用户手册

NEC

RA78K0R Ver. 1.00

汇编器

语言篇

目标设备

78K0R 系列

文件编号　U17835CA1V0UM00 (第一版)
印刷日期　2007年12月　CP(K)

日本印刷

[备忘]

Windows 是微软公司在美国或其他国家中的注册商标或商标。

- **本文档所刊登的内容有效期截至 2007 年 12 月。将来可能未经预先通知而更改。在实际进行生产设计时，请参阅各产品最新的数据表或数据手册等相关资料以获取本公司产品的最新规格。并非所有的产品和/或型号都向每个国家供应。请向本公司销售代表查询产品供应及其他信息。**
- 未经本公司事先书面许可，禁止复制或转载本文件中的内容。本文件所登载内容的错误，本公司概不负责。
- 本公司对于因使用本文件中列明的本公司产品而引起的，对第三者的专利、版权以及其它知识产权的侵权行为概不负责。本文件登载的内容不应视为本公司对本公司或其他人所有的专利、版权以及其它知识产权作出任何明示或默示的许可及授权。
- 本文件中的电路、软件以及相关信息仅用以说明半导体产品的运作和应用实例。用户如在设备设计中应用本文件中的电路、软件以及相关信息，应自行负责。对于用户或其他人因使用了上述电路、软件以及相关信息而引起的任何损失，本公司概不负责。
- 虽然本公司致力于提高半导体产品的质量及可靠性，但用户应同意并知晓，我们仍然无法完全消除出现产品缺陷的可能。为了最大限度地减少因本公司半导体产品故障而引起的对人身、财产造成损害（包括死亡）的危险，用户务必在其设计中采用必要的安全措施，如冗余度、防火和防故障等安全设计。
- 本公司产品质量分为：

  “标准等级”、“专业等级”以及“特殊等级”三种质量等级。

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

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

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

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

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

（注）

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

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

M5 02.11-1

[备忘]

前 言

该手册是为了使用户更容易地、正确地理解**RA78K0R汇编器包**（此后称为**RA78K0R**）中每个程序的基本功能以及描述源程序的方法而设计的。

该手册没有说明如何操作RA78K0R中的各个程序。因此，在你理解了本手册中的内容后，请阅读**RA78K0R汇编器包用户手册操作篇(U17836E)**（此后称为**操作篇**）以用来操作汇编器包中的各个程序。

本手册中与RA78K0R相关的描述适用于1.00版或更高的版本。

[目标读者]

写作本手册的目的，是为了帮助使用微控制器(78K0R系列)进行开发应用的工程师更好的理解设备的功能和指令。

[结构]

本手册由以下六个章节和附录组成：

第1章　概述

概述RA78K0R中的所有基本功能。

第2章　如何描述源程序

概述如何描述源程序，并说明了汇编器中的运算符。

第3章　伪指令

说明了如何写入以及使用伪指令，并包含有应用示例。

第4章　控制指令

说明了如何写入以及使用控制指令，并包含有应用示例。

第5章　宏

解释了所有宏功能，包括宏定义、宏引用和宏扩展。

关于宏的伪指令在**第3章　伪指令**中进行说明。

第6章　产品应用

介绍一些用于描述源程序的推荐方法。

附录

这些附录包括了保留字列表，伪指令列表以及索引。

本手册没有对指令集进行详细说明。关于这些指令的详细信息，敬请参阅软件开发所使用的微控制器用户手册。同样，关于各个体系结构的指令信息，敬请参阅软件开发所使用的微控制器用户手册（硬件版）。

[宏]

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

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

[约定]

以下的符号和缩写用于整本手册中：

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

[相关文件]

与本手册相关的文件（用户手册）如下。
本文中出现的相关文件可能会包含初稿版本。
然而，不会对初稿版本作特殊标记。

文件名		文件编号
RA78K0R V1.00版汇编器	操作	U17836E
	语言	本手册
CC78K0R V1.00版C编译器	操作	U17838E
	语言	U17837E
SM + 系统仿真器	操作	U18010E
PM+ 6.20版		U17990E
ID78K0R-QB 3.20版综合调试器	操作	U17839E

注意　上表列出的相关文档可能会随时改变而不做通知。请确保在设计时使用每个文档的最新版本。

[备忘]

目 录

插图列表

插图编号　　标题　　页码

表格列表

第 1 章　概述

本章节描述了RA78K0R在微控制器软件开发中所起的作用及其特点。

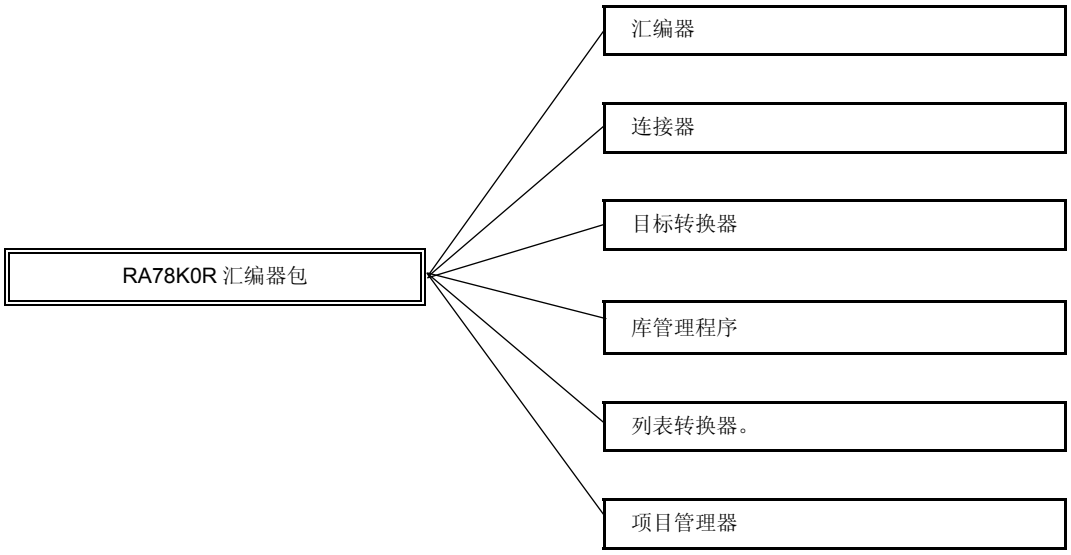
1.1　汇编器概述

RA78K0R汇编器 (后面简称"RA78K0R")是用于将以78K0R系列微控制器的汇编语言来编码的源程序转换为机器语言编码的一系列程序的通用术语。

RA78K0R包含5个程序：汇编器，连接器，对象转换器，库管理和列表转换器。

另外，RA78K0R也提供了PM+，它可以帮助你在Windows中完成对程序的编辑、编译/汇编，连接和调试等一系列操作。

图 1-1　RA78K0R 汇编器包

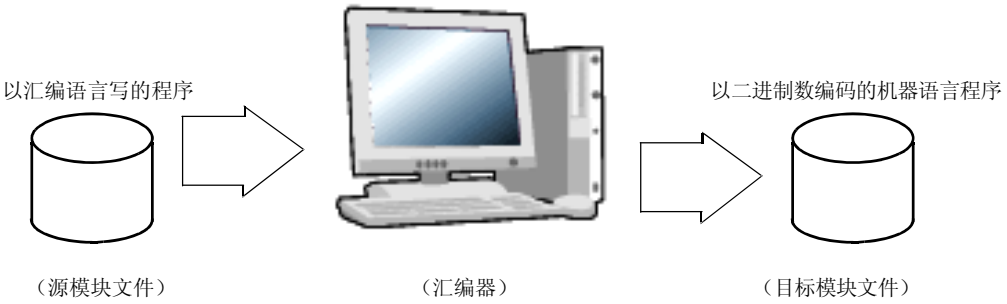


1.1.1　什么是汇编器？

(1)　汇编语言与机器语言

汇编语言是微控制器最基本的程序语言。
微控制器中的微处理器需要程序和数据来完成它的工作。这些程序和数据必须由用户来写入微控制器的内存中。
由微控制器处理的程序和数据是称为机器语言的二进制数的集合。
然而，对于用户来说，机器语言编码非常难记，因此会导致频繁地发生错误。幸运的是，已经存在易于被人理解的方法，它通过使用英文缩写或助记符来表示原始机器语言编码的含义。使用这种符号编码方式的基础程序语言系统称为汇编语言。
由于机器语言是唯一的可以被微控制器直接处理的程序语言，因此需要另一个可以将在汇编语言中创建的程序转换为机器语言的程序。这个程序就称为汇编器。

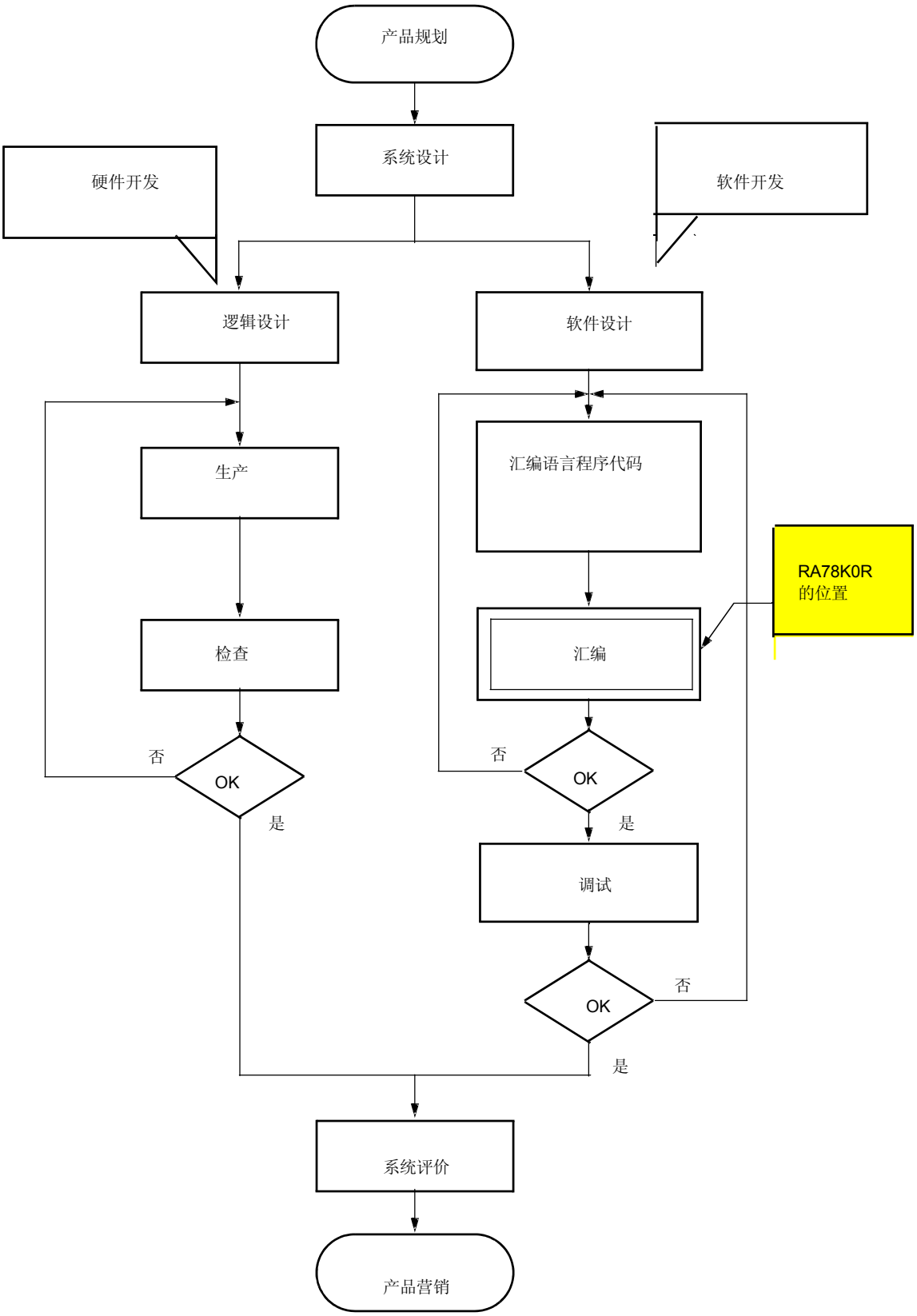
图 1-2　汇编器流程



1.1.2　应用微控制器的产品开发以及RA78K0R的作用

下图说明了“在产品开发过程中汇编语言程序设计”所处的位置。

图 1-3　应用微控制器的产品开发过程



1.1.3　什么是浮动汇编器？

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

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

根据将地址置于机器语言指令的方法，汇编器总的来说可以分成完全汇编器和可重定位汇编器。

- **完全汇编器**

  完全汇编器将以汇编语言进行汇编的机器语言指令放置到绝对地址中。

- **可重定位汇编器**

  在可重定位汇编器中，为汇编语言进行汇编得到的机器语言指令所分配的地址是暂定的。绝对地址随后将由连接器来确定。

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

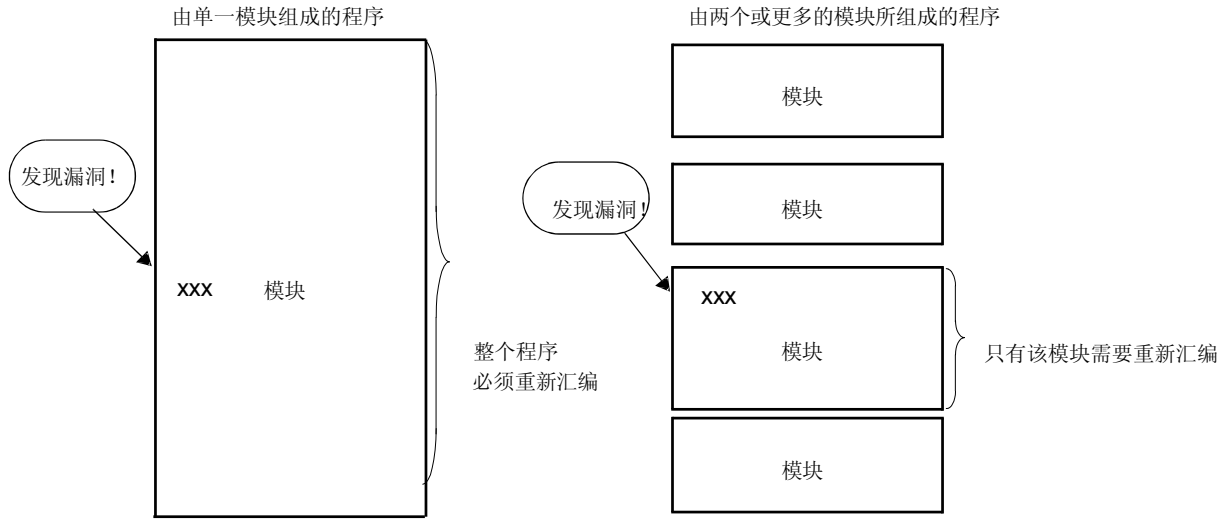
可重定位汇编器是一种适合于模块化程序设计的汇编器，它具有以下优势：

(1) 提高开发效率

同时完成一个大型程序是非常困难的。在这种情况下，将程序根据每个功能分成多个模块能让两个或更多的程序员来并行开发子程序，同时也提高了开发效率。

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

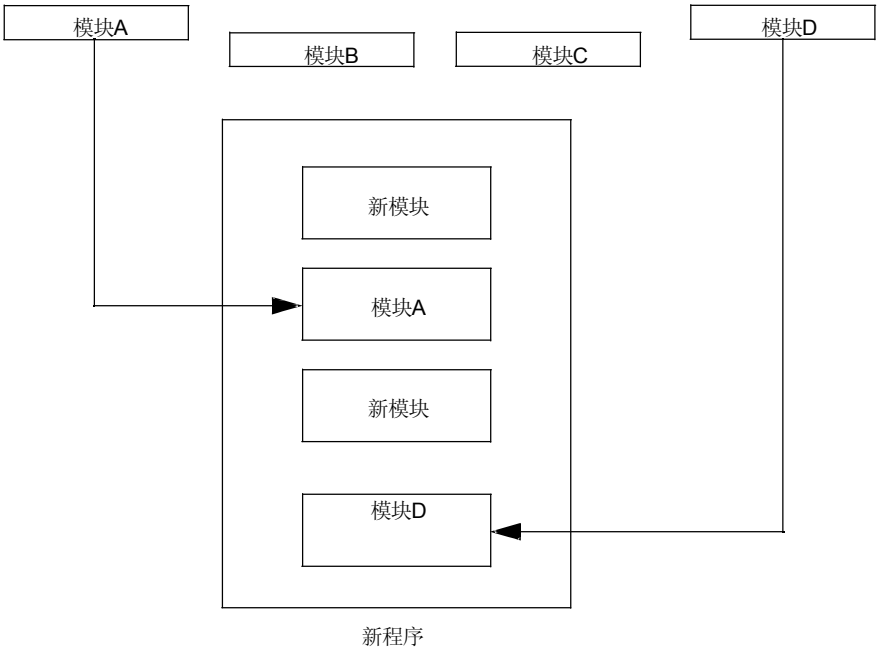
图 1-4　为调试重新进行汇编



(2)　资源利用

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

图 1-5　使用存在的模块的程序开发



1.2　开发程序前的提示

开始开发程序之前请参考以下内容。

1.2.1　RA78K0R的最大使用性能

(1)　汇编器的最大使用性能

条目	最大使用性能
符号数量（局部+公共）	65,535 个符号
可以输出交叉引用表的符号的数量	65,534 个符号注 1
用于一个宏引用的宏体的尺寸	1 MB
所有宏体的总尺寸	10 MB
一个文件中的区段数量	256 个区段
一个文件中的宏和要包含的规范	10,000
一个被包含的文件中的宏和包含的规范	10,000
可重定位的数据注 2	65,535 个项目
行号数据	65,535 个项目
一个文件中的 BR/CALL 指令的数量	32,767 条指令
每行中的字符数	2,048 个字符注 3
符号长度	256 个字符
开关名定义的数量注 4	1,000
开关名的字符长度注 4	31 个字符
区段名称的字符长度	8 个字符
模块名称的字符长度（NAME 伪指令）	256 个字符
宏定义伪指令中的形参数量	16 个参数
宏引用时的实参数量	16 个参数
IRP 伪指令中的实参数量	16 个参数
宏定义内的本地符号数量	64 个符号
宏的嵌套层数（宏引用，REPT 伪指令，IRP 伪指令）	8 层
通过 TITLE 控制指令，-lh 选项指定的字符数量	60 个字符注 5
通过 SUBTITLE 控制指令指定的字符数量	72 个字符
一个文件中 **include** 指令的嵌套层数	8 层
条件汇编嵌套层数	8 层
由-I 选项指定包含文件的路径数量	64 个路径
通过-d 选项定义的符号数量	30 个符号

注1.　不包括模块名称和区块名称的数量。
使用存储器，如果没有存储器，则使用文件。
注2.　当汇编器不能解决符号值时，被传递到连接器中的信息。
例如，当一个外部引用符号被MOV指令引用时，将会在.rel文件中生成两个重定位信息。
注3.　这里包含CR和LF代码。如果在一行中描述2049个或更多的字符，那么将输出一个警告消息并忽略2049之后的字符。
注4.　开关名称由SET/RESET伪指令设置为真/ 假，且被$IF 等一起使用。
注5.　如果在汇编列表文件（“X”）的单行指定的字符数量最大为119，该数字为“X-60”或更少。

(2)　连接器的最大使用性能

项目	最大工作特性
符号数量（局部+公共）	65,535 个符号
同一区段的行号数据	65,535个项目
区段的数量	65,535 个区段[注]
输入模块的数量	1,024 个模块
存储器区域名称的字符长度	256个字符
存储器区域的数量	100个区域[注]
通过-b选项可以指定的库文件数量	64个文件
由-I选项指定包含文件的路径数量	64个路径

注　默认包含这些已经定义的。

1.3　RA78K0R的特点

RA78K0R具有以下特点：

(1) 宏功能

当同一组指令必须在源程序中反复描述时，可以用宏定义来描述一组指令的执行。
通过使用该宏功能，可以提高编码效率和程序的可读性。

(2) 分支指令的优化功能

RA78K0R具有自动分支指令选择导引(BR和CALL)。
为了创建一个带有高存储效率的程序，字节分支指令必须依照分支指令的分支目的范围来进行描述。然而，对于程序员来说，描述一个分支指令的同时，又需要注意每个分支的目的范围是非常麻烦的。通过描述BR指令或CALL指令，汇编器将会依照分支目的范围生成适当的分支指令。这被称为分支指令的优化功能。

(3) 条件汇编功能

使用该功能时，一部分源程序可以依照预先确定的条件指定为汇编语言或非汇编语言。
如果在源程序中描述调试语句，是否将调试语句转换为机器语言，则可以通过为条件汇编设置一个开关来选择。当不再需要调试语句时，源程序可以加入汇编且不会对程序进行大的修改。

