To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

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

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

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

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

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

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

用户手册

RA78K0S

汇编包 1.30 版或升级版

语言

目标设备

78K/0S 系列

文档号　U14877CA1V0UM00（第 1 版）
发布日期　2007 年 8 月 N CP(K)

日本印刷

[备忘录]

Microsoft, Windows 和 Visual C++ 是位于美国或其他国家的微软公司注册商标或商标。
其他商标名或产品名是各经营者的注册商标或商标。

- **本文档所刊登的内容有效期截至 2007 年 8 月。将来可能未经预先通知而更改。在实际进行生产设计时，请参阅各产品最新的数据表或数据手册等相关资料以获取本公司产品的最新规格。**
- 并非所有的产品和/或型号都向每个国家供应。请向本公司销售代表查询产品供应及其他信息。
- 未经本公司事先书面许可，禁止复制或转载本文件中的内容。否则因本文档所登载内容引发的错误，本公司概不负责。
- 本公司对于因使用本文件中列明的本公司产品而引起的，对第三者的专利、版权以及其它知识产权的侵权行为概不负责。本文件登载的内容不应视为本公司对本公司或其他人所有的专利、版权以及其它知识产权作出任何明示或默示的许可及授权。
- 本文件中的电路、软件以及相关信息仅用以说明半导体产品的运作和应用实例。用户如在设备设计中应用本文件中的电路、软件以及相关信息，应自行负责。对于用户或其他人因使用了上述电路、软件以及相关信息而引起的任何损失，本公司概不负责。
- 虽然本公司致力于提高半导体产品的质量及可靠性，但用户应同意并知晓，我们仍然无法完全消除出现产品缺陷的可能。为了最大限度地减少因本公司半导体产品故障而引起的对人身、财产造成损害（包括死亡）的危险，用户务必在其设计中采用必要的安全措施，如冗余度、防火和防故障等安全设计。
- 本公司产品质量分为：

  “标准等级”、“专业等级”以及“特殊等级”三种质量等级。

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

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

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

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

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

（注）

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

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

区域信息

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

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

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

详细信息请联系：

（中国区）

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

[北京]

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

[深圳]

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

[上海]

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

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

[香港]

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

引 言

设计本手册的目的是为方便程序员正确理解 **RA78K0S 汇编包**（以下称 **RA78K0S**）中每个程序的基本功能，以及描述源程序的方法。

本手册不包括如何操作 RA78K0S 的各个程序。因此，理解了该手册的内容后，请阅读 **RA78K0S 汇编包用户操作手册（U14876E）**（以下称**操作**），以运行汇编包内的每个程序。

本手册中与 RA78K0S 有关的描述适用于 1.30 版本及以后的升级版。

[目标读者]

该手册面向理解用于开发微控制器（78K0S 系列）的功能和指令的用户工程师。

[组织]

该手册包括下面 6 个章节和若干附录：

第一章　概述

概述 RA78K0S 的所有基本功能

第二章　如何描述源程序

概述如何描述源程序，并解释汇编器的操作。

第三章　伪指令

解释如何编写和使用伪指令，并辅以应用实例进行说明

第四章　控制指令

解释如何编写和使用控制指令，并辅以应用实例进行说明

第五章　宏

解释所有宏的功能，包括宏定义、宏引用和宏展开。

宏伪指令在**第三章 伪指令**中解释。

第六章　产品应用

推荐一些的描述源程序的方法。

附录

包括保留字列表、伪指令列表、最佳性能表、特征和索引。

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

对于指令的结构，也请参考正对其进行软件开发的微控制器（指令版）用户手册。

[一般建议]

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

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

[编写惯例]

下列符号和缩写适用于整个手册：

:	重复同一格式
[ ]:	方括号内的字符可省略。
{ }:	选择在{}内的某一项
“:”:	括在双引号””内的字符是字符串。
‘ ’:	括在单引号”内的字符是字符串。
( ):	圆括号之间的字符是字符串
< >:	尖括号之间的字符（主要用于标题）是字符串
__:	下划线用于表示一个要点或输入字符串。
Δ:	表示一个或多个空白字符或 TAB 键。
/:	字符分隔符
~:	连续性
黑体字:	字体字符用于表示一个要点或引用点。

[相关文档]

与该手册有关的文档（用户手册）列于下表。

在该手册中伪指令的相关文档包括基础版本。

但是，基础版本不按如下方式标记。

文档名		文档号
RA78K0S 汇编包	操作	U14876E
	语言	本手册
	结构化汇编语言	U11623E
CC78K0S C 编译器	操作	U14871E
	语言	U14872E
SM78K0S, SM78K0 系列基于Windows™的系统模拟器	操作	U14611E
基于 Windows 的项目管理器	操作	U14610E
ID78K0-NS, ID78K0S-NS 基于 Windows 的集成调试器	操作	U14910E
78K/0S 系列操作系统 MX78K0S	基础	U12938E
78K/0S 系列	指令	U11047E

[备忘录]

目 录

第一章 概述.....15
1.1 汇编器简介.....16
1.1.1 什么是汇编器?.....17
1.1.2 什么是重定位汇编器?.....19
1.2 程序开发之前的提醒.....21
1.2.1 RA78K0S 的最佳性能特征.....21
1.3 RA78K0S 的特点.....23

第二章 如何描述源程序.....25
2.1 源程序的基本配置.....25
2.1.1 模块头.....26
2.1.2 模块体.....27
2.1.3 模块尾.....28
2.1.4 源程序的全面配置.....28
2.1.5 源程序的编写示例.....29
2.2 源程序的编写格式.....32
2.2.1 配置语句.....32
2.2.2 字符集.....33
2.2.3 组成语句的字段.....36
2.3 表达式和操作符.....48
2.3.1 操作符的功能.....49
2.3.2 操作符的约束.....64
2.4 比特位定义.....71
2.5 操作数字特性.....74
2.5.1 操作数值的大小和地址范围.....74
2.5.2 指令所需的操作数大小.....76
2.5.3 操作数的符号属性和重定位属性.....77

第三章 伪指令.....81
3.1 伪指令概述.....81
3.2 段定义伪指令.....82
(1) CSEG (代码段).....84
(2) DSEG (数据段).....88
(3) BSEG (位段).....92
(4) ORG (起源).....97
3.3 符号定义伪指令.....100
(1) EQU (等于).....101
(2) SET (设置).....105
3.4 内存初始化和区域保留伪指令.....107
(1) DB (定义字节).....108
(2) DW (定义字).....110
(3) DS (定义存储空间).....112
(4) DBIT (定义位).....114
3.5 链接伪指令.....115
(1) EXTRN (外部的).....116

(2) EXTBIT (外部比特) 118
(3) PUBLIC (公共的) 120
3.6 目标模块名称声明伪指令 122
(1) NAME (名称) 123
3.7 自动分支指令选择伪指令 124
(1) BR (分支) 125
3.8 宏伪指令 127
(1) MACRO (宏) 128
(2) LOCAL (局部的) 130
(3) REPT (重复) 133
(4) IRP (不确定性重复) 135
(5) EXITM (退出宏) 137
(6) ENDM (结束宏) 140
3.9 汇编终止伪指令 142
(1) END (结束) 143

第四章 控制指令 145
4.1 控制指令概述 145
4.2 处理器类型定义控制指令 147
(1) PROCESSOR (处理器) 148
4.3 调试信息输出控制指令 149
(1) DEBUG/NODEBUG (debug/nodebug) 150
(2) DEBUGA/NODEBUGA (debuga/nodebuga) 151
4.4 交叉引用表输出定义控制指令 152
(1) XREF/NOXREF (xref/noxref) 153
(2) SYMLIST/NOSYMLIST (symlist/nosymlist) 154
4.5 包含控制指令 155
(1) INCLUDE (include) 156
4.6 汇编列表控制指令 159
(1) EJECT (eject) 160
(2) LIST/NOLIST (list/nolist) 162
(3) GEN/NOGEN (generate/no generate) 164
(4) COND/NOCOND (condition/no condition) 166
(5) TITLE (title) 168
(6) SUBTITLE (subtitle) 171
(7) FORMFEED/NOFORMFEED (formfeed/noformfeed) 174
(8) WIDTH (width) 175
(9) LENGTH (length) 176
(10) TAB (tab) 177
4.7 条件汇编控制指令 178
(1) IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF 179
(2) SET/RESET (set/reset) 184
4.8 其它控制指令 186

第五章 宏 187
5.1 宏概述 187
5.2 宏应用 188
5.2.1 宏定义 188

5.2.2 宏引用 189
5.2.3 宏扩展 190
5.3 宏内的符号 191
5.4 宏操作符 194

第六章 产品应用 197
附录 A 保留字列表 199
附录 B 伪指令列表 201
附录 C 最佳性能特征 203
附录 D 索引 205

图示列表

图号	标题	页码
1-1	RA78K0S 汇编包	16
1-2	汇编流程	17
1-3	微处理器应用产品的开发过程	18
1-4	调试重汇编	20
1-5	使用已有模块的程序开发	20
2-1	源模块的配置	25
2-2	源模块的整体配置	28
2-3	源模块配置示例	28
2-4	源程序配置	29
2-5	组成语句的字段	32
3-1	段的内存位置	83
3-2	代码段的重定位	84
3-3	数据段的重定位	88
3-4	位段的重定位	92
3-5	绝对段的重定位	97
3-6	两模块间符号的关系	115

表格列表

表号.	标题	页码
1-1	汇编器的最佳性能特征	21
1-2	连接器的最佳性能特征	22
2-1	可在模块头描述的指令	26
2-2	符号类型	36
2-3	由汇编器自动产生的段名	38
2-4	符号的属性和值	39
2-5	表示数字常量的方法	42
2-6	可在操作数字段描述的特殊字符	44
2-7	操作符类型	48
2-8	操作符优先级	49
2-9	重定位属性的类型	64
2-10	用重定位属性组合项和操作符	65
2-11	用重定位属性组合项和操作符（外部引用项）	67
2-12	符号属性的操作类型	68
2-13	用符号属性组合项和操作符	69
2-14	用重定位属性组合第一项和第二项	73
2-15	位符号值	73
2-16	指令操作数指的范围	75
2-17	伪指令操作数指的范围	75
2-18	作为操作数描述的符号属性	78
2-19	作为伪指令的操作数描述的符号属性	79
3-1	伪指令列表	81
3-2	段定义方法和内存地址定位	82
3-3	CSEG 的重定位属性	85
3-4	CSEG 的缺省段名	86
3-5	DSEG 的重定位属性	89
3-6	DSEG 的缺省段名	90
3-7	BSEG 的重定位属性	93
3-8	BSEG 的缺省段名	95
3-9	伪指令位值的操作数表示格式	102
4-1	控制指令列表	145
4-2	控制指令和汇编选项	146

[备忘录]

第一章 概述

本章介绍 RA78K0S 在微控制器软件开发中的作用以及 RA78K0S 的特点。

1.1 汇编器简介

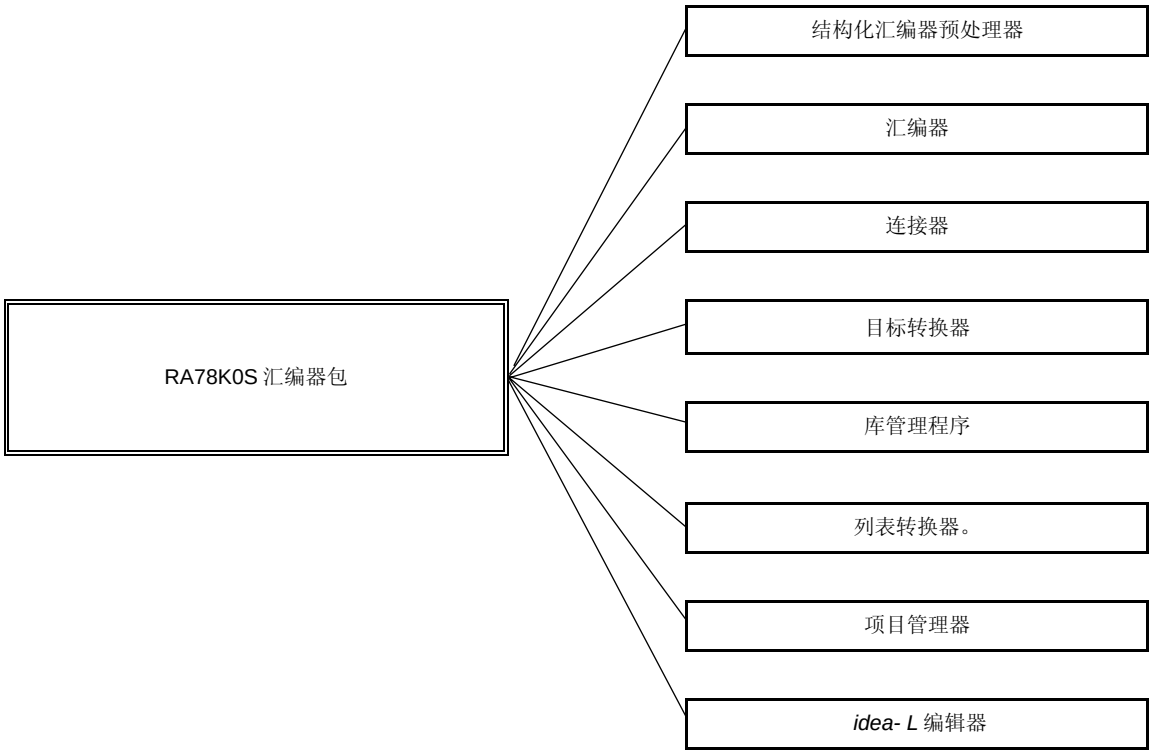
RA78K0S 汇编器包是一个综合术语，指把采用汇编语言为 78K/0S 系列微控制器编写的源程序翻译成机器语言代码的一系列程序。

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

另外， RA78K0S还提供一个项目管理器，对在Windows™上执行程序编辑、编译/汇编、连接、调试等一系列操作起到辅助作用。

该项目管理器提供一个编辑器（*idea- L* 编辑器）。

图 1-1. RA78K0S 汇编器包



1.1.1 什么是汇编器？

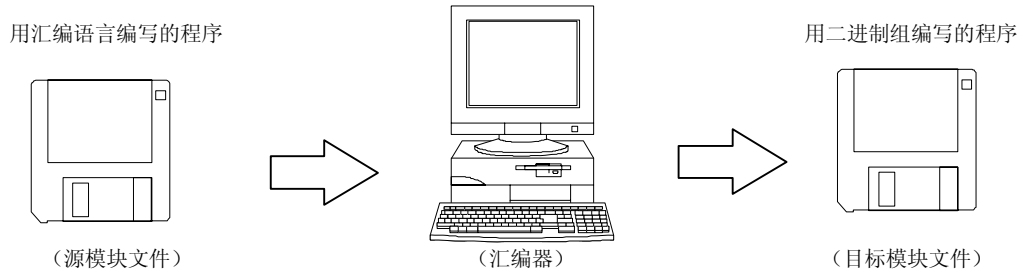
（1）汇编语言和机器语言

汇编语言是针对微控制器的最基础的编程语言。

对于微控制器中的微处理器来说，为了完成它的工作，需要一些程序和数据。这些程序和数据必须通过人（如程序员）编写并存入控制器的内存。由微控制器处理的程序和数据是二进制数的集合，称为机器语言。但是对程序员来说，机器语言代码难以记忆，经常引起错误。幸运的是，存在这样一些方法，即采用英语缩写词或记忆术来表示原始机器语言代码的含义，从而使人们易于理解。使用这些符号编码的程序语言系统称为汇编语言。

由于微控制器必须处理机器语言形式的程序，因此需要另一程序将汇编语言编写的程序翻译成机器语言。这个程序就是汇编器。

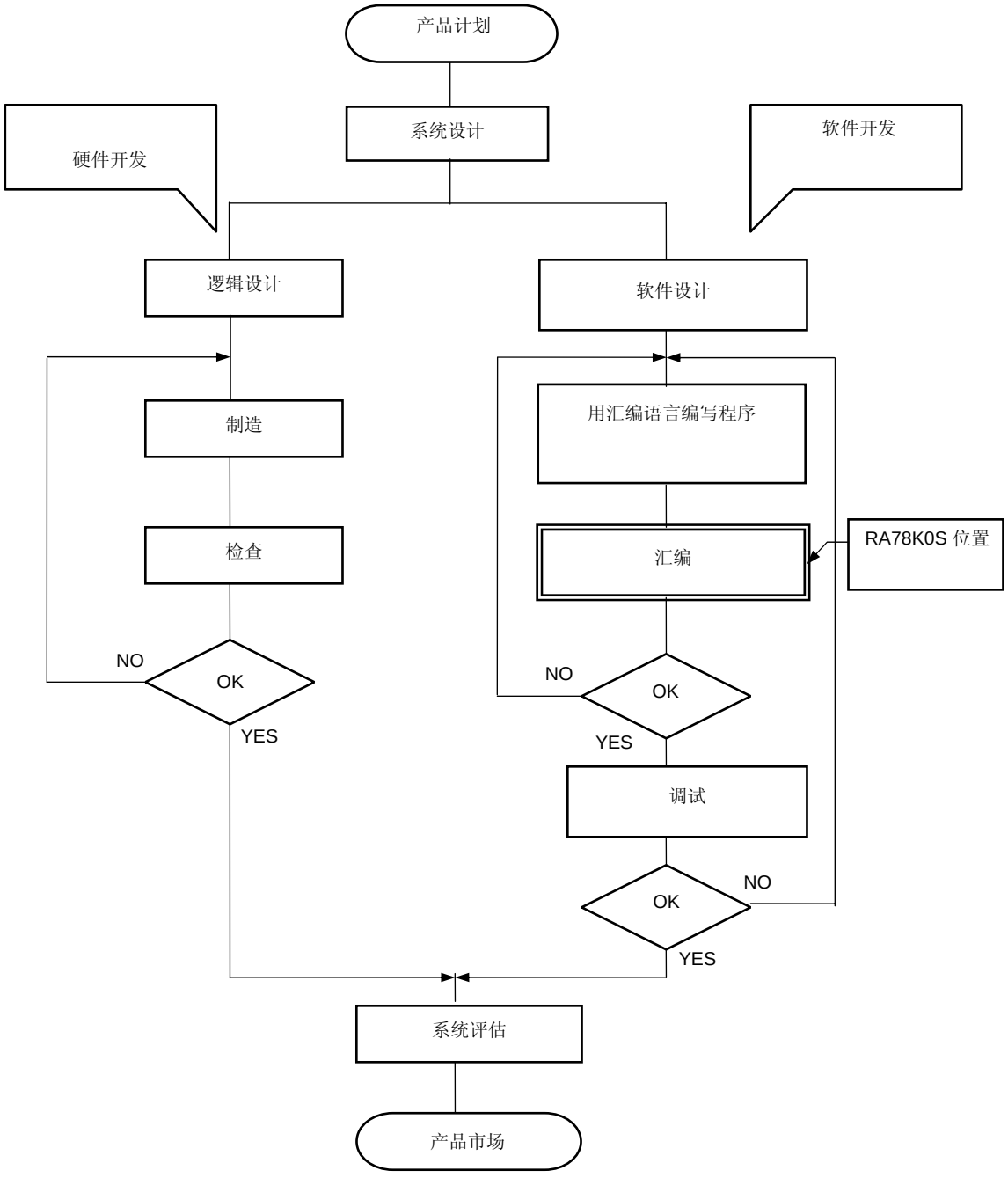
图 1-2. 汇编器流程



（2）　开发应用微控制器的产品及 **RA78K0S** 的作用

图 1-3 应用微控制器的产品的开发过程说明了汇编语言编程在（软件）产品开发过程中的位置。

图 1-3 应用微控制器产品的开发过程



1.1.2 什么是重定位汇编器?

由汇编器从源语言中翻译来的机器语言存储在微控制器的内存中以备使用。为此，必须提前确定各机器语言指令在内存中的存储位置。因此，要对汇编器汇编的机器语言增加一些信息，说明各机器语言指令在内存中的位置。

根据定位机器语言指令地址的方法不同，大体上，汇编器可分为绝对汇编器和可重定位汇编器。

- 绝对汇编器
  绝对汇编器把由汇编语言汇编的机器语言指令定位至绝对地址。
- 可重定位汇编器
  在可重定位汇编器中，暂时确定由汇编语言汇编的机器语言指令的定位地址。随后，通过一个称作连接器的程序来确定绝对地址。

过去，当程序用绝对汇编器创建后，程序员通常必须同时完成编程。然而，若一个大型程序的所有组成部分都在同一时间里创建，程序会变得复杂，使程序的维护和分析都很困难。为了避免这种情况，开发这类大型程序时，将其分成几个子程序对应每个功能单元，这些子程序被称为模块。这种编程技术称为模块化编程。

可重定位汇编器就是一个适于模块化编程的汇编器。

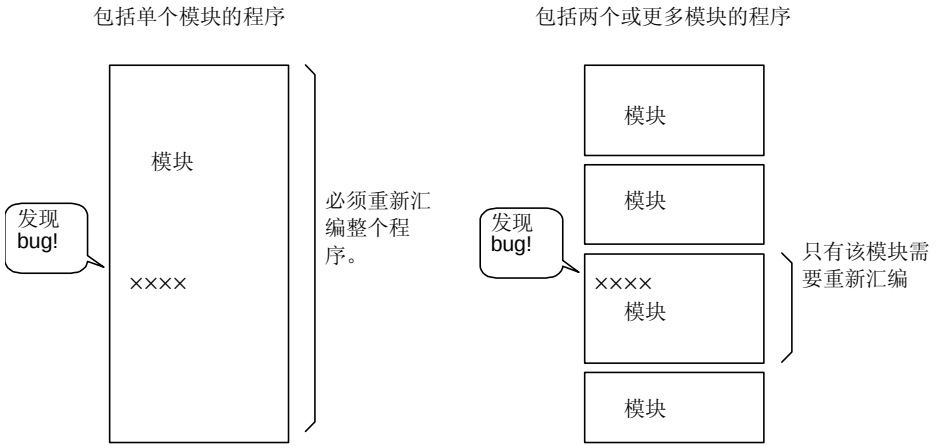
可重定位汇编器的模块化编程具有如下优点。

（1）提高开发效率

同时编写一个大型程序是困难的。在这种情况下，将程序的每个功能划分成几个模块，可保证两个或更多的程序员并行开发程序以开发提高效率。

而且，如果在程序中发现任何 bug，也不需要汇编整个程序，只需更正程序的 一部分，而且只有必须要更正的那个模块需被汇编。这就缩短了调试时间。

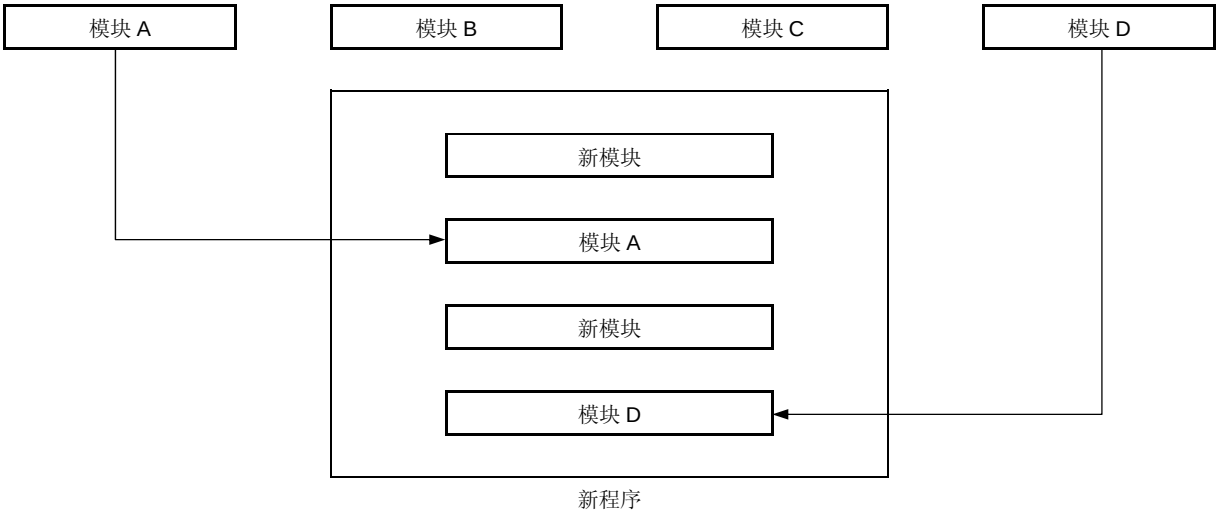
图 1-4.　重汇编调试



（2） 利用资源

已经创建好的具有高可靠性和多用途的模块可用来创建其它程序。收集这些具有多用途的模块作为软件资源，在开发新的程序时就节省了时间和精力。

图 1-5.　使用现有模块进行程序开发



1.2 程序开发之前的提醒

在开始开发程序前，记住下列要点。

1.2.1 RA78K0S 的最佳性能特征

（1）汇编器的最佳性能特征

表 1-1. 汇编器的最佳性能特征

项目	最佳性能特征	
	PC 版	WS 版
符号数（局部+全局）	65,535 符号注解 1	65,535 符号注解 1
交叉引用表可输出的符号数量	65,534 符号注解 2	65,534 符号注解 2
对于一个宏引用，宏程序体的最大空间	1 MB	1 MB
所有宏程序体的总空间	10 MB	10 MB
一个文件内段的数量	256 段	256 段
一个文件内宏和 include 的定义	10,000	10,000
一个 include 文件内宏和 include 的指定	10,000	10,000
重定位数据注解 3	65,535 项	65,535 项
行数量数据	65,535 项	65,535 项
一个文件内 BR 伪指令的数量	32,767 directives	32,767 directives
每行字符数	2,048 characters 注解 4	2,048 characters 注解 4
符号长度	256 characters	256 characters
定义Switch名的数量注解 5	1,000	1,000
Switch名的字符长度注解 5	31 characters	31 characters
一个文件内 include 文件的嵌套层数	8 levels	8 levels

注解 **1.** 采用 XMS。 如无 XMS， 则采用文件。

2. 采用内存，如无内存， 则采用文件

3. 重定位数据是指当汇编器不能确定符号的值时传递给连接器的数据。
比如，当用 MOV 指令指定一个外部引用时，在.rel 文件中生成两项重定位数据。

4. 包括回车代码和回车代码。如果一行为 2049 或更多个字符，则输出错误信息并中止处理操作。

5. 通过 SET/RESET 伪指令并使用$IF 等来设置 switch 名称的真/假

（2）连接器的最佳性能特征

表 1-2. 连接器的最佳性能特征

项目	最佳性能特征	
	PC 版	WS 版
符号数（局部+全局）	65,535 符号	65,535 符号
同一段的行数量数据	65,535 项目	65,535 项目
段的数量	65,535 段	65,535 段
输入模块的数量	1,024 模块	1,024 模块

1.3 RA78K0S 特点

RA78K0S 具有如下特点：

（1）宏功能

当同一组指令必须在一个源文件中反复出现时，可以为这一组指令赋予一个简单的宏名称来定义宏。

通过使用宏功能，可提到编码效率和程序的可读性。

（2）分支指令的优化功能

RA78K0S 有一个可自动选择分支指令的伪指令（BR 伪指令）。

为了创建具有高内存效率的程序，必须根据分支指令的分支目的范围描述一个字节分支指令。但是，对于程序员来说，通过注意每个分支的分支目的范围来描述分支指令却很困难。通过说明 BR 伪指令，汇编器根据分支目的范围生成适当的分支指令。这就是分支指令的优化功能。

