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

RA78K0 Ver. 3.80

汇编器包

语言

目标设备

78K0 系列

文件编号　U17198CA1V0UM00 (第一版)
印刷日期　2007年09月　CP(K)

[备忘录]

Windows 是微软公司在美国或其他国家中的注册商标或商标。

- **本文档所刊登的内容有效期截至 2007 年 09 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。本文件所登载内容的错误，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。

M5 02.11-1

区域信息

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

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

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

详细信息请联系：

（中国区）

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

[北京]

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

[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

介绍

该手册是为了使用户更容易地、正确地理解**RA78K0汇编器包**（此后称为**RA78K0**）中每个程序的基本功能以及描述源程序的方法而设计的。

该手册没有说明如何操作RA78K0中的各个程序。，因此，在你理解了本手册中的内容后，请阅读**RA78K0汇编器包用户操作手册(U17199E)**（此后称为**操作**）以用来操作汇编器包中的各个程序。

本手册中与RA78K0相关的描述适用于3.80版或更高的版本。

[目标读者]

本手册供理解用于开发的微控制器(78K0系列)中的功能和指令的工程师使用。

[结构]

本手册由以下六个章节和附录组成：

第1章 **概述**

概述RA78K0中的所有基本功能。

第2章 **如何描述源程序**

概述如何描述源程序，并说明了汇编器中的运算符。

第3章 **伪指令**

说明了如何写入以及使用伪指令，并包含有应用示例。

第4章 **控制指令**

说明了如何写入以及使用控制指令，并包含有应用示例。

第5章 **宏**

说明了所有宏功能，包括宏定义，宏引用和宏扩展。

第3章　伪指令说明了宏伪指令。

第6章 **产品应用**

介绍一些用于描述源程序的推荐方法。

附录

这些附录包括了保留字列表，伪指令列表，最大性能，特征以及索引。

本手册没有详细说明指令集。关于这些指令的详细信息请参考用于开发软件的微控制器的用户手册。

同样的，关于结构指令的详细信息请参考用于开发软件的微控制器的用户手册（硬件版）。

[宏]

对于第一次使用汇编器的用户来说，请阅读本手册的**第1章　概述**。而对于那些具备一定汇编器知识的用户来说，则可以跳过本手册的**第1章　概述**。然而，请务必阅读**1.2　程序开发前的提示**以及**第2章　如何描述源程序**。

对于那些希望了解汇编器的伪指令以及控制指令的用户来说，请分别阅读**第3章　伪指令**和**第4章　控制指令**。每个伪指令或控制指令的格式，功能，用途以及应用示例均在这些章节中进行了详细的说明。

[约定]

以下的符号和缩写用于整本手册中：

:	重复同一格式。
[ ]:	可以省略封闭在方括号中的字符。
{ }:	在{ }中的一个项被选择。
“ ”:	封闭在“ ”（双引号）中的字符是一个字符串。
‘ ’:	封闭在‘ ’（单引号）中的字符是一个字符串。
( ):	在圆括号间的字符是一个字符串。
< >:	封闭在这些括号中的字符（主要为标题）是一个字符串。
__:	下划线用于说明一个重要点或用于表示输入字符串。
Δ:	表示一个或更多的空白字符或制表符。
/:	字符分隔符
～:	连续
粗体:	粗体字符用于说明一个重要点或一个引用点。

[相关文件]

与本手册相关的文件（用户手册）如下。

显示在该出版中的相关文件可能会包含初级版本。

然而，初级版本不会标记成这样。

文件名		文件编号
RA78K0 3.80版汇编器包	操作	U17199E
	语言	本手册
	结构汇编语言	U17197E
CC78K0　3.70版C编译器	操作	U17201E
	语言	U17200E
SM +系统模拟器	操作	U17246E
	用户开放接口	U17247E
SM78K0系列2.52版系统模拟器	操作	U16768E
PM+5.20版		U16934E
ID78K0-NS　2.52版综合调试器	操作	U16488E
ID78K0-QB　2.81版综合调试器	操作	U16996E
78K0系列	指令	U12326E

[备忘录]

目录

图形列表

图像编号.　　标题，页

表格列表

表格编号.　　标题 ，页

第 1 章　概述

本章节描述了RA78K0在微控制器软件开发中所起的作用及其特点。

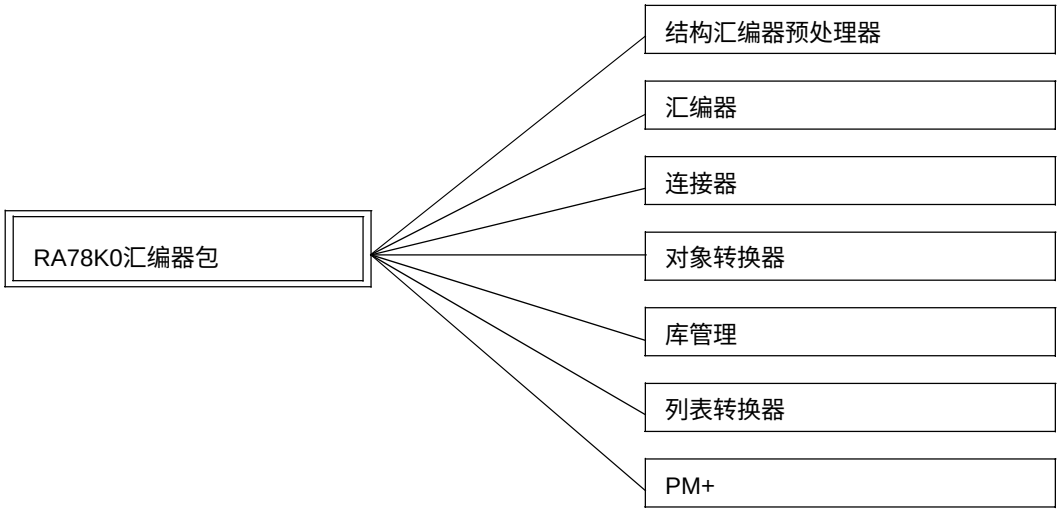
1.1　汇编器概述

RA78K0汇编器包(后面简称"RA78K0")是用于将以78K0系列微控制器的汇编语言来编码的源程序转换为机器语言编码的一系列程序的通用术语。

RA78K0包含六个程序：结构汇编器预处理器，汇编器，连接器，对象转换器，库管理和列表转换器。

另外，RA78K0也提供了PM+，它可以帮助你在Windows中完成对程序的编辑，编译/汇编，连接和调试这一系列操作。

图1-1　RA78K0汇编器包



1.1.1　什么是汇编器？

(1)　汇编语言与机器语言

汇编语言是微控制器最基本的程序语言。

微控制器中的微处理器需要程序和数据来完成它的工作。这些程序和数据必须由用户来写入微控制器的内存中。由微控制器处理的程序和数据是称为机器语言的二进制数的集合。然而，对于用户来说，机器语言编码非常难记，因此会导致频繁地发生错误。幸运的是，已经存在易于被人理解的方法，它通过使用英文缩写或助记符来表示原始机器语言编码的含义。使用这种符号编码方式的基础程序语言系统被称为汇编语言。

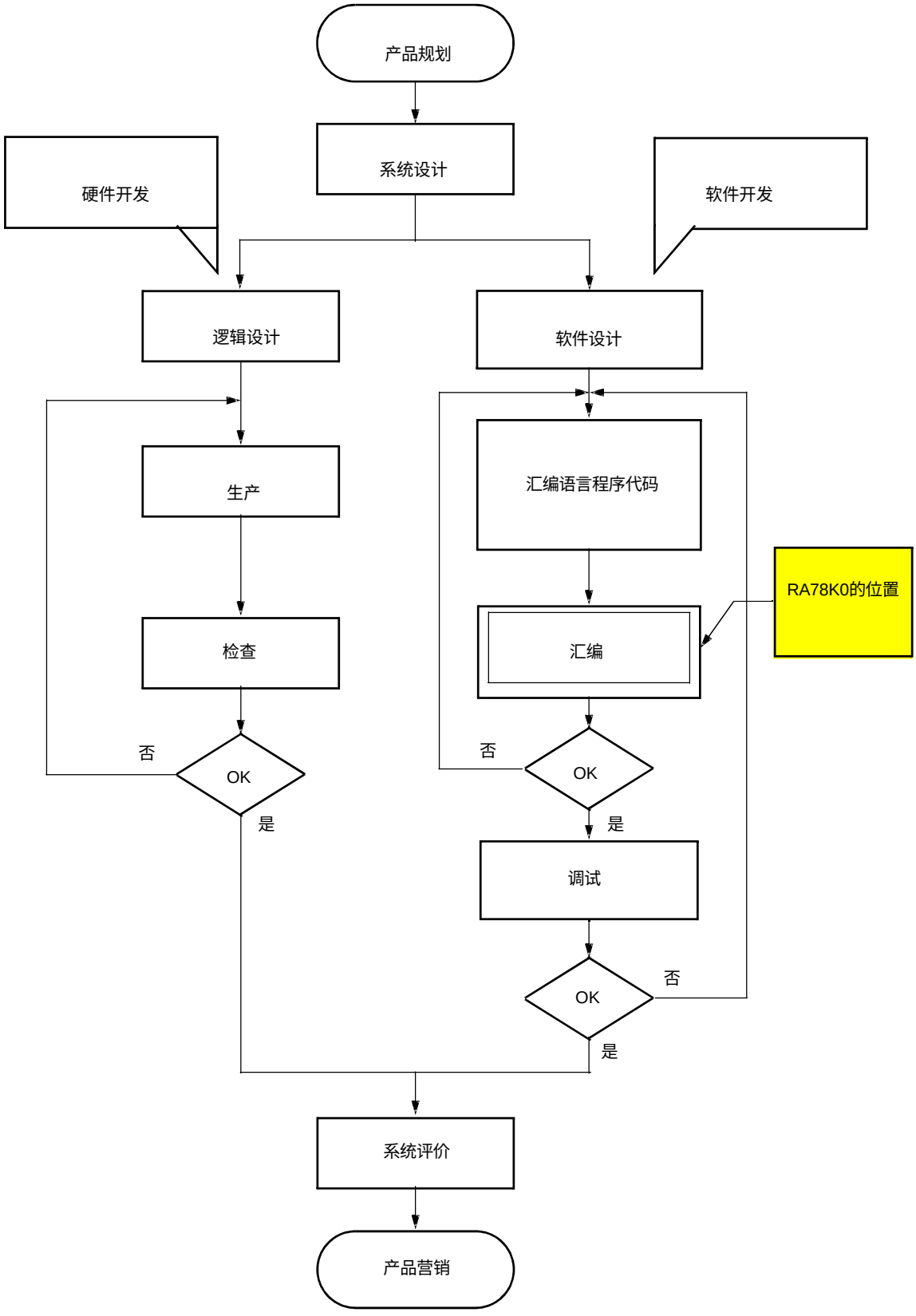
然而，由于机器语言是微控制器可以在其中处理程序的唯一程序语言，因此将需要另一个可以将在汇编语言中创建的程序转换为机器语言的程序。这个程序就称为汇编器。

图1-2　汇编器流程

(2)　应用微控制器产品的开发以及 RA78K0 的作用

图1-3说明了“在产品开发过程中汇编语言程序设计”所处的位置。

图1-3　应用微控制器的产品的开发过程



1.1.2 什么是浮动汇编器？

源语言通过汇编器转换后得到的机器语言在使用前会被写入微控制器的内存中。要完成这个操作，每个机器语言指令在内存中所写入的位置必须已经被确定。

因此，说明每个机器语言指令在内存中所处位置的信息将被添加到由汇编器汇编的机器语言中。

根据将地址置于机器语言指令的方法，汇编器总的来说可以分成绝对汇编器和浮动汇编器。

- **绝对汇编器**

  绝对汇编器将以汇编语言进行汇编的机器语言指令放置到绝对地址中。

- **浮动汇编器**

  在浮动汇编器中，为以汇编语言进行汇编的机器语言指令所确定的地址是暂定的。绝对地址随后将由连接器来确定。

过去，当使用绝对汇编器来创建一个程序时，通常必须同时完成程序设计。然而，如果一个大的程序中所有组成部分被创建为一个唯一的实体，那么程序将变得非常复杂，程序的分析和维护都会变得非常困难。为了避免这种情况，将这类大的程序分成多个子程序来开发，这些子程序称为模块，代表每个功能组件。这种程序设计技术被称为模块化程序设计。

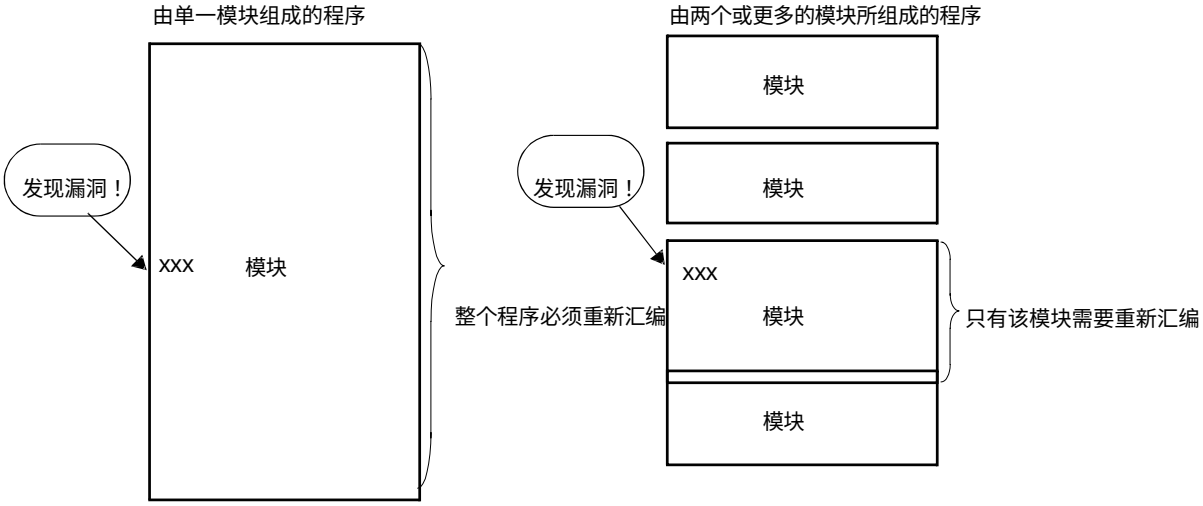
浮动汇编器是一种适合于模块化程序设计的汇编器，它具有以下优势：

(1) 提高开发效率

同时写一个大的程序是非常困难的。在这种情况下，将程序根据每个功能分成多个模块能让两个或更多的程序员来并行开发子程序，同时也提高了开发效率。

此外，如果在程序中发现任何漏洞，将不需要汇编整个程序，而只需改正程序中的一部分，即重新汇编那个必须要改正的模块。这就缩短了调试的时间。

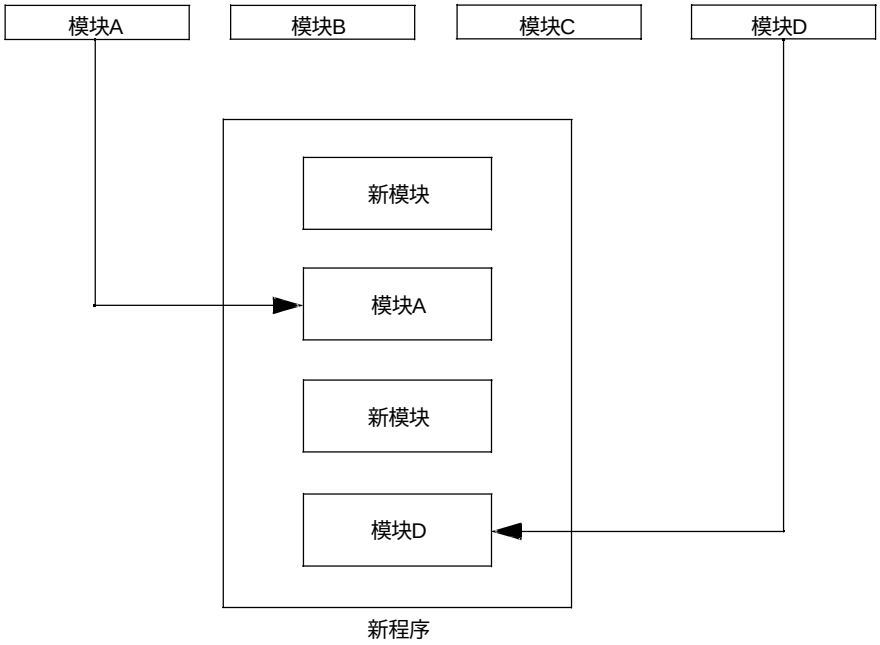
图1-4　为调试重新进行汇编



(2)　资源利用

以前创建的具有高可靠性和通用性的模块可以重新用于另一个程序的创建。如果你将这类高通用性的模块累积为软件资源，你将可以在开发一个新程序的过程中节省时间和人力。

图1-5　使用存在的模块的程序开发



1.2 程序开发前的提示

开始开发程序之前请参考以下内容。

1.2.1 RA78K0的最大使用性能

(1) 汇编器的最大使用性能

表1-1 汇编器的最大使用性能

条目	最大使用性能
符号数量（局部+公共）	65,535 个符号注 1
可以输出交叉引用表的符号的数量	65,534 个符号注 2
用于一个宏引用的宏体的尺寸	1 MB
所有宏体的总尺寸	10 MB
一个文件中的段的数量	256 个段
一个文件中的宏和包含规范	10,000
一个包含文件中的宏和包含规范	10,000
重置数据注 3	65,535 个项目
行号数据	65,535 个项目
一个文件中的BR指令的数量	32,767 个导引
每行中的字符数	2,048个字符注 4
符号长度	256个字符
开关名定义的数量注5	1,000
开关名的字符长度注 5	31个字符
一个文件中包含文件的嵌入层个数	8层

注 1. 使用XMS。如果没有XMS，则使用一个文件。

注 2. 使用内存。如果没有内存，则使用一个文件。

注 3. “浮动数据”是当汇编器不能决定符号值时转移到连接器中的数据。
例如，当使用MOV指令来指示一个外部引用符号时，将会在.rel文件中生成重置数据中的两个条目。

注 4. 这里包含回车代码。如果在一行中描述2049个或更多的字符，那么将输出一个警告消息并忽略等于或大于2049的字符。

注 5. SET/RESET指令将开关名设置为真或假，且开关名同$IF等一起使用。

(2)　连接器的最大使用性能

表1-2　连接器的最大使用性能

项目	最大工作特性
符号数量（局部+公共）	65,535 个符号
同一段的行号数据	65,535个项目
段的数量	65,535 个段
输入模块的数量	1,024 个模块

1.3 RA78K0的特点

RA78K0具有以下特点：

(1) 宏功能

当同一组指令必须在源程序中反复描述时，宏可以通过赋于这一组指令单一的宏名来定义。

通过使用该宏功能，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

RA78K0具有一个自动分支指令选择导引(BR (分支))。

为了创建一个带有高存储器效率的程序，字节分支指令必须依照分支指令的分支目的范围来进行描述。然而，对于程序员来说，描述一个分支指令的同时注意每个分支的目的范围是非常麻烦的。通过描述BR指令，汇编器将会依照分支目的范围生成适当的分支指令。这被称为分支指令的优化功能。

(3) 条件汇编功能

使用该功能时，一部分源程序可以依照预先确定的条件指定为汇编语言或非汇编语言。

如果在源程序中描述调试语句，则可以通过为条件汇编设置一个开关来选择是否将调试语句转换为机器语言。当不再需要调试语句时，源程序可以被汇编且不会对程序进行大的修改。

第 2 章　如何描述源程序

本章节描述了源程序的描述方法，表达式和运算符。

2.1　源程序的基本结构

当通过将源程序分为几个模块来对其进行描述时，成为输入到汇编器中单元的每个模块都将被称为源模块（如果一个源程序由单一的模块组成，那么“源程序”就是“源模块”）。

成为输入到编译程序中的单元的每个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　源模块的结构

2.1.1 模块头

下表2－1中显示的控制指令可以在模块头中进行描述。需注意，这些控制指令只能在模块头中进行描述。

模块头也可以省略。

表2-1　可以在模块头中描述的指令

可以描述的项目	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 - KANJCODE	第4章　控制指令
由高级程序例如 C 语言编译器和结构汇编预处理器输出的特殊控制指令	由高级程序，例如 C 语言编译器和结构汇编预处理器输出的特殊控制指令如下： - TOL_INF - DGS - DGL	

2.1.2 模块体

以下指令不能在模块体中进行描述：

- 与汇编器选项具有相同功能的控制指令

所有其他导引，控制指令以及可以在模块体中进行描述的指令。必须通过将模块体分为称作“段”的单元对其进行描述。

用户可以使用与每个段相应的指令来定义以下四个段：

(1)　代码段

必须以CSEG指令来定义。

(2)　数据段

必须以DSEG指令来定义。

(3)　位段

必须以BSEG指令来定义。

(4)　绝对段

必须通过将字单元地址指定为具有重置属性（AT字单元地址），并以CSEG,DSEG或BSEG指令来定义。该段也可以以ORG指令来定义。

模块体可以以任何段的组合来构成。

然而，在代码段前应该定义一个数据段和一个位段。

2.1.3 模块尾

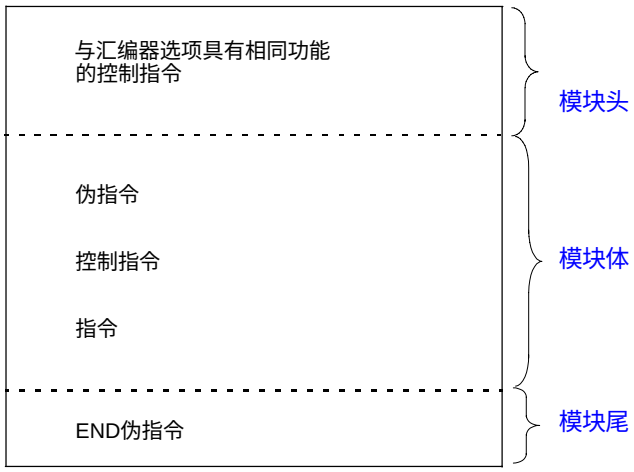
模块尾表明源模块结束。END指令必须在这部分中进行描述。

如果任何不同于注释，空格，制表符或换行符代码的内容接在EDN指令后进行描述，那么编译程序将输出一个警告消息并忽略在END指令后描述的字符。

2.1.4 源程序的总体结构

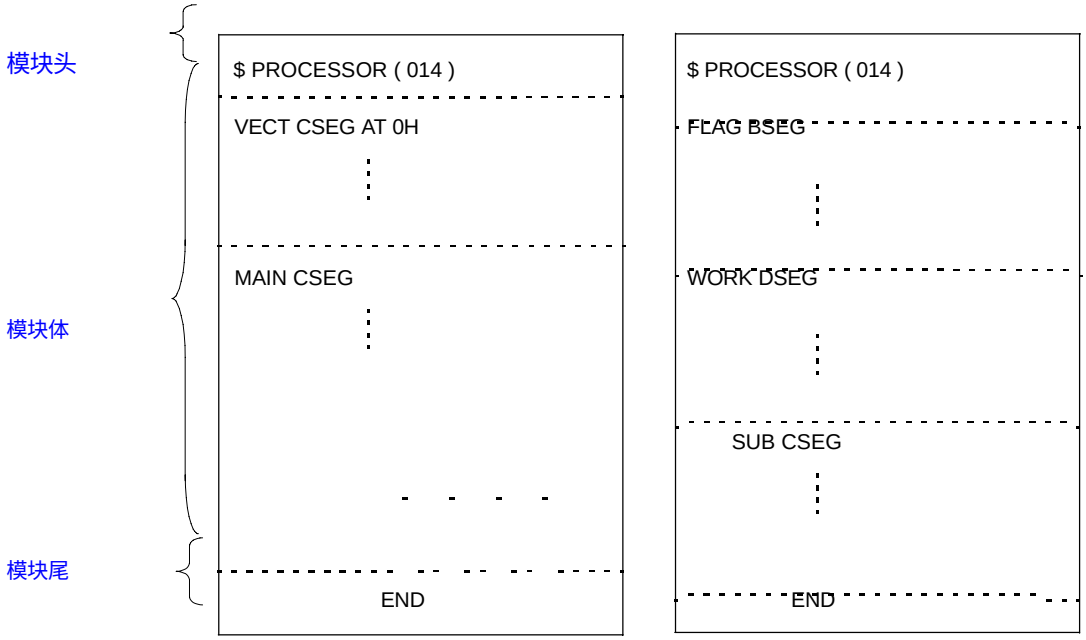
源模块（源程序）的总体结构如下所示。

图2-2　源模块的总体结构



简单的源模块结构示例显示在图2-3中。

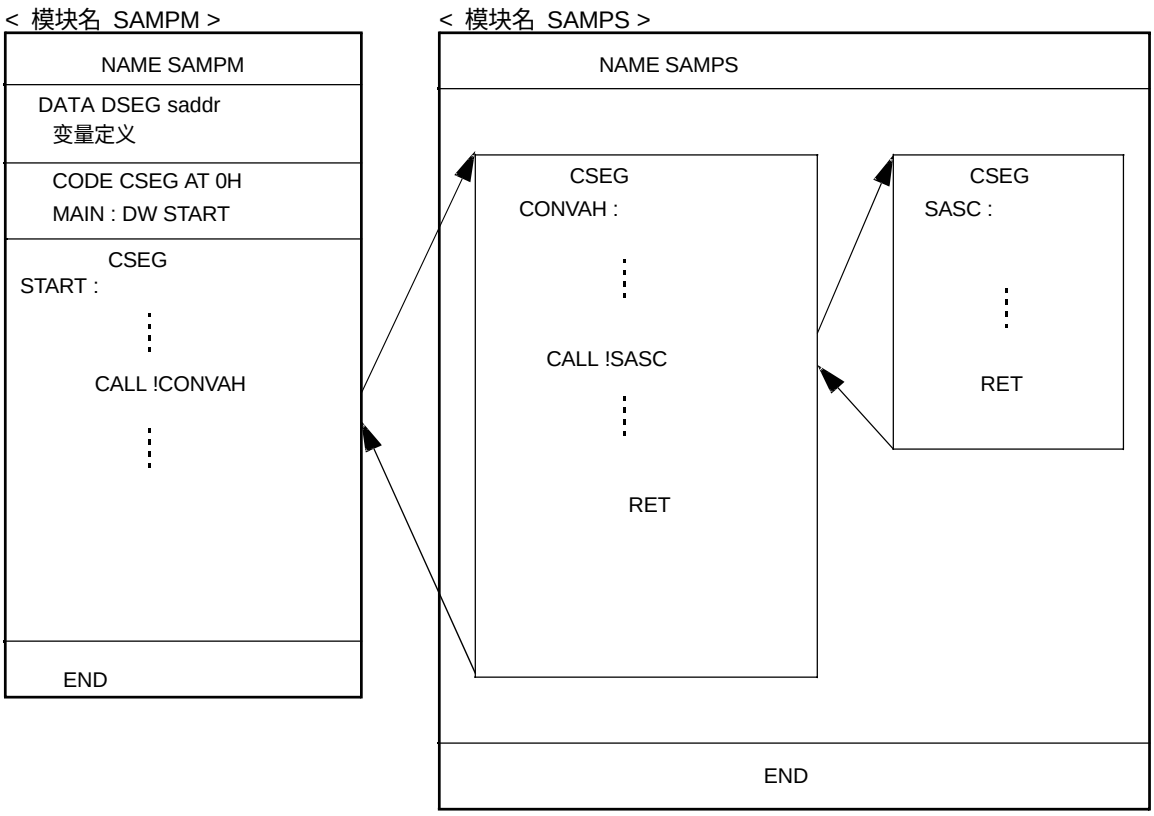
图2-3　源模块结构示例



2.1.5 源程序描述示例

在这小节中，源模块（源程序）的描述示例显示为样例程序。样例程序的结构可以按如下进行简单说明。

图2-4　样例程序的结构



该样例程序通过将单一的源程序分为两个模块来创建。模块“SAMPM”是该程序的主程序，模块“SAMPS”是在主程序中调用的子程序。

<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 :