第 2 章　如何描述源程序

本章节描述了源程序的描述方法，表达式和运算符。

2.1　基本结构

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

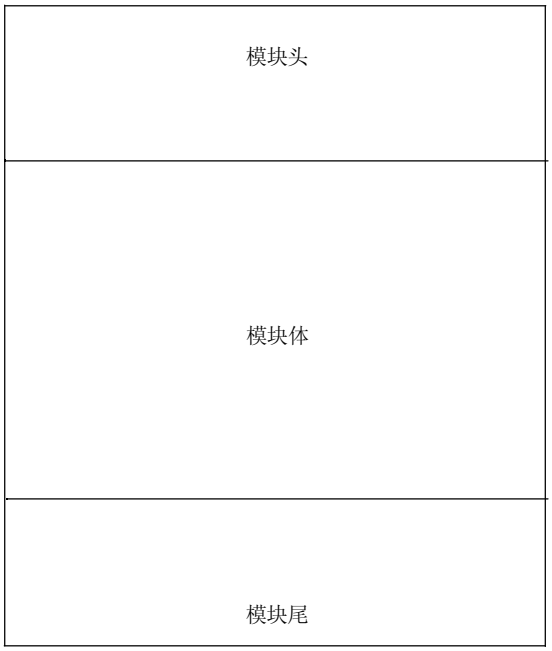
作为输入到编译程序中的单元的每个模块主要由以下三部分组成：

(1)　模块头

(2)　模块体

(3)　模块尾

图 2-1　源模块的结构



2.1.1　模块头

下表中显示的是可以在模块头中进行描述的控制指令。注意，这些控制指令只能在模块头中进行描述。

模块头也可以省略。

表 2-1　可以在模块头中描述的指令

可以描述的项目	说明	在本手册中的章/节
与汇编器选项具有相同功能的控制指令	- PROCESSOR - XREF/NOXREF - DEBUG/NODEBUG, DEBUGA/NODEBUGA - TITLE - SYMLIST/NOSYMLIST - FORMFEED/NOFORMFEED - WIDTH - LENGTH - TAB - KANJICODE	第4章　控制指令
由高级程序例如C语言编译器输出的特殊控制指令	- TOL_INF - DGS - DGL	

2.1.2　模块体

不能在模块体中进行描述以下指令：

- 与汇编器选项具有相同功能的控制指令

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

用户可以使用与每个段相应的指令来定义以下四个段：

(1)　代码段

必须以CSEG指令来定义。

(2)　数据段

必须以DSEG指令来定义。

(3)　位段

必须以BSEG指令来定义。

(4)　绝对段

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

模块体可以以任何段的组合来构成。
然而，在代码区段前应该定义一个数据区段和一个位区段。

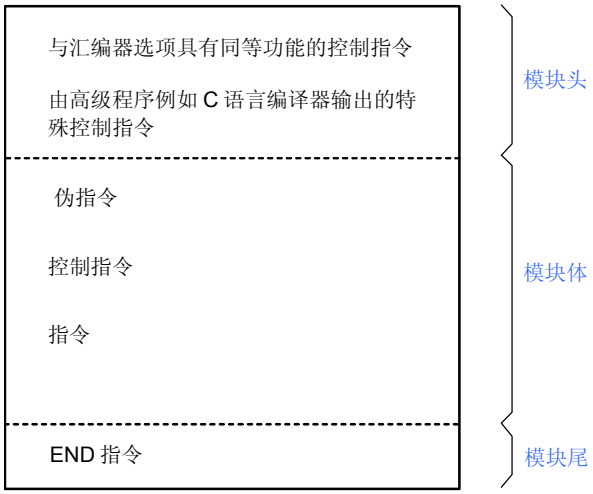
2.1.3　模块尾

模块尾表明源模块结束。在这部分中必须有END指令。
如果任何不同于注释，空格，制表符或换行符代码的内容接在EDN指令后进行描述，那么编译程序将输出一个警告消息并忽略在END指令后描述的字符。

2.1.4　源程序的总体结构

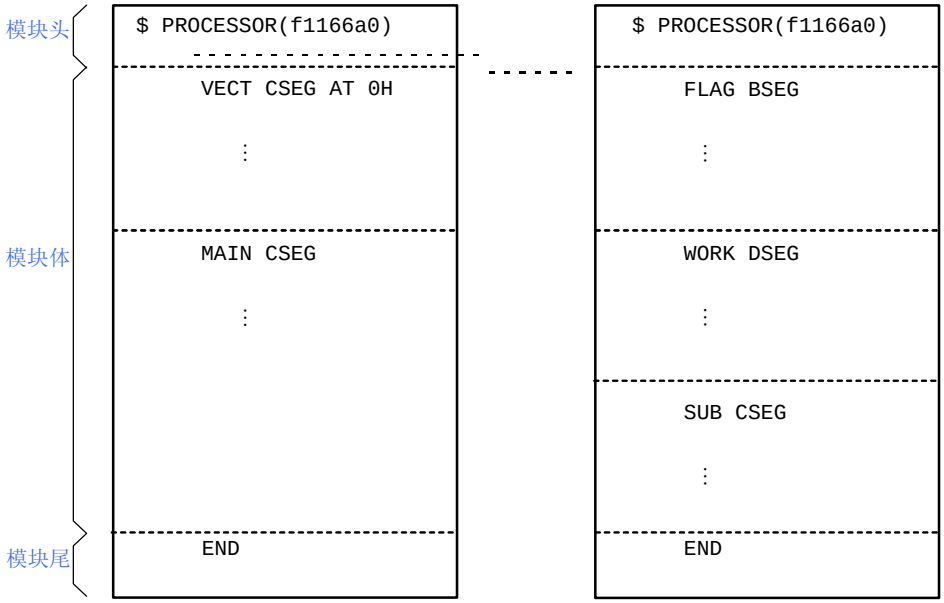
源模块（源程序）的总体结构如下所示。

图 2-2　源模块的总体结构



简单的源模块结构示例显示在图2-3中。

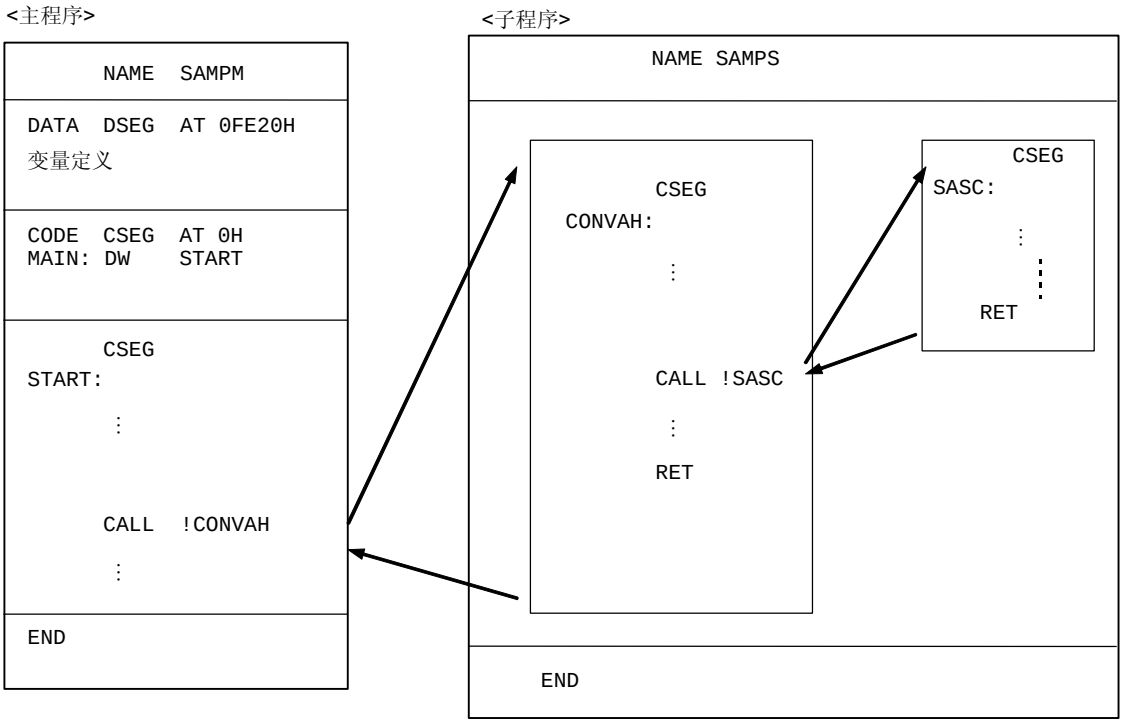
图 2-3　源模块结构示例



2.1.5　源程序描述示例

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

图 2-4　样例程序的结构



<主程序>

```
    NAME SAMPM                          ; (1)
; ************************************************
; HEX -> ASCII Conversion Program
; main-routine
; ************************************************
PUBLIC      MAIN ,      START           ; (2)
EXTRN       CONVAH                      ; (3)
EXTRN       _@STBEG                     ; (4) <-- Error
DATA        DSEG AT 0FFE20H             ; (5)
HDTSA :     DS          1
STASC :     DS          2
CODE        CSEG AT 0H                  ; (6)
MAIN :      DW          START
            CSEG                        ; (7)
START :
    ; 芯片初始化
    MOVW    SP , #_@STBEG
    MOV     HDTSA , #1AH
    MOVW    HL , #LOWW ( HDTSA )            ; set hex 2-code data in HL registor
    CALL    !CONVAH                         ; convert ASCII <- HEX
                                            ; output BC-register <- ASCII code
    MOVW    DE , #LOWW ( STASC )            ; set DE <- store ASCII code table
    MOV     A , B
    MOV     [ DE ] , A
    INCW    DE
    MOV     A , C
    MOV     [ DE ] , A
    BR      $$
    END                             ; (8)
```

(1)　模块名称的声明。

(2)　被另一个模块中作为而外部引用符号引用的符号声明。

(3)　在另一个模块中定义为一个外部引用符号的符号声明。

(4)　从连接器的”-S”选项中生成一个外部引用符号作为堆栈解决符号的声明（如果连接时没有指定”-S”选项，将会产生一个错误）。

(5)　数据段开始声明（位于saddr中）

(6)　代码段开始声明（设置为一个从地址0H开始的绝对区段）

(7)　代码段开始声明（表示绝对段结束）

(8)　模块结束声明

```
      NAME    SAMPS                   ; (1)
; *************************************************************
; HEX -> ASCII Conversion Program
; sub-routine
;
; input condition :         ( HL ) <- hex 2 code
; output condition :        BC-register <- ASCII 2 code
; *************************************************************
PUBLIC      CONVAH                    ; (2)
            CSEG                      ; (3)
CONVAH :
            XOR     A , A
            ROL4    [ HL ]                      ; hex upper code load (4)
            CALL    !SASC
            MOV     B , A                       ; store result

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

; *************************************************************
; subroutine convert ASCII code
;
; input       Acc ( lower 4bits ) <- hex code
; output      Acc <- ASCII code
; *************************************************************
SASC :
            CMP     A , #0AH                    ; check hex code > 9
            BC      $SASC1
            ADD     A , #07H                    ; bias ( +7H )
SASC1 :
            ADD     A , #30H                    ; bias ( +30H )
            RET
            END                       ; (5)
```

(1)　模块名称的声明

(2)　被另一个模块中作为而外部引用符号引用的符号声明。

(3)　代码段开始声明

(4)　由于ROL4指令是针对78K0系列的，而78K0R系列编译器不支持该指令，需要指定汇编选项（-compati）关于汇编选项（-compati），敬请参阅RA78K0R系列汇编器操作篇用户手册。

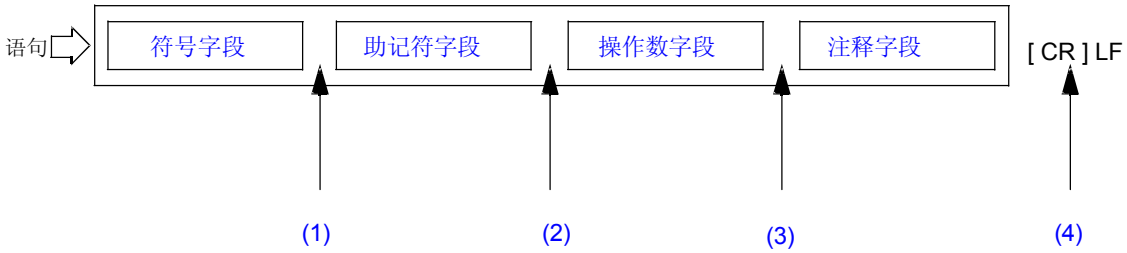
(5)　声明模块的结束

2.2　源程序的描述格式

2.2.1　语句结构

源程序由语句所组成。
每个语句包含下图显示的四个字段。

图 2-5　组成语句的字段



(1)　符号字段和助记符字段间必须用冒号(:)，或一个或更多的空格或制表符隔开（它根据在助记符字段描述的指令决定来决定是否使用冒号或空格）。
(2)　助记符字段和操作数字段间必须用一个或更多的空格或制表符隔开。根据在助记符字段中所描述的指令，可能不需要操作符字段。
(3)　如果使用注释字段，则必须在前面加上分号(;)。
(4)　每一行必须用一个LF码来定界（在LF码前可能会存在一个CR码）。

- 一个语句必须在一行内进行描述。每行最多可以描述 2048 个字符（包括 CR 和 LF）。
  每个 TAB 或独立的 CR 都被计作一个字符。如果描述了 2049 个或更多的字符，将会输出一条警告消息且任何 2049 之后的任何字符都会被忽略。然而，2049 个或更多的字符将被输出到汇编列表中。

- 独立的 CR 不会被输出到汇编列表中。

- 以下的行也可以被描述：

(1)　虚拟行（没有语句描述的行）

(2)　仅由符号字段组成的行

(3)　仅由注释字段组成的行

2.2.2　字符集

可以在一个源文件中进行描述的字符被分为以下三类：

- 语言字符
- 字符数据
- 注释字符

(1)　语言字符

语言字符是用于在源程序中描述指令的字符。
它包括字母，数字和特殊字符。

表 2-2　字母数字式字符

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

表 2-3　特殊字符

字符	名称	主要用途	
?	问号	相当于字母字符的符号	
@	圈 a	相当于字母字符的符号	
_	下划线	相当于字母字符的符号	
	空格	每个字段的分隔	分隔符号
HT (09H)	制表码	相当于空格的字符	
,	逗号	操作数的分隔	
:	冒号	标签的分隔	
;	分号	表示注释字段开始的符号	
CR (0DH)	回车码	表示行结束的符号（在汇编器中忽略）	
LF (0AH)	换行码	表示行结束的符号	

表 2-3　特殊字符

字符	名称	主要用途	
+	加号	加法运算符或正号	汇编器运算符
-	减号	减法运算符或负号	
*	星号	乘法运算符	
/	斜杠	除法运算符	
.	句点	字节的位说明符	
(,)	左，右圆括号	指定要执行的算术运算的顺序的符号	
< , >	不等号	关系运算符	
=	等号	关系运算符	
'	单引号	表示字符常量开始或结束的符号 表示一个完整的宏参数的符号	
$	美元符号	表示位置计数器的符号 表示相当于汇编器选项的控制指令的开始 指定相对寻址的符号	
&	和号	连接符号（在宏体中使用）	
#	井号	指定立即寻址的符号	
!	感叹号	指定绝对寻址的符号	
[ ]	方括号	指定间接寻址的符号	

(2)　字符数据

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

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

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

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

(3)　注释字符

“注释字符“指的是用于描述一个注释语句的字符。

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

2.2.3　符号字段

符号是在符号字段中进行描述的。名词“符号”指的是用于数字数据或地址的名称。
通过使用符号，源程序的内容可以变得易于理解。

(1)　符号类型

根据符号的用途和定义方法，符号被分成在下表中所显示的类型。

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

注意　四种类型的符号，名称，标签，区段名称和宏名称，都可以在符号字段中描述。

(2)　符号描述的约定

所有符号必须依照以下规则进行描述：

(a)　符号必须由字母数字字符和可以用作字母字符的特殊字符(?, @, and _)来组成。
　　数字字符0至9都不能用作符号的首字符。
(b)　每个符号中包含的字符不能多于256个。超过最大符号长度的字符将被忽略。
(c)　保留字不能用作符号。保留字显示在表A-2中。
(d)　同一符号不能被多次定义
　　然而，以SET指令定义的名称可以用SET指令重新定义。
(e)　编译程序可以区别小写和大写字符。
(f)　当在符号字须中描述一个标签时，":"（冒号）必须在标签后立即进行描述。

<正确的符号描述示例>

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

<不正确的符号描述示例>

```
1ABC        EQU       3         ; 数字字符不能用作符号的首字符。
LAB         MOV       A , R0    ; “LAB”是一个标签且必须使用冒号(:)来和助记符字段隔开。
FLAG :      EQU       10H       ; 冒号(:)在名字中不是必需的。
```

<过长符号的示例>

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

<仅由一个符号组成的语句的示例>

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

(3)　关于符号的一些注意事项

符号“??RAnnnn (n = 0000 到 FFFF)”是每次在宏体内开发一个局部符号时都将会被汇编器自动替换的符号。须注意不要对该符号定义两次。

当段名不是通过段定义指令来指定时，编译程序将会自动生成一个段名。这些段名显示在下表中。

复制段名定义会产生一个错误。

段名	指令	重定位属性
?An (n = 0000 到 FFFF)	ORG伪指令	(无)
?CSEG	CSEG伪指令	UNIT
?CSEGUP		UNITP
?CSEGT0		CALLT0
?CSEGFX		FIXED
?CSEGSI		SECUR_ID
?CSEGB		BASE
?CSEGP64		PAGE64KP
?CSEGU64		UNIT64KP
?CSEGMIP		MIRRORP
?CSEGOB0		OPT_BYTE
?DSEG	DSEG 伪指令	UNIT
?DSEGUP		UNITP
?DSEGS		SADDR
?DSEGSP		SADDRP
?DSEGBP		BASEP
?DSEGP64		PAGE64KP
?DSEGU64		UNIT64KP
?BSEG	BSEG 伪指令	UNIT

(4)　符号属性

所有名字和标签都有一个值和一个属性。

值就是定义的数字数据或地址数据本身的值。

区段名称，模块名称和宏名称不含有值。

符号的属性被称作符号属性且必须是显示在下表中的八种类型之一。

属性类型	类别	值
数字	- 被赋于数字常量的名称 - 用EXTRN指令定义的符号 - 数字常量	十进制表示：0 到 1,048,575 十六进制表示：00000H至FFFFFH（无符号型）
地址	- 定义为标签的符号 - 用EQU和SET指令定义为标签的名称	十进制表示：0 到 1,048,575 十六进制表示：00000H至FFFFFH
位	- 定义为位值的名称 - BSEG内的名称 - 用EXTBIT指令定义的符号	0H 至 FFFFFH
SFR	使用EQU伪指令定义的名称，和SFR一样。	SFR区域
SFRP	使用EQU伪指令定义的名称，和SFR一样。	这些属性类型不具有值
CSEG	用CSEG指令定义的段名	
DSEG	用DSEG指令定义的段名	
BSEG	用BSEG指令定义的段名	
模块	用NAME指令定义的模块名（如果没有定义的模块名从输入源文件名中的主要名称中创建）	
宏	用MACRO伪指令定义的宏名	

<例>

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

2.2.4　助记符字段

助记指令，伪指令或宏引用在助记符字段中进行描述。

使用指令或需要一个或更多的操作数的伪指令时，助记符和操作数字段必须用一个或更多的空格或制表符来隔开。

然而，使用以“#”, “$” ,“!”, 或“[”开头的指令的第一操作数时，即使在助记符字段和第一操作数字段间不存在任何内容，汇编器仍将正常执行。

<正确描述的示例>

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

<不正确描述的示例>

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

2.2.5　操作数字段

用于执行指令，伪指令或宏引用的数据(操作数)在操作数字段中进行描述。

根据指令或伪指令，在操作数字段中将不需要操作数，而在操作数字段中必须描述两个或更多的操作数。

当描述两个或更多的操作数时，需用逗号(,)为每个操作数来定界。

以下类型的数据可以在操作数字段中进行描述：

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

所需操作数的尺寸和属性根据指令或伪指令可能会不同。关于操作数的尺寸和属性，请参考“2.15 操作数的特征”。

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

关于每种可以在操作数字段使用的数据类型，详细描述如下

(1)　　常量

常量是一个固定的值或数据项目，也可以被称为立即数。

常量被分为数字常量和字符串常量。

(a)　　数字常量

二进制数，八进制数，十进制数或十六进制数可以被描述为数字常量。

每个数字常量类型的表示方法都显示在下表中。

数字常量将会被处理为无符号的32位数据。

值的范围： 0 ≤ n ≤ 0FFFFFFFFH

描述一个负值时，使用负号运算符。

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

(b)　　字符串常量

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

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

串常量的长度是0到2个字符。

为了将单引号本身用作一个串常量，单引号必须被连续输入两次。

<字符串常量描述示例>

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

(2)　　字符串

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

<字符串的应用示例>

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

(3)　寄存器名

以下寄存器可以在操作数字段中进行描述。

- 通用寄存器
- 通用寄存器对
- 特殊功能寄存器

通用寄存器和通用寄存器对可以以它们的绝对名(R0到R7　和　RP0到RP3)来描述，也可以用它们的功能名(X, A, B, C, D, E, H, L, AX, BC, DE, HL)来描述。

可以在操作数字段中描述的寄存器名根据指令的类型可能会不相同。关于描述每个寄存器名描述方法的详细说明，请参考开发中的软件的各个设备的用户手册。

(4)　特殊字符

下表显示了可以在操作数字段中描述的特殊字符。

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

<特殊字符的应用示例>

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

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

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

(5)　段定义指令的重定位属性

重定位属性可以在操作数字段中进行描述。

关于重定位属性的详细说明，请参考 “3.2　段定义指令”。

(6)　符号

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

<符号的应用示例>

```
VALUE   EQU     1234H
        MOV     A , #VALUE              ; 该描述可以写作“MOV A, #1234H”。
```

(7)　表达式

表达式是与运算符连接的常量，$（表示一个字单元地址），名称或标签。.