（3）条件汇编功能

通过该功能，一部分源程序可根据预先确定的条件指定为汇编或非汇编。

如果在源程序中引入了调试语句，则调试语句是否应翻译成机器语言可通过为条件汇编设置一个开关来选择。如果不再需要调试语句，则汇编源程序时无需对该程序进行大的修改。

[备忘录]

第二章 如何描述源程序

本章介绍源程序的描述方法、书写格式、表达式和操作符。

2.1 源程序的基本配置

当把源程序分成几个模块进行编写时，每个模块变成汇编器的输入单元，称为源模块（如果源程序只包括一个模块，源程序就是源模块。）。

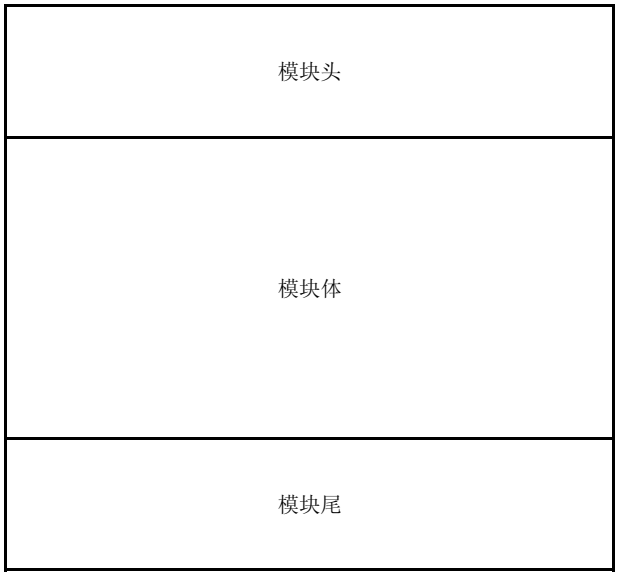
每个成为汇编器输入单元的源模块主要包括如下三个部分：

<1>模块头

<2>模块体

<3 >模块尾

图 2-1. 源模块的配置



2.1.1 模块头

在模块头中可描述在**表 2-1.可在模块头中描述的指令**中列出的控制指令。注意，这些控制指令仅用于模块头中。模块头也可省略。

表 2-1.可在模块头中描述的指令

可编写项	表达式	在该手册中的章/节
与汇编器选项具有同样功能的控制指令	与汇编器选项具有同样功能的控制指令如下： • PROCESSOR • XREF/NOXREF • DEBUG/NODEBUG/DEBUGA/NODEBUGA • TITLE • SYMLIST/NOSYMLIST • FORMFEED/NOFORMFEED • WIDTH • LENGTH • TAB	**第四章 控制指令**
由高级程序如 C 编译器和结构化汇编器预处理器等输出的特殊控制指令	由高级程序如 C 编译器和结构化汇编器预处理器等输出的特殊控制指令： • TOL_INF • DGS • DGL	

2.1.2 模块体

在程序体中不能描述下列指令：

- 与汇编器选项具有同样功能的控制指令

所有其他伪指令、控制指令和指令均可在模块体中描述。
模块体必须被划分为单元进行描述，这些单位称为“段”。
采用与每个段相对应的伪指令，用户可定义下列四种段：

<1> 代码段 .....必须采用 CSEG 伪指令定义
<2> 数据段 .....必须采用 DSEG 伪指令定义
<3> 位段.....必须采用 BSEG 伪指令定义
<4> 绝对段 .....必须采用 CSEG, DSEG, 或 BSEG 伪指令，为重定位属性（AT 定位地址）指定一个定位地址来定义。该段也可由 ORG 伪指令定义。

模块体可由任何段组合进行配置。
但是，应在代码段之前定义数据段和位段。

2.1.3 模块尾

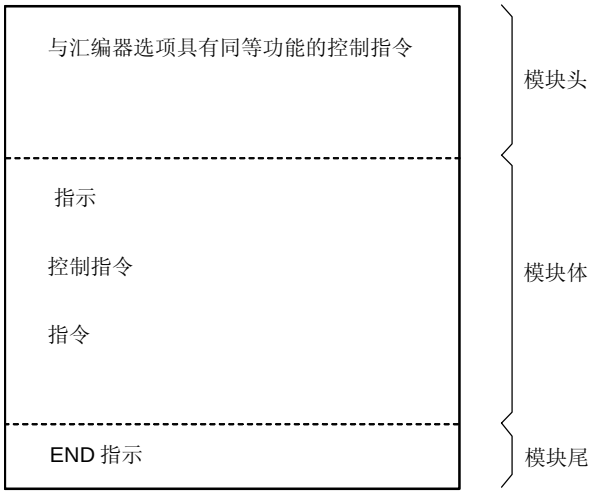
模块尾表示源模块的结束。END 伪指令必须出现在这部分。

除了注释、空格、制表符或换行码，如果在 END 伪指令后出现其他任何代码，汇编器都将输出告警信息，并忽略在 END 伪指令后出现的字符。

2.1.4 源程序整体配置

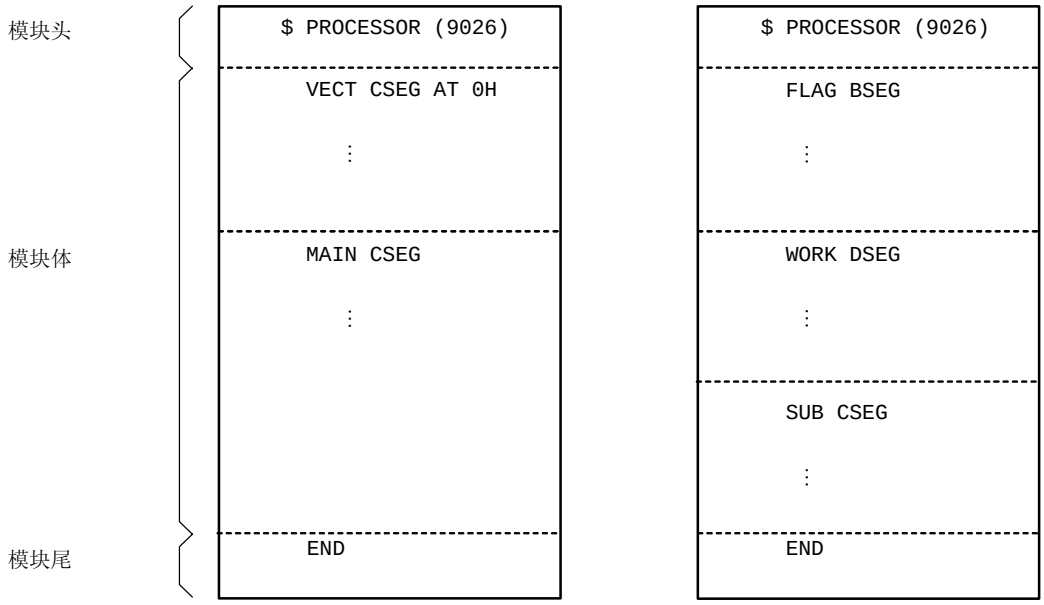
源模块（源程序）的总体配置如下所示。

图 2-2. 源程序的整体配置



简易源模块配置的示例如图 2-3 示。

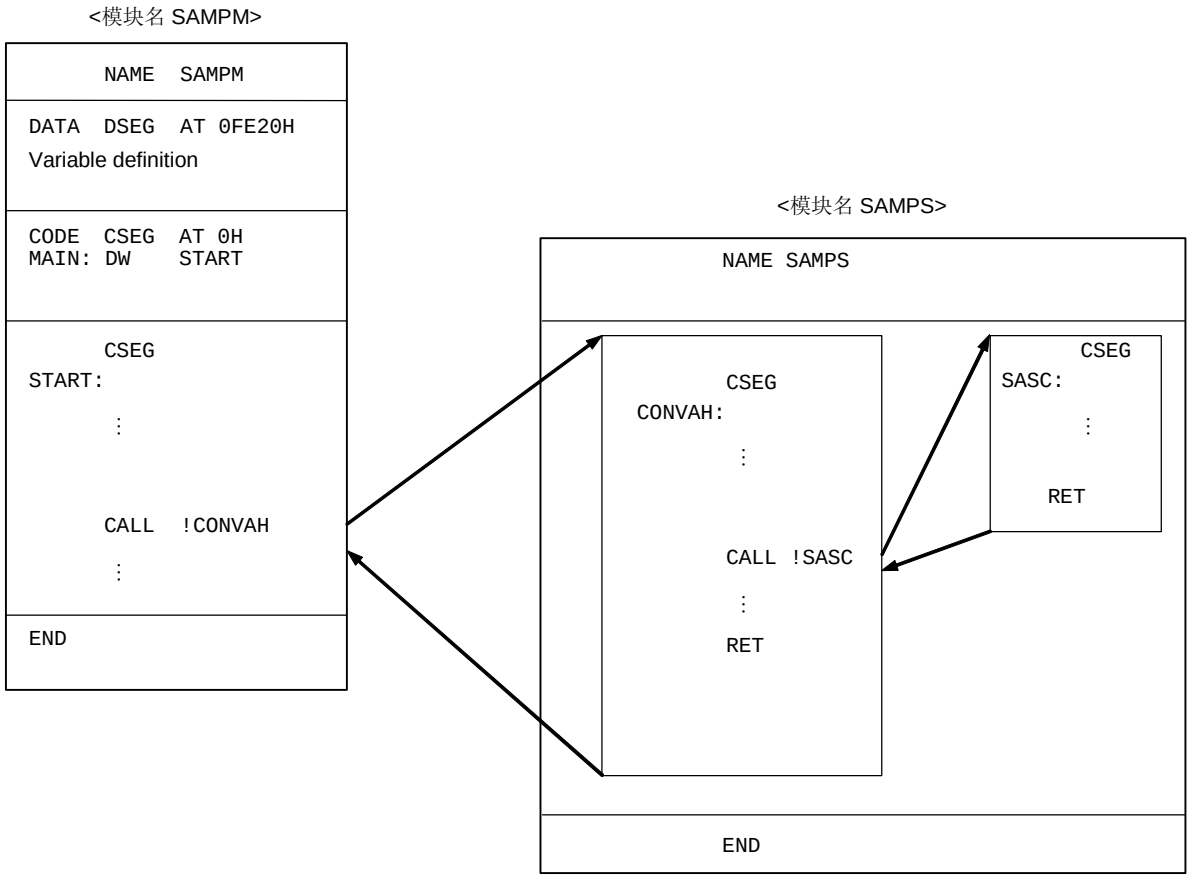
图 2-3. 源模块配置示例



2.1.5 源程序的编写示例

这一小节描述了源模块（源程序）作为源程序的编写实例。源程序的配置简单说明如下。

图 2-4. 源程序的配置



通过将一个简单的源程序分为两个模块完成了创建该样例程序。模块“SAMPM”是该程序的主程序，模块“SAMPS”是主程序内调用的子程序。

<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    AX,#_@STBEG
        MOVW    SP,AX

        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) 声明模块名
(2) 声明在其它模块中引用的符号作为一个外部引用符号
(3) 声明在其它模块中定义的符号作为一个外部引用符号
(4) 声明从链接器的 “-S”选项生成的栈解决符号，作为一个外部引用符号（如果链接时没有指定“-S”选项，将出错）
(5) 声明一个（位于 saddr 的）数据段的开始
(6) 声明一个（位于由地址 0H 起始的绝对段）代码段的开始
(7) 声明一个（标志绝对段结尾的）代码段的开始
(8) 声明模块结束

<子程序>

```
        NAME    SAMPS                                                   ;(9)
;*******************************************************
;*
;*  HEX -> ASCII Conversion Program
;*                  sub-routine
;*
;*  input condition  : (HL) <- hex 2 code
;*  output condition : BC-register <-ASCII 2 code
;*
;*******************************************************

        PUBLIC  CONVAH                                                  ;(10)

        CSEG                                                            ;(11)
CONVAH:
        MOV     A, [HL]
        ROR     A, 1
        ROR     A, 1
        ROR     A, 1
        ROR     A, 1
        AND     A, #0FH         ;hex upper code load
        CALL    !SASC
        MOV     B,A             ;store result

        XOR     A,A
        XCH     A, [HL]
        AND     A, #0FH         ;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
;*******************************************************

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

        END                                                             ;(12)
```

(9)　声明模块名

(10) 声明在其它模块中引用的符号作为一个外部引用符号

(11) 声明代码段的开始

(12) 声明模块结束

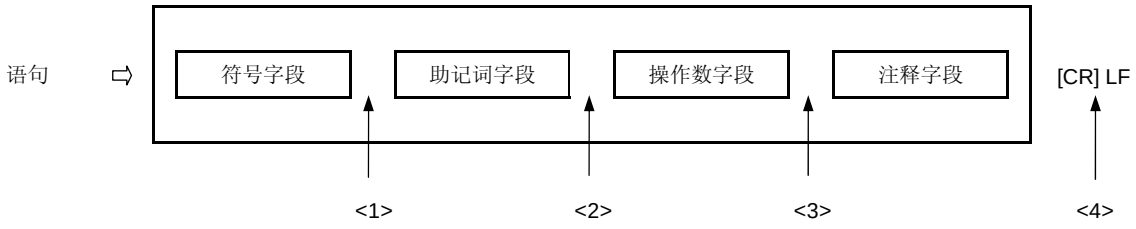
2.2 源程序的描述格式

2.2.1 语句的配置

程序由若干语句构成。

每个语句包括四个字段，如**图 2-5.组成语句的字段**所示。

图 2-5. 组成语句的字段



<1> 符号字段和助记词字段之间必须用冒号（:）或一个或多个空格或制表符分开。使用冒号或是空格取决于在助记词字段内记述的指令。

<2> 助记词字段和操作数字段之间必须用一个或多个空格或制表符分开。根据在助记词字段所记述的指令，可以不需要操作数字段。

<3> 使用注释字段时前面必须有分号。

<4> 每行之间必须用 LF 码界定开（一个 CR 码后可以紧跟着一个 LF 码）。

一条语句必须在一行内写完。每行可输入最多 2,048 个字符（包括 CR 和 LF）。

每个 TAB 或独立 CR 按一个独立字符计数。输入 2,049 或更多个字符时则输出告警信息，第 2,049 个及之后的任何字符都将被忽略。但是第 2,049 个及后续更多字符将输出至汇编列表中。

独立 CR 不输出至汇编列表。

也有可能描述如下代码行。

- 伪行（没有语句描述的行）
- 仅包含符号字段的行
- 仅包含注释字段的行

2.2.2 字符组

可在源文件中被描述的字符分为如下三类：

- 语言字符
- 字符数据
- 注释字符

(1) 语言字符

语言字符用于在源程序中记述指令。语言字符串组包括字母、数字和特殊字符。

[字母字符]

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

[特殊字符]

字符	名称	主要应用	
? @ _	问号 约号 下划线	相当于字母字符 相当于字母字符 相当于字母字符	
Blank HT (09H) , : ; CR (0DH) LF (0AH)	 制表符代码 逗号 冒号 分号 回车代码 换行代码	分隔符号	分隔每个字段 相当于空格字符 操作数分隔符 标记分隔符 表示注释字段开始的符号 表示一行结束的符号（在汇编器中被忽略） 表示一行的结束
+ - * / . (,) < , > =	加号 减号 星号 斜线 句点 左括号和右括号 不等号 等号	汇编器操作符	ADD 操作或正号 SUBTRACT 操作或负号 MULTIPLY 操作符 DIVIDE 操作符 比特位置指定符 指定将要执行的算术操作顺序的符号 关系操作符 关系操作符
'	单引号	• 伪指令字符串常量的开始或结束的符号 • 伪指令一个完整宏参量的符号	
$ & # ! [ ]	美元符号 与 警号 感叹号 方括号	• 伪指令位置计数器符号 • 伪指令一个相当于汇编器选项的控制指令开始的符号 • 伪指令相对地址的符号 串联符号（用于宏程序体） 指定立即寻址的符号 指定绝对寻址的符号 指定间接寻址的符号	

(2) 字符数据

字符数据指用于描述串常量、字符串和控制指令（TITLE, SUBTITLE, INCLUDE）的字符。

[用于字符数据的字符集]

- 除“00H”之外的所有字符均可使用，根据操作系统的不同，代码也不尽相同。输入“00H”将导致错误，并且在结束单引号标志（’）之前的后续字符都将被忽略。
- 如有任何非法字符输入，在输出至汇编列表时汇编器将用“!”代替这些非法字符（独立 CR（0DH）代码不输出至汇编列表）。
- 对 Windows 系统，汇编器把代码“1AH”当作文件的结束（EOF），因此，该代码不能作为输入数据的一部分。

(3) 注释字符

“注释字符”指用于表述注释语句的字符。

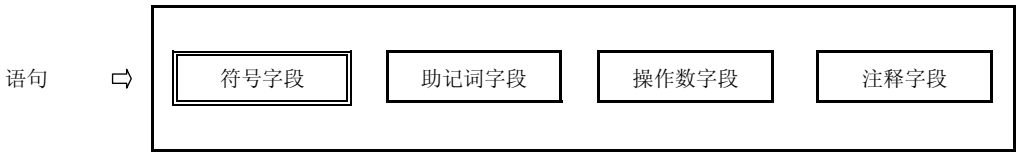
[用于注释的字符集]

- 可用在注释语句中的字符与用于字符数据的字符集相同。但是，即使“00H”码被输入也不会报错。相反，汇编器用“!”代替非法字符并输出至汇编器列表。

2.2.3 组成语句的字段

下面一节详细介绍了组成一条语句的各个字段。

(1) 符号字段



符号在符号字段中描述。术语“符号”指赋予数字数据或地址的名称。
符号的使用使源程序的内容更容易理解。

[符号类型]

根据用途和定义方法， 符号可分为如下几种类型，如表 2-2 示。

表 2-2. 符号类型

符号类型	用途	定义方法
名称	在源程序中用作数字数据或地址	在 EQU, SET 或 DBIT 伪指令的符号字段中描述
标记	在源程序中用作地址数据	在一个符号后加后缀（:）来定义
外部引用名称	在一个模块定义而被另一模块用作引用符号	在 EXTRN 或 EXTBIT 伪指令的操作符字段中描述
段名称	链接操作时使用的符号	在 CSEG, DSEG, BSEG or ORG 伪指令的符号字段中定义
模块名称	符号调试时使用	在 NAME 伪指令的操作数字段描述
宏名称	在源程序中用作宏引用	在 MACRO 伪指令的符号字段描述

[符号描述惯例]

所有符号在描述时需遵循如下规则：

<1> 符号必须由字母字符和可用作等效字母字符的特殊字符（?, @,和 _）构成。数字字符 0-9 中的任何一个都不能作为符号的第一个字符。

<2> 符号最多由 31 个字符构成。超出最大符号长度的字符将被忽略。

<3> 符号中不能使用保留字。**附录 A 保留字列表**指出了保留字。

<4> 同一符号不能定义多次（但是，由 SET 伪指令定义的名称可通过 SET 伪指令重新定义）。

<5> 汇编器区分小写字符和大写字符。

<6> 当在符号字段描述一个标记时，冒号":"紧接在标记后输入。

（正确的符号表述示例）

```
CODE01      CSEG          ; "CODE01"是段名
VAR01       EQU   10H     ; "VAR01"是名称
LAB01:      DW    0       ; "VAR01"表示标记
            NAME SAMPLE   ; "SAMPLE"是模块名
MAC1        MACRO         ; "MAC1"是宏名称。
```

（错误的符号表述示例）

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

（过长的符号示例）

```
A123456789B12 to Y1234567890123456      EQU      70H
```

250

; 字符"6"超出了最大符号长度（256 字符），将被忽略。符号将被定义为"A123456789B12 to Y123456789012345"。

（仅包括一个符号的语句示例）

```
ABCD:                   ; "ABCD"将被定义成一个标记。
```

[关于符号的一些警告]

每次在宏程序体内使用了本地符号时，符号"??RAnnnn (n = 0000 to FFFF)"则自动被汇编器代替。需注意该符号不能被定义两次。

当段名不是被段定义伪指令指定时，汇编器自动产生段名。这些段如**表 2-3 汇编器自动生成的段名称**中显示。

重复定义段名则产生错误。

表 2-3 汇编器自动生成的段的名称

段名称	伪指令	重定义属性
?An	ORG 伪指令	n = 0000 to FFFF
?CSEG	CSEG 伪指令	UNIT
?CSEGUP		UNITP
?CSEGT0		CALLT0
?CSEGFX		FIXED
?CSEGIX		IXRAM
?DSEG	DSEG 伪指令	UNIT
?DSEGUP		UNITP
?DSEGS		SADDR
?DSEGSP		SADDRP
?DSEGIH		IHRAM
?DSEGL		LRAM
?DSEGDSP		DSPRAM
?BSEG	BSEG 伪指令	UNIT

[符号属性]

所有名称和标记都有一个值和一个属性。

值表示被定义的数字数据或地址数据本身。

段名、模块名和宏名没有值。

一个符号具有的属性称为符号属性，必须是下面 8 种类型之一，如**表 2-4 符号属性和值**所示。

表 2-4. 符号的属性和值

属性类型	分类	值
NUMBER	• 分配给数字常量的名称 • 由 EXTRN 伪指令定义的符号 • 数字常量	十进制表示法： 0 to 65535 十六进制表示法： 0H to FFFFH
ADDRESS	• 作为标记定义的符号 • 由 EQU 和 SET 伪指令作为标记定义的名称	
BIT	• 作为比特值定义的名称 • BSEG 中的名称 • 由 EXTBIT 伪指令定义的符号	Saddr 区
CSEG	由 CSEG 伪指令定义的段名	这些属性类型没有值
DSEG	由 DSEG 伪指令定义的段名	
BSEG	由 BSEG 伪指令定义的段名	
MODULE	由 NAME 伪指令定义的段名（如果模块名没有定义，而从输入源文件名的名称中创建）	
MACRO	由 MACRO 伪指令定义的宏名称	

示例

```
TENEQU  10H     ;名称“TEN”具有属性“NUMBER”，值为“10H”。
        ORG     80H
START:  MOV     A,#10H      ; 标记“START”具有属性“ADDRESS”，值为“80H”。
BIT1    EQU     0FE20H.0    ;名称“BIT1”具有属性“BIT”，值为“0FE20H.0”。
```

(2) 助记词字段

助记词字段描述记忆指令、伪指令或宏引用。
由于指令或伪指令需要一个或多个操作数，助记词字段必须与操作数字段用一个或多个空格或制表符分离开来。
但是，即使助记词字段和第一个操作数字段之间没有用空格等隔开，若伪指令的第一个操作数以“#”, “$” ,“!”, or “[”开头，汇编也将会被正确执行。

（正确描述示例）

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

（错误描述示例）

```
        MOVA    #0H         ; 助记词字段和操作数字段之间无空格
        C ALL   ! CONVAH    ;助记词字段内有空格.
        ZZZ                 ; 78K/0S 没有如“ZZZ”这样的指令
```

(3) 操作数字段

操作数字段描述了执行指令、伪指令或宏引用时需要的数据（操作数）。
根据指令或伪指令的不同，操作数字段不需要操作数，或者，两个或多个操作数必须在一个操作数字段进行描述。
当描述两个或多个操作数时，用逗号（,）把每个操作数分隔开。
下列数据类型可在操作数字段描述：

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

所需操作数的大小和属性可依据指令或伪指令的不同而不同。关于操作数的大小和属性，参见 **2.5 操作数的特征。**
关于指令组中操作数的表示格式和描述方法，参见目前正对其进行软件开发的微控制器用户手册。
在操作数字段可被描述的各数据类型详列如下。

[常量]
常量是一个固定的值或数据项，也被称为立即数。
常量分为数字常量和字符串常量。

- 数字常量
  二进制数、八进制数、十进制数或十六进制数都可被认为是数字常量。每个数字常量类型的表示方法见**表 2-5 数字常量表示法**。
  数字常量作为无符号 16 位数来处理。

  值范围：0　n　65,535 (0FFFFH)

  当描述一个负数时，使用减号操作符。

表 2-5 数字常量表示法。

常量	表示法	示例
二进制常量	• 在数字值加字符“B” 或 “Y”作为后缀。	1101B 1101Y
八进制常量	• 在数字值加字符“O” 或 “Q”作为后缀。	74O 74Q
十进制常量	• 数字值形式保持不变，或在数字值加字符“D” 或 “T”作为后缀	128 128D 128T
十六进制常量	• 在数字值加字符“H”作为后缀 • 如果首字符以“A”, “B”, “C”, “D”, “E”，或 “F”开始，该常量钱必须加前缀“0”。	8CH 0A6H

- 字符串常量
  字符串常量由包含在一对单引号（’）内的一串字符表示，字符已在 **2.2.2 字符集**中说明
  作为一个汇编处理结果，字符串常量被转换成 7 位 ASCII 码，优先级位（MSB）设为“0”。
  串常量的长度为 0-2 个字符。
  对于使用单引号标志本身作为串常量的情形，单引号标志必须被连续输入两次。

字符串常量表述示例：

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

[字符串]

字符串由包在一对单引号内的一串字符表示，字符已在 **2.2.2 字符集**中说明。字符串主要用作 DB 伪指令和 TITLE 或 SUBTITLE 控制指令的操作数。

- 字符串的应用实例

```
        CSEG
MAS1:   DB      'YES'       ; 初始化字符串“YES”
MAS2:   DB      'NO'        ; 初始化字符串“NO ”
```

[寄存器名]

下列寄存器可在操作数字段中描述。

- 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用其绝对名称（R0 到 R7、RP0 到 RP3）以及其功能名称（X, A, B, C, D, E, H, L, AX, BC, DE, HL）来描述。

依据指令类型的不同，在操作数字段中可被描述的寄存器名称也可能不同。有关每个寄存器名称描述方法的更多细节，参见正对其进行软件开发的微控制器用户手册。

[特殊字符]

可在操作数字段描述的特殊字符如**表 2-6 可在操作数字段表述的特殊字符** 示。

表 2-6 可在操作数字段表述的特殊字符

特殊字符	功能
$	• 表示具有该操作符的指令的定位地址（或在具有多字节指令的地址中，标识该地址的第一个字节） • 表示一个分支指令的相对寻址模式
!	• 表示一个分支指令的绝对寻址模式 • 表示 addr16 规定，即允许所有内存空间由 MOV 伪指令指定
#	• 表示立即数
[ ]	• 表示间接寻址模式

- 特殊字符应用示例

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

<1> 操作符中的第二个$表示地址 103H。输入“BR $-1”导致同样的操作。
<2> 操作符中的第二个$表示地址 105H。输入“BR $-1+100H”导致同样的操作。

[段定义伪指令的重定位属性]

操作数字段可表述重定义属性。
关于重定义属性的更多细节，参见 **3.2 段定义伪指令**。

[符号]

如果在操作数字段中描述了符号，分配给该符号的地址或值变成了操作数的值。

- 符号应用示例

```
VALUE   EQU     100H
        MOV     A, #VALUE   ;该语句可写作“MOV A, #100H”。
```

[表达式]

表达式是由操作符连起来的一组常量、$（伪指令位置地址）、名称或标记。
在数字值可被表示成指令操作符的地方可以描述表达式。
关于表达式和操作符，参见 **2.3 表达式和操作符**。

- 表达式示例

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

在这个例子中，“TEN-5H”是一个表达式。
该表达式中，名字和数字常量通过一个减号相连。表达式的值为 BH。
因此，该语句也可重新写成“MOV A, #0BH”。

[位术语]

通过比特位置伪指令符可获得一个位术语。关于位术语的更多细节，参见 **2.4 比特位置伪指令符**。

- 位术语示例

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

(4) 注释字段

在注释字段里，注释或备注在输入分号（;）以后进行描述。注释字段的范围从分号到该行的换行代码或 EOF。在注释字段内描述注释语句，可创建一个易于理解的源程序。注释字段的注释语句不受汇编器操作的影响（也就是说，不转换成机器语言），但是将无改变地输出至汇编列表中。

可在注释字段里描述的字符如 **2.2.2 字符集**所示。

（注释示例）

```
        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    AX, #_@STBEG
        MOVW    SP, AX

        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)
```

仅包括注释字段的行

注释在注释字段中被描述的行

2.3 表达式和操作符

表达式由一个或多个操作符将符号、常量、位置地址(由$伪指令)或位术语组合而成。
表达式中，除了操作符之外的元素称为项，叫做第一项、第二项，从左到右，按照他们的显示顺序类推。
表 2-7 操作符类型示出了可用操作符类型。计算的优先级顺序预先确定，见**表 2-8 操作符优先级顺序**。
圆括号“( )”用于改变计算执行的顺序。

示例:　MOV A, #5* (SYM+1)　　; <1>

在上面的<1> 中， “5* (SYM+1)” 是表达式。 “5”是表达式的第一项， “SYM” 和 “1” 分别是第二项和第三项. “*”,“+”和“( )” 是操作符.

