ADDRESS 项 .

N :　运算结果变为 NUMBER 项 .

- :　运算不能执行

2.4.3　如何检查对运算的约束

这里用一个示例说明按照每个项的重定位属性和符号属性进行的运算 .

< 示例 >

```
BR          $TABLE  +  5H
```

这里，假定 "TABLE" 是在重定位代码段中定义的标签 .

(a) 运算符和重定位属性

因为 "TABLE + 5H" 是 " 可重新定位项 + 绝对项 "，所以该操作适用于表 2-12.

运算符的类型 :　　X + Y

项的重定位属性 :　　X : REL, Y : ABS

从表中，我们可以看出结果为 R （也就是说，可重新定位项）.

(b) 运算符和符号属性

因为 "TABLE + 5H" 是 " ADDRESS 项 + NUMBER 项 "，所以该操作适用于表 2-15.

运算符的类型 :　　X + Y

项的重定位属性 :　　X : ADDRESS, Y : NUMBER

从表中，我们可以看出结果为 A （也就是说， ADDRESS 项）.

2.5　位位置说明符

通过使用位位置说明符 (.) 访问位 .

位位置说明符

(1) 句点 (.)

[描述格式]

```
X [ Δ ] . [ Δ ] Y
└──────────┘
   位术语
```

表 2-16　X(第一项) 和 Y(第二项) 的组合

X (第一项)		Y (第二项)
通用寄存器	A	表达式 (0 至 7)
控制寄存器	PSW	表达式 (0 至 7)
特殊功能寄存器	sfr 注	表达式 (0 至 7)
存储器	[HL] 注	表达式 (0 至 7)

注　　关于说明描述的更多细节，请参见各设备的用户手册 .

[功能]

- 位位置说明符的第一项指定字节地址，第二项指定位的位置 . 由该位的位置说明符可以访问一个特殊位 .

[说明]

- 位项是指使用位位置说明符的表达式 .
- 位位置说明符不受运算符优先级顺序的影响 . 位位置说明符的左侧被当作第一项，而右侧被认作第二项 .
- 下列约束适用于第一项：
  (1) 具有 NUMBER 或 ADDRESS 属性的表达式、可由位访问的 SFR 名称或者寄存器名称（A）均可被描述 .
  (2) 当在第一项中描述绝对表达式时，则表达式的范围必须是在 0FE20H 至 0FF1FH 之间 .
  (3) 可以描述外部引用符号 .
- 下列约束适用于第二项：
  (1) 表达式的值必须在 0 至 7 范围内 . 如果该值超出了范围，则会发生错误 .
  (2) 仅可描述具有 NUMBER 属性的绝对表达式 .
  (7) 不可以描述外部引用符号 .

[运算符和重定位属性]

- 在表 2-17 中显示了依据重定位属性组合的第一项和第二项 .

表 2-17　依据重定位属性组合第一项和第二项

组合项 X :	ABS	ABS	REL	REL	ABS	EXT	REL	EXT	EXT
组合项 Y :	ABS	REL	ABS	REL	EXT	ABS	EXT	REL	EXT
X.Y	A	-	R	-	-	E	-	-	-

< 标签说明 >

ABS :　绝对项

EXT :　外部引用项

REL :　重定位项

A :　运算的结果变为绝对项 .

E :　运算的结果变为外部引用项 .

R :　运算的结果变为重定位项 .

- :　运算不能执行 .

[位符号值]

- 在 EQU 伪指令的操作数域通过使用位位置说明符来描述一个位项从而定义一个位符号时 , 位符号将具有的值如下表 2-18 所示 .

表 2-18　位符号的值

操作数类型	符号值
A.bit 注 2	1.bit
PSW.bit 注 2	1FEH.bit
sfr Note 1 .bit 注 2	FFxxH.bit 注 3
expression.bit 注 2	xxxxH.bit 注 4

注　1.　详细描述参见各设备的用户手册 .

注　2.　位 = 0 至 7

注　3.　FFxxH 表示 sfr 的地址 .

注　4.　xxxxH 表示一个表达式的值 .

[应用示例]

```
SET1    0FE20H.3
SET1    A.5
CLR1    P1.2
SET1    1 + 0FE30H.3    ; Equals 0FE31H.3
SET1    0FE40H.4 + 2    ; Equals 0FE40H.6
```

2.6　操作数的特征

需要一个或多个操作数的指令和伪指令类型之间各不相同，它取决于所需操作数的值的大小和范围以及操作数的符号属性。

例如，指令 "MOV r , #byte" 的功能是将 "byte" 指定的值传递至寄存器 "r". 这种情况下，因为 r 是一个 8 位寄存器，所以将被传输的数据 "byte" 的大小必须是 8 位或者更少 .

如果将一个指令描述为 "MOV R0，#100H"，则会发生回避那错误，因为指令的第二个操作数 "100H" 的大小超过 8 位寄存器 R0 的容量 .

因此，当描述一个操作数时，必须注意以下几点：

- 操作数的值的大小或地址是否适合指令的操作数 （数字数据、名称或标签）？
- 符号属性是否适合指令的操作数 （名称或标签）？

2.6.1　操作数的值的大小和地址范围

当作为指令的操作数进行描述时，对数字数据、名称或标签的值的大小和地址范围设置一定的条件 .

对于指令而言，操作数的值的大小和地址范围的条件由每个指令的操作数表示格式决定 . 对于伪指令而言，操作数的值的大小和地址范围的条件由指令的类型决定 .

表 2-19 和表 2-20 列出了这些条件，如下所示 .

表 2-19　指令的操作数的值范围

操作数表示格式	值的范围	
byte	8 位值 0H 至 FFH	
word	16 位值 0H 至 FFFFH	
saddr	FE20H 至 FF1FH	
saddrp	FE20H 至 FF1FH 的偶数值	
sfr	FF00H 至 FFCFH, FFE0H 至 FFFFH	
sfrp	FF00H 至 FFCFH, FFE0H 至 FFFFH 的偶数值	
addr16	MOV, MOVW	0H 至 FFFFH
	其它指令	0H 至 FA7FH
addr5	40H 至 7EH 的偶数值	
bit	3 位值 0 至 7	
n	2 位值 0 至 3	

表 2-20　伪指令的操作数的值范围

伪指令的类型	伪指令	值的范围
段定义伪指令	CSEG AT	0H 至 FFFFH
	DSEG AT	0H 至 FFFFH
	BSEG AT	FE20H 至 FEFFH
	ORG	0H 至 FFFFH
符号定义伪指令	EQU	16 位值 0H 至 FFFFH
	SET	16 位值 0H 至 FFFFH
内存初始化与区域保留伪指令	DB	8 位值 0H 至 FFH
	DW	16 位值 0H 至 FFFFH
	DS	16 位值 0H 至 FFFFH
自动分支指令选择伪指令	BR	0H 至 FFFFH

2.6.2　指令所需的操作数的大小

指令可分为机器指令和伪指令. 对于需要立即数和符号作为操作数的指令，所需操作数的大小根据每个指令的不同有所变化 .

因此，当描述的数据超过了指令所需操作数的大小时，就会发生错误 . 表达式的操作采用 16 位无符号数执行 . 当计算结果超出 0FFFFH（16 位）时，输出一个警告信息 .

但是，当在操作数中描述可重新定位或外部引用符号时，值不在汇编器内确定 . 相反，链接器确定值并检查值的范围 .

2.6.3　操作数的符号属性和重新定位属性

当名称、标签和 $（表示位置计数器）被描述为指令操作数时，它们不一定会被描述为操作数 . 这取决于作为表达式项的符号属性和重定位属性 （见 "2.4 运算约束 "）如果是名称和标签，则取决于引用的方向 .

名称和标签的引用方向可以是后向引用或者前向引用 .

- 后向引用：名称或标签引用为操作数，而该操作数在名称或标签上面 （前面）的行中定义 .
- 前向引用：名称或标签引用为操作数，而该操作数在名称或标签下面 （后面）的行中定义 .

< 示例 >

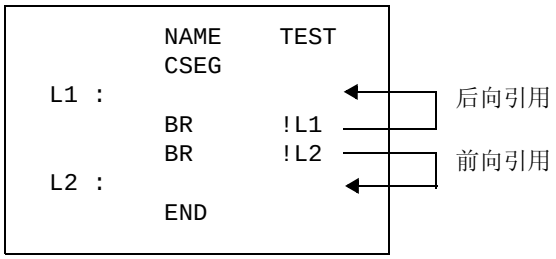


表 2-21 和表 2-22 给出了这些符号属性和重定位属性以及名称和标签的引用方向 .

表 2-21　作为操作数描述的符号特性

符号属性	NUMBER		ADDRESS				NUMBER ADDRESS		sfr 保留字[注 1]
重定位属性	绝对项		绝对项		可重新定位项		外部引用项		
引用模式 / 描述格式	后向	前向	后向	前向	后向	前向	后向	前向	
byte	OK	OK	OK	OK	OK	OK	OK	OK	NG
word	OK	OK	OK	OK	OK	OK	OK	OK	NG
saddr	OK	OK	OK	OK	OK	OK	OK	OK	OK [注 2, 3]
saddrp	OK	OK	OK	OK	OK	OK	OK	OK	OK [注 2, 4]
sfr	OK [注 5]	NG	NG	NG	NG	NG	NG	NG	OK [注 2, 6]
sfrp	NG	NG	NG	NG	NG	NG	NG	NG	OK [注 2, 7]
addr16 [注 8]	OK	OK	OK	OK	OK	OK	OK	OK	NG
addr5	OK	OK	OK	OK	OK	OK	OK	OK	NG
bit	OK	OK	NG	NG	NG	NG	NG	NG	NG
n	OK	OK	NG	NG	NG	NG	NG	NG	NG

< 标签示例 >

前向 :　　表示前向引用 .

后向 :　　表示后向引用 .

OK :　　表示可以这样描述 .

NG :　　表示出错 .

- :　　表示不可以这样描述 .

注　1.　定义的符号将 sfr 或 sfrp （saddr 和 sfr 没有重叠的 sfr 区域）指定为 EQU 伪指令的操作数时，该符号仅能后向引用 . 禁止前向引用 .

注　2.　如果 saddr 区域中的 sfr 保留字已经为一个指令描述，而该指令中 sfr/sfrp 组合由操作数组合中的 saddr/saddrp 变化而来，则代码作为 saddr/saddrp 输出 .

注　3.　Saddr 区域中的 sfr 保留字

注　4.　saddr 区域中的 sfrp 保留字

注　5.　仅绝对表达式

注　6.　仅允许 8 位访问的 sfr 区域

注　7.　仅允许 16 位访问的 sfr 区域

注　8.　当描述将禁止区域（FA80H 至 FADFH）用作 addr16 的值的地址时，不执行检查 .

表 2-22　作为伪指令的操作数描述的符号的特性

符号属性		NUMBER		ADDRESS, SADDR						BIT					
重定位属性		绝对项		绝对项		可重新定位项		外部引用项		绝对项		可重新定位项		外部引用项	
引用方向		后向	前向	后向	前向	后向	前向	后向	前向	后向	前向	后向	前向	后向	前向
伪指令															
ORG		OK 注 1	-	-	-	-	-	-	-	-	-	-	-	-	-
EQU 注 2		OK	-	OK	-	OK 注 3	-	-	-	OK	-	OK 注 3	-	-	-
SET		OK 注 1	-	-	-	-	-	-	-	-	-	-	-	-	-
DB	大小	OK 注 1	-	-	-	-	-	-	-	-	-	-	-	-	-
	初始值	OK	OK	OK	OK	OK	OK	OK	OK	-	-	-	-	-	-
DW	大小	OK 注 1	-	-	-	-	-	-	-	-	-	-	-	-	-
	初始值	OK	OK	OK	OK	OK	OK	OK	OK	-	-	-	-	-	-
DS		OK 注 4	-	-	-	-	-	-	-	-	-	-	-	-	-
BR		OK	-	-	-	-	-	-	-	-	-	-	-	-	-

< 说明 >

OK :　　可以描述

- :　　不可以描述

注　1.　仅能描述绝对表达式 .

注　2.　如果描述的表达式中包括以下任一模式时，则会发生错误 .

- ADDRESS 属性 - ADDRESS 属性
- ADDRESS 属性相关运算符 ADDRESS 属性
- HIGH 绝对 ADDRESS 属性
- LOW 绝对 ADDRESS 属性
- DATAPOS 绝对 ADDRESS 属性

- MASK 绝对 ADDRESS 属性
- 当运算结果可被以上 7 种模式的优化所影响时 .

注　3.　不允许具有重定位项的 HIGH/LOW/BANKNUM/DATAPOS/MASK 运算符创建项 .

注　4.　参考 ″3.4 (3) DS (定义存储空间)″.

第三章 伪指令

本章介绍伪指令。伪指令是控制 RA78K0S 执行系列操作所必须的所有类型的指令。

3.1 概述

指令作为汇编结果被翻译为目标代码 (机器语言)，但是原则上伪指令不能转换成目标代码。伪指令主要有以下功能：

- 方便源程序的描述
- 初始化和保留内存区
- 为汇编器和连接器执行特定的操作提供所需信息

表 3-1 列出了伪指令的类型。

表 3-1 伪指令列表

伪指令类型	伪指令
段定义伪指令	CSEG, DSEG, BSEG, ORG
符号定义伪指令	EQU, SET
内存初始化和区域保留伪指令	DB, DW, DS, DBIT
链接伪指令	EXTRN, EXTBIT, PUBLIC
目标模块名声明伪指令	NAME
自动分支指令选择伪指令	BR
宏伪指令	MACRO, LOCAL, REPT, IRP, EXITM, ENDM
汇编终止伪指令	END

后续章节详细各种伪指令。

描述各伪指令的格式中，"[ ]" 表示方括号中的参数可从定义中省略，"..." 表示同一格式的重复描述。

3.2 段定义伪指令

源模块必须以段为单位进行描述 .

段定义伪指令用于定义这些段 . 段可分为以下四种类型 :

(1) 代码段

(2) 数据段

(3) 位段

(4) 绝对段

段的类型决定了每个段在内存中的地址范围 .

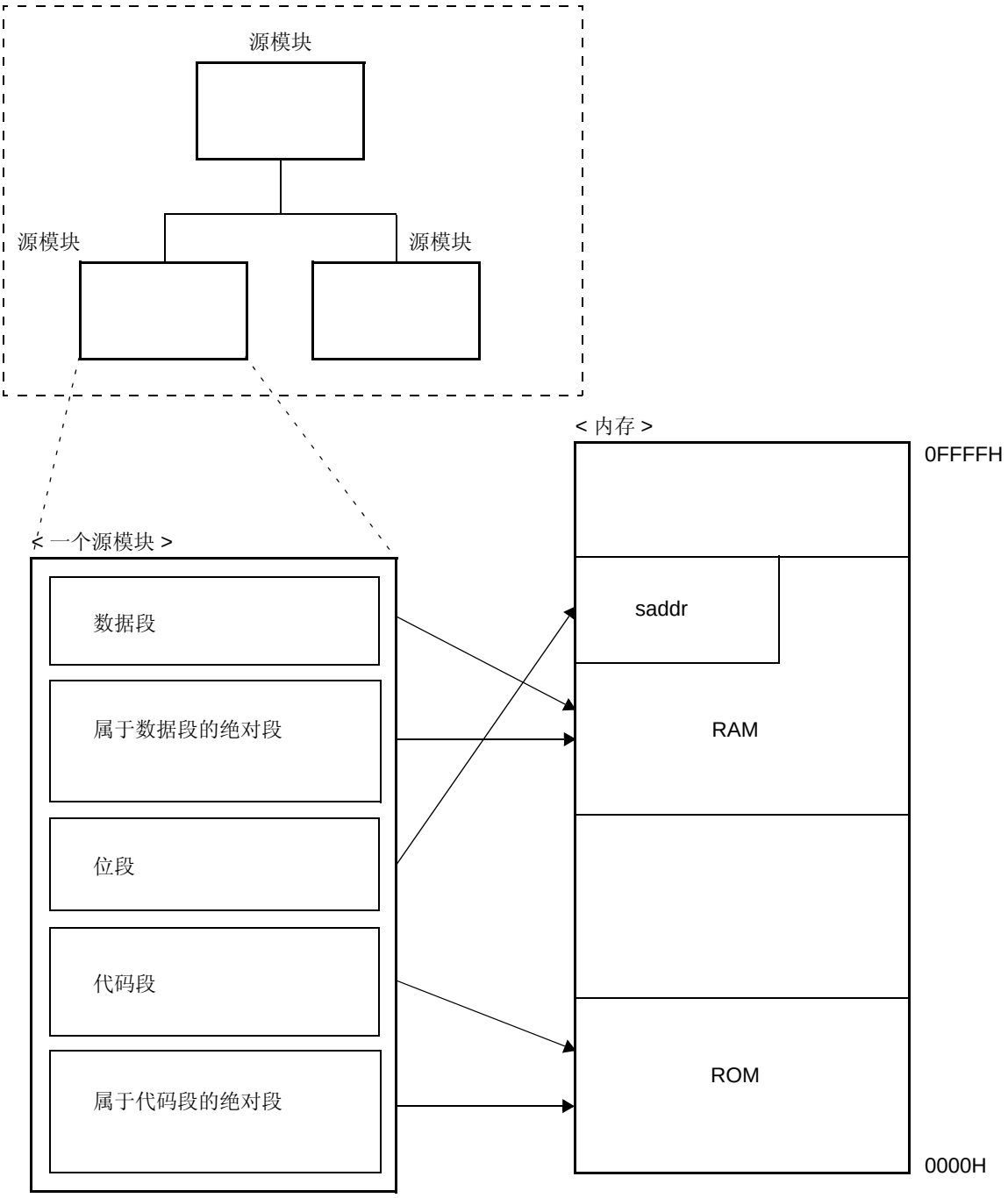
表 3-2 显示了定义各段的方法以及各段定位的内存地址 .

表 3-2 段定义方法和内存地址位置

段类型	定义方法	各段定位的内存地址
代码段	CSEG 伪指令	在内部或外部 ROM 地址
数据段	DSEG 伪指令	在内部或外部 RAM 地址
位段	BSEG 伪指令	在内部 RAM 的 saddr 区
绝对段	用 CSEG, DSEG, 或 BSEG 伪指令为重置属性指定位置地址 （AT 位置地址）	指定地址

如果用户希望确定段内存位置，则将段描述为绝对段. 对于堆栈区, 用户需要在数据段保留一个区域并设置栈指针. 图 3-1 是一个段定位举例 .

图 3-1　段的内存定位



以下段定义伪指令可用：

- CSEG (代码段)
- DSEG (数据段)
- BSEG (位段)
- ORG (原点)

CSEG

(1) CSEG (代码段)

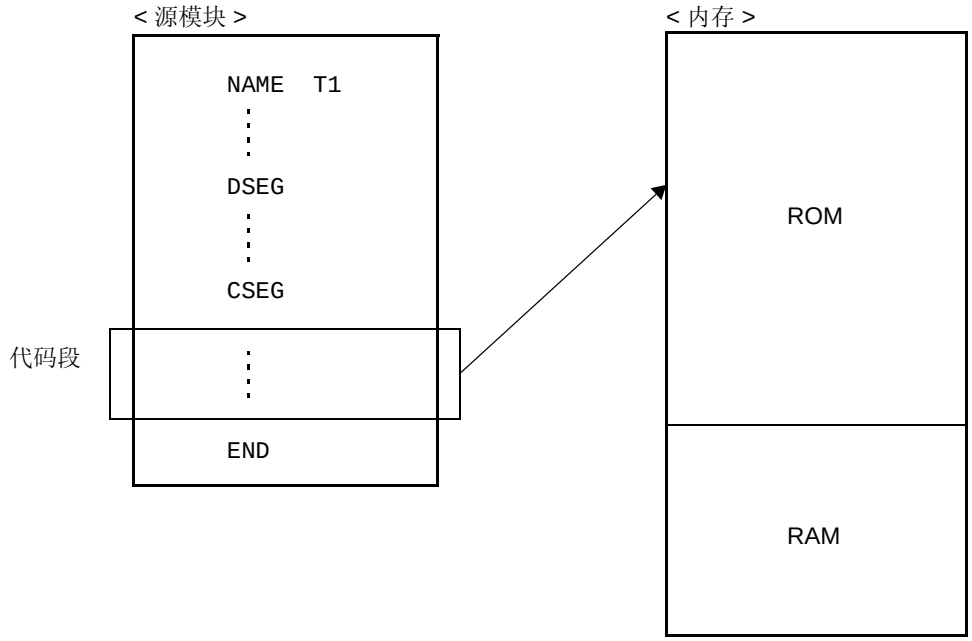
[描述格式 **]**

符号字段	助记符字段	操作字段	注释字段
[segment - name]	CSEG	[relocation - attribute]	[; 注释]

[功能]

- CSEG 伪指令将代码段的起始处显示在汇编器中 .
- CSEG 指令后的所有描述的指令都属于代码段 , 直到该指令遇到段定义伪指令 (CSEG, DSEG, BSEG或ORG) 或END 指令 , 最终这些指令被转换成机器语言后定位在 ROM 地址 .

图 3-2 代码段的重置



[用途]

- CSEG 伪指令用于在该指令定义的代码段中描述指令 ,DB,DW 伪指令等 .(但是 , 若要重置固定地址中的代码段 , 则在操作字段中必须将 ″AT 绝对表达式 ″ 描述为重置属性 .)
- 一个功能单元的描述 , 例如一个子程序应该描述为一个独立的代码段 . 如果该单元相对较大 , 或该子程序有很大的通用性 (例如 , 可用于开发其它程序), 该子程序也应定义成一个独立模块 .

[说明]

- 代码段的开始地址可以用 ORG 伪指令指定 . 它也可以通过描述重置属性 ″AT 绝对表达式 ″ 来指定 .
- 重置属性为代码段定义了单元地址的范围 . 重置属性显示在表 3-3.