                                        ; chip initialize
                                                MOVW    SP , #_@STBEG

        MOV     HDTSA , #1AH
        MOVW    HL , #HDTSA             ; set hex 2-code data in HL register

        CALL    !CONVAH                 ; convert ASCII <- HEX
                                        ;  output  BC-register  <-  ASCII  code
        MOVW    DE , #STASC             ;  set  DE  <-  store  ASCII  code  table
        MOV     A , B
        MOV     [ DE ] , A
        INCW    DE
        MOV     A , C
        MOV     [ DE ] , A

        BR      $$

        END                                     ; (8)
```

(1)　模块名的说明

(2)　从另一个模块中引用为一个外部引用符号的符号说明

(3)　在另一个模块中定义为一个外部引用符号的符号说明

(4)　从连接器的”-S”选项中生成为一个外部引用符号的堆栈解决符号说明（如果连接时没有指定”-S”选项，将会产生一个错误）

(5)　数据段开始说明（位于saddr中）

(6)　代码段开始说明（设置为一个从地址OH开始的绝对段）

(7)　代码段开始说明（表示绝对段结束）

(8)　模块结束说明

<子程序>

```
        NAME    SAMPS               ; (1)
; ************************************************
;
;   HEX -> ASCII Conversion Program
;   sub-routine
;
;  input condition          : ( HL ) <- hex 2 code
;  output condition         : BC-register <-ASCII 2 code
;
; ************************************************
        PUBLIC CONVAH               ; (2)
        CSEG
                            ; (3)
CONVAH :
        XOR     A , A
        ROL4    [ HL ]              ; hex upper code load
        CALL    !SASC
        MOV     B , A               ; store result

        XOR     A , A
        ROL4    [ HL ]              ; hex lower code load
        CALL    !SASC
        MOV     C , A               ; store result


  RET

; ************************************************
;     subroutine convert ASCII code
;
;  input Acc ( lower 4bits )    <- hex code
;  output Acc                   <- ASCII code
; ************************************************

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

        END                                         ; (4)
```

(1)　模块名的说明

(2)　从另一个模块中引用为一个外部引用符号的符号说明

(3)　代码段开始说明

(4)　模块结束说明

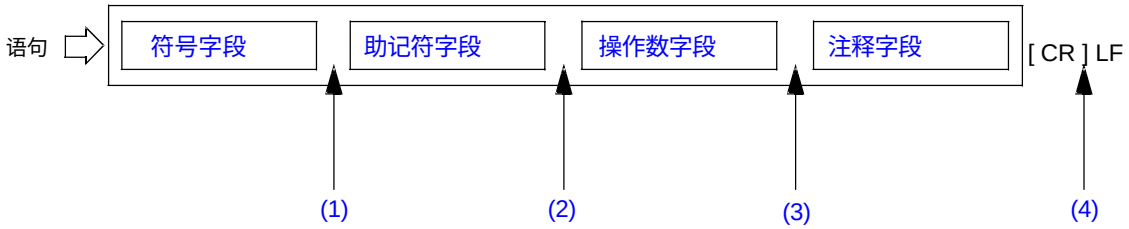
2.2　源程序的描述格式

2.2.1　语句结构

源程序由语句所组成。

每个语句由图2-5中所显示的四个字段组成。

图2-5　组成语句的字段



(1)　符号字段和助记符字段间必须用冒号(:)，或一个或更多的空格或制表符隔开（它根据在助记符字段描述的指令决定来决定是否使用冒号或空格）。

(2)　助记符字段和操作数字段间必须用一个或更多的空格或制表符隔开。根据在助记符字段中所描述的指令，可能不需要操作符字段。

(3)　如果使用注释字段，则必须在前面加上分号(;)。

(4)　每一行必须用一个LF码来定界（在LF码前可能会存在一个CR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或独立的CR都被计作一个字符。如果描述了2049个或更多的字符，将会输出一条警告消息且任何等于或大于2049的字符都会被忽略。然而，2049个或更多的字符将被输出到汇编列表中。

独立的CR不会被输出到汇编列表中。

以下的行也可以被描述：

- 虚拟行（没有语句描述的行）
- 仅由符号字段组成的行
- 仅由注释字段组成的行

2.2.2 字符集

可以在一个源文件中进行描述的字符被分为以下三类：

- 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。它包括字母，数字和特殊字符。

表2-2　字母数字式字符

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 UV 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 uv w x y z

表2-3　特殊字符

字符	名称	主要用途	
? @ _	问号 圈a 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)	 制表码 逗号 冒号 分号 回车码 换行码	 分隔符号 	每个字段的分隔 相当于空格的字符 操作数的分隔 标签的分隔 表示注释字段开始的符号 表示行结束的符号（在汇编器中忽略） 表示行结束的符号
+ - * / . (,) < , > =	加号 减号 星号 斜杠 句点 左，右圆括号 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 位位置说明符 指定要执行的算术运算的顺序的符号 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 - 表示一个完整的宏参数的符号	

表2-3　特殊字符

字符	名称	主要用途
$	美元符号	- 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的开始 - 指定相对寻址的符号
&	和号	连接符号（在宏体中使用）
#	升号	指定立即寻址的符号
!	感叹号	指定绝对寻址的符号
[]	方括号	指定间接寻址的符号

(2)　字符数据

“字符数据”指的是用于描述串常量，字符串和控制指令(TITLE, SUBTITLE, INCLUDE)的字符。

备注　1.　除了”00H”外的所有字符都可以使用（包括kanji（日语字符）；根据不同的操作系统代码可能会不同）。如果”00H”已经被描述，那么将会产生一个错误，并且之后在封闭单引号前的字符将会被忽略。

备注　2.　如果已经描述了任一非法字符，编译程序将会以”!”来代替非法字符用来输出到汇编列表（独立的CR(0DH)码不会被输出到汇编列表中）。

备注　3.　使用Windows时，编译程序将”1AH”码解释为文件结束(EOF)，因此代码不能作为输入数据中的一部分。

(3)　注释字符

“注释字符“指的是用于描述一个注释语句的字符。

备注　可以在注释语句中使用的字符和那些为字符数据设置的字符是相同的。然而，即使已经描述了”00H”码也不会产生错误。相反，编译程序将会通过以”!”来代替非法字符将其输出到汇编列表中。

2.2.3 符号字段

符号是在符号字段中进行描述的。名词“符号”指的是用于数字数据或地址的名称。

通过使用符号，源程序的内容可以变得易于理解。

(1)　符号类型

根据符号的用途和定义方法，符号被分成在表2-4中所显示的类型。

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。 .	该类型在EQU，SET或DBIT指令的符号字段中进行描述。
标签	在源程序中用作地址数据。	该类型通过在符号后使用一个冒号(:)后缀来定义。
外部引用名	通过另一模块来使用由模块定义的符号。	该类型在EXTRN或EXTBIT指令的操作数字段中进行描述。
段名	在连接器运算过程中使用的符号	该类型在CSEG, DSEG, BSEG 或ORG指令的符号字段中定义。
模块名	在符号调试中使用	该类型在NAME指令的操作数字段中进行描述。
宏名	在源程序中用于宏引用	在MACRO指令的符号字段中进行描述。

(2)　符号描述的约定

所有符号必须依照以下规则进行描述：

(a)　符号必须由字母数字字符和可以用作字母字符的特殊字符(?, @, and _)来组成。

数字字符0至9都不能用作符号的首字符。

(b)　符号必须由不多于255个的字符来组成。超过最大符号长度的字符将被忽略。

(c)　保留字不能用作符号。保留字显示在表A-2中。

(d)　同一符号不能被多次定义（然而，以SET指令定义的名称可以用SET指令重新定义）。

(e)　编译程序可以区别小写和大写字符。

(f)　当在符号字须中描述一个标签时，”:”（冒号）必须在标签后立即进行描述。

<正确的符号描述示例>

```
CODE01 CSEG                    “CODE01” 是一个段名。 VAR01
EQU          10H               “VAR01” 是名称。
LAB01 :  DW      0             “LAB01” 是一个标签。
         NAME    SAMPLE        “SAMPLE” 是一个模块名。MAC1
         MACRO                 “MAC1” 是一个宏名。
```

<不正确的符号描述示例>

```
1ABC     EQU     3             ; 数字字符不能用作符号的首字符。

LAB      MOV     A , R0        ; “LAB”是一个标签且必须使用冒号(:)来和助记符字段隔开。

FLAG :   EQU     10H           ; 冒号(:)在名字中不是必需的。
```

<过长符号的示例>

```
A123456789B12 到 Y1234567890123456          EQU     70H
                  250           ; 超过最大符号长度（256个
                                ; 字符）的字符”6”被忽略。
                                ; 符号将会被定义为“A123456789B12 到
                                ; Y123456789012345”
```

<仅由一个符号组成的语句的示例>

```
ABCD :      “ ABCD ” 将会被定义为一个标签。
```

(3)　关于符号的一些注意事项

符号“??RAnnnn (n = 0000 到 FFFF)”是每次在宏体内开发一个局部符号时都将会被汇编器自动替换的符号。须注意不要对该符号定义两次。当段名不是通过段定义指令来指定时，编译程序将会自动生成一个段名。这些段名显示在表2-5中。

复制段名定义会产生一个错误。

表2-5　由编译程序自动生成的段的名称

段名	指令	重置属性
?An (n = 0000 到 FFFF)	ORG伪指令	(无)
?CSEG	CSEG伪指令	UNIT
?CSEGUP		UNITP
?CSEGT0		CALLT0
?CSEGFX		FIXED
?CSEGIX		IXRAM
?CSEGSI		SECUR_ID
?CSEGOB0 to到 4		(无)
?CSEGB0 to到 15		BANK0 to到 15

段名	指令	重置属性
?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

所有名字和标签都有一个值和一个属性。

值就是定义的数字数据或地址数据本身的值。

段名，模块名和宏名不含有值。

符号的属性被称作符号属性且必须是显示在表2-6中的八种类型之一。

表2-6　符号属性和值

属性类型	分类	值
数字	- 被赋于数字常量的名称 - 用EXTRN指令定义的符号 - 数字常量	十进制表示：0 到 65535 十六进制表示：0H　到 FFFFH
地址	- 定义为标签的符号 - 用EQU和SET指令定义为标签的名称	十进制表示 :0 到 1048575 十六进制表示：0H到FFFH
位	- 定义为位值的名称 - BSEG内的名称 - 用EXTBIT指令定义的符号	saddr 区域
CSEG	用CSEG指令定义的段名	这些属性类型不具有值。
DSEG	用DSEG指令定义的段名	
BSEG	用BSEG指令定义的段名	
模块	用NAME指令定义的模块名（如果没有定义的模块名从输入源文件名中的主要名称中创建）。	
宏	用MACRO伪指令定义的宏名	

<例>

```
TEN       EQU     10H                ; 名称"TEN"具有属性"数字"和值"10H"。
ORG       80H
START : MOV       A , #10H           ; 标签"START"具有属性"地址"和值"80H"。
BIT1      EQU     0FE20H.0           ; 名称"BIT1"具有属性"位"和值"0FE20H.0"。
```

2.2.4 助记符字段

助记指令，导引或宏引用在助记符字段中进行描述。

使用指令或需要一个或更多的操作数的导引时，助记符和操作数字段必须用一个或更多的空格或制表符来隔开。然而，使用以"#", "$" ,"!", 或"["开头的指令的第一操作数时，即使在助记符字段和第一操作数字段间不存在任何内容，汇编器仍将正常执行。

<正确描述的示例>

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

<不正确描述的示例>

```
MOVA    #0H                ; 在助记符和操作数字段间不存在空格。
C ALL   !CONVAH            ; 在助记符中存在空格。
ZZZ                        ; 78K0系列没有诸如"ZZZ"的指令。
```

2.2.5 操作数字段

用于执行指令，伪指令或宏引用的数据(操作数)在操作数字段中进行描述。

根据指令或伪指令，在操作数字段中将不需要操作数，而在操作数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，需用逗号(,)为每个操作数来定界。

以下类型的数据可以在操作数字段中进行描述：

- 常量（数字常量和串常量）
- 字符串
- 寄存器名
- 特殊字符($, # , ! , and [ ])
- 段定义导引的重置属性
- 符号
- 表达式
- 位项

所需操作数的尺寸和属性根据指令或导引可能会不同。关于操作数的尺寸和属性，请参考"2.6 操作数的特征"。

关于在指令集中的操作数表达格式和描述方法，请参见开发中软件的微控制器用户手册。

以下详细说明了在操作数字段中描述的每个数据类型。

(1)　常量

常量是一个固定的值或数据项目，也可以被称为立即数。

常量被分为数字常量和字符串常量。

(a)　数字常量

二进制数，八进制数，十进制数或十六进制数可以被描述为数字常量。

每个数字常量类型的表示方法都显示在下表2-7中。

数字常量将会被处理为无符号的16位数据。

值的范围： 0 ≤ n ≤ 65 ,535 (0FFFFH)

描述一个负值时，使用负号运算符。

表2-5　表示数字常量的方法

常量	表示方法	例
二进制常量	将字符”B”或”Y”用作数字值的后缀。	1101B 1101Y
八进制常量	将字符”O”或”Q”用作数字值的后缀。	74O 74Q
十进制常量	数字值按原样描述，或将字符”D”或”T”用作数字值的后缀。	128 128D 128T
十六进制常量	- 将字符”H”用作数字值的后缀。 - 如果第一个字符以“A”, “B”, “C”, “D”, “E”, 或 “F”开头，那么”0”必须用作常量的后缀。	8CH 0A6H

(b)　字符串常量

通过将“2.2.2　字符集”中显示的一串字符封闭在一对单引号(‘)中来表示一个字符串常量。

作为汇编处理的结果，字符串常量通过将校验位（MSB）设为”0”来转换成7位ASCII码。

串常量的长度是0到2个字符。

为了将单引号本身用作一个串常量，单引号必须被连续输入两次。

<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

通过将“2.2.2　字符集”中显示的一串字符封闭在一对单引号(‘)中来表示一个字符串。字符串主要用于在DB指令中的操作数和TITLE或SUBTITLE控制指令。

<字符串的应用示例>

```
        CSEG
MAS1 :  DB      ' YES '             ; 以字符串"YES"开始。
MAS2 :  DB      ' NO '              ; 以字符串"NO"开始。
```

(3)　寄存器名

以下寄存器可以在操作数字段中进行描述。

- 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以它们的绝对名(R0到R7　和　RP0到RP3)来描述，也可以用它们的功能名(X, A, B, C, D, E, H, L, AX, BC, DE, HL)来描述。
可以在操作数字段中描述的寄存器名根据指令的类型可能会不相同。关于描述每个寄存器名描述方法的详细说明，请参考开发中的软件的各个设备的用户手册。

(4)　特殊字符

表2-8显示了可以在操作数字段中描述的特殊字符。

表2-8　可以在操作数字段中描述的特殊字符

特殊字符	功能
$	- 表示具有该操作数的指令的字单元地址（或在地址带有一个多字节指令的情况下，该地址中的第一个字节）。 - 表示分支指令的相对寻址模式。
!	- 表示分支指令的绝对寻址模式。 - 表示允许以一个MOV指令来指定所有内存空间的add16的规范。
#	- 表示立即数。
[ ]	- 表示间接寻址模式。

<特殊字符的应用示例>

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

(a)　在操作数中的第二个$表示地址103H。在同一运算中描述"BR $-1"的结果。

(b)　在操作数中的第二个$表示地址105H。在同一运算中描述"BR　$+100H"的结果。

(5)　段定义指令的重置属性

重置属性可以在操作数字段中进行描述。

关于重置属性的详细说明，请参考 “3.2　段定义指令”。

(6)　符号

如果在操作数字段中描述一个符号，那么分配到该符号中的地址（或值）将变成操作数值。

<符号的应用示例>

```
VALUE    EQU     100H
         MOV     A , #VALUE              ; 该描述可以写作“MOV A, #100H”。
```

(7)　表达式

表达式是与运算符连接的常量，$（表示一个字单元地址），名称或标签。.

表达式可以在表示数字值的地方描述为指令操作数。

关于表达式和运算符的详细资料，请参考“2.3　表达式和运算符”。

<表达式的示例>

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

在该例中，“TEN-5H”是一个表达式。

在该表达式中，名称和数字常量与一个-（减号）运算符相连。表达式的值为BH。

因此，该描述可以被写作“MOV A, #0BH”。

(8)　位项

位项可以通过位位置说明符来获取。关于位项的详细说明，请参考2.5　位位置说明符。

<位项的示例>

```
CLR1      A.5
SET1      1 + 0FE30H.3                  ; 操作数的值为0FE31H.3。
CLR1      0FE40H.4 + 2                  ; 操作数的值为0FE40H.6。
```

2.2.6　注释字段

在注释字段中，注释或备注可以在输入一个分号(;)后进行描述。注释字段是从分号到该行或EOF的换行码。通过在注释字段中描述一个注释语句，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运算（也就是转换为机器语言）的影响，但将会被原封不动地输出到汇编列表中。

可以在注释字段中进行描述的字符是那些显示在“2.2.2　字符集”中的字符。

<注释的示例>

```
            NAME       SAMPM
 ; ****************************************
 ;
 ;          HEX -> ASCII Conversion Program
 ;
 ;          main-routine
 ;
 ; ****************************************
            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 :

                                          ; chip initialize
             MOVW    SP , #_@STBEG

            MOV          HDTSA , #1AH
            MOVW     HL , #HDTSA          ; set hex 2-code data in HL register

            CALL       !CONVAH            ; convert ASCII <- HEX
                                          ; output BC-register <- ASCII code
            MOVW     DE , #STASC          ; set DE <- store ASCII code table
            MOV          A , B
            MOV        [ DE ] , A
            INCW         DE
            MOV          A , C
            MOV        [ DE ] , A

            BR              $$

           END
```

仅由注释字段组成的行

在注释字段中描述注释的行

2.3　表达式和运算符

表达式是与符号，常量，字单元地址（用$表示）中的一个相组合的运算符，或是运算符的组合。
表达式中不同于运算符的元素被称为项，并且按它们的描述顺序从左到右依次被称为第一项，第二项等。
在表2-9中所显示的运算符的类型是可用的，且它们在计算时的优先级已经按表2-10中所显示的被预先规定。
圆括号”()”用于更改执行计算时顺序。

< 例 >

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

在上述<1>，“5* (SYM+1)”是一个表达式。”5”是表达式中的第一项，“SYM” 和 “1”则分别是第二项和第三项，“*”,“+”, 和 “( )”则是运算符。

表2-9　运算符的类型

运算符的类型	运算符
算术运算符	+号, -号, + , - , * , / , MOD
逻辑运算符	NOT , AND , OR , XOR
关系运算符	EQ 或 = , NE 或 < > , GT 或 > , GE 或 >= , LT 或 < , LE 或 <=
位移运算符	SHR , SHL
字节分离运算符	HIGH , LOW , BANKNUM
特殊运算符	DATAPOS , BITPOS , MASK
其它运算符	( )

上述运算符也可以被分为一元运算符，特殊一元运算符，二进制运算符，n元运算符和其他运算符。

一元运算符:	+号, -号, NOT , HIGH , LOW , BANKNUM
特殊一元运算符:	DATAPOS , BITPOS
二进制运算符:	+ , - , * , / , MOD , AND , OR , XOR , EQ 或 = , NE 或 < > , GT 或 > , GE 或 >= , LT 或 < , LE 或 <= , SHR , SHL
n元运算符:	MASK
其他运算符 :	( )

表2-10　运算符的优先级顺序

优先级	优先级别	运算符
高	1	+号 , - 号 , NOT , HIGH , LOW , BANKNUM , DATAPOS , BITPOS , MASK
	2	* , / , MOD , SHR , SHL
	3	+ , -
	4	AND
	5	OR , XOR
低	6	EQ 或 = , NE 或 < > , GT 或 > , GE 或 >= , LT 或 < , LE 或 <=

依照以下规则来执行表达式的运算：

(1)　依照每个运算符的优先级顺序来执行运算。如果在一个表达式中存在两个或更多的具有同一优先级的运算符，将执行最左边的运算符所指定的运算。至于一元运算符，将按从右到左的顺序来执行运算。

(2)　圆括号内的表达式在圆括号外的表达式之前执行。

(3)　允许在两个或更多的一元运算符间的运算。

例：　1 = --1 == 1

-1 = -+1 = -1

(4)　表达式在16位内进行无符号数的计算。如果由于表达式超过16位而在运算时发生溢出，溢出值将被忽略。

(5)　如果一个常量超过16位(0FFFFH)，将会产生一个错误，且计算结果的值将被认为是0。

(6)　在除法中，结果中的小数部分将被清除。如果除数为0，则将会发生一个错误，且结果将会是0。

2.3.1　运算符的功能

各个运算符的功能会在这小节中进行描述。

算术运算符

(1)　+　运算符

[功能]

- 返回表达式第一项和第二项的值的和。

[应用示例]

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

[说明]

- START: BR !106H　　BR指令产生一个转移到“当前字单元地址加6”，也就是产生一个转移到地址“100H+6H=106H”。

因此，上例中的(a)也可以被描述为：START: BR !106H。

(2)　-　运算符

[功能]

- 返回从第一项的值中减去第二项的值所得到的结果。

[应用示例]

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

[说明]

- BR指令产生一个转移到“赋值到BACK减6的地址”，也就是产生一个转移到“100H-6H=0FAH”。

因此，上例中的(b)也可以被描述为：BACK: BR !0FAH。

(3)　*　运算符

[功能]

- 返回表达式的第一和第二项的值的乘积。

[应用示例]

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

[说明]

- 使用EQU指令时，“10H”值在名称”TEN”中进行定义。

“#”表示立即数。表达式“TEN*3”与“10H*3”相同，并且返回值”30H”。

因此，在上述表达式中的(c)也可以被描述为：MOV A,#30H。

(4)　/　运算符

[功能]

- 将表达式第一项的值除以表达式第二项的值，并返回到结果的整数部分。结果的小数部分将会被清除。如果除法运算中的除数（第二项）为0，则将产生一个错误

[应用示例]

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

[说明]

- 除法运算“ 256/50 ”的结果是5余6。

  运算符返回到值 ”5 ”， ”5 ”是除法运算的结果的整数部分。

  因此，上述表达式中的(d)也可以被描述为：MOV A,#5

(5)　MOD

[功能]

- 获取表达式第一项的值除以表达式第二项的值所得结果中的余数。

  如果除数（第二项）为0，则将产生一个错误。

  在MOD运算符的前后均需有一个空格。

[应用示例]

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

[说明]

- 除法运算“ 256/50 ”的结果是5余6。

  MOD运算符返回余数6。

  因此，上述表达式中的(e)也可以描述为：MOV A,#6。

(6)　+ 号

[功能]

- 原封不动地返回表达式中项的值。

[应用示例]

```
FIVE        EQU   +5
```

[说明]

- 原封不动地返回项的值 ”5 ”。

  值 ”5 ”使用EQU指令在名称 ”FIVE ”中定义。

(7)　- 号

[功能]

- 以二进制补码返回表达式中项的值。

[应用示例]