表 2-7. 操作符类型

操作符类型	操作符
算术操作符	+ sign, – sign, +, –, *, /, MOD
逻辑操作符	NOT, AND, OR, XOR
关系操作符	EQ or =, NE or < >, GT or >, GE or >=, LT or <, LE or <=
移位操作符	SHR, SHL
字节分割操作符	HIGH, LOW
特殊操作符	DATAPOS, BITPOS, MASK
其它操作符	(　)

上述操作符也可分为一元操作符、特殊一元操作符、二元操作符、N-元操作符和其它操作符。

一元操作符：　+ 号, – 号, NOT, HIGH, LOW
特殊一元操作符：　DATAPOS, BITPOS
二元操作符：　+, –, *, /, MOD, AND, OR, XOR, EQ or =, NE or < >, GT or >, GE or >=, LT or <, LE or <=, SHR, SHL
N-元操作符：　MASK
其它操作符：　(　)

表 2-8. 操作符优先级顺序

优先级	优先级等级	操作符
较高 ↕ 较低	1	+ sign, – sign, NOT, HIGH, LOW, DATAPOS, BITPOS, MASK
	2	*, /, MOD, SHR, SHL
	3	+, –
	4	AND
	5	OR, XOR
	6	EQ or =, NE or < >, GT or >, GE or >=, LT or <, LE or <=

根据如下原则执行表达式的操作：

<1> 根据赋予每个操作符的优先级顺序执行操作。如果表达式中有两个或多个操作符具有同一优先级，则执行最左边操作符指定的操作。如果是一元操作符，则从右至左执行操作。
<2> 先执行圆括号内的表达式，再执行圆括号外的表达式。
<3> 允许两个或多个一元表达式之间的操作。

示例： 1=––1==1
–1=–+1=–1

<4> 表达式的计算结果在 16 位以内，无符号。如果由于表达式超出 16 位而使操作溢出，则忽略溢出值。
<5> 如果一个常量超出 16 位(0FFFFH)，将出错，且计算时结果值计为 0。
<6> 在做除法运算时，结果的十进制小数部分被删掉。如果除数为 0，则出错，结果为 0。

2.3.1 操作符的功能

本节介绍各操作符的功能。

(1) + (ADD)操作符

[功能]

返回表达式的第一项和第二项的值的和。

[应用示例]

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

[说明]

BR 指令引起跳转至“当前位置地址+6”，即，跳转至“100H+6H=106H”。因此，上述例子中的(a)也可表示成：START: BR !106H。

(2) – (SUBTRACT)操作符

[功能]

返回从第一项的值中减去第二项值的结果。

[应用示例]

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

[说明]

BR 指令引起跳转至“BACK 的地址-6”，即，跳转至“100H-6H=0FAH”。因此，上述例子中的(b)也可表示成：BACK: BR !0FAH。

(3) * (MULTIPLY)操作符

[功能]

返回表达式的第一项和第二项的值的乘积的结果。

[应用示例]

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

[说明]

通过 EQU 伪指令，定义名称 “TEN”的值是“10H”。

“#”表示立即数。表达式“TEN*3”等同于“10H*3”，返回结果值为“30H”。

因此，上述表达式中的(c)也可表示作：MOV A,#30H

(4) / (DIVIDE)操作符

[功能]

用表达式第二项的值去除第一项的值，返回结果的整数部分。结果的十进制小数部分被删掉。如果除法操作的除数(第二项)是 0，则出错。

[应用示例]

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

[说明]

除法式“256/50”的结果是 5，余数是 6。

操作符返回除式结果的整数部分，即值“5”。

因此，上述表达式的(d)也可表示成：MOV A,#5

(5) MOD(余数)操作

[功能]

得到表达式的第一项值除以第二项值的结果的余数。

如果除数(第二项)为 0，则出错。

MOD 操作符前后需有空格符。

[应用示例]

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

[说明]

除法式“256/50”的结果是 5，余数是 6。

MOD 操作符返回余数“6”。

因此，上述表达式的(e)也可表示成：MOV A,#6。

(6) + 号

[功能]

不改变表达式的项的值，且返回

[应用示例]

```
FIVE    EQU    +5
```

[说明]

不改变该项的值“5”，且返回。

值“5”在名称“FIVE”中用 EQU 伪指令定义。

(7) –号

[功能]

返回表达式的项以 2 为基数的值的补集。

[应用示例]

```
NO    EQU    -1
```

[说明]

-1 是 1 的以 2 为基数的补集。

二进制数 0000 0000 0000 0001 以 2 为基数的补集是 1111 1111 1111 1111。

因此，通过 EQU 伪指令，值“0FFFFH”定义给名称“NO”。

(1) NOT 操作符(取反)

[功能]

表达式的项的值逐位取反，并返回结果值。

NOT 操作符和项之间用空格分开。

[应用示例]

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

[说明]

“3H”的逻辑取反的执行过程如下：

```
NOT)      0000 0000 0000 0011
-------------------------------
          1111 1111 1111 1100
```

返回 0FFFCH。

因此，(a)也可表示成：MOVW AX, #0FFFCH

(2) AND 操作符(逻辑乘)

[功能]

表达式的第一项的值和第二项的值之间逐位执行 AND(逻辑乘)操作，并返回结果值。

AND 操作符前后都需要加空格。

[应用示例]

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

[说明]

在“6FAH” 和 “0FH”这两个值之间执行 AND 操作如下：

```
          0000 0110 1111 1010
AND)      0000 0000 0000 1111
-------------------------------
          0000 0000 0000 1010
```

返回结果 0AH。因此，上述表达式中的(b) 也可表示成：MOV A, #0AH

(3) OR 操作(逻辑和)

[功能]

表达式的第一项的值和第二项的值之间逐位执行 OR(逻辑和)操作并返回结果。

OR 操作符前后都需要加空格

[应用示例]

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

[说明]

在“0AH” 和 “1101B”这两个值之间执行 OR 操作如下：

```
              0000 0000 0000 1010
OR)           0000 0000 0000 1101
-------------------------------------
              0000 0000 0000 1111
```

返回结果 0FH。因此，上述表达式的(c)也可表示成： MOV A, #0FH

(4) XOR 操作(互斥逻辑和)

[功能]

表达式的第一项的值和第二项的值之间逐位执行 XOR(互斥逻辑和)操作并返回结果。

XOR 操作符前后都需要加空格

[应用示例]

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

[Explanation] [说明]

在“9AH” 和“9DH” 这两个值之间执行 XOR 操作如下：

```
              0000 0000 1001 1010
XOR)          0000 0000 1001 1101
-------------------------------------
              0000 0000 0000 0111
```

返回结果 7FH。因此，上述表达式的(d)也可表示成： MOV A, #7H

(1) EQ 或 =(等于)操作

[功能]

如果表达式第一项的值等于第二项的值，返回 0FFH(真)；如果不等，返回 00H(假)。

EQ 操作符前后都需要加空格。

[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]

上面(a)中，表达式“A1 EQ (A2+4H)”变成“12C4H EQ (12C0H+4H)”。

由于第一项的值等于第二项的值，操作符返回 0FFH。

上面(b)中，表达式“A1 EQ A2”变成“12C4H EQ 12C0H”。

由于第一项的值不等于第二项的值，操作符返回 00H。

(2) NE 或 < > (不等)操作符

[功能]

如果表达式第一项的值不等于第二项的值，返回 0FFH(真)；如果相等，返回 00H(假)。

NE 操作符前后都需要加空格。

[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]

上面(c)中，表达式“A1 NE A2”变成 “5678H NE 5670H”。

由于第一项的值不等于第二项的值，操作符返回 0FFH。

上面(d)中，表达式“A1 NE (A2+8H)”变成“5678H NE (5670H+8H)”。

由于第一项的值等于第二项的值，操作符返回 00H。

(3) GT 或 > (大于) 操作符

[功能]

如果表达式第一项的值大于第二项的值，返回 0FFH(真)；如果相等或小于第二项的值，返回 00H(假)。
GT 操作符前后都需要加空格。

[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]

上面(e)中，表达式“A1 GT A2”变成“1023H GT 1013H”。
由于第一项的值大于第二项的值，操作符返回 0FFH。
上面(f)中，表达式“A1 GT (A2+10H)”变成“1023H GT (1013H+10H)”。
由于第一项的值等于第二项的值，操作符返回 00H。

(4) GE 或 >= (大于等于) 操作符

[功能]

如果表达式第一项的值大于或等于第二项的值，返回 0FFH(真)；如果第一项的值小于第二项的值，返回 00H(假)。
GE 操作符前后都需要加空格。

[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]

上面(g)中，表达式“A1 GE A2”变成“2037H GE 2015H”。
由于第一项的值大于第二项的值，操作符返回 0FFH。
上面(h)中，表达式“A1 GE (A2+23H)”变成“2037H GE (2015H+23H)”。
由于第一项的值小于第二项的值，操作符返回 00H。

(5) LT 或 < (小于) 操作符

[功能]

如果表达式第一项的值小于第二项的值，返回 0FFH(真)；如果相等或大于第二项的值，返回 00H(假)。
LT 操作符前后都需要加空格。

[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]

上面(i)中，表达式“A1 LT A2”变成“1000H LT 1020H”。
由于第一项的值小于第二项的值，操作符返回 0FFH。
上面(j)中，表达式“(A1+20H) LT A2”变成““(1000H+20H) LT 1020H”。
由于第一项的值等于第二项的值，操作符返回 00H。

(6) LE 或 <= (小于等于) 操作符

[功能]

如果表达式第一项的值小于或等于第二项的值，返回 0FFH(真)；如果第一项的值大于第二项的值，返回 00H(假)。
LE 操作符前后都需要加空格。

[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]

上面(k)中，表达式“A1 LE A2”变成“103AH LE 1040H”。
由于第一项的值小于第二项的值，操作符返回 0FFH。
上面(l)中，表达式“(A1+7H) LE A2”变成“(103AH+7H) LE 1040H”。
由于第一项的值大于第二项的值，操作符返回 00H。

(1) SHR (右移)操作符

[功能]

表达式第一项的值向右移动表达式第二项所指定的位数，并返回右移结果。与指定的移位位数等值的零向右移入高位。

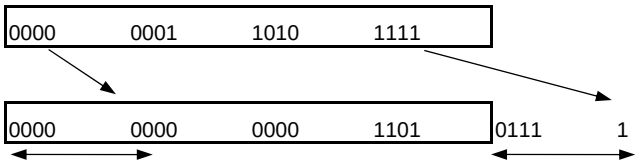
SHR 操作符前后都需要加空格。

[应用示例]

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

[说明]

该操作符将值“01AFH”向右移动 5 位。



插入 0　　右移 5 位

返回值“000DH”。

因此，上面例子中的(a)可表示成：MOV A, #0DH

(2) SHL (左移)操作符

[功能]

表达式第一项的值向左移动表达式第二项所指定的位数，并返回左移结果。与指定的移位位数等值的零相左移入低位。

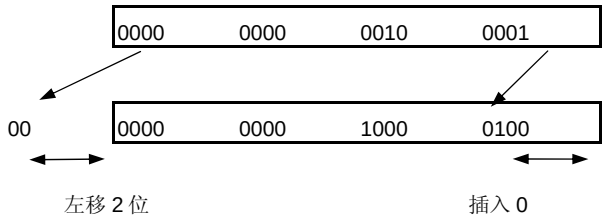
SHL 操作符前后都需要加空格。

[应用示例]

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

[表达式]

该操作符将值“21FH”向左移动 2 位。



返回值“84H”。

因此，上面例子中的(b)可表示成：MOV A, #84H

(1) HIGH 操作符

[功能]

返回一个项的高 8 位值

HIGH 操作符和该项之间需有空格。

[应用示例]

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

[说明]

执行 MOV 指令时，该操作符返回表达式“1234H”的高 8 位值“12H”。

因此，上面例子中的(a)也可表示为：MOV A, #12H

(2) LOW 操作符

[功能]

返回一个项的低 8 位值。

LOW 操作符和项之间需有空格。

[应用示例]

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

[说明]

执行 MOV 指令时，该操作符返回表达式“1234H”的低 8 位值“34H”。

因此，上面例子中的(b)也可表示为：MOV A, #34H

(1) DATAPOS

[功能]

返回一个位符号的地址部分(字节地址)。

[应用示例]

```
SYM    EQU    0FE68H.6

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

[说明]

EQU 伪指令定义了名称“SYM”，值为 0FE68H.6。

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

因此，上面例子的(a)可表示成：MOV A, !0FE68H

(2) BITPOS

[功能]

返回一个位符号的位部分(比特位置)。

[应用示例]

```
SYM    EQU    0FE68H.6

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

[说明]

EQU 伪指令定义了名称“SYM”，值为 0FE68H.6。

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

CLR1 指令清除[HL].6 为 0。

(3) MASK

[功能]

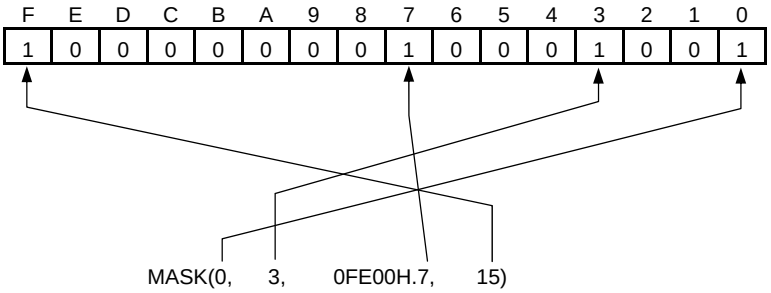
返回一个 16 位值，其中指定比特位置 1，所有其它比特位置 0

[应用示例]

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

[说明]

MOVW 指令返回值“8089H”。



(1) ( )

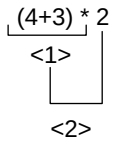
[功能]

使圆括号内的操作比圆括号外的操作先执行。
该操作符用于改变其它操作符的优先级顺序。
如果圆括号嵌套多层，在最内层圆括号的表达式被首先执行。

[应用示例]

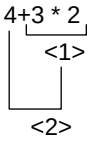
```
MOV  A, #(4+3)*2
```

[说明]



按照表达式<1>和表达式 <2>的顺序计算，返回结果值“14”。

如果没有圆括号



按照表达式<1>和表达式 <2>的顺序计算，返回结果值“10”。
关于操作符的优先级顺序，参见**表 2-8 操作符的优先级顺序。**

2.3.2 操作约束

用操作符连接表达式不同的项来执行表达式的操作。可被称为项的元素包括常量、$、名称和标记。每个项都有一个重定位属性和符号属性。

每个项固有的重定位属性和符号属性的类型限制了可在该项上进行操作的操作符。因此，当描述一个表达式时，很重要的一点是，需要注意构成这个表达式的每一项的重定位属性和符号属性。

(1) 操作符和重定位属性

正如前面提到的，构成一个表达式的每一项都具有重定位属性和符号属性。

当根据重定位属性进行分类时，项可分为三种类型：绝对项、重定位项和外部引用项。

操作的重定位属性类型、每个属性的特点以及可应用于每个属性的项列于**表 2-9. 重定位属性的类型**。

表 2-9.　重定位属性的类型

类型	特点	可应用项
绝对项	值和常量在汇编时确定的项	• 常量 • 在绝对段中定义的标记 • $表示在绝对段中定义的位置地址 • 与常量、上述标记、上述$或绝对值一起定义的名称
重定位项	值和常量不在汇编时确定的项	• 在重定位段中定义的标记 • $表示在重定位段中定义的位置地址 • 与重定位符号一起定义的名称
外部引用项[注释]	被另一模块作为外部引用符号的项	• 由 EXTRN 伪指令定义的标记 • 由 EXTBIT 伪指令定义的标记

注释　下面四个操作符可在外部引用项上操作：‘+’, ‘–’, ‘HIGH’, and ‘LOW’。在一个表达式中，只可表述一个外部引用符号。这种情况下，外部引用符号必须用“+”操作符连接。

表 2-10 用重定位属性组合项和操作符给出了操作符类型和每个操作符可对其进行操作的项的组合。

表 2-10 按照重定位属性区分的项和操作符的组合(1/2)

项的重定位属性 / 操作符类型	X:ABS Y:ABS	X:ABS Y:REL	X:REL Y:ABS	X:REL Y:REL
X + Y	A	R	R	—
X – Y	A	—	R	A[注释]
X * Y	A	—	—	—
X / Y	A	—	—	—
X MOD Y	A	—	—	—
X SHL Y	A	—	—	—
X SHR Y	A	—	—	—
X EQ Y	A	—	—	A[注释]
X LT Y	A	—	—	A[注释]
X LE Y	A	—	—	A[注释]
X GT Y	A	—	—	A[注释]
X GE Y	A	—	—	A[注释]
X NE Y	A	—	—	A[注释]
X AND Y	A	—	—	—
X OR Y	A	—	—	—
X XOR Y	A	—	—	—
NOT X	A	A	—	—
+ X	A	A	R	R
– X	A	A	—	—

<说明>

ABS：绝对项

REL：重定位项

A： 操作结果变成绝对项

R： 操作结果变成可重定位项

—： 操作不可执行

注释 仅当 X 或 Y 不是 HIGH, LOW, DATAPOS 上操作的可重定位项，并且 X 和 Y 都在同一段内定义时才执行该操作。

表 2-10 用重定位属性组合项和操作符(2/2)

项的重定位属性 / 操作符类型	X:ABS Y:ABS	X:ABS Y:REL	X:REL Y:ABS	X:REL Y:REL
HIGH X	A	A	R[注释]	R[注释]
LOW X	A	A	R[注释]	R[注释]
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
MASK (X)	A	A	—	—

<说明>

ABS：绝对项

REL：重定位项

A：　操作结果变成绝对项

R：　操作结果变成可重定位项

—：　操作不可执行

注释 仅当 X 或 Y 不是 HIGH, LOW, DATAPOS 上操作的可重定位项时，该操作才可执行。

下面四个操作符可用于外部引用项：‘+’, ‘–’, ‘HIGH’, 和 ‘LOW’(但是，表达式中只可表述一个外部引用项)。

表 2-11 用重定位属性组合项和操作符 (外部引用项)给出了操作符类型和每个操作符可进行操作的外部引用项的组合。

表 2-11 用重定位属性组合项和操作符 (外部引用项)

项的重定位属性 / 操作符类型	X:ABS Y:EXT	X:EXT Y:ABS	X:REL Y:EXT	X:EXT Y:REL	X:EXT Y:EXT
X + Y	E	E	—	—	—
X – Y	—	E	—	—	—
+ X	A	E	R	E	E
HIGH X	A	E[注释 1]	R[注释 2]	E[注释 1]	E[注释 1]
LOW X	A	E[注释 1]	R[注释 2]	E[注释 1]	E[注释 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

<说明>

ABS：A 绝对项

REL：可重定位项

A： 操作结果变成一个绝对项

E： 操作结果变成一个外部引用项

R： 操作结果变成一个可重定位项

—: 操作不可执行

注释 1. 仅当 X 或 Y 不是 HIGH, LOW, DATAPOS、BITPOS 上操作的外部引用项时，才执行该操作。

2. 仅当 X 或 Y 不是 HIGH, LOW, DATAPOS 上操作的可重定位项时，才执行该操作。

(2) 操作符和符号属性

正如前面描述的，构成表达式的每一项除了具有重定位属性，还具有符号属性。当根据符号属性进行分类时，项可分为两类：NUMBER 项和 ADDRESS 项。

操作中的符号属性类型和可用于每个属性的项示于**表 2-12. 操作中的符号属性类型**。

表 2-12. 操作中的符号属性类型

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

表 2-13 用符号属性组合项和操作符给出了操作符类型和根据符号属性分类时每个操作符可进行操作的项的组合。

表 2-13 用符号属性组合项和操作符

项的符号属性 / 操作符类型	X:ADDRESS Y:ADDRESS	X:ADDRESS Y:NUMBER	X:NUMBER Y:ADDRESS	X:NUMBER Y:NUMBER
X + Y	—	A	A	N
X – Y	N	A	—	N
X * Y	—	—	—	N
X / Y	—	—	—	N
X MOD Y	—	—	—	N
X SHL Y	—	—	—	N
X SHR Y	—	—	—	N
X EQ Y	N	—	—	N
X LT Y	N	—	—	N
X LE Y	N	—	—	N
X GT Y	N	—	—	N
X GE Y	N	—	—	N
X NE Y	N	—	—	N
X AND Y	—	—	—	N
X OR Y	—	—	—	N
X XOR Y	—	—	—	N
NOT X	—	—	N	N
+ X	A	A	N	N
– X	—	—	N	N
HIGH X	A	A	N	N
LOW X	A	A	N	N
DATAPOS X	A	A	N	N
MASK X	N	N	N	N

<说明>

ADDRESS： ADDRESS 项

NUMBER： NUMBER 项

A： 操作结果变成 ADDRESS 项

N： 操作结果变成 NUMBER 项

—： 操作不执行.

(3) 如何检查对操作的限制

这里用一个示例说明按照每一个项的重定位属性和符号属性进行的操作。

示例　BR　$TABLE+5H

这里，假定“TABLE”是在可重定位代码段内定义的标记。

<1> 操作符和重定位属性

由于“TABLE+5H”是“可重定位项 + 绝对项”，该操作适用于**表 2-10 用重定位属性组合项和操作符**。

操作符类型　X+Y

项的重定位属性　X:REL, Y:ABS

因此，结果是 R(即，可重定位项)。

<2> 操作符和重定位属性

由于“TABLE+5H”是“ADDRESS 项 + NUMBER 项”，该操作适用于**表 2-13 用符号属性组合项和操作符**。

操作符类型　X+Y

项的符号属性　X:ADDRESS, Y:NUMBER

因此，结果是 A(即，ADDRESS 项)。

2.4 比特位置标识符

可使用比特位置标识符(.)访问位。

(1) 句点(.) (比特位置标识符)

[描述格式]

X [Δ] . [[Δ] Y

位术语

X(第一项)和 Y(第二项)的组合

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

注释 关于特殊说明的更多细节，参见各设备的用户手册。

[功能]

- 比特位置标识符的第一项表示字节地址，第二项表示一个比特的位置。通过比特位置标识符可访问一个特殊位。

[说明]

- 比特项指使用比特位置标识符的表达式。
- 比特位置标识符不受操作符优先级顺序的影响。比特位置标识符的左侧被当作是第一项，而右侧则认为是第二项。
- 下列约束适于第一项：
  <1> 具有 NUMBER 或 ADDRESS 属性的表达式、可以通过 8 位访问的 SFR 名或寄存器名(A)均可被描述。
  <2> 若第一项是一个绝对表达式，则表达式的范围在 0FE20H 至 0FF1FH 之间。
  <3> 外部引用符号也可被表述。
- 下列约束适于第二项：
  <1> 表达式的值必须在 0 到 7 范围内。如果该值超出了范围，则输出错误信息。
  <2> 仅可描述具有 NUMBER 属性的绝对表达式。
  <3> 不可描述外部引用符号。

[操作和重定位属性]

- **表 2-14 依据重定位属性组合第一项和第二项**给出了依据重定位属性对第一项和第二项的组合。

表 2-14. 依据重定位属性组合第一项和第二项

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

<说明>

ABS: 绝对项
REL: 重定位项
EXT: 外部引用项

A: 操作的结果变成绝对项。
R: 操作的结果变成重定位项。
E: 操作的结果变成外部引用项。
—: 操作不执行。

[位符号值]

- 通过在 EQU 伪指令的操作数域使用比特位置标识符来描述一个位项可以定义一个位符号。**表 2-15 位符号的值**给出了位符号具有的值。

表 2-15. 位符号的值

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

注释 **1.** 详细描述参见各设备的用户手册。
2. bit = 0 to 7。
3. FFxxH 表示 sfr 的地址。
4. xxxxH 表示一个表达式的值。

[应用示例]

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

2.5 操作数的特征

指令和伪指令需要一个或几个操作数。指令类型之间的差别取决于所需操作数的值的大小和地址范围以及操作数的符号属性。

例如，指令“MOV r, #byte”的功能是将“byte”表示的值传递到寄存器“r”。此时，由于 r 是 8 位寄存器，将被传递的“byte”数据的大小必须是 8 位或更少。

如果一个指令写成“MOV R0, #100H”，由于指令的第二个操作数“100H”的大小超过 8 位寄存器 R0 的容量，则出现汇编错误。

因此，当表述一个操作符时，须注意以下几点：

- 操作数的值的大小或地址范围是否适于指令的操作符(数字数据、名称或标记)?
- 符号属性是否适于指令的操作数(名称或标记)?