表 3-3 CSEG 的重置属性

重置属性	描述格式	说明
CALLT0	CALLT0	通知汇编器定位该指定段 , 从而使段的起始地址在地址范围 0040H 至 007F 内变成 2 的倍数 . 为定义子程序的入口地址的代码段指定重置属性 , 而子程序用一个字节指令 "CALLT" 来调用 .
FIXED	FIXED	命令汇编器将指定段的开始地址定位在 0800H 到 0FFFH的地址范围.为定义子程序的代码段指定重置属性 , 而子程序用 2 个字节指令 "CALLF" 来调用 .
AT	AT 绝对表达式	命令汇编器将指定段定位在绝对地址 (0000H 至 FFFFH).
UNIT	UNIT	命令汇编器将指定段定位在任一地址 (0080H 至 FA7FH).
UNITP	UNITP	命令汇编器将指定段定义在任一地址 , 这样地址的起始处可能时偶数 (0080H 至 FA7EH).
IXRAM	IXRAM	命令汇编器将指定段定位于内部扩展 RAM 区 .

- 如果没有为代码段指定重置属性 , 则汇编器将假定已经指定 ″UNIT″.
- 如果指定表 3-3 以外的重置属性 , 则汇编器会输出错误信息并且假定已经指定 ″UNIT″. 如果各代码段的容量超出其重置属性指定区域的容量 , 则会发生错误 .
- 如果带重置属性 "AT" 的绝对表达式是非法的，则汇编器将输出错误信息并假定表达式的值为 "0" 继续进行处理 .

- 通过在 CSEG 伪指令的符号字段中描述一个段名 , 代码段可以被命名 . 如果没有为代码段指定段名 , 则汇编器将自动赋予代码段一个默认段名 . 代码段的默认段名如下所示 :

表 3-4 CSEG 的默认段名

重置属性	默认段名
CALLT0	?CSEGT0
FIXED	?CSEGFX
UNIT (或省略)	?CSEG
UNITP	?CSEGUP
XRAM	?CSEGIX
AT	段名不能被省略 .

- 当重置属性为 AT 时 , 如果省略段名 , 则会产生错误 .
- 如果两个或更多的代码段具有相同的重置属性 (AT 除外), 那么这些代码段可能具有相同的段名 . 在汇编器中这些同名的代码段被当做一个代码段来处理 .

  如果同名段的重置属性不同 , 则会产生错误 . 因此 , 就重置属性来说 , 同名段的个数为 1.
- 在两个或更多的模块中的同名代码段在连接时被组合成一个代码段 .
- 没有段名可引用为符号 .
- 可以由汇编器输出的段的总数最多为 255 个别名 , 其中包括那些用 ORG 伪指令所定义的段 . 同名段被计作 1 个段 .
- 可以当作段名的字符的最多为 8 个 .
- 可区分段名的大写和小写字符 .
- 在 80H 到 84H 地址范围指定选项字节 (各器件有所不同).

  如果为指定没有选项字节特征的芯片指定选项字节 , 则会导致错误 .

  当不为具有选项字节特性的芯片指定选项字节时 , 在各地址上定义一个 "?CSEGOB0 至 ?CSEGOB4" 默认段 , 并通过读取器件文件设置初始值 .

［应用举例 ］

```
        NAME    SAMP1
C1      CSEG            ; (1)

C2      CSEG    CALLT0  ; (2)

        CSEG    FIXED   ; (3)

C1      CSEG    CALLT0  ; (4)

        CSEG            ; (5)

        END
```

〈 说明 〉

(1) 汇编器将段名解释为 " C1", 将重置属性解释为 " UNIT ".

(2) 汇编器将段名解释为 " C2", 将重置属性解释为 " CALLT0".

(3) 汇编器将段名解释为 "?CSEGFX ", 将重置属性解释为 " FIXED ".

(4) 由于在 (1) 中段名 "C1" 被定义为重置属性 "UNIT", 因此产生错误 .

(5) 汇编器将段名解释为 "?CSEG", 将重置属性解释为 "UNIT".

DSEG

(2) DSEG (数据段)

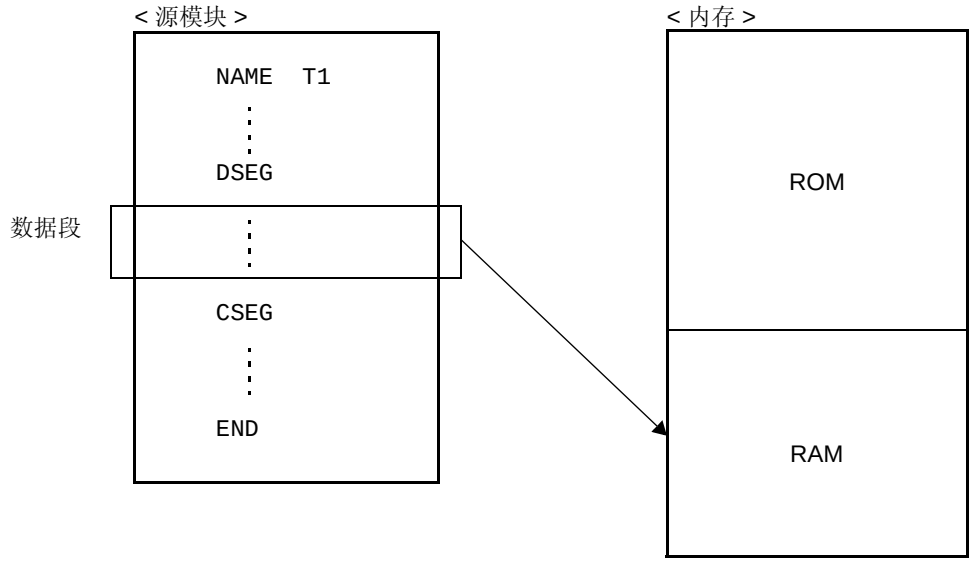
[描述格式 **]**

符号字段	助记符字段	操作字段	注释字段
[segment - name]	DSEG	[relocation - attribute]	[; 注释]

［功能 ］

- DSEG 伪指令将数据段的起始处显示在汇编器中 .
- 在 DSEG 伪指令后由 DS 伪指令定义的内存都属于数据段 , 直到该伪指令遇到段定义伪指令 (CSEG, DSEG, BSEG, 或 ORG) 或者 END 伪指令为止 , 并且最终将其保留在 RAM 地址中 .

图 3-3 数据段的重置



［用途 ］

- DS 伪指令主要在由 DSEG 伪指令定义的数据段中进行描述 . 数据段定位于 RAM 区 . 因此 , 在任何数据段中都不能描述指令 .
- 在数据段中 , 程序所使用的 RAM 工作区被 DS 伪指令保留 , 并且为各工作区附加上标签 . 当描述源程序时将使用这个标签 .

  保留为数据段的每个区域由连接器进行定位 , 这样它就不会与 RAM 中的任何其它工件区重叠 (堆栈区 , 通用寄存器区和由其他模块定义的工作区).

[说明]

- 数据段的开始地址可以用 ORG 伪指令指定 . 也可以在 DSEG 伪指令的操作字段中用绝对表达式描述重置属性 “AT”, 进而指定开始地址 .
- 重置属性为数据段定义了单元地址的范围 . 数据段中可用的重置属性如下所示 :

表 3-5 DSEG 的重置属性

重置属性 e	描述格式	说明
SADDR	SADDR	命令汇编器将指定的段在 saddr 区进行定位 (saddr 区 :0FE20H 至 0FEFFH).
SADDRP	SADDRP	命令汇编器将指定的段在 saddr 区的偶数地址进行定位 (saddr 区 :0FE20H 至 0FEFFH)
AT	AT 绝对表达式	命令汇编器在绝对地址中的指定段进行定位 .
UNIT	UNIT 或未指定	命令汇编器将指定段在地址中的任一位置 (在内存区称为 "RAM") 的进行定位 .
UNITP	UNITP	命令汇编器将指定段在地址中的偶数地址 (在内存区称为 "RAM") 的进行定位 .
IHRAM	IHRAM	命令汇编器将在高速 RAM 区的指定段进行定位
LRAM	LRAM	命令汇编器将在低速 RAM 区的指定段进行定位 .
DSPRAM	DSPRAM	命令汇编器将指定段在显示 RAM 区进行定位 .
IXRAM	IXRAM	命令汇编器将指定段定位于内部扩展 RAM 区 .

- 如果没有为数据段指定重置属性 , 则汇编器将假定已经指定 "UNIT".
- 如果指定列表 3-5 重置属性以外的重置属性 , 则汇编器会输出错误信息并且假定已经指定 "UNIT". 如果各数据段的容量超出其重置属性指定区域的容量 , 则会发生错误 .
- 如果用重置属性 “AT” 指定绝对表达式 , 则汇编器会输出错误信息并假定表达式的值为 “0” 继续进行处理 .
- 通过在 DSEG 伪指令的符号字段中描述一个段名 , 数据段可以被命名 .

  如果没有指定数据段的段名 , 则汇编器将自动赋予一个默认段名 . 数据段的默认段名如下所示 :

表 3-6 DSEG 的默认段名

重置属性	默认段名
SADDR	?DSEGS
SADDRP	?DSEGSP
UNIT (或未指定)	?DSEG
UNITP	?DSEGUP
IHRAM	?DSEGIH
LRAM	?DSEGL
DSPRAM	?DSEGDSP
IXRAM	?DSEGIX

表 3-6 DSEG 的默认段名

重置属性	默认段名
AT	段名不能被省略 .

- 如果两个或更多的数据段具有相同的重置属性 (AT 除外), 那么这些数据段可能具有相同的段名 . 这些段在汇编器中作为一个数据段处理 .
- 如果重置属性为 SADDRP, 则指定段被定位 , 使得紧随 DSEG 伪指令描述后的地址变成 2 的倍数 .
- 如果同名段的重置属性不同 , 则会产生错误 . 因此 , 就重置属性来说 , 同名段的个数为 1.
- 在两个或更多的模块中的同名数据段在连接时被组合成一个数据段 .
- 没有段名可引用为符号 .
- 可以由汇编器输出的段的总数最多为 255 个别名 , 其中包括那些用 ORG 伪指令所定义的段 . 同名段被计作 1 个段 .
- 可以当作段名的字符的最多为 8 个 .
- 可区分段名的大写和小写字符 .

［应用举例 ］

```
        NAME    SAMP1

        DSEG                    ; (1)
WORK1 : DS      1
WORK2 : DS      2
    CSEG
        MOV     A , !WORK1      ; (2)
        MOV     A , WORK1       ; (3)
        MOVW    DE , #WORK2     ; (4)
        MOVW    AX , WORK2      ; (5)
    END
```

〈 说明 〉

(1) 用 DSEG 伪指令定义数据段的开始 . 因为其重置属性被忽略 , 所以假定为 "UNIT". 默认段名为 "?DSEG".

(2) 此描述相当于 " MOV A, !addr16".

(3) 此描述相当于 "MOV A, saddr". 可重新定位标签 "WORK1" 不能描述为 " saddr ". 因此 , 这样描述的结果为产生错误 .

(4) 此描述相当于 "MOVW rp, #word".

(5) 此描述相当于 "MOVW AX, saddrp".

可重新定位标签 "WORK2" 不能描述为 " saddrp ". 因此 , 这样描述的结果为产生错误 .

BSEG

(3) BSEG (位段)

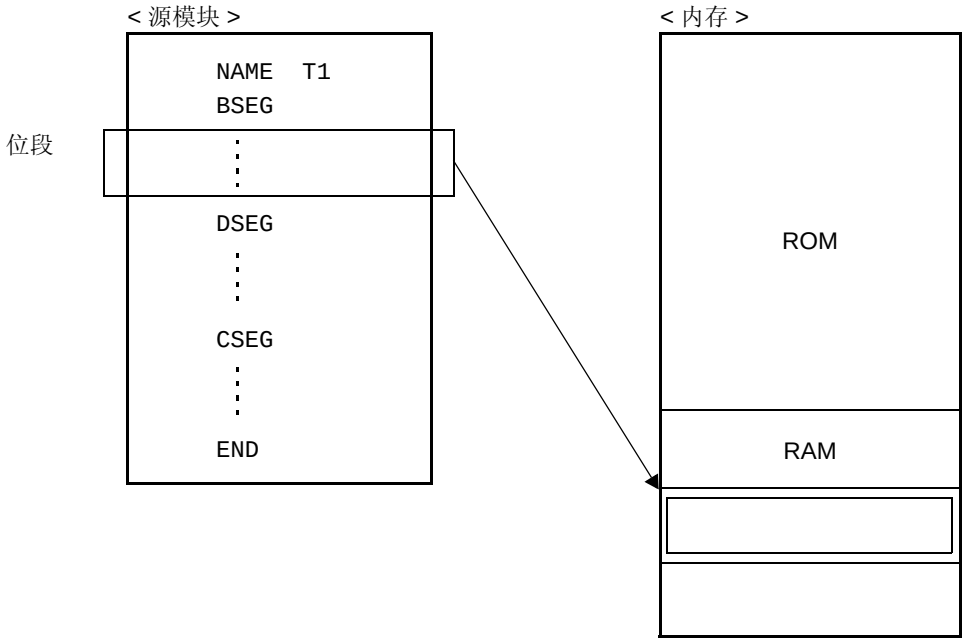
[描述格式 **]**

符号字段	助记符字段	操作字段	注释字段
[segment - name]	BSEG	[relocation - attribute]	[; 注释]

［功能 ］

- BSEG 伪指令将位段的起始处显示在汇编器中 .
- 位段是定义用于源模块 RAM 地址的段 .
- 在 BSEG 伪指令后用 DBIT 伪指令定义的内存区属于位段，直到该伪指令遇到段定义指令 (CSEG,DSEG 或 BSEG) 或 END 伪指令为止 .

图 3-4　位段的重置



［ 用途 ］

- 在 BSEG 伪指令定义的位段中描述 DBIT 伪指令 (参见 **[** 应用举例 **]**).
- 没有指令可在位段中描述 .

[说明]

- 可通过在重置属性字段中描述 "AT 绝对表达式 " 指定位段的开始地址 .
- 重置属性为位段定义了单元地址的范围 . 位段中可用的重置属性如下所示 :

表 3-7 BSEG 的重置属性

重置属性	描述格式	说明
AT	AT 绝对表达式	命令汇编器将指定段的起始地址定位于绝对地址的第 0 位 . 禁止以位为单位的指定 (FE20H 至 FEFFH).
UNIT	UNIT (或没有指定)	命令汇编器将指定段在任一地址中进行定位 (FE20H 至 FEFFH).

- 如果没有为位段指定重置属性 , 则汇编器将假定已经指定 "UNIT".
- 如果指定列表 BSEG 的重置属性以外的重置属性 , 则汇编器会输出错误信息并且假定已经指定 "UNIT". 如果各位段的容量超出其重置属性指定区域的容量 , 则会发生错误 .
- 汇编器和连接器中 , 位段的位置计数器都以 "0xxxx.b" 格式显示 .(字节地址为十六进制 4 位数 , 位位置为十六进制 1 位数 (0 到 7)).

(1) 使用绝对位段

表 3-8 定位计数器 (绝对)

字节地址	位位置							
	0	1	2	3	4	5	6	7
0FE20H	0FE20H.0	0FE20H.1	0FE20H.2	0FE20H.3	0FE20H.4	0FE20H.5	0FE20H.6	0FE20H.7
0FE21H	0FE21H.0	0FE21H.1	0FE21H.2	0FE21H.3	0FE21H.4	0FE21H.5	0FE21H.6	0FE21H.7

(2) 使用可重置位段

表 3-9 定位计数器 (可重新定位)

字节地址	位位置							
	0	1	2	3	4	5	6	7
0H	0H.0	0H.1	0H.2	0H.3	0H.4	0H.5	0H.6	0H.7
1H	1H.0	1H.1	1H.2	1H.3	1H.4	1H.5	1H.6	1H.7

备注 在重置位段中 , 字节地址从段的开始位置的字节单元中指定一个偏移量值 .

在目标转化器输出的符号表中 , 显示并输出从位定义区域开始的位偏移量 .

表 3-10 符号值与位偏移量

符号值	位偏移量
00FE20H.0	0000
00FE20H.1	0001

符号值	位偏移量
00FE20H.2	0002
:	:
00FE20H.7	0007
00FE21H.0	0008
00FE21H.1	0009
:	:
00FE80H.0	0300
:	:

- 如果用重置属性 “AT” 指定绝对表达式 , 则汇编器会输出错误信息并假定表达式的值为 “0” 继续进行处理 .
- 通过在 BSEG 伪指令的符号字段中描述一个段名 , 位段可以被命名 .

  如果没有指定位段的段名 , 则汇编器将自动赋予一个默认段名 . 下表列出了默认段名 :

表 3-11　BSEG 的默认段名

重置属性	默认段名
UNIT (或没有指定)	?BSEG
AT	段名不可被省略 .

- 如果重置属性为 "UNIT", 两个或更多的数据段可以具有相同的段名 (AT 除外). 这些段在汇编器中作为一个段处理 .

  因此 , 就重置属性来说 , 同名段的个数为 1.
- 在两个或更多的模块中的同名位段在连接时被组合成一个位段 .
- 没有段名可引用为符号 .
- 可在位段中描述的指令有 DBIT,EQU,SET,PUBLIC,EXTBIT,EXTRN,MACRO,REPT,IRP,ENDM 伪指令 , 宏定义和宏引用 . 描述其它的指令会产生错误 .
- 可以由汇编器输出的段的总数最多为 255 个别名段 , 其中包括那些用 ORG 伪指令所定义的段 . 同名段被计作 1 个段 .
- 可以当作段名的字符的最多为 8 个 .
- 可区分段名的大写和小写字符 .

[应用举例]

```
        NAME    SAMP1

FLAG    EQU     0FE20H

FLAG0   EQU     FLAG.0          ; (1)
FLAG1   EQU     FLAG.1          ; (2)

        BSEG                    ; (3)
FLAG2   DBIT

        CSEG
        SET1    FLAG0           ; (4)
        SET1    FLAG2           ; (5)

        END
```

〈 说明 〉

(1) 位地址 (0FE20H 的位 0) 根据字节地址边界的考虑因素来定义 .

(2) 位地址 (1FE20H 的位 0) 根据字节地址边界的考虑因素来定义 .

(3) 用 BSEG 伪指令定义的位段 .

因为它的重置属性被忽略 , 所以假定位重置属性为 "UNIT" 而段名为 "?BSEG". 在各位段中 , 用 DBIT 伪指令为每个位定义一个位工作区 . 位段应该在模块体的前面部分进行描述 . 不考虑字节地址边界 , 在位段中定义的位地址 FLAG2 被定位 .

(4) 此描述可用 "SET1 FLAG.0" 代替 . FLAG 表示一个字节地址 .

(5) 在该描述中 , 没有考虑字节地址边界 .

ORG

(4) ORG (原点)

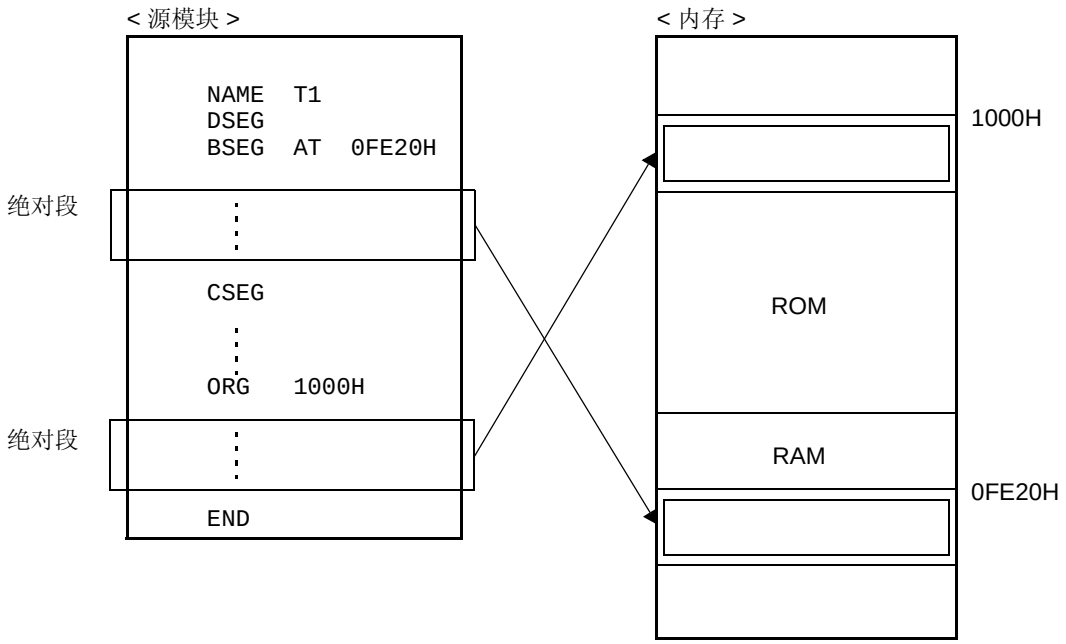
[描述格式 **]**

符号字段	助记符字段	操作字段	注释字段
[segment - name]	ORG	[relocation - attribute]	[; 注释]

［功能 ］

- ORG 伪指令设置由其位置计数器的操作数指定的表达式的值 .
- 在 ORG 伪指令之后描述的指令或保留内存区域属于绝对段 , 直到该指令遇到段定义伪指令 (CSEG, DSEG, BSEG, 或 ORG) 或 END 伪指令为止 , 它们从操作数指定的地址中进行定位 .

图 3-5 绝对段的位置



［ 用途 ］

- 指定 ORG 伪指令用来对特定地址中的代码段或数据段进行定位 .