```
NO          EQU  -1
```

[说明]

- -1变成1的二进制补码。

  二进制数0000　0000　0000　0001的二进制补码

  变成: 1111 1111 1111 1111

  因此，使用EQU指令时，值“0FFFFH”将在名称”NO”中定义。

逻辑运算符

(1)　NOT

[功能]

- 将表达式中项的值按位取反并返回结果。

  在NOT运算符和项之间需有一个空格。

[应用示例]

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

[说明]

- 按以下方式在”3H”上执行逻辑非：

NOT)	0000	0000	0000	0011
	1111	1111	1111	1100

返回0FFFCH。

因此，(a)可以被描述为：MOVW　AX,　#0FFFCH

(2)　AND

[功能]

- 在表达式的第一项的值和第二项的值之间按位执行AND（逻辑与）运算并且返回结果。

  在AND运算符的前后均需有一个空格。

[应用示例]

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

[说明]

- 按以下方式在 “6FAH” 和 “0FH” 两个值之间执行AND运算：

	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)也可以被描述为：MOV A, #0AH

(3)　OR

[功能]

- 在表达式的第一项的值和第二项的值之间按位执行OR（逻辑或）运算并且返回结果。

  在OR运算符的前后均需有一个空格。

[应用示例]

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

[说明]

- 按以下方式在 “0AH” 和 “1101B” 两个值之间执行OR运算：

	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)也可以被描述为：MOV A, #0FH

(4)　XOR

[功能]

- 在表达式的第一项的值和第二项的值之间按位执行Exclusive-OR（逻辑异或）运算并且返回结果。在XOR运算符的前后均需有一个空格。

[应用示例]

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

[说明]

- 按以下方式在 “9AH” 和 “9DH” 两个值之间执行XOR运算：

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

返回结果7H。

上述表达式中的(d)也可以被描述为：MOV A, #7H

关系运算符

(1)　EQ 或 =　运算符

[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），如果两项的值不相等则返回00H（假）。在EQ运算符的前后均需有一个空格。

[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]

- 在上述的(a)中，表达式“A1 EQ (A2+4H)”变成“12C4H EQ (12C0H+4H)”。
  运算符返回0FFH，因为第一项的值等于第二项的值。
- 在上述的(b)中，表达式“A1 EQ A2” 变成 “12C4H EQ 12C0H”。
  运算符返回00H，因为第一项的值不等于第二项的值。

(2)　NE 或 < >运算符

[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），如果两项的值相等则返回00H（假）。在NE运算符的前后均需有一个空格。

[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]

- 在上述的(c)中，表达式“A1 NE A2”变成“5678H NE 5670H”。
  运算符返回0FFH，因为第一项的值不等于第二项的值。

- 在上述的(d)中，表达式“A1 NE (A2+8H)”变成“5678H NE (5670H+8H)”。
  运算符返回00H，因为第一项的值等于第二项的值。

(3)　GT 或 >　运算符

[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），如果表达式中第一项的值等于或小于第二项的值则返回00H（假）。
  在GT运算符的前后均需有一个空格。

[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]

- 在上述的(e)中，表达式“A1 GT A2”变成“1023H GT 1013H”。
  运算符返回0FFH，因为第一项的值大于第二项的值。
- 在上述的(f)中，表达式“A1 GT (A2+10H)”变成“1023H GT (1013H+10H)”。
  运算符返回00H，因为第一项的值等于第二项的值。

(4)　GE 或 >=　运算符

[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），如果表达式中第一项的值小于第二项的值则返回00H（假）。
- 在GE运算符的前后均需有一个空格。

[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]

- 在上述的(g)中，表达式“A1 GE A2”变成“2037H GE 2015H”。
  运算符返回0FFH，因为第一项的值大于第二项的值。

- 在上述的(h)中，表达式“A1 GE (A2+23H)”变成“2037H GE (2015H+23H)”。
  运算符返回00H，因为第一项的值小于第二项的值。

(5)　LT 或 <　运算符

[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），如果表达式中第一项的值等于或大于第二项的值则返回00H（假）。
  在LT运算符的前后均需有一个空格。

[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]

- 在上述的(i)中，表达式“A1 LT A2”变成“1000H LT 1020H”。
  运算符返回0FFH，因为第一项的值小于第二项的值。
- 在上述的(j)中，表达式“(A1+20H) LT A2”变成“(1000H+20H) LT 1020H”。
  运算符返回00H，因为第一项的值等于第二项的值。

(6)　LE 或 <=　运算符

[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），如果表达式中第一项的值大于第二项的值则返回00H（假）。
  在LE运算符的前后均需有一个空格。

[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]

- 在上述的(k)中，表达式“A1 LE A2”变成“103AH LE 1040H”。
  运算符返回0FFH，因为第一项的值小于第二项的值。

- 在上述的(l)中，表达式“(A1+7H) LE A2”变成“(103AH+7H) LE 1040H”。
  运算符返回00H，因为第一项的值大于第二项的值。

位移动运算符

(1)　SHR

[功能]

- 返回值，值是通过将表达式中第一项的值向右移动由第二项的值所指定的位数而获得的。相当于位移动的指定位数的0被移动到高位中。

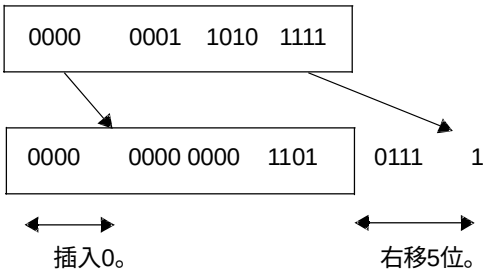
  在SHR运算符的前后均需有一个空格。

[应用示例]

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

[说明]

- 该运算符将值“01AFH”右移5位。



返回值“000DH”。

因此，上例中的(a)也可以描述为：MOV A, #0DH

(2)　SHL

[功能]

- 返回值，值是通过将表达式中第一项的值向左移动由第二项的值所指定的位数而获得的。相当于位移的指定位数的0被移动到低位中。

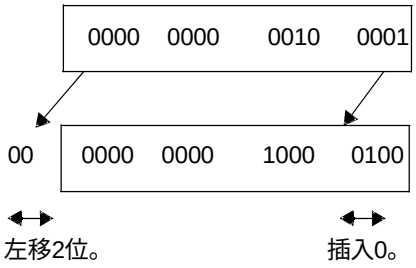
  在SHL运算符的前后均需有一个空格。

[应用示例]

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

[说明]

- 该运算符将值“21H”左移2位。



返回值“84H”。

因此，上例中的(b)也可以描述为：MOV A, #84H

字节分离运算符

(1)　HIGH

[功能]

- 返回项的高位中的8位值。

  在HIGH运算符和项之间需有一个空格。

[应用示例]

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

[Explanation]

- 通过执行MOV指令，该运算符可以返回表达式“1234H”的高位中的8位值“12H”。

因此，上例中的(a)也可以被描述为：MOV A, #12H

(2)　LOW

[功能]

- 返回项的低位中的8位值。

  在LOW运算符和项之间需有一个空格。

[应用示例]

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

[说明]

- 通过执行MOV指令，该运算符可以返回表达式“1234H”的低位中的8位值“34H”。

因此，上例中的(b)也可以被描述为：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指令以0FE68H.6的值来定义名称“SYM”。

  “DATAPOS SYM” 表示 “DATAPOS 0FE68H.6”，并返回“0FE68H”。

  因此，在上例中的(a)也可以被描述为：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指令以0FE68H.6的值来定义名称“SYM”。

  “BITPOS.SYM” 表示 “BITPOS 0FE68H.6”，并返回”6”。

  CLR1指令将[HL].6清0。

(3)　MASK

[功能]

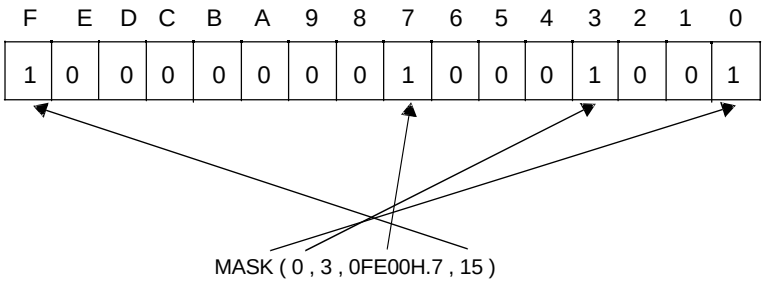
- 返回一个16位值，值中的指定位位置为1且其他位位置均被设为0。

[应用示例]

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

[说明]

- MOVW指令返回值“8089H”。



(4)　BANKNUM

[功能]

- 返回在其中定义了符号的段所在位置的库的个数。

在BANKNUM运算符和符号之间需有一个空格。

[应用示例]

```
CSEG      BANK1
SYM :      NOP
 :

MOV        A , #BANKNUM SYM   ; ( c )
```

[说明]

- 通过BANDNUM运算符返回1，它是在其中定义了标签”SYM”的段所在位置的库的个数。

  因此，上例中的(c)也可以描述为：MOV A , #1。

  然而，须注意，如果在其中定义了”SYM”的段所在位置的内存区不同于库，那么将返回”0”。

其他运算符

(1)　( ) operator

[功能]

- 使得圆括号中的运算比圆括号外的运算优先被执行。

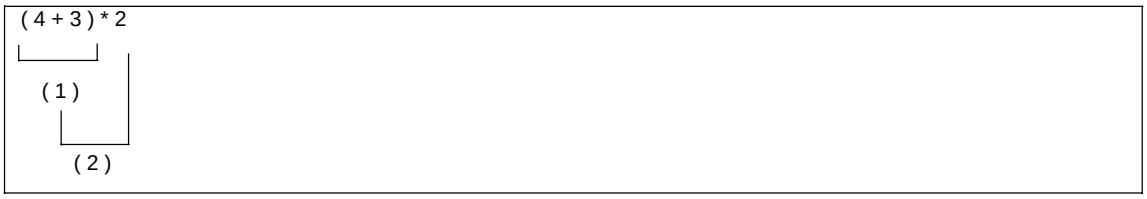
该运算符用于改变其他运算符的优先级别。

如果圆括号被嵌入多个级别中，最内部圆括号中的表达式将先被计算。

[应用示例]

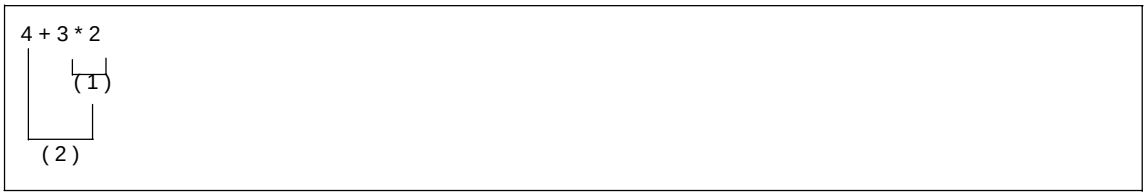
```
MOV     A , # ( 4 + 3 ) * 2
```

[说明]



按表达式<1> 和<2>的顺序来执行计算，且返回值”14”来作为结果。

如果没有使用圆括号，



将按以上显示的顺序来执行计算，且返回会下”10”来作为结果。

关于运算符的优先级别，参见表2-10。

2.4　运算限制

通过将项与运算符连接来执行表达式的运算。描述为项的元素包括常量，$，名称和标签。每个项具有一个重置属性和一个符号属性。

根据每个项内固有的重置属性和符号属性的类型，在项上执行的运算符将会受限制。因此，当描述一个表达式时，注意构成表达式中每项的重置属性和符号属性是非常重要的。

2.4.1　运算符和重置属性

正如先前所提到的，构成表达式的每个项都具有一个重置属性和一个符号属性。

当根据项的重置属性来分类时，项可以被分为三类：绝对项，浮动项和外部引用项。

在运算中的重置属性的类型，每个属性的性质，适用于每个属性的项显示在表　2-11中。

表2-11　重置属性的类型

类型	性质	适用项
绝对项	值和常量在汇编时间中确定的项	- 常量 - 在一个绝对段中定义的标签 - 表示在一个绝对段中定义的字单元地址的$ - 以常量，上述标签，上述$或绝对值定义的名称
浮动项	值不在汇编时间中确定的项	- 在一个浮动段中定义的标签 - 表示在浮动段中定义的字单元地址的$ - 以一个浮动符号定义的名称
外部引用项[注]	在外部引用另一模块中的符号的项	- 以EXTRN指令定义的标签 - 以EXTBIT指令定义的名称

注　以下五个运算符可以在外部引用项上起作用：‘+’, ‘–’, ‘HIGH’, ‘LOW’和”BANKNUM”。只有一个外部引用符号可以在表达式中进行描述。在这种情况下，外部引用符号必须与一个”+”运算符连接。

运算符的类型和可以在上面执行每个运算符的项的组合显示在表2-12中。

表2-12　按重置属性组合的项和运算符(浮动项)

项的重置属性 运算符的类型	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
BANKNUM X	A	A	A注 2	A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

<表格说明>

ABS :　　绝对项

REL :　　浮动项

A :　　运算的结果变成一个绝对项。

R :　　运算的结果变成一个浮动项。

- :　　不能执行运算。

注　1.　只有当X和Y在同一段中进行定义，并且它们不是在上面执行HIGH，LOW，DATAPOS的浮动项时才可以执行运算。

注　2.　只有当X和Y不是执行HIGN，LOW，BANDKNUM，DATAPOS的浮动项时才可以执行运算。

以下五个运算符可以在外部引用项上执行："+" , "-" , "HIGH" , "LOW" , 和 "BANKNUM"（然而，须注意只有一个外部引用项可以在表达式中进行描述）。

运算符类型和可以在上面执行运算符的外部引用项的组合依照表2-13中的重置属性进行分类。

表2-13　按重置属性组合的项和运算符(外部引用项)

项的重置属性 运算符的类型	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
BANKNUM X	A	E注 1	A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

<表格说明>

ABS :　绝对项

EXT :　外部引用项

REL :　浮动项

A :　运算的结果变成一个绝对项。

E :　运算的结果变成一个外部引用项。

R :　运算的结果变成一个浮动项。

- :　不能执行运算。

注　1.　只有当X和Y不是在上面执行HIGH , LOW , BANKNUM , DATAPOS , BITPOS的外部引用项时才可以执行运算。

注　2.　只有当X和Y不是在上面执行HIGH，LOW，BANKNUM， DATAPOS的浮动项时才可以执行运算。

2.4.2　运算符和符号属性

正如先前所提到的，构成表达式的每个项除了具有一个重置属性外还具有一个符号属性。当根据项的符号属性来分类时，项可以被分为两种类型：数字项和地址项。

在运算中的符号属性的类型和适用于每个属性的项显示在表2-14中。

表2-14　运算中符号属性的类型

符号属性的类型	适用项
数字项	- 具有数字属性的符号 - 常量
地址项	- 具有地址属性的符号 - 表示位置计数器的$

运算符的类型和根据项的符号属性可以在上面执行每个运算符的项的组合显示在表　2-15中。

表2-15　按符号属性组合的运算符和项

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
BANKNUM X	A	A	-	-
DATAPOS X	A	A	N	N
MASK X	N	N	N	N

＜表格说明＞

ADDRESS :　地址项

NUMBER :　数字项

A :　运算的结果变成一个地址项。

N :　运算的结果变成一个数字项。

- :　不能执行运算。

2.4.3　如果检查运算中的限制

此处显示了每项中根据重置属性和符号属性的运算示例。

< 例 >

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

这里假设”TABLE”是一个在浮动码段中定义的标签。

(a)　运算符和重置属性

因为“TABLE+5H”是“浮动项+绝对项”，因此该运算可以用于表2-12。

运算符的类型 :　X + Y

项的重置属性 :　X : REL , Y : ABS

从表中，你可以发现结果为R（也就是一个浮动项）。

(b)　运算符和符号属性

因为“TABLE+5H”是“地址项+数字项”，因此该运算可以用于表2-15。

运算符的类型 :　X + Y

项的符号属性 :　X : 地址 , Y : 数字

从表中，你可以发现结果为A（也就是一个地址项）。

2.5　位位置说明符

可以通过使用位位置说明符(.　　)来访问位。

位位置说明符

(1)　句点(.)

[格式描述]

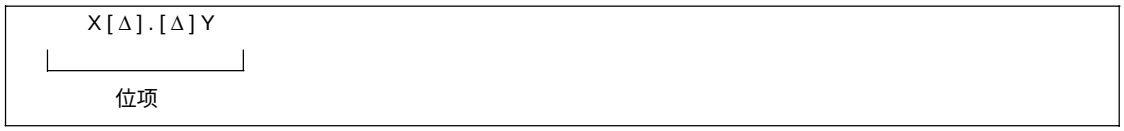


表2-16　X（第一项）和Y（第二项）的组合

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

注　关于特定描述的详细说明，参见各个设备的用户手册。

[功能]

- 位位置说明符以它的第一项指定一个字节地址，以它的第二项指定位的位置。通过该位位置说明符可以访问特定的位。

[说明]

- 位项指的是一个使用位位置说明符的表达式。
- 位位置说明符不受运算符的优先级别的影响。位位置说明符的的左侧被认为是第一项，它的右侧被认为是第二项。
- 以下限制用于第一项：
  (1) 带有数字或地址属性的表达式，能够进行位访问的SFR的名称或寄存器名（A）可以被描述。
  (2) 绝对值表达式在第一项中进行描述时，它必须在0FE20H到0FF1FH的范围内。
  (3) 可以描述一个外部引用符号。
- 以下限制用于第二项：
  (1) 表达式的值必须在0到7的范围内。如果超过该值范围，将会产生一个错误。
  (2) 只有带有数字属性的绝对表达式可以被描述。
  (3) 没有外部引用符号可以被描述。

[运算和重置属性]

- 按重置属性组合的第一和第二项显示在表2-17中。

表2-17　按重置属性组合的第一和第二项

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

<表格说明>

ABS :　绝对项

EXT :　外部引用项

REL :　浮动项

A :　运算的结果变成一个绝对项。

E :　运算的结果变成一个外部引用项。

R :　运算的结果变成一个浮动项。

- :　不能执行运算。

[位符号的值]

- 通过描述在EQU指令的操作数字段中使用位位置说明符的位项来定义一个位符号时，位符号将会具有的值会显示在下表2-18中。

表2-18　位符号的值

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

注　1.　关于详细的描述，请参考各个设备的用户手册。

注　2.　位=0到7

注　3.　FFxxH表示sfr的地址。

注　4.　xxxxH表示表达式的值。

[应用示例]

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

2.6　操作数的特征

需要一个或多个操作数的指令和导引不同于在所需操作数值的的尺寸和地址范围中的各个类型的指令以及在操作数的符号属性中的各个类型的指令。

例如，指令“MOV r, #byte”可以将由”byte”所表示的值转移到寄存器”r”中。假使这样，因为r是一个8位寄存器，因此要转移的数据”byte”的尺寸必须是8位或小于8位。

如果指令被描述为“MOV R0, #100H”，将产生一个汇编错误，因为指令的第二操作数“100H”的尺寸起来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：

- 操作数的尺寸或地址范围是否适合于指令的操作数（数字数据，名称或标签）？
- 符号属性是否适合于指令的操作数（名称或标签）？

2.6.1　操作数的值的尺寸和地址范围

可以为描述为指令的操作数的数字数据，名称或标签的值的尺寸和地址范围设置一定的条件。

使用指令时，为操作数值的尺寸和地址范围所设置的条件是由每个指令的操作数表达格式来支配的。使用指令时，为操作数值的尺寸和地址范围所设置的条件是由指令的类型来支配的。

这些条件显示在下面的表2-19和表2-20中。

表2-19　指令的操作数值的范围

操作数表达格式	值的范围	
byte	8位值0H到FFH	
word	16位值0H到FFFFH	
saddr	FE20H到FF1FH	
saddrp	FE20H到FF1FH中的偶数值	
sfr	FF00H 到 FFCFH, FFE0H 到 FFFFH	
sfrp	FF00H到FFCFH, FFE0H到FFFFH中的偶数值	
addr16	MOV, MOVW	0H 到 FFFFH
	其他指令	0H 到 FA7FH
addr11	800H 到 FFFH	
addr5	40H 到 7EH中偶数值	
bit	3位值0到7	
n	2位值0到3	

表2-20　指令的操作数值的范围

指令的类型	指令	值的范围
段定义指令	CSEG AT	0H 到 FFFFH
	DSEG AT	0H 到 FFFFH
	BSEG AT	FE20H 到 FEFFH
	ORG	0H 到 FFFFH

表2-20　指令的操作数值的范围

指令的类型	指令	值的范围
符号定义指令	EQU	16位值 0H 到 FFFFH
	SET	16位值 0H 到 FFFFH
内存初始化和区域保留指令	DB	8位值 0H 到 FFH
	DW	16位值 0H 到 FFFFH
	DS	16位值 0H 到 FFFFH
自动分支指令选择指令	BR	0H 到 FFFFH

2.6.2　指令所需的操作数的尺寸

指令可以被分为机器指令和伪指令。对于需要立即值和符号来作为操作数的指令来说，所需的操作数的尺寸根据每个指令而变化。

因此，当超过指令所需操作数的尺寸的数据被描述时，将产生一个错误。表达式的运算按无符号数的16位来完成。如果计算结果超过0FFFFH（16位），则将输出一个警告消息。

然而，当在操作数中描述浮动符号或外部引用符号时，值在汇编器中不会被确定。取而代之的是，连接器将会确定值并检查值的范围。

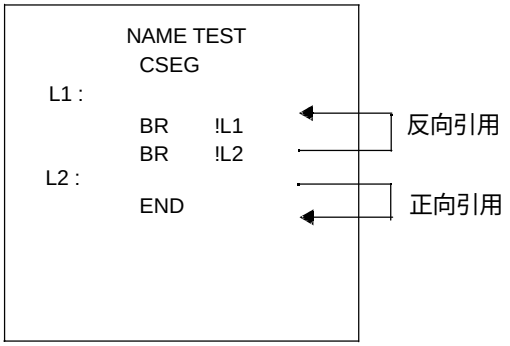
2.6.3　操作数的符号属性和重置属性

当将名称，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。这取决于用作它们的表达式中的项的符号属性和重置属性（参见表4　运算限制），也取决于名称和标签的引用方法。

名称和标签的引用方法可以是反向引用或正向引用。

- 反向引用... 引用为操作数的名称或标签，操作数在名称或标签之前在行中被定义。
- 正向引用... 引用为操作数的名称或标签，操作数在名称或标签之后在行中被定义。

<例>



这些符号属性和重置属性，与名称及标签的引用方法一样，都显示在表2-21和表2-22中。

表2-21　符号描述为操作数时的特性

符号属性	数字		地址				数字 地址		sfr保留字[注1]
重置属性	绝对项		绝对项		浮动项		外部引用项		
引用模式 格式描述	反向	正向	反向	正向	反向	正向	反向	正向	
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
addr11	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

<表格说明>

正向 :　　这表示正向引用。

反向 :　　这表示反向引用。

OK :　　这表示可以描述。

NG :　　这表示一个错误。

- :　　这表示不能描述。

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指令的操作数时必须是反向引用。禁止正向引用。

注　2.　如果在saddr区域中的sfr保留字已经为一个指令而描述（指令中sfr/sfrp的组合由存在于操作数组合中的saddr/saddrp变化而来），那么代码将输出为saddr/saddrp。

注　3.　在saddr区域中的sfr保留字

注　4.　在saddr区域中的sfrp保留字

注　5.　只有绝对表达式

注　6.　只有允许8位访问的sfr保留字

注　7.　只有允许16位访问的sfr保留字

注　8.　当描述将禁止区域(FA80H到FADFH)用作addr16的值的地址时，将不会执行检查。

表2-22　符号被描述为指令的操作数时的特性

符号属性		数字		地址, SADDR						位					
重置属性		绝对项		绝对项		浮动项		外部引用项		绝对项		浮动项		外部引用项	
引用方向 伪指令		反向 -	正向	反向 -	正向	反向 -	正向	反向 -	正向	反向 -	正向	反向 -	正向	反向 -	正向
ORG		Ok注1	-	-	-	-	-	-	-	-	-	-	-	-	-
EQU Note 2		OK	-	OK	-	OK注3	-	-	-	OK	-	OK注3	-	-	-
SET		OK注1	-	-	-	-	-	-	-	-	-	-	-	-	-
DB	尺寸	OK注1	-	-	-	-	-	-	-	-	-	-	-	-	-
	初始值	OK	OK	OK	OK	OK	OK	OK	OK	-	-	-	-	-	-
DW	尺寸	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

- :　　不能描述

注　1.　只有绝对表达式可以被描述。

注　2.　如果包含以下任一模式的表达式被描述，则将产生一个错误。

- 地址属性 – 地址属性
- 地址属性关系运算符　地址属性
- HIGH绝对地址属性
- LOW绝对地址属性
- BANKNUM绝对地址属性
- DATAPOS绝对地址属性
- MASK绝对地址属性
- 当运算结果能受上述7种优化模式的影响时。

注　3.　不允许具有浮动项的HIGH/LOW/DATAPOS/MASK运算符所创建的项

注　4.　参考”3.4 (3) DS（定义存储器）”

第 3 章　伪指令

本章节说明了伪指令。伪指令是指示RA78K0所需的所有类型的指令执行一系列处理的指令。

3.1　伪指令概述

作为汇编的结果，指令被转换成目标代码（机器语言），但伪指令在原则上不会被转换成目标代码。伪指令主要包括以下功能：

- 使源程序的描述更为容易
- 初始化内存并保留内存区
- 提供汇编器所需的信息和用于完成指定数据处理的连接器所需的信息

表3-1　显示了伪指令的类型。

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/区域保留伪指令	DB , DW , DS , DBIT
连接伪指令	PUBLIC , EXTRN , EXTBIT
目标模块名说明伪指令	NAME
自动转移指令选择伪指令	BR
宏伪指令	MACRO , LOCAL , REPT , IRP , EXITM , ENDM
汇编终止伪指令	END

以下的节说明了每个伪指令的详情。

在每个伪指令的格式描述中，“[　　]”表示在方括号中的参数可能从规范中省略，“...”则表示在同一格式中的重复描述。

3.2　段定义伪指令

源模块必须在段单元中进行描述。

段定义伪指令用于定义这些段。段被分为以下四种类型：

(1)　代码段

(2)　数据段

(3)　位段

(4)　绝对段

段的类型决定了段被分配到的内存中的地址范围。

表3-2　显示了每个段的定义方法以及段被分配到的内存地址。

表3-2　段定义方法和内存地址的位置

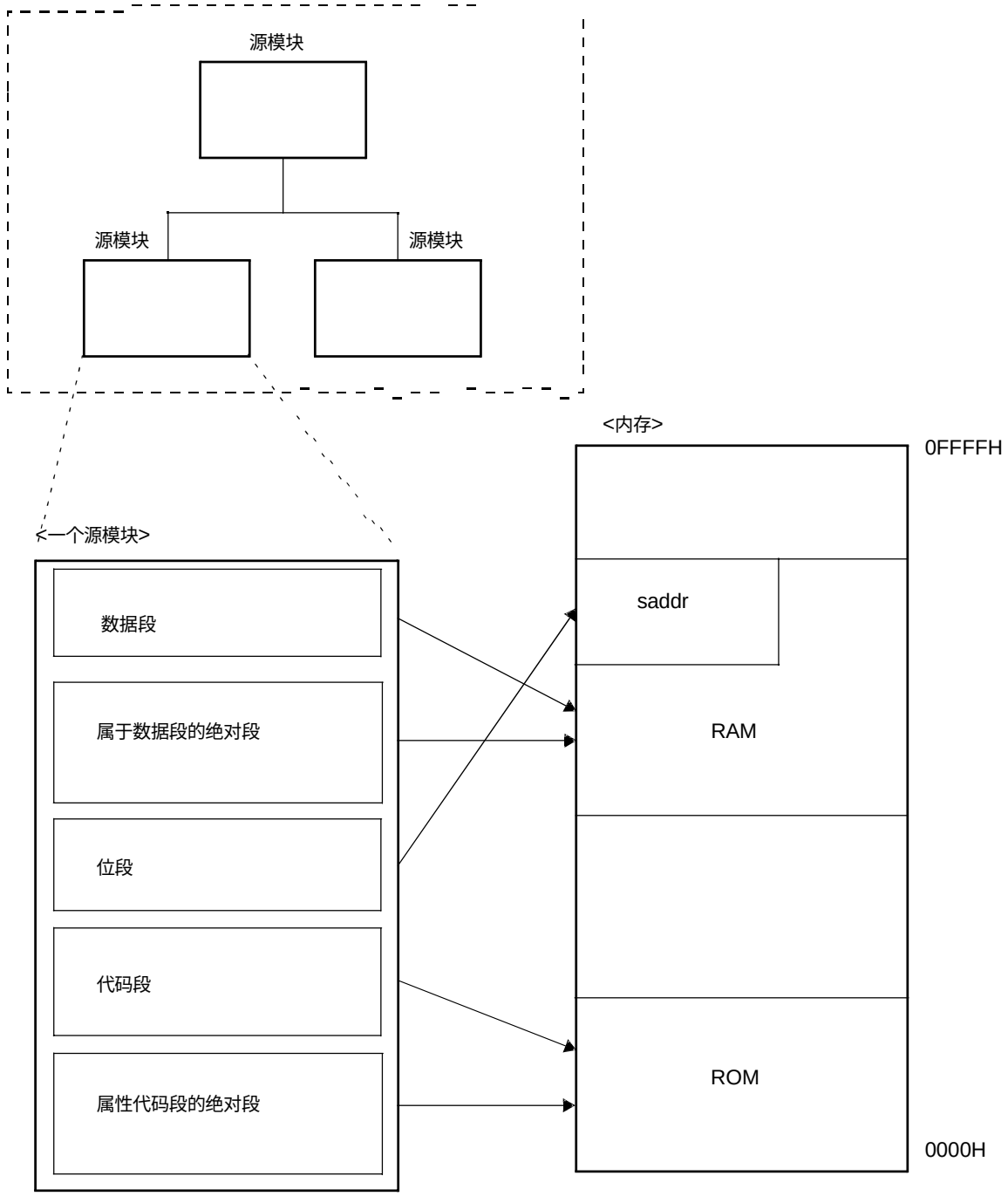
段类型	定义方法	段被分配到的内存地址
代码段	CSEG伪指令	在内部或外部ROM地址中
数据段	DSEG伪指令	在内部或外部RAM地址中
位段	BSEG伪指令	在内部RAM的saddr区域中
绝对段	使用CSEG，DSEG，或BSEG伪指令将字单元地址（AT字单元地址）指定为重置属性	指定地址

如果用户想确定段的内存位置，则需将段描述为一个段。对于堆栈区域，用于需要在数据段中保留该区域并在堆栈指针中对其进行设置。当指定安全ID时，段不能被置于85H到8EH的地址范围中。

当使用片上调试功能时，段不能被置于02H到03H以及84H的地址范围中，也不能被置于8FH到片上调试程序尺寸+1的范围内。

段位置的示例显示在图3-1中。

图3-1　段的内存位置



可以使用以下的段定义伪指令。

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

CSEG

(1)　CSEG (代码段)

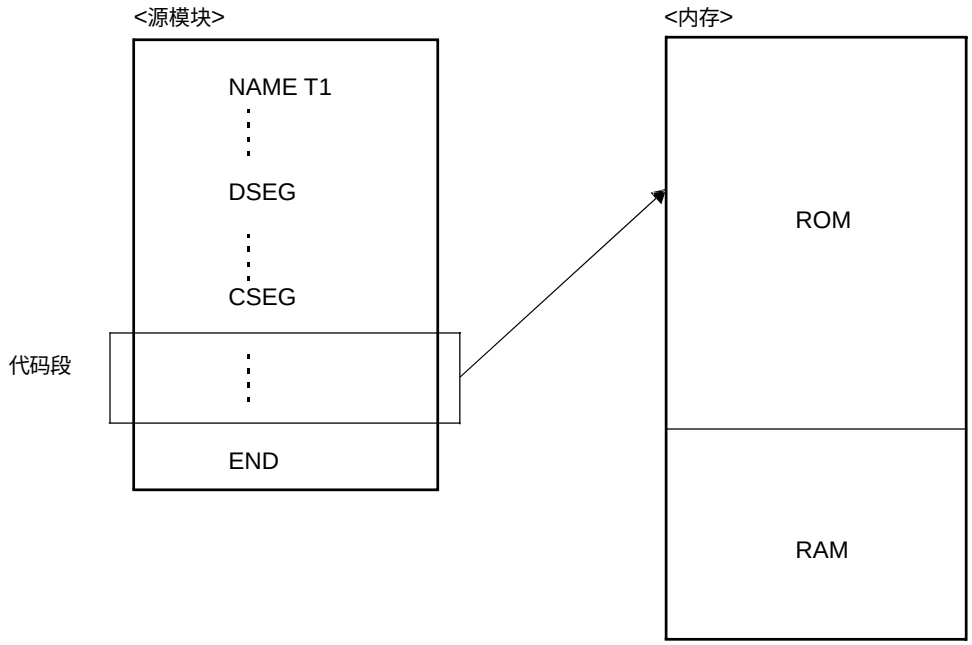
[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[段名]	CSEG	[重置属性]	[；注释]

[功能]

- CSEG伪指令将代码段的起始处显示在汇编器中。
- 所有在CSEG伪指令后描述的指令都属于代码段，直到该伪指令遇到一个段定义伪指令（CSEG，DSEG，BSEG或ORG）或END伪指令，并且最终这些指令将在转换为机器语言后被置于一个ROM地址中。

图3-2　代码段的重置



[用途]

- CSEG伪指令用于描述CSEG伪指令所定义的代码段中的指令，DB，DW伪指令等。（然而，为了重置固定地址中的代码段，“AT 绝对表达式”必须在操作数字段中被描述为重置属性。）
- 一个功能单元描述，例如一个子程序，应该被定义为一个单一的代码段。如果单元相对较大或子程序具有高通用性（也就是说，可以用于其他程序的开发），那么子程序应该被定义为一个单一的模块。

[说明]

- 代码段的开始地址可以用ORG伪指令来指定。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字单元地址的范围。重置属性显示在表3-3中。

表3-3　CSEG的重置属性

重置属性	格式描述	说明
CALLT0	CALLT0	命令汇编器对指定段进行定位，这样可以使段的开始地址在0040H到007FH的范围内变成2的倍数。为用于定义 子程序的输入地址的代码段来指定该重置属性，其中子程序将以1字节指令"CALLT"来调用。
FIXED	FIXED	命令汇编器对在地址范围0800H到0FFFH内的指定段进行定位。为用于定义子程序的代码段来指定该重置属性， 其中子程序将以2字节指令"CALLF"来调用。
AT	AT绝对表达式	命令汇编器将指定段置于一个绝对地址(0000H到FEFFH)中。
UNIT	UNIT	命令汇编器将指定段置于任一地址(0080H到FA7FH)中。
UNITP	UNITP	命令汇编器将指定段置于任一地址中，这样地址的起始处可能是偶数(0080H到FA7EH)。
IXRAM	IXRAM	命令汇编器将指定段置于内部扩展RAM中。
SECUR_ID	SECUR_ID	它是一个安全ID的特定属性。除了安全ID不指定任何内容。 命令汇编器将指定段置于0085H到008EH的地址范围内。
BANK0 到 15	BANK0 到 15	命令汇编器将指定段置于x8000H到xBFFFH (x = 0 到 F)的地址范围内。
BANK0 AT 到 BANK15 AT	绝对表达式 到 BANK15 AT 绝对表达式	BANK0 AT　　命令汇编器将指定段置于x8000H到xBFFFH(x = 0 到 F)的绝对地址范围内。

- 如果没有为代码段指定重置属性，汇编器将假设"UNIT"已经被指定。
- 如果指定了一个不同于表3-3中的重置属性，汇编器将输出一个错误消息并假设"UNIT"已经被指定。如果代码段的尺寸超过了由其重置属性所指定的区域尺寸，则将会产生一个错误。
- 如果以重置属性"AT"指定的绝对表达式是非法的，汇编器将输出一个错误消息并且通过假设表达式的值为"0"来继续执行数据处理。

通过在CSEG伪指令的符号字段中描述一个段名，代码段将可以被命名。如果没有为代码段指定段名，汇编器

将自动赋于代码段一个默认的段名。

代码段的默认段名显示在表3-4中。

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
IXRAM	?CSEGIX
SECUR_ID	?CSEGSI
none	?CSEGOB0 到 ?CSEGOB4
AT	不能省略段名。
BANK0 to到 15	?CSEGB0 - ?CSEGB15

- 当重置属性为AT时，如果省略了段名，则将会产生一个错误。
- 如果两个或更多的代码段具有同一重置属性（除了AT），那么这些代码段可能会具有同一段名。在汇编器中这些同名的代码段被当作一个代码段来处理。

  如果同名段的重置属性不相同，则将会产生一个错误。因此，对于每个重置属性来说，同名段的个数为1。
- 在两个或更多的不同模块中的同名代码段在连接时被合并为一个代码段。
- 没有段名可以被引用为一个符号。
- 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大个数为8。
- 段名的大写和小写字符被区分。
- 选项字节在80H到84H的地址范围内被指定（这些地址因设备而不同）。如果选项字节为没有选项字节特征的片而指定，则将产生一个错误。

  选项字节没有为具有选项字节特征的片而指定时，则应将"?CSEGOB0 到 ?CSEGOB4"内的默认段定义到各个地址中，并设置从设备文件中读取的初始值。

[应用示例]