2.5.1 操作数的值的大小和地址范围

在描述可作为指令的操作数时，对数字数据的值的大小和地址范围、名称或标记等方面设置了一定条件。

对于指令而言，操作数的大小和地址范围的条件由每个指令的操作数描述格式决定。对于伪指令而言，操作数的大小和地址范围的条件由指令类型决定。

表 2-16. 指令的操作数的值范围和**表 2-17. 伪指令的操作数的值范围**列出了这些条件。

表 2-16. 指令的操作数的值范围

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

表 2-17. 伪指令的操作数的值范围

伪指令类型	伪指令	值范围
段定义伪指令	CSEG AT	0H to FEFFH
	DSEG AT	0H to FEFFH
	BSEG AT	FE20H to FEFFH
	ORG	0H to FEFFH
符号定义伪指令	EQU	16-bit value 0H to FFFFH
	SET	16-bit value 0H to FFFFH
内存初始化和区域保留伪指令	DB	8-bit value 0H to FFH
	DW	16-bit value 0H to FFFFH
	DS	16-bit value 0H to FFFFH
自动分支指令选择伪指令	BR	0H to FEFFH

2.5.2 指令所需的操作数大小

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

因此，当所表述的数据超过了指令所需的操作数大小时，就会发生错误。表达式的操作采用 16 位无符号数。如果赋值超过 0FFFFH(16 位)，则输出告警信息。

但是，若在操作数中表述的是可重定位符号或外部引用符号，则值不在汇编器内确定。相反，由链接器确定这些值并检查值的范围。

2.5.3 操作数的符号属性和重定位属性

当名称、标记和**$**(表示位置计数器)作为指令操作数表述时，它们也可以不作为操作数来描述。这取决于作为表达式项的符号属性和重定位属性(见 **2.3.2 操作约束**)。如果是名称和标记，则取决于引用的方向。

名称和标记的引用方向可为后向引用和前向引用。

- 后向引用 … 被引用的名称或标记作为操作数，该操作数在名称或标记上面(前面)的代码行中定义
- 前向引用 … 被引用的名称或标记作为操作数，该操作数在名称或标记下面(后面)的代码行中定义

[示例]

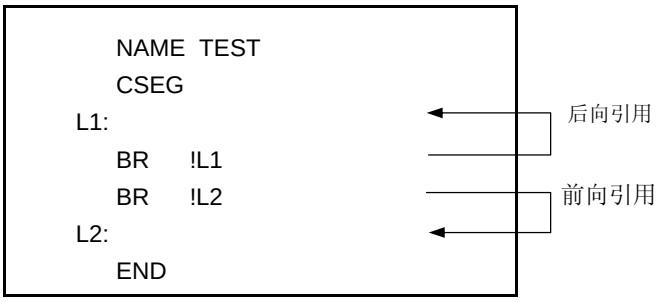


表 2-18 作为操作数被表述的符号特性和**表 2-19 作为伪指令的操作数被表述的符号特性**中给出了这些符号属性和重定位属性，以及名称和标记的引用方向。

表 2-18 . 作为操作数被表述的符号特性

符号属性	NUMBER		ADDRESS				NUMBER ADDRESS		sfr 保留字[注释1]
重定位属性	绝对项		绝对项		可重定位项		外部引用项		
引用模式 / 表示格式	后向	前向	后向	前向	后向	前向	后向	前向	
byte	O	O	O	O	O	O	O	O	×
word	O	O	O	O	O	O	O	O	×
saddr	O	O	O	O	O	O	O	O	O[注释2, 3]
saddrp	O	O	O	O	O	O	O	O	O[注释2, 4]
sfr	O[注释5]	×	×	×	×	×	×	×	O[注释2, 6]
sfrp	×	×	×	×	×	×	×	×	O[注释2, 7]
addr16[注释8]	O	O	O	O	O	O	O	O	×
addr5	O	O	O	O	O	O	O	O	×
bit	O	O	×	×	×	×	×	×	×
n	O	O	×	×	×	×	×	×	×

<说明>

前向：　表示前向引用。

后向：　表示后向引用。

O:　表示可以这样描述。

×:　表示错误

—:　表示不可以这样描述。

注释　**1.** 被定义符号指定 sfr 或 sfrp(指 sad dr 或 sfr 不会覆盖的 sfr 域)作为 EQU 伪指令的操作数，该符号仅可作为后向引用。禁止前向引用。

2. 如果一个指令的操作数组合里存在经由 saddr/saddrp 转变得到的 sfr/sfrp 组合，且 saddr 域的 sfr 保留字已为该指令所描述，则代码作为 saddr/saddrp 输出。

3. saddr 域的 sfr 保留字

4. saddr 域的 sfrp 保留字

5. 仅绝对表达式

6. 仅允许 8 位访问的 sfr 保留字

7. 仅允许 16 位访问的 sfr 保留字

8. 当禁用域(FA80H to FADFH)的地址作为 addr16 的值表示时，不执行检查。

表 2-19. 作为伪指令的操作数被描述的符号特性

符号属性		NUMBER		ADDRESS, SADDR1, SADDR2						BIT					
重定位属性		绝对项		绝对项		可重定位项		外部引用项		绝对项		可重定位项		外部引用项	
引用方向 / 伪指令		后向	前向	后向	前向	后向	前向	后向	前向	后向	前向	后向	前向	后向	前向
ORG		O[注释 1]	—	—	—	—	—	—	—	—	—	—	—	—	—
EQU[注释 2]		O	—	O	—	O[注释 3]	—	—	—	O	—	O[注释 3]	—	—	—
SET		O[注释 1]	—	—	—	—	—	—	—	—	—	—	—	—	—
DB	大小	O[注释 1]	—	—	—	—	—	—	—	—	—	—	—	—	—
	初始值									—	—	—	—	—	—
DW	大小	O[注释 1]	—	—	—	—	—	—	—	—	—	—	—	—	—
	初始值	O	O	O	O	O	O	O	O	—	—	—	—	—	—
DS		O[注释 4]	—	—	—	—	—	—	—	—	—	—	—	—	—
BR		O	—	—	—	—	—	—	—	—	—	—	—	—	—

O: 可以描述　　—: 不可以描述

注释 **1.** 仅绝对表达式可描述。

2. 如果表达式中包含下列模式之一，则出错：

- ADDRESS 属性 - ADDRESS 属性
- ADDRESS 属性相关操作符 ADDRESS 属性
- HIGH 绝对 ADDRESS 属性
- LOW 绝对 ADDRESS 属性
- DATAPOS 绝对 ADDRESS 属性
- MASK 绝对 ADDRESS 属性
- 当操作结果可被上述 6 模式的优化所影响时

3. 不允许具有可重定位项的 HIGH/LOW/DATAPOS/MASK 操作符创建一个项。

4. 参考 **3.4 (3) DS**.

[备忘录]

第三章　伪指令

本章对伪指令进行了说明。伪指令是控制 RA78K0S 执行一系列操作处理所必须的所有类型指令的指令。

3.1 伪指令概述

指令作为汇编结果被翻译成目标代码(机器语言)。但是原则上，伪指令不能转换成目标代码。伪指令主要具有如下功能：

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

表 3-1 伪指令列表列出了伪指令类型

表 3-1. 伪指令列表

序号	伪指令的类型	伪指令
1	段定义伪指令	CSEG, DSEG, BSEG, ORG
2	符号定义伪指令	EQU, SET
3	内存初始化/区域保留伪指令	DB, DW, DG, DS, DBIT
4	链接伪指令	PUBLIC, EXTRN, EXTBIT
5	对象模块名声明伪指令	NAME
6	自动分支指令选择伪指令	BR
7	宏伪指令	MACRO, LOCAL, REPT, IRP, EXITM, ENDM
8	汇编终止伪指令	END

后续章节详细解释各种伪指令。

在描述各伪指令的格式时，“[　]”表示中括号内的参数可从定义中省略，“...”表示同一格式的重复描述。

3.2 段定义伪指令

源模块必须以段为单位进行描述。

段定义伪指令用于定义以下四种类型的段：

- 代码段
- 数据段
- 位段
- 绝对段

段的类型决定了每个段在内存中的地址范围。

表 3-2 段定义方法和内存地址位置列出了定义每个段的方法以及每个段的内存地址。

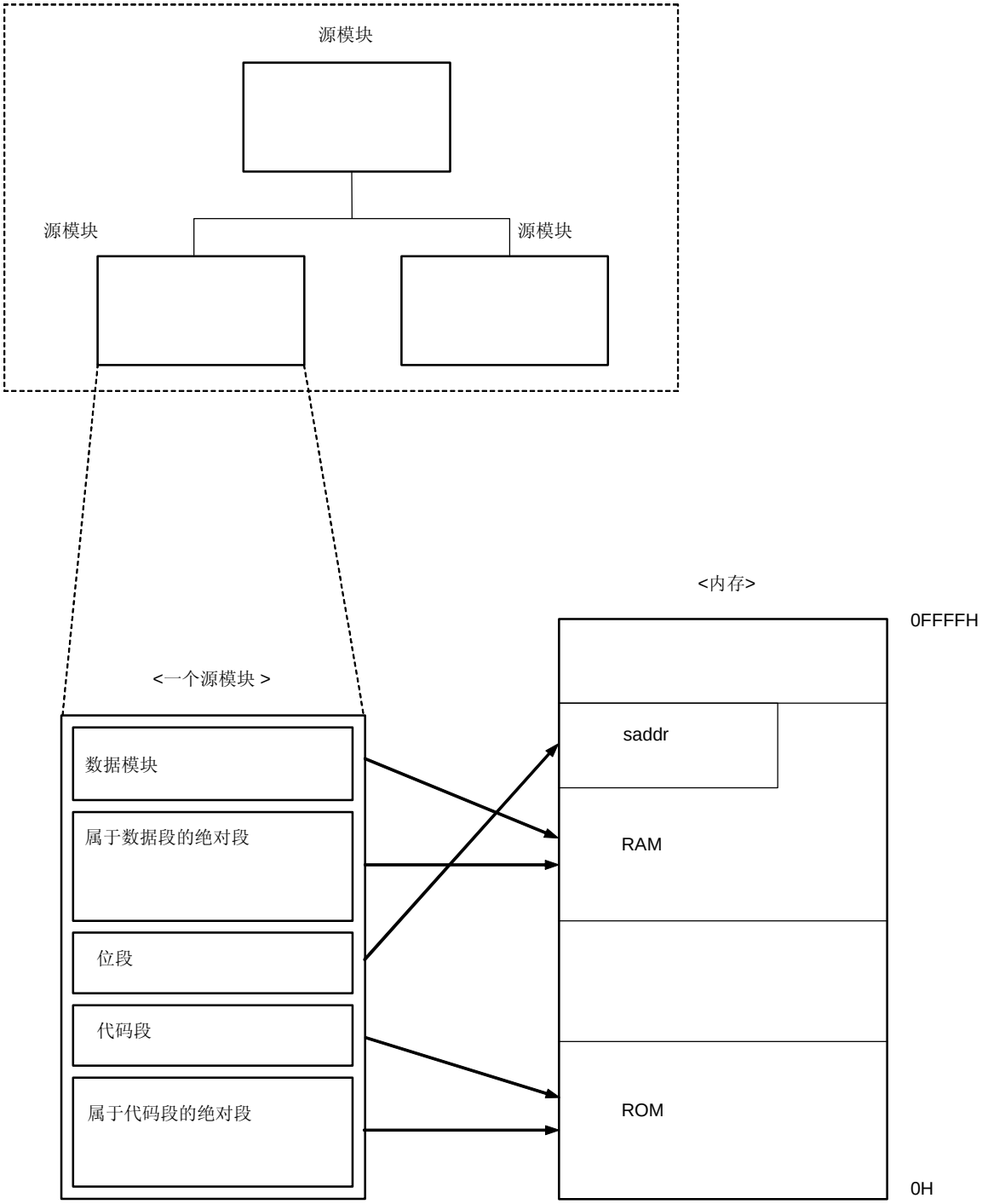
表 3-2. 段定义方法和内存地址位置

段类型	定义方法	每个段的内存地址
代码段	CSEG 伪指令	内部或外部 ROM 地址内
数据段	DSEG 伪指令	内部或外部 RAM 地址内
位段	BSEG 伪指令	内部 RAM 的 saddr 区内
绝对段	用 CSEG、DSEG 或 BSEG 伪指令为重定位属性指定位置地址(AT 位置地址)	指定地址

如果用户想要确定段的内存地址，则将段当作绝对段进行描述。对于堆栈区，用户需要在数据段保留一个区域并设置栈指针。

图 3-1 段的内存位置给出了一个段位置的例子。

图 3-1 段的内存位置


源模块
源模块
源模块
<内存>
0FFFFH
<一个源模块 >
数据模块
属于数据段的绝对段
位段
代码段
属于代码段的绝对段
saddr
RAM
ROM
0H

CSEG 代码段 CSEG

(1) CSEG (代码段)

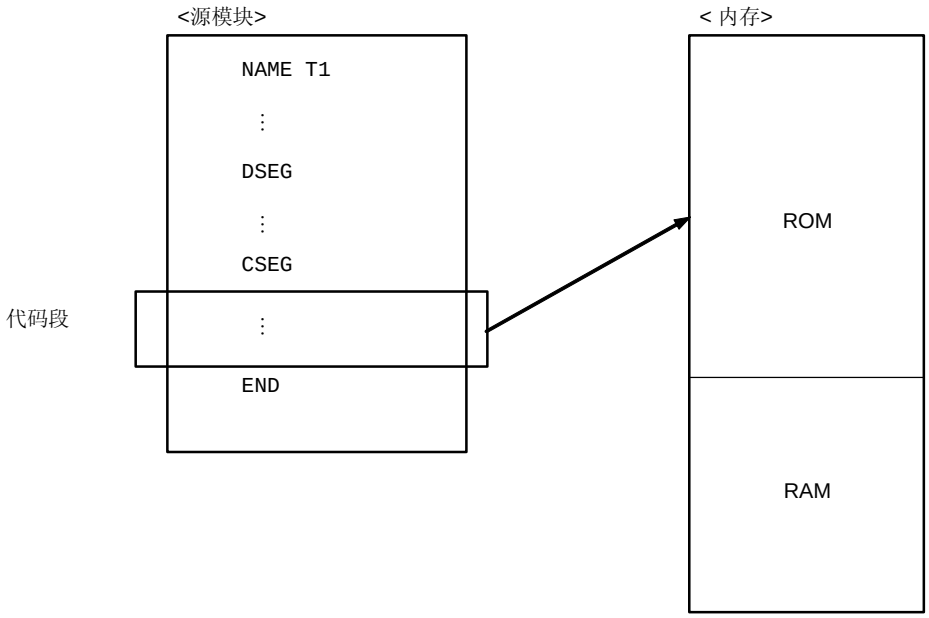
[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[段名]	CSEG	[重定位属性]	[;注释]

[功能]

- CSEG 伪指令向汇编器指出代码段的开始。
- 在遇到一个段定义伪指令(CSEG, DSEG, BSEG 或 ORG)或 END 伪指令之前，CSEG 伪指令后所描述的所有指令属于代码段。这些指令在转换成机器语言之后，最终被设置在 ROM 地址内。

图 3-2. 代码段的重定位



[用途]

- CSEG 伪指令用于描述被 CSEG 指令定义的代码段中的指令、DB、DW 伪指令等。
  (但是，为了从一个固定地址中重新定位该代码段，在操作数字段内必须把“AT 绝对表达式”作为其重定位属性。)
- 描述一个如子程序这样的功能单元时，应将其定义成一个独立代码段。如果该单元相对较大，或该子程序用户很多(例如，可用于开发其它程序)，该子程序也应定义成一个独立模块。

CSEG 代码段 CSEG

[说明]

- 代码段的起始地址可用 ORG 伪指令指定。也可通过描述重定位属性“AT 绝对表达式”来指定。
- 重定位属性为一个代码段定义了位置地址的范围。**表 3-3 CSEG 的重定位属性**示出了重定位属性。

表 3-3. CSEG 的重定位属性

重定位属性	描述格式	说明
CALLT0	CALLT0	通知汇编器定位该指定段，从而使段的起始地址在地址范围 0040H 至 007F 内变成 2 的倍数。为代码段指定重定位属性，该属性定义了被 1 字节指令 "CALLT"调用的子程序的入口地址。
FIXED	FIXED	通知汇编器在地址范围 0800H 至 0FFFH 内确定该指定段的开始。
AT	AT 绝对表达式	通知汇编器确定该指定段至一个绝对地址(0000H 至 FEFEH)
UNIT	UNIT	通知汇编器确定该指定段至任何地址(0008H 至 FA7FH)
UNITP	UNITP	通知汇编器确定该指定段至任何地址(0008H 至 FA7FH)，从而可使地址的起点是一个偶数(0008H 至 FA7FH)。
IXRAM	IXRAM	通知汇编器确定该指定段至内部扩展 RAM

- 如果没有为代码段指定重定位属性，则汇编器假定指定了“UNIT”。
- 如果指定的重定位属性超出了**表 3-3 CSEG 的重定位属性**的范围，则汇编器输出错误信息，并假定指定了”UNIT”。如果代码段的大小超过了重定位属性指定的范围，也将导致错误发生。
- 如果用重定位属性”AT” 指定的绝对表达式是非法的，汇编器将输出错误信息，并假定表达式的值为 0，继续处理操作。

通过在 CSEG 指令的符号字段描述一个段名可命名一个代码段。如果没有为代码段指定段名，汇编器自动给代码段赋予一个缺省段名。代码段的缺省段名列于**表 3-4 CSEG 的缺省段名。**

CSEG 代码段 CSEG

表 3-4. CSEG 的缺省段名

重定位属性	缺省段名
CALLT0	?CSEGT0
FIXED	?CSEGFX
UNIT (or omitted)	?CSEG
UNITP	?CSEGUP
IXRAM	?CSEGIX
AT	不能忽略段名

- 当重定位属性为 AT，却省略了段名，则产生错误。
- 如果两个或多个代码段具有同样的重定位属性(AT 除外)，这些代码段可能具有同样的段名。这些同名代码段在汇编器内作为一个单独的代码段处理。
  如果同名代码段的重定位属性不同，则产生错误。因此，每个重定位属性的同名代码段数量为 1。
- 位于两个或多个不同模块内的同名代码段在链接时被组合成一个单独的代码段。
- 段名不能作为符号被引用。
- 汇编器可输出的段的最大数量是 255 个别名段，其中包括 ORD 指令定义的段。同名段作为一个段对待。
- 段名可识别字符最多为 8 个。
- 段名字符区分大小写。

CSEG 代码段 CSEG

[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) 汇编器认为段名是"C1"，重定位属性是"UNIT"。

(2) 汇编器认为段名是"C2"，重定位属性是"CALLT0"。

(3) 汇编器认为段名是"?CSEGFX"，重定位属性是"FIXED"。

(4) 由于在(1)中，段名"C1"被认为具有重定位属性"UNIT"，则出错。

(5) 汇编器认为段名是"?CSEG"，重定位属性是"UNIT"。

DSEG 数据段 DSEG

(2) DSEG(数据段)

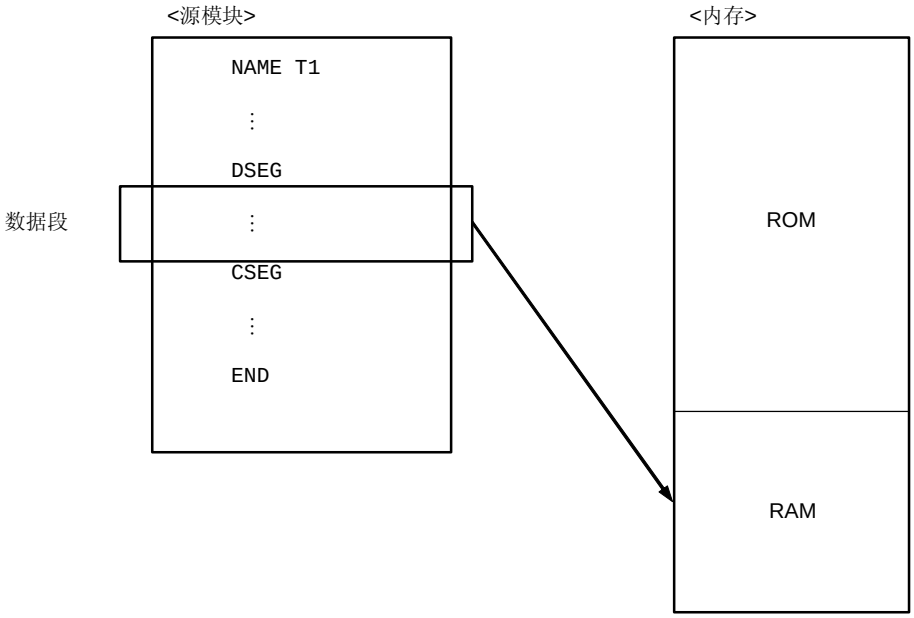
[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[段名]	DSEG	[重定位属性]	[;注释]

[功能]

- DSEG 伪指令向汇编器指出数据段的开始。
- 在遇到一个段定义伪指令(CSEG, DSEG, BSEG 或 ORG)或 END 伪指令之前，DSEG 伪指令后面由 DS 伪指令定义的内存属于数据段，且最终保留在 RAM 地址内。

图 3-3. 数据段的重定位



[用途]

- DS 伪指令主要在由 DSEG 伪指令定义的数据段中表述。数据段位于 RAM 区。因此，在数据段内不表述任何指令。
- 在数据段内，用于程序的 RAM 工作区被 DS 伪指令所保留，并对每个工作区附一个标记。在描述源程序时使用该标记。

  作为数据段被保留的区由连接器定位，因此不会与 RAM 上的其它工作区(堆栈区域、通用寄存器区域和被其它模块定义的工作区)重叠。

DSEG 数据段 DSEG

[说明]

- 可通过 ORG 伪指令指定数据段的起始地址。也可通过在 DSEG 伪指令的操作数字段中，在描述重定位属性"AT"后面紧跟一个绝对表达式来指定。
- 重定位属性为数据段定义了位置地址的范围。**表 3-5 DSEG 的重定位属性**示出了数据段可用的重定位属性。

表 3-5. DSEG 的重定位属性

重定位属性	描述格式	说明
SADDR	SADDR	通知汇编器在 saddr 区(saddr 区：0FE20H 至 0FEFFH)设置指定段
SADDRP	SADDRP	通知汇编器从 saddr 区(saddr 区：0FE20H 至 0FEFFH)的偶数地址中设置指定段
AT	AT 绝对表达式	通知汇编器在绝对地址中设置指定段
UNIT	UNIT 或没有指定	通知汇编器在任何位置(名为 RAM 的内存区)设置指定段
UNITP	UNITP	通知汇编器在偶数地址(名为 RAM 的内存区)的任何位置设置指定段
IHRAM	IHRAM	通知汇编器在高速 RAM 区设置指定段
LRAM	LRAM	通知汇编器在低速 RAM 区设置指定段
DSPRAM	DSPRAM	通知汇编器在显示 RAM 区设置指定段
IXRAM	IXRAM	通知汇编器在内部扩展 RAM 区设置指定段

- 如果数据段没有指定重定位属性，则汇编器假定已指定了"UNIT"。
- 如果指定的重定位属性超出了**表 3-4 DSEG 的重定位属性**的范围，则汇编器输出错误信息，随后假定已指定了"UNIT"。如果数据段的大小超过了重定位属性指定的范围，也将导致错误发生。
- 如果重定位属性"AT"指定的绝对表达式是非法的，汇编器将输出错误信息，并假定表达式的值为 0，继续处理操作。
- 通过在 DSEG 指令的符号字段描述一个段名来命名一个数据段。
  如果没有为数据段指定段名，汇编器自动给数据段赋予一个缺省段名。数据段的缺省段名列于**表 3-6 DSEG 的缺省段名。**

DSEG 数据段 DSEG

表 3-6. DSEG 的缺省段名

重定位属性	缺省段名
SADDR	?DSEGS
SADDRP	?DSEGSP
UNIT (or no specification)	?DSEG
UNITP	?DSEGUP
IHRAM	?DSEGIH
LRAM	?DSEGL
DSPRAM	?DSEGDSP
IXRAM	?DSEGIX
AT	不可忽略段名

- 如果两个或多个数据段具有同样的重定位属性(AT 除外)，这些数据段可以具有同样的段名。这些同名段在汇编器内作为一个单独的数据段处理。
- 如果重定位属性是 SADDRP，则定位该指定段，使紧跟在 DSEG 伪指令之后的地址变成 2 的倍数。
- 如果同名数据段的重定位属性不同，则产生错误。因此，每个重定位属性的同名数据段数量为 1。
- 位于两个或多个不同模块内的同名数据段在连接时被组合成一个单独的数据段。
- 段名不能作为符号被引用。
- 汇编器可输出的段的最大数量是 255 个别名段，其中包括 ORD 指令定义的段。同名段作为一个对待。
- 段名可识别字符最多为 8 个。
- 段名区分大小写字符。

DSEG 数据段 DSEG

[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) 数据段的起始由 DSEG 伪指令定义。由于重定位属性被省略，则假定重定义属性为"UNIT"，缺省段名为"?DSEG"。

(2) 该描述相当于"MOV A, !addr16"。

(3) 该描述相当于"MOV A, saddr"。可重定位标记 "WORK1" 不能写成 "saddr". 因此，采用这种描述的结果是产生错误。

(4) 该描述相当于"MOVW rp, #word"。

(5) 该描述相当于"MOVW AX, saddrp"。
可重定位标记 "WORK2" 不能表示成 "saddrp"。 因此，该描述的结果将产生错误。

BSEG 位段 BSEG

(3) BSEG（位段）

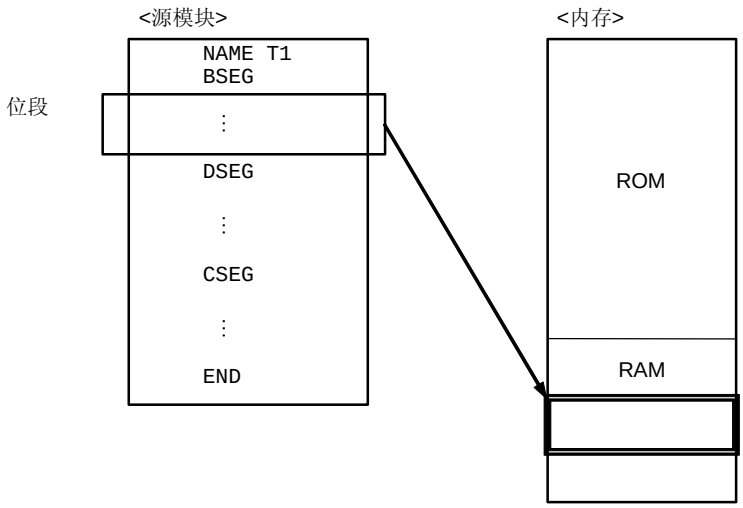
[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[段名]	BSEG	[重定位属性]	[;注释]

[功能]

- BSEG 伪指令向汇编器指出位段的开始。
- 位段是定义了在源模块中使用的 RAM 地址的段。
- 在遇到一个段定义伪指令(CSEG, DSEG, BSEG 或 ORG)或 END 伪指令之前，在 BSEG 伪指令之后由 DBIT 伪指令定义的内存属于位段。

图 3-4. 位段的重定位



[用途]

- 在由 BSEG 伪指令定义的位段中描述 DBIT 伪指令(参见**应用示例**)。
- 任何位段都不可以描述指令

BSEG 位段 BSEG

[说明]

- 位段的起始地址可通过在重定位属性字段表述"AT 绝对表达式"来指定。
- 重定位属性定义了位段的位置地址范围。**表 3-7 BSEG 的重定位属性**示出了位段可用的重定位属性。

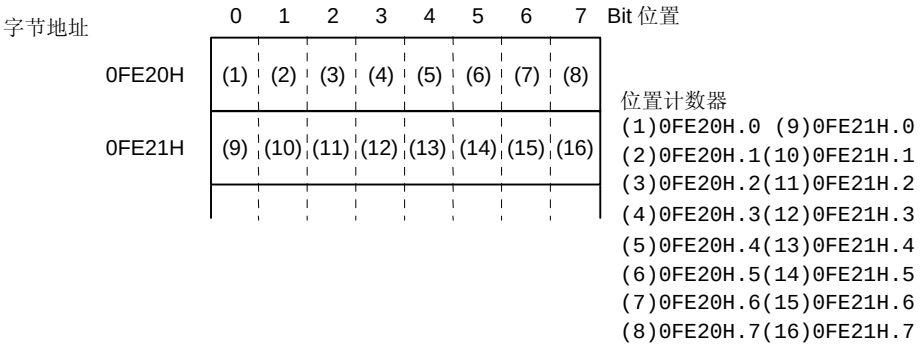
表 3-7 BSEG 的重定位属性

重定位属性	描述格式	说明
AT	AT 绝对表达式	通知汇编器把指定段的起始地址定位于绝对地址的第 0 位。禁止在位单元(FE20H to FEFFH)指定。
UNIT	UNIT (或没有指定)	通知汇编器在任何位置(FE20H 至 FEFFH)定位该指定段