［ 说明 ］

- 用 ORG 伪指令定义的绝对段属于在 ORG 伪指令之前用 CSEG 或 DSEG 定义的代码段或数据段 .
  在属于数据段的绝对段中 , 没有指令可以被描述 .
  属于位段的绝对段不能用 ORG 伪指令进行描述 .
- 用 ORG 伪指令定义的代码段或数据段被解释为具有重置属性 "AT" 的代码段或数据段 .
- 通过在 ORG 伪指令的符号字段中描述一个段名 , 绝对段可以被命名 . 可以当作段名的字符的最多为 8.
- 如果没有为绝对段定义段名 , 则汇编器将自动赋予默认段名 "?A00xxxx", 此处指定字段的 "xxxx" 表示 4 位数十六进制开始地址 (0000 到 FFFF).

- 如果在 ORG 伪指令之前既没有描述 CSEG 伪指令也没有描述 DSEG 伪指令，则由 ORG 伪指令定义的绝对段被解释为代码段中的绝对段 .
- 如果名称或标签被描述为 ORG 伪指令的操作数，则该名称或标签必须是在源模块中已经被定义的绝对项 .
- 没有段名可引用为符号 .
- 可以由汇编器输出的段的总数最多为 255 个别名段，其中包括那些用段定义伪指令所定义的段 . 同名段被计作 1 个段 .
- 可以当作段名的字符的最多为 8.
- 可区分段名的大写和小写字符 .

［应用举例 ］

```
        NAME    SAMP1

        DSEG
        ORG     0FE20H          ; (1)
SADR1 : DS      1
SADR2 : DS      1
SADR3 : DS      2

MAIN0   ORG     100H
        MOV     A , SADR1       ; (2)

        CSEG                    ; (3)
MAIN1   ORG     1000H           ; (4)
        MOV     A , SADR2
        MOVW    AX , SADR3
        END
```

〈 说明 〉

(1) 一个属于数据段的绝对段被定义 . 该绝对段将从地址 "FE20H" 开始的短指令直接寻址区进行定位 . 因为省略了指定段名，所以汇编器自动赋予段名 "?A00FE20".

(2) 由于在属于数据段的绝对段中没有指令被描述，所以将发生错误 .

(3) 该伪指令声明了代码段的开始位置 .

(4) 该绝对段被定位于从地址 "1000H" 开始的区域中 .

3.3　符号定义伪指令

符号定义伪指令为描述源模块使用的数字数据赋予名称．这些名称阐明了每个数据值的含义，使源模块的内容更易理解．

符号定义伪指令将源模块中使用名称的值告诉汇编器．

EQU (等于) 和 SET (设置) 两个伪指令可用于符号定义．

可以使用以下符号定义伪指令：

- EQU (等于)
- SET (设置)

EQU

(1) EQU (等于)

[描述格式 **]**

符号字段	助记符字段	操作字段	注释字段
name	EQU	expression	[; 注释]

[功能 **]**

- EQU 伪指令定义一个在操作字段中指定的表达式的名称 , 而该名称具有值和属性 （符号属性和重置属性） .

[用途]

- 用 EQU 伪指令将源模块中使用的数字数据定义为名称 , 并替代数字数据 , 描述指令的操作数中的名称 .
  建议在源模块中频繁使用的数字数据定义为一个名称 . 如果必须在源模块中改变数据值 , 所需要做的只是改变名称的操作数的值 .

[说明]

- 当在 EQU 伪指令的操作数中描述一个名称或标签时 , 需使用已经在源模块中定义的名称或标签 .
  没有外部引用项可被描述为该伪指令的操作数 .
- 包括由 HIGH/LOW/DATAPOS/BITPOS 运算符创建的项的表达式不可以被描述 , 这些运算符的操作数中有一个重置项 .
- 如果描述的表达式有以下模式的操作数 , 则会导致错误 :
  (1) 有 ADDRESS 属性的表达式 1- 有 ADDRESS 属性的表达式 2
  (2) 有 ADDRESS 属性的表达式 1 关系运算符 有 ADDRESS 属性的表达式 2
  (3) 以上表达式 (a) 或 (b) 满足以下条件 (1) 和 (2) 任一条件 :
    (a) 如果具有 ADDRESS 属性的表达式 1 中的标签 1 和具有 ADDRESS 属性的表达式 2 中的标签 2 属于同一字段 , 并且不能确定目标代码的字节数的 BR 伪指令在两个标签之间描述 .
    (b) 如果标签 1 和标签 2 在段中不相同 , 而且不能确定目标代码字节数的 BR 伪指令在段的开始位置和标签之间进行描述 .
  (4) 具有 ADDRESS 属性的 HIGH 绝对表达式 .
  (5) 具有 ADDRESS 属性的 LOW 绝对表达式 .
  (6) 具有 ADDRESS 属性的 DATAPOS 绝对表达式 .
  (7) 具有 ADDRESS 属性的 BITPOS 绝对表达式 .

(8) 表达式 4), (5), (6) 或 (7) 中满足下述条件 **(a)**：

(a) 在具有 ADDRESS 属性的表达式中的标签与该标签所属的段的开始位置之间的描述不能立即确定目标代码字节数的 BR 伪指令 .

- 如果在操作数的描述格式中存在错误，则汇编器将输出错误信息，但会尝试将操作数的值保存为名称的值，该名称在符号字段中被描述至它可以分析的程度 .
- 用 EQU 伪指令定义的名称不能在同一源模块中重新定义 .
- 已经用 EQU 伪指令定义了位数值的名称具有和数值一样的地址和位位置 .
- 表 3-12 显示了可描述为 EQU 伪指令操作数的位数值以及可以引用这些位数值的范围 :

表 3-12　代表了比特值操作符的表示格式

操作符类型	符号值	引用范围
A.bit 注 1	1.bit	只能在同一模块中引用 .
PSW.bit 注 1	1FEH.bit	
sfr 注 2.bit 注 1	0FFxxH 注 3.bit	
saddr.bit 注 1	0xxxxH 注 4.bit	可在其它模块中引用 .
expression.bit 注 1	0xxxxH 注 4.bit	

注　1.　1bit = 0 至 7

注　2.　关于描述的详细情况，参见各器件的用户手册 .

注　3.　"0FFxxH" 表示 sfr 的地址 .

注　4.　"0xxxxH" 表示 saddr 区 (FE20H 至 FF1FH).

[应用举例 **]**

```
        NAME    SAMP1

WORK1   EQU     0FE20H      ; (1)
WORK10  EQU     WORK1.0     ; (2)
P02     EQU     P0.2        ; (3)
A4      EQU     A.4         ; (4)
PSW5    EQU     PSW.5       ; (5)

        SET1    WORK10      ; (6)
        SET1    P02         ; (7)
        SET1    A4          ; (8)
        SET1    PSW5        ; (9)

        END
```

< 说明 >

(1) 名称 "WORK1" 具有值 "0FE20H", 符号属性 "NUMBER" 和重置属性 "ABSOLUTE".

(2) 名称 "WORK10" 被赋予操作数格式 "saddr.bit" 中的位数值 "WORK1.0". 上述 (1) 中，在操作数中描述的 "WORK1" 已经定义为 "0FE20H".

(3) 名称 "P02" 被赋予操作数格式 "sfr.bit" 中的位数值 "P0.2".

(4) 名称 "A4" 被赋予操作数格式 "A.bit" 中的位数值 "A.4".

(5) 名称 "PSW5" 被赋予操作数格式 "PSW.bit" 中的位数值 "PSW.5".

(6) 该描述相当于 "SET1 saddr.bit".

(7) 该描述相当于 "SET1 sfr.bit".

(8) 该描述相当于 "SET1 A.bit".

(9) 该描述相当于 "SET1 PSW.bit".

同 (3) 至 (5) 中一样 , 已经定义了 "sfr.bit", "A.bit" 和 "PSW.bit" 的名称只能在同一模块中引用 .

已经定义了 "saddr.bit" 的名称也可从其它模块中引用为外部定义符号 .（参见 "3.5 (2) EXTBIT (外部位)"）.

作为举例中汇编源模块的结果 , 生成以下汇编列表 :

< 汇编列表 >

```
        Assemble list

ALNO    STNO    ADRS    OBJECT      M I SOURCE STATEMENT

   1       1                               NAME    SAMP
   2       2

   3       3            ( FE20 )           WORK1   EQU     0FE20H          ; (1)
   4       4            ( FE20.0 )         WORK10  EQU     WORK1.0         ; (2)
   5       5            ( FF00.2 )         P02     EQU     P0.2            ; (3)
   6       6            ( 0001.4 )         A4      EQU     A.4             ; (4)
   7       7            ( 01FE.5 )         PSW5    EQU     PSW.5           ; (5)
   8       8    0000    0A20                       SET1    WORK10          ; (6)
   9       9    0002    2A00                       SET1    P02             ; (7)
  10      10    0004    61CA                       SET1    A4              ; (8)
  11      11    0006    5A1E                       SET1    PSW5            ; (9)
  12      12
  13      13                                       END
```

< 说明 >

对于汇编列表的第 2 行到第 5 行 , 在目标代码字段指出了作为名称定义的位值的位地址值 .

SET

(2) SET (设置)

[描述格式 **]**

符号字段	助记符字段	操作字段	注释字段
name	SET	absolute - expression	[; 注释]

[功能]

- SET 伪指令定义一个在操作字段中指定的表达式的名称 , 而该名称具有值和属性 （符号属性和重置属性） .
- 用 SET 伪指令定义的名称的值和属性可以在同一模块中重新定义 . 这些值和属性一直有效直到相同的名称被重新定义为止 .

[用途]

- 将源模块中使用的数字数据 （变量）定义为名称 , 并替代数字数据 （变量） , 描述指令的操作数中的名称 . 如果希望在源模块中改变名称的值 , 可再次使用 SET 伪指令为相同的名称定义一个不同的值 .

[说明]

- 绝对表达式必须在 SET 伪指令的操作字段中描述 .
- SET 指令可在源程序中任意位置描述 . 但是 , 已经用 SET 伪指令定义的名称不能被正向引用 .
- 如果在已经通过 SET 伪指令定义了名称的语句中检测到错误 , 那么汇编器将输出一个错误消息 , 但会尝试将操作数的值保存为名称的值 , 该名称在符号字段中被描述至它可以分析的程度 .
- 用 EQU 伪指令定义的符号不能用 SET 伪指令重新定义 .
  用 SET 伪指令定义的符号不能用 EQU 伪指令重新定义 .
- 不能定义位符号 .

[应用举例 **]**

```
        NAME    SAMP1

COUNT   SET     10H             ; (1)

        CSEG
        MOV     B , #COUNT      ; (2)
LOOP :
        DEC     B
        BNZ     $LOOP

COUNT   SET     20H             ; (3)

        MOV     B , #COUNT      ; (4)
        END
```

< 说明 >

(1) 名称 "COUNT" 具有值 "10H", 符号属性 "NUMBER" 和重置属性 "ABSOLUTE". 它们的值和属性一直有效直到其被以下 (3) 中 SET 伪指令重新定义为止 .

(2) 名称 "COUNT" 的值 "10H" 被转移到寄存器 B.

(3) 名称 "COUNT" 的值变成 "20H".

(4) 名称 "COUNT" 的值 "20H" 被转移到寄存器 B.

3.4　内存初始化和区域保留伪指令

内存初始化伪指令定义源程序中使用的常量数据 .

被定义常量数据的值作为目标代码产生 .

区域保留伪指令保存程序中使用的内存区域 .

以下显示的内存初始化和区域保留伪指令可用 :

- DB (定义字节)
- DW (定义字)
- DS (定义内存)
- DBIT (定义位)

DB

(1) DB (定义字节)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[label :]	DB	(size)	[; 注释]
		or	
[label :]	DB	initial - value [, ...]	[; 注释]

[功能]

- DB 伪指令命令汇编器对一个字节区域进行初始化 . 被进行初始化的字节的个数可以被指定为 " 容量 ".
- DB 伪指令也可以命令汇编器在字节单元中对内存区域进行初始化 , 其中字节单元具有在操作字段中所指定的初始值 .

[用途]

- 当定义程序中使用的表达式或字符串时使用 DB 伪指令 .

[说明]

- 如果操作字段中的值被加上了括号 , 则汇编器假设为容量已经被指定 . 否则会假设一个初始值 .
- 禁止在位段中描述 DB 伪指令

(1) 带有容量规范 :

(a) 如果在操作字段中指定容量 , 则汇编器会初始化一个区域 , 该区域等同于带有值 "00H" 的字节的指定个数 .

(b) 绝对表达式必须被描述为一个容量 . 如果容量描述是非法的 , 则汇编器将输出一个错误信息并且不执行初始化 .

(2) 带有初始值规范 :

(a) 表达式

表达式的值必须是 8 位数据 . 因此 , 操作数的值必须在 0H 至 0FFH 范围中 . 如果值超出 8 位 , 汇编器将使用该值的低 8 位作为有效值并输出一个错误信息 .

(b) 字符串

如果字符串被描述为操作数 , 则会为串中的各字符保留一个 8 位 ASCII 码 .

- 在 DB 伪指令的语句行中可以指定两个或更多的初始值 .
- 包括一个重置符号或外部引用符号的表达式可以描述为一个初始值 .

[应用举例]

```
                NAME    SAMP1
                CSEG
WORK1 :         DB      ( 1 )                   ; (1)
WORK2 :         DB      ( 2 )                   ; (1)
                CSEG
MASSAG :        DB      ' ABCDEF '              ; (2)
DATA1 :         DB      0AH , 0BH , 0CH         ; (3)
DATA2 :         DB      ( 3 + 1 )               ; (4)
DATA3 :         DB      ' AB ' + 1              ; (5)

                END
```

< 说明 >

(1) 由于已经指定容量 , 所以汇编器将对每个带有 "00H" 的字节区域进行初始化 .

(2) 6 字节区域可用字符串 'ABCDEF' 进行初始化 .

(3) 3 字节区域可用 "0AH, 0BH, 0CH" 进行初始化 .

(4) 4 字节区域可用 "00H " 进行初始化 .

(5) 因为表达式 "AB" + 1 的值为 4143H (4142H + 1), 超出了范围 0 至 0FFH, 所以该描述会产生错误 .

DW

(2) DW (定义字)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[label :]	DB	(size)	[; 注释]
		or	
[label :]	DB	initial - value [, ...]	[; 注释]

[功能]

- DW 伪指令命令汇编器对一个字区域进行初始化 . 被进行初始化的字的个数可以被指定为 " 容量 ".
- DW 伪指令也可以命令汇编器在字单元 (2 个字节) 中对内存区域进行初始化 , 其中字单元具有在操作字段中所指定的初始值 .

[用途]

- 当定义 16 位数字常量例如 : 程序中使用的地址或数据时使用 DW 伪指令 .

[说明]

- 如果操作字段中的值被加上了括号 , 则汇编器假设为容量已经被指定；否则将假定一个初始值 .
- 禁止在位段中描述 DW 伪指令 .

⑴ 带有容量规范 :

(a) 如果在操作字段中指定容量,则汇编器会初始化一个区域,该区域等同于带有值"00H"的字的指定个数.

(b) 绝对表达式必须被描述为一个容量 . 如果容量描述是非法的 , 则汇编器将输出一个错误信息并且不执行初始化 .

⑵ 带有初始值规范 :

(a) 常量

16 位或更少 .

(b) 表达式

表达式的值必须存储为 16 位数据

没有字符串可以描述为初始值 .

- 指定的初始值的高 2 位被存储在 HIGH 地址而该值的低 2 位被存储在 LOW 地址 .
- 在 DW 伪指令的语句行中可以指定两个或更多的初始值 .
- 包括一个重置符号或外部引用符号的表达式可以描述为一个初始值 .

[应用举例]

```
        NAME    SAMP1
        CSEG
WORK1 : DW      ( 10 )          ; (1)
WORK2 : DW      ( 128 )         ; (1)
        CSEG
        ORG     10H
        DW      MAIN            ; (2)
        DW      SUB1            ; (2)

        CSEG
MAIN :
        CSEG
SUB1 :

DATA :  DW      1234H , 5678H   ; (3)

        END
```

< 说明 >

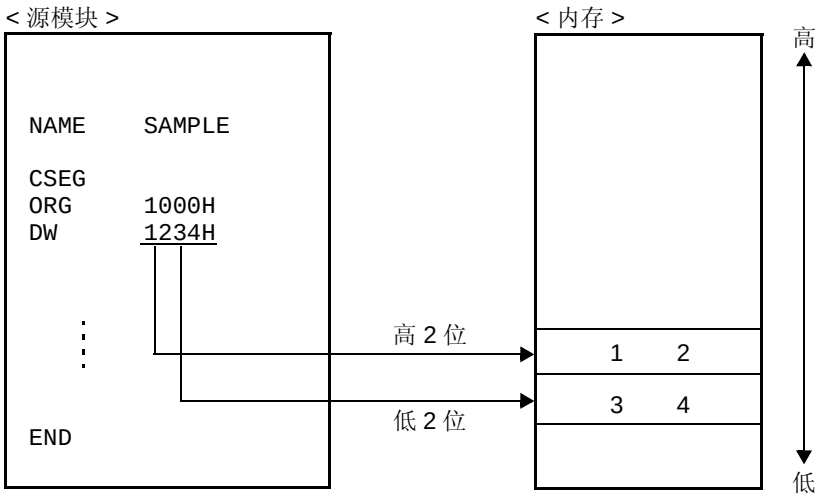
(1) 由于已经指定容量 , 所以汇编器将对每个带有 "00H" 的字区域进行初始化 .

(2) 用 DW 伪指令定义向量入口地址 .

(3) 2 字区域可用 ″34127856″ 进行初始化 .

备注　　内存的 HIGH 地址用字值的高 2 位来初始化 . 内存的 LOW 地址用字值的低 2 位来初始化 .

< 举例 >



DS

(3) DS (定义内存)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[label :]	DS	absolute - expression	[; 注释]

[功能]

- DS 伪指令命令汇编器保留操作字段中指定的字节数的内存区域 .

[用途]

- DS 伪指令主要用于保留程序中使用的内存 (RAM) 区域 . 如果指定了一个标签 , 则被保留的内存区域的第一个地址的值赋给该标签 . 在源模块中 , 标签用于描述对内存的操作 .

[说明]

- 用 DS 伪指令保留的区域的内容是未知的 (不确定的).
- 指定的绝对表达式将用无符号的 16 位计算 .
- 当操作数的值为 "0" 时 , 没有区域可被保留 .
- 禁止在位段中描述 DS 伪指令 .
- 用 DS 伪指令定义的符号 (标签) 只能进行反向引用 .
- 只可以在操作字段中描述以下从绝对表达式扩展而来的参数 :
  (1) 一个常量
  (2) 表达式，其中执行带常量的运算 (常量表达式)
  (3) 用常量或常量表达式定义的 EQU 符号或 SET 符号
  (4) 如果 " 具有 ADDRESS 属性的表达式 1" 中的标签 1 和 " 具有 ADDRESS 属性的表达式 2" 中的标签 2 都被重置 , 则两个标签必须在同一段中定义 .
  但是任意发生下列情况的一种就会产生错误 :
  (5) 如果标签 1 和标签 2 属于同一字段 , 且不能确定目标代码字节数的 BR 伪指令在两标签间进行描述 .
  (6) 如果标签 1 和标签 2 在段中不相同 , 且不能确定目标代码字节数的 BR 伪指令在任一标签和标签所属字段的开始位置间进行描述 .
  (7) 执行运算的上述 (1) 至 (4) 的任一一个表达式中 .

- 操作字段中不可以对以下参数进行描述 :

  (1) 外部引用符号

  (2) 已经用 EQU 伪指令定义了 " 具有 ADDRESS 属性的表达式 1 – 具有 ADDRESS 属性的表达式 2" 的符号

  (3) 位置计数器在表达式 1 或表达式 2 中以 " 具有 ADDRESS 属性的表达式 1 – 具有 ADDRESS 属性的表达式 2" 格式被描述

  (4) 用 EQU 伪指令定义的符号和可以执行运算符 HIGH / LOW / DATAPOS / BITPOS 的具有 ADDRESS 属性的表达式

[应用举例]

```
                NAME    SAMPLE
                DSEG
TABLE1 :        DS      10              ; (1)
WORK1 :         DS      1               ; (2)
WORK2 :         DS      2               ; (3)
                CSEG
                MOVW    HL , #TABLE1
                MOV     A , !WORK1
                MOVW    BC , #WORK2
                END
```

< 说明 >

(1) 保留 10 字节工作区 , 但是区域的内容未知 (不确定). 标签 "TABLE1" 分配在地址的开始 .

(2) 保留 1 字节工作区域 .

(3) 保留 2 字节工作区域 .

DBIT

(4) DBIT (定义位)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[name]	DBIT	None	[; 注释]

[功能]

- DBIT 伪指令命令汇编器在位字段中保留 1 位内存 .

[用途]

- 用 DBIT 伪指令在一个位段中保留一位区域 .

[说明]

- DBIT 伪指令只能在位段中进行描述 .
- 用 DBIT 伪指令保留的 1 位区域的内容是未知的 (不确定的).
- 如果在符号字段中指定了一个名称 , 则根据它的值该名称具有一个地址和一个位位置 .
- 定义的名称可以在 saddr.bit 要求的位置进行描述 .

[应用举例]

```
        NAME    SAMPLE
        BSEG
BIT1    DBIT            ; (1)
BIT2    DBIT            ; (1)
BIT3    DBIT            ; (1)

        CSEG
        SET1    BIT1    ; (2)

        CLR1    BIT2    ; (3)

        END
```

< 说明 >

(1) 通过三个 DBIT 伪指令 , 汇编器将保留三个 1 位区域 , 并定义名称 (BIT1, BIT2 和 BIT3), 每个名称具有一个地址和一个位位置作为它的值 .

(2) 该描述相当于 "SET1 saddr.bit", 并且将上述 (1) 中保留的位区域的名称 "BIT1" 描述为操作数 "saddr.bit".