```
            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”，因此将产生一个错误。

(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)

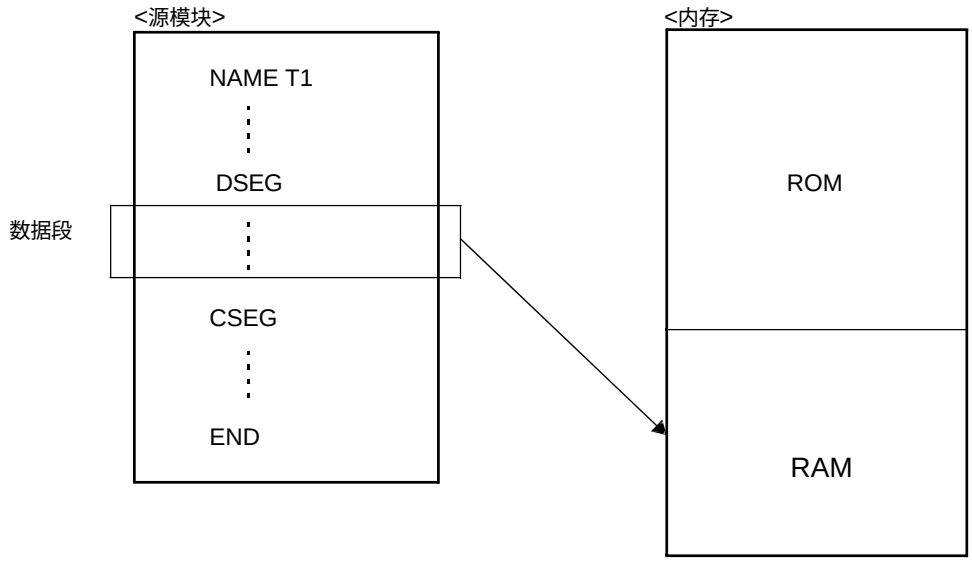
[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[段名]	DSEG	[重置属性]	[；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，直到该伪指令遇到一个段定义伪指令（CSEG，DSEG，BSEG或ORG）或END伪指令，并且最终它将被保留在RAM地址内。

图3-3　数据段的重置



[用途]

- DS伪指令主要在由DSEG伪指令所定义的数据段中进行描述。数据段被置于RAM区域内。因此，在任何数据段中都不能描述指令。
- 在一个数据段中，程序中所使用的RAM工件区通过DS伪指令来保留，且标签被连接到每个工作区中。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伪指令来指定。它也可以通过描述重置属性"AT"来指定，重置属性"AT"接在DSEG伪指令的操作数字段中的绝对表达式之后。

- 重置属性为数据段定义了字单元地址的范围。数据段中可用的重置属性显示在表3-5中。

表3-5　DSEG的重置属性

重置属性	格式描述	说明
SADDR	SADDR	命令汇编器对saddr区域（saddr区域：0FE20H到0FEFFH）中的指定段进行定位。
SADDRP	SADDRP	命令汇编器对saddr区（saddr区域：0FE20H到0FEFFH）的偶数地址中的指定段进行定位。
AT	AT 绝对表达式	命令汇编器对绝对地址中的指定段进行定位
UNIT	UNIT或没有规范	命令汇编器对任一位置（在内存区名为"RAM"的内存内）中的指定段进行定位。
UNITP	UNITP	命令汇编器对偶数地址（在内存区名为"RAM"的内存内）中的任一位置的指定段进行定位。
IHRAM	IHRAM	命令汇编器对高速RAM区中的指定段进行定位。
LRAM	LRAM	命令汇编器对低速RAM区中的指定段进行定位。
DSPRAM	DSPRAM	命令汇编器对显示RAM区中的指定段进行定位。
IXRAM	IXRAM	命令汇编器对内部扩展RAM区中的指定段进行定位。

- 如果没有为数据段指定重置属性，汇编器将假设"UNIT"已经被指定。
- 如果指定了一个不同于表3-5中的重置属性，汇编器将输出一个错误消息并假设"UNIT"已经被指定。如果每个数据段的尺寸超过了其重置属性所指定的区域尺寸，则将会产生一个错误。
- 如果由重置属性"AT"指定的绝对表达式是非法的，汇编器将输出一个错误消息并通过假设表达式的值为"0"来继续来继续执行数据处理。
- 通过在DSEG伪指令的符号字段中描述一个段名，数据段将可以被命名。如果没有为数据段指定段名，汇编器将自动赋于数据段一个默认的段名。数据段的默认段名显示在表3-6中。

表3-6　DSEG的默认段名

重置属性	默认段名
SADDR	?DSEGS
SADDRP	?DSEGSP

表3-6　DSEG的默认段名

重置属性	默认段名
UNIT (或没有指定)	?DSEG
UNITP	?DSEGUP
IHRAM	?DSEGIH
LRAM	?DSEGL
DSPRAM	?DSEGDSP
IXRAM	?DSEGIX
AT	不能省略段名。

- 如果两个或更多的数据段具有同一重置属性（除了AT），那么这些数据段可能会具有同一段名。在汇编器中这些段被当作一个数据段来处理。
- 如果重置属性为SADDRP，指定地址将被定位，这样在DSEG伪指令被描述后，地址将会立即变为2的倍数。
- 如果同名段的重置属性不相同，则将会产生一个错误。因此，对于每个重置属性来说，同名段的个数为1。
- 在两个或更多的不同模块中的同名数据段在连接时被合并为一个代码段。
- 没有段名可以被引用为一个符号。
- 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大个数为8。
- 段名的大写和小写字符被区分。

[应用示例]

```
        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)　数据段的起始处由DSEG伪指令所定义。因此省略了其重置属性，因为将假设为”UNIT”。默认段名为”?DSEG”。

(2)　该描述相当于“MOV A, !addr16”。

(3)　该描述相当于“MOV A, saddr”。浮动标签”WORK1”不能被描述为”saddr”。因此，该描述会产生一个错误。

(4)　该描述相当于“MOVW rp, #word”。

(5)　该描述相当于“MOVW AX, saddrp”。
浮动标签“WORK2”不能被描述为”saddrp”。因此，该描述会产生一个错误。

BSEG

(3)　BSEG (位段)

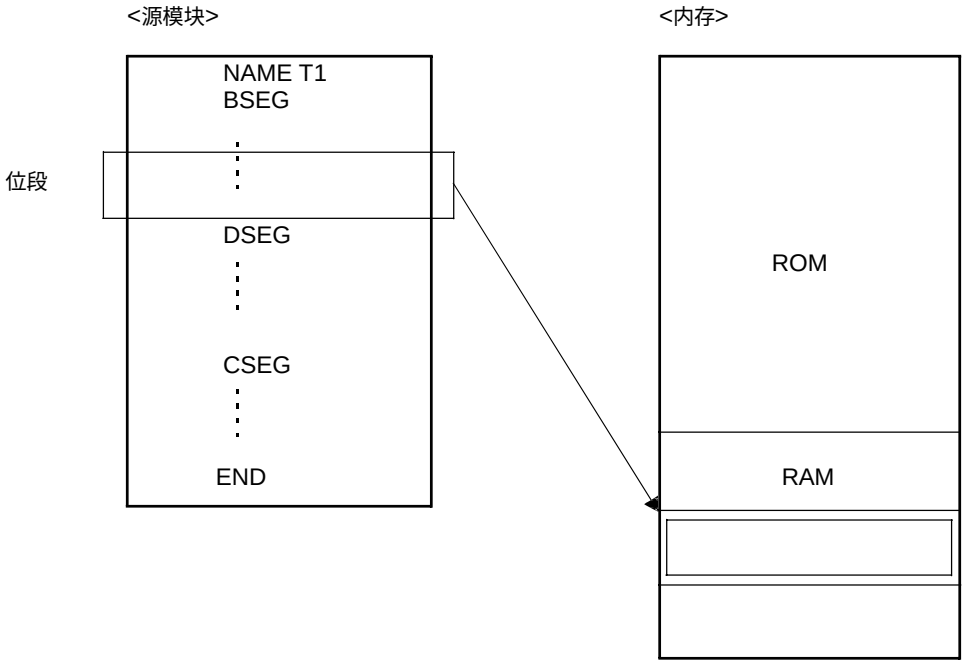
[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[段名]	BSEG	[重置属性]	[；注释]

[功能]

- BSEG伪指令将位段的起始处显示在汇编器中。
- 位段是用于定义在源模块中所使用的RAM地址的段。
- 在BSEG伪指令后由DBIT伪指令定义的内存区属于位段，直到该伪指令遇到一个段定义伪指令（CSEG，DSEG，或 BSEG）或 END 伪指令。

图3-4　位段的重置



[用途]

- 在BSEG伪指令所定义的位段中描述DBIT伪指令（参见[应用示例]）。
- 不能在任何位段中描述指令。

[说明]

- 位段的开始地址可以通过描述在重置属性字段中的"AT　绝对表达式"来指定。

- 重置属性为位段定义了字单元地址的范围。可用于位段中的重置属性显示在表3-7中。

表3-7　BSEG的重置属性

重置属性	格式描述	说明
AT	AT 绝对表达式	命令汇编器对绝对地址0位中指定段的开始地址进行定位。禁止以位为单位的指定（FE20H到FEFFH）。
UNIT	UNIT　(或没有指定)	命令汇编器对任一位置(FE20H到FEFFH)中的指定段进行定位。

- 如果没有为位段指定重置属性，汇编器将假设指定了“UNIT”。
- 如果指定了一个不同于表3-7中的重置属性，汇编器将输出一个错误消息并假设”UNIT”已经被指定。如果位段的尺寸超过了由其重置属性所指定的区域尺寸，则将会产生一个错误。
- 在汇编器和连接器中，位段中的位置计数器显示在格式“0xxxx.b”中(字节地址为16进制的4位数，位位置是16进制的1位数(0到7))。

(1)　有绝对位段

表3-8　位置计数器（绝对）

字节地址	位位置							
	0	1	2	3	4	5	6	7
0FE20H	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
0FE21H	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)

<位置计数器>

(1) 0FE20H.0　　(9) 0FE21H.0
(2) 0FE20H.1　　(10) 0FE21H.1
(3) 0FE20H.2　　(11) 0FE21H.2
(4) 0FE20H.3　　(12) 0FE21H.3
(5) 0FE20H.4　　(13) 0FE21H.4
(6) 0FE20H.5　　(14) 0FE21H.5
(7) 0FE20H.6　　(15) 0FE21H.6
(8) 0FE20H.7　　(16) 0FE21H.7

(2)　有浮动位段

表3-9　位置计数器（浮动）

字节地址	位位置							
	0	1	2	3	4	5	6	7
0H	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
1H	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)

<位置计数器>

(1) 0H.0　　(9) 1H.0
(2) 0H.1　　(10) 1H.1
(3) 0H.2　　(11) 1H.2
(4) 0H.3　　(12) 1H.3
(5) 0H.4　　(13) 1H.4
(6) 0H.5　　(14) 1H.5
(7) 0H.6　　(15) 1H.6
(8) 0H.7　　(16) 1H.7

备注　在一个浮动位段内，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。
在由目标转换器输出的符号列表中，定义了一个位的区域的开始位置的位偏移量将会被显示并输出。

表3-10　符号值和位偏移量显示

符号值	位偏移量
00FE20H.0	0000
00FE20H.1	0001
00FE20H.2	0002
:	:
00FE20H.7	0007
00FE21H.0	0008
00FE21H.1	0009
:	:
00FE80H.0	0300
:	:

- 如果以重置属性“AT”指定的绝对表达式是非法的，汇编器将输出一个错误消息并通过假设表达式的值为”0”来继续执行数据处理。

- 通过描述在BSEG伪指令的符号字段中的段名，可以对位段进行命名。如果没有为位段指定段名，汇编器将自动赋于位段一个默认的段名。下表显示了默认段名。

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	不能省略段名。

- 如果重置属性是"UNIT"，两个或更多的数据段可以具有同一段名（除了AT）。这些段在汇编器中将作为一个段来处理。

因此，对于每个重置属性来说，具有同一段名的段的个数是1。

- 在两个或更多的不同模块中的同名位段在连接时将合并为一个位段。
- 没有段名可以被引用为一个符号。
- 可以在位段中描述的指令是DBIT，EQU，SET，PUBLIC，EXTBIT，EXTRN，MACRO，REPT，IRP，ENDM伪指令，宏定义和宏引用。描述不同于这些指令的指令将会产生一个错误。
- 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。具有同一名称的段被计作1个段。
- 可以当作段名的字符的最大个数为8。
- 段名的大写和小写字符被区分。

[应用示例]

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

(3)　位段通过BSEG来定义。

由于省略了其重置属性，因此将假设重置属性为”UNIT”，段名为”?BSEG”。在每个位段中，都会通过DBIT伪指令为每个位来定义一个位工作区。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，在位段内定义的位地址FLAG2将会被定位。

(4)　该描述可以以“SET1 FLAG.0”来代替。这个FLAG表示一个字节地址。

(5)　在这个描述中，不会考虑字节地址边界。

ORG

(4)　ORG (原点)

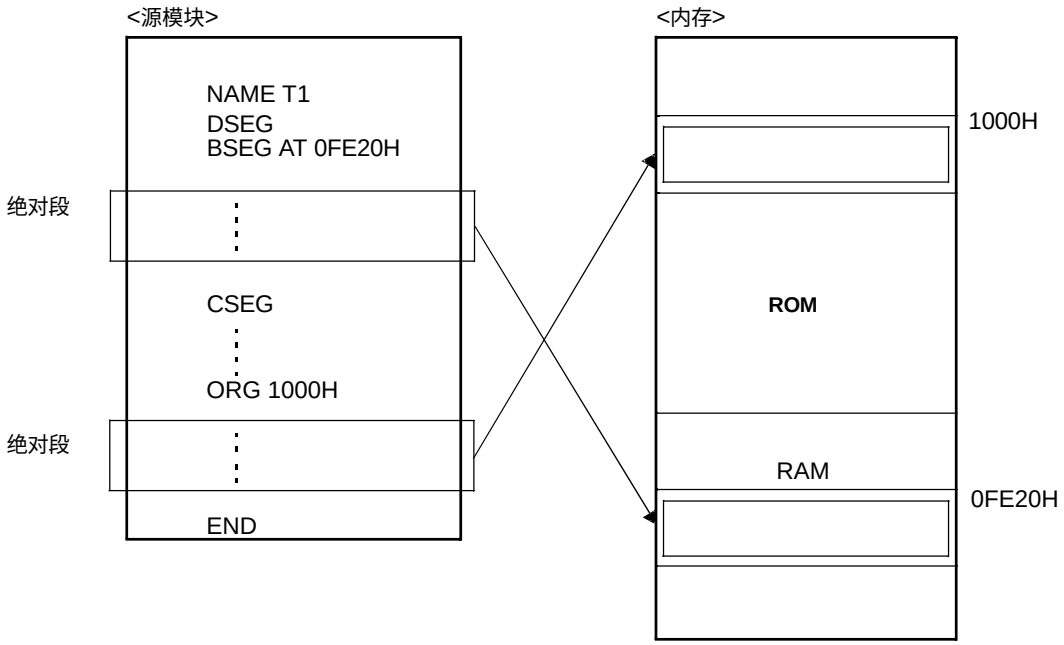
[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[段名]	ORG	[重置属性]	[；注释]

[功能]

- ORG伪指令设置了由其位置计数器的操作数所指定的表达式的值。
- 在ORG伪指令之后所描述的指令和保留内存区都属于一个绝对段，直到该伪指令遇到段定义伪指令(CSEG, DSEG, BSEG，或ORG)或END伪指令，它们从操作数所指定的地址中来定位。

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和数据段进行定位。

[说明]

- 以ORG伪指令定义的绝对段属于在该ORG伪指令之前以CSEG或DSEG伪指令定义的代码段和数据段。

  在属于数据段的绝对段内，没有指令可以被描述。

  属于位段的绝对段不能以ORG伪指令来描述。
- 以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。

- 通过描述在ORG伪指令的符号字段中的段名，可以对绝对段进行命名。可以当作段名的字符的最大个数为8。
- 如果没有为绝对段指定段名，那么汇编器将自动赋于其默认的段名“?A00xxxx”， 其中“xxxx”表示指定段的4位数的十六进制的开始地址（0000到FEFF)。
- 如果在ORG伪指令前没有描述CSEG和DSEG伪指令，那么以ORG伪指令定义的绝对段将被解释为代码段中的绝对段。
- 如果名称和标签被描述为ORG伪指令的操作数，那么名称和标签必须是一个已经在源模块中定义的绝对段。
- 没有段名可以被引用为一个符号。
- 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ORG伪指令所定义的段。具有同一名称的段被计作1个段。
- 可以当作段名的字符的最大个数为8。
- 段名的大写和小写字符被区分。

[应用示例]