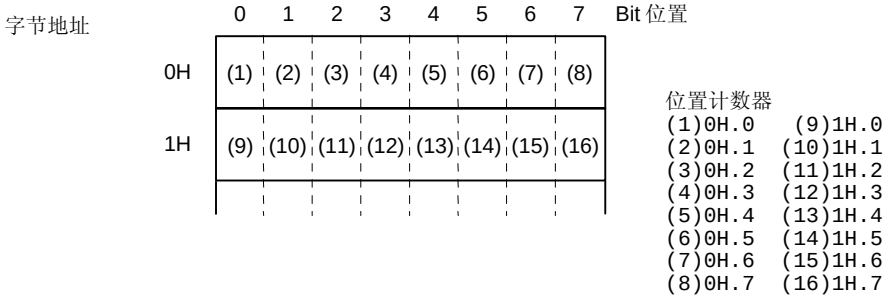
- 如果没有为位段指定重定位属性，则汇编器假定已指定了"UNIT"。
- 如果指定的重定位属性超出了**表 3-7 BSEG 的重定位属性**的范围，则汇编器输出错误信息，随后假定指定了"UNIT"。如果每个位段的大小超过了重定位属性指定的范围，也将导致错误发生。
- 在汇编器和连接器中，位段的位置计数器显示格式为"0xxxx.b"(字节地址是十六进制表示的 4 位数字，比特位置是十六进制表示的 1 位数字(0 到 7))。

BSEG 位段 BSEG

使用绝对位段



使用可重定位位段



备注 在可重定位位段内，字节地址指定了从段起始开始的字节偏移量。

在目标转化器输出的符号表中，显示并输出从位定义区域开始的 Bit 偏移量。

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

BSEG 位段 BSEG

- 如果重定位属性"AT"指定的绝对表达式是非法的，汇编器将输出错误信息，并假定表达式的值为 0，继续处理操作。
- 通过在 BSEG 指令的符号字段描述一个段名来命名一个位段。
  如果没有为位段指定段名，汇编器自动给位段赋予一个缺省段名。下表列出了位段的缺省段名。

表 3-8. BSEG 的缺省段名

重定位属性	缺省段名
UNIT (或没有指定)	?BSEG
AT	段名不可忽略。

- 如果重定位属性是"UNIT"，则两个或多个数据段可具有同样的段名(AT 除外)。这些段在汇编器内作为一个单独的段处理。因此，每个重定位属性的同名段的数量为 1。
- 在两个或多个不同模块内的同名位段在连接时将组合成一个单独的位段。
- 段名不能作为符号被引用。
- 仅可在位段中描述的指令有 DBIT、 EQU、SET、PUBLIC、EXTBIT、EXTRN、MACRO、REPT、IRP、ENDM 伪指令、宏定义和宏引用。超出这些指令之外的描述均将引起错误。
- 汇编器可输出的段的最大数量是 255 个别名段，其中包括 ORD 指令定义的段。同名段作为一个计列。
- 段名可识别字符最多为 8 个。
- 段名区分大小写字符。

BSEG 位段 BSEG

[应用示例]

```
NAME   SAMP1

FLAG   EQU  0FE20H
FLAG0  EQU  FLAG.0    ;(1)
FLAG1  EQU  FLAG.1    ;(1)

BSEG                  ;(2)
FLAG2  DBIT

CSEG
SET1   FLAG0          ;(3)
SET1   FLAG2          ;(4)

END
```

<说明>

(1) 位地址(0FE20H 中的第 0 位和第 1 位)的定义遵循和字节地址边界同样的注意事项。

(2) 由 BSEG 伪指令来定义位段。
由于忽略了重定位属性，则假定重定位属性为"UNIT"，段名为"?BSEG"。每个位段中，用 DBIT 伪指令为每个位定义位工作区。位段应在模块体的前部分描述。定位在位段内定义的位地址 FLAG2 不需考虑字节地址边界。

(3) 该句描述可由 "SET1 FLAG.0"代替。FLAG 表示字节地址。

(4) 该描述中，没有考虑字节地址边界。

(4) ORG（起源）

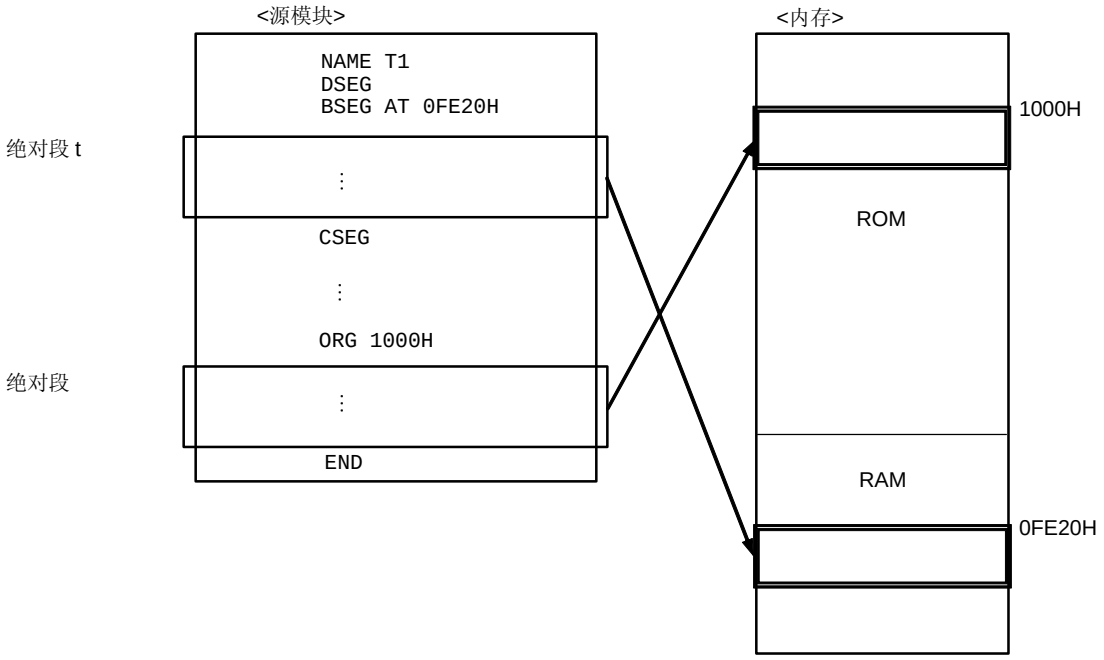
[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[段名]	ORG	[重定位属性]	[;注释]

[功能]

- ORG 伪指令设置的表达式的值由位置计数器的操作数指定。.
- 在遇到段定义伪指令(CSEG、DSEG、BSEG 或 ORG)或 END 伪指令之前，ORG 伪指令之后所描述的指令或保留内存区都属于一个绝对段，而且这些绝对段位于操作数指定的地址之后。

图 3-5. 绝对段的位置



[用途]

- 指定 ORG 伪指令从一个指定地址定位一个代码段或数据段。

ORG 起源 ORG

[说明]

- ORG 伪指令定义的绝对段属于紧接在 ORG 指令之前用 CSEG 或 DSEG 伪指令定义的代码段或数据段。
  在属于数据段的绝对段内，不可描述任何指令。
  属于位段的绝对段不可用 ORG 伪指令描述。
- 用 ORG 伪指令定义的代码段或数据段被认为是重定位属性”AT”的代码段或数据段。
- 在 ORG 伪指令的符号字段描述段名可命名一个绝对段。作为段名可被 识别的字符最多是 8 个。
- 如果没有给绝对段指定段名，汇编器将自动分配一个缺省段名”?A00xxxx”，其中”xxxx”表示指定段的 4 位十六进制起始地址(0000 to FEFF)。
- 如果在 ORG 伪指令之前 CSEG 和 DSEG 伪指令都没有描述，由 ORG 伪指令定义的绝对段则解释为代码段中的绝对段。
- 如果名称或标记作为在 ORG 伪指令的操作数，则该名称或标记必须是已经在源模块中定义的绝对项。
- 段名不可以作为符号被引用。
- 汇编器可输出的段的总量为 255 个别名段，包括段定义伪指令所定义的段。同名段按 1 个计列。
- 段名可识别字符最多为 8 个。

[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) 定义了一个属于数据段的绝对段。该绝对段将位于从地址”FE20H”起始的短直接寻址区。
由于省略了段名的定义，汇编器自动分配段名”?A00FE20”。

(2) 由于在属于代码段的绝对段内没有描述指令，则出错。

(3) 该伪指令声明代码段的开始。

(4) 该绝对段位于从地址”1000H”起始的区。

3.3 符号定义伪指令

符号定义伪指令给用于描述源模块所需的数字数据分配名称。这些名称阐明了每个数据值的含义，使源模块的内容更易理解。

符号定义伪指令告诉汇编器将在源模块中使用的各名字的值。

EGU 和 SET 这两个指令可用于符号定义。

EQU 等于 EQU

(1) EQU（等于）

[编写格式]

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

[功能]

- EQU 伪指令定义一个名称，该名称具有在操作数字段定义的表达式的值和属性(符号属性和重定位属性)。

[用途]

- 采用 EQU 伪指令，把将要在源模块中使用的数字数据定义为名称，且在指令的操作数中描述数字数据时用该名称代替。
  建议将把源模块中频繁使用的数字数据定义为一个名称。如果需要改变源模块中的数据值，则只需改变名称的操作数的值(参见**应用示例**)。

[说明]

- 当一个名称或标记将在 EQU 伪指令的操作数中描述时，使用已在源模块中定义的名称或标记。
  外部引用项不可以作为该伪指令的操作数。
- 如果一个表达式包含一个由在操作数中具有可重定位项的 HIGH/LOW/DATAPOS/BITPOS 操作符创建的项，则不能描述该表达式。
- 如果所描述的表达式具有下列任何操作符模式，则将产生错误：
  (a) 具有 ADDRESS 属性的表达式 1 - 具有 ADDRESS 属性的表达式 2
  (b) 具有 ADDRESS 属性的表达式 1 关系操作符 具有 ADDRESS 属性的表达式 2
  (c) 下列条件<1>和 <2>之一在上述表达式(a)或(b)中实现：
  <1> 如果标记 1 在具有 ADDRESS 属性的表达式 1 中，标记 2 在具有 ADDRESS 属性的表达式 2 中，标记 1 和标记 2 属于同一段，而且如果两个标记之间描述的 BR 伪指令不能确定目标代码的字节数
  <2> 如果标记 1 和标记 2 位于不同的段，且如果在段的开始部分和标记之间所表述的 BR 伪指令不能确定目标代码的字节数
  (d) 具有 ADDRESS 属性的 HIGH 绝对表达式
  (e) 具有 ADDRESS 属性的 LOW 绝对表达式
  (f) 具有 ADDRESS 属性的 DATAPOS 绝对表达式
  (g) 具有 ADDRESS 属性的 BITPOS 绝对表达式
  (h) 下面条件<3>在表达式(d), (e), (f)或 (g)中实现：
  <3> 如果一个 BR 指令在带有 ADDRESS 属性的标记和标记所属的段的段开头之间被描述，而此 BR 指令不能确定目标代码的字节数 。

EQU 等于 EQU

- 如果操作符的书写格式中有错，汇编器将输出一个错误信息，但会尝试存储操作数的值作为在符号字段描述的名称的值，以便可以分析。
- 用 EQU 伪指令定义的名称不能在同一源模块内重复定义。
- 若一个名称已用 EQU 伪指令定义了一个比特值，该名称将拥有一个地址，并把比特位置作为它的值。
- **表 3-9 代表了比特值的操作符的表示格式**显示了可作为 EQU 伪指令的操作数描述的比特值，以及这些比特值可引用的范围。

表 3-9 代表了比特值的操作符的表示格式

操作数类型	符号值	引用范围
A.bit 注解 1	1.bit	只可在同一模块内被引用
PSW.bit 注解 1	1FEH.bit	
Sfr 注解 2.bit 注解 1	0FFxxH 注解 3.bit	
saddr.bit 注解 1	0xxxxH 注解 4.bit	可从另一模块被引用
expression.bit 注解 1	0xxxxH 注解 4.bit	

注解 **1**. bit = 0 to 7

2. 更详细描述，参见各设备的用户手册

3. “0FFxxH”表示 sfr 的地址。

4. “0xxxxH”表示 saddr 区(FE20H to FF1FH)。

EQU 等于 EQU

[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”的值为”0FE20H”，符号属性为”NUMBER”，重定位属性为”ABSOLUTE”。

(2) 名称”WORK10”的比特值是”WORK1.0”，其操作数格式为”saddr.bit”。名称”WORK1”在操作数中被表述，已在(1)中定义了值”0FE20H”。

(3) 名称”P02”被赋比特值”P0.2”，其操作符格式为”sfr.bit”。

(4) 名称”A4”被赋比特值”A.4”，其操作符格式为”A. bit”。

(5) 名称”PSW5”被赋值”PSW.5”，其操作符格式为”PSW.bit”.

(6) 该语句描述相当于”SET1 saddr.bit”。

(7) 该语句描述相当于”SET1 sfr.bit”。

(8) 该语句描述相当于”SET1 A.bit”。

(9) 该语句描述相当于”SET1 PSW.bit”。

在(3)到(5)中已定义”sfr.bit”, “A.bit”和 “PSW.bit”的名称仅可在同一模块内被引用。
已定义”saddr.bit”的名称也可从另一模块作为外部定义符号被引用(参见 **3.5 (2) EXTBIT**)。
产生下列汇编列表作为例子中源模块的汇编结果。

EQU 等于 EQU

```
        Assemble list

ALNO  STNO ADRS   OBJECT      M I SOURCE   STATEMENT

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

<说明>

对于汇编列表的第 2 行到第 5 行，在目标代码字段指出作为名称定义的比特值的位地址值。

SET 设置 SET

(2) SET（设置）

[编写格式]

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

[功能]

- SET 伪指令定义的名称具有在操作数字段指定的表达式的值和属性(符号属性和重定位属性)。
- 用 SET 伪指令定义的名称的值和属性在同一模块内可被重新定义。直道同一名称被重新定义之前这些值和属性都有效。

[用途]

- 把将在源模块中使用的数字数据(变量)定义为一个名称，且在指令操作数描述中用该名称代替数字数据(变量)。如果需要改变源模块的名称的值，可再次使用 SET 伪指令为同一名称定义一个不同的值。

[说明]

- 必须在 SET 伪指令的操作符字段描述一个绝对表达式。
- SET 伪指令可在源程序的任何地方被描述。但是，已经被 SET 伪指令定义的名称不能被前向引用。
- 如果在一条语句中检查出的错误是因为该语句中的名称用 SET 伪指令定义，则汇编器输出错误信息，但将尝试存储该操作数的值作为在符号字段中所描述的名称的值，以便进行分析。
- 用 EQU 伪指令定义的符号不能用 SET 伪指令重定义。
  用 SET 伪指令定义的符号不能用 EQU 伪指令重定义。
- 不能定义位符号。

SET set SET

[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” 的值为”10H”，符号属性为 “NUMBER”，重定位属性为 “ABSOLUTE”。在值和属性被 SET 伪指令(说明(3)中)重新定义之前，这些值和属性都有效。

(2) 名称”COUNT”的值”10H”被传递给寄存器 B 。

(3) 名称”COUNT” 的值变成”20H”。

(4) 名称”COUNT”的值” 2 0H”被转移给寄存器 B 。

3.4 内存初始化和区域保留伪指令

内存初始化伪指令定义了将要在源程序中使用的常量数据。
被定义常量数据的值作为目标代码产生。
区域保留伪指令保存将要在程序中使用的内存区域。

DB 定义字节 DB

(1) DB （定义字节）

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[标记:]	DB	{(size) 初始值[,...]}	[;注释]

[功能]

- DB 伪指令通知汇编器初始化一个字节区域。将被初始化的字节的大小定义为“size”。
- DB 伪指令还通知汇编器初始化一块以字节为单位的内存区，初始值由操作数字段的初始值指定。

[用途]

- 使用 DB 伪指令定义一个在程序中使用的表达式或字符串。

[说明]

- 如果操作数字段的值加上了括弧，则汇编器认为指定了初始化值的大小。否则，假定一个初始值。
- DB 伪指令不能在位段中描述。

定义了大小时：

- 若在操作数字段指定了初始值大小，则汇编器初始化一个与指定字节数等值的区域，初始值为“00H”.
- 绝对表达式必须被描述成一个数量大小。如果大小描述是非法的，则汇编器输出一个错误信息，且停止执行初始化。

定义了初始值时：

- 下面两个参数可被指定为初始值：

  <1> 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 范围内。如果值超过 8 位，汇编器将只使用该值的低 8 位作为有效数据，并输出一个错误信息。

  <2> 字符串

  如果字符串描述为操作数，则将为串中的每个字符保留一个 8 位的 ASCII 码。
- 在 DB 伪指令的一个语句行内，可指定两个或多个初始值。
- 包含一个可重定位符号或外部引用符号的表达式可描述为一个初始值。

[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) 由于指定了数量大小，汇编器将初始化从值“00H”开始的每个字节区域。

(2) 初始化一个 6 字节区，初始化值是字符串 ‘ABCDEF’。

(3) 初始化一个 3 字节区，初始化值是“0AH, 0BH, 0CH”。

(4) 初始化一个 4 字节区，初始化值是“00H”。

(5) 由于表达式‘AB’ +1 的值为 4143H (4142H+1)，超出了 0H 至 0FFH 的范围，该语句描述将产生错误。

DW　　定义字　　DW

(2) DW （定义字）

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[标记:]	DW	{(size) 初始值[,...]}	[;注释]

[格式]

- DW 伪指令通知汇编器初始化一个字区域。将要被初始化的字的大小可被指定为“size”。
- DW 伪指令还通知汇编器初始化一块以字（2 字节）为单位的内存区，初始值由操作数字段的初始值指定。

[用途]

- 使用 DW 伪指令定义一个在程序中使用的 16 位数字常量如地址或数据。

[说明]

- 如果操作数字段的值加上了括弧，则汇编器认为指定了初始化值的大小。否则，假定一个初始值。
- DW 伪指令不能在位段中描述。

指定了大小时：

- 若在操作数字段指定了初始值大小，则汇编器初始化一个与指定字节数等值的区域，初始值为“00H”。
- 绝对表达式必须被描述成一个大小。如果初始化值的大小描述是非法的，则汇编器输出一个错误信息，且停止执行初始化。

指定了初始值时：

- 下列两个参数可指定为初始值：
  <1> 常量
  16 位或更少
  <2> 表达式
  表达式的值必须存储为 16 位数据。
  字符串不能被描述为一个初始值。
- 被指定初始值的高 2 位数字存储于高地址里，低 2 位数字存储于低地址里。
- 在 DW 伪指令的一个语句行内，可指定两个或多个初始值。
- 包含可重定位符号或外部引用符号的表达式可表述为一个初始值。

DW　　定义字　　DW

[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>

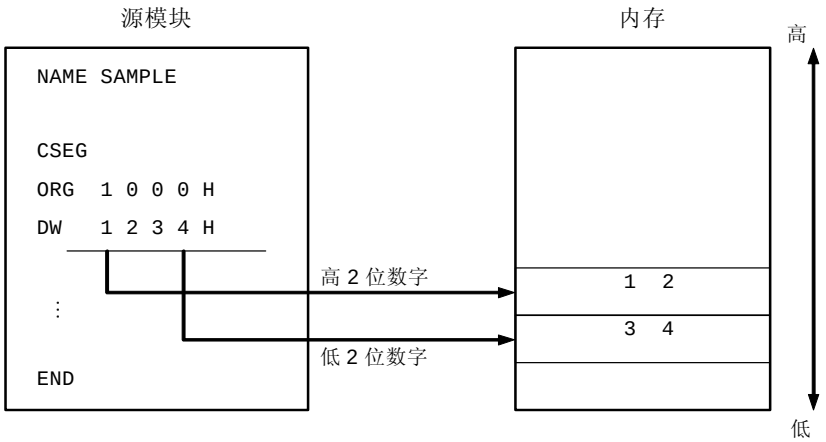
(1) 由于指定了初始化大小，汇编器将初始化每个字，初始值为 “00H”。

(2) 用 DW 伪指令定义矢量入口地址。

(3) 初始化一个 2 字区域，初始值为“34127856”。

警告　用字的值的高 2 位数字初始化内存的高地址。用字的值的低 2 位数字初始化内存的低地址。

示例：



DS 定义存储空间 DS

(3) DS （定义存储空间）

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[标记:]	DS	绝对表达式}	[;注释]

[功能]

- DS 伪指令通知汇编器为操作数字段中指定的字节数保留一个内存区。

[用途]

- DS 伪指令主要用于保留一个内存区（RAM）供程序使用。如果指定了一个标记，被保留内存区域的第一个地址的值赋给该标记。在源模块中，该标记用于描述使用该内存。

[说明]

- 将由 DS 伪指令保留的区域的值是未知的（不确定）。
- 被指定的绝对表达式的值用无符号 16 位表示。
- 若操作数值是“0”，则不保留区域。
- DS 伪指令不可在位段内描述。
- DS 伪指令定义的符号（标记）仅可后向引用。
- 只有下列从绝对表达式中扩展得到的参数可以在操作数字段描述：
  - <1> 常量
  - <2> 将对其执行操作的带常量表达式（常量表达式）
  - <3> 带常量或常量表达式定义的 EQU 符号或 SET 符号
  - <4> 带 ADDRESS 属性的表达式 1 -带 ADDRESS 属性的表达式 2
    如果在“带 ADDRESS 属性的表达式 1’中的标记 1 和在“带 ADDRESS 属性的表达式 2’中的标记 2 都是可重定位的，两标记必须在同一段内定义。
    但是，下列两种情况之一将导致错误：
    - (a) 如果标记 1 和标记 2 属于同一段，而且，在两标记之间描述的 BR 伪指令不能确定目标代码的字节数
    - (b) 如果标记 1 和标记 2 位于不同的段，而且，在任一标记和标记所属的段的起始位置之间所描述的 BR 伪指令不能确定目标代码的字节数
  - <5> 将在上述表达式<1> 至<4>的任何一个之上执行操作
- 下列参数不可在操作数字段描述：
  - <1> 外部引用符号
  - <2> 已经用 EQU 伪指令定义了“带 ADDRESS 属性的表达式 1 - 带 ADDRESS 属性的表达式 2”的符号
  - <3> 位置计数器($)在表达式 1 或表达式 2 中以“带 ADDRESS 属性的表达式 1 -带 ADDRESS 属性的表达式 2”的形式描述
  - <4> 用 EQU 伪指令定义的表达式具有 ADDRESS 属性的符号，且 HIGH/LOW/DATAPOS/BITPOS 操作符将对其进行操作。

DS　　定义存储空间　　DS

[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，但是区域的值未定（不确定）。标记“TABLE1”位于地址的起始位置。

(2)　保留一个 1 字节工作区。.

(3)　保留一个 2 字节工作区。

DBIT 定义位 DBIT

(4) DBIT （定义位）

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[名称]	DBIT	无	[;注释]

[功能]

- DBIT 伪指令通知汇编器在位段内保留一个 1 比特内存区域。

[用途]

- 使用 DBIT 伪指令在位段内保留一个 1 比特内存区。

[说明]

- DBIT 伪指令仅在位段内描述。
- 用 DBIT 伪指令保留的 1 位区域的值未知（不确定）。
- 如果在符号字段指定了名称，该名称拥有一个地址，且比特位置作为它的值。
- 被定义名称可在需要 saddr.bit 的地方进行描述。

[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) 通过三个 DBIT 伪指令, 汇编器将保留三个 1 比特区域，并定义名称(BIT1, BIT2, and BIT3)，每个名称具有一个地址和一个比特位置作为它的值。

(2) 该语句描述相当于 “SET1 saddr.bit”，并且将保留在上述(1)中位区域的名称 “BIT1”作为操作数 “saddr.bit”。

(3) 该语句描述相当于“CLR1 saddr.bit”，定义名称 “BIT2”为 “saddr.bit”。

3.5 链接伪指令

链接伪指令阐明了引用其他模块内所定义的符号的相关性。

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

在模块 1 中，符号的外部引用声明表示必须引用在另一模块内定义的符号。在模块 2 中，符号的外部定义声明表示所定义符号可能在另一模块内被引用。

符号被首次引用时外部引用声明和外部定义声明均正确定义。

链接伪指令的功能是建立这种解释关系，且在下列两种情况下可用：

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

图 3-6. 两模块之间的符号的关系

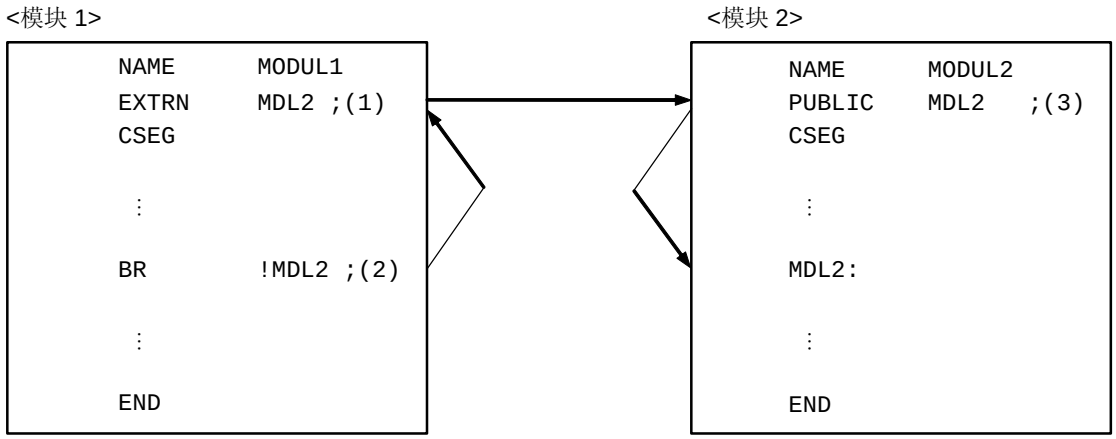


图 3-6 中，模块 2 中定义的符号 “MDL2”在模块 1 的（2）中被引用。因此，在（1）中，符号作为外部引用由 EXTRN 伪指令声明。

在模块 2 的（3）中，将被模块 1 引用的符号“MDL2”被 PUBLIC 伪指令声明为一个外部定义。

链接器检查符号的外部引用是否与符号的外部定义相一致。

EXTRN　　外部的　　EXTRN

(1) EXTRN（外部）

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[标记:]	EXTRN	符号名[,...]	[;注释]

[功能]

- EXTRN 伪指令向连接器声明，另一模块中的符号（除了位符号）将在该模块中被引用。

[用途]

- 当引用了一个在另一模块中定义的符号时，必须使用 EXTRN 伪指令将该符号声明为一个外部引用。

[说明]

- EXTRN 伪指令可在源程序的任何位置描述（参见 **2.1 源程序的基本配置**）。
- 操作数字段最多可指定 20 个符号，每个符号名之间用逗号（,）分开。
- 若引用的符号具有比特值，该符号必须用 EXTBIT 伪指令声明为外部引用。
- 用 EXTRN 伪指令声明的符号必须在另一模块内用 PUBLIC 伪指令声明。
- 宏名称不能描述成 EXTRN 伪指令的操作数（宏名称可参见**第五章 宏**）。
- 对于一个完整模块，EXTRN 伪指令仅保证符号的一个 EXTRN 声明。对该符号的第二个以及后续的 EXTRN 声明，连接器将输出告警信息。
- 已经被声明的符号不能再作为 EXTRN 伪指令的操作数。相反，已经被 EXTRN 声明的符号不能被重新定义或由其他伪指令声明。
- EXTRN 伪指令定义的符号可用于引用一个 saddr 区。

EXTRN 外部的 EXTRN

[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 伪指令声明将在(2) 和(3)中被引用的符号“SYM1” 和“SYM2”为外部引用。在操作数字段可描述两个或多个符号。

(2) DW 伪指令引用符号 “SYM1”。

(3) MOV 伪指令引用符号 “SYM2”，输出引用了 saddr2 区的代码。

(4) 符号“SYM1” 和“SYM2” 声明为外部定义。.

(5) 定义符号“SYM1”。

(6) 定义符号“SYM2”

EXTBIT 外部位 EXTBIT

(2) EXTBIT （外部位）

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[标记:]	EXTBIT	位符号名[,...]	[;注释]

[功能]

- EXTBIT 伪指令向连接器声明，该模块将引用已在另一模块中定义且具有 saddr.bit 值的符号。

[用途]

- 当引用的符号具有一个比特值且在已另一模块中被定义时，必须使用 EXTBIT 伪指令将该符号声明为一个外部引用。

[说明]

- EXTBIT 伪指令可在源程序的任何地方被描述。
- 在操作数字段最多可指定 20 个符号，各符号间用（,）分开。
- 由 EXTBIT 伪指令声明的符号必须在另一模块内用 PUBLIC 伪指令声明。
- 在一个完整模块，EXTBIT 伪指令仅保证符号的一个 EXTBIT 声明。对该符号的第二和后续 EXTBIT 声明，连接器将输出告警信息。

EXTBIT 外部位 EXTBIT

[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 EXTRN 伪指令声明符号“FLAG1” 和“FLAG2” 将作为外部引用符号。在操作数字段可描述两个或多个符号。

(2) SET1 伪指令引用符号 “FLAG1”。该语句描述相当于“SET1 saddr.bit”。

(3) CLR1 伪指令引用符号 “FLAG2”， 该语句描述相当于“CLR1 saddr.bit”。

(4) PUBLIC 伪指令定义符号“FLAG1” 和 “FLAG2”。.

(5) DBIT 伪指令定义符号“FLAG1”作为 SADDR 区的位符号。

(6) DBIT 伪指令定义符号“FLAG2”作为 SADDR 区的位符号。

PUBLIC　　全局的　　PUBLIC

(3) PUBLIC （全局的）

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[标记:]	PUBLIC	符号名[,...]	[;注释]

[功能]

- PUBLIC 伪指令向连接器声明，在操作数字段描述的符号将被另一模块引用。

[用途]

- 当定义的符号（包括位符号）将被另一模块引用时，必须使用 PUBLIC 伪指令声明该符号为外部定义。

[说明]

- 可在源程序的任何地方声明 PUBLIC 伪指令。
- 在操作数字段最多可指定 20 个符号，各符号间用（,）分开。
- 将要在操作数字段描述的符号必须在同一模块内定义。
- 在一个完整模块内，EXTBIT 伪指令仅保证符号的一个 EXTBIT 声明。对该符号的第二和后续 EXTBIT 声明，连接器将输出告警信息。
- 下列符号不可用作 PUBLIC 伪指令的操作数：
  - 用 SET 伪指令定义的名称
  - 在同一模块内用 EXTRN 或 EXTBIT 伪指令定义的符号
  - 段名
  - 模块名
  - 宏名
  - 不在该模块内定义的符号
  - 由 EQU 伪指令定义的其操作数具有位属性的符号
  - 由 EQU 伪指令定义的 sfr 的符号（但是，sfr 区和 saddr 区重叠的区域除外）

PUBLIC 全局的 PUBLIC

[应用示例]

示例程序包含三个模块

<源模块 1>

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

A1      EQU     10H
A2      EQU     0FE20H.1

        CSEG
        BR      B1
        SET1    C1
        END
```