(3) 该描述相当于 "CLR1 saddr.bit" 并且将名称 "BIT2" 描述为 "saddr.bit".

3.5 链接伪指令

链接伪指令阐明了引用其他模块内所定义的符号的相关性 .

考虑这样一种情形 : 将一个程序分成两个模块进行创建 , 即模块 1 和模块 2. 在模块 1 中 , 当引用在模块 2 中定义的符号时 , 在各个模块中未经声明不得使用 . 因此 , 需要在两个模块之间发布一些诸如 “ 我想使用该符号 ” 和 “ 你可以使用该符号 ” 等的信号或指示 .

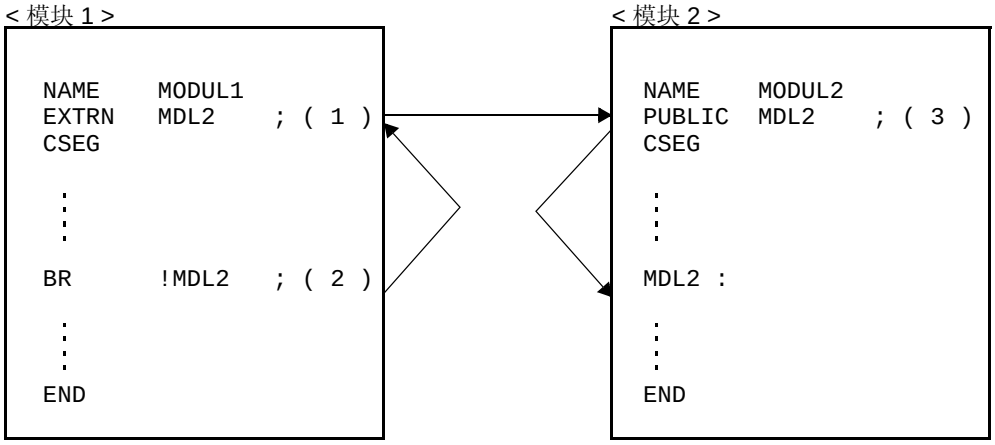
在模块 1 中 , 符号的外部引用声明表示必须引用在另一模块内定义的符号 . 在模块 2 中 , 符号的外部定义声明表示定义的符号可以在另一模块中引用 .

当两个外部引用声明和外部定义声明有效时 , 符号可被第一次引用 .

链接伪指令的功能是建立这种相互关系 , 以下两种类型可用 :

- 声明符号的外部引用 : EXTRN (外部) 和 EXTBIT (外部位) 伪指令
- 声明符号的外部定义 : PUBLIC (全局) 伪指令

图 3-6 两模块间符号的关系



在图 3-6 的模块 1 中 , (2) 中引用了模块 2 中定义才符号 "MDL2". 因此 , 该符号在 (1) 中用 EXTRN 伪指令声明为外部引用 .

在模块 2 中 ,(3) 中用 PUBLIC 伪指令将从模块 1 中引用的符号 "MDL2" 声明为外部定义 .

链接器检查符号的外部引用是否与符号的外部定义相一致 .

以下链接伪指令可用 :

- EXTRN (外部)
- EXTBIT (外部位)
- PUBLIC (全局)

EXTRN

(1) EXTRN (外部)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[label :]	EXTRN	symbol - name [, ...]	[; 注释]

[功能]

- EXTRN 伪指令向连接器声明其它模块中的符号 (位符号除外) 将在该模块中引用 .

[用途]

- 当引用其它模块中的符号时 ,EXTRN 伪指令必须将符号声明为外部引用 .

[说明]

- EXTRN 伪指令可以在源程序中任意位置描述 (参见 "2.1 基本构成 ").
- 在操作字段中通过用逗号 (,) 将每个符号名隔开 , 最多可以指定 20 个符号 .
- 当引用具有位值的符号时 , 必须用 EXTBIT 伪指令将该符号声明为外部引用 .
- 用 EXTRN 伪指令声明的符号在其它模块中必须用 PUBLIC 伪指令声明 .
- 没有宏名可描述为 EXTRN 伪指令的操作数 (参见有关宏名 " 第五章 宏 ").
- EXTRN 伪指令只允许在整个模块中有一个相对于符号的 EXTRN 声明 . 对于符号的第二个以及之后的 EXTRN 声明 , 连接器输出警告信息 .
- 已经被声明的符号不能描述为 EXTRN 伪指令的操作数 . 反之 , 已经被声明为 EXTRN 的符号不能用其它任何伪指令进行重新定义或声明 .
- 用 EXTRN 伪指令定义的符号可用于 saddr 区域的引用 .

[应用举例]

< 模块 1 >

```
        NAME    SAMP1
        EXTRN   SYM1 , SYM2     ; (1)
        CSEG
S1 :    DW      SYM1            ; (2)
        MOV     A , SYM2        ; (3)
        END
```

< 模块 2 >

```
        NAME    SAMP2
        PUBLIC  SYM1 , SYM2     ; (4)
        CSEG
SYM1    EQU     0FFH            ; (5)
DATA1   DSEG    SADDR
SYM2 :  DB      012H            ; (6)
        END
```

< 说明 >

(1) EXTRN 伪指令将 (2) 和 (3) 中引用的符号 "SYM1" 和 "SYM2" 声明为外部引用 . 在操作字段中有可能描述两个或更多的符号 .

(2) 该 DW 指令引用符号 "SYM1".

(3) 该 MOV 指令引用符号 "SYM2" 并输出引用 saddr 区的代码 .

(4) 符号 "SYM1" 和 "SYM2" 被声明为外部定义 .

(5) 符号 "SYM1" 被定义 .

(6) 符号 "SYM2" 被定义 .

EXTBIT

(2) EXTBIT (外部位)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[label :]	EXTBIT	bit - symbol - name [, ...]	[; 注释]

[功能]

- EXTBIT 伪指令向连接器声明其它模块中的具有 saddr.bit 值的位符号将在该模块中引用 .

[用途]

- 当引用一个具有位数值并且已经在其它模块中定义的符号时 , 必须使用 EXTBIT 伪指令将该符号声明为外部引用 .

[说明]

- EXTBIT 指令可在源程序中任意位置描述 .
- 在操作字段中通过用逗号 (,) 将每个符号隔开 , 最多可以指定 20 个符号 .
- 用 EXTBIT 伪指令声明的符号在其它模块中必须用 PUBLIC 伪指令声明 .
- EXTBIT 伪指令只允许在整个模块中有一个对于符号的 EXTBIT 声明 . 对于符号的第二个以及之后的 EXTBIT 声明 , 连接器输出警告信息 .

[应用举例]

< 模块 1 >

```
NAME    SAMP1
EXTBIT  FLAG1 , FLAG2           ; (1)
CSEG
SET1    FLAG1                   ; (2)
CLR1    FLAG2                   ; (3)
END
```

< 模块 2 >

```
        NAME    SAMP2
        PUBLIC  FLAG1 , FLAG2   ; (4)
        BSEG
FLAG1   DBIT                    ; (5)
FLAG2   DBIT                    ; (6)
        CSEG
        NOP
        END
```

〈 **说明** 〉

(1) EXTBIT 伪指令声明引用的 "FLAG1" 和 "FLAG2" 符号为外部引用 . 在操作字段中有可能描述两个或更多的符号 .

(2) SET1 指令引用符号 "FLAG1". 该描述相当于 "SET1 saddr.bit".

(3) CLR1 指令引用符号 "FLAG2". 该描述相当于 " CLR1 saddr.bit ".

(4) PUBLIC 伪指令定义符号 "FLAG1" 和 "FLAG2".

(5) DBIT 伪指令将符号 "FLAG1" 定义为 SADDR 区域的位符号 .

(6) DBIT 伪指令将符号 " FLAG2" 定义为 SADDR 区域的位符号 .

PUBLIC

(3) PUBLIC (全局)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[label :]	PUBLIC	symbol - name [, ...]	[; 注释]

[功能]

- PUBLIC 伪指令向连接器声明在操作字段中描述的符号是其它模块中要引用的符号 .

[用途]

- 当定义一个其它模块可引用的符号 (包括位符号) 时 , 必须用 PUBLIC 伪指令将该符号声明为外部定义 .

[说明]

- PUBLIC 指令可在源程序中任意位置描述 .
- 在操作字段中通过用逗号 (,) 将每个符号名隔开 , 最多可以指定 20 个符号 .
- 在操作字段中描述的符号必须在同一模块中进行定义 .
- PUBLIC伪指令只允许在整个模块中有一个对于一个符号的PUBLIC声明. 对符号的第二个以及之后的PUBLIC声明将被连接器忽略 .
- 以下符号不能用作 PUBLIC 伪指令的操作数 .
  (1) 用 SET 伪指令定义的名称
  (2) 用 EXTRN 或 EXTBIT 伪指令在相同模块中定义的符号
  (3) 段名
  (4) 模块名
  (5) 宏名
  (6) 没有在模块中定义的符号
  (7) 由 EQU 伪指令定义的其操作数具有位属性的符号
  (8) 用 EQU 伪指令定义了 sfr 的符号 (但是 , 不包括 sfr 和 saddr 重叠的区域)

[应用举例]

< 模块 1 >

```
        NAME    SAMP1
        PUBLIC  A1 , A2         ; (1)
        EXTRN   B1
        EXTBIT  C1

A1      EQU     10H
A2      EQU     0FE20H.1

        CSEG
        BR      B1
        SET1    C1
        END
```

< 模块 2 >

```
        NAME    SAMP2
        PUBLIC  B1              ; (2)
        EXTRN   A1
        CSEG
B1 :
        MOV     C , #LOW ( A1 )
        END
```

< 模块 3 >

```
        NAME    SAMP3
        PUBLIC  C1              ; (3)
        EXTBIT  A2
C1      EQU     0FE21H.0
        CSEG
        CLR1    A2
        END
```

< 说明 >

(1) PUBLIC 伪指令声明符号 "A1" 和 "A2" 将被其它模块引用 .

(2) PUBLIC 伪指令声明符号 "B1" 将被其它模块引用 .

(3) PUBLIC 伪指令声明符号 "C1" 将被其它模块引用 .

3.6 目标模块名声明伪指令

目标模块名声明伪指令对由 RA78K0S 汇编器创建的目标模块赋予一个模块名称 .

以下目标模块名声明伪指令可用 :

- NAME (名称)

NAME

(1) NAME (名称)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[label :]	NAME	object - module - name	[; 注释]

[功能]

- NAME 伪指令给汇编器输出的目标模块赋予一个在操作字段中描述的目标模块名 .

[用途]

- 调试器在调试中需要每一个目标模块都有一个模块名 .

[说明]

- NAME 指令可在源程序中任意位置描述 .
- 关于模块名描述的惯例 , 参见 "2.2.3 符号字段 " 中关于符号描述的惯例 .
- 可被定义为模块名的符号是那些除 "(" (28H) 或 ")" (29H) 或 kanji 字符外汇编器软件的操作系统所允许的字符 .
- 没有模块名可描述为除 NAME 外其它任意伪指令的操作数或者任意指令的操作数 .
- 如果 NAME 伪指令被忽略 , 则汇编器将假定输入源模块字段的主要名称 (前 8 个字符) 为模块名 . 在 Windows 版本中 , 主要名称被转换为用于索引的大写字母 . 如果指定两个或更多的模块名 , 则汇编器将输出警告信息并且忽略第二个及之后的模块名声明 .
- 在操作字段中描述的模块名不能超出 8 个字符 .
- 可区分符号名的大写和小写字符 .

[应用举例]

```
        NAME    SAMPLE  ; (1)
        DSEG
BIT1 :  DBIT

        CSEG
        MOV     A , B

        END
```

< 说明 >

(1) NAME 伪指令声明 "SAMPLE" 为模块名 .

3.7 自动分支指令选择伪指令

无条件分支指令直接把分支目的地址作为操作数 . 可使用如下两个指令 "BR !addr16" 和 "BR $addr16". 这些指令根据分支目的地址范围选择并使用最适当的操作数 . 由于每个伪指令的字节数不同 , 为了创建具有高内存利用率的程序 , 有必要使用具有最小字节数的指令 . 然而 , 在描述分支指令时考虑地址范围也是件很麻烦的事 .

因此 , 需要有一个伪指令根据分支目的的地址范围 , 指导汇编器自动选择两字节或三字节分支指令 . 这就是自动分支指令选择伪指令 .

以下自动分支指令选择伪指令可用 :

- BR (分支)

BR

(1) BR (分支)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[label :]	BR	expression	[; 注释]

[功能]

- BR 伪指令命令汇编器根据操作字段中指定的表达式值的范围自动选择 2 或 3 个字节的 BR 转移指令 , 并根据选择的指令生成目标代码 .

[用途]

- 如果分支目的地址在相邻 BR 伪指令的地址 -80H 至 +7FH 范围内 , 则可以描述 2 字节分支指令 "BR $addr16". 使用该指令与使用 3 字节分支指令 “BR !addr16” 相比 , 所需内存空间可减少一个字节 . 为创建一个具有高内存使用效率的程序 , 应积极使用 2 字节分支指令 .

  但是当描述分支指令时 , 考虑转移目的地址的范围是非常麻烦的 . 因此 , 如果不清楚是否可以描述 2 字节分支指令 , 则使用 BR 伪指令 .
- 当能够确定是使用 2 字节或 3 字节分支指令时 , 则采用相应的指令 . 与采用 BR 伪指令相比可缩短汇编时间 .

[说明]

- BR 伪指令只能在代码字段中使用 .
- 直接跳转目的地址可描述为 BR 伪指令的操作数 . 在表达式的开始不能描述表示当前位置计数器的 “$”.
- 为进行优化 , 必须满足以下条件 :

  (1) 在表达式中不能多于 1 个标签或正向引用符号 .

  (2) 不要描述带有 ADDRESS 属性的 EQU 符号 .

  (3) 不能对 “ 具有 ADDRESS 属性的表达式 1 – 具有 ADDRESS 属性的表达式 2” 描述用 EQU 定义的符号 .

  (4) 不要描述一个带有ADDRESS属性的表达式,其中在表达式上已经进行了HIGH/LOW/DATAPOS/ BITPOS 运算符的运算 .

  如果不满足这些条件 , 则选择 3 字节 BR 伪指令 .

[应用举例]

```
ADDRESS                 NAME    SAMPLE
                C1      CSEG    AT      50H
000050H                 BR      L1                  ; (1)
000052H                 BR      L2                  ; (2)
    :
00007DH         L1 :
    :
007FFFH         L2 :
                        END
```

< 说明 >

(1) BR伪指令产生2字节分支指令(BR $addr16),这是因为该行与分支目的地址间的位移在-80H和+7FH范围内.

(2) BR 伪指令将被 3 字节分支指令 (BR !addr16) 所替代 , 这是因为该行与转移目的地址间的位移不在 -80H 和 +7FH 范围内 .

3.8 宏伪指令

在描述一个源程序时 , 反复书写一串频繁使用的指令组是很令人厌烦的 , 也会增加描述或编码错误 .

使用宏伪指令的宏功能可省去重复描述同一组指令的需要 , 从而提高了程序编码的效率 . 宏的基本功能就是用一个名称代替一系列语句 .

以下宏伪指令可用 :

- MACRO (宏)
- LOCAL (局部)
- REPT (重复)
- IRP (不确定重复)
- EXITM (退出宏)
- ENDM (结束宏)

MACRO

(1) MACRO (宏)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
macro - name	MACRO : Macro body : ENDM	[formal - parameter [, ...]]	[; 注释] [; 注释]

[功能]

- MACRO 伪指令通过将符号字段中指定的宏名赋给在 MACRO 伪指令和 ENDM 伪指令之间描述的一系列语句 (称为宏体), 来执行宏定义 .

[用途]

- 用一个宏名定义源程序中频繁使用的系列语句 . 进行该定义后 , 只描述定义的宏名 (参见 "5.2.2 宏引用 "), 相应于宏名的宏体就会被展开 .

[说明]

- MACRO 伪指令必须与 ENDM 伪指令配对使用 .
- 关于在符号字段中描述的宏名 , 参见 "2.2.3 符号字段 " 中的符号描述惯例 .
- 若要引用宏 , 在助记符字段中描述已定义的宏名 .
- 关于在操作字段中描述形式参数 , 应用规则与在符号字段中描述相同 .
- 在每个宏伪指令中最多可以描述 16 个形式参数 .
- 形式参数仅在宏体中有效 .
- 任何保留字被描述为形式参数都会发生错误 . 但是 , 若用户定义的符号被描述 , 其将被优先识别为形式参数 .
- 形式参数的数目必须与实际参数的数目相同 .
- 在宏程序体内定义的名称或标记如果用 LOCAL 伪指令声明 , 则只对一次宏扩展有效 .
- 宏嵌套 (即在宏体中引用其它宏) 最多允许有八个层次 , 包括 REPT 和 IRP 伪指令 .
- 可在独立源模块中定义的宏的数目没有特别限制 . 也就是说 , 只要有可用的内存空间就可以定义宏 .
- 形式参数的定义行、引用行和符号名称不会被输出到交叉引用表 .
- 在一个宏体中一定不要定义两个或更多的段 . 如果进行定义 , 则会输出错误 .

[应用举例]

```
        NAME    SAMPLE

ADMAC   MACRO   PARA1 , PARA2   ; (1)
        MOV     A , #PARA1
        ADD     A , #PARA2
        ENDM                    ; (2)

        ADMAC   10H , 20H       ; (3)

        END
```

< 说明 >

(1) 通过指定宏名 "ADMAC" 和参数 "PARA1" 和 "PARA2" 定义一个宏 .

(2) 该伪指令表示宏定义结束 .

(3) 宏 "ADMAC" 被引用 .

LOCAL

(2) LOCAL (局部)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
None	LOCAL	symbol - name [, ...]	[; 注释]

[功能]

- LOCAL 伪指令声明在操作字段中指定的符号名是一个仅在宏体中有效的局部符号 .

[用途]

- 如果一个在宏体中定义了符号的宏被多次引用 , 则汇编器将输出一个符号双重定义的错误 . 通过使用 LOCAL 伪指令 , 可以多次引用 (也称调用) 在宏体中定义了符号的宏 .

[说明]

- 关于符号名称在操作字段中描述的惯例 , 参见 "2.2.3 符号字段 " 中的符号描述惯例 .
- 声明为LOCAL的符号在每次宏扩展时都会被符号"??RAn"(n = 0000至FFFF)代替. 完成宏替换后符号"??RAn" 将按全局符号的方式处理 , 并被保存到符号表中 , 这样就可以在符号名 "??RAn" 下被引用 .
- 如果在宏体中描述一个符号而且宏体被引用不止一次 , 则意味该符号在源模块中将被定义不止一次 . 因此 , 有必要声明符号为仅在宏体中有效的局部符号 .
- LOCAL 伪指令只能在宏定义中使用 .
- LOCAL 伪指令必须在使用操作字段中指定的符号之前被描述 (也就是说 ,LOCAL 伪指令必须在宏体的开始位置被描述).
- 在源模块中用 LOCAL 伪指令定义的符号名称必须各不相同 (也就是说 , 在各宏中使用的局部符号不能使用相同的名称).
- 只要局部符号都在一行 , 在操作字段中可以描述的局部符号的数目不受限制 . 但是 , 在一个宏体中的符号数目不能超过 64 个 . 如果声明 65 或更多个符号 . 则汇编器会输出错误信息并将宏定义保存为空的宏体 . 此时即使宏被调用也没有内容被扩展 .
- 用 LOCAL 伪指令定义的宏不能被嵌入 .
- 用 LOCAL 伪指令定义的符号不能从宏外被调用 (引用).
- 没有保留字可以在操作字段中被描述为符号名 . 但是 , 如果一个与用户定义符号相同的符号被描述 , 优先识别该符号为局部符号 .
- 声明为 LOCAL 伪指令的操作数的符号不被输出到交叉引用表和符号列表中 .
- 在宏扩展时 LOCAL 伪指令的语句行不被输出 .
- 如果在宏定义中执行一个 LOCAL 声明 , 而宏定义中的符号与宏定义中的形式参数具有相同的名称 , 则将输出一个错误 .

［应用举例 ］

```
        NAME    SAMPLE

        ; 宏定义
MAC1    MACRO
        LOCAL   LLAB            ; (1)
LLAB :                          ;
        BR      $LLAB           ; (2)
        ENDM                    ;

        ; 源文本
REF1 :  MAC1                    ; (3)

        BR      !LLAB           ; (4)   <-- 该描述是错误的 .

        REF2 :  MAC1            ; (5)

        END
```

〈说明 〉

(1) LOCAL 伪指令将符号名 "LLAB" 定义为局部符号 .

(2) BR 指令在宏 MAC1 中引用局部符号 "LLAB".

(3) 宏引用调用 MAC1.

(4) 因为局部符号 "LLAB" 在宏 MAC1 的定义外被引用 , 所以该描述导致错误 .

(5) 宏引用调用 MAC1.

以上应用举例的汇编列表如下：

< 汇编列表 >

```
        Assemble list

ALNO    STNO    ADRS    OBJECT  M I     SOURCE STATEMENT

  1       1                                     NAME    SAMPLE
  2       2                     M       MAC1    MACRO
  3       3                     M               LOCAL   LLAB            ; (1)
  4       4                     M       LLAB :
  5       5                     M               BR      $LLAB           ; (2)
  6       6                     M               ENDM
  7       7
  8       8     000000                  REF1 :  MAC1                    ; (3)
          9                     #1      ;
         10     000000         #1      ??RA0000:
         11     000000  14FE   #1              BR      $??RA0000       ; (2)
  9      12
 10      13     000002  2C0000                 BR      !LLAB           ; (4)
***  ERROR E2407 , STNO  13 ( 0 ) Undefined symbol reference ' LLAB '
***  ERROR E2303 , STNO  13 ( 13 ) Illegal expression
 11      14
 12      15     000005                  REF2 :  MAC1                    ; (5)
         16                     #1      ;
         17     000005         #1      ??RA0001 :
         18     000005  14FE   #1              BR      $??RA0001       ; (2)
 13      19
 14      20                                     END
```

REPT

(3) REPT (重复)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[label :]	REPT : ENDM	absolute - expression	[; 注释] [; 注释]

[功能]

- REPT 伪指令命令汇编器重复扩展该指令与 ENDM 伪指令之间描述的系列语句 (称为 REPT-ENDM 块), 重复次数等于操作字段中指定表达式的值 .