表达式可以在表示数字值的地方描述为指令操作数。

关于表达式和运算符的详细资料，请参考“2.3　表达式和运算符”。

<表达式的示例>

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

在该例中，“TEN-5H”是一个表达式。

在该表达式中，名称和数字常量与一个-（减号）运算符相连。表达式的值为“BH”。

因此，该描述可以被写作“MOV A, #0BH”。

(8)　位项

位项可以通过位位置说明符来获取。关于位项的详细说明，请参考2.14　字节中位的位置说明符。

<位项的示例>

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

2.2.6　注释字段

在注释字段中，注释或备注可以在输入一个分号(;)后进行描述。
注释字段是从分号到该行或EOF的换行码。
通过在注释字段中描述一个注释语句，可以创建一个易于理解的源程序。
在注释字段中的注释语句不受汇编器运算（也就是转换为机器语言）的影响，但将会被原封不动地输出到汇编列表中。
可以在注释字段中进行描述的字符是那些显示在“2.2.2　字符集”中的字符。

<注释的示例>

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

            PUBLIC   MAIN , START
            EXTRN    CONVAH
            EXTRN    @STBEG
  DATA      DSEG     saddr
  HDTSA :   DS       1
  STASC :   DS       2

  CODE      CSEG     AT 0H
  MAIN :    DW       START

            CSEG
  START :
                                        ; chip initialize    仅由注释字段组成的行
            MOVW     SP , #_@STBEG
            MOV      HDTSA , #1AH
            MOVW     HL , #HDTSA        ; set hex 2-code data in HL register
            CALL     !CONVAH            ; convert ASCII <- HEX
                                        ; output BC-register <- ASCII code
            MOVW     DE , #STASC        ; set DE <- store ASCII code table
            MOV      A , B
            MOV      [ DE ] , A
            INCW     DE
            MOV      A , C
            MOV      [ DE ] , A
            BR       $$

            END
```

仅由注释字段组成的行

在注释字段中描述注释的行

2.3　表达式和运算符

表达式是与符号，常量，字单元地址（用$表示）中的一个相组合的运算符，或是运算符的组合。
表达式中不同于运算符的元素被称为项，并且按它们的描述顺序从左到右依次被称为第一项，第二项等。
在表2-4中所显示的运算符的类型是可用的，且它们在计算时的优先级已经按表2-5中所显示的被预先规定。
圆括号"()"用于更改执行计算时顺序。

< 例 >

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

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

表 2-4　运算符的类型

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

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

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

表 2-5　运算符的优先级顺序

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

依照以下规则来执行表达式的运算：

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

(2)　圆括号内的表达式在圆括号外的表达式之前执行。
(3)　允许在两个或更多的一元运算符间的运算。
例：　1 = --1 == 1
　　-1 = -+1 = -1
(4)　表达式在32位内进行无符号数的计算。
如果由于表达式超过32位而在运算时发生溢出，溢出值将被忽略。
(5)　如果一个常量超过32位(0FFFFH)，将会产生一个错误，且计算结果的值将被认为是0。
(6)　在除法中，结果中的小数部分将被清除。如果除数为0，则将会发生一个错误，且结果将会是0。
(7)　负值要用2的补码形式表示。
(8)　在汇编过程中认为外部引用符号的值为零（该值的赋入是在连接时决定）。
(9)　在操作区段描述的表达式结果必须满足指令的要求。
如果在需要8位操作数的指令中使用了重定位表达式或使用外部引用的表达式，目标将从低8位的值获得，并要求重定位信息按照16位输出。然后连接器检测已经被确定的值是否可以用8位来表示，如果溢出则在连接时发生错误。
如果描述中出现绝对表达式，汇编器决定该表达式的值，并检查该值是否在要求范围内。
比如，MOV指令要求8位操作数，所以使用的值应该在0H到0FFH之类。

<正确表达的示例>

```
MOV     A , #'2*' AND 0FH
MOV     A , #4 * 8 * 8 - 1
```

<错误表达的示例>

```
MOV     A , #'2*.
MOV     A , #4 * 8 * 8
```

<求值示例>

表达式	评估值
2 + 4 * 5	22
(2 + 3) * 4	20
10 / 4	2
0 - 1	0FFFFFFFFH
-1 > 1	00H (False)
EXT[注] + 1	1

注　EXT：外部引用符号

2.4　算术运算符

可以使用的算术运算符描述如下：

- +
- -
- *
- /
- MOD
- + sign
- - sign

+　运算符

[功能]

- 返回表达式第一项和第二项的值的和。

[应用示例]

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

[说明]

- BR指令产生一个转移到“当前字单元地址加6”，也就是地址跳转到“100H+6H = 106H”。
  因此，上例中的(a)也可以被描述为：START: BR !106H。

- **运算符**

[功能]

- 返回从第一项的值中减去第二项的值所得到的结果。

[应用示例]

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

[说明]

- BR指令产生一个转移到“赋值到BACK减6的地址”，也就是产生一个转移到“100H-6H=0FAH”。
  因此，上例中的(b)也可以被描述为：BACK: BR !0FAH。

***　运算符**

[功能]

- 返回表达式的第一和第二项的值的乘积。

[应用示例]

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

[说明]

- 使用EQU指令将"TEN"的值定义为"10H"。
  "#"表示立即数。表达式"TEN*3"与"10H*3"相同，并且返回值"30H"。
  因此，在上述表达式中的(c)也可以被描述为：MOV A,#30H。

/ 运算符

[功能]

- 将表达式第一项的值除以表达式第二项的值，并返回到结果的整数部分。

结果的小数部分将会被清除。如果除法运算中的除数（第二项）为0，则将产生一个错误

[应用示例]

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

[说明]

- 除法运算“256/50”的结果是5余6。
  运算符返回到值”5”，”5”是除法运算的结果的整数部分。
  因此，上述表达式中的(d)也可以被描述为：MOV A, #5

MOD

[功能]

- 获取表达式第一项的值除以表达式第二项的值所得结果中的余数。
  如果除数（第二项）为0，则将产生一个错误。
  在MOD运算符的前后均需有一个空格。

[应用示例]

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

[说明]

- 除法运算“256/50”的结果是5余6。
  MOD运算符返回余数6。

  因此，上述表达式中的(e)也可以描述为：MOV A,#6。

+ 号

[功能]

- 原封不动地返回表达式中项的值。

[应用示例]

```
        FIVE    EQU     +5
```

[说明]

- 原封不动地返回项的值”5”。

  使用EQU指令定义变量”FIVE”的值为”5”。

- 号

[功能]

- 以二进制补码返回表达式中项的值。

[应用示例]

```
NO    EQU      -1
```

[说明]

- -1变成1的二进制补码。

  二进制数0000　0000　0000　0001的二进制补码变成: 1111 1111 1111 1111

  因此，使用EQU指令定义变量”NO”的值为“0FFFFH”。

2.5　逻辑运算符

可以使用的逻辑运算符描述如下：

- NOT
- AND
- OR
- XOR

NOT

[功能]

- 将表达式中项的值按位取反，并将结果返回。
  在NOT运算符和项之间需有一个空格。

[应用示例]

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

[说明]

- 按以下方式在”3H”上执行逻辑非：

NOT)	0000	0000	0000	0000	0000	0000	0011
	1111	1111	1111	1111	1111	1111	1100

返回0FFFFFFFCH。

因此，(a)可以被描述为：MOVW　AX, #0FFFFFFFCH

AND

[功能]

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

  在AND运算符的前后均需有一个空格。

[应用示例]

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

[说明]

- 按以下方式在 “6FAH” 和 “0FH” 两个值之间执行AND运算：

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

返回结果0AH。因此，上述表达式中的(b)也可以被描述为：MOV A, #0AH

OR

[功能]

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

在OR运算符的前后均需有一个空格。

[应用示例]

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

[说明]

- 按以下方式在“0AH”和“1101B”两个值之间执行OR运算：

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

返回结果0FH。

因此，上述表达式中的(c)也可以被描述为：MOV A, #0FH

XOR

[功能]

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

[应用示例]

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

[说明]

- 按以下方式在“9AH”和“9DH”两个值之间执行XOR运算：

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

返回结果7H。

上述表达式中的(d)也可以被描述为：MOV A, #7H

2.6　关系运算符

可以使用的关系运算符描述如下：

- EQ (=)
- NE (< >)
- GT (>)
- GE (>=)
- LT (<)
- LE (<=)

EQ （=）

[功能]

- 如果表达式中第一项的值等于第二项的值则返回0FFH（真），如果两项的值不相等则返回00H（假）。

  在EQ运算符的前后均需有一个空格。

[应用示例]

```
A1      EQU     12C4H
A2      EQU     12C0H

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

[说明]

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

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

NE（< >）

[功能]

- 如果表达式中第一项的值不等于第二项的值则返回0FFH（真），如果两项的值相等则返回00H（假）。

  在NE运算符的前后均需有一个空格。

[应用示例]

```
A1      EQU     5678H
A2      EQU     5670H

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

[说明]

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

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

GT （>）

[功能]

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

在GT运算符的前后均需有一个空格。

[应用示例]

```
A1      EQU     1023H
A2      EQU     1013H

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

[说明]

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

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

GE（>=）

[功能]

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

值则返回00H（假）。

- 在GE运算符的前后均需有一个空格。

[应用示例]

```
A1      EQU     2037H
A2      EQU     2015H

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

[说明]

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

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

LT（<）

[功能]

- 如果表达式中第一项的值小于第二项的值则返回0FFH（真），如果表达式中第一项的值等于或大于第二项的值则返回00H（假）。
  在LT运算符的前后均需有一个空格。

[应用示例]

```
A1      EQU     1000H
A2      EQU     1020H

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

[说明]

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

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

LE（<=）

[功能]]

- 如果表达式中第一项的值小于或等于第二项的值则返回0FFH（真），如果表达式中第一项的值大于第二项的值则返回00H（假）。
  在LE运算符的前后均需有一个空格。

[应用示例]

```
A1      EQU     103AH
A2      EQU     1040H

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

[说明]

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

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

2.7　循环移动运算符

可以使用的循环移动运算符描述如下：

- SHR
- SHL

SHR

[功能]

- 返回的值是将表达式中第一项的值向右移动，移动的位数由第二项的值所指定。相当于循环移动过程中，指定位数个0被移动到字节的高位中。

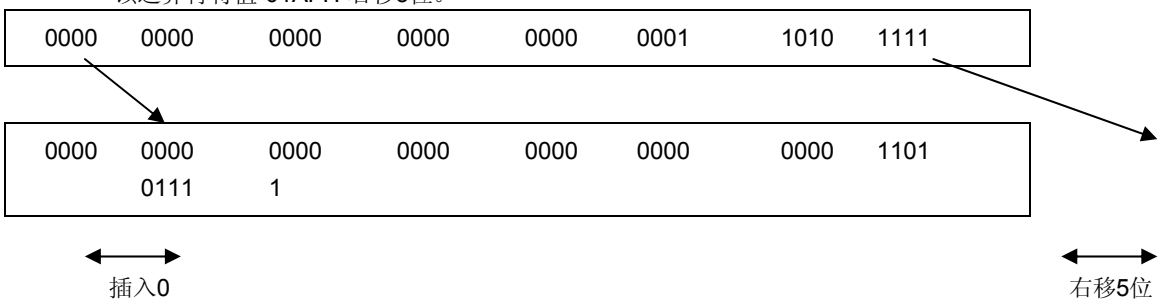
在SHR运算符的前后均需有一个空格。

[应用示例]

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

[说明]

- 该运算符将值“01AFH”右移5位。



返回值“000DH”。
因此，上例中的(a)也可以描述为：MOV A, #0DH

SHL

[功能]

- 返回的值是将表达式中第一项的值向左移动，移动的位数由第二项的值所指定。相当于循环移动过程中，指定位数个0被移动到字节的低位中。

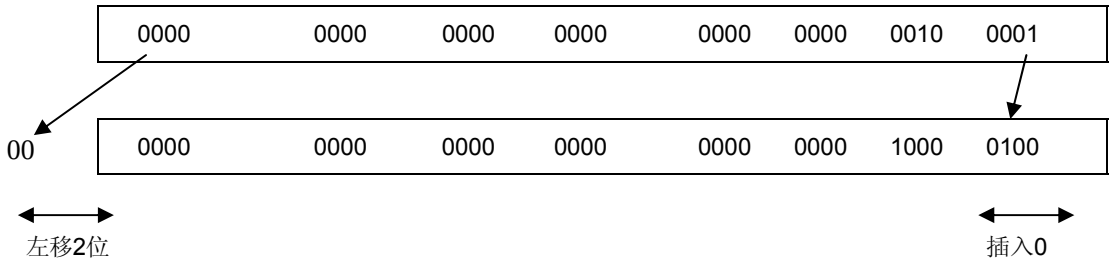
在SHL运算符的前后均需有一个空格。

[应用示例]

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

[说明]

- 该运算符将值“21H”左移2位。



返回值“84H”。
因此，上例中的(b)也可以描述为：MOV A, #84H

2.8　字节分离运算符

可以使用的字节分离运算符描述如下：

- HIGH
- LOW

HIGH

[功能]

- 返回项的高位中的8位值。

在HIGH运算符和项之间需有一个空格。

[应用示例]

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

[说明]

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

因此，上例中的(a)也可以被描述为：MOV A, #12H

LOW

[功能]

- 返回项的低位中的8位值。
  在LOW运算符和项之间需有一个空格。

[应用示例]

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

[说明]

- 通过执行MOV指令，该运算符可以返回表达式“1234H”的低位中的8位值“34H”。
  因此，上例中的(b)也可以被描述为：MOV A, #34H

2.9　字分离运算符

可以使用的字分离运算符描述如下：

- HIGHW
- LOWW

HIGHW

[功能]

- 返回项的高位中的16位值。

在HIGHW运算符和之间需有一个空格。

[应用示例]

```
MOVW    AX , #HIGHW   12345678H    ; (a)
MOV     ES , #HIGHW   LAB          ; (b)
MOVW    AX , ES:!LAB
```

[说明]

- 通过执行MOVW指令，该运算符可以返回表达式“12345678H”的高位中的16位值“1234H”。
因此，上例中的(a)也可以被描述为：MOVW　AX, #1234H

- 通过执行（b）行的MOV指令，标签LAB的高端地址就被赋给ES寄存器。

[备注]

- 对某个SFR名称执行HIGHW操作，使用下列描述的任何一种方法：

```
HIGHW Δ SFR 名称
```

或

```
HIGHW[Δ] ([Δ]SFR 名称[Δ])
```

操作中获得的结果是绝对NUMBER属性的操作数。
不能对SFR名称进行其他操作。

<例>

```
Symbol field    Mnemonic field    Operand field
                MOVW              RP0 , #HIGHW PM0
                MOVW              RP1 , #HIGHW PM1 + 1H      ; 等价于#( HIGHW PM1 ) + 1
                MOVW              RP1 , #HIGHW ( PM1 + 1H )  ; 会返回一个错误
                                                             ; 因为对 SFR 名称指定了
                                                             ; HIGH, LOW, HIGHW, LOWW
                                                             ; 之外的操作数。
```

LOWW

[功能]

- 返回项的低位中的16位值。
  在LOWW运算符和项之间需有一个空格。

[应用示例]

```
MOVW    A , #LOWW 12345678H        ; ( a )
```

[说明]

- 通过执行MOVW指令，该运算符可以返回表达式“12345678H”的低16位值“1234H”。
  因此，上例中的(a)也可以被描述为：MOVW　AX, #5678H

[备注]

- 对某个SFR名称执行LOWW操作，使用下列描述的任何一种方法：

```
LOWW Δ SFR 名称
```

或

```
LOWW[Δ] ([Δ]SFR 名称[Δ])
```

操作中获得的结果是绝对NUMBER属性的操作数。
不能对SFR名称进行其他操作。

<例>

```
Symbol field    Mnemonic field    Operand field
                MOVW              RP0 , # LOWW PM0
                MOVW              RP1 , # LOWW PM1 + 1H      ; 等价于#( LOWW PM1 ) + 1
                MOVW              RP1 , # LOWW ( PM1 + 1H )  ; 会返回一个错误
                                                             ; 因为对 SFR 名称指定了
                                                             ; HIGH, LOW, HIGHW, LOWW
                                                             ; 之外的操作数。
```

2.10 特殊运算符

可以使用的特殊运算符描述如下：

- DATAPOS
- BITPOS
- MASK

DATAPOS

[功能]

- 返回位符号的地址部分（字节地址）。

[应用示例]

```
SYM     EQU     0FE68H.6
        MOV     A , !DATAPOS SYM            ; ( a )
```

[说明]

- EQU指令将 “SYM” 的值定义为0FE68H.6。
  “DATAPOS SYM” 表示 “DATAPOS　0FE68H.6”，并返回“0FE68H”。
  因此，在上例中的(a)也可以被描述为：MOV A, !0FE68H

BITPOS

[功能]

- 返回位符号的位部分（位位置）。

[应用示例]