<源模块 2>

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

<源模块 3>

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

<说明>

(1) PUBLIC 伪指令声明符号“A1”和“A2”将被其他模块引用。

(2) PUBLIC 伪指令声明符号“B1”将被其他模块引用。

(3) PUBLIC 伪指令声明符号“C1”将被其他模块引用。

3.6　目标模块名声明伪指令

目标模块名声明伪指令给将被 RA78K0S 汇编器创建的目标模块赋予一个模块名称。

NAME　　名称　　NAME

(1) NAME （名称）

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[标记:]	NAME	目标模块名	[;注释]

[功能]

- NAME 伪指令把在操作数字段声明的目标模块名赋给由汇编器输出的目标模块。

[用途]

- 在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]

- 可在源程序的任何地方声明 NAME 伪指令。
- 对于模块名描述的约定，参见 **2.2.3 组成语句的字段**中的符号描述惯例。
- 可以做为模块名的字符都是汇编器软件操作系统所允许的字符，但“ (”,“ (28H)”, “)”或 “ (29H)”除外。
- 除了 NAME 伪指令，模块名不能作为其他伪指令或指令的操作数。
- 如果省略了 NAME 伪指令，汇编器假定输入源模块文件的原名称（前 8 个字符）作为模块名。 在 Windows 版本中，原名称被转换成大写字母以便于检索。如果指定了两个或多个模块名，汇编器将输出告警信息，并忽略第二个和后续的模块名声明。
- 在操作数字段被描述的模块名不得超过 8 个字符。
- 符号名字符区分大小写。

[应用示例]

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

        CSEG
        MOV     A,B

        END
```

<说明>

(1) NAME 伪指令声明“SAMPLE”为模块名。

3.7 自动分支指令选择伪指令

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

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

BR 分支 BR

(1) BR (分支)

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[标记:]	BR	表达式	[;注释]

[功能]

- BR 伪指令通知汇编器根据在操作数字段定义的表达式值的范围，自动选择 2 字节或 3 字节 BR 分支指令，并产生应用于被选择指令目标代码。

[用途]

- 如果分支目的点位于从 BR 伪指令后-80H 到+70H 的地址范围内，则可以描述 2 字节分支伪指令“BR $addr16”。使用该指令与使用 3 字节分支指令“BR !addr16”相比，所需内存空间可减少一个字节。为了创建具有高内存利用率的程序，应正确使用 2 字节分支指令。
  但是，在描述分支指令时考虑分支目的点的地址范围是件很麻烦的事。因此，如果不能确定是否可描述 2 字节分支指令时，应使用 BR 伪指令。
- 当能够确定是使用 2 字节或 3 字节分支指令时，则采用相应的指令。与采用 BR 伪指令相比可缩短汇编时间。

BR 分支 BR

[说明]

- BR 伪指令值仅可用于一个代码段内。
- 直接跳转目的点可作为 BR 伪指令的操作数。在表达式的开始不能描写表示当前位置计数器的“$”。
- 必须满足下列条件以达到最优化：

  <1> 表达式内不能多于 1 个标记或前向引用符号。

  <2> 不能描述具有 ADDRESS 属性的 EQU 符号。

  <3> 不能对“具有 ADDRESS 属性的表达式 1 – 具有 ADDRESS 属性的表达式 2”描述用 EQU 定义的符号。

  <4> 不能描述已经由 HIGH/LOW/DATAPOS/BITPOS 操作符进行操作的具有 ADDRESS 属性的表达式。

  如果这些条件不满足，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) 由于该行与分支目的点之间的位置在–80H 和+7FH 范围内，BR 伪指令产生一个 2 字节分支指令(BR $addr16) 。

(2) 由于该行与分支目的点之间的位置在–80H 和 +7FH 范围内，BR 伪指令将由 3 字节分支指令(BR !addr16)代替。

3.8 宏伪指令

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

使用宏伪指令来定义宏功能可省去重复编写同一组指令的需要，从而提高了程序编码的效率。宏的基本功能就是用一个名字代替一系列语句。

宏伪指令包括 MACRO、LOCAL、REPT、IRP、EXITM 和 ENDM。

这一节将详细描述每个宏伪指令。有关宏功能的更多细节，参见**第 5 章 宏**。

MACRO　　宏　　MACRO

(1) MACRO (宏)

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
宏名称	MACRO	[形式参量 [,...]]	[;注释]
	⋮		
	宏程序体		
	⋮		
	ENDM		[;注释]

[功能]

- MACRO 伪指令通过把在符号字段指定的宏名称赋给在该伪指令和 ENDM 伪指令之间描述的一串语句(称为宏程序体)来执行宏定义。

[用途]

- 将在源程序中经常使用的一系列语句定义成一个宏名称。定义之后仅描述了被定义的宏名称(用于宏引用)，与宏名称对应的宏程序体随后被展开。

[说明]

- MACRO 伪指令必须与 ENDM 伪指令成对出现。
- 对于在符号字段描述的宏名称，参见 **2.2.3 组成语句的字段**中的符号描述惯例
- 在助记词字段描述该被定义宏名称来引用一个宏(参见**应用示例**)。
- 对于将要在操作数字段描述的形式参量，其描述方式遵循与符号描述惯例相同的原则。
- 每个宏伪指令最多只能描述 16 个形式参量。
- 形式参量仅在宏程序体内有效。
- 如果保留字被描述成形式参量，将导致错误。然而，如果描述了一个用户定义的符号，优先识别该符号为形式参量。
- 形式参量的数量必须与实际参数的数量一致。
- 在宏程序体内定义的名称或标记如果用 LOCAL 伪指令声明，则只对一次宏扩展有效。
- 宏的嵌套(也即，在宏程序体内引用了其他宏)最多允许 8 级，包括 REPT 和 IRP 伪指令。
- 在一个独立源模块内定义宏的数量没有特别限制。也就是说，只要内存空间可用，就可以定义宏。
- 形式参量定义行、引用行和符号名不输出到前后对照表中。
- 在一个宏程序体中不能定义两个或多个段，否则输出错误信息。

MACRO 宏 MACRO

[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) 定义了一个宏，其宏名称为 “ADMAC”，并两个形式参量“PARA1”和“PARA2”。

(2) 该伪指令表示宏定义的结束。

(3) 宏“ADMAC”被引用。

LOCAL 局部的 LOCAL

(2) LOCAL(局部的)

[编写格式]

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

[功能]

- LOCAL 伪指令声明了在操作数字段指定的符号是一个局部符号，该符号仅在宏程序体内有效。

[用途]

- 如果一个宏在程序体内定义了符号，且该宏被多次引用，则汇编器将对该符号输出一个双重定义错误。使用 LOCAL 伪指令则可以引用(或调用)一个在宏程序体内定义了符号的宏，且可被引用(或调用)多次。

[说明]

- 关于在操作数字段定义符号名的一些惯例，参见 **2.2.3 组成语句的字段**中关于符号描述的惯例。
- 在每个宏扩展中，被声明为 LOCAL 的符号将用符号“??RAn” (其中 n=0000 ~ FFFF)所代替。宏替换之后的符号“??RAn”将按照全局符号的方式处理，且存于符号表中，从而可以在符号名“??RAn”下被引用。
- 如果一个符号在宏程序体内被描述且该宏被引用多次，这意味着，该符号将在源模块内被定义多次。因此，有必要声明该符号是一个局部符号，仅在宏程序体内有效。
- LOCAL 伪指令仅可在宏定义内使用。
- 在使用操作数字段指定的符号之前必须描述 LOCAL 伪指令(也即，LOCAL 伪指令必须在宏程序体的开始被描述)。
- 在源模块内，将要采用 LOCAL 伪指令定义的符号名必须不同(也即，在每个宏内使用的局部符号不能同名)。
- 可在操作数字段指定的局部符号的数量没有限制，只要位于在同一行内。但是，在宏程序体的符号的数量限制为 64 个。如果声明了 65 个或更多个符号，汇编器将输出错误信息，存储该宏定义为一个空的宏程序体。即使宏被调用也不会被扩展。
- 用 LOCAL 伪指令定义的宏不能被嵌套。
- 用 LOCAL 伪指令定义的符号不能在宏以外被调用(或引用)。
- 保留字不能被描述成操作数字段内的符号名。但是，如果一个符号名作为用户定义符号被描述，优先识别该符号为本地符号。
- 作为 LOCAL 伪指令的操作数被声明的符号不在前后对照表和符号表中输出。
- 宏扩展时不输出 LOCAL 伪指令的语句行。
- 如果在宏定义内创建的 LOCAL 声明的符号与宏定义的形式参量同名，则输出错误信息。

LOCAL　　局部的　　LOCAL

[应用示例]

<源程序>

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

REF1:   MAC1                    ;(3)

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

REF2:   MAC1                    ;(5)

        END
```

<说明>

(1) LOCAL 定义符号名“LLAB”为局部符号。

(2) BR 伪指令引用了宏 MAC1 内的局部符号“LLAB”。

(3) 宏引用调用了宏 MAC1。

(4) 由于局部符号 “LLAB”在宏 MAC1 定义之外被引用，该描述导致错误。

(5) 宏引用调用了宏 MAC1.

上述应用示例的汇编列表如下所示。

<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 F407, STNO    13 (    0) Undefined symbol reference 'LLAB'
***   ERROR F303, STNO    13 (   13) Illegal expression
 11   14
 12   15 000005              REF2:   MAC1                ;(5)
      16               #1   ;
      17 000005          #1   ??RA0001:
      18 000005  14FE      #1          BR      $??RA0001      ;(2)
 13   19
 14   20                        END
```

REPT　　重复　　REPT

(3) REPT (重复)

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[标记:]	REPT	绝对表达式	[;注释]
	⋮		
	ENDM		[;注释]

[功能]

- REPT 伪指令通知汇编器重复展开在该伪指令和 ENDM 伪指令之间(称为 REPT-ENDM 程序体)的一系列语句，重复次数等于操作数字段指定的绝对表达式的值。

[用途]

- 在源程序中使用 REPT 和 ENDM 伪指令重复描述一系列语句。

[说明]

- 如果 REPT 伪指令没有和 ENDM 伪指令成对出现，则出错。
- 在 REPT-ENDM 程序体内，宏引用、REPT 伪指令和 IRP 伪指令最多可被嵌套 8 层。
- 如果在 REPT-ENDM 程序体内出现了 EXITM 伪指令，则汇编器停止对 REPT-ENDM 程序块后续部分的展开。
- 在 REPT-ENDM 程序体内可描述汇编控制指令。
- 在 REPT-ENDM 程序体内不可描述宏定义。
- 在操作数字段描述的绝对表达式按照无符号 16 位数计算。如果表达式的值为 0，则不展开任何内容。

REPT　　重复　　REPT

[应用示例]

<源程序>

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

← REPT-ENDM 程序块

<说明>

(1)　REPT 伪指令通知汇编器连续 3 次展开 REPT-ENDM 程序块。

(2)　该伪指令表示 REPT-ENDM 程序块的结束。

当汇编上述源程序时，REPT-ENDM 程序块按照如下汇编列表展开：

<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 程序块被展开了 3 次。在汇编列表里，没有显示在源模块中被 REPT 伪指令定义的语句(1)和(2)。

IRP 不确定重复 IRP

(4) IRP (不确定重复)

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[标记:]	IRP	形式参量 <[实际参数 [,...]]>	[;注释]
	⋮		
	ENDM		[;注释]

[功能]

- IRP 伪指令通知汇编器重复展开在该伪指令和 ENDM 伪指令之间描述的一系列语句，重复次数等于实际参数的值，而其形式参量被在操作数字段指定的实际参数代替。

[用途]

- 在源程序内使用 IRP 和 ENDM 伪指令重复描述一系列语句，仅有部分语句变成变量。

[说明]

- IRP 伪指令必须与 ENDM 成对出现。
- 在操作数字段最多可描述 16 个实际参数。
- 在 IRP-ENDM 程序体内，宏引用、REPT 伪指令和 IRP 伪指令最多可被嵌套 8 层
- 如果在 IRP-ENDM 程序体内出现了 EXITM 伪指令，汇编器停止对 IRP-ENDM 程序块后续部分的展开。
- 不可以在 IRP- ENDM 程序体内描述宏定义。
- 在 IRP-ENDM 程序体内可描述汇编控制指令。

IRP 不确定重复 IRP

[应用示例]

<源程序>

```
        NAME    SAMP1
        CSEG

        IRP     PARA,<0AH,0BH,0CH>  ; (1)
        ADD     A,#PARA
        MOV     [DE],A
        ENDM                        ; (2)

        END
```

IRP-ENDM 块

<说明>

(1) 形式参量是“PARA”，实际参数是下列三个 “0AH”, “0BH”和“0CH”。
IRP 伪指令通知汇编器展开 IRP-ENDM 程序块 3 次（即实际参数的值），同时形式参量“PARA”被实际参数 “0AH”, “0BH”和“0CH” 代替。

(2) 该伪指令表示 IRP-ENDM 程序块的结束。

当汇编上述源程序时，IRP-ENDM 程序块被展开，形式如下面汇编表所示：

<汇编列表>

```
                NAME    SAMP1
                CSEG

                ADD     A,#0AH      ; (3)
                MOV     [DE],A
                ADD     A,#0BH      ; (4)
                MOV     [DE],A
                ADD     A,#0CH      ; (5)
                MOV     [DE],A

                END
```

被语句(1)和(2) 定义的 IRP-ENDM 程序块被展开了 3 次（相当于实际参数的值）。

(3) ADD 指令中，PARA 被 0AH 所代替。

(4) ADD 指令中，PARA 被 0BH 所代替

(5) ADD 指令中，PARA 被 0CH 所代替。

EXITM 退出宏 EXITM

(5) EXITM (退出宏)

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
[标记:]	EXITM	无	[;注释]

[功能]

- EXITM 伪指令强行终止 MACRO 伪指令定义的宏程序体的展开，以及 REPT-ENDM 程序体或 IRP-ENDM 程序体的重复。

[用途]

- 在由 MACRO 伪指令定义的宏程序体内使用了条件汇编功能(参见 **4.7 条件汇编控制指令**)时使用该功能。
- 如果条件汇编功能与宏程序体内的其他指令结合起来使用，则部分不必汇编的源程序有可能被汇编，除非强行使用 EXITM 伪指令从宏内返回控制。此时，要确定使用 EXITM 伪指令。

[说明]

- 如果 EXITM 伪指令在宏程序体内描述，在 ENDM 伪指令之前的指令都将作为宏程序体被存储。
- EXTIM 伪指令仅表示宏扩展期间宏的结束。
- 如果在 EXITM 伪指令的操作数字段内有内容描述，汇编器将输出错误信息，但是仍执行 EXITM 处理。
- 如果宏程序体内出现 EXITM 伪指令，汇编器将强行从 IF/_IF/ELSE/ELSEIF/_ELSEIF/ENDIF 程序块的嵌套层里返回汇编器进入宏程序体的那一级嵌套层。
- 如果 EXITM 伪指令在 INCLUDE 文件中出现，而该 INCLUDE 文件由扩展宏程序体内的 INCLUDE 控制指令产生的，则汇编器将认为 EXITM 伪指令有效，并在那一级上终止宏扩展。

EXITM 退出宏 EXITM

[应用示例]

- 这个例子使用了条件汇编控制指令。参见 **4.7 条件汇编控制指令**
- 关于宏程序体和宏扩展，参见**第 5 章 宏。**

<源程序>

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

      CSEG
$      SET(SW1)                          ; (10)
      MAC1                   ; (11)
L1:    NOP
      END
```

宏程序体
IF 块
ELSE 块
IF 块
ELSE 块
宏引用

EXITM　　退出宏　　EXITM

<说明>

(1) 宏“MAC1”使用宏程序体内的条件汇编功能（2）和（4）到（8）
(2) 这里定义了一个 IF 块用于条件汇编。如果开关名“SW1”为真（非“0”），则汇编 ELSE 块。
(3) 该伪指令强行终止在（4）及后面的宏程序体的展开。
　　如果省略了 EXITM 伪指令，当宏被展开时，汇编器继续（6）及后面的汇编处理
(4) 此处定义了一个 ELSE 块用于条件汇编。如果开关名“SW1”为假（“0”），则汇编 ELSE 块。
(5) ENDIF 控制指令表示条件汇编的结束。
(6) 此处定义了另一个 IF 块。如果开关名“SW2”为真（非“0”），则汇编后面的 IF 块。
(7) 此处定义了另一个 ELSE 块。如果开关名“SW2”为假（“0”），则汇编后面的 ELSE 块。
(8) ENDIF 指令表示结束在（6）和（7）中的条件汇编处理。
(9) 该伪指令表示宏程序体的结束。
(10) SET 控制指令给开关名”SW1”赋真值（非 0），并设置条件汇编的条件。
(11) 该宏引用调用了宏”MAC1”。

当汇编上述例子中的源程序时，发生宏展开，如下所示。

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

宏被展开的部分

通过引用（11）中的宏，扩展了宏“MAC1”的宏程序体。由于在（10）中开关名“SW1”设真值，则汇编宏程序体内的第一个 IF 块。又因为在 IF 块的结尾描述了 EXITM 伪指令，不再执行后续的宏展开。

ENDM 结束宏 ENDM

(6) ENDM (结束宏)

[编写格式]

符号字段	助记词字段	操作数字段	注释字段
无	ENDM	无	[;注释]

[功能]

- ENDM 伪指令通知汇编器终止执行一系列作为宏功能定义的语句。

[用途]

- ENDM 伪指令必须总是位于 MACRO、REPT 和/或 IRP 伪指令之后的一系列语句的结尾处。

[说明]

- MACRO 伪指令和 ENDM 伪指令之间被描述的一系列语句成为宏程序体。
- REPT 伪指令和 ENDM 伪指令之间被描述的一系列语句成为 REPT-ENDM 程序体。
- IRP 伪指令和 ENDM 伪指令之间被描述的一系列语句成为 IRP-ENDM 程序体

[应用示例]

示例 1 <MACRO-ENDM>

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

ENDM	end macro	ENDM

示例 2 <REPT-ENDM>

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

示例 3 <IRP-ENDM>

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

3.9 汇编终止伪指令

汇编终止伪指令通知汇编器源模块结束。必须在每个源模块的结尾处描述该汇编终止伪指令。

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

END　　结束　　END

(1) END (结束)

[编写格式]

符号字段	助记词字段	操作数符	注释字段
无	END	无	[;注释]

[功能]

- END 伪指令通知汇编器源模块结束。

[用途]

- END 伪指令必须总是位于每个源模块的最后。

[说明]

- 汇编器连续汇编一个源模块直到源模块中出现 END 伪指令。因此，END 伪指令必需在每个源模块的结尾处。
- END 伪指令后面总是输入一个换行(LF)代码。
- 如果在 END 伪指令后面出现得语句不是空格、tab、LF 或注释语句，则汇编器输出告警信息。

[应用示例]

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

<说明>

(1)　总是在每个源模块的结尾处描述 END 伪指令。

[备忘录]

第 4 章 控制指令

这一章解释控制指令。控制指令为汇编器的操作提供了详细指导。

4.1 控制指令概述

在源程序中描述的控制指令为汇编器的操作提供详细的指导。
这些指令不受目标代码生成的影响。
可用的控制指令类型如下所示。

表 4-1. 控制指令列表

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

在源程序中描述控制指令的方式与描述汇编伪指令的方式相同。
对于**表 4-1 控制指令列表**中列出的控制指令，下列指令与在汇编程序起始命令行指定的汇编选项具有相同的功能。
控制指令与命令行汇编选项之间的对应关系列于**表 4-2. 控制指令与汇编选项**。

表 4-2.　控制指令与汇编选项

控制指令	汇编选项
PROCESSOR	-C
DEBUG/NODEBUG	-G/-NG
DEBUGA/NODEBUGA	-GA/-NGA
XREF/NOXREF	-KX/-NKX
SYMLIST/NOSYMLIST	-KS/-NKS
FORMFEED/NOFORMFEED	-LF/-NLF
TITLE	-LH
WIDTH	-LW
LENGTH	-LL
TAB	-LT

关于指定控制指令和通过命令行指定汇编选项的方法，参见 **RA78K0S 汇编器包操作**。

4.2 处理器类型定义控制指令

在源模块文件中，处理器类型定义控制指令指定了用于汇编的目标设备的类型。

PROCESSOR　　processor　　PROCESSOR

(1) PROCESSOR (processor)

[编写格式]

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

[功能]

- 在源模块文件中，PROCESSOR 控制指令指定了用于汇编的目标设备的处理器类型

[用途]

- 用于汇编的目标设备的处理器类型必须在源模块文件或汇编程序的起始命令行指定。
- 如果各源模块文件没有定义用于汇编的目标设备的处理器类型，则在每次汇编操作中指定处理器类型。在各源模块文件内指定汇编目标设备可以节约时间，而在汇编程序起始部分指定则浪费了时间。

[说明]

- 只能在源模块文件的开头部分描述 PROCESSOR 控制指令。如果在其他地方描述了该控制指令，汇编将失败。
- 如果指定的处理器类型与用于汇编的实际目标设备不符，汇编失败。
- 在模块头只可指定一个 PROCESSOR 控制指令。
- 用于汇编的目标设备的处理器类型也可用位于汇编程序起始命令行的汇编选项（-C）指定。如果源模块文件中定义的处理器类型与起始命令行中的定义不同，汇编器将输出告警信息，并优先选择在起始命令行中定义的处理器类型。
- 即使在起始命令行定义了汇编选项（-C），汇编器也会对 PROCESSOR 控制指令执行语法检查。
- 如果在源模块文件或起始命令行都没有指定处理器类型，则汇编失败。

[应用示例]

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

        NAME            TEST

        CSEG
          ⋮
```

4.3 调试信息输出控制指令

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

(1) DEBUG/NODEBUG (debug/nodebug)

[编写格式]

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

[功能]

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

[用途]

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

[说明]

- 只能在源模块文件的开头部分描述 DEBUG 或 NODEBUG 控制指令。
- 如果省略了 DEBUG 或 NODEBUG 控制指令，则汇编器假设指定了 DEBUG 控制指令。
- 通过在起始命令行使用汇编选项（-C）来定义增加局部符号信息。
- 如果在源模块文件中的控制指令定义与在起始命令行中的定义不同，则在命令行中的定义优先。
- 即使指定了汇编选项（-C），汇编器也会对 DEBUG 或 NODEBUG 控制指令进行语法检查。

(2) DEBUGA/NODEBUGA (debuga/nodebuga)

[编写格式]

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

[功能]

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

[用途]

- 当在汇编器或在结构化源程序级上进行调试时使用 DEBUGA 控制指令。在源程序级上进行调试将需要集成调试器。
- 当下述情况时使用 NODEBUGA 控制指令：
  1. 执行无汇编源的调试。
  2. 仅需要对象（如用 PROM 估值）

[说明]

- 只可在源模块文件的开头部分描述 DEBUGA 或 NODEBUGA 控制指令。
- 如果省略了 DEBUGA 或 NODEBUGA 控制指令，汇编器将假定指定了 DEBUGA 控制指令。
- 如果指定了两个或多个控制指令，最后被指定的控制指令优先级高于其他指令。
- 使用位于起始命令行的汇编选项（-GA/-NGA）可指定增加汇编源程序的调试信息。
- 如果在源模块文件中的控制指令定义与在起始命令行中的定义不同，则在命令行中的定义优先。
- 即使指定了汇编选项（- NGA），汇编器也会对 DEBUGA 或 NODEBUGA 控制指令执行语法检查。
- 如果通过 C 编译器或结构化汇编预处理器来编译或结构化汇编某调试信息输出，则在汇编该输出汇编源时不描述调试信息输出控制指令。汇编时所必需的控制指令作为 C 编译器或结构化汇编预处理器的控制语句输出至汇编源。

4.4 交叉引用表输出定义控制指令

在源模块文件中，交叉引用表输出指定控制指令用于指定输出或不输出交叉引用表。

(1) XREF/NOXREF (xref/noxref)

[编写格式]