[用途]

- 用 REPT 伪指令和 ENDM 伪指令在源程序中重复描述系列语句 .

[说明]

- 如果 REPT 伪指令不与 ENDM 伪指令配对使用则将会发生错误 .
- 在 REPT-ENDM 块中 , 宏引用、 REPT 伪指令和 IRP 伪指令可被嵌套最多 8 个层次 .
- 如果在 REPT-ENDM 块中出现 EXITM 伪指令 , 则汇编器对 REPT-ENDM 块之后的扩展被终止 .
- 汇编控制指令可以在 REPT-ENDM 块中描述 .
- 宏定义不可以在 REPT-ENDM 块中描述 .
- 在操作字段中描述的绝对表达式用无符号 16 位进行计算 . 如果表达式的值为 0, 没有内容被扩展 .

[应用举例]

```
NAME    SAMP1
CSEG
        ; REPT-ENDM 块
REPT    3                       ; (1)
        INC     B
        DEC     C
        ; 源文本
ENDM                            ; (2)
END
```

< 说明 >

(1) REPT 伪指令命令汇编器将 REPT-ENDM 块连续扩展三次 .

(2) 该伪指令表示 REPT-ENDM 块结束 .

以上源程序进行汇编时 ,REPT-ENDM 块被扩展如以下汇编列表 :

< 汇编列表 >

```
NAME    SAMP1
CSEG
REPT    3
        INC     B
        DEC     C
ENDM
INC     B
DEC     C
INC     B
DEC     C
INC     B
DEC     C
END
```

由语句 (1) 和 (2) 定义的 REPT-ENDM 块已经被扩展三次 . 在汇编列表中 , 不显示源模块中被 REPT 伪指令定义的语句 (1) 和 (2).

IRP

(4) IRP (不确定重复)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[label :]	IRP	formal - parameter , <[actual - parameter [, ...]]>	[; 注释]
	:		
	ENDM		[; 注释]

[功能]

- IRP 伪指令命令汇编器重复扩展在 IRP 伪指令和 ENDM 伪指令之间描述的系列语句 (称为 IRP-ENDM 块), 重复次数等于用操作字段中用指定的实际参数替代形式参数时实际参数的个数

[用途]

- 用 IRP 伪指令和 ENDM 伪指令在源程序中重复描述系列语句 , 其中只有部分语句变成变量 .

[说明]

- IRP 伪指令必须与 ENDM 伪指令配对使用 .
- 在操作字段中最多可以描述 16 个实际参数 .
- 在 IRP-ENDM 块中 , 宏引用、 REPT 伪指令和 IRP 伪指令可被嵌套最多 8 个层次 .
- 如果在 IRP-ENDM 块中出现 EXITM 伪指令 , 则汇编器对 IRP-ENDM 块之后的扩展被终止 .
- 宏定义不可以在 IRP-ENDM 块中描述 .
- 汇编控制指令可以在 IRP-ENDM 块中描述 .

[应用举例]

```
NAME    SAMP1
CSEG

IRP     PARA , < 0AH , 0BH , 0CH >      ; (1)
        ; IRP-ENDM 块
        ADD     A , #PARA
        MOV     [ DE ] , A
ENDM                                    ; (2)
        ; 源文本
END
```

< 说明 >

(1) 形式参数是"PARA",实际参数是以下三个: "0AH", "0BH", and "0CH". IRP伪指令通知汇编器展开IRP-ENDM 程序块 3 次 (即实际参数的值), 同时形式参数 “PARA” 被实际参数 “0AH”, “0BH” 和 “0CH” 代替 .

(2) 该伪指令表示 REPT-ENDM 块结束 .

以上源程序进行汇编时 ,IRP-ENDM 块被扩展如以下汇编列表 :

< 汇编列表 >

```
NAME    SAMP1
CSEG
        ; IRP-ENDM 块
ADD     A , #0AH                ; (3)
MOV     [ DE ] , A
ADD     A , #0BH                ; (4)
MOV     [ DE ] , A
ADD     A , #0CH                ; (5)
MOV     [ DE ] , A
        ; 源文本
END
```

由语句 (1) 和 (2) 定义的 IRP-ENDM 块已经被扩展三次 (等于实际参数的个数).

(3) 在 ADD 指令中 ,PARA 被 0A 替代 .

(4) 在 ADD 指令中 ,PARA 被 0B 替代 .

(5) 在 ADD 指令中 ,PARA 被 0C 替代 .

EXITM

(5) EXITM (退出宏)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
[label :]	IEXITM	None	[; 注释]

[功能]

- EXITM伪指令强制终止由MACRO伪指令定义的宏体的扩展以及对REPT-ENDM或IRP-ENDM块的重复扩展.

[用途]

- 当用 MACRO 伪指令定义的宏体中使用条件汇编功能 (参见 "4.7 条件汇编控制指令 ") 时 , 主要使用该功能 .
- 如果条件汇编功能与其它指令在宏体中联合使用 , 则不必汇编的源程序有可能被汇编 , 除非用 EXITM 伪指令强制控制从宏中返回 . 这种情况下 , 一定要使用 EXITM 伪指令 .

[说明]

- 如果在宏体中描述 EXITM 伪指令 , 则直到 ENDM 伪指令的指令都将保存为宏体 .
- EXITM 伪指令表示仅在宏扩展中结束宏 .
- 如果在 EXITM 伪指令的操作字段中描述一些内容 , 汇编器将输出一个错误 , 但是会执行 EXITM 处理 .
- 如果宏程序体内出现 EXITM 伪指令 , 汇编器将强行从 IF/_IF/ELSE/ELSEIF/_ELSEIF/ENDIF 程序块的嵌套层里返回汇编器进入宏体的那一级嵌套层 .
- 如果 EXITM 伪指令出现在 INCLUDE 文件中 , 其中 INCLUDE 文件是扩展在宏体中描述的 INCLUDE 控制指令的结果 , 汇编器将接受 EXITM 伪指令并认为其有效 , 然后在该层终止宏扩展 .

[应用举例]

- 在此处的举例中使用了条件汇编控制指令 . 参见 "4.7 条件汇编控制指令 ".
- 关于宏体和宏表达式参见 " 第五章　宏 " .

```
        NAME    SAMP1
MAC1    MACRO                           ; (1)
        ; 宏体
        NOT1    CY
$       IF ( SW1 )                      ; (2)   <-- IF 块
                BT      A.1 , $L1
                EXITM                   ; (3)
$       ELSE                            ; (4)   <-- ELSE 块
                MOV1    CY , A.1
                MOV     A , #0
$       ENDIF                           ; (5)
$       IF ( SW2 )                      ; (6)   <-- IF 块
                BR      [ HL ]
$       ELSE                            ; (7)   <-- ELSE 块
                BR      [ DE ]
$       ENDIF                           ; (8)
        ; 源文本
        ENDM                            ; (9)

        CSEG
$       SET ( SW1 )                     ; (10)
        MAC1                            ; (11)  <-- 宏引用
L1 :    NOP
        END
```

< 说明 >

(1) 宏 "MAC1" 在宏体中使用条件汇编功能 (2) 和 (4) 至 (8).

(2) 此处为条件汇编定义了一个 IF 块 . 如果转换名 "SW1" 为真 (非 "0"), 则 ELSE 模块将被汇编 .

(3) 该伪指令强制终止 (4) 中及随后的宏体的扩展 .

如果省略了 EXITM 伪指令 , 当宏被展开时 , 汇编器继续 (6) 及后面的汇编处理 .

(4) 此处为条件汇编定义了一个 ELSE 块 . 如果转换名 "SW1" 为假 ("0"), 则 ELSE 模块将被汇编 .

(5) ENDIF 控制指令表示条件汇编的结束 .

(6) 此处为条件汇编定义了另一个 IF 块 . 如果转换名 "SW2" 为真 (非 "0"), 则以下的 IF 模块将被汇编 .

(7) 此处为条件汇编定义了另一个 ELSE 块 . 如果转换名 "SW2" 为假 ("0"), 则 ELSE 模块将被汇编 .

(8) ENDIF 指令表示 (6) 和 (7) 中的条件汇编处理结束 .

(9) 该伪指令表示宏体结束 .

(10) SET 伪指令给转换名称 "SW1" 赋真值 (非 "0"), 并且设置条件汇编的条件 .

(11) 宏引用调用 "MAC1".

当汇编上述例子中的源程序时 , 发生宏展开 , 如下所示 :

< 汇编列表 >

```
        NAME    SAMP1
MAC1    MACRO                   ; (1)
            :
        ENDM                    ; (9)
        CSEG
$       SET ( SW1 )             ; (10)
        MAC1                    ; (11)
        ; 宏扩展部分
        NOT1    CY
$       IF ( SW1 )
                BT      A.1 , $L1
        ; 源文本
L1 :    NOP
        END
```

通过访问 (11) 中的宏 , 宏 "MAC1" 的宏体被扩展 . 由于在 (10) 中转换名 “SW1” 设真值 , 则汇编宏体内的第一个 IF 块 . 因为 EXITM 伪指令在 IF 块的结束位置被描述 , 所以随后的宏扩展不被执行 .

ENDM

(6) ENDM (结束宏)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
None	ENDM	None	[; 注释]

[功能]

- ENDM 伪指令命令汇编器终止执行定义为宏功能的系列语句 .

[用途]

- ENDM 伪指令必须始终在 MACRO, REPT 和 / 或 IRP 伪指令后的一系列语句的结尾处进行描述 .

[说明]

- MACRO 伪指令和 ENDM 伪指令之间描述的系列语句变成宏体 .
- REPT 伪指令和 ENDM 伪指令之间描述的系列语句变成 REPT-ENDM 块 .
- IRP 伪指令和 ENDM 伪指令之间描述的系列语句变成 IRP-ENDM 块 .

[应用举例]

例 1 < MACRO-ENDM >

```
        NAME    SAMP1
ADMAC   MACRO   PARA1 , PARA2
                MOV     A , #PARA1
                ADD     A , #PARA2
        ENDM
            :
        END
```

例 2 < REPT-ENDM >

```
NAME    SAMP2
CSEG
    :
REPT    3
        INC     B
        DEC     C
ENDM
    :
END
```

例 3 < IRP-ENDM >

```
NAME    SAMP3
CSEG
    :
IRP     PARA , < 1 , 2 , 3 >
        ADD     A , #PARA
        MOV     [ DE ] , A
ENDM
    :
END
```

3.9 汇编终止伪指令

汇编终止伪指令通知汇编器源模块结束 . 汇编终止伪指令必须始终在每个源模块的结尾处进行描述 .

汇编器把位于汇编终止伪指令之前的一系列语句作为源模块进行处理 . 因此 , 如果在 REPT 块或 IRP 块内 ENDM 之前出现汇编终止伪指令 , 则 REPT 块或 IRP 块无效 .

以下汇编终止伪指令可用 :

- END (结束)

END

(1) END (结束)

[描述格式]

符号字段	助记符字段	操作字段	注释字段
None	END	None	[; 注释]

[功能]

- END 伪指令通知汇编器源模块结束 .

[用途]

- END 伪指令必须始终在每个源模块的结尾处进行描述 .

[说明]

- 汇编器连续汇编源模块直到在源模块中出现 END 伪指令 . 因此 , 在各源模块的结尾处都需要 END 伪指令 .
- 在 END 伪指令后总是输入换行 (LF) 码 .
- 如果在 END 伪指令后出现任何不同于空格、制表符、 LF 或注释的其它语句 , 则汇编器将输出一个警告信息 .

[应用举例]

```
NAME    SAMPLE
DSEG
    :
CSEG
    :
END             ; (1)
```

< 说明 >

(1) 总是在各源模块的结尾处描述 END 伪指令 .

第四章　　控制指令

本章说明控制指令 . 控制指令为汇编器的操作提供了详细指导 .

4.1　概述

在源程序中描述的控制指令为汇编器的操作提供详细的指导 .

这些指令不受代码生成影响 .

控制指令在以下类型时可用 :

表 4-1　控制指令列表

控制指令类型	控制指令
处理器类型定义控制指令	PROCESSOR
调试信息输出控制指令	DEBUG / NODEBUG, DEBUGA / NODEBUGA
前后对照列表输出控制指令	XREF / NOXREF, SYMLIST / NOSYMLIST
包含控制指令	INCLUDE
汇编列表控制指令	EJECT, LIST / NOLIST, GEN / NOGEN, COND / NOCOND, TITLE, SUBTITLE, FORMFEED / NOFORMFEED, WIDTH, LENGTH, TAB
条件汇编控制指令	IF / _IF / ELSEIF / _ELSEIF / ELSE / ENDIF, SET / RESET
其它汇编指令	TOL_INF, DGL, DGS

在源程序中描述控制指令的方式与描述汇编伪指令的方式相同 .

对于表 4-1 列出的控制指令 , 下列指令与在汇编程序起始命令行指定的汇编选项具有相同的功能 .

控制指令和命令行汇编选项之间的对应关系在 表 4-2 中给出 .

表 4-2　控制指令与汇编选项

控制指令	汇编选项
PROCESSOR	-c
DEBUG / NODEBUG	-g / -ng
DEBUGA / NODEBUGA	-ga / -nga
XREF / NOXREF	-kx / -nkx
SYMLIST / NOSYMLIST	-ks / -nks
TITLE	-lh
FORMFEED / NOFORMFEED	-lf / -nlf
WIDTH	-lw
LENGTH	-ll
TAB	-lt

关于指定控制指令和通过命令行指定汇编选项的方法，参见 RA78K0S 系列汇编器包操作用户手册 .

4.2 处理器类型定义控制指令

在源文件模块中，处理器类型定义控制指令定义用于汇编的目标设备的类型 .

以下类型的处理器类型定义控制指令可用 :

- PROCESSOR (processor)

PROCESSOR

(1) PROCESSOR (processor)

[编写格式]

```
[ Δ ] $ [ Δ ] PROCESSOR [ Δ ]( [ Δ ] processor type [ Δ ] )
[ Δ ] $ [ Δ ] PC [ Δ ] ( [ Δ ] processor type [ Δ ] )   ; 缩写形式
```

[功能]

- 在源文件模块中,PROCESSOR 控制指令定义用于汇编的目标设备的类型.

[用途]

- 用于汇编的目标设备的处理器类型必须在源文件模块文件或起始命令行指定.
- 如果各源文件模块没有定义用于汇编的目标设备的处理器类型 , 则必须在每次汇编操作中指定处理器类型 . 因此, 通过在各汇编模块文件中指定用于汇编的目标设备, 可在启动汇编程序时节省时间并减少麻烦.

[说明]

- PROCESSOR 控制指令仅能在源模块文件的开头部分进行描述 . 如果在其它地方描述该控制指令 , 则汇编将会失败.
- 如果指定的处理器类型与实际的目标设备不符, 则汇编将会失败.
- 在模块开头仅能指定一个 PROCESSOR 控制指令.
- 用于汇编的目标设备的处理器类型也可用汇编选项 (-c) 指定 . 如果源模块文件中定义的处理器类型与起始命令行定义的不同, 汇编器会输出警告信息, 并优先选择在起始命令行定义的处理器类型.
- 即使在起始命令行定义了汇编选项 (-c), 汇编器也会对 PROCESSOR 控制指令执行语法检查.
- 如果在源模块文件或起始命令行都未指定处理器类型, 则会汇编失败.

[应用示例]

```
$       PROCESSOR ( 9026 )
$       DEBUG
$       XREF

        NAME    TEST
            :
        CSEG
```

4.3 调试信息输出控制指令

在源模块文件中，用调试信息输出控制指令指定是否对由源模块文件创建的目标模块文件输出调试信息．

以下调试信息输出控制指令可用：

- DEBUG / NODEBUG (调试 / 不调试)
- DEBUGA / NODEBUGA (调试 / 不调试)

DEBUG / NODEBUG

(1) DEBUG / NODEBUG (调试 / 不调试)

[编写格式]

```
[ Δ ] $ [ Δ ] DEBUG      ; 缺省假定
[ Δ ] $ [ Δ ] DG         ; 缩写形式
[ Δ ] $ [ Δ ] NODEBUG
[ Δ ] $ [ Δ ] NODG       ; 缩写形式
```

[功能]

- DEBUG 控制指令通知汇编器对目标模块文件增加局部符号信息 .
- NODEBUG 控制指令通知汇编器不对目标模块文件增加局部符号信息 . 但是 , 在这种情况下会对目标模块文件输出一个段名 .
- " 局部符号信息 " 指除了模块名称和 PUBLIC, EXTRN, 以及 EXTBIT 符号之外的符号 .

[用法]

- 当执行包含局部符号的符号调试时 , 使用 DEBUG 控制指令 .
- 以下情况时使用 NODEBUG 控制指令 :
  (1) 仅对全局符号执行符号调试时
  (2) 执行无符号调试时
  (3) 仅需要目标时 (如用 PROM 估值)

[说明]

- DEBUG 或 NODEBUG 控制指令仅能在源模块文件的开头部分进行描述 .
- 如果省略 DEBUG 或 NODEBUG 控制指令 , 则汇编器会假定已指定了 DEBUG 指令 .
- 通过在起始命令行使用汇编选项 (-g / -ng) 来定义增加局部符号信息 .
- 如果在源模块文件中的控制指令与在起始命令行中定义的不同 , 则在命令行中的定义优先 .
- 即使指定了汇编选项 (-ng), 汇编器也会对 DEBUG 或 NODEBUG 控制指令进行语法检查 .

DEBUGA / NODEBUGA

(2) DEBUGA / NODEBUGA (调试 / 不调试)

[编写格式]

```
[ Δ ] $ [ Δ ] DEBUGA              ; 缺省假定
[ Δ ] $ [ Δ ] NODEBUGA
```

[功能]

- DEBUGA 控制指令通知汇编器对目标模块文件增加汇编源调试信息 .
- NODEBUGA 控制指令通知汇编器不对目标模块文件增加汇编源调试信息 .

[用法]

- 当在汇编程序或在结构化汇编源程序级上进行调试时使用 DEBUGA 控制指令 . 在源程序级上进行调试时将需要集成调试器 .
- 下列情况下使用 NODEBUGA 控制指令 :
  (1) 执行无汇编源的调试
  (2) 仅需要对象 (如用 PROM 估值)

[说明]

- DEBUGA 或 NODEBUGA 控制指令仅能在源模块文件的开头部分进行描述 .
- 如果省略 DEBUGA 或 NODEBUGA 控制指令 , 则汇编器会假定已指定了 DEBUGA 指令 .
- 如果指定了两个或更多的控制指令 , 则最后被指定的控制指令优先级高于其它控制指令 .
- 使用位于起始行的汇编选项 (-ga / -nga) 可指定增加汇编源程序的调试信息 .
- 如果在源模块文件中定义的控制指令与在起始命令行中定义的不同 , 则在命令行中定义的控制指令优先 .
- 即使已指定了汇编选项 (-nga), 汇编器也会对 DEBUGA 或 NODEBUA 控制指令进行语法检查 .
- 如果通过 C 编译器或结构化汇编预处理器来编译或结构化汇编某调试输出信息 , 则在汇编该输出汇编源时不描述调试信息输出控制指令 . 汇编时所必需的控制指令作为 C 编译器或结构化汇编预处理器的控制语句输出到汇编源 .

4.4 交叉引用列表输出指定控制指令

在源模块文件中，交叉引用列表输出指定控制指令用于指定是否输出交叉引用列表 .

以下交叉引用列表输出指定控制指令可用 :

- XREF / NOXREF (xref / noxref)
- SYMLIST / NOSYMLIST (symlist / nosymlist)

XREF / NOXREF

(1) XREF / NOXREF (xref / noxref)

[编写格式]

```
[ Δ ] $ [ Δ ] XREF
[ Δ ] $ [ Δ ] XR        ; 缩写形式
[ Δ ] $ [ Δ ] NOXREF    ; 缺省假定
[ Δ ] $ [ Δ ] NOXR      ; 缩写形式
```

[功能]

- XREF 控制指令通知汇编器输出交叉引用列表至汇编列表文件 .
- NOXREF 控制指令通知汇编器不输出交叉引用列表至汇编列表文件 .

[用途]

- 当所获得的信息来自从源模块文件中定义的每个符号均被引用或者有多少这样的符号在源模块中被引用时 ，使用 XREF 控制指令输出交叉引用列表 .
- 如果必须在每个汇编操作中指定是否输出交叉引用列表，则在源模块文件中指定 XREF 和 NOXREFI 控制指令可节省时间和精力 .

[说明]

- XREF 或 NOXREF 控制指令仅能在源模块文件的开头部分进行描述 .
- 如果指定了两个或两个以上的控制指令，则最后被指定的控制指令优先级高于其它的控制指令 .
- 也可通过起始命令行的汇编选项 (-kx / -nkx) 来指定是否输出交叉引用列表 .
- 如果源模块文件中定义的控制指令与起始命令行中定义的控制指令不同 ， 则在命令行中定义的控制指令优先于源模块中定义的控制指令 .
- 即使在起始命令行指定了汇编选项 (-np) , 汇编器也会对 XREF / NOXREF 控制指令进行语法检查 .