```
SYM     EQU     0FE68H.6
        CLR1    [ HL ] .BITPOS SYM          ; ( b )
```

[说明]

- EQU指令将 “SYM” 的值定义为0FE68H.6。
  “BITPOS.SYM” 表示 “BITPOS 0FE68H.6”，并返回”6”。
  CLR1指令将[HL].6清0。

MASK

[功能]

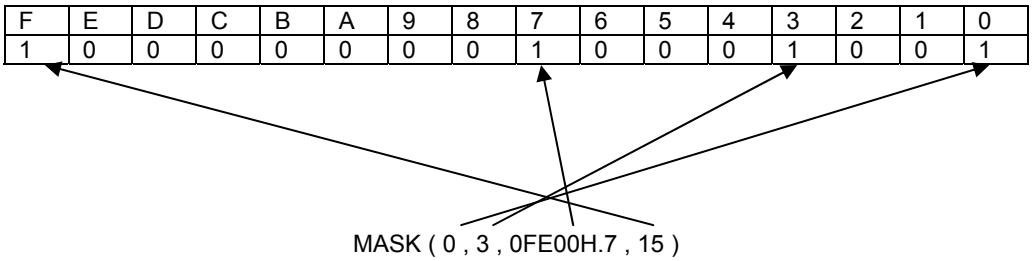
- 返回一个16位值，值中的指定位位置为1且其他位位置均被设为0。

[应用示例]

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

[说明]

- MOVW指令返回值“8089H”。



2.11 其他运算符

可以使用的其他运算符描述如下：

- ( )

( )

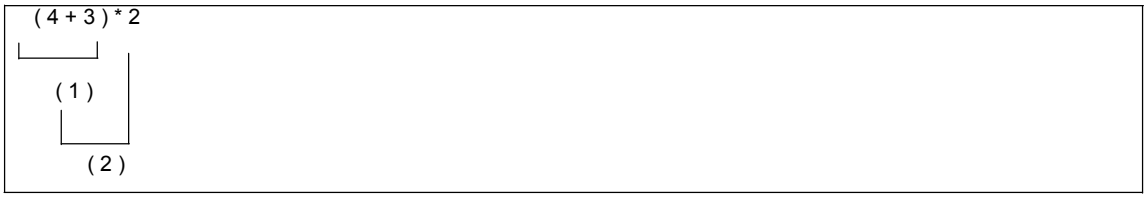
[功能]

- 使得圆括号中的运算比圆括号外的运算优先被执行。
  该运算符用于改变其他运算符的优先级别。
  如果圆括号被嵌入多个级别中，最内部圆括号中的表达式将先被计算。

[应用示例]

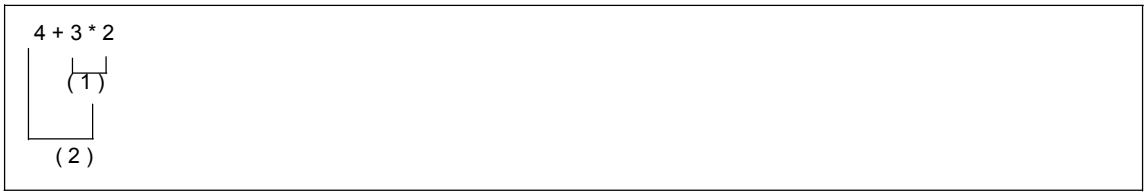
```
MOV     A , # ( 4 + 3 ) * 2
```

[说明]



按表达式<1> 和<2>的顺序来执行计算，且返回值”14”来作为结果。

如果没有使用圆括号，



将按以上显示的顺序来执行计算，且返回会下”10”来作为结果。
关于运算符的优先级别，参见表2-5。

2.12 运算限制

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

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

2.12.1 运算符和重定位属性

正如先前所提到的，构成表达式的每个项都具有一个重定位属性和一个符号属性。

当根据项的重定位属性来分类时，项可以被分为三类：绝对项，浮动项和外部引用项。

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

表 2-6　重定位属性的类型

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

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

运算符的类型和可以在上面执行每个运算符的项的组合显示在表2-12中。

表 2-7　按照重定位属性区分的项和操作符的组合

操作符类型	项的重定位属性			
	X: ABS Y: ABS	X: ABS Y: REL	X: REL Y: ABS	X: REL Y: REL
X + Y	A	R	R	—
X – Y	A	—	R	A 注1
X * Y	A	—	—	—
X / Y	A	—	—	—
X MOD Y	A	—	—	—
X SHL Y	A	—	—	—
X SHR Y	A	—	—	—
X EQ Y	A	—	—	A 注1
X LT Y	A	—	—	A 注1
X LE Y	A	—	—	A 注1
X GT Y	A	—	—	A 注1
X GE Y	A	—	—	A 注1
X NE Y	A	—	—	A 注1
X AND Y	A	—	—	—
X OR Y	A	—	—	—
X XOR Y	A	—	—	—
NOT X	A	A	—	—
+ X	A	A	R	R
– X	A	A	—	—
HIGH　X	A	A	R 注2	R 注2
LOW　X	A	A	R 注2	R 注2
HIGHW　X	A	A	R 注2	R 注2
LOWW　X	A	A	R 注2	R 注2
MASK　(X)	A	A	—	—
DATAPOS　X.Y	A	—	—	—
BITPOS　X.Y	A	—	—	—
MASK　(X.Y)	A	—	—	—
DATAPOS　X	A	A	R	R
BITPOS　X	A	A	A	A

<表格说明>

ABS :　　绝对项

REL :　　浮动项
A :　　运算的结果变成一个绝对项。
R :　　运算的结果变成一个浮动项。

- :　　不能执行运算。

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

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

以下五个运算符可以在外部引用项上执行：“+”，“-”，“HIGH”，“LOW”，和 “BANKNUM”（然而，须注意只有一个外

部引用项可以在表达式中进行描述）。
运算符类型和可以在上面执行运算符的外部引用项的组合依照表2-13中的重定位属性进行分类。

表 2-8　用重定位属性组合项和操作符 (外部引用项)

操作符类型	项的重定位属性				
	X:ABS Y:EXT	X:EXT Y:ABS	X:REL Y:EXT	X:EXT Y:REL	X:EXT Y:EXT
X + Y	E	E	—	—	—
X – Y	—	E	—	—	—
+ X	A	E	R	E	E
HIGH X	A	E 注1	R 注2	E 注1	E 注1
LOW X	A	E 注1	R 注2	E 注1	E 注1
HIGHW X	A	E 注1	R 注2	E 注1	E 注1
LOWW X	A	E 注1	R 注2	E 注1	E 注1
MASK (X)	A	—	—	—	—
DATAPOS X.Y	—	—	—	—	—
BITPOS X.Y	—	—	—	—	—
MASK (X.Y)	—	—	—	—	—
DATAPOS X	A	E	R	E	E
BITPOS X	A	E	A	E	E

<表格说明>

ABS :　　绝对项

EXT :　　外部引用项

REL :　　浮动项

A :　　运算的结果变成一个绝对项。
E :　　运算的结果变成一个外部引用项。
R :　　运算的结果变成一个浮动项。

- :　　不能执行运算。

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

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

2.12.2 运算符和符号属性

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

在运算中的符号属性的类型和适用于每个属性的项显示在表2-14中。

表 2-9　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

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

表 2-10　用符号属性组合项和操作符

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

＜表格说明＞
ADDRESS :　地址项
NUMBER :　数字项
A :　运算的结果变成一个地址项。
N :　运算的结果变成一个数字项。
- :　不能执行运算。

2.12.3 如果检查运算中的限制

此处显示了每项中根据重定位属性和符号属性的运算示例。

< 例 >

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

这里假设"TABLE"是一个在浮动码段中定义的标签。

[运算符和重定位属性]

因为"TABLE+5H"是"浮动项+绝对项"，因此该运算可以用于表2-7。

运算符的类型： X + Y

项的重定位属性： X : REL , Y : ABS

从表中，你可以发现结果为R（也就是一个浮动项）。

[运算符和符号属性]

因为"TABLE+5H"是"地址项+数字项"，因此该运算可以用于表2-10。

运算符的类型： X + Y

项的符号属性： X : 地址 , Y : 数字

从表中，你可以发现结果为A（也就是一个地址项）。

2.13 绝对表达式的定义

如果在汇编时就已经对表达式求值，并决定了该表达式的值。这样的表达式被称为绝对表达式。

下列表达式被称为绝对表达式：

- 常量
- 表达式中只包含常量（常量表达式）
- 常量，使用EQU符号定义的常量表达式，或者SET符号
- 涉及上述目标某种操作的表达式。

备注　　这些符号仅支持向后引用。

2.14 位位置说明符

可以通过使用位位置说明符(.)来访问位。

。

[格式描述]

```
X [ Δ ] . [ Δ ] Y
└──────────┘
    位项
```

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

注　关于特定描述的详细说明，参见各个设备的用户手册。

[功能]

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

特定的位。

[说明]

- 位项指的是一个使用位位置说明符的表达式。
- 位位置说明符不受运算符的优先级别的影响。位位置说明符的的左侧被认为是第一项，它的右侧被认为是第二项。

- 以下限制用于第一项：
  (1) 带有数字或地址属性的表达式，能够进行位访问的SFR的名称或寄存器名（A）可以被描述。

  (2) 绝对值表达式在第一项中进行描述时，它必须在0FE20H到0FF1FH的范围内。

  (3) 可以描述一个外部引用符号。

- 以下限制用于第二项：
  (1) 表达式的值必须在0到7的范围内。如果超过该值范围，将会产生一个错误。

  (2) 只有带有数字属性的绝对表达式可以被描述。

  (3) 没有外部引用符号可以被描述。

[运算和重定位属性]

- 按重定位属性组合的第一和第二项显示在表2-17中。

按重定位属性组合的第一和第二项。

项组合 X	ABS	ABS	REL	REL	ABS	EXT	REL	EXT	EXT
项组合 Y	ABS	REL	ABS	REL	EXT	ABS	EXT	REL	EXT
X.Y	A	—	R	—	—	E	—	—	—

<表格说明>

ABS :　绝对项
EXT :　外部引用项
REL :　浮动项
A :　运算的结果变成一个绝对项。
E :　运算的结果变成一个外部引用项。
R :　运算的结果变成一个浮动项。
- :　不能执行运算。

[位符号的值]

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

操作数类型	符号值
A.位[注2]	1.位
PSW. 位[注2]	FFFFAH.位
sfr注1.位[注2]	FFFxxH.位[注3]
expression. 位[注2]	xxxxH. 位[注4]

注　1.　关于详细的描述，请参考各个设备的用户手册。
注　2.　位=0到7。
注　3.　FFFxxH表示sfr的地址。
注　4.　xxxxxH 表示表达式的值。

[应用示例]

```
SET1    0FFE20H.3
SET1    A.5

CLR1    P1.2
SET1    1 + 0FFE30H.3          ; 等于 0FFE31H.3
SET1    0FFE40H.4 + 2          ; 等于 0FFE40H.6
```

2.15 操作数的特征

需要一个或多个操作数的指令和伪指令不同于在所需操作数值的的尺寸和地址范围中的各个类型的指令以及在操作数的符号属性中的各个类型的指令。
例如，指令“MOV r, #byte”可以将由”byte”所表示的值转移到寄存器”r”中。假使这样，因为r是一个8位寄存器，因此要转移的数据”byte”的尺寸必须是8位或小于8位。
如果指令被描述为“MOV R0, #100H”，将产生一个汇编错误，因为指令的第二操作数“100H”的尺寸起来8位寄存器R0的容量。

因此，当描述一个操作数时，必须注意以下几点：

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

2.15.1 操作数的值的尺寸和地址范围

可以为描述为指令的操作数的数字数据，名称或标签的值的尺寸和地址范围设置一定的条件。
使用指令时，为操作数值的尺寸和地址范围所设置的条件是由每个指令的操作数表达格式来支配的。使用指令时，为操作数值的尺寸和地址范围所设置的条件是由指令的类型来支配的。

这些条件显示在下表中。

表 2-11　指令的操作数值的范围

操作数表达格式	值的范围	
字节	8-位值 0H to FFH	
字	word [B] word [C] word [BC]	数值常数和NUMBER属性符号0H至FFFFH ADDRESS属性符号 在(1)范围内或者(2)区域内 (1) F0000H至FFFFFH (2) 当MAA=0（01000H至0xxxxH）时，映射到RAM空间的区域，或当MAA=1（11000H至1xxxxH）时[注1]映射到RAM空间的区域
	ES : word [B] ES : word [C] ES : word [BC]	数值常数和NUMBER属性符号 0H至FFFFH ADDRESS属性符号 0H至FFFFH
	其它	16-位值 0H 至 FFFFH
Saddr	FFE20H至FFF1FH[注4]	
Saddrp	FFE20H至FFF1FH[注4]之间的偶地址值	
Sfr	FFF20H至FFFFFH：特殊功能寄存器符号（SFR的符号），以及数值常数和NUMBER属性符号[注5]	
Sfrp	FFF20H至FFFFFH：特殊功能寄存器符号（可以以16位单元操控的SFR的符号，必须为偶数值），以及数值常数和NUMBER属性符号[注5]	
Addr20	!!addr20	0H至FFFFFH
	$addr20	0H至FFFFFH，跟在跳转指令之后，可以跳转的范围为-80H至+7FH。
	$!addr20	0H至FFFFFH，跟在跳转指令之后，可以跳转的范围为-8000H至+7FFFH。

操作数表达格式	值的范围	
Addr16	! addr16 (BR, CALL指令)	0H 至 FFFFH （对数值常量和符号指定同样的范围）
	! addr16 (BR, CALL之外的指令)	数值常数和NUMBER属性符号[注3] 0H至FFFFH ADDRESS属性符号 在(1)范围内或者(2)区域内 (1) F0000H至FFFFFH (2) 当MAA=0（01000H至0xxxxH）时，映射到RAM空间的区域，或当MAA=1（11000H至1xxxxH）时[注1]映射到RAM空间的区域。
	ES: ! addr16	数值常数和NUMBER属性符号 0H至FFFFH ADDRESS属性符号[注3] 0H至FFFFH
	! addr16.bit	DBIT符号，SFBIT属性或SABIT属性位符号，使用EQU伪指令定义的位符号（只有操作数含有ADDRESS属性符号时）。 在(1)范围内或者(2)区域内 (1) F0000H至FFFFFH (2) 当MAA=0（01000H至0xxxxH）时，映射到RAM空间的区域，或当MAA=1（11000H至1xxxxH）时[注1]映射到RAM空间的区域 其他的位符号 0H至FFFFH
	ES: ! addr16.bit	DBIT符号，SFBIT属性或SABIT属性位符号，使用EQU伪指令定义的位符号（只有操作数含有ADDRESS属性符号时），0H至FFFFH。 其他的位符号 0H至FFFFH
Addr5	0080H至00BFH（callt指令表区域，必须为偶数值）	
Bit	3位的值 0至7	
N	2位的值 0至3	

注1　映射到RAM空间的地址范围会根据使用设备器件而有所不同，更多细节，敬请参阅所使用设备的用户手册。
注2　要将sfr或2ndsfr作为操作数，可以描述为!sfr或!2ndsfr。这些会作为!addr16的操作数输出到代码中。
注3　指定使用16位数据时只能用偶地址。
注4　为了和78K0系列兼容，FE20H至FF1FH的值可以描述为数值常数和NUMBER属性符号。
注5　对于数值常数和NUMBER属性符号，检查指定地址的SFR读/写存取限制。

[操作数的符号属性会影响addr16或字支持的值范围，原因如下]

当使用addr16或字，对操作数的符号属性的描述会影响操作数允许的范围值。原因描述如下。
关于更多细节，敬请参阅“2.2.3(4) 符号属性”。

（1） !addr16(BR, CALL之外的指令)

示例1解释了为什么addr16的操作数允许值的范围会变化，可以选择为数值常数和NUMBER属性符号和ADDRESS属性符号。

<示例1>

```
NUMBER0     EQU     0F100H              ; (a)
NUMBER1     EQU     0F102H
NUMBER2     EQU     0F103H
D0          DSEG    AT   0FF100H
ADDRESS0:   DS      1
ADDRESS1:   DS      1
ADDRESS2:   DS      1
            CSEG
            MOV     !NUMBER0 , A        ; (b)
            MOV     !0F100H , A         ; (c)
            MOV     !ADDRESS0 , A       ; (d)
```

示例1中的（a）行的代码包含NUMBER属性符号，下列解释了当该符号被用作!addr16操作数的情况。

在指令“MOV !addr16 , A”中，基于!addr16的操作数执行直接寻址。“将寄存器A中的值转移到地址0FF100H”的过程在（b）行进行。（a）行的代码包含NUMBER属性符号，它可以用（c）行内容替代，于是，!addr16使用的NUMBER属性符号NUMBER0和数值0F100H其实为地址0FF100H。

也就是说，使用NUMBER属性符号，!addr16(BR, CALL之外的指令)可以使用的值范围是“0H至FFFFH”，其实就是地址F0000H至FFFFFH。

下列解释了标签ADDRESS0被用作!addr16操作数的情况。

代码中的（d）行出现的ADDRESS0的符号值表明RAM空间的“FxxxxH至FFFFFH”，而addr16的目标范围是“0H至FFFFH”，这就导致了一个错误的发生。如果标签ADDRESS0（ADDRESS属性符号）被用作操作数，通过选择操作数的值域为“F0000H至FFFFFH”，代码可以简化。

也就是说，使用ADDRESS属性符号，!addr16(BR, CALL之外的指令)可以使用的值范围是“0H至FFFFH”，所以应该在程序这样描述操作数。

而且，将ROM区域映射到RAM区域必须用!addr16来处理。

在示例2种，MO的区段位于ROM空间，这块区域被映射到RAM空间内。当MAA = 0时，MO区段位于“01000H 至 0xxxxH”，或者当MAA = 1时，MO区段位于“11000H 至 1xxxxH”。

代码中的（e）行出现的ADDRESS0的符号值表明RAM空间内的“01000H 至 0xxxxH”或“11000H 至 1xxxxH”。这样就使得（e）行的描述可以实现，它引用了一个符号，该符号位于将要被映射的区段中，!addr16可以接受的范围是“01000H 至 0xxxxH”或“11000H 至 1xxxxH”。

也就是说，使用ADDRESS属性符号，!addr16(BR, CALL之外的指令)可以使用的值范围是“01000H 至 0xxxxH”或“11000H 至 1xxxxH”，所以应该在程序这样描述操作数。

<示例2>

```
M0          CSEG    MIRRORP
ADDRESS0:   DB      12H
ADDRESS1:   DB      34H
ADDRESS2:   DB      56H
            CSEG
            MOV     A , !ADDRESS0          ; (e)
```

（2）　Es:!addr16

示例3解释了为什么 Es:!addr16 的操作数允许值的范围会变化，可以选择为数值常数和NUMBER属性符号和ADDRESS属性符号。

<示例3>

```
DATA        CSEG    AT   12345H
ADDRESS0:   DB      12H
ADDRESS1:   DB      34H
ADDRESS2:   DB      56H
            CSEG
            MOV     ES , #HIGHW ADDRESS0        ; (f)
            MOV     A , ES:!ADDRESS0            ; (g)
```

下列解释说明了执行（f）行和（g）行的“将ADDRESS0的值转移到寄存器A中”的过程。

代码中的（g）行出现的ADDRESS0的符号值为“12345H”，而addr16的目标范围是“0H至FFFFH”，这就导致了一个错误的发生。

这就说明了如果能将ADDRESS0的范围限制在“0H至FFFFH”的话，（g）行的描述方式就可以实现，代码也能够简化。

也就是说，使用ADDRESS属性符号，ES: !addr16可以使用的值范围是“0H至FFFFH”，所以应该在程序这样描述操作数。

（3）　!addr16.bit，ES:!addr16.bit

示例4解释了为什么 !addr16.bit 和 ES:!addr16.bit 的操作数允许值的范围会变化，可以选择为DBIT符号、SFBIT属性、SABIT属性位符号、使用EQU伪指令定义的位符号（只有操作数含有ADDRESS属性符号时）以及其它符号。

<示例4>

```
            BSEG
DBITSYM0    DBIT                        ; (h)
DBITSYM1    DBIT
DBITSYM2    DBIT
BIT1_PM0    EQU     PM0.1               ; (i)
BIT2_P0     EQU     P0.2                ; (j)
            DSEG
ADDRESS0:   DS      1
ADDRESS1:   DS      1
ADDRESS2:   DS      1
ADR_BIT0    EQU     ADDRESS0.0          ; (k)
ADR_BIT1    EQU     ADDRESS0.1
ADR_BIT2    EQU     ADDRESS0.2
            CSEG
            SET1    !DBITSYM0           ; (l)
            SET1    !BIT1_PM0           ; (m)
            SET1    !BIT2_P0            ; (n)
            SET1    !ADR_BIT0           ; (o)
```

上例说明描述字的操作数时，可以使用（h）行描述的DBIT符号，（i）行和（j）行描述的SFBIT属性和SABIT属性位符号，以及（k）行描述的使用EQU伪指令定义的位符号（只有操作数含有ADDRESS属性符号时）。操作如（i）至（o）行语句所示，所以值的范围决定于描述时使用的符号属性。

同样的原因也适用于ES:!addr16.bit使用的符号属性值域。

（4）　word（字）

示例5解释了为什么字的操作数允许值的范围会变化，可以选择为数值常数和NUMBER属性符号和ADDRESS属性符号。

<示例5>