```
                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)　属于数据段的绝对段被定义。该绝对段将从地址”FE20H”开始位置的短直接寻址区中进行定位。

由于省略了段名的指定，因此汇编器将自动赋于其一个段名“?A00FE20”。

(2)　由于在属于数据段的绝对段中没有指令被描述，因此将产生一个错误。

(3)　该伪指令定义了代码段的开始位置。

(4)　该绝对段被置于从地址”1000H”开始的区域中。

3.3　符号定义伪指令

符号定义伪指令将名称赋于用于描述源模块的数字数据。这些名称对每个数据值的含义进行分类，并使源模块的内容变得易于理解。符号定义伪指令将用在源模块中的每个名称的值告诉汇编器。

EQU（相等）和SET（设置）这两个伪指令可以用于符号定义。以下的符号定义伪指令为可用的。

- EQU（相等）
- SET（设置）

EQU

(1)　EQU (相等)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
名称	EQU	表达式	[；注释]

[功能]

- EQU伪指令定义了在操作数字段中所指定的表达式的名称，该名称具有值和属性（符号属性和重置属性）。

[用途]

- 通过EQU伪指令将源模块中使用的数字数据定义为名称，并描述代替数字数据的指令的操作数中的名称。建议将经常用在源模块中的数字数据定义为一个名称。如果你必须在源模块中改变数据的值，那么你所要做的就是改变该名称的操作数值。

[说明]

- 当要在EQU伪指令的操作数中描述名称和标签时，则应使用已经在源模块中被定义的名称或标签。没有外部引用项可以被描述为该伪指令的操作数。
- 包含一个由HIGH/LOW/DATAPOS/BITPOS运算符创建的项的表达式不能被描述，其中HIGH/LOW/DATAPOS/BITPOS运算符在它的操作数中具有一个浮动项。
- 如果一个具有以下任一操作数的模式的表达式被描述，则将会产生一个错误：
  (1)　有地址属性的表达式1－有地址属性的表达式2
  (2)　有地址属性的表达式1　关系运算符　有地址属性的表达式2
  (3)　下列任一条件<1>和<2>满足上述的表达式(a)或(b)：
    (a)　如果带有地址属性的表达式1中的标签1和带有地址属性的表达式2中的标签2属于同一段，且如果目标代码的字节个数不能确定的BR伪指令在两个标签中进行描述时
    (b)　如果标签1和标签2在段中不相同，且如果目标代码的字节个数不能确定的BR伪指令在段的开始位置和标签间进行描述时
  (4)　有地址属性的HIGH绝对表达式
  (5)　有地址属性的LOW绝对表达式
  (6)　有地址属性的BANKNUM绝对表达式
  (7)　有地址属性的DATAPOS绝对表达式
  (8)　有地址属性的BITPOS绝对表达式
  (9)　下列(a) 满足表达式(4) , (5) , (6) , (7) , or (8) :

(a)　如果目标代码的字节个数不能立即确定的BR伪指令在带有地址属性的表达式和标签所属段的开始位置间进行描述时

- 如果在操作数的格式描述中存在一个错误，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。
- 以EQU伪指令定义的名称不能在同一源模块内重新定义。
- 已经用EQU伪指令定义了一个位值的名称将会和值一样具有一个地址和位位置。
- 表 3-12显示了可以描述为EQU伪指令的操作数的位值以及可以引用这些位值的范围。

表3-12 表示位值的操作数的表示格式

操作数类型	符号值	引用范围
A.位[注 1]	1.位	只可以在同一模块内引用。
PSW. 位[注1]	1FEH.位	
sfr[注 2].bit[注 1]	0FFxxH[注 3].位	
saddr.位[注1]	0xxxxH[注 4].位	可以从其他模块引用
expression.位[注1]	0xxxxH[注 4].位	

注　1.　1位=0到7

注　2.　关于详细描述，请参考每个设备的用户手册。

注　3.　“0FFxxH”表示sfr的地址。

注　4.　“0xxxxH”表示saddr区域(FE20H到FF1FH)。

[应用示例]

```
        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”被赋于在操作数格式“saddr.bit”中的位值“WORK1.0”。

上述(1)中，在操作数中描述的“WORK1”已经在值“0FE20H”中定义。

(3)　名称“P02”被赋于在操作数格式“sfr.bit”中的位值“P0.2”。

(4)　名称“A4”被赋于在操作数格式“A.bit”中的位值“A.4”。

(5)　名称“PSW5”被赋于在操作数格式“PSW.bit”中的位值“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)到(5)中一样，已经定义了“sfr.bit”, “A.bit”, and “PSW.bit”的名称只能在同一模块中引用。

已经定义了“saddr.bit”的名称也可以从另一模块中被引用为一个外部定义符号（参见 “3.5 (2) EXTBIT（外部位）”）。

作为在示例中汇编源模块的结果，以下汇编表将会被生成。

< 汇编列表 >

```
          汇编列表

 ALNO STNO ADRS OBJECT M | 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>

在汇编表的行(2)到(5)中，定义为名称的位值中的位地址值显示在目标代码字段中。

SET

(2)　SET (设置)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
名称	SET	绝对表达式	[；注释]

[功能]

- SET伪指令定义了在操作数字段中所指定的表达式的名称，该名称具有值和属性（符号属性和重置属性）。
- 以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。这些值和属性都是有效的一直到同一名称被定义。

[用途]

- 将用于源模块中的数字数据（变量）定义为一个名称，并描述代替数字数据的指令的操作数中的名称。
  如果你想在源模块中改变名称的值，通过再次使用SET伪指令可以为同一名称定义一个不同的值。

[说明]

- 绝对表达式必须在SET伪指令的操作数字段中进行描述。
- SET伪指令可以在源程序中的任何位置中进行描述。然而，已经由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名句中检测到错误，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。
- 以EQU伪指令定义的符号不能以SET伪指令重新定义。
  而以SET伪指令定义的符号也不能以EQU伪指令来重新定义。
- 不能定义一个位符号。

[应用示例]

```
          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”。值和属性都是有效的直至它们通过下列(3)中的SET伪指令进行重新定义。

(2)　名称“COUNT”的值“10H”被转移到寄存器B中。

(3)　名称“COUNT”的值被转变为“20H”。

(4)　名称“COUNT”的值“20H”被转移到寄存器B中。

3.4　内存初始化与区域保留伪指令

内存初始伪指令定义了用于源程序中的常量数据。

定义的常量数据的值被生成为目标代码。

区域保留伪指令保留用于程序中的内存区域。

以下的内存初始和区域保留伪指令为可用的。

- DB（定义字节）
- DW（定义字）
- DS（定义存储器）
- DBIT（定义位）

DB

(1) DB (字义字节)

[Description Format]

符号字段	助记符字段	操作数字段	注释字段
[标签:]	DB	(尺寸)	[; 注释]
[标签:]	DB	或 初始值[, ...]	[; 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。要进行初始化的字节的个数可以被指定为"尺寸"。
- 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数字段中所指定的初始值。

[用途]

- 当定义在程序中所使用的表达式或字符串时将使用DB伪指令。

[说明]

- 如果操作数字段中的值被加上括号，那么汇编器将假设指定了一个尺寸。否则将会假设一个初始值。
- DB伪指令不能在位段中进行描述。

(1) 带有尺寸规范：

(a) 如果在操作数字段中指定了一个尺寸，那么汇编器将会初始化一个区域，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尺寸。如果尺寸描述是非法的，那么汇编器将输出一个错误消息，且不会执行初始化。

(2) 带有初始值规范：

(a) 以下两个参数可以被指定为初始值：

(i) 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 到0FFH的范围内。如果值超过了8位，汇编器将只使用值中的低8位作为有效数据并输出一个错误消息。

(ii) 字符串

如果一个字符串被描述为操作数，那么一个8位ASCII码将为串中的每个字符而保留。

- 两个或更多的初始值可以在DB伪指令的语句行中进行指定。
- 包含一个浮动符号或外部引用符号的表达式可以被描述为一个初始值。

[应用示例]

```
            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)　由于尺寸已经被指定，因此汇编器将会对每个带有值“00H”的字节区域进行初始化。

(2)　6字节区域可用字符串‘ABCDEF’来初始化。

(3)　3字节区域可用“0AH, 0BH, 0CH”来初始化。

(4)　4字节区域可用“00H”来初始化。

(5)　由于表达式‘AB’ +1=4143H (4142H+1)的值超过0到0FFH的范围，因此该描述将会产生一个错误。

DW

(2) DW (定义字)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[标签:]	DB	(尺寸)	[; 注释]
	或		
[标签:]	DB	初始值[, ...]	[; 注释]

[功能]

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

[用途]

- 当定义一个16位的数字常量，例如在程序中所使用的地址或数据时，将会使用DW伪指令。

[说明]

- 如果操作数字段中的值被加上括号，那么汇编器将假设指定了一个尺寸。否则将会假设一个初始值。
- DW伪指令不能在位段中进行描述。

(1) 带有尺寸规范：

(a) 如果在操作数字段中指定了一个尺寸，那么汇编器将会初始化一个区域，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尺寸。如果尺寸描述是非法的，那么汇编器将输出一个错误消息，且不会执行初始化。

(2) 带有初始值规范：

(a) 以下两个参数可以被指定为初始值：

(i) 常量

16位或小于16位

(ii) 表达式

表达式的值必须被保存为一个16位数据。

没有字符串可以被描述为一个初始值。

- 指定的初始值中的上2位被保存在HIGH地址中，值的低2位则被保存在LOW地址中。
- 两个或更多的初始值可以在DW伪指令的语句行中进行指定。
- 包含一个浮动符号或外部引用符号的表达式可以被描述为一个初始值。

[应用示例]

```
                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>

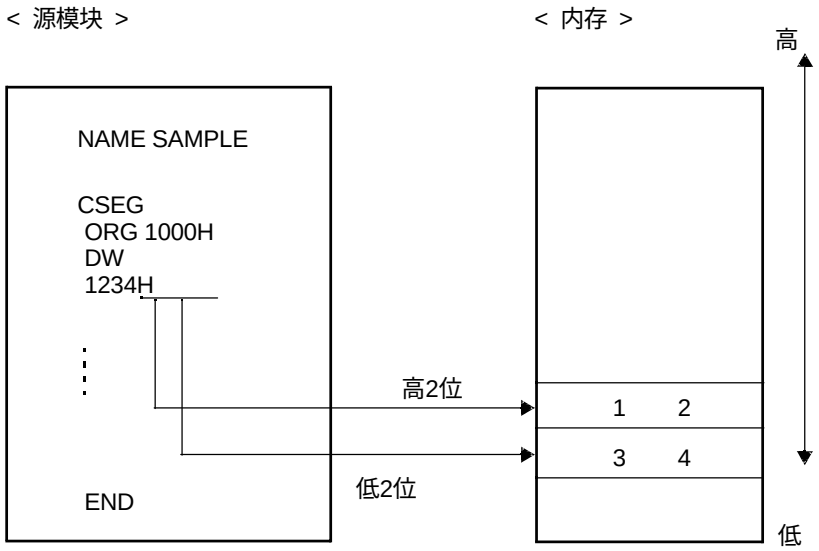
(1)　由于尺寸已经被指定，因此汇编器将会对每个带有值“00H”的字进行初始化。

(2)　矢量输入地址可用DW伪指令来定义。

(3)　一个2字区域可用值“34127856”来初始化。

备注　　内存中的HIGH地址通过字值中的高2位来初始化，内存中的LOW地址则通过字值中的低2位来初始化。

< 例 >



DS

(3)　DS (定义存储器)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[标签:]	DS	绝对表达式	[; 注释]

[功能]

- DS伪指令命令汇编器为在操作数字段中所指定的字节个数保留一个内存区。

[用途]

- DS伪指令主要用于保留在程序中使用的内存（RAM）区。如果指定了一个标签，那么保留的内存区中第一地址的值将被赋于标签。在源模块中该标签用于描述对内存的操作。

[说明]

- 用这个DS伪指令所保留的区域中的内容是未知的（不确定的）。
- 指定的绝对表达式将用无符号数的16位来计算。
- 当操作数的值为”0”时，没有区域可以保留。
- DS伪指令不能在位段中进行描述。
- 以DS伪指令定义的符号（标签）只能进行反向引用。
- 只有以下的从绝对表达式中扩展而来的参数可以在操作数字段中进行描述：

(1)　一个常量

(2)　可以在其中执行运算的带有常量的表达式（常量表达式）

(3)　使用一个常量或常量表达式定义的EQU符号或SET符号

(4)　带有地址属性的表达式1－带有地址属性的表达式2

如果在“带有地址属性的表达式1”中的标签1以及在“带有地址属性的表达式2”中标签2都是浮动的，那么这两个标签必须在同一段中进行定义。

然而，在以下两种情况中的任一情况下将产生一个错误：

(5)　如果标签1和标签2属性同一段，且如果目标代码的字节个数不能确定的BR伪指令在两个标签中进行描述时

(6)　如果标签1和标签2在段中不相同，且如果目标代码的字节个数不能确定的BR伪指令在任一标签和标签所属的段的开始位置间进行描述时

(7)　要在其中执行运算的上述<1>至<4>的表达式中的任一表达式。

- 以下参数不能在操作数字段中进行描述：

(1)　外部引用符号

(2)　使用EQU伪指令定义了“带有地址属性的表达式1－带有地址属性的表达式2”的符号

(3)　位置计数器($)以“带有地址属性的表达式1－带有地址属性的表达式2”的形式在表达式1或表达式2中进行描述。

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

[应用示例]

```
            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)　保留一个10字节的工作区域，但该区域的内容是未知的（不确定的）。标签“TABLE1”被分配到地址的开始位置。

(2)　保留1字工作区域。

(3)　保留2字工作区域。

DBIT

(4)　DBIT (定义位)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[名称]	DBIT	无	[；注释]

[功能]

- DBIT伪指令命令汇编器在一个位段内保留一个1字内存区。

[用途]

- 使用DBIT伪指令在一个位段内保留一个位区。

[说明]

- DBIT伪指令只在一个位段中进行描述。
- 使用DBIT伪指令保留的1字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，那么该名称将具有一个地址和一个位位置，并将它们作为它的值。
- 定义的名称可以在需要saddr.位的位置进行描述。

[应用示例]

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

        CSEG
        SET1        BIT1        ; (2)
        CLR1        BIT2        ; (3)
        END
```

<说明>

(1)　通过这三个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)，其中每个名称都具有一个地址和一个位位置，并将其作为它的值。

(2)　该描述相当于“SET1 saddr.bit”，并且描述了位区中的名称“BIT1”，其中位区在上述(1)中被保留为操作数“saddr.bit”。

(3)　该描述相应　于“CLR1 saddr.bit”，并且将名称“BIT2”描述为“saddr.bit”。

3.5　　联接伪指令

连接伪指令阐明了引用在其他模块中定义的符号的相对性。

考虑一种情况，在这种情况中一个程序通过被分为两种模块：模块1和模块2来创建。在模块1中，当在模块2中定义的符号被引用时，符号在模块中没有声明时不能被使用。由于这个原因，一些种类的信号或指令，例如“我想使用符号”或“你可以使用符号”需要在两个模块间发出。

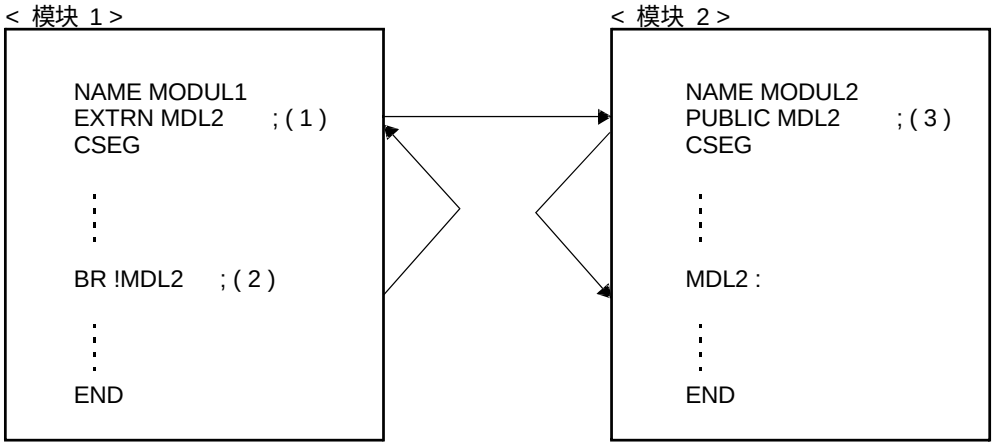
在模块1中，符号的外部引用说明被发出，用来表示在其他模块中定义的符号必须被引用。在模块2中，符号的外部定义说明被发出，用来表示在其他模块中定义的符号可以被引用。

当有效地生成外部引用说明和外部定义说明时，符号可以被首次引用。

用于建立这种相互关系的连接伪指令功能在以下两种类型中有效：

- 用于说明符号的外部定义：PUBLIC（公共的）伪指令
- 用于说明符号的外部引用：EXTRN（外部） 和 EXTBIT（外部位） 伪指令

图3-6　在两个模块间的符号的关系



在图3-6的模块1中，在模块2中定义的符号“MDL2”在(2)中被引用。因此，在(1)中符号通过使用EXTRN伪指令被定义为外部引用。

在模块2中，从模块1中引用的符号“MDL2”在(3)中通过使用PUBLIC伪指令被定义为一个外部定义。

连接器会检查符号的这个外部引用是否相当于符号的外部定义。

以下的连接伪指令是可用的。

- EXTRN（外部的）
- EXTBIT（外部位）
- PUBLIC（公共的）

EXTRN

(1)　EXTRN (外部的)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[标签:]	EXTRN	符号名[　, ...]	[; 注释]

[功能]

- EXTRN伪指令向连接器说明了在其他模块中的符号（不同于位符号）将在该模块中进行引用。

[用途]

- 当引用在其他模块中定义的符号时，EXTRN伪指令必须用于将符号定义为一个外部引用。

[说明]

- EXTRN伪指令可以在源程序中的任何位置中进行描述（参见“2.1　源程序的基础结构”）。
- 通过使用逗号(,)将每个符号名隔开的操作数字段中最多可以指定20个符号。
- 当引用一个具有位值的符号时，必须通过使用EXTBIT伪指令将符号定义为一个外部引用。
- 以EXTRN伪指令说明的符号必须使用PUBLIC伪指令在另一模块中进行说明。
- 没有宏名可以被描述EXTRN伪指令的操作数（参见相对于宏名的“第5章　宏”）。
- EXTRN伪指令在整个模块中只允许一个相对于符号的EXTRN说明。对于符号的第二个以及之后的EXTRN说明，连接器将输出一个警告消息。
- 已经被说明的符号不能描述为EXTRN伪指令的操作数。相反地，一个已经被定义为EXTRN的符号不能以任何其他伪指令来重新定义或说明。
- 以EXTRN伪指令定义的符号可以用于saddr区域的引用。

[应用示例]

< 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)　该EXTRN将(2)和(3)中引用的“SYM1” 和 “SYM2”定义为外部引用。

(2)　该DW指令引用符号“SYM1”。

(3)　该MOV指令引用符号“SYM2”并输出一个用于引用saddr区的代码。

(4)　符号“SYM1” 和 “SYM2”被定义为外部定义。

(5)　符号“SYM1”被定义。

(6)　符号“SYM2”被定义。

EXTBIT

(2)　EXTBIT (外部位)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[标签:]	EXTBIT	位符号名[, ...]	[; 注释]

[功能]

- EXTBIT伪指令向连接器说明了在其他模块中具有saddr.位值的的位符号将会该模块中被引用。

[用途]

- 当引用一个具有位值的且已经在其他模块中被定义的符号时，EXTBIT伪指令必须被用于将符号定义为一个外部引用。

[说明]

- EXTBIT可以在源程序中的任何位置中进行描述。
- 通过使用逗号(,)将每个符号名隔开的操作数字段中最多可以指定20个符号。
- 以EXTBIT伪指令说明的符号必须使用PUBLIC伪指令在另一模块中进行说明。
- EXTBIT伪指令在整个模块中只允许一个相对于符号的EXTRN说明。对于符号的第二个以及之后的EXTRN说明，连接器将输出一个警告消息。

[应用示例]

< 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)　该EXTBIT伪指令说明了引用为外部引用的符号“FLAG1”和“FLAG2”。两个或更多的符号可以在操作数字段中进行描述。

(2)　该SET1指令引用了符号“FLAG1”。该描述相当于“SET1 saddr.bit”。

(3)　该CLR1指令引用了符号“FLAG2”。该描述相当于“CLR1 saddr.bit”。

(4)　该PUBLIC伪指令定义了符号“FLAG1” 和“FLAG2”。

(5)　该DBIT将符号“FLAG1”定义为SADDR区域中的位符号。

(6)　该DBIT将符号“FLAG2”定义为SADDR区域中的位符号。

PUBLIC

(3) PUBLIC (公共的)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[标签:]	PUBLIC	符号名[, ...]	[; 注释]

[功能]

- PUBLIC伪指令向连接器说明了在操作数字段中描述的符号是将要从其他模块中引用的符号。

[用途]

- 当定义一个将要从其他模块中引用的符号（包括位符号）时，PUBLIC伪指令必须用于将该符号定义为一个外部定义。

[说明]

- PUBLIC伪指令可以在源程序中的任何位置进行描述。
- 通过使用逗号(,)将每个符号名隔开的操作数字段中最多可以指定20个符号。
- 要在操作数字段中描述的符号必须在同一模块内进行定义。
- PUBLIC伪指令在整个模块中只允许一个相对于符号的PUBLIC说明。对于符号的第二个以及之后的PUBLIC说明，连接器将输出一个警告消息。
- 以下符号不能被用途PUBLIC伪指令的操作数：
  (1) 以DET伪指令定义的名称
  (2) 以EXTRN或EXTBIT伪指令在同一模块中定义的符号
  (3) 段名
  (4) 模块名
  (5) 宏名
  (6) 没有在模块内定义的符号
  (7) 使用EQU伪指令定义了带有位属性的操作数的符号
  (8) 使用EQU伪指令定义了一个sfr的符号（然而，sfr区域和saddr区域重叠的区域被排除）

[应用示例]

- 由三个模块组成的程序的示例

< 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
```

< Module 3 >

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

<说明>

(1)　该PUBLIC伪指令说明了符号“A1”和“A2”将会从其他模块中进行引用。

(2)　该PUBLIC伪指令说明了符号“B1”将会从另一个模块中进行引用。

(3)　该PUBLIC伪指令说明了符号“C1”将会从另一个模块中进行引用。

3.6　目标模块名说明伪指令

目标模块名说明伪指令将一个模块名赋于一个由RA78K0汇编器创建的目标模块。

以下的目标模块名定义伪指令是可用的。

- NAME (命名)

NAME

(1)　NAME (命名)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[标签:]	NAME	目标模块名	[; 注释]

[功能]

- NAME伪指令将在操作数字段中描述的目标模块名赋于由汇编器输出的目标模块。

[用途]

- 在使用调试器进行符号调试时，每个目标模块都需要一个模块名。

[说明]

- NAME伪指令可以在源程序中的任何位置进行描述。
- 关于模块名描述的约定，请参见在“2.2.3　符号字段”中的符号描述约定。
- 可以指定为模块名的字符是那些不同于“ (”,“ (28H)”, “)” 、 “ (29H)”或日文字符的并且由汇编软件的操作系统所允许的字符。
- 没有模块名可以被描述为不同于NAME的伪指令的操作数，或指令的操作数。
- 如果省略了NAME伪指令，汇编器将假设输入源模块文件的主要名称（前8个字符）为模块名。在Windows版本中，主要名称被转换成大写字母以用于检索。如果指定了两个或更多的模块名，汇编器将输出一个错误消息并忽略第二个和之后的模块名说明。
- 在操作数字段中描述的模块名不能超过8个字符。
- 符号名中的大写和小写字符被区分。

[应用示例]