SYMLIST / NOSYMLIST

(2) SYMLIST / NOSYMLIST (symlist / nosymlist)

[编写格式]

```
[ Δ ] $ [ Δ ] SYMLIST
[ Δ ] $ [ Δ ] NOSYMLIST          ; 缺省假定
```

[功能]

- SYMLIST 控制指令通知汇编器输出一个符号列表到列表文件 .
- NOSYMLIST 控制指令通知汇编器不输出符号列表至列表文件 .

[用途]

- 使用 SYMLIST 控制指令输出一个符号列表 .

[说明]

- SYMLIST 或 NOSYMLIST 控制指令仅能在源模块文件的开头部分进行描述 .
- 如果指定两个或两个以上的控制指令 , 则最后指定的控制指令优先级高于其它控制指令 .
- 也可通过起始命令行的汇编选项 (-ks / -nks) 指定是否输出交叉引用列表 .
- 如果源模块文件中定义的控制指令与起始命令行中定义的控制指令不同 , 则在起始命令行中定义的控制指令优先于在源模块中定义的控制指令 .
- 即使在命令起始行指定了汇编选项 (-np) , 汇编器也会对 SYMLIST / NOSYMLIST 控制指令进行语法检查 .

4.5 包含控制指令

在源模块文件中，包含控制指令用于说明在源模块文件中包含另一个模块文件 .

通过有效利用该控制指令，可在描述源程序时节省时间和精力 .

以下控制指令可用：

- INCLUDE (include)

INCLUDE

(1) INCLUDE (include)

[编写格式]

```
[ Δ ] $ [ Δ ] INCLUDE [ Δ ] ( [ Δ ] filename [ Δ ] )
[ Δ ] $ [ Δ ] IC [ Δ ] ( [ Δ ] filename [ Δ ] )          ; 缩写形式
```

[功能]

- INCLUDE 控制指令通知汇编器从汇编源程序的指定行开始 , 插入并展开一个指定文件的内容 .

[用途]

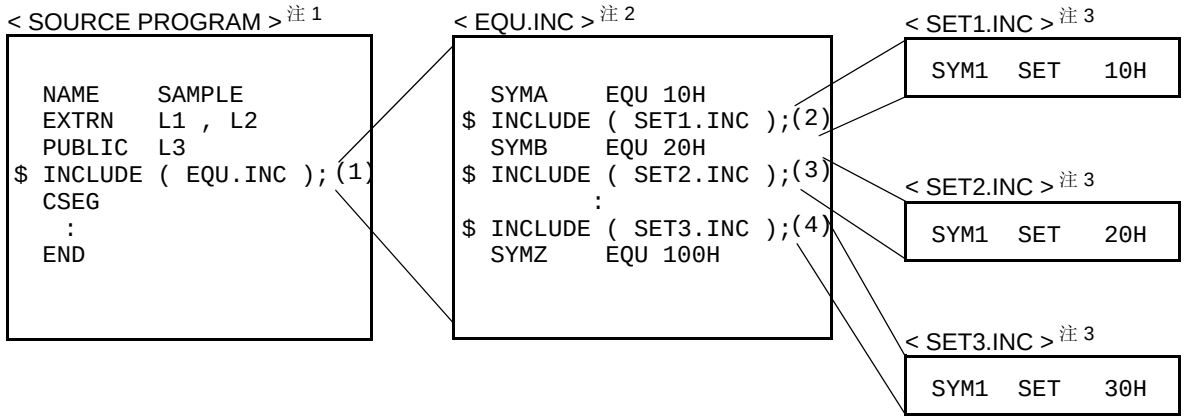
- 可被两个或两个以上源模块共享的一大组语句应组合成一个称为 INCLUDE 文件的独立文件 . 如果必须在每个源模块中使用该组语句 , 则用 INCLUDE 控制指令指定所需的 INCLUDE 文件的文件名 . 在描述源模块中使用该控制指令可大大减少时间和精力 .

[说明]

- INCLUDE 控制指令仅能在普通源程序中进行描述 .
- INCLUDE 文件的路径或驱动名称可通过汇编选项 (-I) 指定 .
- 汇编器按照以下顺序搜索 INCLUDE 文件的读取路径 :
  (1) 指定没有路径名定义的 INCLUDE 文件时
    (a) 存在源文件的路径
    (b) 由汇编选项 (-I) 指定的路径
    (c) 由环境变量 NC78K0S 指定路径
  (2) 指定具有路径名的 INCLUDE 文件时
    如果指定的 INCLUDE 文件具有以 (\ ) 开头的驱动名称或路径名称，则 与 INCLUDE 文件一起指定的路径将作为 INCLUDE 文件名的前缀 . 如果使用相对路径 (不以 \ 开头) 指定 INCLUDE 文件 , 则按照上述 (a) 的顺序将路径名作为 INCLUDE 文件名的前缀 .
- INCLUDE 文件允许嵌套的层数最多为 7 级 . 换言之 , 在汇编列表中 INCLUDE 文件显示的嵌套层数最多为 8 (这里，术语 " 嵌套 " 指在一个 INCLUDE 文件中一个或多个 INCLUDE 文件的定义).
- 不必在 INCLUDE 文件中描述 END 指令 .
- 如果不能打开指定的 INCLUDE 文件 , 则汇编器终止操作 .
- 必须用 INCLUDE 文件内与 ENDIF 控制指令成对出现的 IF 或 _IF 控制指令关系 INCLUDE 文件 . 如果在 INCLUDE文件入口处的IF级与紧接着INCLUDE文件展开的IF级不符,则汇编器输出错误信息并强制IF级返回到 INCLUDE 文件展开入口处的 IF 级 .

- 当在 INCLUDE 文件中定义宏时 , 必须在 INCLUDE 文件中关闭宏定义 . 如果在 INCLUDE 文件中意外的出现了 ENDM 指令 (没有对应的 MACRO 指令), 则会输出错误信息并忽略 ENDM 指令 . 如果在 INCLUDE 文件里描述的 MACRO 伪指令缺少 ENDM 指令 , 则编译器会输出错误信息 , 但会假定已描述了响应的 ENDM 伪指令将宏定义进行处理 .

[应用示例]



注 1. 在一个源文件中可定义两个或两个以上的 $IC 控制指令. 同一个 INCLUDE 文件也可被多次指定.

注 2. 可为 INCLUDE 文件 ″EQU.INC″ 指定两个或两个以上的 $IC 控制指令 .

注 3. 在任何的INCLUDE 文件 ″SET1.INC″, ″SET2.INC″, 和 ″SET3.INC″中都不能指定$IC控制指令.

< 说明 >

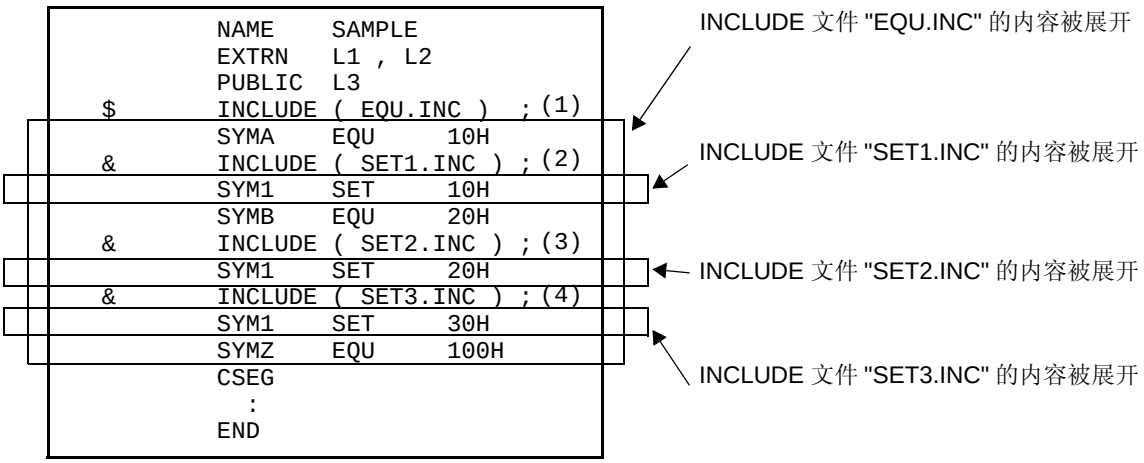
(1) 该控制指令指定 "EQU.INC" 为 INCLUDE 文件 .

(2) 该控制指令指定 "SET1.INC" 为 INCLUDE 文件 .

(3) 该控制指令指定 "SET2.INC" 为 INCLUDE 文件 .

(4) 该控制指令指定 "SET3.INC" 为 INCLUDE 文件 .

当汇编该源程序时 , INCLUDE 文件的内容展开如下 :



4.6 汇编列表控制指令

在源模块文件中，使用汇编列表控制指令控制汇编列表的输出格式，诸如换页，抑制列表输出和副标题输出等.

汇编列表控制指令包括：

- EJECT (eject)
- LIST / NOLIST (list / nolist)
- GEN / NOGEN (generate / no generate)
- COND / NOCOND (condition / no condition)
- TITLE (title)
- SUBTITLE (subtitle)
- FORMFEED / NOFORMFEED (formfeed / noformfeed)
- WIDTH (width)
- LENGTH (length)
- TAB (tab)

EJECT

(1) EJECT (eject)

[编写格式]

```
[ Δ ] $ [ Δ ] EJECT
[ Δ ] $ [ Δ ] EJ          ; 缩写形式
```

[缺省假定]

- 未指定 EJECT 控制指令 .

[功能]

- EJECT 控制指令会导致汇编器执行汇编列表的换页操作 (走页).

[用途]

- 在汇编模块需要弹出汇编列表的那一行描述 EJECT 控制指令 .

[说明]

- EJECT 控制指令仅能在普通源程序中进行描述 .
- 在输出 EJECT 控制指令本身的图标之后执行汇编列表的换页操作 .
- 如果在起始命令行指定汇编选项 (-np) 或 (-llo), 或通过另一个控制指令禁止输出汇编列表 , 则 EJECT 控制指令变为无效 . 关于这些汇编选项 , 参见 RA 78K0S 系列汇编器包的操作 用户手册 .
- 如果在 EJECT 控制指令之后有非法描述 , 则汇编器会输出错误信息 .

[应用示例]

```
            :
        MOV     [ DE+ ] , A
        BR      $$
$       EJECT                   ; (1)
        CSEG
            :
        END
```

< 说明 >

(1) 当使用 EJECT 控制指令执行换页操作时 , 输出的汇编列表如下所示 .

< 汇编列表 >

```
            :
        MOV     [ DE+ ] , A
        BR      $$
$       EJECT                   ; (1)
------------------------------------------------ 换页 ---------
        CSEG
            :
        END
```

LIST / NOLIST

(2) LIST / NOLIST (list / nolist)

[编写格式]

```
[ Δ ] $ [ Δ ] LIST      ; 缺省假定
[ Δ ] $ [ Δ ] LI        ; 缩写形式
[ Δ ] $ [ Δ ] NOLIST
[ Δ ] $ [ Δ ] NOLI      ; 缩写形式
```

[功能]

- LIST 控制指令指出必须开始汇编器汇编列表输出的行 .
- NOLIST 控制指令指出必须抑制汇编器汇编列表输出的行 .

  所有在NOLIST控制指令之后被描述的语句都将被汇编, 但是不会输出至汇编列表, 直到在源程序中出现 LIST 控制指令 .

[用途]

- 使用 NOLIST 控制指令来限制汇编列表输出的数量 .
- 使用 LIST 控制指令来取消由 NOLIST 控制指令指定的汇编列表输出的抑制 .

  通过组合使用 NOLIST 和 LIST 控制指令 , 可以控制汇编列表输出的数量以及列表的内容 .

[说明]

- LIST / NOLIST 控制指令仅能在普通源程序中进行描述 .
- NOLIST 控制指令用来抑制汇编列表输出并不打算停止汇编处理进程 .
- 如果在 NOLIST 控制指令之后指定 LIST 控制指令 , 则会在汇编列表中再次输出 LIST 控制指令之后描述的语句 . LIST 或 NOLIST 控制指令的图标也会在汇编列表中输出 .
- 如果 LIST 和 NOLIST 控制指令均未被指定 , 则将源模块中的所有语句输出到汇编列表 .

[应用示例]

```
        NAME    SAMP1
$       NOLIST        ; (1)
DATA1   EQU     10H   ; 该语句不会输出至汇编列表 .
DATA2   EQU     11H   ; 该语句不会输出至汇编列表 .
            :         ; 该语句不会输出至汇编列表 .
DATAX   EQU     20H   ; 该语句不会输出至汇编列表 .
DATAY   EQU     20H   ; 该语句不会输出至汇编列表 .
$       LIST          ; (2)
        CSEG
            :
        END
```

< 说明 >

(1) 由于这里指定了 NOLIST 控制指令 , 所以 "$ NOLIST" 之后的语句和 (2) 中 LIST 控制指令之前的语句将不会在汇编列表中输出 . NOLIST 控制指令本身的图标也会在汇编列表中输出 .

(2) 由于这里指定了 LIST 控制指令 , 所以该控制指令之后的语句会再次在汇编列表中输出 . LIST 控制指令本身的图标也会在汇编列表中输出 .

GEN / NOGEN

(3) GEN / NOGEN (generate / no generate)

[编写格式]

```
[ Δ ] $ [ Δ ] GEN        ; 缺省假定
[ Δ ] $ [ Δ ] NOGEN
```

[功能]

- GEN 控制指令通知汇编器将宏定义行 , 宏引用行和宏展开行输出至汇编列表 .
- NOGEN 控制指令通知汇编器将宏定义行和宏引用行 , 但抑制宏展开行 .

[用途]

- 使用 GEN / NOGEN 控制指令来限制汇编列表输出的数量 .

[说明]

- GEN / NOGEN 控制指令仅能在普通源程序中进行描述 .
- 如果 GEN 和 NOGEN 控制指令均未被指定 , 则宏定义行 , 宏引用行和宏展开行都将被输出至汇编列表 .
- 在 GEN 或 NOGEN 控制指令本身的图像输出至汇编列表之后 , 才发生指定的列表控制 .
- 即使在 NOGEN 控制指令控制了列表输出之后 , 汇编器仍继续其处理 , 并累加语句数量计数器 (STNO).
- 如果在 NOGEN 控制指令之后指定了 GEN 控制指令 , 汇编器将恢复输出宏展开行 .

[应用示例]

```
        NAME    SAMP
$       NOGEN               ; (1)
ADMAC   MACRO   PARA1 , PARA2
                MOV     A , #PARA1
                ADD     A , #PARA2
        ENDM
        CSEG
        ADMAC   10H , 20H
        END
```

当汇编上述源程序时，输出汇编列表如下所示．当汇编该源程序时，汇编列表将被展开如下：

< 汇编列表 >

```
        NAME    SAMP1
$       NOGEN               ; (1)
ADMAC   MACRO   PARA1 , PARA2
                MOV     A , #PARA1
                ADD     A , #PARA2
        ENDM
        CSEG
        ADMAC   10H , 20H
        MOV     A , #10H    ; 不输出宏展开行．
        AUD     A , #20H    ; 不输出宏展开行．
        END
```

< 说明 >

(1) 因为指定了 NOGEN 控制指令，所以宏展开行将不输出至汇编列表．

COND / NOCOND

(4) COND / NOCOND (condition / no condition)

[编写格式]

```
[ Δ ] $ [ Δ ] COND      ; 缺省假定
[ Δ ] $ [ Δ ] NOCOND
```

[功能]

- COND 控制指令通知汇编器将已满足汇编条件的行以及不满足汇编条件的行输出至汇编列表 .
- NOCOND 控制指令告知汇编器仅将已经满足汇编条件的行输出至汇编列表 . 不满足汇编条件的行以及已经描述了 IF/_IF, ELSEIF/_ELSEIF, ELSE 和 ENDIF 的行将被抑制 .

[用途]

- 使用 COND / NOCOND 控制指令来限制汇编列表输出的数量 .

[说明]

- COND / NOCOND 控制指令仅能在普通源程序中进行描述 .
- 如果 COND 和 NOCOND 控制指令均未被指定，则汇编器将已满足汇编条件的行以及不满足汇编条件的行输出至汇编列表 .
- 在 COND 或 NOCOND 控制指令本身的图像输出至汇编列表之后，才发生指定的列表控制 .
- 即使在 NOCOND 控制指令控制了列表输出之后，汇编器仍累加 ALNO 和 STNO 计数 .
- 如果在 NOCOND 控制指令之后指定 COND 控制指令，则汇编器将恢复输出那些满足汇编条件的行以及已经描述了 IF/_IF, ELSEIF/_ELSEIF, ELSE 和 ENDIF 的行 .

[应用示例]

```
        NAME    SAMP

$       NOCOND
$       SET ( SW1 )
$       IF ( SW1 )              ; 该部分虽被汇编，但是不输出至汇编列表 .
                MOV A , #1H
$       ELSE                    ; 该部分虽被汇编，但是不输出至汇编列表 .
                MOV A , #0H     ; 该部分虽被汇编，但是不输出至汇编列表 .
$       ENDIF                   ; 该部分虽被汇编，但是不输出至汇编列表 .
        END
```

TITLE

(5) TITLE (title)

[编写格式]

```
[ Δ ] $ [ Δ ] TITLE [ Δ ] ( [ Δ ] ' title - string ' [ Δ ] )
[ Δ ] $ [ Δ ] TT [ Δ ] ( [ Δ ] ' title - string ' [ Δ ] ) ; 缩写形式
```

[默认缺省]

- 当没有指定 TITLE 控制指令时，将空着汇编列表头的 TITLE 列 .

[功能]

- TITLE 控制指令指明将在汇编列表，符号表列表或交叉引用表的每一页开头的 TITLE 列上要打印的字符串 .

[用途]

- 使用 TITLE 控制指令在列表的每一页上打印一个标题，从而使列表的内容容易识别 .
- 如果每次汇编时必须由汇编选项指定标题 ， 那么在源模块中描述该控制指令在启动汇编器后会节约时间和精力 .

[说明]

- TITLE 控制指令仅能在源模块文件的开头部分进行描述 .
- 如果同时指定了两个或两个以上的 TITILE 控制指令，则汇编器仅认为最后指定的 TTIEL 控制指令有效 .
- 标题字符串最多可指定 60 个字符 . 如果被指定标题串包含 61 个或更多字符，汇编器只认为标题字符串的前 60 个字符有效 .
- 如果要使用单引号标志 （’）作为标题串的一部分，则连续写两个单引号标志 .
- 如果没有指定标题串 （标题串的字符数 =0），汇编器保留 TITLE 列空白 .
- 如果在指定的标题串中发现 **2.2.2 字符集** 之外的任何字符，汇编器将输出 “!” 代替 TITLE 列中的非法字符 .
- 汇编列表的标题也可通过位于汇编器起始命令行的汇编选项 （-lh）指定 .

[应用示例]

```
$       PROCESSOR ( 9026 )
$       TITLE ( ' THIS IS TITLE ' )
        NAME    SAMPLE
        CSEG
        MOV     A , B
        END
```

当汇编上述源程序时，输出汇编列表如下一页显示（每页行数为 72）.

< 汇编列表 >

```
78K/0S Series Assembler Vx.xx    THIS IS TITLE    Date:xx xxx xxxx  Page : 1

Command :       sample.asm
Para-file :
In-file :       SAMPLE.ASM
Obj-file :      SAMPLE.REL
Prn-file :      SAMPLE.PRN

         Assemble list

ALNO    STNO    ADRS    OBJECT  M I     SOURCE STATEMENT

 1       1                              $       PROCESSOR ( 9026 )
 2       2                              $       TITLE ( ' THIS IS TITLE ' )
 3       3                                      NAME    SAMPLE
 4       4       ----                           CSEG
 5       5       0000    63                     MOV     A , B
 6       6                                      END

Segment information :

ADRS    LEN     NAME

0000    0001H   ?CSEG

Target chip : uPD789026
Device file : Vx.xx
Assembly complete , 0 error ( s ) and 0 warning ( s ) found. ( 0 )
```

SUBTITLE

(6) SUBTITLE (subtitle)

[编写格式]

```
[ Δ ] $ [ Δ ] SUBTITLE [ Δ ] ( [ Δ ] ' title - string ' [ Δ ] )
[ Δ ] $ [ Δ ] ST [ Δ ] ( [ Δ ] ' title - string ' [ Δ ] ) ; 缩写形式
```

[默认缺省]

- 当没有指定 SUBTITLE 控制指令时，将空着汇编列表头的 TITLE 部分 .

[功能]

- SUBTITLE 控制指令指定将在汇编列表每页的 SUBTITLE 部分打印的字符串 .

[用途]

- 使用 SUBTITLE 控制指令在汇编列表的每一位打印一个副标题，从而使汇编列表的内容易于识别 . 每页副标题的字符串可以改变 .

[说明]

- SUBTITLE 控制指令仅能在普通源程序中进行描述 .
- 副标题字符串最多可指定 72 个字符 .

  如果指定的字符包括 73 个或更多的字符 , 则汇编器将只承认的前 72 字符有效 . 将一个 2- 字节字符计为两个字符 , 并将制表符计为一个字符 .
- 由 SUBTITLE 控制指令指定的字符串将打印在已经指定 SUBTITLE 控制指令的那页的后一页上 . 但是，如果在一页的顶部 （首行）指定了该控制指令，则将在该页上打印副标题 .
- 如果没有指定 SUNBTITLE 控制指令，则汇编器将空着 SUBTITLE 部分 .
- 如果要使用单引号标志 （'）作为字符串的一部分，则连续写两个单引号标志 .
- 如果 SUBTITLE 部分的字符串是 0, 则空着 SUNTITLE 列 .
- 如果在指定的副标题串中发现 **2.2.2 字符集**之外的任何字符，汇编器将输出 "!" 代替 SUBTITLE 列中的非法字符 . 如果描述了 CR （0DH）字符，则会导致错误且汇编列表里没有内容输出 . 如果描述了 00H 字符，则从 00H 到结束单引号 （'）之间的任何内容都不会输出 .