```
                DSEG
ADDRESS0:       DS      1
ADDRESS1:       DS      1
ADDRESS2:       DS      1
                CSEG
                MOV     B , #0
                MOV     ADDRESS0[B] , A         ; (p)
                MOV     C , #1
                MOV     ADDRESS0[C] , A         ; (q)
                MOVW    BC , #2
                MOV     ADDRESS0[BC] , AX       ; (r)
```

使用“字”作为操作数的语句，比如（p）至（r）行出现的word[B], word[C]和word[BC]，一般也可以接受标签（ADDRESS属性符号）作为操作数。这样通过!addr16同样的方式来描述标签就可以简化代码。同样的原因，ES:word[B], ES:word[C], ES:word[BC]的代码也可以被简化。

表 2-12　伪指令的操作数的值范围

伪指令类型	伪指令	值范围
区段定义伪指令	CSEG AT	0H 至 FEFFH（不包括 SFR 和 2ndSFR）
	DSEG AT	0H 至 FEFFH（不包括 SFR 和 2ndSFR）
	BSEG AT	0H 至 FEFFH（不包括 SFR 和 2ndSFR）
	ORG	0H 至 FEFFH（不包括 SFR 和 2ndSFR）
符号定义伪指令	EQU	20-bit 值 0H 至 FFFFH
	SET	20-bit 值 0H 至 FFFFH
存储器初始化和区域保留伪指令	DB	8-bit 值 0H 至 FFH
	DW	16-bit 值 0H 至 FFFFH
	DG	20-bit 值 0H 至 FFFFH
	DS	8-bit 值 0H 至 FFFFH
自动分支指令选择伪指令	BR/CALL	0H 至 FEFFH

2.15.2 指令所需的操作数的尺寸

指令可以被分为机器指令和伪指令。对于需要立即值和符号来作为操作数的指令来说，所需的操作数的尺寸根据每个指令而变化。因此，当描述的数据超过指令所需操作数的尺寸时，将产生一个错误。

表达式的运算按照无符号数的32位来进行。如果计算结果超过0FFFFFFFF H（32位），则将输出一个警告消息。

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

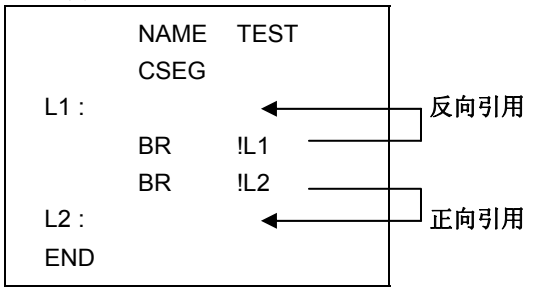
2.15.3 操作数的符号属性和重定位属性

当名称、标签和$（表示位置计数器）被描述为指令操作数时，它们不一定会被描述为操作数。这取决于用作它们的表达式中的项的符号属性和重定位属性（参见2.12　操作限制），也取决于名称和标签的引用方向。

名称和标签的引用方向可以是反向引用或正向引用。

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

<例>



关于这些符号属性和重定位属性，与名称及标签的引用方向一样，都显示在如下表。

表 2-13　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	后向	前向	后向	前向	后向	前向	后向	前向	
描述格式	byte	OK	OK	OK	OK	OK	OK	OK	OK	NG
	word	OK	OK	OK	OK	OK	OK	OK	OK	NG
	saddr	OK	OK	OK	OK	OK	OK	OK	OK	O[注2, 3]
	saddrp	OK	OK	OK	OK	OK	OK	OK	OK	O[注2, 4]
	sfr	OK[注5]	OK	OK	OK	OK	OK	OK	OK	O[注2, 6]
	sfrp	NG	NG	NG	NG	NG	NG	NG	NG	O[注2, 7]
	addr20	OK	OK	OK	OK	OK	OK	OK	OK	NG
	addr16	OK	OK	OK	OK	OK	OK	OK	OK	NG
	addr8	OK	OK	OK	OK	OK	OK	OK	OK	NG
	addr5	OK	OK	OK	OK	OK	OK	OK	OK	NG
	bit	OK	OK	NG	NG	NG	NG	NG	NG	NG
	n	OK	OK	NG	NG	NG	NG	NG	NG	NG

<说明>

前向：　表示前向引用。
后向：　表示后向引用。
OK：　表示可以这样描述。
NG：　表示错误
—:　表示不可以这样描述。

注 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。

注 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。

注 3. saddr 域的 sfr 保留字

注 4. saddr 域的 sfrp 保留字

注 5. 仅绝对表达式

注 6. 仅允许 8 位访问的 sfr 保留字

注 7. 仅允许 16 位访问的 sfr 保留字

表 2-14　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	
伪指令 \ 引用方向		后向	前向	后向	前向	后向	前向	后向	前向	后向	前向	后向	前向	后向	前向
ORG		OK 注1	—	—	—	—	—	—	—	—	—	—	—	—	—
EQU 注2		OK	—	OK	—	OK 注3	—	—	—	OK	—	OK 注3	—	—	—
SET		OK 注1	—	—	—	—	—	—	—	—	—	—	—	—	—
DB	大小	OK 注1	—	—	—	—	—	—	—	—	—	—	—	—	—
	初始值	OK	OK	OK	OK	OK	OK	OK	OK	—	—	—	—	—	—
DW	大小	OK 注1	—	—	—	—	—	—	—	—	—	—	—	—	—
	初始值	OK	OK	OK	OK	OK	OK	OK	OK	—	—	—	—	—	—
DG	大小	OK 注1	—	—	—	—	—	—	—	—	—	—	—	—	—
	初始值	OK	OK	OK	OK	OK	OK	OK	OK	—	—	—	—	—	—
DS		OK 注4	—	—	—	—	—	—	—	—	—	—	—	—	—
BR		OK	—	—	—	—	—	—	—	—	—	—	—	—	—

OK: 可以描述

—: 不可以描述

注 1. 只有绝对表达式可以被描述。

注 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
- HIGHW 绝对 ADDRESS 属性
- LOWW 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 8 模式的优化所影响时

注 3. 不允许 HIGH/LOW/HIGHW/LOWW/DATAPOS/MASK 操作符创建一个可重定位的项。

注 4. 参考 “**3.4** 存储器初始化和区域保留伪指令”。

第 3 章　伪指令

本章对伪指令进行了说明。
伪指令是控制 RA78K0R 执行一系列操作处理所必需的指令，可以指导所有类型的指令。

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

下表列出了伪指令的类型

表 3-1　伪指令列表

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

后续章节将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　区段定义伪指令

源模块必须以区段为单位进行描述。
区段定义伪指令用于定义以下四种类型的段：

- 代码区段
- 数据区段
- 位区段
- 绝对区段

段的类型决定了每个区段在存储器中的地址范围。
下表列出了每个区段的定义方法以及每个区段的存储地址。

表 3-2　段定义方法和内存地址位置

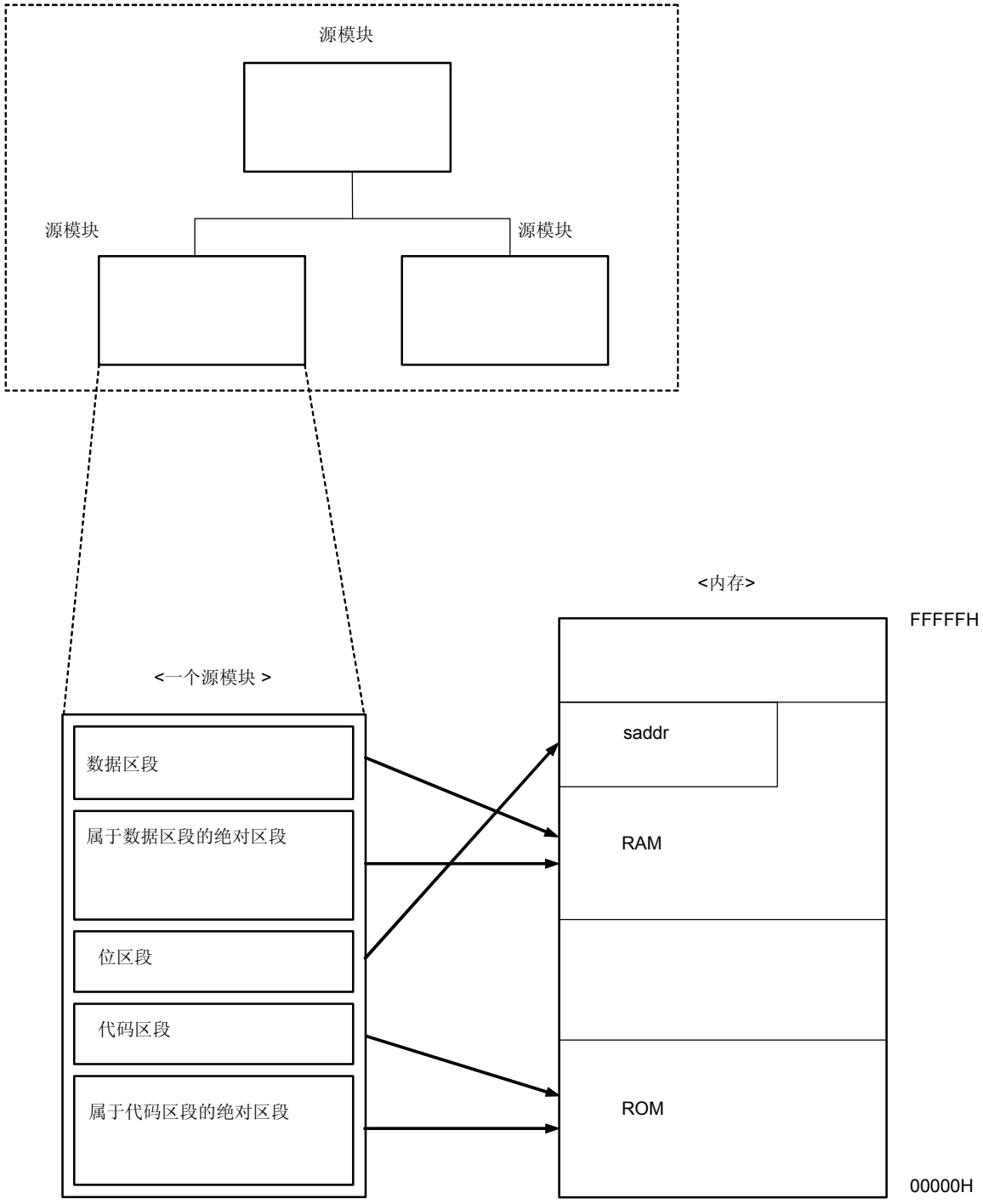
区段类型	定义方法	每个区段的存储地址
代码区段	CSEG 伪指令	内部或外部 ROM 地址内
数据区段	DSEG 伪指令	内部或外部 RAM 地址内
位区区段	BSEG 伪指令	内部 RAM 的 saddr 区内
绝对区段	用 CSEG、DSEG 或 BSEG 伪指令指定位置地址(AT 位置地址)，为重定位属性	指定地址

如果用户想要确定区段的分配地址，则将区段当作绝对区段进行描述。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。
下表给出了一个区段不能分配的区域示例。

选项字节区域	C0 至 C2H (用户的选项字节), C3H (在片调试功能选项字节)
当指定安全 ID 时	C4H 至 CDH
当使用在片调试功能时	02H 至03H, CE至 D7H (为在片调试功能保留) 程序的区域从用户指定的-go 选项地址开始

区段分配的示例如下。

图 3-1　段的内存位置



可以使用的区段定义伪指令描述如下：

- CSEG
- DSEG
- BSEG
- ORG

CSEG

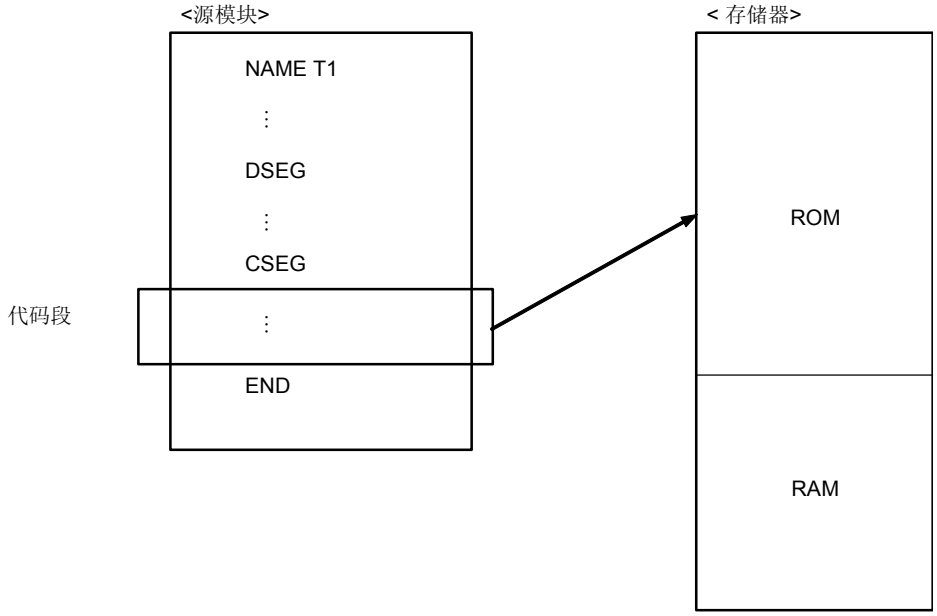
[描述格式]

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。
- 在遇到一个区段定义伪指令(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。

[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 的重定位属性给出了重定位属性。

表 3-3　CSEG 的重定位属性

重定位属性	描述格式	说明
CALLT0	CALLT0	通知汇编器分配该指定区段，从而使段的起始地址在地址范围 0080H 至 00BF 内的偶地址。
FIXED	FIXED	通知汇编器分配该区段的起始地址在范围 000C0H 至 0FFFH 内。
BASE	BASE	通知汇编器分配该区段的起始地址在范围 000C0H 至 0FFFH 内。
AT	AT 绝对表达式	通知汇编器分配该区段至一个绝对地址(SFR 和 2ndSFR 除外)
UNIT	UNIT	通知汇编器分配该区段至任何地址(存储器 ROM 区域的 000C0H 至 EFFFFH)
UNITP	UNITP	通知汇编器分配该指定段至任何地址，从而可使地址的起点是偶地址(存储器 ROM 区域的 000C0H 至 EFFFFH)。
IXRAM	IXRAM	通知汇编器确定该区段至内部扩展 RAM
SECUR_ID	SECUR_ID	这是安全 ID 指定属性。只能用来说明安全 ID。 通知汇编器分配该区段的起始地址在范围 000C4H 至 000CDH 内。
PAGE64KP	PAGE64KP	通知汇编器分配该区段的起始地址在不超过 64KB 边界的 ROM 存储器区域，从而可使地址的起点是偶地址。不同文件中的同名区段不做合并处理。
UNIT64KP	UNIT64KP	通知汇编器分配该区段的起始地址在不超过 64KB 边界的 ROM 存储器区域，从而可使地址的起点是偶地址。同名区段不做合并处理。

重定位属性	描述格式	说明
MIRRORP	MIRRORP	通知汇编器分配该区段的起始地址在 RAM 空间的映射区域中。当 MAA=0 时，映射区域的地址为(01000H 至 0xxxxH)；当 MAA=1 时，映射区域的地址为(11000H 至 1xxxxH)
OPT_BYTE	OPT_BYTE	用于设置用户选项字节和在片调试属性。不能用于其它用途。 通知汇编器分配该区段的起始地址在范围 000C0H 至 000C3H 内。

注　各个设备器件映射在 RAM 空间的地址范围会有所不同。

- 如果没有为代码段指定重定位属性，则汇编器认为指定了“UNIT”。
- 如果指定的重定位属性不在**表 3-3** 的列表范围内，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 伪指令的符号字段描述一个段名即可命名一个代码段。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下表中。

重定位属性	缺省段名
CALLT0	?CSEGT0
FIXED	?CSEGFX
UNIT (缺省)	?CSEG
UNITP	?CSEGUP
IXRAM	?CSEGIX
BASE	?CSEGB
SECUR_ID	?CSEGSI
PAGE64KP	?CSEGP64
UNIT64KP	?CSEGU64
MIRRORP	?CSEGMIP
OPT_BYTE	?CSEGOB0
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。

- 代码区段的描述可以分为多个单元，一个模块文件中描述的同样重定位属性和同名代码区段被汇编器处理为一系列的区段。
  注意 1　　描述代码区段时，重定位属性用 AT，则不能被分为多个单元。
  注意 2　　如果必要，则插入一个字节间隔，于是 CALLT0 重定位属性指定的地址可能是偶地址。
- 只有当数据区段的重定位属性是 UNIT, CALLT0, FIXED, UNITP, BASE, PAGE64KP, UNIT64KP, MIRRORP, 或 SECUR_ID 时，两个以上不同的模块中可以出现同名的数据区段，并且在连接过程中合并为一个数据区段。
- 区段名称不能作为符号被引用。
- 汇编器可输出区段的最大数量是 255 个别名段，其中包括 ORG 伪指令定义的段。同名区段计为一个区段。
- 段名可识别字符最多为 8 个。
- 段名字符区分大小写。
- 使用 OPT_BYTE 指定用户选项字节和在片调试。
  如果为芯片指定的选项字节不是用选项字节特征来完成的，则会发生错误。
  如果用户选项字节所指定的芯片不支持用户选项字节特征，在每个地址都定义一个“?CSEGOB0”默认区段，并从设备文件读取初始值。

[应用示例]

```
        NAME    SAMP1
C1              CSEG                        ; (1)
C2              CSEG    CALLT0              ; (2)
        CSEG    FIXED               ; (3)
C1              CSEG    CALLT0              ; (4) <-- Error
        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

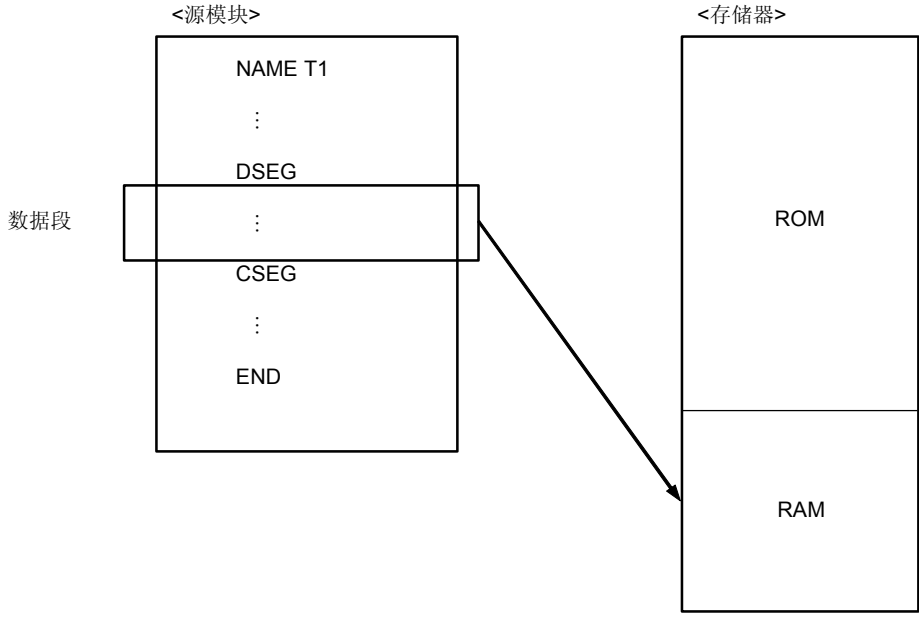
[描述格式]

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。
- 在遇到一个区段定义伪指令(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，在数据区段内不出现该指令。
- 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签。在描述源程序时使用该标签。
  每个保留为数据区段的区域由连接器分配定位，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。

[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 的重定位属性给出了数据区段可用的重定位属性。

表 3-4　DSEG 的重定位属性

重定位属性	描述格式	说明
SADDR	SADDR	通知汇编器将指定区段分配在 saddr 区域(saddr 区域：FFE20H 至 FFEFFH)
SADDRP	SADDRP	通知汇编器将指定区段分配在 saddr 区域(saddr 区域：FFE20H 至 FFEFFH)的偶数地址
AT	AT 绝对表达式	通知汇编器将指定区段分配在绝对地址(除 SFR 和 2ndSFR 之外)
UNIT	UNIT 或没有指定	通知汇编器将指定区段分配在任何位置(名为 RAM 的内存区)
UNITP	UNITP	通知汇编器将指定区段分配在任何位置的偶数地址(名为 RAM 的内存区)
BASEP	BASEP	通知汇编器将指定区段分配在内部 RAM 区，起始地址可能为偶地址（saddr 区域：FxxxxH 至 FFEFFH；不包括 FxxxxH 地址之外的外部 RAM 区域）
PAGE64KP	PAGE64KP	通知汇编器分配该区段的起始地址在不超过 64KB 边界的 RAM 存储器区域，从而可使地址的起点是偶地址。不同文件中的同名区段不做合并处理。
UNIT64KP	UNIT64KP	通知汇编器分配该区段的起始地址在不超过 64KB 边界的 RAM 存储器区域，从而可使地址的起点是偶地址。同名区段不做合并处理。

注　　xxxx 所代表的地址会根据使用的设备器件而有所不同。

- 重定位属性和 78K0 系列汇编器兼容，它的功能表达方式也是"UNIT"。

下表列出来 78K0 系列的 DSEG 重定位属性。

重定位属性	描述格式
IHRAM	IHRAM
LRAM	LRAM
DSPRAM	DSPRAM
IXRAM	IXRAM

- 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不在**表 3-4** 的列表范围内，则汇编器输出错误信息，并假定指定了"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。

在数据区段不可以出现机器语言指令（包括 BR 伪指令），如果出现有这类描述，发生错误且该行被忽略。

- 通过在 DSEG 伪指令的符号字段描述一个段名即可命名一个数据段。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下表中。

重定位属性	缺省段名
SADDR	?DSEGS
SADDRP	?DSEGSP
UNIT (缺省)	?DSEG
UNITP	?DSEGUP
IHRAM	?DSEGIH
LRAM	?DSEGL
DSPRAM	?DSEGDSP
IXRAM	?DSEGIX
BASEP	?DSEGBP
PAGE64KP	?DSEGP64
UNIT64KP	?DSEGU64
AT	区段名称不可忽略