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

        CSEG
        MOV     A , B

        END
```

<说明>

(1)　该NAME伪指令将“SAMPLE”定义为模块名。

3.7　自动转移指令选择伪指令

无条件转移指令直接将转移目的地址描述为它们的操作数。两个这类的指令，“BR !addr16” 和 “BR $addr16”，可以使用。这些指令依照转移目的地址的范围来选择并使用最合适的操作数。由于每个的伪指令的字节的个数均不同，因此为了创建一个具有高内存利用效率的程序，将需要使用带有最小字节个数的指令。然而当描述转移指令时，考虑该地址范围将是非常麻烦的。

因为这个原因，对于伪指令来说将有必要依照转移目的地址的范围来直接指示汇编器去自动选择2字节的或3字节的转移指令。这个被称为自动转移指令选择伪指令。

以下的自动转移指令选择伪指令是可用的。

- BR (转移)

BR

(1) BR (转移)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[标签:]	BR	表达式	[; 注释]

[功能]

- BR伪指令命令汇编器依照在操作数字段中所指定的表达式值的范围来自动选择一个2字节或3字节的BR转移指令，并生成一个适用于所选指令的目标代码。

[用途]

- 如果转移目的地在与BR伪指令相邻的地址中的–80H到+7FH的范围内，那么将可以描述2字节转移指令“BR $addr16”。与使用3字节转移指令“BR !addr16”相比，使用这个指令时，所需的内存空间可以减少一个字节。为了创建一个带有高内存利用效率的程序，应积极使用2字节转移指令。

  然而当描述转移指令时，考虑转移目的地址的范围将是非常麻烦的。因此，如果不清楚是否可以描述2字节转移指令时应使用BR伪指令。
- 如果明确你可以描述一个2字节或3字节转移指令，那么应描述适用的指令。这与描述BR伪指令相比将缩短汇编的时间。

[说明]

- BR伪指令只可以在一个代码段内使用。
- 直接转移目的地可以被描述为BR伪指令的操作数。“$”在表达式开始位置的当前位置计数器不能被描述。
- 为了进行优化，必须满足以下条件。

  (1) 在表达式中只有1个标签或正向引用符号。

  (2) 不描述带有地址属性的EQU符号。

  (3) 不描述为“带有地址属性的表达式1－带有地址属性的表达式2”定义的EQU符号。

  (4) 不描述带有地址属性的表达式，其中在表达式上已经执行了HIGH/LOW/DATAPOS/BITPOS运算符。

  如果不符合这些条件，则将选择3字节BR指令。

[应用示例]

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

<说明>

(1)　该BR伪指令将生成一个2字节转移指令(BR $addr16)，因为在该行与转移目的地间的位移在–80H到+7FH的范围内。

(2)　该BR伪指令将被3字节转移指令(BR　!addr16)所代替，因为在该行与转移目的地间的位移在–80H到+7FH的范围外。

3.8　宏伪指令

当你在描述一个源程序时，一次又一次的重复描述一系列频繁使用的指令组是非常麻烦的，且这可能会增加描述的个数或编码错误。

通过宏伪指令来使用宏功能时，将可以消除重复描述同一组指令的必要，从而增加程序的编码效率。宏的基础功能是用一个名称来代替一系列的语句。

宏伪指令包括MACRO（宏），LOCAL（局部），REPT（重复），IRP（不定重复），EXITM（退出宏），和ENDM（结束宏）。

在这节中，将详细说明每个宏伪指令。关于宏功能的详细说明，参见 第 5 章　宏。

以下的宏伪指令是可用的。

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

MACRO

(1)　MACRO (宏)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
宏名	MACRO : 宏体 : ENDM	[形式参数[, ...]]	[; 注释] [; 注释]

[功能]

- 宏伪指令通过将在符号字段中指定的宏名赋于在该伪指令和ENDM伪指令间所描述的一系列语句（称为宏体）来执行宏定义。

[用途]

- 以一个宏名来定义在源程序中一系列频繁使用的语句。在定义后将只描述所定义的宏名（参见“5.2.2　宏引用”），并且相应于宏名的宏体将会被扩展。

[说明]

- 宏伪指令必须与ENDM伪指令配套使用。
- 关于在符号字段中所描述的宏名，请参见在“2.2.3　符号字段”中的符号描述约定。
- 要引用一个宏，则在助记符字段中描述已经被定义的宏名。
- 对于在操作数字段中描述的形式参数来说，将会应用与符号描述约定相同的规则。
- 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。
- 如果任一保留字被描述为一个形式码数，则将会产生一个错误。然而，如果描述了一个用户定义符号，那么它被当作形式参数时将会具有优先级。
- 形式参数的个数必须与实际参数的个数相同。
- 如果宏体通过LOCAL伪指令来说明，那么宏体内所定义的名称或标签对于之前的宏扩展将会有是效的。
- 宏嵌套（也就是在宏体内的其他宏）最高允许有8个级别，包括REPT和IRP伪指令。
- 可以在单一源模块中定义的宏的个数没有特别的限制。在其它字中，只要有可用的内存空间，宏都可以被定义。
- 形式参数定义行，引用行和符号名不会被输出到交叉引用表。

- 不能在宏体内定义两个或更多的段。如果定义了两个或更多的段，则将会输出一个错误。

[应用示例]

```
                NAME            SAMPLE

ADMAC       MACRO       PARA1，PARA2    ；(1)
                MOV         A，#PARA1
                ADD         A，#PARA2
                ENDM
                                ;     (2)     ADMAC
                10H，20H                        ；(3)
                END
```

<说明>

(1)　宏通过指定一个宏名“ADMAC”和两个形式参数“PARA1” 和“PARA2”来定义。

(2)　该伪指令表示宏定义结束。

(3)　宏“ADMAC”被引用。

LOCAL

(2)　LOCAL (局部)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
无	LOCAL	符号名[, ...]	[; 注释]

[功能]

- LOCAL伪指令说明了在操作数字段中指定的符号名是一个仅在宏体内有效的局部符号。

[用途]

- 如果一个在宏体内定义了符号的宏被不只一次的引用，那么汇编器将会输出两倍的符号定义错误。通过使用LOCAL伪指令，你可以不只一次地引用（或调用）一个在宏体内定义了符号的宏。

[说明]

- 关于在操作数字段中定义的符号名的约定，请参见在“2.2.3　符号字段”中的符号描述约定。
- 在每个宏扩展中，定义为LOCAL的符号都将会被一个符号“??RAn”（其中n = 0000到 FFFF)所代替。在宏替换后的符号“??RAn”将会按与全局符号相同的方式来处理，并会被保存在符号表中，且可以在符号名“??RAn”下被引用。
- 如果符号在宏体内被描述且宏被多次引用，这将意味着符号将在源模块中将会被定义多次。因为这个原因，则将有必要说明符号是一个仅在宏体内有效的局部符号。
- LOCAL伪指令只能在一个宏定义内使用。
- LOCAL伪指令必须在使用操作数字段中所指定的符号前被描述（换句话说，LOCAL伪指令必须在宏体的开始位置进行描述）。
- 在源模块内通过LOCAL伪指令所定义的符号名均不能相同（换句话说，同一名称不能用于在每个宏中所使用的局部符号）。
- 只要局部符号都在一行内，那么可以在操作数字段中指定的局部符号的个数将不会受限制。然而，宏体内符号的个数限制为64个。如果定义了65个或更多的局部符号，汇编器将会输出一个错误消息并将宏定义保存为一个空的宏体。此时即使宏被调用也不会有内容被扩展。
- 以LOCAL伪指令定义的宏不能被嵌入。
- 以LOCAL伪指令定义的符号不能从宏外来调用（引用）。
- 在操作数字段中，没有保留字可以被描述为符号名。然而，如果描述了一个与用户定义符号的符号名相同的符号名，那么它被当作一个局部符号时将会具有优先级。

- 定义为LOCAL伪指令的操作数的符号不会被输出到交叉引用表和符号表列表中。
- 宏扩展时，LOCAL伪指令的语句行不会被输出。
- 如果在宏定义内生成了一个LOCAL说明，其中对于宏定义来说符号具有与该宏定义的形式参数相同的名称，那么将会产生一个错误。

[应用示例]

```
          NAME      SAMPLE
              ; 宏定义
MAC1    MACRO
          LOCAL LLAB                    ; (1)
LLAB :                                  ;
          BR $LLAB                      ; (2)
          ENDM                            ;
              ; Source text
REF1 :     MAC1                         ; (3)

          BR !LLAB                      ; (4)          <--该描述不正确。

          REF2 :     MAC1               ; (5)

          END
```

<说明>

(1) 该LOCAL伪指令将符号名“LLAB”定义为一个局部符号。

(2) 该BR指令在宏MAC1内引用符号“LLAB”。

(3) 该宏引用调用了宏MAC1。

(4) 由于局部符号“LLAB”是从宏MAC1定义外引用的，因此该描述将会导致一个错误。

(5) 该宏引用调用了宏MAC1。

以下显示了上述应用示例的汇编列表。

< 汇编列表 >

```
汇编列表

 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 ) 未定义的符号引用' LLAB '
*** ERROR E2303 , STNO 13 ( 13 ) 非法表达式
  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)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[标签:]	REPT	绝对表达式	[；注释]
	:		
	ENDM		[；注释]

[功能]

- REPT伪指令命令汇编器重复扩展在该伪指令和ENDM伪指令（被称为REPT-ENDM模块）间所描述的一系列语句，重复的次数等同于在操作数字段中所指定的表达式的值。

[用途]

- 使用REPT和ENDM伪指令来重复描述在源程序中的一系列的语句。

[说明]

- 如果REPT伪指令和ENDM伪指令不配套，则将会产生一个错误。
- 在REPT-ENDM模块中，宏引用，REPT伪指令和IRP伪指令可以最多被嵌套8个层次。
- 如果在REPT-ENDM模块中出现EXITM伪指令，将终止随后的通过汇编器实现的REPT-ENDM模块的扩展。
- 汇编控制指令可以在REPT-ENDM模块中进行描述。
- 宏定义不能在REPT-ENDM模块中进行描述。
- 在操作数字段中描述的绝对表达式以无符号数的16位数来计算。如果表达式的值为0，那么没有内容会被扩展。

[应用示例]

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

<说明>

(1)　该REPT伪指令命令汇编器将REPT-ENDM模块连续扩展三次。

(2)　该伪指令表示REPT-ENDM块结束。

当对上述源程序进行汇编时，REPT-ENDM模块将会被扩展为如下列汇编列表中所显示的那样：

<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模块已经被扩展了三次。在汇编列表中，由源模块中的REPT伪指令所定义的定义语句(1)和(2)将不会被显示。

IRP

(4)　IRP (不定重复)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[标签:]	IRP	形式参数, <[实际参数[, ...]]>	[; 注释]
	:		
	ENDM		[; 注释]

[功能]

- IRP伪指令命令汇编器重复扩展在该伪指令和ENDM伪指令（被称为IRP -ENDM模块）间所描述的一系列语句，重复的次数等于以操作数字段中指定的实际参数代替形式参数时的实际参数的个数。

[用途]

- 使用IRP和ENDM伪指令在源程序中重复描述一系列语句，其中只有一些语句变成变量。

[说明]

- IRP伪指令必须与ENDM伪指令配套。
- 在操作数字段中最多可以描述16个实际参数。
- 在IRP -ENDM模块中，宏引用，REPT伪指令和IRP伪指令可以最多被嵌套8个层次。
- 如果在IRP -ENDM模块中出现EXITM伪指令，将终止随后的通过汇编器实现的IRP -ENDM模块的扩展。
- 宏定义不能在IRP-ENDM模块中进行描述。
- 汇编控制指令可以在IRP -ENDM模块中进行描述。

[应用示例]

```
NAME    SAMP1
CSEG

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

<说明>

(1)　形式参数为“PARA”，实际参数则是以下三种：“0AH”, “0BH”，和 “0CH”。当以实际参数“0AH”, “0BH”，和“0CH”来代替形式参数“PARA”时，该IRP伪指令命令汇编器将IRP-ENDM模块扩展三次（也就是实际参数的个数）。

(2)　该伪指令表达IRP-ENDM模块结束。

当对上述源程序进行汇编时，IRP -ENDM模块将会被扩展为如下列汇编列表中所显示的那样：

< 汇编列表>

```
NAME     SAMP1
CSEG
         ; IRP-ENDM  模块
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模块已经被扩展三次（等于实际参数的个数）。

(3)　在该ADD指令中，PARA被0AH所代替。

(4)　在该ADD指令中，PARA被0BH所代替。

(5)　在该ADD指令中，PARA被0CH所代替。

EXITM

(5)　EXITM (退出宏)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
[标签:]	EXITM	无	[; 注释]

[功能]

- EXITM伪指令强行终止由MACRO伪指令所定义的宏体的扩展以及REPT-ENDM模块和IRP-ENDM模块的重复扩展。

[用途]

- 当在由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，将主要使用该功能。
- 如果条件汇编功能与宏体的的其他指令结合使用，那么源程序中不能进行汇编的部分将可能被汇编，除非通过强行使用EXITM伪指令使得控制退出了宏。在这种情况下，应确保使用EXITM伪指令。

[说明]

- 如果在宏体中描述了EXITM伪指令，那么相当于ENDM伪指令的指令将会被保存为宏体。
- EXITM伪指令表示仅在宏扩展中结束宏。
- 如果一些内容在EXITM伪指令的操作数字段中被描述，那么汇编器将会输出一个错误消息，但不会执行EXITM处理。
- 如果在宏体中出现EXITM伪指令，那么当汇编器进入宏体时，它将强行将IF/_IF/ELSE/ELSEIF/_ELSEIF/ENDIF模块的嵌入层返回到层中。
- 如果在INCLUDE文件中出现EXITM伪指令中，其中INCLUDE文件是通过扩展在宏体中所描述的INCLUDE控制指令所得到的，那么汇编器将会接受EXITM伪指令并认为它是有效的，然后在该层终止宏扩展。

[应用示例]

- 在这个例子中，条件汇编控制指令被使用。参见“4.7　条件汇编控制指令”。
- 关于宏体和宏扩展，请参见“第5章　宏”。

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

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

<说明>

(1)　宏“MAC1”在宏体内使用条件汇编功能(2)以及(4)到(8)。

(2)　此处定义了一个用于条件汇编的IF模块。如果开关名“SW1”是真的（非”0”），那么ELSE模块将会被汇编。

(3)　该伪指令在（4）和随后的条件汇编功能中强行终止宏体的扩展。
如果该EXTIM伪指令被省略，那么当宏被扩展时汇编器将继续在(6)和随便的条件汇编功能中执行汇编。

(4)　此处定义了一个用于条件汇编的ELSE模块。如果开关名“SW1”是假的(“0”)，那么将对ELSE模块进行汇编。

(5)　该ENDIF控制指令表示条件汇编结束。

(6)　此处定义了另一个用于条件汇编的IF模块。如果开关名“SW2”是真的（非”0”），那么将对以下IF模块进行汇编。

(7)　此处还定义了另一个用于条件汇编的ELSE模块。如果开关名“SW2”是假的(“0”)，那么将对ELSE模块进行汇编。

(8)　该ENDIF指令表示在(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)　该SET控制指令将真值赋于开关名“SW1”并设置条件汇编的条件。

(11) 该宏引用调用了宏“MAC1”。

当上例中的源程序被汇编时，将会发生如下的宏扩展。

< 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
```

宏“MAC1”的宏体通过参考在(11)中的宏来进行扩展。由于真值被设置到(10)中的开关名“SW1”中，因此将对在宏体中的第一个IF模块进行汇编。由于EXITM伪指令在IF模块的结尾被描述，因此将不会执行随后的宏扩展。

ENDM

(6) ENDM (结束宏)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
无	ENDM	无	[；注释]

[功能]

- ENDM伪指令指示汇编器终止执行定义为宏功能的一系列语句。

[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]

- 在MACRO伪指令和ENDM伪指令间描述的一系列语句将变成一个宏体。
- 在REPT伪指令和ENDM伪指令间描述的一系列语句将变成一个REPT-ENDM模块。
- 在IRP伪指令和ENDM伪指令间描述的一系列语句将变成一个IRP -ENDM模块。

[应用示例]

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

汇编终止伪指令将通知汇编器源模块结束。该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。

汇编器将相当于汇编终止伪指令的一系列语句当作一个源模块来处理。因此，如果在REPT模块或IRP模块中的ENDM前存在汇编终止伪指令，那么REPT模块或IRP模块都将变为无效。

以下的汇编终止伪指令是可用的。

- END（ 结束 ）

END

(1)　END (结束)

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
无	END	无	[；注释]

[功能]

- EDN伪指令向汇编器显示源模块结束。

[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]

- 汇编器继续对源模块进行汇编直至在源模块中出现END伪指令。因此，END伪指令在每个源模块的结尾处是必需的。
- 在END伪指令后始终输入一个换行（LF）码。
- 如果在END伪指令后出现任何不同于空格，制表符，LF或注释的语句，汇编器将输出一个警告消息。

[应用示例]

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

<说明>

(1)　始终在每个源模块的结尾处描述END伪指令。

第 4 章　控制指令

本章节说明了控制指令。控制指令提供了关于汇编器的运算的详细说明。

4.1　控制指令概述

在源程序中描述的控制指令提供了关于汇编器的运算的详细说明。

这些指令不以目标代码的生成为条件。

控制指令在以下类型中可用：

表4-1　控制指令列表

控制指令的类型	控制指令
处理器类型指定控制指令	PROCESSOR
调试信息输出控制指令	DEBUG / NODEBUG , DEBUGA / NODEBUGA
交叉引用列表输出规范控制指令	XREF / NOXREF , SYMLIST / NOSYMLIST
包含控制指令	INCLUDE
汇编列表控制指令	EJECT , TITLE , SUBTITLE ,LIST / NOLIST ,GEN / NOGEN ,COND / NOCOND ,FORMFEED / NOFORMFEED ,WIDTH , LENGTH , TAB
条件汇编控制指令	SET / RESET ,IF / _IF / ELSEIF / _ELSEIF / ELSE / ENDIF
其他控制指令	DGL , DGS , TOL_INF

控制指令在源程序中以和汇编导引相同的方式进行描述。

控制指令列于表4-1中，以下指令与可以在汇编器的启动命令行中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间的对应关系显示在表4-2中。

表4-2　控制指令和汇编选项

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
FORMFEED / NOFORMFEED	-LF / -NLF
TITLE	-LH
WIDTH	-LW
LENGTH	-LL
TAB	-LT

关于通过命令行来指定控制指令和汇编选项的方法，请参见RA78K0　汇编器包操作用户手册。

4.2　处理器类型指定控制指令

处理器类型指定控制指令在源模块文件中指定用于汇编的目标设备的类型。

以下的处理器类型指定控制指令是可用的。

- PROCESSOR (处理器)

PROCESSOR

(1)　PROCESSOR (处理器)

[格式描述]

```
[ ∆ ] $ [ ∆ ] PROCESSOR [ ∆ ]( [ ∆ ] processor type [ ∆ ] )
[ ∆ ] $ [ ∆ ] PC [ ∆ ] ( [ ∆ ] processor type [ ∆ ] )                    ; 简略格式
```

[功能]

- PROCESSOR控制指令在源模块文件中指定用于汇编的目标设备处理器的类型。

[用途]

- 用于汇编的目标设备的处理器类型必须始终在源模块文件或汇编器的启动命令行中进行指定。
- 如果你在每个源模块文件中省略了用于汇编的目标设备处理器类型的指定，那么你将必须在每次汇编运算时指定处理器的类型。因此，通过在每个源模块文件中指定用于汇编的目标设备，那么在启动汇编器时你将可以节省时间和不必要的麻烦。

[说明]

- PROCESSOR控制指令只能在源模块文件的节头中进行描述。如果在其他地方描述了控制指令，那么汇编器将被中止。
- 如果指定的处理器类型不同于用于汇编的实际目标设备，那么汇编器将被中止。
- 在模块头中只有一个PROCESSOR控制指令可以被指定。
- 用于汇编的目标设备处理器的类型也可以通过汇编器启动命令行中的汇编选项(-C)来指定。如果所指定的处理器类型在源模块文件和启动命令行中不相同，那么汇编器将输出一条警告消息，并优先执行启动命令行中的处理器类型的指定。
- 即使当汇编选项(-C)在启动命令行中已经被指定时，汇编器也会对PROCESSOR控制指令执行语法检查。
- 如果处理器类型既没有在源模块文件中被指定也没有在启动命令行中被指定，那么汇编器将会被终止。

[应用示例]

```
$       PROCESSOR ( 014 )
$       DEBUG
$       XREF

        NAME    TEST
          :
        CSEG
```

4.3　调试信息输出控制指令

调试信息输出控制指令用于在源模块文件中指定调试信息输出或不输出到从源模块文件中创建的目标模块文件。以下的调试信息输出控制指令是可用的。

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

DEBUG / NODEBUG

(1)　DEBUG / NODEBUG (调试/不调试)

[格式描述]

[Δ] $ [Δ] DEBUG	; 默认假定
[Δ] $ [Δ] DG	; 简略格式
[Δ] $ [Δ] NODEBUG	
[Δ] $ [Δ] NODG	; 简略格式

[功能]

- DEBUG控制指令命令汇编器将局部符号信息添加到一个目标模块文件中。
- NODEBUG控制指令命令汇编器不将局部符号信息添加到一个目标模块文件中。然而，也是在这种情况下，段名将会被输出到一个目标模块文件中。
- “局部符号信息”也是就不同于模块名和PUBLIC，EXTRN和EXTBIT的符号。

[用途]

- 在执行包含局部符号的符号调试时使用DEBUG控制指令。
- 在以下情况下使用NODEBUG控制指令：
  (1)　仅对全局符号执行符号调试
  (2)　没有符号时执行调试
  (3)　只需要对象（对于使用PROM的计算）

[说明]

- DEBUG或NODEBUG控制指令只能在源模块文件的节头中进行描述。
- 如果省略了DEBUG或NODEBUG控制指令，汇编器将假设已经指定了DEBUG控制指令。
- 可以通过使用启动命令行中的汇编器选项(-G/-NG)来指定局部符号信息的增加。
- 如果源模块文件中控制指令的指定不同于启动命令行中的指定，那么在命令行中的指定将会优先。
- 即使当汇编选项(-NG)已经被指定时，汇编器也会对DEBUG 或NODEBUG控制指令执行语法检查。

DEBUGA / NODEBUGA

(2)　DEBUGA / NODEBUGA (debuga / nodebuga)

[格式描述]

[∆] $ [∆] DEBUGA	; 默认假定
[∆] $ [∆] NODEBUGA	

[功能]

- DEBUGA控制指令命令汇编器将汇编器源调试信息添加到一个目标模块文件中。
- NODEBUGA控制指令命令汇编器不将汇编器源调试信息添加到一个目标模块文件中。

[用途]

- 在汇编器或结构汇编预处理器的源级中执行调试时使用DEBUGA控制指令。在源级中进行调试时需要一个集成调试器。
- 在以下情况下使用NODEBUGA控制指令：
  (1)　没有汇编源级时执行调试
  (2)　只需要对象（对于使用PROM的计算）

[说明]

- DEBUGA或NODEBUGA控制指令只能在源模块文件的节头中进行描述。
- 如果省略了DEBUGA或NODEBUGA控制指令，汇编器将假设已经指定了DEBUGA控制指令。
- 如果指定了这些控制指令中的两个或更多的指令，那么最后指定的控制指令将优先于其他指令。
- 可以通过使用启动命令行中的汇编器选项(-GA/-NGA)来指定汇编源调试信息的增加。
- 如果源模块文件中控制指令的指定不同于启动命令行中的指定，那么在命令行中的指定将会优先。
- 即使当汇编选项(-NGA)已经被指定时，汇编器也会对DEBUGA 或NODEBUGA控制指令执行语法检查。
- 如果进行编译的或进行结构编译的调试信息由C语言汇编器或结构汇编预处理器来输出，那么当对输出汇编源进行汇编时，将不会描述调试信息输出控制指令。汇编所需的控制指令被当作控制语句，通过C语言汇编器或结构汇编预处理器输出到汇编源中。

4.4　交叉引用列表输出规范控制指令

交叉引用列表输出规范控制指令被用在源模块文件中用来指定交叉引用列表输出或不输出。

以下的交叉引用列表输出规范控制指令是可用的。

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

XREF / NOXREF

(1)　XREF / NOXREF (xref / noxref)

[格式描述]

[Δ] $ [Δ] XREF	
[Δ] $ [Δ] XR	; 简略格式
[Δ] $ [Δ] NOXREF	; 默认假定
[Δ] $ [Δ] NOXR	; 简略格式

[功能]

- XREF控制指令命令汇编器将一个交叉引用列表输出到一个汇编列表文件中。
- NOXREF控制指令命令汇编器不将一个交叉引用列表输出到一个汇编列表文件中。

[用途]

- 当你想知道关于在源模块文件中定义的每个符号是在哪里引用的，或在源模块文件中一共引用了多少符号的信息时，则使用XREF控制指令用来输出一个交叉引用列表。
- 如果你必须在每个汇编运算中指定交叉引用列表输出或不输出，那么通过在源模块文件中指定XREF和NOXREF控制指令将可以节省时间和劳力。

[说明]

- XREF或NOXREF控制指令只能在源模块文件的节头中进行描述。
- 如果指定了这些控制指令中的两个或更多的指令，那么最后指定的控制指令将优先于其他指令。
- 交叉引用列表输出或不输出也可以通过在启动命令行中的汇编选项(-KX/-NKX)来指定。
- 如果源模块文件中控制指令的指定不同于启动命令行中的指定，那么在命令行中的指定将会优先。
- 即使当汇编选项(-NP)已经在启动命令行中被指定时，汇编器也会对XREF/NOXREF控制指令执行语法检查。

SYMLIST / NOSYMLIST

(2)　SYMLIST / NOSYMLIST (symlist / nosymlist)

[格式描述]