[应用示例]

```
        NAME    SAMP
        CSEG
$       SUBTITLE ( ' THIS IS SUBTITLE 1 ' )     ; (1)
$       EJECT                                   ; (2)
        CSEG
$       SUBTITLE ( ' THIS IS SUBTITLE 2 ' )     ; (3)
$       EJECT                                   ; (4)
$       SUBTITLE ( ' THIS IS SUBTITLE 3 ' )     ; (5)
        END
```

< 说明 >

(1) 该控制指令指定字符串 "THIS IS SUBTITLE 1".

(2) 该控制指令指定一个换页操作 .

(3) 该控制指令指定字符串 "THIS IS SUBTITLE 2".

(4) 该控制指令指定一个换页操作 .

(5) 该控制指令指定字符串 "THIS IS SUBTITLE 3".

该示例的汇编列表如下所示 (每页指定行数为 80).

< 汇编列表 >

```
78K/0S Series Assembler Vx.xx Date : xx xxx xxxx  Page : 1

Command :       -c9026 sample.asm
Para-file :
In-file :       SAMPLE.ASM
Obj-file :      SAMPLE.REL
Prn-file :      SAMPLE.PRN

        Assemble list

ALNO  STNO  ADRS  OBJECT  M I SOURCE STATEMENT

 1     1                        NAME SAMP
 2     2    ----                CSEG
 3     3                     $  SUBTITLE ( ' THIS IS SUBTITLE 1 ' ) ; (1)
 4     4                     $  EJECT                               ; (2)
------------------------------------------------ age ejection ---------------
78K/0S Series Assembler Vx.xx Date:xx xxx xxxx  Page: 2

THIS IS SUBTITLE 1

ALNO  STNO  ADRS  OBJECT  M I SOURCE STATEMENT

 5     5    ----                CSEG
 6     6                     $  SUBTITLE ( ' THIS IS SUBTITLE 2 ' ) ; (3)
 7     7                     $  EJECT                               ; (4)
------------------------------------------------ age ejection ---------------
78K/0S Series Assembler Vx.xx Date:xx xxx xxxx  Page: 3

THIS IS SUBTITLE 2

ALNO  STNO  ADRS  OBJECT  M I SOURCE STATEMENT

 8     8                     $  SUBTITLE ( ' THIS IS SUBTITLE 3 ' ) ; (5)
 9     9                        END

Target chip : uPD789026
Device file : Vx.xx
Assembly complete , 0 error ( s ) and 0 warning ( s ) found. ( 0 )
```

FORMFEED / NOFORMFEED

(7) FORMFEED / NOFORMFEED (formfeed / noformfeed)

[编写格式]

```
[ Δ ] $ [ Δ ] FORMFEED
[ Δ ] $ [ Δ ] NOFORMFEED          ; 缺省假定
```

[功能]

- FORMFEED 控制指令通知汇编器在汇编列表文件的最后输出一个 FORMFEED 代码 .
- NOFORMFEED 控制指令通知汇编器在汇编列表文件的最后不输出 FORMFEED 代码 .

[用途]

- 在打印了汇编列表文件之后，使用 FORMFEED 控制指令打开一个新页 .

[说明]

- FORMFEED 或 NOFORMFEED 控制指令仅能在源模块文件的开头部分进行描述 .
- 打印汇编列表时，如果在一页的中间指示打印结束，则不打印列表的最后一页 . 在这种情况下，使用 FORMFEED 控制指令或汇编选项 （-lf ）在汇编列表的最后增加一个 FORMFEED 代码 .

  很多情况下，FORMFEED 代码会在文件末尾输出 . 因此 , 如果在文件末尾存在 FORMFEED 代码 , 则会弹出一张不多余的空白页 . 为了防止这种情况 , 将 NOFORMFEED 控制指令或汇编选项 (-nlf) 设置为缺省值 .
- 如果同时指定了两个或两个以上的 FORMFEED / NOFORMFEED 控制指令，则最后指定的控制指令变为有效 .
- 也可通过位于汇编程序起始命令行的汇编选项 (-lf) 或 (-nlf)，指定输出或不输出进纸代码 .
- 如果源模块中的控制指令定义 （FORMFEED/NOFORMFEED）与在起始命令行中的指定 （-lf / -nlf）不同，则起始命令行中的指定优先于在源模块中的定义 .
- 即使在起始命令行定义了汇编选项 (-np), 汇编器也会对 FORMFEED 或 NOFORMFEED 控制指令执行语法检查 .

WIDTH

(8) WIDTH (width)

[编写格式]

```
[ Δ ] $ [ Δ ] WIDTH [ Δ ] ( [ Δ ] columns - per - line [ Δ ] )
```

[缺省假定]

$WIDTH (132)

[功能]

- WIDTH 控制指令指明列表文件每一行的列（字符）数 . "columns-per-line" 必须是 72 至 260 范围内的一个值 .

[用途]

- 使用 WIDTH 控制指令改变列表文件每行的列数 .

[说明]

- WIDTH 控制指令仅能在源模块文件的开头部分进行描述 .
- 如果同时指定了两个或两个以上的 WIDTH 控制指令，则最后指定的控制指令变为有效 .
- 也可通过位于汇编器起始命令行的汇编选项 （-lw）来指定列表文件每行的列数 .
- 如果在源模块文件中定义的控制指令 (WIDTH) 与在起始命令行中的定义 (-lw) 不同，则在命令行中的定义优先于在源模块中的定义 .
- 即使在起始命令行定义了汇编选项（ -np ），汇编器也会对 WIDTH 控制指令执行语法检查 .

LENGTH

(9) LENGTH (length)

[编写格式]

```
[ Δ ] $ [ Δ ] LENGTH [ Δ ] ( [ Δ ] lines - per - page [ Δ ] )
```

[缺省假定]

$LENGTH (66)

[功能]

- LENGTH 控制指令定义列表文件每页的行数 . "lines-per-page" 可能是 "0" 或 20 至 32767 范围内的一个值 .

[用途]

- 使用 LENGTN 控制指令改变列表文件每页的行数 .

[说明]

- LENGTH 控制指令仅能在源模块文件的开头部分进行描述 .
- 如果同时指定了两个或两个以上的 LENGTH 控制指令 , 则最后指定的控制指令将变为有效 .
- 也可通过位于汇编器起始命令行的汇编选项 （-ll）来指定列表文件每行的列数 .
- 如果在源模块文件中控制指令的定义 (LENGTH) 与在起始命令行中的定义 (-ll) 不同 , 则在命令行中的定义优先于在源模块中的定义 .
- 即使在起始命令行定义了汇编选项 (-np), 汇编器也会对 LENGTH 控制指令执行语法检查 .

TAB

(10) TAB (tab)

[编写格式]

```
[ Δ ] $ [ Δ ] TAB [ Δ ] ( [ Δ ] number - of - columns [ Δ ] )
```

[缺省假定]

$TAB (8)

[功能]

- TAB 控制指令规定列表文件中 tab 键一跳的列数 . "number-of-columns" 可为 0 至 8 范围内的一个值 .
- TAB 控制指令指定 tab 键一跳的列数 , 通过用几个空格字符代替源模块中的 HT （Horizontal Tabulation）码 , 该列数成为输出任何列表的表格处理的基础 .

[用途]

- 当使用 TAB 控制指令减少了任何列表每行的字符个数时 , 使用 HT 码减少空格数量 .

[说明]

- TAB 控制指令仅能在源模块文件的开头部分进行描述 .
- 如果同时指定了两个或两个以上的 TAB 控制指令，则仅最后指定的控制指令变为有效 .
- 也可通过位于汇编器起始命令行的汇编选项 （-lt）来指定 tab 键一跳的列数 .
- 如果在源模块文件中控制指令的定义 (TAB) 与在起始命令行中的定义 (-lt) 不同 , 则在命令行中的定义优先于在源模块中的定义 .
- 即使在起始命令行定义了汇编选项 (-np), 汇编器也会对 TAB 控制指令执行语法检查 .

4.7 条件汇编控制指令

条件汇编控制指令用于在源模块中选择一系列语句，通过设置条件汇编转换选择对其进行汇编或不进行汇编．

这些控制指令包括 IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF 控制指令和 SET/RESET 控制指令．

通过有效利用这些控制指令，可以使源模块排除不必要的语句，汇编时可尽量少地改变或不改变源模块．

以下条件汇编控制指令可用：

- IF / _IF / ELSEIF / _ELSEIF / ELSE / ENDIF
- SET / RESET (set / reset)

IF / _IF / ELSEIF / _ELSEIF / ELSE / ENDIF

(1) IF / _IF / ELSEIF / _ELSEIF / ELSE / ENDIF

[编写格式]

```
    [ Δ ] $ [ Δ ] IF [ Δ ] ( [ Δ ] switch - name [ [ Δ ] : [ Δ ] switch - name
] ... [ Δ ] )
or  [ Δ ] $ [ Δ ] _IFDconditional - expression
          :
[ Δ ] $ [ Δ ] ELSEIF [ Δ ] ( [ Δ ] switch - name [ [ Δ ] : [ Δ ] switch - name
] ... [ Δ ] )
or  [ Δ ] $ [ Δ ] _ELSEIFDconditional - expression
          :
[ Δ ] $ [ Δ ] ELSE
          :
[ Δ ] $ [ Δ ] ENDIF
```

[功能]

- 控制指令设置条件限制源程序语句的汇编 .

  在 IF 或 _IF 控制指令和 ENDIF 控制指令之间的源程序语句被有条件汇编 .

- 如果条件表达式的估值或被 IF 或 _IF 控制指令 （也就是 IF 或 _IF 条件）指定的 switch 名为真 （除了 00H） , 源程序中从 IF 或 _IF 条件指令直到出现下一个条件汇编控制指令（ELSEIF/_ELSEIF, ELSE 或 ENDIF）之间的源语句都将被汇编 . 对于后续的汇编处理 , 汇编器将继续处理 ENDIF 控制指令之后的语句 .

  如果 IF 或 _IF 条件为假（00H）, 源程序中从 IF 或 _IF 条件指令直到出现下一个条件汇编控制指令（ELSEIF/_ELSEIF, ELSE, 或 ENDIF）之间的源语句都将不会被汇编 .

- 仅当 ELSEIF 或 _ELSEIF 控制指令之前所描述的所有条件汇编控制指令的条件都不满足时 （例如 , 估值为假）, 才检查 ELSEIF 或 _ELSEIF 控制指令的真假状态 .

  如果条件表达式的估计值或被 ELSEIF 或 _ELSEIF 条件指令 （也就是 ELSEIF 或 _ELSEIF 条件）指定的 switch 名为真 , 源程序中从 ELSEIF 或 _ELSEIF 条件指令直到出现下一个条件汇编控制指令 （ELSEIF/_ELSEIF, ELSE, 或 ENDIF）之间的源语句都将被汇编 . 对于后续的汇编处理 , 汇编器将继续处理 ENDIF 控制指令之后的语句 .

  如果 ELSEIF 或 _ELSEIF 条件为假 , 源程序中从 ELSEIF 或 _ELSEIF 条件指令直到出现下一个条件汇编控制指令 （ELSEIF/_ELSEIF, ELSE 或 ENDIF）之间的源语句都不会被汇编 .

- 如果所有在 ELSE 控制指令之前所描述的 IF/_IF 和 ELSEIF/_ELSEIF 控制指令的条件都得不到满足 （例如 , 所有的 switch 名为假）, 源程序中从 ELSE 条件指令直到出现 ENDIF 条件汇编控制指令之间的源语句都将被汇编 .

-

-

- ENDIF 控制指令通知汇编器终止源语句进行条件汇编 .

[用途]

- 通过这些条件汇编控制指令 , 无须对源程序进行大的修改就可以改变需要汇编的语句 .
- 如果源程序中所描述的调试语句只在程序开发中需要 , 是否需要汇编 （翻译成机器语言）该调试语句可通过设置条件汇编 switch 来指定 .

[说明]

- IF 和 ELSEIF 控制指令用于 switch 名的真 / 假条件判断 , 而 _IF 和 _ELSEIF 控制指令用于条件表达式的真 / 假条件判断 .
  IF/ELSEIF和_IF/_ELSEIF都可以合并起来使用. 换言之, ELSEIF/_ELSEIF可以和IF或_IF及ENDIF成对使用.
- 为条件表达式描述一个绝对表达式 .
- 描述 switch 名的规则同符号描述惯例相同（详情 参见 **2.2.3 符号域**）. 但是 , 可作为 switch 名被识别的最大字符数总是 31.
- 如果用 IF 或 ELSEIF 控制指令指定了两个或两个以上的 switch 名 , 每个 switch 名用分号 （;）分开 .). 每个模块最多可以使用 5 个 switch 名 .
- 当两个或两个以上的 switch 名用 IF 或 ELSEIF 控制指令指定时 , 如果一个 switch 名的值为真 , 则判断满足 IF 或 ELSEIF 条件 .
- 将由 IF 或 ELSEIF 控制指令指定的每个 switch 名的值必须使用 SET 或 RESET （set/reset）控制指令定义 . 因此 , 如果源模块中 , 由 IF 或 ELSEIF 控制指令指定的 switch 名的值没有提前使用 SET 或 RESET 控制指令设置 , 则假定值被复位 .
- 如果指定的 switch 名或条件表达式包含非法描述 , 汇编器将输出一个错误信息 , 并确定计算值为假 .
- 描述 IF 或 _IF 控制指令时 , IF 或 _IF 控制指令必须总是与 ENDIF 控制指令成对出现 .
- 如果在宏程序体内描述了 IF_ENDIF 块 , 而且已由 EXITM 处理将控制从宏里返回 , 汇编器将强迫 IF 级返回宏程序体入口的那一级 . 这种情况 下 , 不会导致错误 .
- 在一个 IF_ENDIF 块内描述另一个 IF_ENDIF 块称为 IF 控制指令的嵌套 . IF 控制指令的嵌套最多允许 8 级 .
- 在条件汇编中 , 对没有汇编的语句不产生目标代码 , 但这些语句会没有改变地输出到汇编列表里 . 如果不想输出这些语句 , 则使用 $NOCOND 控制指令 .

[应用示例]

< 示例 1 >

```
        text0
$       IF ( SW1 )              ; (1)
                text1
$       ENDIF                   ; (2)
            :
        END
```

< 说明 >

(1) 如果 switch 名 “SW1” 的值为真，则汇编 “text1” 中的语句 .

如果 switch 名 “SW1” 的值为假，则不汇编 “text1” 中的语句 .

switch 名 "SW1" 的值已由在 "text0" 中描述的 SET 或 RESET 控制指令设为真或假 .

(2) 该指令表示条件汇编源语句范围的结束 .

< 示例 2 >

```
        text0
$       IF ( SW1 )              ; (1)
                text1
$       ELSE                    ; (2)
                text2
$       ENDIF                   ; (3)
            :
        END
```

< 说明 >

(1) switch 名 “SW1” 的值已由在 “text0” 中描述的 SET 或 RESET 控制指令设为真或假 .

如果 switch 名 “SW1” 的值为真 , 将汇编 “text1” 中的语句 , 而不汇编 “text2” 中的语句 .

(2) 如果 （1）中 switch 名 “SW1” 的值为假 , 则不将汇编 “text1” 中的语句 , 而汇编 “text2” 中的语句 .

(3) 该指令表示条件汇编源语句范围的结束 .

< 示例 3 >

```
        text0
$       IF ( SW1 : SW2 )        ; (1)
                text1
$       ELSEIF ( SW3 )          ; (2)
                text2
$       ELSEIF ( SW4 )          ; (3)
                text3
$       ELSE                    ; (4)
                text4
$       ENDIF                   ; (5)
            :
        END
```

< 说明 >

(1) switch 名 “SW1” 、 “SW2”, 和 “SW3” 的值已由在 “text0” 中描述的 SET 或 RESET 控制指令设为真或假 . 如果switch名 “SW1”或“SW2”的值为真,将汇编“text1”中的语句,而不汇编“text2” “text3”,和 “text4”中的语句. 如果 switch 名 “SW1” 和 “SW2” 的值为假 , 则不汇编 “text1” 中的语句 , 而（2）之后的语句将被有条件汇编 .

(2) 如果 （1）中 switch 名 “SW1” 和 “SW2” 的值为假 ,switch 名 “SW3” 的值为真 , 则将汇编 “text2” 中的语句 , 而不汇编 “text1”, “text3”, 和 “text4” 中的语句 .

(3) 如果 （1）中 switch 名 “SW1” 和 “SW2” 的值以及 (2) 中 “SW3” 的值为假 , 而 switch 名 “SW4” 的值为真 , 则将汇编 “text3” 中的语句 , 而不汇编 “text1”, “text2”, 和 “text4” 中的语句 .

(4) 如果 （1）中 switch 名 “SW1” 和 “SW2” 的值、 (2) 中 “SW3” 的值以及 （3）中 “SW4” 的值都为假 , 则将汇编 “text4” 中的语句 , 而不汇编 “text1”, “text2”, 和 “text3” 中的语句 .

(5) 该指令表示条件汇编源语句范围的结束 .

< 示例 4 >

```
        text0
$       _IF ( SYMA )            ; (1)
                text1
$       _ELSEIF ( SYMB = SYMC ) ; (2)
                text2
$       ENDIF                   ; (3)
            :
        END
```

< 说明 >

(1) switch 名 “SYMA” 的值已由在 “text0” 中由 EQU 或 SET 控制指令定义 . 如果符号名 “SYMA” 为真 （非 0） , 则汇编 “text1” 中的语句 , 不汇编 “text2” 中的语句 .

(2) 如果符号名 “SYMA” 为 0, 而 “SYMB” 和 “SYMC” 值相同 , 则汇编 “text2” 中的语句 .

(3) 该指令表示条件汇编源语句范围的结束 .

SET / RESET

(2) SET / RESET (set / reset)

[编写格式]

```
[ Δ ] $ [ Δ ] SET [ Δ ] ( [ Δ ] switch - name [ [ Δ ] : [ Δ ] switch - name ]
... [ Δ ] )
[ Δ ] $ [ Δ ] RESET [ Δ ] ( [ Δ ] switch - name [ [ Δ ] : [ Δ ] switch - name
] ... [ Δ ] )
```

[功能]

- SET 或 RESET 对每个将被 IF 或 ELSEIF 控制指令指定的 switch 名赋值 .
- SET 控制指令对每个在其操作数中指定的 switch 名赋一个真值 （0FFH）.
- RESET 控制指令对每个在其操作数中指定的 switch 名赋一个假值 （00H）.

[用途]

- 描述 SET 控制指令对每个将被 IF 或 ELSEIF 控制指令指定的 switch 名赋一个真值 （0FFH）.
- 使用 RESET 控制指令对每个将被 IF 或 ELSEIF 控制指令指定的 switch 名赋一个假值 （00H）.

[说明]

- 用 SET 或 RESET 控制指令描述至少一个 switch 名 .

  描述 switch 名的规则同符号描述惯例相同 （参见 **2.2.3 符号域**）. 但是 , 可作为 switch 名被识别的最大字符数总是 31.
- 除了保留字和其他 switch 名 , 指定的 switch 名可以与用户定义的符号相同 .
- 如果用 SET 或 RESET 控制指令指定了两个或两个以上的 switch 名 , 每个 switch 名用分号（;）分开 . 每个模块最多可以使用 1000 个 switch 名 .
- 一个已经用 SET 控制指令设为 “ 真 ” 的 switch 名 , 可用 RESET 控制指令改为 “ 假 ”. 反之亦然 .
- 在源模块中 , 对于将要由 IF 或 ELSEIF 控制指令指定的 switch 名 , 必须用 SET 或 RESET 指令在描述 IF 或 ELSEIF 控制指令之前定义一次 .
- switch 名不能输出到交叉引用表中 .

[应用示例]

```
$       SET ( SW1 )             ; (1)
            :
$       IF ( SW1 )              ; (2)
                text1
$       ENDIF                   ; (3)
            :
$       RESET ( SW1 : SW2 )     ; (4)
            :
$       IF ( SW1 )              ; (5)
                text2
$       ELSEIF ( SW2 )          ; (6)
                text3
$       ELSE                    ; (7)
                text4
$       ENDIF                   ; (8)
            :
        END
```

< 说明 >

(1) 该指令给 switch 名 “SW1” 赋一个真值 (0FFH).

(2) 由于已经在上面的 （1）中给 switch 名 “SW1” 赋了真值 , 则将执行 “text1” 中的语句 .

(3) 该语句表示从 （2）开始的条件汇编源语句范围的结束 .

(4) 该指令分别给 switch 名 “SW1” 和 “SW2” 赋一个假值 (00H).

(5) 由于在上面的 （4）中给 switch 名 “SW1” 赋值为假 , 则不汇编 “text2” 中的语句 .

(6) 由于也在上面的 （4）中给 switch 名 “SW2” 赋值为假 , 则也不汇编 “text3” 中的语句 .

(7) 由于在上面的 （5）和 （6) 中给 switch 名 “SW1” 和 “SW2” 赋值为假 , 则汇编 “text4” 中的语句 .

(8) 该指令表示从 （5）开始的条件汇编源语句范围的结束 .

4.8 其它控制指令

以下控制指令是诸如 C 编译器和结构化汇编预处理器等这样的高级程序输出的专用控制指令：

- $TOL_INF
- $DGS
- $DGL

第五章　　宏

本章说明如何使用宏功能 . 当在源程序中重复描述一系列的语句时，宏是非常有用的功能 .

5.1　概述

当必须在源程序中描述一系列或一组指令时，宏功能对于程序描述非常有用 . 宏功能指把通过 MACRO 和 ENDM 伪指令作为宏程序体定义的一系列语句 （一个指令组）在宏名称被引用处展开 .

宏用于提高源程序的编码效率，并区别于子程序 .

宏和子程序具有不同的特征，如下所示 . 为了高效的利用，需根据特定用途选择宏或子程序 .