- 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。
- 数据区段的描述可以分为多个单元，一个模块文件中描述的同样重定位属性和同名数据区段被汇编器处理为一系列的区段。
  注意 1　　描述数据区段时，重定位属性用 AT，则不能被分为多个单元。
  注意 2　　如果重定位属性是 SADDR，则需要时可以插入一个字节间隔，于是紧跟在 DSEG 伪指令之后的地址可能是偶地址。
- 如果重定位属性是 SADDRP，则该指定段分配在紧跟 DSEG 伪指令之后的偶地址。
- 如果同名数据段的重定位属性不同，则产生错误。因此，每种重定位属性的同名数据段数量为 1。
- 只有当数据区段的重定位属性是 UNIT, UNITP, SADDR, SADDRP, LRAM, IHRAM, DSPRAM, IXRAM, BASEP, PAGE64KP 或 UNIT64KP 时，两个以上不同的模块中可以出现同名的数据区段，并且在连接过程中合并为一个数据区段。
- 区段名称不能作为符号被引用。
- 汇编器可输出区段的最大数量是 255 个别名段，其中包括 ORG 伪指令定义的段。同名区段计为一个区段。
- 段名可识别字符最多为 8 个。
- 段名字符区分大小写。

[应用示例]

```
            NAME    SAMP1
            DSEG                        ; (1)
WORK1 :     DS      2
WORK2 :     DS      1
            CSEG
            MOV     A , !WORK2          ; (2)
            MOV     A , WORK2           ; (3) <-- Error
            MOVW    DE , #WORK1         ; (4)
            MOVW    AX , WORK1          ; (5) <-- Error
            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”。
　　可重定位标签 “WORK2” 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”。
　　可重定位标签 “WORK1” 不能表示成 “saddrp”。 因此，该描述的结果将产生错误。

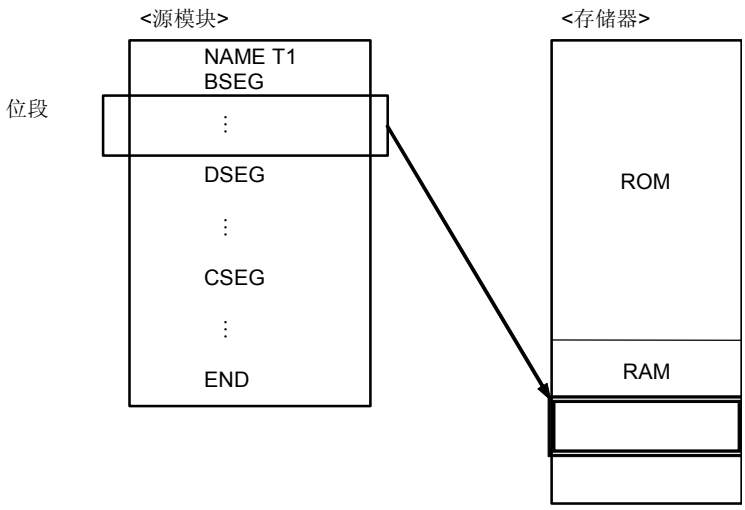
BSEG

[描述格式]

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 伪指令向汇编器指示位段的开始。
- 位区段定义了在源模块中使用的 RAM 地址。
- 在遇到一个区段定义伪指令(CSEG, DSEG, BSEG 或 ORG)或 END 伪指令之前，在 BSEG 伪指令之后由 DBIT 伪指令定义的存储器区域属于位段。



[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

[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 的重定位属性给出了位段可用的重定位属性。

表 3-5　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	通知汇编器把指定区段的起始地址分配在绝对地址的第 0 位。禁止以位作单元来指定(00000HF 至 FFFFFH)（除 SFR 和 2ndSFR 之外）。
UNIT	UNIT (或缺省)	通知汇编器可以将指定区段分配在任何位置(FFE20H 至 FFEFFH)。

- 如果位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不在**表 3-5** 的列表范围内，则汇编器输出错误信息，并假定指定了"UNIT"。如果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))。

<绝对位段>

字节地址	Bit 位置							
	0	1	2	3	4	5	6	7
0FFE20H	0FFE20H.0	0FFE20H.1	0FFE20H.2	0FFE20H.3	0FFE20H.4	0FFE20H.5	0FFE20H.6	0FFE20H.7
0FFE21H	0FFE21H.0	0FFE21H.1	0FFE21H.2	0FFE21H.3	0FFE21H.4	0FFE21H.5	0FFE21H.6	0FFE21H.7

<可重定位位段>

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

备注　在可重定位位段内，字节地址指定了从该段起始位置开始的偏移量，该偏移量以字节为单位。
在目标转化器输出的符号表中，列出了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
⋮	⋮

- 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。

重定位属性	缺省段名
UNIT (或缺省)	?BSEG
AT	区段名称不可忽略。

- 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。
- 同名的位段必需有同样的重定位属性 UNIT（当重定位属性为 AT 时，禁止将多个区段指定为相同的名称。）
- 如果在一个模块中的同名区段的重定位属性不是 UNIT，会发生错误并忽略该行。
- 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。
- 位段被连接器分配在 0H 至 FFFFFH。
- 在位段中不能出现标签。
- 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。除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 256 个别名段，其中包括 ORG 指令定义的段。同名的位段作为一个区段。
- 段名可识别字符最多为 8 个。
- 段名区分大小写字符。

[应用示例]

```
        NAME    SAMP1
FLAG    EQU     0FFE20H
FLAG0   EQU     FLAG.0              ; (1)
FLAG1   EQU     FLAG.1              ; (2)
        BSEG                        ; (3)
FLAG2   DBIT

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

<说明>

(1) 位地址(0FFE20H 中的第 0 位)的定义遵循和字节地址边界同样的注意事项。

(2) 位地址(0FFE20H 中的第 1 位)的定义遵循和字节地址边界同样的注意事项。

(3) 由 BSEG 伪指令来定义位段。
因为忽略了重定位属性，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。

(4) 该句描述可由 “SET1 FLAG.0”代替。FLAG 表示字节地址。

(5) 该描述中，没有考虑字节地址边界。

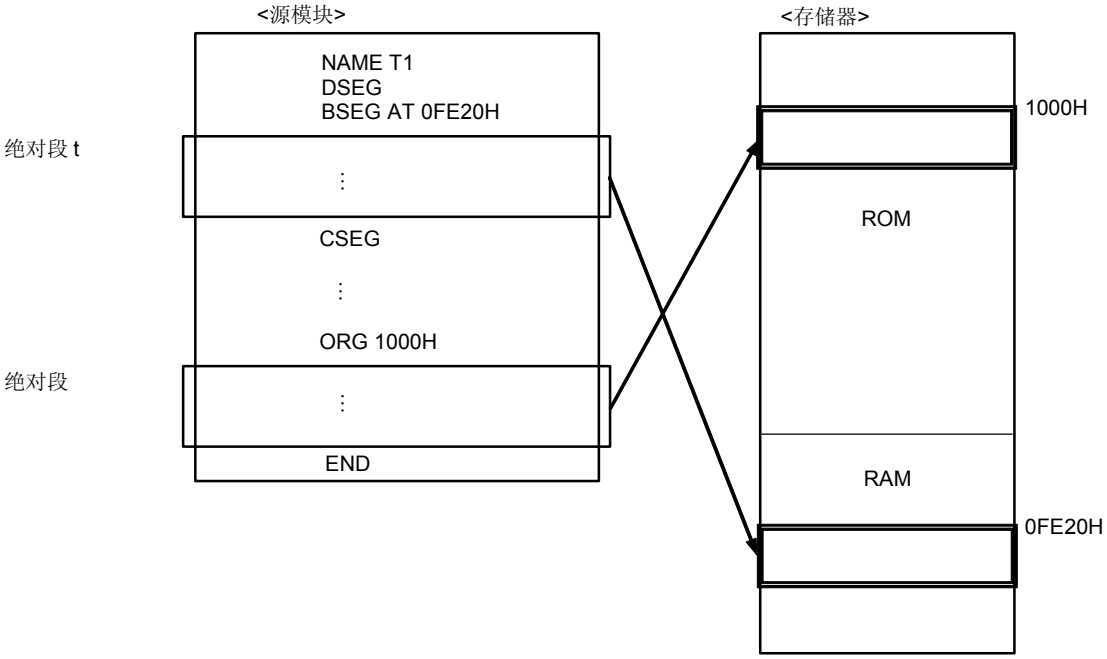
ORG

[描述格式]

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。
- 在遇到区段定义伪指令(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。



[用途]

- 指定 ORG 伪指令从一个指定地址分配一个代码段或数据段。

[说明]

- 使用 ORG 伪指令定义的绝对段属于 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。
- 如果在一个模块中用 ORG 伪指令定义的同名区段，和用 CSEG 或 DSEG 伪指令定义的带有 AT 属性的区段，以同样的方式进行处理。
- 如果在不同模块中用 ORG 伪指令定义的同名区段，和用 CSEG 或 DSEG 伪指令定义的带有 AT 属性的区段，以同样的方式进行处理。
- 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”?A0nnnnnn”，其中”nnnnn”表示指定段的 5 位十六进制起始地址(00000 至 FFFFF)。
- 如果在 ORG 伪指令之前没有出现 CSEG 和 DSEG 伪指令，由 ORG 伪指令定义的绝对段则被解释为代码段中的绝对段。
- 如果 ORG 伪指令的操作数是名称或标签，则该名称或标签必须是已经在源模块中定义的绝对项。
- 如果绝对表达式描述中出现非法目标，或者对绝对表达式的求值结果超过了 00000H 至 FFFFFH 的范围，汇编器输出一个错误并继续处理，认为绝对表达式的求值结果为 00000H。
- 操作数的绝对表达式以 32 位为单位进行求值。
- 段名不可以作为符号被引用。
- 汇编器可输出的段的总量为 256 个别名段，包括区段定义伪指令所定义的段。同名的区段算为一个。
- 段名可识别字符最多为 8 个。
- 段名字符区分大小写。

[应用示例]

```
            NAME    SAMP1
            DSEG
ORG         0FFE20H                         ; (1)
SADR1 :     DS      1
SADR2 :     DS      1
SADR3 :     DS      2
MAIN0       ORG     100H
            MOV     A , SADR1               ; (2) <-- Error
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区段的绝对段。
该绝对段将位于从地址”FFE20H”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”?A0FFE20”。

(2) 由于在属于代码段的绝对段内不能描述指令，所以发生一个错误。

(3) 该伪指令声明代码段的开始。

(4) 该绝对段位于从地址”1000H”起始的区域。

3.3　符号定义伪指令

符号定义伪指令用于对描述源模块使用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个名称的值。

可以使用的符号定义伪指令描述如下：

- EQU
- SET

EQU

[描述格式]

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

[功能]

- EQU 伪指令定义一个名称，该名称说明了操作数字段中指定表达式的值和属性(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。

[说明]

- EQU 伪指令可以出现在源程序的任何位置。
- 使用 EQU 伪指令定义过的符号不能用 SET 伪指令再定义，也不能作为标签。并且使用 SET 伪指令定义过的符号也不能用 EQU 伪指令再定义，同样也不能作为标签
- 当 EQU 伪指令的操作数中描述一个名称或标签时，使用已在源模块中定义的名称或标签。
  外部引用项不可以作为该伪指令的操作数。
- 如果表达式包含一个由在操作数中具有可重定位项的 HIGH/LOW/HIGHW/LOWW/DATAPOS/BITPOS 操作符创建的项，则表达式的描述不成立。
- 如果所描述的表达式的操作数符合下列模式的任何一个，都将产生错误：
  (1) 具有 ADDRESS 属性的表达式 1 - 具有 ADDRESS 属性的表达式 2
  或者下列条件<1>和 <2>之一在上述表达式(a)或(b)中实现：
  <a> 如果标签 1 在具有 ADDRESS 属性的表达式 1 中，标签 2 在具有 ADDRESS 属性的表达式 2 中，标签 1 和标签 2 属于同一段，而且如果两个标签之间描述的 BR 伪指令导致不能确定目标代码的字节数。
  <b> 如果标签 1 和标签 2 位于不同的段，且如果在区段的起始到标签之间出现的 BR 伪指令导致不能确定目标代码的字节数。
  (2) 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。
  (3) 具有 ADDRESS 属性的 HIGH 绝对表达式。
  (4) 具有 ADDRESS 属性的 LOW 绝对表达式。
  (5) 具有 ADDRESS 属性的 HIGHW 绝对表达式。

(6) 具有 ADDRESS 属性的 LOWW 绝对表达式。
(7) 具有 ADDRESS 属性的 DATAPOS 绝对表达式。
(8) 具有 ADDRESS 属性的 BITPOS 绝对表达式。
(9) 下面条件<a>在表达式(3)至(9)中实现：
<a> 如果在带有 ADDRESS 属性的标签和标签所属的段的起始地址之间描述一个 BR 指令，则 BR 指令会导致不能确定目标代码的字节数。

- 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。
- 下表显示了可作为 EQU 伪指令的操作数描述的比特值，以及可以引用这些比特值的范围。

操作数类型	符号值	引用范围
A.bit 注 1	1.bit	只可在同一模块内被引用
PSW.bit 注 1	0FFFFAH.bit	
Sfr 注 2.bit 注 1	0FFFXXH 注 3.bit	
2ndSfr 注 2.bit 注 1	0FXXXXH 注 4.bit	
saddr.bit 注 1	0FFXXXH 注 5.bit	可从另一模块被引用
expression.bit 注 1	0XXXXXH 注 6.bit	

注1. 1bit = 0 to 7。
注2. 更详细描述，参见各个设备的用户手册。
注3. “0FFFxxH”表示 sfr 的地址。
注4. 0FXXXXH 表示 2ndsfr 区域。
注5. “0xxxxH”表示 saddr 区(0FFE20H 至 0FFF1FH)。
注6. 0XXXXXH 表示 0H 至 0FFFFFH。

[应用示例]

```
        NAME    SAMP1
WORK1   EQU     0FFE20H                 ; (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”0FFE20H”，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”0FFE20H”。
(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(4)到(5)中已定义 “A.bit”和 “PSW.bit”的名称仅可在同一模块内被引用。
已定义”sfr.bit”，”saddr.bit”和”expression.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 连接伪指令**)。
作为示例中源模块的汇编结果，产生下列汇编列表。

```
                Assemble list
ALNO  STNO  ADRS   OBJECT     M I SOURCE STATEMENT
   1     1                               NAME        SAMP
   2     2
   3     3          ( FFE20 )  WORK1     EQU         0FFE20H     ; (1)
   4     4          ( FFE20.0 ) WORK10    EQU         WORK1.0     ; (2)
   5     5          ( FFF00.2 ) P02       EQU         P0.2        ; (3)
   6     6          ( 00001.4 ) A4        EQU         A.4         ; (4)
   7     7          ( FFFFA.5 ) PSW5      EQU         PSW.5       ; (5)
   8     8
   9     9     00000  710220               SET1        WORK10      ; (6)
  10    10     00003  712200               SET1        P02         ; (7)
  11    11     00006  71CA                 SET1        A4          ; (8)
  12    12     00008  715AFA               SET1        PSW5        ; (9)
  13    13
  14    14                                 END
```

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

[描述格式]

符号字段	助记词字段	操作数字段	注释字段
[名称]	SET	[重定位属性]	[;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。
  直到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 伪指令重新定义。
  用 SET 伪指令定义的符号不能用 EQU 伪指令重新定义。
- 不能定义位符号。

[应用示例]

```
        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　存储器初始化和区域保留伪指令

存储器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存储器区域。

可以使用的存储器初始化和区域保留伪指令描述如下：

- DB
- DW
- DG
- DS
- DBIT

DB

[描述格式]

符号字段	助记词字段	操作数字段	注释字段
[标签：]	DB	（大小）	[;注释]
[标签：]	DB	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。
  将被初始化的字节数量可以由“大小（size）”来指定。
- DB 伪指令还通知汇编器对一块存储器区域初始化，以字节为单位，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。

（1） 指定了大小时：

(a)若在操作数字段指定了大小，则汇编器初始化一段区域，该区域的大小等于指定的字节数量，初始值为“00H”.

(b)绝对表达式必须能够描述数量大小。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。

（2） 指定了初始值时：

(a) 表达式

表达式的值必须是 8 位数据。因此，操作数的值必须在 0H 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。

(b)字符串

如果描述的操作数是字符串，则将为串中的每个字符保留一个 8 位的 ASCII 码。

- DB 伪指令不能在位段中描述。
- 在 DB 伪指令的一个语句行内，可以指定两个或多个初始值。
- 初始值可以是包含可重定位符号的表达式，也可以是包含外部引用符号的表达式。

[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) <-- Error
            END
```

<说明>

(1)　由于指定了数量大小，汇编器将用初始值“00H”对每个字节区域进行初始化。

(2)　初始化了 6 字节长度的区域，初始值是字符串 ‘ABCDEF’。

(3)　初始化 3 字节长度的区域，初始值是“0AH, 0BH, 0CH”。

(4)　初始化 4 字节长度的区域，初始值是“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

[描述格式]

符号字段	助记词字段	操作数字段	注释字段
[标签：]	DW	（大小）	[;注释]
[标签：]	DW	初始值[,。。]	[;注释]

[功能]
- DW 伪指令通知汇编器初始化一个字区域。
  将被初始化的字数量可以由“大小（size）”来指定。
- DB 伪指令还通知汇编器对一块存储器区域初始化，以字（2 字节）为单位，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 位数值常量如地址或数据。

[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。

（1） 指定了大小时：
(a)若在操作数字段指定了大小，则汇编器初始化一段区域，该区域的大小等于指定的字数量，初始值为“00H”.
(b)绝对表达式必须能够描述数量大小。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。

（2） 指定了初始值时：
(a)常量
16 位或更少
(b)表达式
表达式的值必须存储为 16 位数据。
字符串不能用来描述初始值。

- DW 伪指令不能在位段中描述。
- 被指定初始值的高 2 位数字存储于 HIGH 地址内，低 2 位数字存储于 LOW 地址内。
- 在 DW 伪指令的一个语句行内，可指定两个或多个初始值。
- 初始值可以是包含可重定位符号的表达式，也可以是包含外部引用符号的表达式。

[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>

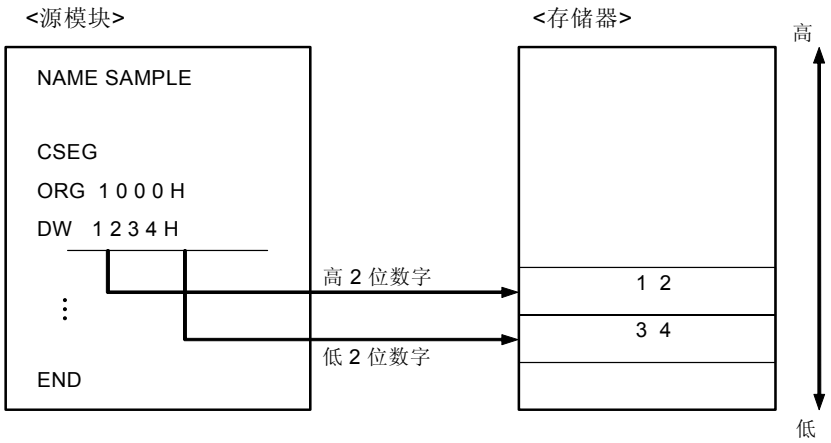
(1)　由于指定了数量大小，汇编器将用初始值“00H”对每个字进行初始化。

(2)　用 DW 伪指令定义矢量入口地址。

(3)　初始化了 2 字长度的区域，初始值是“34127856”。

注意　　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：



DG

[描述格式]

符号字段	助记词字段	操作数字段	注释字段
[标签：]	DG	（大小）	[;注释]
[标签：]	DG	初始值[,。。]	[;注释]

[功能]
- DG 伪指令通知汇编器以 32 位（4 字节）为单位，对某个 20 位的区域进行初始化。初始值和区域数量可以由操作数来指定。
- DG 伪指令还通知汇编器对一块存储器区域初始化，以字（2 字节）为单位，初始值由操作数字段的初始值指定。

[用途]
- 使用 DG 伪指令定义一个 20 位数值常量，比如在程序中使用的地址或数据。

[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，认为是给出了一个初始值。

（1） 指定了大小时：

(a)若在操作数字段指定了大小，则汇编器初始化一段区域，该区域的大小等于指定数量 X4 字节，初始值为“00H”。

(b)绝对表达式必须能够描述数量大小。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。

（2） 指定了初始值时：

(a)常量

20 位或更少

(b)表达式

表达式的值必须存储为 16 位数据。

字符串不能用来描述初始值。

- DG 伪指令不能在位段中描述。
- 指定初始值的最高字节被存储在 HIGH WORD 地址，最低字节被存储在 LOW 地址，最低 2 字节的高位字节被存储于 HIGH 地址内。
- 在 DG 伪指令的一个语句行内，可指定两个或多个初始值。
- 初始值可以是包含可重定位符号的表达式，也可以是包含外部引用符号的表达式。

[应用示例]

```
        NAME   SAMP1
DATA1 : DG     12345H，56789H      ; (1)
DATA2 : DG     ( 10 )              ; (2)
        END
```