```
[Δ]$[Δ]XREF
[Δ]$[Δ]XR                    ; 缩写格式
[Δ]$[Δ]NOXREF                ; 缺省假设
[Δ]$[Δ]NOXR                  ; 缩写格式
```

[功能]

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

[用途]

- 当所获得的信息来自从源模块文件中定义的每个符号均被引用或者有多少这样的符号在源模块中被引用时，使用 XREF 控制指令输出交叉引用表。
- 为了指定在每个汇编操作中输出或不输出交叉引用表，在源模块文件中定义 XREF 或 NOXREF 控制指令可节省时间和精力。

[说明]

- 只可在源模块文件的开头部分描述 XREF 或 NOXREF。
- 如果指定了两个或多个控制指令，最后被指定的控制指令优先级高于其他指令。
- 也可通过在起始命令行的汇编选项（-KX/-NKX）中指定输出或不输出交叉引用表。
- 如果源模块文件定义的处理器类型与起始命令行中的定义不同，则在起始命令行中的定义优先于在源模块中的定义。
- 即使指定了汇编选项（-NP），汇编器也会对 XREF/NOXREF 控制指令进行语法检查。

(2) SYMLIST/NOSYMLIST (symlist/nosymlist)

[编写格式]

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

[功能]

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

[用途]

- 使用 SYMLIST 控制指令输出一个符号列表。

[说明]

- 只可在源模块文件的开头部分描述 SYMLIST 或 NOSYMLIST。
- 如果指定了两个或多个控制指令，最后被指定的控制指令优先级高于其他指令。
- 也可通过起始命令行的汇编选项（-KS/-NKS）中规定输出或不输出交叉引用表。
- 如果源模块文件定义的处理器类型与起始命令行中的定义不同，则在起始命令行中的定义优先于在源模块中的指定。
- 即使指定了汇编选项（-NP），汇编器也会对 SYMLIST/NOSYMLIST 控制指令进行语法检查。

4.5 包含控制指令

包含控制指令用于源模块文件中，说明在源模块文件中包含另一模块文件。
在编写源程序时有效利用该控制指令可节省时间和精力。

INCLUDE　　include　　lNCLUDE

(1) INCLUDE (include)

[编写格式]

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

[功能]

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

[用途]

- 可被两个或多个源模块共享的一大组语句应组合成一个独立文件，称为 INCLUDE 文件。如果该语句组必须在每个源模块中使用，用 INCLUDE 控制指令指定所需 INCLUDE 文件的文件名。通过使用该控制指令，编写源模块的时间和精力可大大减少。

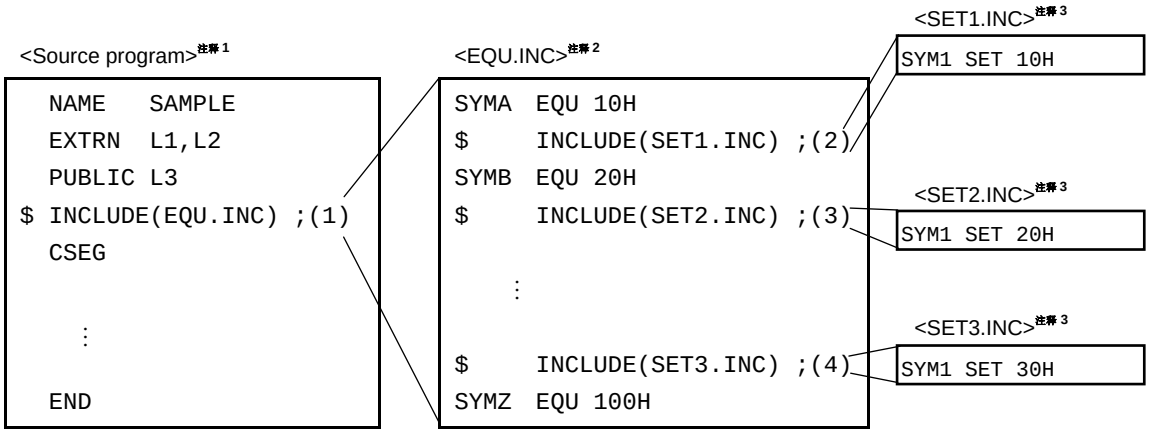
[说明]

- 只可在普通源程序中描述 INCLUDE 控制指令。
- INCLUDE 文件的路径名或驱动器名可通过汇编选项（-I）指定。
- 汇编器按照下列顺序搜索 INCLUDE 文件的读取路径：
  (a) 当指定的 INCLUDE 文件没有路径名定义时：
    <1> 源文件包含路径
    <2> 汇编选项(-I)指定的路径
    <3> 环境变量 INC78K0S 指定的路径
  (b) 若指定 INCLUDE 的文件具有以(\)开头的驱动器名或路径名时，与 INCLUDE 文件一起指定的路径将加在 INCLUDE 文件名上作为前缀。如果 INCLUDE 文件由相对路径指定（不是以(\)开头），加在 INCLUDE 文件名前作为前缀的路径名称按照上面（a）的顺序进行。
- INCLUDE 文件的嵌套层数最多为 7 级。换句话说，在汇编列表中 INCLUDE 文件显示的嵌套级数最多为 8 级（这里，嵌套指在一个 INCLUDE 文件中指定一个或多个 INCLUDE 文件）。
- 在 INCLUDE 文件中不需要描述 END 伪指令。
- 如果被指定 INCLUDE 文件不能打开，则汇编器终止操作。

INCLUDE include INCLUDE

- INCLUDE 文件必须用在 INCLUDE 文件内与 ENDIF 控制指令成对出现的 IF 或_IF 控制指令内。如果在 INCLUDE 文件展开入口处的 IF 级与紧跟着 INCLUDE 文件展开的 IF 级不符，则汇编器输出错误信息，强迫 IF 级返回 INCLUDE 文件展开入口处的 IF 级。
- 若在 INCLUDE 文件中定义了宏，宏定义必须包含在 INCLUDE 文件内。如果在 INCLUDE 文件里意外地出现了 ENDM 伪指令（没有相应的 MACRO 伪指令），将输出错误信息并忽略 ENDM 伪指令。如果对于在 INCLUDE 文件里描述的 MACRO 伪指令缺少 ENDM 伪指令，汇编器输出错误信息，但将假定相应的 ENDM 伪指令已经描述，仍处理该宏定义。

[应用示例]



注释 **1.** 一个源文件里可定义两个或多个$IC 控制指令。 同一 INCLUDE 文件也可被指定多次。
2. 对 “EQU.INC”.文件可指定两个或多个$IC 控制指令。
3. 在任何 INCLUDE 文件“SET1.INC”, “SET2.INC”, and “SET3.INC”中都不能指定$IC 控制指令。

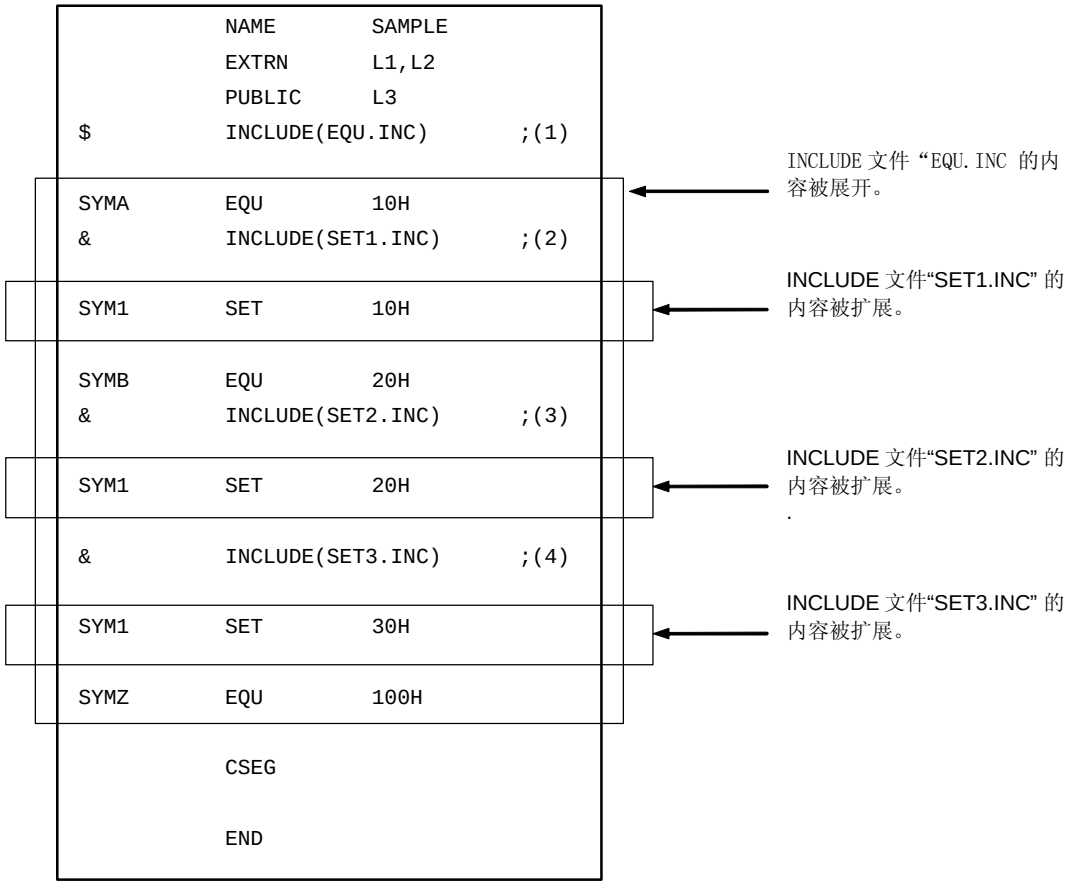
<说明>

(1) 该控制指令指定 “EQU.INC 为 INCLUDE 文件。

(2), (3), (4) 这些控制指令指定“SET1.INC”, “SET2.INC”和“SET3.INC”为 INCLUDE 文件。

当该源程序被汇编时，INCLUDE 文件的内容被展开成如下形式：

INCLUDE　　include　　INCLUDE



4.6 汇编列表控制指令

在源模块文件中，使用汇编列表控制指令控制汇编列表的输出格式如换页、抑制列表输出和副标题输出。
汇编列表控制指令包括：

- EJECT
- LIST and NOLIST
- GEN and NOGEN
- COND and NOCOND
- TITLE
- SUBTITLE
- FORMFEED and NOFORMFEED
- WIDTH
- LENGTH
- TAB

EJECT　　eject　　EJECT

(1) EJECT (eject)

[描述格式]

```
[Δ]$[Δ]EJECT
[Δ]$[Δ]EJ                                    ; 缩写格式
```

[缺省假设]

- 没有指定 EJECT 控制指令。

[功能]

- EJECT 控制指令促使汇编器执行汇编列表的换页（走纸）操作。

[用途]

- 在源模块中需要弹出一页汇编列表的那一行，编写 EJECT 控制指令。

[说明]

- EJECT 控制指令仅可在普通源程序中描述。
- 在输出 EJECT 控制指令本身的图标之后执行汇编列表的换页操作。
- 如果在起始命令行指定了汇编选项（-NP 或（-LLO），或者汇编列表输出被另一个控制指令去能，EJECT 控制指令变成无效。关于汇编器选项，参见 **RA78K0S 汇编器包的操作**。
- 如果在 EJECT 控制指令后面有非法描述，汇编器将输出错误信息。

[应用示例]

<源模块>

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

<说明>

(1) 当通过 EJECT 控制指令执行换页操作时，输出的汇编列表形式如下所示。

```
          ⋮
        MOV     [DE+], A
        BR      $$

$       EJECT
--------------------------------------- 换页
        CSEG
          ⋮
        END
```

(2) LIST/NOLIST (list/nolist)

[编写格式]

[Δ]$[Δ]LIST	; Default assumption
[Δ]$[Δ]LI	; Abbreviated format
[Δ]$[Δ]NOLIST	
[Δ]$[Δ]NOLI	; Abbreviated format

[功能]

- LIST 控制指令通知汇编器汇编列表输出的起始行。
- NOLIST 控制指令通知汇编器汇编列表输出必须抑制的行。
  所有在 NOLIST 控制指令定义后面被描述的源语句都将被汇编，但是不会输出至汇编列表，直到在源程序中出现 LIST 控制指令。

[用途]

- 使用 NOLIST 控制指令限制汇编列表输出的数量。
- 使用 LIST 控制指令取消对 NOLIST 控制指令指定的汇编列表输出的抑制。
  使用 NOLIST 和 LIST 控制指令的组合，可以控制汇编列表输出的数量以及列表的内容。

[说明]

- 仅可在普通源程序中描述 LIST/NOLIST 控制指令。
- NOLIST 控制指令的功能是抑制汇编列表的输出，并不打算终止汇编处理的进程。
- 如果在 NOLIST 控制指令之后指定了 LIST 控制指令，在 LIST 控制指令之后所描述的语句将再一次在汇编列表中输出。LIST 或 NOLIST 控制指令的图象也将在汇编列表中输出。
- 如果 LIST 或 NOLIST 控制指令都没有被指定，源模块中的所有语句都将输出至汇编列表。

[应用示例]

```
            NAME    SAMP1
$           NOLIST          ; (1)
DATA1   EQU     10H
DATA2   EQU     11H
                :
DATAX   EQU     20H
DATAY   EQU     20H
$           LIST            ; (2)
            CSEG
                :
            END
```

这部分语句将不会输出至汇编列表中。

<说明>

(1) 由于这里指定了 NOLIST 控制指令， “$ NOLIST” 之后和（2）LIST 控制指令之前的语句将不会在汇编列表中输出。NOLIST 控制指令自身的图象将输出至汇编列表。

(2) 由于这里指定了 LIST 控制指令，该控制指令之后的语句将再次输出至汇编列表中。LIST 控制指令自身的图象也将输出至汇编列表。

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

[编写格式]

[编写格式]

[Δ]$[Δ]GEN　　; 缺省假定

[Δ]$[Δ]NOGEN

[功能]

- GEN 控制指令告诉汇编器将宏定义行、宏引用行和宏展开行输出至汇编列表。
- NOGEN 控制指令告诉汇编器输出宏定义行和宏引用行，但是抑制宏展开行。

[用途]

- 使用 GEN/NOGEN 控制指令限制汇编列表输出的数量。

[说明]

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

[应用示例]

<源程序>

```
            NAME    SAMP
$           NOGEN
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，输出的汇编列表如下所示。

```
            NAME    SAMP
$           NOGEN
ADMAC       MACRO   PARA1,PARA2
            MOV     A,#PARA1
            ADD     A,#PARA2
            ENDM
            CSEG
            ADMAC   10H, 20H

            MOV     A,#10H
            AUD     A,#20H

        END
```

不输出宏展开部分。

<说明 n>

(1) 由于指定了 NOGEN 控制指令，宏展开行将不输出到汇编列表里。

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

[编写格式]

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

[功能]

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

[用途]

- 使用 CONG/NOCOND 条件控制限制汇编列表输出的数量。

[说明]

- 仅可在普通源程序中描述 COND/NOCOND 控制指令。
- 如果 COND 和 NOCOND 控制指令都没有被指定，则汇编器将已经满足汇编条件的行以及不满足汇编条件的行都输出至汇编列表。
- 在 COND 或 NOCOND 控制指令本身的图像输出至汇编列表之后，才发生指定的列表控制。
- 即使在 NOCOND 控制指令控制了列表输出之后，汇编器仍累加 ALNO 和 STNO 计数。
- 如果在 NOCOND 控制指令之后指定 COND 控制指令，则汇编器将恢复输出那些满足汇编条件的行以及已经描述了 IF/_IF, ELSEIF/_ELSEIF, ELSE 和 ENDIF 的行。

[应用示例]

<源程序>

```
        NAME    SAMP

$       NOCOND
$       SET(SWI)

$       IF(SWI)

        MOV     A,#1H

$       ELSE
        MOV     A,#0H
        ENDIF

        END
```

这部分虽然已被汇编，但是不输出至汇编列表中。

TITLE　　title　　TITLE

(5) TITLE (title)

[编写格式]

```
[Δ]$[Δ]TITLE[Δ]([Δ]'title-string'[Δ])
[Δ]$[Δ]TT[Δ]([Δ]'title-string'[Δ])                    ; 缩写格式
```

[缺省假设]

- 如没有指定 TITLE 控制指令，将空着汇编列表头的 TITLE 列。

[功能]

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

[用途]

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

[说明]

- 仅可在源模块文件的开头部分描述 TITLE 控制指令。
- 如果同时指定了两个或多个 TITILE 控制指令，汇编器将认为最后指定的 TTIEL 控制指令有效。
- 标题字串最多可指定 60 个字符。如果被指定标题串包含 61 个或更多字符，汇编器只承认标题串的前 60 个字符有效。
  但是，如果汇编列表文件指定的每行字符长度（数量为“X”）是 119 个字符或更小，则将被接受“X-60”个字符。
- 如果要使用单引号标志（’）作为标题串的一部分，则连续写两个单引号标志。
- 如果没有指定标题串（标题串的字符数=0），汇编器保留 TITLE 列空白。
- 如果在指定的标题串中发现 **2.2.2 字符组**之外的任何字符，汇编器将输出“!”代替 TITLE 列中的非法字符。
- 汇编列表的标题也可通过位于汇编器起始命令行的汇编选项（-LH）指定。

TITLE title TITLE

[应用示例]

<源模块>

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

当汇编上述源程序时，输出汇编列表如下一页显示（每页行数为 72）。

TITLE　　title　　TITLE

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

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

     Assemble list

 ALNO  STNO ADRS  OBJECT     M I  SOURCE STATEMENT

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

Segment information:

 ADRS LEN NAME

 0000  0001H  ?CSEG

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

(6) SUBTITLE (subtitle)

[编写格式]

```
[Δ]$[Δ]SUBTITLE[Δ]([Δ]'character-string'[Δ])
[Δ]$[Δ]ST[Δ]([Δ]'character-string'[Δ])                    ; Abbreviated format
```

[缺省假设]

- 没有指定 SUBTITLE 控制指令时，空着汇编列表头部的 SUBTITLE 部分。

[功能]

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

[用途]

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

[说明]

- 仅可在普通源模块中描述 SUBTITLE 控制指令。
- 副标题最多可指定 72 个字符。
  如果指定的字符串包括 73 个或更多字符，汇编器将只承认串的前 72 个字符有效。2 字节字符按 2 个字符计，制表符计为一个字符。
- 由 SUBTITLE 控制指令指定的字符串将打印在已经指定 SUBTITLE 控制指令的那页的后一页上。但是，如果在一页的顶部（首行）指定了该控制指令，则将在该页上打印副标题。
- 如果没有指定 SUNBTITLE 控制指令，汇编器空着 SUBTITLE 部分。
- 如果要使用单引号标志(')作为标题串的一部分，则连续写两个单引号标志。
- 如果 SUBTITLE 部分的字符串是 0，则空着 SUNTITLE 列。
- 如果在指定的标题串中发现 **2.2.2 字符组**之外的任何字符，汇编器输出“!”代替 SUBTITLE 列中的非法字符。如果描述了 CR（0DH）字符，则出错，且汇编列表里没有内容输出。如果描述了 00H 字符，则从 00H 到结束单引号（'）之间的任何内容都不会输出。

SUBTITLE　　subtitle　　SUBTITLE

[应用示例]

<源模块>

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

<说明>

(1) 该控制指令指定字符串‘THIS IS SUBTITLE 1’。

(2) 该控制指令指定一个换页操作。

(3) 该控制指令指定字符串 ‘THIS IS SUBTITLE 2’。

(4) 该控制指令指定一个换页操作。

(5) 该控制指令指定字符串 ‘THIS IS SUBTITLE 3’。

SUBTITLE subtitle SUBTITLE

该示例的汇编列表如下所示（每页指定行数为 80）。

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

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

 ALNO  STNO ADRS   OBJECT     M I  SOURCE STATEMENT

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

THIS IS SUBTITLE 1

 ALNO  STNO ADRS   OBJECT     M I  SOURCE STATEMENT

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

THIS IS SUBTITLE 2

 ALNO  STNO ADRS   OBJECT     M I  SOURCE STATEMENT

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

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

(7) FORMFEED/NOFORMFEED (formfeed/noformfeed)

[编写格式]

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

[功能]

- FORMFEED 控制指令通知汇编器在汇编列表文件的最后输出一个 FORMFEED 代码。
- NO FORMFEED 控制指令通知汇编器不在汇编列表文件的最后输出 FORMFEED 代码。

[用途]

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

[说明]

- 仅可在源模块文件的开头部分描述 FORMFEED 或 NOFORMFEED 控制指令。
- 打印汇编列表时，如果在一页的中间指示打印结束，则不打印列表的最后一页。在这种情况下，使用 FORMFEED 控制指令或汇编选项（-LF）在汇编列表的最后增加一个 FORMFEED 代码。
  很多情况下，FORMFEED 代码在文件末尾输出。因此，如果在列表文件的末尾出现了 FORMFEED 代码，则会弹出一张空白页。为了避免这种情况，设置 NOFORMFEED 控制指令或汇编选项（-NLF）作为缺省值。
- 如果同时指定了两个或多个 FORMFEED/NOFORMFEED 控制指令，只有最后一个被指定的控制指令有效。
- 也可通过位于汇编程序起始命令行的汇编选项（-LF）或（-NLF），指定输出或不输出进纸代码。
- 如果源模块中的控制指令定义（FORMFEED/NOFORMFEED）与在起始命令行中的指定（-LF/-NLF）不同，则起始命令行中的指定优先于在源模块中的定义。
- 即使在起始命令行指定了汇编选项（-NP），汇编器仍将对 FORMFEED 或 NOFORMFEED 控制指令执行语法检查。

(8) WIDTH (width)

[编写格式]

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

[缺省假定]

$WIDTH (132)

[功能]

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

[用途]

- 使用 WIDTH 控制指令改变列表文件每行的列数。

[说明]

- 仅可在源模块文件的开头部分描述 WIDTH 控制指令。
- 如果同时指定了两个或多个 WIDTH 控制指令，只有最后被指定的控制指令有效。
- 也可通过位于汇编器起始命令行的汇编选项（-LW）来指定列表文件每行的列数。
- 如果源模块中的控制指令定义（WIDTH）与在起始命令行中的指定（-LW）不同，起始命令行中的指定优先于在源模块中的定义。
- 即使在起始命令行指定了汇编选项（-NP），汇编器仍将对 WIDTH 控制指令执行语法检查。

(9) LENGTH (length)

[编写格式]

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

[缺省假定]

$LENGTH (66)

[功能]

- LENGTH 控制指令定义列表文件每页的行数。“lines-per-page”可以是 0， 或 20 至 32767 范围内的一个值。

[用途]

- 使用 LENGTN 控制指令改变列表文件每页的行数。

[说明]

- 只可在源模块文件的开头部分描述 LENGTH 控制指令。
- 如果同时指定了两个或更多个 LENGTH 控制指令，只有最后被指定控制指令有效。
- 也可通过位于汇编器起始命令行的汇编选项（-LW）来指定列表文件每行列数。
- 如果源模块中的控制指令定义（LENGTH）与在起始命令行中的指定（-LL）不同，起始命令行中的指定优先于在源模块中的定义。
- 即使在起始命令行指定了汇编选项（-NP），汇编器仍将对 LENGTH 控制指令执行语法检查。

TAB tab TAB

(10) TAB (tab)

[编写格式]

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

[缺省假定]

$TAB (8)

[功能]

- TAB 控制指令规定列表文件中 TAB 键一跳的列数。“number-of-columns”可以是 0 至 8 范围内的任一值。
- TAB 控制指令指定 TAB 键一跳的列数，通过用几个空格字符代替源模块中的 HT（Horizontal Tabulation）码，该列数成为输出任何列表的表格处理的基础。

[用途]

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

[说明]

- 只可在源模块文件的开头部分描述 TAB 控制指令
- 如果同时指定了两个或更多个 TAB 控制指令，只有最后被指定控制指令有效。
- 也可通过位于汇编器起始命令行的汇编选项（-LT）来指定 TAB 键一跳的列数。
- 如果源模块中的控制指令定义（TAB）与在起始命令行中的指定（-LT）不同，起始命令行中的指定优先于在源模块中的定义。
- 即使在起始命令行 指定了汇编选项（-NP），汇编器仍将对 TAB 控制指令执行语法检查。

4.7 条件汇编控制指令

条件汇编控制指令用于在源模块中选择一系列语句，通过设置条件汇编 switch 选择对其进行汇编或不汇编。
这些控制指令包括 IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF 控制指令和 SET/RESET 控制指令。
有效使用这些控制指令，可以使源模块排除不必要语句，汇编时可尽量少地改变或不改变源模块。