```
[ Δ ] $ [ Δ ] SYMLIST
[ Δ ] $ [ Δ ] NOSYMLIST                                    ; 默认假定
```

[功能]

- SYMLIST控制指令命令汇编器将符号列表输出到列表文件中。
- NOSYMLIST控制指令命令汇编器不将符号列表输出到列表文件中。

[用途]

- 使用SYMLIST控制指令来输出一个符号列表。

[说明]

- SYMLIST或NOSYMLIST控制指令只能在源模块文件的节头中进行描述。
- 如果指定了这些控制指令中的两个或更多的指令，那么最后指定的控制指令将优先于其他指令。
- 符号列表的输出也可以通过启动命令行中的汇编选项(-KS / -NKS)来指定。
- 如果源模块文件中控制指令的指定不同于启动命令行中的汇编选项的指定，那么在命令行中的指定将优先于在源模块中的指定。
- 即使当汇编选项(-NP)已经在启动命令行中被指定时，汇编器也会对SYMLIST/NOSYMLIST控制指令执行语法检查。

4.5　包含控制指令

包含控制指令使用在源模块文件中，用来指定在源模块文件中的其他模块文件的包含。
通过有效使用该控制指令，你可以在描述源程序时节省时间和劳力。
以下的包含控制指令是可用的。

- INCLUDE (包含)

INCLUDE

(1)　INCLUDE (包含)

[格式描述]

```
[ ∆ ] $ [ ∆ ] INCLUDE [ ∆ ] ( [ ∆ ] filename [ ∆ ] )
[ ∆ ] $ [ ∆ ] IC [ ∆ ] ( [ ∆ ] filename [ ∆ ] )                    ; 简略格式
```

[功能]

- INCLUDE控制指令命令汇编器插入并扩展指定文件中的内容，该指定文件从用于汇编的源程序中的指定行开始。

[用途]

- 可以被两个或更多的源模块共享的语句中相对较大的组应该被组合为同INCLUDE文件一样的单一的文件。如果必须在每个源模块中使用语句组，那么则通过使用INCLUDE控制指令来指定所需的INCLUDE文件的文件名。使用该控制指令时，你可以大大减少描述源模块的时间和劳力。

[说明]

- INCLUDE控制指令只能在普通源程序中进行描述。
- INCLUDE文件的驱动名和路径名可以通过汇编选项(-I)来指定。
- 汇编器按以下顺序搜索INCLUDE文件的读取路径：

  (1)　不使用路径名来指定INCLUDE文件时

  　　(a)　存在源文件的路径

  　　(b)　通过汇编选项(-I)指定的路径

  　　(c)　通过环境变量INC78K0指定的路径

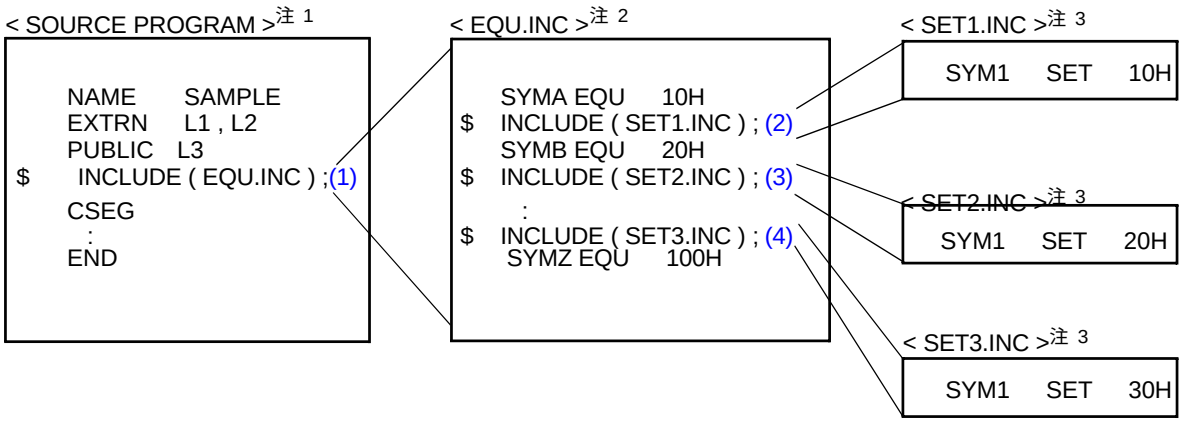
  (2)　使用路径名来指定INCLUDE文件时

  　　如果使用驱动名或以(\)开头的路径名来指定INCLUDE文件，那么以INCLUDE文件指定的路径将会作为INCLUDE文件名的前缀。如果使用相对路径（不以(\)开头）来指定INCLUDE文件，那么路径名将会按上述(a)中所描述的顺序来作为INCLUDE文件名的前缀。

- INCLUDE文件的嵌套最多允许有7个层次。换句话说，在汇编列表的INCLUDE文件中所显示的嵌套层最多为8个（此处的项"嵌套"也就是在INCLUDE文件中的一个或多个其他INCLUDE文件的指定）。
- END导引不需要在INCLUDE文件中进行描述。
- 如果不能打开指定的INCLUDE文件，组译顺将中止操作。
- INCLUDE文件必须以IF或_IF控制指令来结尾，其中IF或_IF控制指令在INCLUDE文件内与一个ENDIF指令配套。如果在INCLUDE文件扩展的入口处的IF层与INCLUDE文件扩展后的的IF层不一致，汇编器将会输出一条错误消息，并将IF层强行恢复成INCLUDE文件扩展入口处的IF层。

- 在INCLUDE文件中定义一个宏时，必须在INCLUDE文件中结束宏定义。如果在INCLUDE文件中出现一个意料外的ENDM导引（没有相应的MACRO导引），则将会输出一个错误并且忽略ENDM导引。如果对于在INCLUDE文件中描述的MACRO导引来说漏掉了ENDM导引，汇编器将会输出一条错误消息，但会通过假设已经描述了相应的ENDM导引来继续执行宏定义。

[应用示例]



注　1.　可以在源文件中指定两个或更多的$IC控制指令。同一INCLUDE文件可以被指定多次。

注　2.　可以为INCLUDE文件“EQU.INC”指定两个或更多的$IC控制指令。

注　3.　不能在任何INCLUDE文件“SET1.INC”, “SET2.INC”，和 “SET3.INC”中指定$IC控制指令。

<说明>

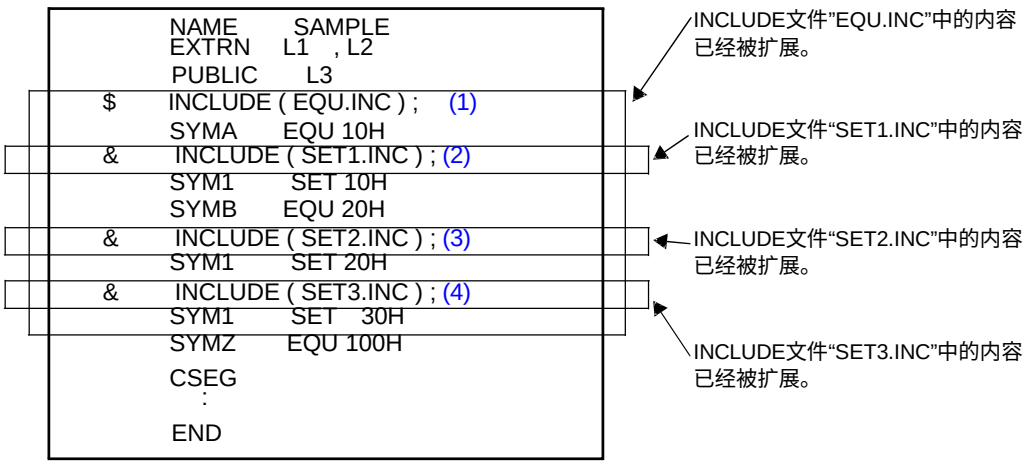
(1)　该指令可以将“EQU.INC”指定为INCLUDE文件。

(2)　该指令可以将“SET1.INC”指定为INCLUDE文件。

(3)　该指令可以将“SET2.INC”指定为INCLUDE文件。

(4)　该指令可以将“SET3.INC”指定为INCLUDE文件。

当该源程序被汇编时，INCLUDE文件中的内容将会按如下进行扩展：



4.6　汇编列表控制指令

汇编列表控制指令用在源模块文件中用于控制输出汇编列表的格式，例如页弹出，对列表输出的抑制以及输出小标题。

汇编列表控制指令包括：

- EJECT（弹出）
- LIST/NOLIST（列表/无列表）
- GEN/NOGEN（生成/不生成）
- COND/NOCOND（条件/无条件）
- TLTLE（标题）
- SUBTITLE（小标题）
- FORMFEED/NOFORMFEED（换页/不换页）
- WIDTH（宽）
- LENGTH（长）
- TAB（制表）

EJECT

(1)　EJECT（弹出）

[格式描述]

```
[ Δ ] $ [ Δ ] EJECT
[ Δ ] $ [ Δ ] EJ                                          ; 简略格式
```

[默认假设]

- EJECT控制指令不会被指定。

[功能]

- EJECT控制指令使得汇编器执行汇编列表的页弹出（换页）。

[用途]

- 在需要汇编列表页弹出的源模块的行中对EJECT控制指令进行描述。

[说明]

- EJECT控制指令只能在普通的源程序中进行描述。
- 在输出EJECT控制指令本身的映像后执行汇编列表的页弹出。
- 如果在启动命令行中指定了汇编选项(-NP)或(-LLO)，或者通过另一控制指令禁止了汇编列表输出，那么EJECT控制指令将变为无效的。关于这些汇编选项，请参见RA78K0系列的汇编器包操作用户手册。
- 如果在EJECT控制指令后接有非法描述，汇编器将输出一条错误消息。

[应用示例]

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

<说明>

(1)　当使用EJECT控制指令执行页弹出时，输出汇编列表看上去将会像以下列表一样。

<汇编列表>

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

LIST / NOLIST

(2)　LIST/NOLIST(列表/无列表)

[格式描述]

[Δ] $ [Δ] LIST	; 默认假设
[Δ] $ [Δ] LI	; 简略格式
[Δ] $ [Δ] NOLIST	
[Δ] $ [Δ] NOLI	; 简略格式

[功能]

- LIST控制指令为汇编器显示了必须开始汇编列表输出时所在的行。
- NOLIST控制指令为汇编器显示了必须抑制汇编列表输出时所在的行。

  所有在NOLIST控制指令指定后描述的源语句都将会被汇编，但只有直到在源程序中出现LIST控制指令时才会被输出到汇编列表上。

[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控制指令所指定的汇编列表输出的抑制。

  通过使用NOLIST和LIST控制指令的组合，你可以控制汇编列表输出的数量以及列表的内容。

[说明]

- LIST/NOLIST控制指令只能在普通的源程序中进行描述。
- 用于抑制汇编列表输出的NOLIST控制指令功能不能用来停止汇编处理。
- 如果在NOLIST控制指令后指定了LIST控制指令，那么在LIST控制指令后描述的语句将会被再次输出到汇编列表中。LIST或NOLIST控制指令的映像也将会被输出到汇编列表中。
- 如果LIST和NOLIST控制指令都没有被指定，那么在源模块中的所有语句都将会被输出到汇编列表中。

[应用示例]

```
            NAME    SAMP1
$           NOLIST                  ; (1)
DATA1       EQU     10H             ; 语句将不会被输出到汇编列表中
DATA2       EQU     11H             ; 语句将不会被输出到汇编列表中
:                                   ; 语句将不会被输出到汇编列表中
DATAX       EQU     20H             ; 语句将不会被输出到汇编列表中
DATAY       EQU     20H             ; 语句将不会被输出到汇编列表中
$           LIST                    ; (2)
            CSEG
              :
            END
```

<说明>

(1)　由于此处指定了NOLIST控制指令，因此在“$ NOLIST”后直到在(2)中的LIST控制指令间的语句都不会被输出到汇编列表中。而NOLIST控制指令本身的映像将会被输出到汇编列表中。

(2)　由于此处指定了LIST控制指令，因此在该控制指令后的语句将会被输出到汇编列表中。而LIST控制指令本身的映像也将会被输出到汇编列表中。

GEN / NOGEN

(3)　GEN/NOGEN（生成/不生成）

[格式描述]

```
[ Δ ] $ [ Δ ] GEN                                          ; 默认假设
[ Δ ] $ [ Δ ] NOGEN
```

[功能]

- GEN控制指令命令汇编器将宏定义行，宏引用行和宏扩展行输出到一个汇编列表中。
- NOGEN控制指令命令汇编器输出宏定义行和宏引用行，用来抑制宏扩展行。

[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控制指令只能在普通的源程序中进行描述。
- 如果GEN和NOGEN控制指令都没有被指定，那么宏定义行，宏引用行和宏扩展行将会被输出到汇编列表中。
- 在GEN或NOGEN控制指令本身的映像被输出到汇编列表后，将会执行指定列表控制。
- 即使在由NOGEN控制指令进行列表输出控制后，汇编器仍将继续其自身的处理以及语句数量（STNO）增量的计算。
- 如果在NOGEN控制指令后指定了GEN控制指令，汇编器将会继续宏扩展行的输出。

[应用示例]

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

当上述源程序被汇编时，输出汇编列表将看上去就像上面所显示的那样。当该源程序被汇编时，汇编列表将会按如下进行扩展：

<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             ; 宏扩展行不会被输出。
        AUD    A , #20H             ; 宏扩展行不会被输出。
        END
```

<说明>

(1)　由于指定了NOGEN控制指令，因此宏扩展行将不会被输出到汇编列表中。

COND / NOCOND

(4)　COND / NOCOND（条件/无条件）

[格式描述]

```
[ Δ ] $ [ Δ ] COND                                          ; 默认假设
[ Δ ] $ [ Δ ] NOCOND
```

[功能]

- COND控制指令命令汇编器输出对于汇编列表来说满足条件汇编环境的行以及那些没有满足条件汇编环境的行。
- NOCOND控制指令命令汇编器只输出对于汇编列表来说满足条件汇编环境的行。不满足条件汇编环境的行以及在其中描述了IF / _IF，ELSEIF / _ELSEIF，ELSE ,以及ENDIF的行的输出都将被禁止。

[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控制指令只能在普通的源程序中进行描述。
- 如果COND和NOCOND控制指令都没有被指定，汇编器将输出对于汇编列表来说满足条件汇编环境的行以及那些没有满足条件汇编环境的行。
- 在COND或NOCOND控制指令本身的映像被输出到汇编列表后，将会执行指定列表控制。
- 即使在由NOGEN控制指令进行列表输出控制后，汇编器仍将继续ALNO和STON增量的计算。
- 如果在NOCOND控制指令后指定了COND控制指令，汇编器将会继续输出没有满足条件汇编环境的行以及已经在其中描述了IF /_IF , ELSEIF / _ELSEIF , ELSE， 和ENDIF的行。

[应用示例]

```
        NAME    SAMP

$       NOCOND
$       SET ( SW1 )
$       IF ( SW1 )                  ; 即使已经汇编，这部份也不会被输出到列表中。
        MOV     A , #1H
$       ELSE                        ; 即使已经汇编，这部份也不会被输出到列表中。
          MOV   A , #0H             ; 即使已经汇编，这部份也不会被输出到列表中。
$       ENDIF                       ; 即使已经汇编，这部份也不会被输出到列表中。
        END
```

TITLE

(5)　TITLE（标题）

[格式描述]

```
[ ∆ ] $ [ ∆ ] TITLE [ ∆ ] ( [ ∆ ] ' title - string ' [ ∆ ] )
[ ∆ ] $ [ ∆ ] TT [ ∆ ] ( [ ∆ ] ' title - string ' [ ∆ ] )                    ; 简略格式
```

[默认假设]

- 当TITLE控制指令没有被指定时，汇编表头的标题列将会被留空。

[功能]

- TITLE控制指令指定了在汇编列表，符号表格列表或交叉引用列表每个页眉处的标题列中所要输出的字符串。

[用途]

- 使用TITLE控制指令在列表每页中输出一个标题，这样可以使得列表中的内容易于辩认。
- 如果在每次进行汇编时你都需要使用汇编选项来指定一个标题，那么你可以通过在源模块文件中描述该控制指令来启动汇编器，这样可以节省时间和劳力。

[说明]

- TITLE控制指令只能在源模块文件的节头中进行描述。
- 如果同时指定了两个或更多的TITLE控制指令，那么汇编器将只会接收最后指定的TITLE控制指令作为有效指令。
- 最多有60个字符可以被指定为标题串。如果指定的标题串由61个或更多的字符组成，那么汇编器将只会接收串中的前60个字符作为有效字符。

  然而，如果汇编列表文件（数量"X"）中每行的字符长度规范是119个字符或更少，那么"X-60个字符"将可以被接受。
- 如果单引用(')被用作标题串的一部分，则应连续描述两次单引号。
- 如果没有标题串被指定（在标题串中的字符个数为0），那么汇编器将会将标题列留空。
- 如果在指定标题串中发现任何不包含于"2.2.2　字符集"中的字符，那么汇编器将会在标题列中输出"!"来代替非法字符。
- 汇编列表的标题也可以使用在汇编器启动命令行中的汇编选项(-LH)来指定。

[应用示例]

```
$          PROCESSOR ( 014 )
$          TITLE ( ' THIS IS TITLE ' )
           NAME       SAMPLE
           CSEG
           MOV      A , B
           END
```

当上述源程序被汇编时，输出汇编列表将会按以下来显示（带有指定为72页的每页中的行数）。

<汇编列表>

```
78K/0系列汇编器Vx.xx    这是标题    日期:xx xxx xxxx    页：1

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

             汇编列表

ALNO    STNO       ADRS       OBJECT M I  SOURCE STATEMENT

1          1                                  $          PROCESSOR ( 014 )
2          2                                  $          TITLE ( ' THIS IS TITLE ' )
3          3                                             NAME    SAMPLE
4          4 ----                                        CSEG
5          5 0000 63                                     MOV      A , B
6          6                                             END

段信息：

ADRS LEN NAME

0000 0001H ?CSEG

目标片: uPD78014
设备文件: Vx.xx
汇编完成，发现0个错误和0个警告。(0)
```

SUBTITLE

(6)　SUBTITLE（小标题）

[格式描述]

```
[ Δ ] $ [ Δ ] SUBTITLE [ Δ ] ( [ Δ ] ' title - string ' [ Δ ] )
[ Δ ] $ [ Δ ] ST [ Δ ] ( [ Δ ] ' title - string ' [ Δ ] )                    ; 简略格式
```

[默认假设]

- 当SUBTITLE控制指令没有被指定时，汇编列表头的SUBTITLE节将会被留空。

[功能]

- SUBTITLE控制指令指定了在汇编列表的每个页眉处的SUBTITLE节中所要输出的字符串。

[用途]

- 使用SUBTITLE控制指令将小标题输出到汇编列表的每页上，这样可以使得汇编列表易于辩认。小标题的字符串可能根据每个页而改变。

[说明]

- SUBTITLE控制指令只能在普通的源程序中进行描述。
- 最多有72个字符可以被指定为小标题串。
  如果指定的标题串由73个或更多的字符组成，那么汇编器将只会接收串中的前72个字符作为有效字符。一个2个字节的字符被计作两个字符，而制作符则被计作一个字符。
- 以SUBTITLE控制指令指定的字符串将会被输出到页上的SUBTITLE节中，其中该页在已经指定了SUBTITLE控制指令的页之后。然而，如果在页的顶端（第一行）指定了控制指令，那么小标题将会被输出到该页上。
- 如果没有指定SUBTITLE控制指令，那么汇编器将会把SUBTITLE节留空。
- 如果一个单引号(')被用作字符串的一部分，那么将连续描述两次单引号。
- 如果在SUBTITLE节中的字符串为0，那么SUBTITLE列将会被留空。
- 如果在指定的小标题串中发现了不属于"2.2.2　字符集"的任何字符，汇编器将会在SUBTITLE列中输出"!"来代替非法字符。如果CR（0DH）被描述，则将产生一个错误，且不会有任何内容被输出到汇编列表中。如果描述了00H，那么从该点到封闭的单引号(')中的内容将不会被输出。

[[应用示例]

```
        NAME   SAMP
        CSEG
$       SUBTITLE（’这是小标题1’）          ; (1)
$       EJECT                              ; (2)
        CSEG
$       SUBTITLE（’这是小标题2’）          ; (3)
$       EJECT                              ; (4)
$       SUBTITLE  （’这是小标题3’）        ; (5)
      END
```

<说明>

(1)　该控制指令指定了字符串（’这是小标题1’）。

(2)　该控制指令指定了一个页弹出。

(3)　该控制指令指定了字符串（’这是小标题2’）。

(4)　该控制指令指定了一个页弹出。

(5)　该控制指令指定了字符串（’这是小标题3’）。

(6)　该例的汇编列表如下所示（带有指定为80页的每页中的行数）。

<汇编列表>

```
78K/0系列汇编器　Vx.xx 日期 : xx xxx xxxx 页 : 1

Command ：     c014 sample.asm
Para-file :
 In-file :          SAMPLE.ASM
Obj-file :         SAMPLE.REL
Prn-file :         SAMPLE.PRN
            汇编列表

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)
-------------------------------------------------------------------------------------------------------------------------------
78K/0系列汇编器　Vx.xx 日期 : xx xxx xxxx 页 : 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)
-------------------------------------------------------------------------------------------------------------------------------
78K/0系列汇编器　Vx.xx 日期 : xx xxx xxxx 页 : 3

THIS IS SUBTITLE 2

ALNO    STNO       ADRS       OBJECT M I  SOURCE STATEMENT

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

目标片: uPD78014
设备文件 : Vx.xx
汇编完成，发现0个错误和0个警告。(0)
```

FORMFEED / NOFORMFEED

(7)　FORMFEED / NOFORMFEED (换页 / 不换页)

[格式描述]

```
[ Δ ] $ [ Δ ] FORMFEED
[ Δ ] $ [ Δ ] NOFORMFEED                                    ; 默认假定
```

[功能]

- FORMFEED控制指令命令汇编器在汇编列表文件的结尾处输出一个FORMFEED代码。
- NOFORMFEED控制指令命令汇编器不在汇编列表文件的结尾处输出一个FORMFEED代码。

[用途]

- 当你想在输出汇编列表文件中的内容后开始新的一页时，可以使用FORMFEED控制指令。

[说明]

- FORMFEED或NOFORMFEED控制指令只能在源模块文件的节头处进行描述。
- 在输出汇编列表时，如果在页的中间结束输出，那么表的最后一页可能不会输出。假使这样，则通过使用FOMFEED控制指令或汇编选项（－LF）将一个FORMFEED代码添加到汇编列表的结尾处。

  在很多情况下，FORMFEED代码都会被输出到文件的结尾处。由于这个原因，如果在列表文件的结尾处存在一个FORMFEED代码，那么可能会弹出不需要的空白面。为了避免这种情况，NOFORMFEED控制指令或汇编选项(-NLF)会被设为一个默认值。
- 也可以通过汇编器启动命令行中的汇编选项(-LF)或(-NLF)来指定输出或不输出换页代码。
- 如果源模块中的控制指令规范（FORMFEED/NOFORMFEED）不同于启动命令行中的规范(-LF/_NLF)，那么启动命令行中的规范将优先于源模块中的规范。
- 即使当汇编选项(-NP)已经在启动命令行中被指定时，汇编器也会对FORMFEED或NOFORMFEED控制指令执行语法检查。

WIDTH

(8)　WIDTH (宽)

[格式描述]

[∆] $ [∆] WIDTH ∆] ([∆] columns - per - line [∆])

[默认假定]

$WIDTH (132)

[功能]

- WIDTH控制指令指定了列表文件每行中的列（字符）的个数。”每行中的列的个数”必须是一个在72到260范围内的值。

[用途]

- 当你想更改列表文件中每行的列的个数时，可以使用WIDTH控制指令。

[说明]

- WIDTH控制指令只能在源模块文件的节头处进行描述。
- 如果同时指定两个或更多的WIDTH控制指令，那么只有最后指定的那个控制指令将会是有效的。
- 列表文件中每行的列的个数也可以通过汇编器启动命令行中的汇编选项(-LW)来指定。
- 如果在源模块中的控制指令规范（WIDTH）不同于启动命令行中的规范(-LW)，那么启动命令行中的规范将优先于源模块中的规范。
- 即使当汇编选项(-NP)已经在启动命令行中被指定时，汇编器也会对WIDTH控制指令执行语法检查。

LENGTH

(9)　LENGTH（长）

[格式描述]

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

[默认假定]

$LENGTH (66)

[功能]

- LENGTH控制指令指定了列表文件中每页的行数。”每页的行数”可能为”0”，也可能是在20到32767范围内的一个值。

[用途]

- 当你想更改列表文件中每页的行数时，可以使用LENGTH控制指令。

[说明]

- LENGTH控制指令只能在源模块文件的节头处进行描述。
- 如果同时指定两个或更多的LENGTH控制指令，那么只有最后指定的那个控制指令将会是有效的。
- 列表文件中每行的分格数也可以通过汇编器启动命令行中的汇编器选项(-LL)来指定。
- 如果在源模块中的控制指令规范（LENGTH）不同于启动命令行中的规范(-LL)，那么启动命令行中的规范将优先于源模块中的规范。
- 即使当汇编选项(-NP)已经在启动命令行中被指定时，汇编器也会对LENGTH控制指令执行语法检查。

TAB

(10)　TAB (制表)

[格式描述]

[∆] $ [∆] TAB [∆] ([∆] number - of - columns [∆])

[默认假定]

$TAB (8)

[功能]

- TAB控制指令将列的个数指定为列表文件中的制表位的个数。”列的个数”可以是0到8的范围内的一个值。
- TAB控制指令指定了列的个数，其中列变为制表处理的基础，用于通过将表上的多个空白字符代替源模块文件中的HT（水平制表）代码来输出任一的列表。

[用途]

- 当通过使用TAB控制指令减少了任一列表中的每行的字符数时，使用HT代码来减少空白字符的数量。

[Explanation][说明]