<说明>

(1)　4 字节区域被初始化，初始值为"4523010089670500"。

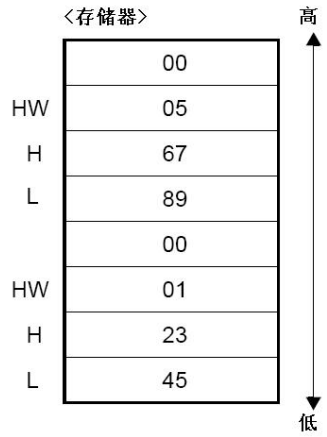
(2)　40 字节（10 X 4 字节）区域用"00H" 来初始化。

注意　　对于 20 位的值来说，存储器中的 HIGH WORD 地址用指定初始值的最高字节赋值，存储器中的 LOW 地址用指定初始值的最低字节赋值，存储器中的 HIGH 地址用最低 2 字节的高位字节来赋值。

<示例>

<源模块>

```
        NAME   SAMP1
DATA1 : DG
   12345H，56789H
        END
```



HW : HIGH WORD
H　: HIGH
L　: LOW

DS

[描述格式]

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

[功能]

- DS 伪指令通知汇编器保留一个存储器区域，长度为操作数字段中指定的字节数量。

[用途]

- DS 伪指令主要用于保留一个内存区（RAM）供程序使用。
  如果指定了一个标签，保留存储器区域的首地址赋给该标签。在源模块中，该标签用于描述存储器内的使用。

[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（标签）仅可后向引用。
- 只有从绝对表达式中扩展得到的下列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’中的标签 1 和在“带 ADDRESS 属性的表达式 2’中的标签 2 都是可重定位的，两标签必须在同一区段内定义。
  但是，下列两种情况之一将导致错误：
  (a) 如果标签 1 和标签 2 属于同一区段，而且，在两标签之间有 BR 伪指令，所以不能确定目标代码的字节数。
  (b) 如果标签 1 和标签 2 位于不同的区段，而且，在任一标签和所属区段起始位置之间出现 BR 伪指令的描述，导致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。

[应用示例]

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

<说明>

(1)　保留一个 10 字节的工作区域，但是区域的内容未定（不确定）。标签“TABLE1”作为起始地址标签。
(2)　保留一个 1 字节工作区域。
(3)　保留一个 2 字节工作区域。

DBIT

[描述格式]

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

[功能]

- DBIT 伪指令通知汇编器在位区段内保留一个 1 位的存储器区域。

[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 伪指令只能在位段内描述。
- 用 DBIT 伪指令保留的 1 位区域的内容未知（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, addr16.bit, ES:addr16.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

链接伪指令明确了引用其他模块定义符号的相关性。

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，当引用了模块 2 中定义的符号时，未经声明不得在各个模块中使用该符号。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。

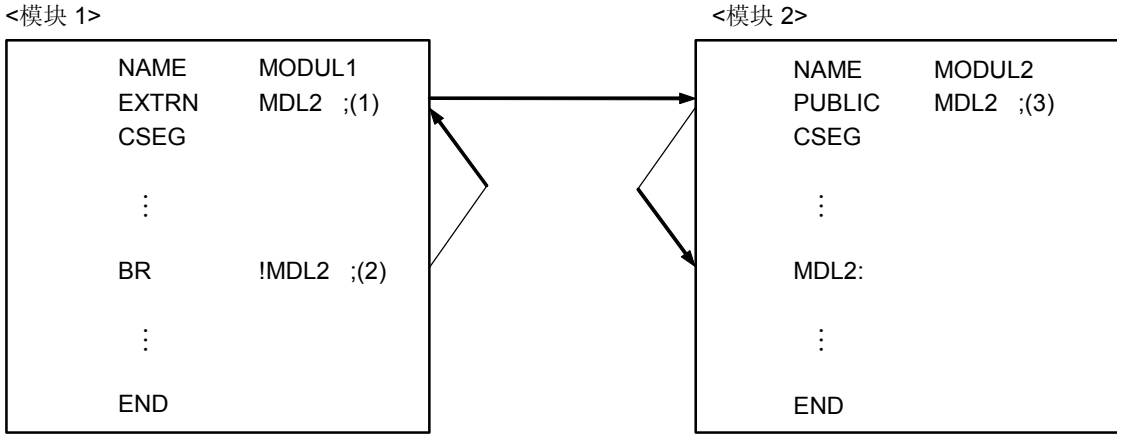
在模块 1 中，符号的外部引用声明表示必须引用在另一模块内定义的符号。在模块 2 中，符号的外部定义声明表示所定义符号可能在另一模块内被引用。

符号被首次引用时外部引用声明和外部定义声明均应正确定义。

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：

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

图 3-2　两模块之间的符号的关系



在上图中，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，用 EXTRN 伪指令对这个被作为外部引用的符号进行声明。

在模块 2 的（3）行中，用 PUBLIC 伪指令对模块 1 将要引用的符号“MDL2”进行声明，声明为外部定义。

链接器检查符号的外部引用是否与符号的外部定义相一致。

可以使用的链接伪指令描述如下：

- EXTRN
- EXTBIT
- PUBLIC

EXTRN

[描述格式]

符号字段	助记词字段	操作数字段	注释字段
[标签：]	EXTRN	符号名[，...]	[;注释]
[标签：]	EXTRN	BASE（符号名[，...]）	[;注释]

[功能]

- EXTRN 伪指令向连接器声明，将要在该模块中引用另一模块中的符号（除了位符号）。

[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。
- 导致的操作根据操作数描述格式的不同而有所区别。

BASE（符号名[，...]）	指定的符号被认为是 64KB 区域（0H 至 0FFFF）内的某块区域的符号，可以被引用。
未指定重定位属性	在经过连接器分配后，根据 PUBLIC 声明的区域来执行处理，然后可以被引用。

[说明]

- EXTRN 伪指令可以出现在源程序的任何位置（参见 **2.1 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。
- 如果用 EXTRN 伪指令声明的符号没有在该模块中引用，也不会发生错误。
- 宏名称不能描述为 EXTRN 伪指令的操作数（宏名称可参见**第五章 宏**）。
- 在一个模块内，EXTRN 伪指令仅允许对某个符号使用一个 EXTRN 声明。对该符号的第二个以及后续的 EXTRN 声明，连接器将输出警告信息。
- 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。
- 使用 EXTRN 伪指令定义的符号可以引用 64KB 区域（0H 至 0FFFF）内的某块区域。可以从 64KB 区域内引用 BASE（符号名）格式 声明的符号名称。

[应用示例]

<模块 1>

```
        NAME    SAMP1
        EXTRN   SYM1 , SYM2 , BASE (SYM3)       ; (1)
        CSEG
S1 :    DW      SYM1                            ; (2)
        MOV     A , SYM2                        ; (3)
        BR      !SYM3                           ; (4)
        END
```

<模块 2>

```
        NAME    SAMP2
        PUBLIC  SYM1 , SYM2 , SYM3              ; (4)
        CSEG
SYM1    EQU     0FFH                            ; (5)
DATA1           DSEG                            SADDR
SYM2 :          DB                              012H        ; (6)
C1      CSEG    BASE
SYM3 :          MOV                             A , #20H    ; (7)
        END
```

<说明>

(1) 该 EXTRN 伪指令声明“SYM1”，“SYM2”和“SYM3”符号，它们将在(2)、(3)和(4)中被作为外部应用符号来引用。
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 saddr 区域的代码。

(4) BR 伪指令引用符号 “SYM3”，并输出引用 64KB 区域（0H 至 0FFFF）内的某块区域的代码。符号“SYM1”，“SYM2”和“SYM3”被声明为外部定义。.

(5) 定义符号“SYM1”。

(6) 定义符号“SYM2”。

(7) 定义符号“SYM3”。

EXTBIT

[描述格式]

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

[功能]

- EXTBIT 伪指令向连接器声明，该模块将引用已在另一模块中定义的位符号。

[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外部引用。

[说明]

- EXTBIT 伪指令可以出现在源程序的任何位置。
- 在操作数字段最多可指定 20 个符号，各个符号间用（,）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。
- 在一个模块内，EXTBIT 伪指令仅允许对某个符号使用一个 EXTBIT 声明。对该符号的第二个以及后续的 EXTBIT 声明，连接器将输出警告信息。
- 如果用 EXTBIT 伪指令声明的符号没有在该模块中引用，也不会发生错误。

[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 伪指令声明的符号“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 区域的位符号。

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

PUBLIC

[描述格式]

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

[功能]

- PUBLIC 伪指令向连接器声明，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]

- PUBLIC 伪指令可以出现在源程序的任何位置。
- 在操作数字段最多可指定 20 个符号，各个符号间用（,）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。
- 在一个模块内，PUBLIC 伪指令仅允许对某个符号使用一个 PUBLIC 声明。对该符号的第二个以及后续的 PUBLIC 声明，连接器将输出警告信息。
- 在每个 bit 区域的位符号都可以用 PUBLIC 声明。
- 下列符号不可用作 PUBLIC 伪指令的操作数：
  (1) 用 SET 伪指令定义的名称
  (2) 在同一模块内用 EXTRN 或 EXTBIT 伪指令定义的符号
  (3) 区段名称
  (4) 模块名称
  (5) 宏名称
  (6) 不在该模块内定义的符号
  (7) 由 EQU 伪指令定义的其操作数具有 SBIT 属性的符号
  (8) 由 EQU 伪指令定义的 sfr 和 2ndSFR 符号（但是，sfr 区和 saddr 区重叠的区域除外）

[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     0FFE20H.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     0FFE21H.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　目标模块名称声明伪指令

目标模块名称声明伪指令给将被 RA78K0R 汇编器创建的目标模块赋予一个模块名称。

可以使用的目标模块名称声明伪指令描述如下：

- NAME

NAME

[描述格式]

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

[功能]

- NAME 伪指令把操作数字段声明的目标模块名称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]

- NAME 伪指令可以出现在源程序的任何位置。
- 对于模块名称描述的约定，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)”或 2 字节字符除外。
- 模块名称只能用在 NAME 伪指令中，其他任何智力功能或伪指令都不支持模块名称作为操作数。
- 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 256 个字符）作为模块名。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 256 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 伪指令将 “SAMPLE” 声明为模块名。

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。可使用的两个指令为“BR !addr20”和“BR $addr20”。

由于每个伪指令的字节数不同，用户必须根据分支目的点的地址范围来选择并使用最适当的操作数。这样有助于创建高效使用内存的程序。

因此，需要有一个伪指令根据分支目的点的地址范围，指导 RA78K0R 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。

可以使用的自动分支指令选择伪指令描述如下：

- BR
- CALL

BR

[描述格式]

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

[功能]

- BR 伪指令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、3-字节或 4-字节 BR 分支指令，并产生适用于选中指令的目标代码。

[用途]

- 在下表列出的分支指令中，汇编器决定分支目标地的地址范围，并自动选择输出一个指令，该指令使用最少数量的字节。如果不清楚是否可以描述 2-字节分支指令，则使用 BR 伪指令

分支指令	描述
“BR saddr20” (2 字节)	如果分支目的点的地址范围在-80H 到+70H 内，从 BR 伪指令后的地址开始
“BR !addr20” (3 字节)	如果分支目的点的地址范围在 64KB 内
“BR $!addr20” (3 字节)	计算从分支目的点的偏移，如果偏移量在-8000H 至 +7FFFH 范围内，则可以使用
“BR !!addr20” (4 字节)	在上述情况之外使用

如果操作数（分支目的点）被分排在重定位区段，和伪指令所处的区段不同，并且在 BASE 区域之外，伪指令将会被 4-字节指令并输出。

如果伪指令和操作数（分支目的点）都分配在不同的区段，并都在 BASE 区域之外，它们的类型也不同的话，即使操作数分配在绝对区段中，伪指令将会被 4-字节指令替代。

如果伪指令和分支目的点分配在不同的区段，并都在 BASE 区域之内，伪指令将会被 3-字节指令（BR !addr20）替代。

备注　不同的类型意味着如果 BR 伪指令被分配在绝对区段，不同类型分配在不同的重定位区段；或者如果 BR 伪指令被分配在重定位区段，不同类型分配在同一个绝对区段。

如果确定知道应该使用 2 字节、3 字节或 4-字节分支指令，则应该选择最适合的指令。和 BR 指令比较，这样可以节省汇编时间。

[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出现“$”，它表示当前位置计数器。
- 必须满足下列条件，才能够进行优化：
  <1> 表达式内标签或前向引用符号的数量不能多于 1 个。
  <2> 不能描述具有 ADDRESS 属性的 EQU 符号。
  <3> 在“具有 ADDRESS 属性的表达式 1 – 具有 ADDRESS 属性的表达式 2”时，不要描述使用 EQU 定义过的符号。
  <4> 不能在已经由 HIGH/LOW/HIGHW/LOWW/DATAPOS/BITPOS 操作符进行操作中描述具有 ADDRESS 属性的表达式。

  如果这些条件不满足，则将选择 4 字节 BR 指令。

[应用示例]

```
ADDRESS         NAME    SAMPLE
          C1    CSEG    AT      50H
00050H            BR      L1                      ; (1)
00052H            BR      L2                      ; (2)
00055H            BR      L3                      ; (3)
0007DH    L1 :
0FFFFH    L2 :
10000H    L3 :
          C2    CSEG    AT      20050H
20050H            BR      L4                      ; (4)
27FFFH    L4 :
                  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 $addr20) 。
(2) 由于该 BR 指令的分支目的点在 64KB 范围内，BR 伪指令将被 3 字节分支指令(BR !addr20)代替。
(3) BR 指令将被 4 字节分支指令(BR !!addr20)代替。
(4) 由于该行与分支目的点之间的位置在–8000H 和 +7FFFH 范围内，BR 伪指令将由 3 字节分支指令(BR !addr20)代替。

CALL

[描述格式]

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

[功能]

- CALL 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 3-字节或 4-字节 BR 分支指令，并产生适用于所选指令的目标代码。

[用途]

- 在下表列出的分支指令中，汇编器决定分支目标地的地址范围，并自动选择输出一个指令，该指令使用最少数量的字节。如果不清楚是否可以描述 3-字节分支指令，则使用 CALL 伪指令

分支指令	描述
“CALL !addr20” (3 字节)	如果分支目的点的地址范围在 64KB 内，则可以使用
“CALL $!addr20” (3 字节)	计算从分支目的点的偏移，如果偏移量在-8000H 至 +7FFFH 范围内，则可以使用
“CALL !!addr20” (4 字节)	在上述情况之外使用

如果操作数（分支目的点）被分排在重定位区段，和伪指令所处的区段不同，并且在 BASE 区域之外，伪指令将会被 4-字节指令并输出。

如果伪指令和操作数（分支目的点）都分配在不同的区段，并都在 BASE 区域之外，它们的类型也不同的话，即使操作数分配在绝对区段中，伪指令将会被 4-字节指令替代。

如果伪指令和分支目的点分配在不同的区段，并都在 BASE 区域之内，伪指令将会被 3-字节指令（CALL !addr20）替代。

备注　不同的类型意味着如果 CALL 伪指令被分配在绝对区段，不同类型分配在不同的重定位区段；或者如果 CALL 伪指令被分配在重定位区段，不同类型分配在同一个绝对区段。

如果确定知道应该使用 3 字节或 4-字节分支指令，则应该选择最适合的指令。和 CALL 指令比较，这样可以节省汇编时间。

[说明]

- CALL 伪指令值仅可用于一个代码段内。
- 直接跳转目的点可作为 CALL 伪指令的操作数。
- 必须满足下列条件，才能够进行优化：
  <1> 表达式内标签或前向引用符号的数量不能多于 1 个。
  <2> 不能描述具有 ADDRESS 属性的 EQU 符号。
  <3> 在“具有 ADDRESS 属性的表达式 1 – 具有 ADDRESS 属性的表达式 2”时，不要描述使用 EQU 定义过的符号。
  <4> 不能在已经由 HIGH/LOW/HIGHW/LOWW/DATAPOS/BITPOS 操作符进行操作中描述具有 ADDRESS 属性的表达式。

  如果这些条件不满足，则将选择 4 字节 CALL 指令。

[应用示例]

```
ADDRESS         NAME    SAMPLE
          C1    CSEG    AT      50H
00050H            CALL    L1                      ; (1)
00053H            CALL    L2                      ; (2)
08052H    L1 :
0FFFFH    L2 :
          C2    CSEG    AT      20050H
20050H            CALL    L3                      ; (3)
27FFFH    L3 :
                  END
```

<说明>

(1) 由于该 CALL 指令的分支目的点在 64KB 范围内，CALL 伪指令将被 3 字节分支指令(CALL !addr20)代替。

(2) CALL 指令将被 4 字节分支指令(CALL !!addr20)代替。

(3) 由于该行与分支目的点之间的位置在–8000H 和 +7FFFH 范围内，CALL 伪指令将由 3 字节分支指令(CALL !addr20)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的数量。

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。

可以使用的宏伪指令描述如下：

- MACRO
- LOCAL
- REPT
- IRP
- EXITM
- ENDM

MACRO

[描述格式]

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

[功能]

- MACRO 伪指令通过给该伪指令和 ENDM 伪指令之间描述的一串语句(称为宏程序体)赋于宏名称来执行宏定义，该宏名称在符号字段内指定。

[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，只要出现了被定义的宏名称(参见“5.2.2 宏引用”)，与宏名称对应的宏程序体就会在此处展开。

[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
- 要引用一个宏，只需描述该伪指令在助记词字段定义的宏名称即可。
- 对于将要在操作数字段描述的形式参量，其描述方式与描述符号的惯例相同。
- 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，如果出现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签如果用 LOCAL 伪指令声明，则只对一次宏扩展有效。
- 宏的嵌套(也就是，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，只要内存空间够用，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。
- 在一个宏程序体定义中不能出现两个或多个区段，否则输出错误信息。

[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

[描述格式]

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]

- 如果在某个宏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 的符号将被符号“??RAnnnn” (其中 nnnn=0000 至 FFFF)所代替。宏替换之后的符号“??RAnnnn”将按照全局符号来处理，且存于符号表中，从而可以用符号名“??RAnnnn”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。
- LOCAL 伪指令的描述必须在使用操作数字段指定的符号之前 (也即，LOCAL 伪指令必须在宏程序体的开始被描述)。
- 在源模块内，用 LOCAL 伪指令定义的符号名必须各不相同(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。

[应用示例]

```
        NAME    SAMPLE
        ; Macro definition
MAC1    MACRO
        LOCAL   LLAB                    ; (1)
LLAB : ;
        BR      $LLAB                     ; (2)
        ENDM ;
        ; Source text
REF1 :  MAC1                            ; (3)
??RA0000 :
        BR      $??RA0000               ; (2)
        BR      !LLAB                   ; (4) <-- Error
REF2 :  MAC1                            ; (5)
??RA0001 :
        BR      $??RA0001               ; (2)
        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>

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

REPT

[描述格式]

符号字段	助记词字段	操作数字段	注释字段
[标签：]	REPT ⋮ ENDM	绝对表达式	[;注释] [;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 程序块）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 程序块内，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 程序块内可描述汇编控制指令。
- 在 REPT-ENDM 程序块内不可描述宏定义。
- 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。

[应用示例]

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

<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

[描述格式]

符号字段	助记词字段	操作数字段	注释字段
[标签:]	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（称为 IRP-ENDM 程序块），重复次数等于实际参数的值，而其形式参量被操作数字段内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，仅有部分语句成为变量。

[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 程序块内，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 程序块内出现了 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 程序块内描述宏定义。
- 在 IRP-ENDM 程序块内可描述汇编控制指令。

[应用示例]

```
        NAME  SAMP1
        CSEG
        IRP     PARA , <0AH , 0BH , 0CH>                ; (1)
              ; IRP-ENDM block
        ADD     A , #PARA
        MOV      [ DE ] , A
        ENDM                                              ; (2)
              ; Source text
        END
```

<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
              ; IRP-ENDM block
        ADD     A , #0AH                                ; (3)
        MOV      [ DE ] , A
        ADD     A , #0BH                                ; (4)
        MOV     [ DE ] , A
        ADD     A , #0CH                                ; (5)
        MOV     [ DE ] , A
              ; Source text
        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

[描述格式]

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

[功能]

- EXITM 伪指令强行终止 MACRO 伪指令定义的宏程序体的展开，也会强行终止 REPT-ENDM 程序块或 IRP-ENDM 程序块的重复。

[用途]

- 在 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，才可以使用该功能。
- 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。