(1) 子程序

- 把一个必须在程序中重复多次的处理过程看作一个独立的子程序 . 子程序将通过汇编器转换成机器语言，但仅可转换一次 .
- 要调用子程序，仅需描述一个子程序调用指令即可（通常，设置参数的指令可在子程序之前或之后描述）. 有效使用子程序可提高内存的使用效率 .
- 通过把程序中的一系列处理进程编成子程序，可使程序结构化（这种结构化使程序员容易理解程序的整个结构，使程序设计变得容易）.

(2) 宏

- 宏的基本功能是用一个名字代替一组指令 .

  采用 MACRO 和 ENDM 伪指令作为宏程序体定义的一系列（或一组）指令将在宏名称被引用处被展开 .
- 当汇编器发现一个宏引用时，汇编器展开该宏程序体，并将这组指令转换成机器语言，同时，在宏引用时用实际参量代替宏程序体内的形式参量 .
- 可对宏描述一些参量 .

  例如，如果有些指令组，其处理过程相同而操作数中所描述的数据不同，则可通过将形式参量分配给数据来定义一个宏 . 在宏引用时通过描述宏名和实际参量，汇编器可处理多种仅在语句描述上不同的指令组 .

使用子程序的编程技术主要用来减少内存空间并使程序结构化，而宏用于提高程序的编码效率 .

5.2 宏的使用

5.2.1 宏定义

使用 MACRO 和 ENDM 伪指令来定义宏 .

[编写格式]

符号字段	助记词字段	操作字段	注释字段
宏名称	MACRO : ENDM	[[形式参量 [, ...]]]	[; 注释] [; 注释]

[功能]

- MACRO 伪指令通过把在符号字段指定的宏名称分配给在该伪指令和 ENDM 伪指令之间所描述的一系列语句（称为宏程序体）来执行一个宏定义 .

[应用示例]

```
ADMAC   MACRO   PARA1 , PARA2
        MOV     A , #PARA1
        ADD     A , #PARA2
        ENDM
```

< 说明 >

以上的示例说明了一个简单的宏定义，即，将 “PARA1” 和 “PARA2” 的值相加，结果存于寄存器 A 中 . 给定宏名称为 “ADMAC”, “PARA1” 和 “PARA2” 是形式参量 .

详情参见 "3.8 Macro 伪指令 ".

5.2.2 宏引用

要调用宏，必须在源程序的记忆字段描述该已定义的宏名称 .

[编写格式]

符号字段	助记词字段	操作字段	注释字段
[标记 :]	宏名称	[[实际参量 [, ...]]	[; 注释]

[功能]

- 该语句调用分配给在记忆字段指定的宏名称的宏程序体 .

[用途]

- 使用该语句描述来调用宏程序体 .

[说明]

- 要在记忆字段指定的宏名称必须在宏引用之前已经定义 .
- 每行最多可指定 16 个实际参量，各实际参量之间用逗号（,）分开 .
- 组成实际参量的字符串中不能包含空格 .
- 当在实际参量中描述逗号（,）、分号（;）、空格或 tab 键时，用一对单引号标志把包含这些特殊字符的字符串括起来 .
- 形式参量按照从左到右的顺序用相应的实际参量代替 .
  如果形式参量的数量不等于实际参量的数量，则输出告警信息 .

[应用示例]

```
        NAME    SAMPLE
ADMAC   MACRO   PARA1 , PARA2
                MOV     A , #PARA1
                ADD     A , #PARA2
        ENDM

        CSEG
            :
        ADMAC   10H , 20H
            :
        END
```

< 说明 >

宏引用调用已定义的宏名称 “ADMAC”. 10H 和 20H 是实际参量 .

5.2.3 宏展开

汇编器按照如下方式处理宏：

- 在宏名称被引用处，汇编器展开与该被引用宏名称相对应的宏程序体 .
- 汇编器采用与汇编其它语句同样的方式汇编被展开的宏程序体 .

5.2.4 应用示例

当 "5.2.2 宏引用 " 中引用的宏被汇编时，宏程序体将被展开，如下所示 .

```
        NAME    SAMPLE

        ; 宏定义
ADMAC   MACRO   PARA1 , PARA2
                MOV     A , #PARA1
                ADD     A , #PARA2
        ENDM

        ; 源文本
        CSEG
            :

        ; 宏展开
        ADMAC   10H , 20H               ; (1)
        MOV     A , #10H
        ADD     A , #20H

        ; 源文本
            :
        END
```

< 说明 >

(1) 通过 (1) 中的宏引用，宏程序体被展开 . 宏程序体内的形式参量将被实际参量代替 .

5.3 宏内的符号

可在宏内定义的符号分为两类：全局符号和局部符号 .

(1) 全局符号

- 全局符号是可以从源程序内的任何语句被引用的符号 .
  因此，如果定义了该全局符号的宏被引用多次以展开一系列语句时，该符号将产生重定义错误 .
- 没有用 LOCAL 伪指令定义的符号为全局符号 .

(2) 局部符号

- 局部符号是用 LOCAL 伪指令定义的符号（参见 "3.8 宏伪指令 "）.
- 局部符号可在用 LOCAL 伪指令声明为 LOCAL 的宏内被引用 .
- 在宏之外不能引用局部符号 .

[应用示例]

< 源程序 >

```
        NAME    SAMPLE
            ; 宏定义
MAC1    MACRO
                LOCAL   LLAB    ; (1)
LLAB :
            :
GLAB :
                BR      LLAB    ; (2)
                BR      GLAB    ; (3)
        ENDM
            :
            ; 源文本
REF1 :  MAC1                    ; (4) <-- 宏引用
            :
        BR      LLAB            ; (5) <-- 该描述错误 .
        BR      GLAB            ; (6)
            :
REF2 :  MAC1                    ; (7) <-- 宏引用
            :
        END
```

< 说明 >

(1) 该 LOCAL 伪指令定义标记 "LLAB" 为局部符号 .

(2) BR 伪指令引用宏 "MAC1" 里的局部符号 "LLAB".

(3) BR 指令引用宏 "MAC1" 里的全局符号 "GLAB" .

(4) 该语句引用宏 "MAC1".

(5) BR 伪指令在宏 "MAC1" 定义之外引用局部符号 "LLAB". 当汇编源程序时该描述会产生错误 .

(6) BR 伪指令在宏 "MAC1" 定义之外引用全局符号 "GLAB".

(7) 该语句引用宏 “MAC1”. 同一宏被引用了两次 .

当汇编上述示例中的源程序时，宏程序体将被展开成如下形式 .

< 汇编列表 >

```
        NAME    SAMPLE
              :
REF1 :  MAC1
              ; 宏展开
??RA0000 :
              :
GLAB :                              <-- 错误
        BR      ??RA0000
        BR      GLAB
              ; 源文本
              :
        BR      !LLAB               <-- 错误
        BR      !GLAB
              :
REF2 :  MAC1
              ; 宏展开
??RA0001 :
              :
GLAB :                              <-- 错误
        BR      ??RA0001
        BR      GLAB
              ; 源文本
              :
        END
```

< 说明 >

在宏 "MAC1" 内已定义了全局符号 "GLAB". 由于宏 "MAC1" 被引用了两次，当再次在宏程序体内展开一系列语句时，全局符号 "GLAB" 产生一个重定义错误 .

5.4 宏操作符

宏操作符有两种类型：'& (和号)' 和 '" (单引号)".

(1) & (联结)

- 和号 "&" 将宏程序体内的字符串彼此联结起来 . 宏展开时 , 和标记左侧的字符串与该标记右侧的字符串联结起来 . 连接字符串操作之后 , "&" 本身不再出现 .
- 宏定义时 , 一个符号里 "&" 前后的字符串可视为形式参量或 LOCAL 符号 . 宏展开时 , "&" 前后的形式参量或 LOCAL 符号被看作是一个符号 , 且可连接成一个符号 .
- 括在一对单引号标记之内的 "&" 可作为数据进行处理 .
- 两个连续 "&" 符号当作一个单独的 "&" 符号进行处理 .

[应用示例]

< 宏定义 >

```
MAC     MACRO   P
LAB&P :                         <-- 形式参量 'P' 被识别 .
                D&B     10H
                DB      ' P '
                DB      P
                DB      ' &P '
        ENDM
```

< 宏引用 >

```
        MAC     1H
LAB1H :
        DB      10H     <-- 'D' 和 'B' 联结变为 'DB'.
        DB      ' P '
        DB      1H
        DB      ' &P '  <-- 括在一对单引号之间的 & 作为数据处理 .
```

(2) '(单引号标记)

- 如果一个被一对单引号括起来的字符串在宏引用行或 IRP 伪指令或分割字符之后的实际参量的开头被描述，该字符串将被解释为一个实际参量．该字符串将传递给实际参量，且不包括单引号标记．
- 如果在宏程序体内存在一个被一对单引号括起来的字符串，则字符串被当作数据进行处理．
- 要使用单引号标记作为文本里的单引号标记，连续两次书写该单引号标记．

[应用示例]

```
        NAME    SAMP
MAC1    MACRO   P
                IRP     Q , < P >
                        MOV     A , #Q
                ENDM
        ENDM

        MAC1    ' 10 , 20 , 30 '
```

当上述示例中的源程序被汇编时，宏 “MAC1” 将被展开如下形式．

< 汇编列表 >

```
IRP     Z , < 10 , 20 , 30 >
        MOV     A , #Q
ENDM
MOV     A , #10                 ; IRP 展开
MOV     A , #20                 ; IRP 展开
MOV     A , #30                 ; IRP 展开
```

第六章　产品应用

本章介绍一些推荐的有效利用 RA 78K0S 汇编包的方法 .

6.1　启动汇编器时节省时间并减少故障

有些控制指令具有与汇编选项相同的功能，且必须总是在启动汇编器时使用；这样的例子包括处理器类型定义（-c）. 建议在源模块文件中描述这样的控制指令 . 特别地，不可省略处理器的定义，应在源模块文件的开头部分通过 PROCESSOR 控制指令对其进行指定 . 这样就避免了每次启动汇编程序时在起始命令行指定汇编选项（-c）的需要 . 记住，如果没有在起始命令行指定汇编选项，则将导致错误，此时，汇编器需要从起始行使用正确的汇编选项重新启动 .

交叉引用表输出控制指令（XREF）也需要在模块的开头被指定 .

< 示例 >

```
$       PROCESSOR ( 9026 )
$       KANJICODE SJIS
$       XRFF

        NAME    TEST

        CSEG
            :
        END
```

6.2 如何开发具有高内存利用率的程序

与其它数据内存区域相比，短直接寻址区是一块可用短字节长度指令访问的区域.

因此，通过有效使用这个区域可开发出具有高内存利用率的程序.

在一个模块内声明该短直接寻址区域．在这种方式下，即使所有打算定位在短直接寻址区域的变量也不能定位于此，且改变起来也很容易，从而只有那些将会被频繁访问的变量位于短直接寻址区.

[应用示例]

< 模块 1 >

```
        PUBLIC  TMP1 , TMP2
WORK    DSEG    AT      0FE20H
TMP1 :  DS      2               ; 字
TMP2 :  DS      1               ; 字节
```

< 模块 2 >

```
        EXTRN   TMP1 , TMP2
SAB     CSEG
        MOVW    TMP1 , #1234H
        MOV     TMP2 , #56H
            :
```

附录 A　　保留字列表

以下六种类型的保留字可用：机器语言指令，伪指令，控制指令，操作符，寄存器名和 sfr 符号．保留字是由汇编器预先保留的字符串，不能用于其它特别目的．

在源程序各自的字段中可描述的保留字类型如下表所示．

表 A-1　保留字的类型

类型	说明
符号字段	该字段中无保留字可描述．
助记词字段	仅机器语言指令和指示可在该字段中描述．
操作字段	仅操作符，sfr 符号和寄存器名可在该字段中描述．
注释字段	所有保留字均可在该字段中描述．

对于 sfr 列表，参见各设备的用户手册．

对于中断请求源列表，参见各设备的用户手册．

对于机器语言指令和寄存器名列表，参见各设备的用户手册．

表 A-2 保留字列表

类型	保留字				
操作符	AND	BITPOS	DATAPOS	EQ (or =)	GE (or >=)
	GT (or >)	HIGH	LE (or <=)	LOW	LT (or <)
	MASK	MOD	NE (or < >)	NOT	OR
	SHL	SHR	XOR		
伪指令	AT	BR	BSEG	CALLT0	CSEG
	DB	DBIT	DS	DSEG	DSPRAM
	DW	END	ENDM	EQU	EXITM
	EXTBIT	EXTRN	FIXED	IHRAM	IRP
	IXRAM	LOCAL	LRAM	MACRO	NAME
	ORG	PUBLIC	REPT	SADDR	SADDRP
	SET	UNIT	UNITP		
控制指令	COND / NOCOND			DEBUG / NODEBUG	
	DEBUGA / NODEBUGA [DG / NODG]			EJECT [EJ]	
	FORMFEED / NOFORMFEED			GEN / NOGEN	
	IF / _IF / ELSEIF / _ELSEIF / ELSE / ENDIF			INCLUDE [IC]	
	LENGTH			LIST / NOLIST [LI / NOLI]	
	PROCESSOR [PC]			SET / RESET	
	SUBTITLE [ST]			SYMLIST / NOSYMLIST	
	TAB			TITLE [TT]	
	WIDTH			XREF / NOXREF [XR / NOXR]	
其它	DGL	DGS	SFR	SFRP	TOL_INF

备注 中括号中的内容在缩写格式的控制指令之后 .

附录 B　　伪指令列表

表 B-1　伪指令列表

伪指令				功能分类	备注
符号字段	助记词字段	操作字段	注释字段		
[段名]	CSEG	[重定位属性]	[; 注释]	声明代码段的开始	
[段名]	DSEG	[重定位属性]	[; 注释]	声明数据段的开始	
[段名]	BSEG	[重定位属性]	[; 注释]	说明位段的开始 .	
[段名]	ORG	绝对表达式	[; 注释]	声明绝对段的开始	禁止操作数内的符号前向引用 .
名称	EQU	表达式	[; 注释]	定义一个名称 .	名称 : 符号 禁止操作数内的符号前向引用或外部引用 .
名称	SET	绝对表达式	[; 注释]	定义一个重定义名	名称 : 符号 禁止操作数内的符号前向引用 .
[标记 :]	DB	(size) 初始值 [, ...]	[; 注释]	初始化或保留一个字节的数据区域	标记 : 符号 字符串可位于初始值的位置 .
[标记 :]	DW	(size) 初始值 [, ...]	[; 注释]	初始化或保留一个字节的数据区域	标记 : 符号
[标记 :]	DS	绝对表达式	[; 注释]	保留字节数据区域	名称 : 符号 禁止操作数内的符号前向引用 .
名称	DBIT	无	[; 注释]	保留位数据区域	名称 : 符号 禁止操作数内的符号前向引用 .

表 B-1　伪指令列表

伪指令				功能分类	备注
符号字段	助记词字段	操作字段	注释字段		
[标记 :]	PUBLIC	符号名 [, ...]	[; 注释]	声明一个外部定义名称 .	
[标记 :]	EXTRN	符号名 [, ...]	[; 注释]	声明一个外部引用名称 .	
[标记 :]	EXTBIT	位符号名称 [, ...]	[; 注释]	声明一个外部引用名称 .	符号名限定在具有比特值的符号名中 .
[标记 :]	NAME	对象模块名	[; 注释]	定义一个模块名称 .	模块名 : 符号
[标记 :]	BR	表达式	[; 注释]	自动选择一个分支指令 .	标记 : 符号
宏名称	MACRO	[形式参量 [, ...]]	[; 注释]	定义一个宏 .	宏名称 : 符号
[标记 :]	LOCAL	符号名 [, ...]	[; 注释]	定义一个仅在宏内部有效的符号 .	仅能在宏定义中使用 .
[标记 :]	REPT	绝对表达式	[; 注释]	在宏展开期间指定重复次数 .	标记 : 符号
[标记 :]	IRP	格式参量 , < 实际参量 [, ...]>	[; 注释]	给形式参量分配一个实际参量 .	标记 : 符号
[标记 :]	EXITM	无	[; 注释]	中断宏扩展 .	仅能在宏定义中使用 .
无	ENDM	无	[; 注释]	终止宏定义 .	仅能在宏定义中使用 .
无	END	无	[; 注释]	表示源模块结束 .	

附录 C 索引

A
绝对汇编器 ... 17
绝对段 ... 24
绝对项 ... 57
ADDRESS 项 ... 35, 60
字母字符 ... 30
?An ... 34
AND 操作符 ... 46
区域保留伪指令 ... 95
汇编器 ... 14
汇编器选项 ... 133
汇编器包 ... 14
汇编语言 ... 15
汇编列表控制指令 ... 145
汇编终止伪指令 ... 130
AT ... 76, 77, 80, 81, 84, 85
自动分支指令选择伪指令 ... 112

B
后向引用 ... 68
二进制常量 ... 37
BIT ... 35
位段 ... 24
位符号 ... 65
BITPOS 操作符 ... 54
BR ... 113
?BSEG ... 34
BSEG ... 35, 83

C
CALLT0 ... 76, 77
字符集 ... 30
字符串常量 ... 37
代码段 ... 24, 75
注释字段 ... 40, 182
联结 ... 178
COND ... 152
条件汇编控制指令 ... 163
条件汇编功能 ... 21
常量 ... 37
控制指令 ... 132
交叉引用列表输出指定控制指令 ... 139
?CSEG ... 34
CSEG ... 35, 75
?CSEGFX ... 34
?CSEGIX ... 34
?CSEGT0 ... 34
?CSEGUP ... 34

D
数据段 ... 24
DATAPOS 操作符 ... 54
DB ... 96
DBIT ... 102
DEBUG ... 137
调试信息输出控制指令 ... 136
DEBUGA ... 138
十进制常量 ... 37
DGL ... 171
DGS ... 171
伪指令 ... 72, 184
DS ... 100
?DSEG ... 34
DSEG ... 35, 79
?DSEGDSP ... 34
?DSEGIH ... 34
?DSEGIX ... 34
?DSEGL ... 34
?DSEGS ... 34
?DSEGSP ... 34
?DSEGUP ... 34
DSPRAM ... 80, 81
DW ... 98

E
EJECT ... 146
ELSE ... 164
ELSEIF ... 164
END ... 131
ENDIF ... 164
ENDM ... 128
EQ 操作符 ... 48
EQU ... 90
EXITM ... 125
表达式 ... 41
外部引用名称 ... 32
外部引用项 ... 57

F
FIXED ... 76, 77
FORMFEED ... 159
前向引用 ... 68

G
GE 操作符 r ... 49
GEN ... 150
通用寄存器 ... 38
通用寄存器对 ... 38
全局符号 ... 176
GT 操作符 ... 48

H
十六进制常量 ... 37
HIGH 操作符 ... 53

I
IF ... 164
IHRAM ... 80, 81
包含控制指令 ... 142
IRP ... 123
IRP-ENDM 块 ... 123
IXRAM ... 76, 80, 81

L
标记 ... 32
LE 操作符 ... 50
LENGTH ... 161
库管理程序 ... 14
连接伪指令 ... 103
连接程序 ... 14
LIST ... 148
列表转换器 ... 14
INCLUDE ... 143
LOCAL ... 118
局部符号 ... 176
LOW 操作符 ... 53
LRAM ... 80, 81
LT 操作符 ... 49

M
机器语言 ... 15
MACRO ... 35, 116
宏 ... 172
宏定义 ... 173
宏伪指令 ... 115
宏展开 ... 175
宏功能 ... 21
宏名称 ... 32
宏操作符 ... 178
宏引用 ... 174
MASK 操作符 ... 55
内存初始化伪指令 ... 95
助记词字段 ... 36, 182
MOD (余数) 操作符 ... 44
模块化程序设计 ... 17
MODULE ... 35
模块体 ... 24
模块头 ... 23
模块名 ... 32
模块尾 ... 24

N
名称 ... 32
NE 操作符 ... 48
NOCOND ... 152
NODEBUG ... 137
NODEBUGA ... 138
NOFORMFEED ... 159
NOGEN ... 150
NOLIST ... 148
NOSYMLIST ... 141
NOT 操作符 ... 46
NOXREF ... 140
NUMBER ... 35
NUMBER 项 ... 60
数字常量 ... 37

O
对象转换器 ... 14
八进制常量 ... 37
操作数 ... 66
操作数字段 ... 36, 182
操作符 ... 41
优化功能 ... 21
OR 操作符 ... 47
操作符的优先顺序 ... 42
ORG ... 87

P
PM+ ... 14
PROCESSOR ... 135
处理器类型定义控制指令 ... 134

R
可重新定位汇编器 ... 17
可重新定位项 ... 57
可重新定位属性 ... 57, 68
REPT ... 121
REPT-ENDM 块 ... 121
RESET ... 169

S
SADDR ... 80, 81
SADDRP ... 80, 81
段名 ... 32
段 ... 24
SET ... 93, 169
SHL (左移) 操作符 ... 52
SHR (右移) 操作符 ... 51
源模块 ... 22, 131
特殊字符 ... 38
专用功能寄存器 ... 38
结构化汇编器预处理器 ... 14
子程序 e ... 172
SUBTITLE ... 156
符号 ... 176
符号属性 ... 35, 68
符号定义伪指令 ... 89
符号字段 ... 32, 182
SYMLIST ... 141

T
TAB ... 162
TITLE ... 154
TOL_INF ... 171

U
UNIT ... 76, 77, 80, 84, 85
UNIT (或无定义) ... 81
UNITP ... 76, 77, 80, 81

W

WIDTH ... 160

X

XOR 操作符 ... 47
XRAM ... 77
XREF ... 140

详细信息请联系：

中国区

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

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

[北京]

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

[上海]

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

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

[深圳]

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

[香港]

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

[成都]

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

[长春]

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

[大连]

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