- TAB控制指令只能在源模块文件的节头处进行描述。
- 如果同时指定两个或更多的TAB控制指令，那么只有最后指定的那个控制指令将会是有效的。
- 制表位的个数也可以通过汇编器启动命令行中的汇编选项(-LT)来指定。
- 如果在源模块中的控制指令规范（TAB）不同于启动命令行中的规范(-LT)，那么启动命令行中的规范将优先于源模块中的规范。
- 即使当汇编选项(-NP)已经在启动命令行中被指定时，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

通过为条件汇编设置开关，条件汇编控制指令被用于在源模块中选择一系列的语句来作为可以汇编或不能汇编的语句。

这些控制指令由IF / _IF / ELSEIF / _ELSEIF / ELSE / ENDIF控制指令和SET / RESET (设置/ 重设)控制指令组成。

通过有效地使用这些控制指令，你可以在对源模块有微小改动或没有改动的情况下对排除了不需要的语句的源模块进行汇编。

可以使用以下的条件汇编控制指令。

- IF / _IF / ELSEIF / _ELSEIF / ELSE / ENDIF
- SET / RESET

IF /_IF / ELSEIF / _ELSEIF / ELSE / ENDIF

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

[格式描述]

```
    [ Δ ] $ [ Δ ] IF [ Δ ] ( [ Δ ] switch – name [ [ Δ ] : [ Δ ] switch – name ] ... [ Δ ] )
or      [ Δ ] $ [ Δ ] _IFΔconditional – expression
          :
    [ Δ ] $ [ Δ ] ELSEIF [ Δ ] ( [ Δ ] switch – name [ [ Δ ] : [ Δ ] switch – name ] ... [ Δ ] )
or      [ Δ ] $ [ Δ ] _ELSEIFΔconditional – expression
          :
    [ Δ ] $ [ Δ ] ELSE
          :
    [ Δ ] $ [ Δ ] ENDIF
```

[功能]

- 控制指令为用于汇编的限制源语句设置条件。

在IF 或 _IF控制指令及ENDIF控制指令间的描述的源语句可以进行条件汇编。

- 如果条件表达式的评估值或由IF或__IF控制指令（也就是IF 或 _IF 条件）指定的开关名是真的（不同于00H），那么源程序中在该IF或__IF控制指令后描述的源语句将直到下一个条件汇编控制指令(ELSEIF / _ELSEIF , ELSE ，或 ENDIF)出现时才会进行汇编。为了执行之后的汇编，汇编器将会紧着在ENDIF控制指令后来处理语句。

如果IF 或 _IF 条件是假的（00H），那么源程序中在该IF或__IF控制指令后描述的源语句直到下一个条件汇编控制指令(ELSEIF / _ELSEIF , ELSE ，或 ENDIF)出现时也不会进行汇编。

- 只有当ELSEIF或_ELSEIF控制指令前描述的所有条件汇编控制指令的条件都没有被满足时（也就是评价值是假的），才会检查ELSEIF或_ELSEIF是真还是假。

如果由ELSEIF或_ELSEIF控制指令（也就是ELSEIF或_ELSEIF条件）指定的条件表达式的评估值或开关名是真的，那么在ELSEIF或_ELSEIF控制指令控制指令后描述的源语句将直到下一个条件汇编控制指令(ELSEIF / _ELSEIF , ELSE , 或 ENDIF)出现时才会进行汇编。为了执行之后的汇编，汇编器将会紧着在ENDIF控制指令后来处理语句。

如果ELSEIF或_ELSEIF条件是假的（00H），那么源程序中在该ELSEIF或_ELSEIF控制指令后描述的源语句直到下一个条件汇编控制指令(ELSEIF / _ELSEIF , ELSE ，或 ENDIF)出现时也不会进行汇编。

- 如果在ELSE前描述的所有IF / _IF和ELSEIF / _ELSEIF控制指令的条件都没有被满足（也就是所有的开关名都是假的），那么在该ELSE控制指令后描述的源语句将直到源程序中的ENDIF控制指令出现时才会进行汇编。

- ENDIF控制指令向汇编器显示了用于条件汇编的源语句的结束。

[用途]

- 使用这些条件汇编控制指令时，用于汇编的源语句可以被更改，且不会对源程序有很大的修改。
- 如果只在程序开发过程中必需的用于调试的语句在源程序中被描述，调试语句是否应该进行汇编（转换为机器语言）可以通过为条件汇编设置开关来指定。

[说明]

- IF和ELSEIF控制指令用于通过开关名来进行真/假条件判断，而_IF和_ELSEIF控制指令则通过使用条件表达式来进行真/假条件判断。

  IF / ELSEIF和_IF / _ELSEIF可以用在组合中。换名话说，ELSEIF / _ELSEIF可以和IF 或 _IF以及 ENDIF配对使用。
- 为一个条件表达式描述绝对表达式。
- 描述开关名的规则与符号描述的约定相同（详情请参见"2.2.3　符号字段"）。然而，可以被当作开关名的最大的字符数始终为31。
- 如果使用IF或ELSEIF控制指令指定两个或更多的开关名，则应用冒号(:)将每个开关名隔开。每个模块最多可以使用5个开关名。
- 当使用IF或ELSEIF控制指令指定了两个或更多的开关名时，如果其中的一个开关名的值是真的，则将会判定满足了IF或ELSEIF条件。
- 使用IF或ELSEIF控制指令指定的每个开关名的值必须通过SET / RESET (设置 / 重设)控制指令来定义。因此，如果IF或ELSEIF控制指令指定的开关名的值没有预先通过SET或RESET控制指令在源模块中进行设置，那么将假设进行重设。
- 如果指定的开关名或条件表达式包含一个非法描述，汇编器将输出一个错误消息并确定评估值是假的。
- 当描述IF或__IF控制指令时，IF或__IF控制指令必须始终与ENDIF控制指令一同使用。
- 如果IF－ENDIF模块在宏体中被描述且控制通过EXITM操作从该层的宏中被返回，那么汇编器将强行把IF层返回到宏体入口处的层中。这样就不会产生错误。
- 在另一个IF－ENDIF模块中的IF－ENDIF的模块的描述被认为是IF控制指令的嵌套。IF控制指令最多允许有8个嵌套。

- 在条件汇编中，目标代码不会为没有汇编的语句而生成，但这些语句将会被输出到汇编列表中，其中语句没有任何改动。如果你不希望输出这些语句，可以使用$NOCOND控制指令。

[应用示例]

< Example 1 ><例1>

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

<说明>

(1)　如果开关名”SW1”的值是真的，那么在” text1”中的语句将会进行汇编。
如果开关名”SW1”的值是假的，那么在” text1”中的语句将不会进行汇编。
在” text0”中描述的开关名”SW1”的值已经通过SET或RESET控制指令被设置为真或假。

(2)　该指令显示了条件汇编的源语句范围的结尾。

<例2>

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

<说明>

(1)　在” text0”中描述的开关名”SW1”的值已经通过SET或RESET控制指令被设置为真或假。
如果开关名”SW1”的值是真的，在”文本1”中的语句将会进行汇编，而在” text2”中的语句将不会进行汇编。

(2)　如果开关名”SW1”的值是假的，那么在” text1”中的语句将不会进行汇编，而在” text2”中的语句将会进行汇编。

(3)　该指令显示了条件汇编的源语句范围的结尾。

<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)　在” text0”中描述的开关名”SW1”，“SW2”，和 “SW3”的值已经通过SET或RESET控制指令被设置为真或假。

如果开关名”SW1”或 “SW2”的值是真的，在”文本1”中的语句将会进行汇编，而在” text2”， ” text3” 和” text4”中的语句将不会进行汇编。

I如果开关名”SW1”或 “SW2”的值是假的，在” text1”中的语句将不会进行汇编，而在（2）之后的语句将会进行条件汇编。

(2)　如果在（1）中的开关名”SW1”，“SW2”值是假的而开关名”SW3”的值是真的，那么在” text2”中的语句将会进行汇编，而在” text1”， ” text3” 和” text4”的语句则不会进行汇编。

(3)　如果在（1）中的开关名”SW1”，“SW2”的值以及在（2）中的”SW3”的值都是假的而”SW4”值是真的，那么在” text3”中的语句将会进行汇编，而” text1”， ” text2” 和” text4”中的语句则不会进行汇编。

(4)　如果在（1）中的开关名”SW1”，“SW2”的值，在（2）中的”SW3”的值以及在（3）中的”SW4”的值均是假的，那么在” text4”中的语句将会进行汇编，而在” text1”， ” text2”和 ” text3”中的语句则不会进行汇编。

(5)　该指令显示了条件汇编的源语句范围的结尾。

<例4>

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

<说明>

(1)　在” text0”中描述的开关名”SYMA”的值已经通过EQU或SET伪指令进行定义。如果符号名”SYMA”是真的（非”0”），那么在” text1”中的语句将会进行汇编，而” text2”中的语句则不会进行汇编。

(2)　如果符号名”SYMA”的值为”0”且”SYMB”和”SYMC”具有相同的值，那么” text2”中的语句将会进行汇编。

(3)　该指令显示了条件汇编的源语句范围的结尾。

SET / RESET

(2)　SET / RESET（设置/重设）

[格式描述]

[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和RESET控制指令赋于每个通过IF或ELSEIF控制指令指定的开关名一个值。
- SET控制指令赋于在其操作数中指定的每个开关名一个真值（0FFH）。
- RESET控制指令赋于在其操作数中指定的每个开关名一个假值（00H）。

[用途]

- 描述SET控制指令，用于赋于通过IF或ELSEIF控制指令指定的每个开关名一个真值（0FFH）。
- 描述RESET控制指令，用于赋于通过IF或ELSEIF控制指令指定的每个开关名一个假值（00H）。

[说明]

- 使用SET和RESET控制指令时，至少要描述一个开关名。

  描述开关名的约定与描述符号的约定是一致的（参见"2.2.3　符号字段"）。然而，可以被当作开关名的最大的字符数始终为31。
- 指定的开关名可能和不同于保留字的用户自定义符号以及其他开关名相同。
- 如果使用SET或RESET控制指令指定两个或更多的开关名，则应用冒号(:)将每个开关名隔开。每个模块最多可以使用1000个开关名。
- 通过SET控制指令设置为 “ 真 ” 的开关名可以通过RESET控制指令设置为 “ 假 ” ，反之亦然。
- 在描述IF或ELSEIF控制指令前，以IF或ELSEIF控制指令指定的开关名必须通过SET或RESET控制指令在源模块中至少定义一次。
- 开关名不会被输出到交叉参考列表中。

[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)　该指令赋于开关名”SW1”一个真值（0FFH）。

(2)　由于在上述（1）中，已经赋于开关名”SW1”一个真值，因此” text1”中的语句将会进行汇编。

(3)　该指令显示了从（2）开始的条件汇编的源语句范围的结尾。

(4)　该指令分别赋于开关名“SW1”和 “SW2”一个假值（00H）。

(5)　由于在上述（4）中，已经赋于开关名”SW1”一个假值，因此” text2”中的语句将不会进行汇编。

(6)　由于在上述（4）中，已经赋于开关名”SW2”一个假值，因此” text3”中的语句将不会进行汇编。

(7)　由于在上述（5）和（6）中的开关名“SW1” 和“SW2”的值均是假的，因此” text4”中的语句将会进行汇编。

(8)　该指令显示了从（5）开始的条件汇编的源语句范围的结尾。

4.8　其它控制指令

以下的控制指令是由高级程序例如C编译器和结构汇编预处理器输出的特殊控制指令。

$TOL_INF

$DGS

$DGL

第 5 章　宏

本章节说明了如何使用宏功能。当你需要在源程序中重复描述一系列的语句时，宏是一个非常有用的功能。

5.1　宏的概述

当你必须在源程序中重复描述一系列的指令或一组指令时，宏功能对于程序描述是非常有用的。宏功能适用于将通过MACRO和ENDM伪指令定义为宏体的一系列语句（一个指令组）扩展到宏名被引用的位置。

宏用于增加源程序的编码效率，并且它不同于子程序。

宏和子程序具有如下说明的明显特征。为了有效的使用，请依照特定的目的来选择一个宏或一个子程序。

(1)　子程序

- 将一个必须在程序中重复多次的过程描述为一个单一的子程序。这个子程序由汇编器一次性转换为机器语言。
- 要调用子程序时，你只需要描述一个子程序的调用指令（通常情况下，用于设置参数的指令也会在子程序之前或之后进行描述）。

  子程序的有效使用可以使程序内存被高效的使用。
- 通过将程序中的一系列处理过程编码为子程序，程序可以被结构化（对于程序员来说，结构化可以使程序的整体结构变得易于理解，并且使程序设计变得简单）。

(2)　宏

- 宏的基本功能是以一个名称来代替一组指令。

  通过MACRO和ENDM伪指令定义为宏体的一系列（或组）的指令将会被扩展到宏名被引用的位置。
- 当汇编器发现一个宏引用时，汇编器将会扩展宏体并将指令组转换为机器语言，同时也会将宏引用时的实际参数来代替宏体的形式参数。
- 可以为宏描述参数。

  例如，如果存在在处理过程中是相同的，但在操作数中要描述的数据中却不相同的指令组时，将会通过把形式参数赋于数据来定义一个宏。通过在宏引用时描述宏名和实际参数，汇编器可以妥善处理不同的指令组，其中指令组只在部分的语句描述中有所不同。

使用子程序的编程技术主要用于减少内存尺寸和结构程序，而宏则是用于增加程序的编码效率。

5.2 宏的应用

5.2.1 宏定义

通过MACRO和ENDM伪指令来定义一个。

[格式描述]

符号字段	助记符字段	操作数字段	注释字段
macro – name宏名	MACRO	[[formal - parameter [, ...]]	[；注释]
	:		
	ENDM		[；注释]

[功能]

- MACRO伪指令通过将符号字段中指定的宏名赋于在MACRO伪指令和ENDM伪指令间描述的一系列语句（称为宏体）来执行宏定义。

[应用示例]

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

<说明>

上例显示了一个简单的宏定义，它指定了“PARA1”和“PARA2”两个值和寄存器A中的保存结果的和。宏被赋于一个名称“ADMAC”，而 “PARA1”和“PARA2”则是形式参数。

详情请参见”3.8 宏伪指令”。

5.2.2 宏引用

为了调用一个宏，已经定义的宏名必须在源程序的助记符字段中进行描述。

[格式描述]

符号字段	段助记符字段	操作数字段	注释字段
[标签：]	宏名	[[实际参数 [, ...]]	[; 注释]

[功能]

- 该语句描述调用了宏体，该宏体被赋于助记符字段中所指定的宏名。

[用途]

- 使用该语句描述来调用一个宏体。

[说明]

- 要在助记符字段中指定的宏名在宏引用前必须已经被定义。
- 每行最多可以指定16个实际参数，并且使用逗号(,)将每个实际参数隔开。
- 在组成一个实际参数的字符串中不能描述空白字符。
- 当在实际参数中描述一个逗号(,)，分号(;)，空格或制表符时，应使用一对单引号来封闭包含这些特殊字符中任一字符的字符串。
- 形式参数按从左到右的顺序被与之相应的实际参数所代替。如果形式参数的数量与实际参数的数量不相等，则将输出一条警告消息。

[应用示例]

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

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

<说明>

该宏引用调用了已经定义的宏名“ADMAC”。 10H和20H为实际参数。

5.2.3 宏扩展

汇编器按以下方式来处理宏：

- 汇编器依照所引用的宏名将宏体扩展到宏名被引用的位置。
- 汇编器按与其他语句相同的方式对扩展的宏体中的语句进行汇编。

5.2.4 应用示例

当对"5.2.2 宏引用"中所引用的宏进行汇编时，宏体将会按以下所显示的进行扩展。

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

            ; Source text
        CSEG
            :
            ; Macro expansion
        ADMAC 10H , 20H             ; (1)
        MOV     A , #10H
        ADD     A , #20H
            ; Source text
            :
        END
```

<说明>

(1) 通过在（1）中的宏引用，宏体将会被扩展。宏体中的形式参数将会被实际参数所代替。

5.3　宏内的符号

可以在宏中定义的符号被分为两种类型：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中的任何语句中进行引用的符号。

  因此，如果已经定义了全局符号的宏被不止一次地引用以扩展一系列的语句时，那么符号将会产生双重定义错误。

- 不使用LOCAL伪指令定义的符号是全局符号。

(2)　局部符号

- 局部符号是以LOCAL伪指令定义的符号（参见”3.8　宏伪指令”）。
- 可以通过LOCAL伪指令来引用定义为LOCAL的宏内的局部符号。
- 不能从宏外引用任何局部符号。

[应用示例]

<源程序>

```
        NAME    SAMPLE
          ; Macro definition
MAC1    MACRO
                LOCAL   LLAB        ; (1)
LLAB :
          :
GLAB :
                BR          LLAB    ; (2)
                BR          GLAB    ; (3)

          ENDM
          :
          ; Source text
REF1 :  MAC1                        ; (4) <-- Macro reference
          :
        BR          LLAB            ; (5) <-- This description is erroneous.
        BR          GLAB            ; (6)
          :
REF2 :  MAC1                        ; (7) <-- Macro reference
          :
        END
```

<说明>

(1) 该LOCAL伪指令将标签“LLAB”定义为一个局部符号。

(2) 该BR指令在宏“MAC1”中引用了局部符号“LLAB”。

(3) 该BR指令在宏“MAC1”中引用了全局符号“GLAB”。

(4) 该语句引用了宏“MAC1”。

(5) 该BR指令从宏“MAC1”的定义外引用了局部符号“LLAB”。当对源程序进行汇编时，该描述将会产生一个错误。

(6) 该BR指令从宏“MAC1”的定义外引用了全局符号“GLAB”。

(7) 该语句引用了宏“MAC1”。且同一宏被引用两次。

当上例中的源程序进行汇编时，宏体将会按以下显示的进行扩展。

<汇编列表>

```
          NAME      SAMPLE
                :
REF1 :    MAC1
                ; Macro expansion
??RA0000 :
                :
GLAB :                                  <-- Error
          BR            ??RA0000
          BR            GLAB
                ; Source text
                :
          BR            !LLAB           <-- Error
          BR            !GLAB
                :
REF2 :    MAC1
                ; Macro expansion
??RA0001 :
                :
GLAB :                                  <-- Error
          BR            ??RA0001
          BR            GLAB
                ; Source text
                :
          END
```

<说明>

全局符号“GLAB”已经在宏“MAC1”中被定义。由于宏“MAC1”被引用了两次，因此作为宏体中一系列语句的扩展结果，全局符号“GLAB”将会产生一个双重定义错误。

5.4 宏运算符

两种类型的宏运算符是可用的：“& (和)” and “' (单引号)”。

(1) &（连接）

- 在宏体中，和号“&”将一个字符串与另一个字符串连接在一起。宏扩展时，和号左边的字符串被连接到和号右边的字符串。在连接到串后，“&”本身将会消失。
- 在宏定义时，符号中在“&”之前或之后的串被当作形式参数或局部符号。在宏扩展时，在“&”之前或之后形式参数或局部符号被评定为一个符号，并且在符号中可以被连接。
- 封闭在单引号中的“&”符号被简单地当作数据来处理。
- 两个连续描述的“&”符号被当作一个“&”符号来处理。

[应用示例]

<宏定义>

```
MAC       MACRO P
LAB&P :                 <-- Formal parameter 'P' is recognized.
              D&B   10H
              DB    ' P '
              DB    P
              DB    ' &P '
      ENDM
```

<宏引用>

```
      MAC   1H
LAB1H :
      DB    10H     <-- 'D' and 'B' are concatenated and become 'DB'.
      DB    ' P '
      DB    1H
      DB    ' &P '  <-- & enclosed in a pair of single quotation marks is simply handled as data.
```

(2) ' (单引号)

- 如果以一对单引号封闭的字符串在宏引用行或IRP伪指令中的实际参数的开始处，或在一个分隔字符后进行描述，那么字符串将会被解释成实际参数。没有封闭在单引号中的字符串将会被传递到实际参数中。
- 如果在宏体中存在一个以单引号封闭的字符串，那么字符串将被简单地当作数据来处理。
- 为了将单引号用作文本中的单引号，则应连续描述两次单引号。

[应用示例]

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

        MAC1   ' 10 , 20 , 30 '
```

当上例中的源程序被汇编时，宏“MAC1”将会按以下所示的进行扩展。

<汇编列表>

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

第 6 章　产品应用

本章节介绍了几种可以有效使用RA78K0汇编器包的方法。

6.1　节省启动汇编器的时间和麻烦

一些控制指令具有同汇编选项相同的功能，且当启动汇编器时必须始终使用这些指令；这些示例中包含了处理器类型规范(-C)。建议在源模块文件中描述这类控制指令。特别是不能被省略的处理器类型规范应该使用PROCESSOR控制指令在源模块文件的节头中进行指定。这可以避免你每次启动汇编器时都要在启动命令行中指定汇编选项(-C)的必要。需记住，如果你忘记在启动命令行中指定该汇编选项，则将会产生一个错误，并且你将不得不以正确的汇编选项从开始处再次启动汇编器。

输出控制指令（XREF）的交叉引用列表也应该在模块头中进行指定。

<例>

```
$       PROCESSOR ( 014 )
$       KANJICODE SJIS
$       XRFF

        NAME    TEST

        CSEG
          :
        END
```

6.2　如何以高的内存利用率来开发程序

短直接寻址区域与其它数据内存区域相比是可以通过短字节长的指令进行访问的区域。

因此，通过有效地使用该区域，可以开发具有高内存利用率的程序。

在一个模块中说明短直接寻址区域。这样，即使所有你想置于短直接寻址区域中的所有变量都不能被放置在其中，你也可以很容易的进行改动使只用于访问的变量被置于短直接寻址区域中。

[应用示例]

<模块1>

```
PUBLIC TMP1 , TMP2
WORK    DSEG  AT 0FE20H
TMP1 :  DS    2                 ; word
TMP2 :  DS    1                 ; byte
```

<模块2>

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

附录 A　保留字列表

保留字在六种类型中是可用的：机器语言指令，伪指令，控制指令，运算符，寄存名和sfr符号。保留字是由汇编器事先保留的字符串，不能用于不是指定目的的其他目的。

可以在源程序的各个字段中描述的保留字的类型如下显示：

表A-1　保留字的类型

类型	说明
符号字段	在该字段中不能描述保留字。
助记符字段	只有机器语言指令和伪指令可以在该字段中进行描述。
操作数字段	只有运算符，sfr符号和寄存器名可以在该字段中进行描述。
注释字段	所有保留字都可以在该字段中进行描述。

关于sfr列表的详细信息请参考各个设备的用户手册。

关于中断请求源列表的详细信息请参考各个设备的用户手册。

关于机器语言指令和寄存器名列表的详细信息请参考各个设备的用户手册。

表 A-2　保留字列表

类型	保留字				
运算符	AND	BANKNUM	BITPOS	DATAPOS	EQ
	GE	GT	HIGH	LE	LOW
	LT	MASK	MOD	NE	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			lNCLUDE [IC]	
	LENGTH			LIST/ NOLIST [LI / NOLI]	
	PROCESSOR [PC]			SET / RESET	
	SUBTITLE [ST]			SYMLIST / NOSYMLIST	
	TAB			TITLE [TT]	
	WIDTH			XREF / NOXREF [XR / NOXR]	
其他	DGL	DGS	SFR	SFRP	TOL_INF

备注　接在控制指令之后的方括号中的项表示简略格式。

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	助记符字段	操作数字段	注释字段		
[段名]	CSEG	[重置属性]	[;注释]	表明一个代码段开始	
[段名]	DSEG	[重置属性]	[;注释]	表明一个数据段开始	
[段名]	BSEG	[重置属性]	[;注释]	表明一个位段开始	
[段名]	ORG	绝对表达式	[;注释]	表明一个绝对段开始	禁止操作数内符号的正向引用。
名称	EQU	表达式	[;注释]	定义一个名称	名称：符号 禁止操作数内符号的正向引用或外部引用。
名称	SET	绝对表达式	[;注释]	定义一个重新定义的名称	名称：符号 禁止操作数内符号的正向引用。
[标签：]	DB	（尺寸）初始值[, …]	[;注释]	初始化或保留一个字数据区域。	标签：符号 可以置于带有初始值的空间的字符串。
[标签：]	DW	（尺寸）初始值[, …]	[;注释]	初始化或保留一个字数据区域。	标签：符号
[标签：]	DS	绝对表达式	[;注释]	保留字节数据区域	名称：符号 禁止操作数内符号的正向引用。
名称	DBIT	无	[;注释]	保留一个位数据区域	名称：符号 禁止操作数内符号的正向引用。

表B-1　伪指令列表

伪指令				功能分类	备注
符号字段	助记符字段	操作数字段	注释字段		
[标签：]	PUBLIC	符号名[, …]	[;注释]	表明一个外部定义名。	
[标签：]	EXTRN	符号名[, …]	[;注释]	表明一个外部定义名。	
[标签：]	EXTBIT	位符号名[, …]	[;注释]	表明一个外部定义名。	仅限那些具有位值的符号名
[标签：]	NAME	目标模块名	[;注释]	定义一个模块名。	模块名：符号
[标签：]	BR	表达式	[;注释]	自动选择一个分支指令。	标签：符号
宏名	MACRO	[形式参数[, …]]	[;注释]	定义一个宏。	宏名：符号
[标签：]	LOCAL	符号名[, …]	[;注释]	定义一个仅在宏内有效的符号。	只能用在宏定义中。
[标签：]	REPT	绝对表达式	[;注释]	在宏扩展期间指定重复次数。	标签：符号
[标签：]	IRP	形式参数<实际参数[, …]>	[;注释]	将实际参数赋于形式参数	标签：符号
[标签：]	EXITM	无	[;注释]	中断宏扩展	只能用在宏定义中。
无	ENDM	无	[;注释]	终止宏定义	只能用在宏定义中。
无	END	无	[;注释]	表明源模块结束。	

附录 C 索引