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

[编写格式]

```
  [Δ]$[Δ]IF[Δ]([Δ]switch-name[[Δ]:[Δ]switch-name]…[Δ])
or [Δ]$[Δ]_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]

- 控制指令设置条件限制源程序语句的汇编。
  在 IF 或_IF 控制指令和 ENDIF 控制指令之间的源程序语句容易被有条件汇编。
- 如果条件表达式的估计值或被IF或_IF条件指令（也就是IF或_IF条件）指定的switch名为真（除了 00H），源程序中从IF或_IF条件指令直到出现下一个条件汇编控制指令（ELSEIF/_ELSEIF, ELSE或 ENDIF）之间的源语句都将被汇编。对于后续的汇编处理，汇编器将继续处理ENDIF控制指令之后的语句。
  如果 IF 或_IF 条件为假（00H），源程序中从 IF 或_IF 条件指令直到出现下一个条件汇编控制指令（ELSEIF/_ELSEIF, ELSE, or ENDIF）之间的源语句都将不会被汇编。
- 只有当ELSEIF或_ELSEIF控制指令之前所描述的所有条件汇编控制指令的条件都不满足时，才检查ELSEIF或_ELSEIF条件指令的真假状态。
  如果条件表达式的估计值或被 ELSEIF 或_ELSEIF 条件指令（也就是 ELSEIF 或_ELSEIF 条件）指定的 switch 名为真（除了 00H），源程序中从 ELSEIF 或_ELSEIF 条件指令直到出现下一个条件汇编控制指令（ELSEIF/_ELSEIF, ELSE,或 ENDIF）之间的源语句都将被汇编。对于后续的汇编处理，汇编器将继续处理 ENDIF 控制指令之后的语句。
  如果 ELSEIF 或_ELSEIF 条件为假，源程序中从 ELSEIF 或_ELSEIF 条件指令直到出现下一个条件汇编控制指令（ELSEIF/_ELSEIF, ELSE 或 ENDIF）之间的源语句都不会被汇编。
- 如果所有在ELSE控制指令之前所描述的IF/_IF和ELSEIF/_ELSEIF控制指令的条件都不满足，源程序中从ELSE条件指令直到出现ENDIF条件汇编控制指令之间的源语句都将被汇编。
- ENDIF控制指令通知汇编器终止源语句进行条件汇编。

[用途]

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

[功能]

- IF 和 ELSEIF 控制指令用于 switch 名的真/假条件判断，而_IF 和_ELSEIF 控制指令用于条件表达式的真/假条件判断。

  IF/ELSEIF 和_IF/_ELSEIF 都可以合并起来使用。也就是说，ELSEIF/_ELSEIF 可以和 IF 或_IF 及 ENDIF 成对使用。
- 为一个条件表达式描述一个绝对表达式。
- 描述 switch 名的规则同符号描述惯例一致（具体细节参见 **2.2.3 组成语句的域**）。但是，可作为 switch 名被识别的最大字符数总是 31。
- 如果用 IF 或 ELSEIF 控制指令指定了两个或多个 switch 名，每个 switch 名用分号（;）分开。每个模块最多可使用 5 个 switch 名。
- 当两个或多个 switch 名用 IF 或 ELSEIF 控制指令指定时，如果一个 switch 名的值为真，则判断满足 IF 或 ELSEIF 条件。
- 将由 IF 或 ELSEIF 控制指令指定的每个 switch 名的值必须使用 SET 或 RESET 控制指令定义（**参见 4.7 (2) SET/RESET**）。因此，如果源模块中，由 IF 或 ELSEIF 控制指令指定的 switch 名的值没有提前使用 SET 或 RESET 控制指令设置，则假定值被复位。
- 如果指定的 switch 名或条件表达式包含非法描述，汇编器将输出一个错误信息，并确定计算值为假。
- 描述 IF 或_IF 控制指令时，IF 或_IF 控制指令必须总是与 ENDIF 控制指令成对出现。
- 如果在宏程序体内描述了 IF_ENDIF 块，而且已由 EXITM 处理将控制从宏里返回，汇编器将强迫 IF 级返回宏程序体入口的那一级。这种状况下不会出现错误。
- 在一个 IF_ENDIF 块内描述另一个 IF_ENDIF 块称为 IF 控制指令的嵌套。IF 控制指令的嵌套最多允许 8 级。
- 在条件汇编中，对没有汇编的语句不产生目标代码，但这些语句会原样输出到汇编列表里。使用$NOCOND 控制指令可避免输出这些语句。

[应用示例]

示例 1

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

<说明>

(1) 如果 switch 名 “SW1”的值为真，将汇编“text1”中的语句。
如果 switch 名 “SW1”的值为假，则不汇编“text1”中的语句。
switch 名 “SW1”的值已由在“text0”中描述的 SET 或 RESET 控制指令设为真或假。

(2) 该指令表示条件汇编源语句范围的结束。

示例 2

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

<说明>

(1) switch 名 “SW1”的值已由在“text0”中描述的 SET 或 RESET 控制指令设为真或假。
如果 switch 名 “SW1”的值为真，将汇编“text1”中的语句，而不汇编“text2”中的语句。

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

(3) 该指令表示条件汇编源语句范围的结束。

示例 3

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

<说明>

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

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

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

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

(5) 该指令表示条件汇编源语句范围的结束

示例 4

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

<说明>

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

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

(3) 该指令表示条件汇编源语句范围的结束。

SET/RESET　　set/reset　　SET/RESET

(2) SET/RESET (set/reset)

[编写格式]

[∆]$[∆]SET[∆]([∆]switch-name[[∆]:[∆]switch-name]…[∆])
[∆]$[∆]RESET[∆]([∆]switch-name[[∆]:[∆]switch-name]…[∆])

[功能]

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

[用途]

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

[说明]

- 至少要用 SET 或 RESET 控制指令描述一个 switch 名。
  描述 switch 名所采用的原则与描述符号的惯例（2.2.3 组成语句的域）相同。但是，可作为 switch 名被识别的最大字符数总是 31 个。
- 除了保留字和其他 switch 名，指定的 switch 名可以与用户定义的符号相同。
- 如果用 SET 或 RESET 控制指令指定了两个或多个 switch 名，每个 switch 名用分号（;）分开。每个模块可最多使用 1000 个 switch 名。
- 一个已经用 SET 控制指令设为“真”的 switch 名，可用 RESET 控制指令改为“假”。反之亦然。
- 在源模块中，对于将要由 IF 或 ELSEIF 控制指令指定的 switch 名，必须用 SET 或 RESET 指令在描述 IF 或 ELSEIF 控制指令之前定义一次。
- switch 名不能输出到交叉引用表中。

[应用示例]

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

        END
```

<说明>

(1) 该指令给 switch 名“SW1”赋一个真值 (0FFH)。

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

(3) 该语句表示从（2）开始的条件汇编源语句范围的结束。

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

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

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

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

(8) 该指令表示从（5）开始的条件汇编源语句范围的结束。

4.8 其他控制指令

下列控制指令是由类似于 C 编译器和结构化汇编预处理器这样的高级程序输出的专用控制指令。

$TOL_INF
$DGS
$DGL

第五章 宏

本章解释如何使用宏功能。当在源程序中重复描述一系列语句时，宏是非常有用的功能。

5.1 宏概述

当在源程序中重复描述一系列或一组指令时，采用宏功能来描述程序非常实用。宏功能是指把通过 MACRO 和 ENDM 伪指令作为宏程序体定义的一系列语句（一个指令组）在宏名称被引用的地方展开。

宏用于提高源程序的编码效率，宏区别于子程序。

宏和子程序具有明显的特征，如下文所述。为了提高应用效率，根据特殊用途选择宏或子程序。

(1) 子程序

- 把一个必须在程序中重复多次的处理过程看作一个独立的子程序。子程序将被汇编器转换成机器语言，但仅可转换一次。
- 简单地编写一个子程序调用指令即可调用子程序（通常，设置参数的指令可在子程序之前或之后描述）。
  有效使用子程序可提高程序的内存使用效率。
- 通过把程序中的一系列处理进程编成子程序可以使程序结构化（这种结构化使程序员容易理解程序的整个结构，使程序设计变得容易）。

(2) 宏

- 宏的基本功能是用一个名字代替一组指令。
  采用 MACRO 和 ENDM 伪指令作为宏程序体定义的一系列（或一组）指令将在宏名称被引用的地方被展开。
- 当汇编器发现一个宏引用时，汇编器展开该宏程序体，并将这组指令转换成机器语言，同时，在宏引用时用实际参量代替宏程序体内的形式参量。
- 对宏可描述一些参量。
  比如，如果有些指令组，其处理过程相同，而操作数中所描述的数据不同，则可定义一个宏，把该数据分配给形式参量。在宏引用时通过描述宏名和实际参量，汇编器可处理多种仅在语句描述上不同的指令组。

使用子程序这种编程技术主要用来减少内存空间并使程序结构化，而使用宏则提高了程序的编码效率。

5.2 宏的使用

5.2.1 宏定义

使用 MACRO 和 ENDM 伪指令来定义宏。

[编写格式]

符号字段	记忆字段	操作数字段	注释字段
宏名称	MACRO	[形式参量 [,…]]	[;注释]
	⋮		
	ENDM		

[功能]

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

[应用示例]

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

<说明>

上面的例子说明了一个简单的宏定义，即，将“PARA1”和“PARA2”的值相加，结果存于寄存器 A 中。宏的名称是“ADMAC”，“PARA1”和“PARA2”是形式参量。

更多细节参见 **3.8 宏伪指令**中的**（1）MACRO（macro）。**

5.2.2 宏引用

为了调用宏，必须在源程序的记忆字段描述该已定义宏名称。

[编写格式]

符号字段	记忆字段	操作数字段	注释字段
[标记:]	宏名称	[实际参量 [,...]]	[;注释]

[功能]

- 该语句调用分配给在记忆字段指定的宏名称的宏程序体。

[用途]

- 使用该语句调用宏程序体。

[说明]

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

[应用示例]

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

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

<说明>

宏引用调用已经定义的宏名“ADMAC”。10H 和 20H 是实际参量。

5.2.3 宏展开

汇编器按照如下方式处理宏：

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

[应用示例]

When the macro referenced in **5.2.2 Macro reference** is assembled, the macro body will be expanded as shown below.当在 **5.2.2 宏引用**中被引用的宏被汇编时，宏程序体将被展开成如下形式。

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

        CSEG
          ⋮
        ADMAC   10H,20H         ; (1)
        MOV     A,#10H
        ADD     A,#20H
          ⋮
        END
```

宏定义

宏展开

<说明>

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

5.3 宏内的符号

可在宏内定义的符号分为两类：全局符号和局部符号

(1) 全局符号

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

(2) 局部符号

- 局部符号是用 LOCAL 伪指令定义的符号（参加 **3.8 宏伪指令**中的**（2）局部符号（local）**）。
- 局部符号可在用 LOCAL 伪指令声明为 LOCAL 的宏内被引用。
- 在宏之外不能引用局部符号。

[应用示例]

<源程序>

```
        NAME    SAMPLE
MAC1    MACRO
        LOCAL   LLAB            ; (1)
LLAB:
          ⋮
GLAB:                                          宏定义
        BR      LLAB            ; (2)
        BR      GLAB            ; (3)
        ENDM
          ⋮
REF1:   MAC1                    ; (4)  ←       宏引用
          ⋮
        BR      LLAB            ; (5)  ←       该语句描述是错误的。
        BR      GLAB            ; (6)
          ⋮                                    宏引用
REF2:   MAC1                    ; (7)  ←
          ⋮
        END
```

<说明>

(1)　该 LOCAL 伪指令定义标记 “LLAB”为局部符号。

(2)　BR 伪指令引用宏“MAC1”里的局部符号“LLAB”。

(3)　BR 伪指令引用宏“MAC1”里的全局符号“GLAB”

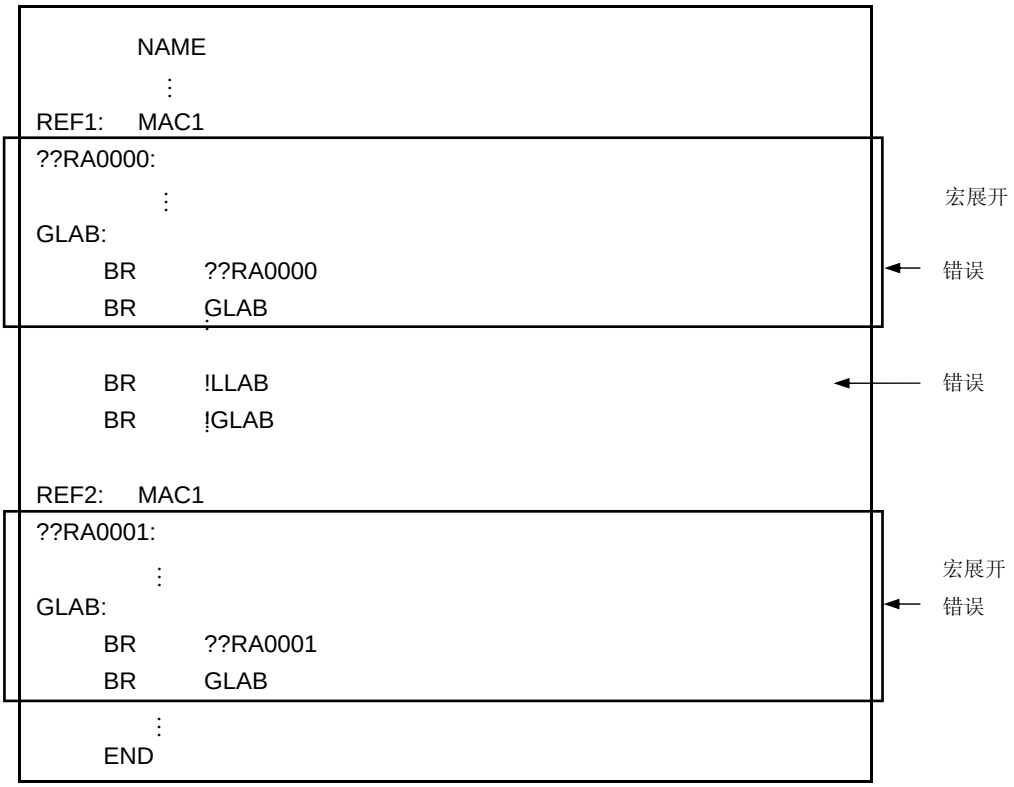
(4)　该语句引用宏“MAC1”。

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

(6)　BR 伪指令在宏“MAC1”定义之外引用全局符号“GLAB”。

(7)　该语句引用宏“MAC1”.同一宏被引用了两次。

当汇编上述例子的源程序时，宏程序体将被展开成如下形式。



<说明>

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

5.4 宏操作符

宏操作符有两种类型：“&”（和记号）和“”（单引号标记）。

(1) &（联接）

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

[应用示例]

宏定义

```
MAC     MACRO   P
LAB&P:
        D&B     10H
        DB      'P'
        DB      P
        DB      '&P'
        ENDM
```

← 形式参量 “P” 被识别。

宏引用

```
        MAC     1H
LAB1H:
        DB      10H
        DB      'P'
        DB      1H
        DB      '&P'
```

← “D” 和 “B” 被连接在一起，变成“DB”。

← 在一对单引号之间的& 作为数据处理。

(2) ' (单引号标记)

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

[应用示例]

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

        MAC1    '10,20,30'
```

当上述示例中的源程序被汇编时，宏"MAC1"将被展开如下形式。

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

IRP 扩展

[备忘录]

本章介绍了一些推荐的有效利用 RA78K0S 汇编包的方法。

有几种方式可有效利用 RA78K0S 来汇编源模块。本章只介绍其中一些技术。

(1) 启动汇编器时节省时间，减少故障

有些控制指令具有与汇编选项同样的功能，而且必须总在启动汇编器时使用，这些例子包括处理器类型定义（-C）和汉字码定义（-ZS/-ZE/-ZN）（仅限日语版）。建议在源模块文件中描述这样的控制指令。特别地，处理器类型定义不可省略，它应在源模块文件的开头部分通过 PROCESSOR 控制指令来指定。这样就避免了每次启动汇编程序时在起始命令行指定汇编选项（-C）的需要。记住，如果汇编选项没有在起始命令行指定，则将导致错误，此时，汇编器需要从起始行，用正确的汇编选项重新启动。

交叉引用表输出控制指令（XREF）也需要在模块的开头被指定。

示例

```
$       PROCESSOR(9026)
$       KANJICODE NONE
$       XREF

        NAME        TEST

        CSEG
          ⋮
```

(2) 如何开发具有高内存利用率的程序

与其它数据内存区域相比，短的直接寻址区是一块可用短字节长度指令访问的区域。

因此，有效使用这个区域可开发出具有高内存利用率的程序。

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

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

以下 6 种类型中可使用保留字：机器语言指令、伪指令、控制指令、操作符、寄存器名和 sfr 符号。保留字是由汇编器预先保留的字符串，不能用于其它特别目的。

在源程序各自的字段中可描述的保留字类型如下表所示。

符号字段	该字段中无保留字可描述
助记词字段	只有机器语言指令和指示可在该字段中描述。
操作数字段	只有操作符、sfr 符号和寄存器名可在该字段中描述。
注释字段	所有保留字均可在该字段中描述。

对于 sfr 列表，参见各设备的**特殊功能寄存器表。**

对于中断请求源列表，参见各设备文件中的**使用注意事项**

对于机器语言指令和寄存器名列表，参见各设备的用户手册

（1）保留字列表

操作符	AND	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	DEBUG	DEBUGA	DG	EJ
	EJECT	ELSE	ELSEIF	_ELSEIF	ENDIF
	FORMFEED	GEN	IC	IF	_IF
	INCLUDE	LENGTH	LI	LIST	NOCOND
	NODEBUG	NODEBUGA	NOFORMFEED	NOGEN	NOLI
	NOLIST	NOSYMLIST	NOXR	NOXREF	PC
	PROCESSOR	RESET	SET	ST	SUBTITLE
	SYMLIST	TAB	TITLE	TT	WIDTH
	XR	XREF			
其它	DGL	DGS	SFR	SFRP	TOL_INF

[备忘录]

（2）伪指令列表

序号	伪指令				功能/分类	备注
	符号字段	助记词字段	操作数字段	注释字段		
1	[段名]	CSEG	[重定位属性]	[;注释]	声明一个代码段的开始	
2	[段名]	DSEG	[重定位属性]]	[;注释]	声明一个数据段的开始	
3	[段名]	BSEG	[重定位属性]]	[;注释]	声明一个位段的开始	
4	[段名]	ORG	绝对表达式	[;注释]	声明一个绝对段的开始	禁止操作数内符号的前向参考。
5	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前向参考或外部参考。
6	名称	SET	绝对表达式	[;注释]	定义一个重定义名	名称:符号 禁止操作数内符号的前向参考
7	[标记:]	DB	{(size) 初始值 [,...]}	[;注释]	初始化或保留一个字节的数据区	标记:符号 字符串可代替初始值设置。
8	[标记:]	DW	{(size)初始值 [,...]}	[;注释]	初始化或保留一个字数据区字段	标记:符号
9	[标记:]	DS	绝对表达式	[;注释]	保留字节数据区字段	名称:符号 禁止操作数内符号的前向参考
10	名称	DBIT	无	[;注释]	保留比特数据区字段	名称:符号 禁止操作数内符号的前向参考
11	[标记:]	PUBLIC	符号名 [,...]	[;注释]	声明一个外部定义名称。	
12	[标记:]	EXTRN	符号名 [,...]	[;注释]	声明一个外部定义名称。	
13	[标记:]	EXTBIT	位符号名称[,...]	[;注释]	声明一个外部定义名称	符号名称限定在具有比特值的符号名中。
14	[标记:]	NAME	对象模块名	[;注释]	定义模块名	模块名: 符号
15	[标记:]	BR	表达式	[;注释]	自动选择一个分支指令	标记: 符号

序号	伪指令				功能/分类	备注
	符号字段	助记词字段	操作数字段	注释字段		
16	宏名称	MACRO	[形式参量 [,...]]	[;注释]	定义一个宏	宏名称: 符号
17	[标记:]	LOCAL	符号名 [,...]	[;注释]	定义一个符号，仅在宏内有效	仅在宏定义中使用
18	[标记:]	REPT	绝对表达式	[;注释]	在宏扩展期间指定重复次数	标记: 符号
19	[标记:]	IRP	形式参量, <实际参量 [,...]>	[;注释]	给形式参量分配一个实际参数	标记: 符号
20	[标记:]	EXITM	无	[;注释]	中断宏扩展	仅在宏定义中使用
21	无	ENDM	无	[;注释]	终止宏定义	仅在宏定义中使用
22	无	END	无	[;注释]	表示源模块的结束	

附录 C 最佳性能特征

（1）汇编器的最佳性能特征

项目	最佳性能特征	
	PC 版	WS 版
符号（局部+全局）数	65,535 符号[注解 1]	65,535 符号[注解 1]
交叉引用表可输出的符号数	65,534 符号[注解 2]	65,534 符号[注解 2]
对于一个宏引用，宏程序体的最大空间	1 MB	1 MB
所有宏程序体的全部空间	10 MB	10 MB
一个文件中段的数量	256 段	256 段
一个文件内的宏和 include 定义	10,000	10,000
一个 include 文件内的宏和 include 定义	10,000	10,000
再定位数据[注解 3]	65,535 项目	65,535 项目
行数量数据	65,535 项目	65,535 项目
一个文件中 BR 伪指令的数量	32,767 伪指令	32,767 伪指令
一行内的字符数	2,048 字符[数届 4]	2,048 字符[数届 4]
符号长度	256 字符	256 字符
定义Switch名的数量[注解 5]	1,000	1,000
Switch名的字符长度[注解 5]	31 字符	31 字符
一个文件内 include 文件的嵌套层数	8 层	8 层

注解 **1.** 采用 XMS。如无 XMS，则采用文件。

2. 采用内存。如无内存，则采用文件。

3. “再定位数据”是指当汇编器不能确定符号的值时传递到连接器的数据。
例如，当通过 MOV 指令查阅一个外部引用符号时，在 .rel 文件中生成两项再定位数据。

4. 包含回车符和换行符。如果一行中输入 2,049 个或更多个字符，则输出错误信息并终止处理操作。

5. 通过 SET/RESET 伪指令并使用$IF 等来设置 switch 名称的真/假。

（2）连接器的最佳性能特征

项目	最佳性能特征	
	PC 版	WS 版
符号（局部+全局）数	65,535 符号	65,535 符号
同一段的行数量数据	65,535 项	65,535 项
段的数量	65,535 段	65,535 段
输入模块数量	1,024 模块	1,024 模块

[备忘录]

??RAn 130
?An 38
?BSEG 38, 95
?CSEG 38, 86
?CSEGFX 38, 86
?CSEGIX 38, 86
?CSEGT0 ..... 38, 86
?CSEGUP 38, 86
?DSEG 38, 90
?DSEGDSP 38, 90
?DSEGIH 38, 90
?DSEGIX 38, 90
?DSEGL 38, 90
?DSEGS 38, 90
?DSEGSP 38, 90
?DSEGUP 38, 90

[A]

绝对汇编器 19
绝对段 27, 82, 97
绝对项 64, 78
实际参数 189, 195
ADDRESS 39, 78
ADDRESS 项 68
字母字符 33
AND 操作符 48, 53
区域保留伪指令 107
汇编器 16, 21
汇编器选项 26, 146
汇编包 16
汇编语言 17
汇编表控制指令 159
汇编终止伪指令 142
AT 重定位属性 .....
..... 85, 86, 89, 90, 93, 95
自动分支指令选择伪指令 124

[B]

后向参考 77
二进制数 42
位 39
位访问 71
位段 27, 82, 92
位符号 73
BITPOS 操作符 48, 61
BR 指令 23, 125
BSEG 伪指令 92

[C]

CALLT0 重定位属性 85, 86
CALLT 指令 85
字符组 33
字符串常量 42
代码段 27, 82, 84
注释字段 46, 201
串联 194
COND 控制指令 166
条件汇编功能 23, 137, 166, 178
常量 41
控制指令 27, 145
交叉引用表输出指定控制指令 152
CSEG 伪指令 84

[D]

数据段 27, 82, 88
DATAPOS 操作符 48, 61
DB 伪指令 108
DBIT 伪指令 114
DEBUG 控制指令 26, 150
调试信息输出控制指令 149
DEBUGA 控制指令 26, 151
十进制数 42
DGL 控制指令 26, 186
DGS 控制指令 27, 186
伪指令 81, 201
DS 伪指令 112
DSEG 伪指令 88
DSPRAM 重定位属性 89, 90
DW 伪指令 110

[E]

EJECT 控制指令 160
ELSE 控制指令 179
ELSEIF 控制指令 179
END 伪指令 143
ENDIF 控制指令 179
ENDM 伪指令 140
EQ 操作符 48, 55
EQU 伪指令 101

EXITM 伪指令.....137
表达式.....48
EXTBIT 伪指令.....118
外部定义声明.....115, 120
外部引用声明.....115, 116, 118
外部引用项.....64, 78
EXTRN 伪指令.....116

[F]

FIXED 重定位属性.....85, 86
形式参数.....128, 187, 194
FORMFEED 控制指令.....26, 174
前向参考.....77

[G]

GE 操作符.....48, 56
GEN 控制指令.....164
通用寄存器.....43
通用寄存器对.....43
全局符号.....130, 191
GT 操作符.....48, 56

[H]

十六进制数.....42
HIGH 操作符.....48, 60

[I]

idea-L 编辑器.....16
IF 控制指令.....179
IHRAM 重定位属性.....89, 90
INCLUDE 控制指令.....156
包含控制指令.....155
IRP 伪指令.....135
IRP-ENDM 程序快.....135
IXRAM 重定位属性.....85, 86, 89, 90

[L]

标记.....36
LE 操作符.....48, 57
LENGTH 控制指令.....26, 176
库管理程序.....16
行.....32
链**接**伪指令.....115
链**接**器.....16, 22
LIST 控制指令.....162
列表转换器.....16
LOCAL 伪指令.....130
LOCAL 符号.....191
LOW 操作符.....48, 60
LRAM 重定位属性.....89, 90
LT 操作符.....48, 57

[M]

机器语言.....17
宏.....23, 187
宏程序体.....128, 130, 137, 189
宏定义.....164, 188
MACRO 伪指令.....128, 187
宏伪指令.....127
宏表达式.....164, 190
宏名称.....36, 128, 188, 189
宏操作符.....194
宏引用.....164, 189
MASK 操作符.....48, 62
内存初始化伪指令.....107
记忆法.....40
记忆字段.....40, 199
MOD 操作符.....48, 52
模块化编程.....19
模块体.....25, 27
模块头.....25, 26
模块名.....36, 122, 123
模块尾.....25, 28

[N]

名称.....36, 101
NAME 伪指令.....123
NE 操作符.....48, 55
NOCOND 控制指令.....166
NODEBUG 控制指令.....26, 150
NODEBUGA 控制指令.....26, 151
NOFORMFEED 控制指令.....26, 174
NOGEN 控制指令.....166
NOLIST 控制指令.....162
NOSYMLIST 控制指令.....26, 154
NOT 操作符.....48, 53
NOXREF 控制指令.....26, 153
NUMBER.....39, 78
文件数.....21
NUMBER 项.....68
数字字符.....33
数字常量.....42

[O]

目标转换器 16
目标模块 123, 150
目标数 42
操作数 41, 74, 76
操作数字段 41, 199
操作符 48
操作符优先次序 49
优化功能 23
OR 操作符 48, 54
ORG 伪指令 97

[P]

PROCESSOR 控制指令 26, 148
处理器类型指定控制指令 147
工程管理器 16
PUBLIC 伪指令 120

[R]

可重定位汇编器 19
可重定位项 64, 78
可重定位属性 64, 77, 85, 89, 93
REPT 伪指令 133
REPT-ENDM 程序块 133
RESET 控制指令 184

[S]

SADDR 重定位属性 89, 90
SADDRP 重定位属性 89, 90
段 21, 27, 83
段定义伪指令 84
段名 36, 86, 90, 95, 98
SET 控制指令 184
SET 伪指令 105
SHL 操作符 48, 59
SHR 操作符 48, 58
源模块 25, 143
特殊字符 34, 44
特殊功能寄存器 43
语句 32
结构化汇编器预处理器 16
子程序 189
SUBTITLE 控制指令 171
副标题节 171
Switch 名 180, 183
符号 21, 36, 191, 199
符号属性 38, 77
符号定义伪指令 100
SYMLIST 控制指令 26, 154

[T]

TAB 控制指令 26, 177
TITLE 控制指令 26, 168
TOL_INF 控制指令 26, 186

[U]

UNIT 重定位属性 85, 86, 89, 90, 93, 95
UNITP 重定位属性 85, 86, 89, 90

[W]

WIDTH 控制指令 26, 175

[X]

XOR 操作符 48, 54
XREF 控制指令 26, 153

[备忘录]