[应用示例]

```
            NAME    SAMP1
MAC1        MACRO                           ; (1)
            ; macro body
            NOT1    CY
$   IF ( SW1 )                              ; (2) <-- IF block
            BT      A.1 , $L1
            EXITM                           ; (3)
$   ELSE                                    ; (4) <-- ELSE block
            MOV1    CY , A.1
            MOV     A , #0
$   ENDIF                                   ; (5)
$   IF ( SW2 )                              ; (6) <-- IF block
            BR      [ HL ]
$   ELSE                                    ; (7) <-- ELSE block
            BR      [ DE ]
$   ENDIF                                   ; (8)
            ; Source text
            ENDM                            ; (9)
            CSEG
$   SET ( SW1 )                             ; (10)
            MAC1                            ; (11) <-- Macro reference
L1 :        NOP
            END
```

<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”。

备注　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。

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)
    ; Macro-expanded part
                NOT1    CY
$    IF ( SW1 )
                BT      A.1 , $L1
    ; Source text
L1 :            NOP
                END
```

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

[描述格式]

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被定义为宏功能的语句。

[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
```

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。

可以使用的汇编终止伪指令描述如下：

- END

END

[描述格式]

符号字段	助记词字段	操作数字段	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。

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

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

在源程序中描述控制指令的方式与描述汇编伪指令的方式相同。

对于表 4-1 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下表中。

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 **RA78K0R 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汇编的目标设备类型。

可以使用的处理器类型定义控制指令描述如下：

- PROCESSOR

PROCESSOR

[描述格式]

```
[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。
- 如果各个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。在各个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可能会有问题。

[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。
- 关于可以指定的处理器类型名称，敬请参阅所使用设备的用户手册或者“设备文件操作注意事项”。
- 如果指定的处理器类型名称与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 ( f1166a0 )
$  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目标模块文件输出调试信息，该目标模块文件由源模块文件创建而来。

可以使用的调试信息输出控制指令描述如下：

- DEBUG/NODEBUG
- DEBUGA/NODEBUGA

DEBUG/NODEBUG

[描述格式]

[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。
- 如果指定了两个以上的这种控制指令，最后指定的控制指令有效。
- 通过在起始命令行使用汇编选项（-G/-NG）来指定增加局部符号信息。
- 如果在源模块文件中定义的控制指令与在起始命令行中的定义不同，则在命令行中的定义规范优先。
- 即使指定了汇编选项（-NG），汇编器也会对 DEBUG 或 NODEBUG 控制指令进行语法检查。

DEBUGA/NODEBUGA

[描述格式]

[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. 执行调试时无需汇编源文件。
  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。
- 如果编译 C 编译器输出的调试信息，则在汇编该输出汇编源时不描述调试信息输出控制指令。汇编时所必需的控制指令作为 C 编译器的控制语句输出至汇编源。

4.4　交叉引用列表输出定义控制指令

在源模块文件中，交叉引用列表输出指定控制指令用于指定输出或不输出交叉引用列表。

可以使用的交叉引用列表输出定义控制指令描述如下：

- XREF/NOXREF
- SYMLIST/NOSYMLIST

XREF/NOXREF

[描述格式]

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

[用途]

- 当用户需要了解源模块文件中定义的每个符号在何处被引用，或者有多少这样的符号在源模块中被引用，则使用 XREF 控制指令输出交叉引用列表。
- 为了指定在每个汇编操作中输出或不输出交叉引用列表，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列表。
- 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，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。

SYMLIST/NOSYMLIST

[描述格式]

[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列表。
- 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，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。

可以使用的包含控制指令描述如下：

- INCLUDE

INCLUDE

[描述格式]

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

[功能]

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

[用途]

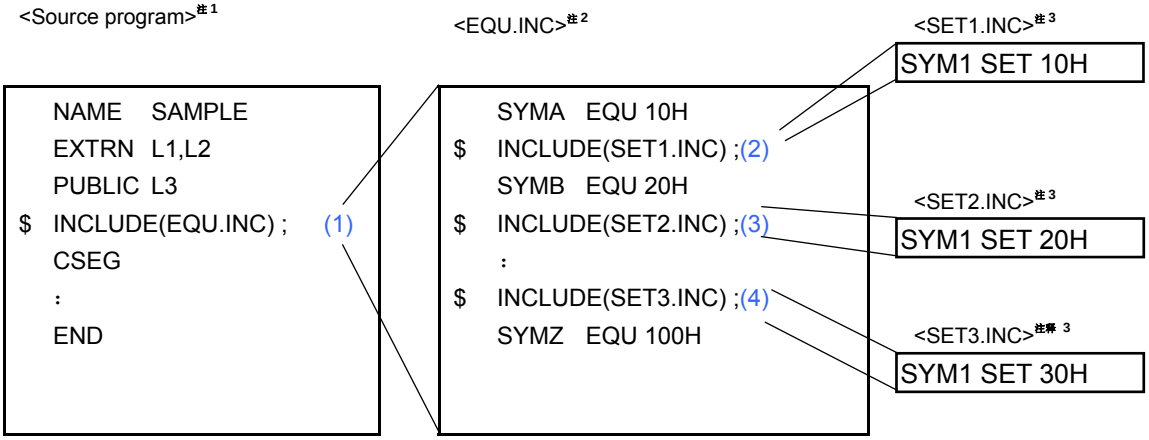
- 被两个或多个源模块共享的一大组语句可以组合成一个独立文件，称为 INCLUDE 文件。如果该语句组必须在每个源模块中使用，用 INCLUDE 控制指令指定所需 INCLUDE 文件的文件名。通过使用该控制指令，编写源模块的时间和精力可大大减少。

[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 INC78K0R 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 文件名前作为前缀。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 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 处。

- 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 文件里出现了预料之外的 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。

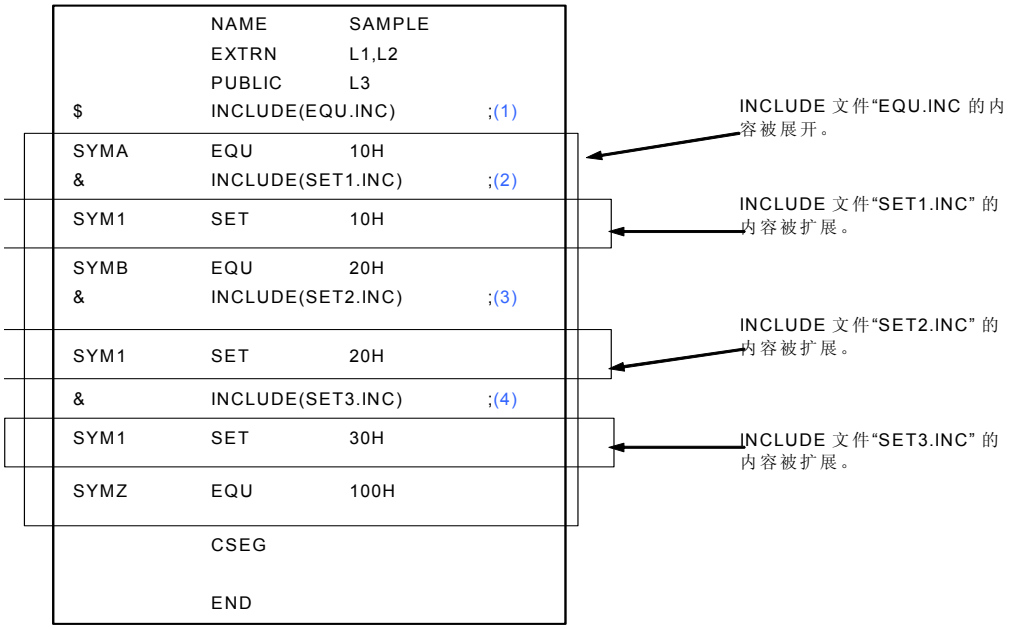
[应用示例]



(1)　该控制指令指定“EQU.INC” 为 INCLUDE 文件。
(2)　该控制指令指定“SET1.INC” 为 INCLUDE 文件。
(3)　该控制指令指定“SET2.INC” 为 INCLUDE 文件。
(4)　该控制指令指定“SET3.INC” 为 INCLUDE 文件。

注 1.　一个源文件里可定义两个或多个$IC 控制指令。同一 INCLUDE 文件也可被指定多次。
注 2.　对 “EQU.INC”.文件可指定两个或多个$IC 控制指令。
注 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。

当该源程序被汇编时，INCLUDE 文件的内容被展开为如下形式：



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，来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。

可以使用的汇编列表控制指令描述如下：

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

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（进纸）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 **RA78K0R 系列汇编器包的操作篇的汇编选项**。
- 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]

```
    :
    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。

输出的汇编列表形式如下所示。

<汇编列表>

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

LIST/NOLIST

[描述格式]

```
[Δ]$[Δ]LIST                    ; 缺省假定
[Δ]$[Δ]LI                      ; 缩写格式
[Δ]$[Δ]NOLIST
[Δ]$[Δ]NOLI                    ; 缩写格式
```

[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还会在汇编列表中输出。LIST 或 NOLIST 控制指令的映像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     ; 该语句不会输出到汇编列表
DATA2   EQU     11H     ; 该语句不会输出到汇编列表
        :               ; 该语句不会输出到汇编列表
DATAX   EQU     20H     ; 该语句不会输出到汇编列表
DATAY   EQU     20H     ; 该语句不会输出到汇编列表
$       LIST                            ; (2)
        CSEG
        :
        END
```

<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 控制指令自身也将输出至汇编列表。

GEN/NOGEN

[描述格式]

```
[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 控制指令本身输出至汇编列表之后，才发生指定的列表控制。
- 即使在 NOGEN 控制指令控制了列表输出之后，汇编器仍继续其处理，并累加语句数量计数器(STNO)。
- 如果在 NOGEN 控制指令之后指定了 GEN 控制指令，汇编器将恢复输出宏展开行。

[应用示例]

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

当上述源程序被汇编时，输出的汇编列表如下所示。

<汇编列表>

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

<说明>

(1)　由于指定了 NOGEN 控制指令，宏展开行将不输出到汇编列表中。

COND/NOCOND

[描述格式]

```
[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或者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 COND/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或者不满足汇编条件的行都输出至汇编列表。
- 在 COND 或 NOCOND 控制指令本身输出至汇编列表之后，才发生指定的列表控制。
- 即使在 NOCOND 控制指令控制了列表输出之后，汇编器仍累加 ALNO 和 STNO 计数。
- 如果在 NOCOND 控制指令之后指定 COND 控制指令，则汇编器将恢复输出那些满足汇编条件的行以及已经描述了 IF/_IF, ELSEIF/_ELSEIF, ELSE 和 ENDIF 的行。

[应用示例]

```
    NAME  SAMP
  $ NOCOND
  $ SET ( SW1 )
  $ IF ( SW1 )            ; 这部分虽然已被汇编，但是不输出至汇编列表中。
        MOV    A, #1H
  $ ELSE                  ; 这部分虽然已被汇编，但是不输出至汇编列表中。
        MOV    A , #0H                ; 这部分虽然已被汇编，但是不输出至汇编列表
中。
  $ ENDIF                 ; 这部分虽然已被汇编，但是不输出至汇编列表中。
    END
```

TITLE

[描述格式]

```
[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，汇编列表头的 TITLE 列将保持为空白。

[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列表的每一页开头的 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将会节约时间和精力。

[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。

[应用示例]

```
$       PROCESSOR(F1166A0)
$       TITLE('THIS IS TITLE')
        NAME  SAMPLE
        CSEG
        MOV    A,B
        END
```

当汇编上述源程序时，输出汇编列表如下一页显示（每页行数为 72）。

<汇编列表>

```
78K0R Series Assembler Vx.xx       THIS IS TITLE        Date: xx xxx xxxx Page: 1

Command      : -ll72 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 ( f1166a0 )
2       2                                   $         TITLE ( 'THIS IS TITLE' )
3       3                                             NAME          SAMPLE
4       4       ----                                  CSEG
5       5       00000   63                            MOV           A , B
6       6                                             END

Segment     information :
ADRS            LEN     NAME
00000  00001H  ?CSEG

Target chip     : uPD78F1166_A0
Device file     : Vx.xx
Assembly complete , 0 error(s) and 0 warning(s) found. (0)
```

SUBTITLE

[描述格式]

```
[Δ]$[Δ]SUBTITLE[Δ]([Δ]'character-string'[Δ])
[Δ]$[Δ]ST[Δ]([Δ]'character-string'[Δ])        ；缩写格式
```

[缺省假设]

- 没有指定 SUBTITLE 控制指令时，汇编列表头部的 SUBTITLE 部分将保持空白。

[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页打印一个副标题，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 SUBTITLE 控制指令，汇编器的 SUBTITLE 部分为空白。
- 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 SUBTITLE 列为空白。
- 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（'）之间的所有内容都不会输出。

[应用示例]

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

<说明>

(1)　该控制指令指定字符串‘THIS IS SUBTITLE 1’。

(2)　该控制指令指定一个换页操作。

(3)　该控制指令指定字符串 ‘THIS IS SUBTITLE 2’。

(4)　该控制指令指定一个换页操作。

(5)　该控制指令指定字符串 ‘THIS IS SUBTITLE 3’。

该示例的汇编列表如下所示（每页指定行数为 80）。

<汇编列表>

```
78K0R Series Assembler Vx.xx                    Date: xx xxx xxxx Page: 1

Command      : -cf1166a0   -ll80   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)
------------------------------------------------------- 换页 -------------
78K0R 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)
------------------------------------------------------- 换页 -------------
78K0R 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

Segment   informations :
ADRS          LEN                NAME
00000         00000H             ?CSEG

Target chip     : uPD78F1166_A0
Device file     : Vx.xx
Assembly complete , 0 error(s) and 0 warning(s) found. (0)
```

FORMFEED/NOFORMFEED

[描述格式]

```
[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，如果在一页的中间打印结束，则列表的最后一页可能会丢失。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 代码即可。
  大多数情况下，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。
- 如果在源模块中定义的控制指令（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。

WIDTH

[描述格式]

[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。

LENGTH

[描述格式]

[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（-LL）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

[描述格式]

[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，该列数作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。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。

可以使用的条件汇编控制指令描述如下：

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

IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF

[描述格式]

[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 都可以结合使用。也就是说，ELSEIF/_ELSEIF 可以和 IF 或_IF 及 ENDIF 成对使用。
- 条件表达式一般都描述为一个绝对表达式。
- 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 符号域**）。但是，可作为 switch 名被识别的最大字符数总是 31。
- 如果用 IF 或 ELSEIF 控制指令指定了两个或多个 switch 名，每个 switch 名称之间用分号（;）分开。
  每个模块最多可使用 5 个 switch 名。
- 当用 IF 或 ELSEIF 控制指令指定两个或多个 switch 名称时，如果一个 switch 名的值为真，则 IF 或 ELSEIF 条件被判定为真。
- 由 IF 或 ELSEIF 控制指令指定的每个 switch 名称的值必须使用 **SET/RESET** 控制指令定义。因此，如果在源模块中，由 IF 或 ELSEIF 控制指令指定的 switch 名称的值没有提前使用 SET 或 RESET 控制指令设置，则假定该值被复位。
- 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 名称 “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 名称 “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 名称 “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 名称 “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 名称 “SW1”和“SW2”的值为假，switch 名称 “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 名称 “SW1”和“SW2”的值以及(2) 中“SW3” 的值为假，而 switch 名称 “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 名称 “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

[描述格式]

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

[功能]

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

[用途]

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

[说明]

- 使用 SET 或 RESET 控制指令，至少要描述一个 switch 名称。
  描述 switch 名称所采用的原则与描述符号的惯例（参见 2.2.3 符号域）相同。但是，可作为 switch 名称被识别的最大字符数只有 31 个。
- 除了保留字和其他 switch 名称，指定的 switch 名称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 名称，每个 switch 名称用分号（;）分开。每个模块可最多使用 1000 个 switch 名称。
- 一个已经用 SET 控制指令设为“真”的 switch 名称，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 名称，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 名称不能输出到交叉引用列表中。

[应用示例]

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

<说明>

(1)　该指令给 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　日文汉字（kanji）码控制指令

使用日文汉字（kanji）码控制指令指定在注释行中出现的日文汉字（2-字节字符）的编码格式（2-字节编码）。

可以使用的日文汉字（kanji）码控制指令描述如下：

- KANJICODE

日文汉字（kanji）码

[描述格式]

```
[Δ]$[Δ]KANJICODE[Δ]kanji-code
```

[默认设置]

- $KANJICODE SJIS

[用途]

- 可以用来指定在注释行中出现的日文汉字（2-字节字符）的编码格式（2-字节编码）。

[功能]

- 只可在源模块文件的开头部分描述日文汉字码（KANJICODE）控制指令
- 如果同时指定了两个或更多个日文汉字码（KANJICODE）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（-ZS/-ZE/-ZN）来停止日文汉字码（KANJICODE）。
- 如果源模块中的控制指令定义（KANJICODE）与在起始命令行中的指定（-ZS/-ZE/-ZN）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（-ZS/-ZE/-ZN），汇编器仍将对 KANJICODE 控制指令执行语法检查。

4.9　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 5 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。

宏功能是指通过 MACRO 和 ENDM 伪指令定义为宏程序体的一系列语句（一个指令组），在宏名称被引用的位置展开整个宏程序体。

宏用于提高源程序的编码效率，和子程序有所区别。

宏和子程序具有明显的不同特征，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，使程序设计变得更容易）。

(2)　宏

- 宏的基本功能是用一个名字代替一组指令。
  用 MACRO 和 ENDM 伪指令定义为宏程序体的一系列（或一组）指令将在宏名称被引用的地方被展开。当汇编器发现宏名称被引用时，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。
- 宏可以带有参量。
  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存储器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　宏的定义

使用 MACRO 和 ENDM 伪指令来定义宏。

[描述格式]

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

[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**”。

5.2.2　宏的引用

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。

[描述格式]

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

[功能]

- 该语句调用在记忆字段指定的宏名称对应的宏程序体。

[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
        ENDM
        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　宏的展开

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。
- 对于汇编器来说，被展开的宏程序体与其它汇编语句的处理方式相同。

5.2.4　应用示例

当在 **5.2.2 宏引用**中对被引用的宏进行汇编时，宏程序体将被展开为如下形式。

```
        NAME    SAMPLE
        ; 宏的定义
ADMAC   MACRO           PARA1 , PARA2
                MOV     A , #PARA1
                ADD     A , #PARA2
        ENDM

        ; Source text
        CSEG
        :
        ; 宏的展开
        ADMAC           10H , 20H       ; (1)
        MOV     A , #10H
        ADD     A , #20H
        ; Source text
        :
        END
```

<说明>

（1） 通过宏引用，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，如果在某个宏内部定义了某全局符号，这个宏被多次引用从而展开一系列语句时，该全局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**）。
- 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]

```
        NAME SAMPLE
        ; 宏的定义
MAC1    MACRO
        LOCAL   LLAB                    ; (1)
LLAB :                                  ; (2)
        :
GLAB :                                  ; (3)
        BR      LLAB                ; (4)
        BR      GLAB                ; (5)
        ENDM
        :
        ; Source text
REF1 :  MAC1                            ; (6) <--宏的引用
        :
        BR      LLAB                ; (7) <-- 错误
        :
REF2 :  MAC1                            ; (8) <--宏的引用
        :
GLAB :                                  ; (9) <--错误
        :
END
```

<说明>

(1)　该 LOCAL 伪指令定义标签 “LLAB”为局部符号。
(2)　该 LOCAL 伪指令定义标签 “LLAB”为局部符号。
(3)　该 LOCAL 伪指令定义标签 “GLAB”为全局符号。
(4)　BR 伪指令引用宏“MAC1”里的局部符号“LLAB”。
(5)　BR 伪指令引用宏“MAC1”里的全局符号“GLAB”
(6)　该语句引用宏“MAC1”。
(7)　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。
(8)　该语句引用宏“MAC1”。同一个宏被引用了两次。

(9)　该 LOCAL 伪指令定义标签“GLAB”为全局符号。
同一个标签被定义了两次。
在汇编源程序时该描述会产生错误。

当汇编上述例子的源程序时，宏程序体将被展开为如下形式。

<汇编列表>

```
        NAME    SAMPLE
        :
REF1 :  MAC1
        ; 宏的展开
??RA0000 :
        :
GLAB :                                  <-- 错误
        BR      ??RA0000
        BR      GLAB
        ; Source text
        :
        BR      !LLAB                   <--错误
        BR      !GLAB
        :
REF2 :  MAC1
        ; 宏的展开
??RA0001 :
        :
GLAB :                                  <--错误
        BR      ??RA0001
        BR      GLAB
        ; Source text
        :
        END
```

<说明>

在宏“MAC1”内定义了全局符号“GLAB”。由于宏“MAC1”被引用了两次，当宏程序体内一系列语句被再次展开时，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“”（单引号标签）。

(1) &（联接）

- 和记号“&”将宏程序体内的字符串彼此连接起来。在宏被展开时，和标签左侧的字符串与该标签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。在宏展开时，“&”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。
- 括在一对单引号标签之内的“&”可作为数据进行处理。
- 两个连续“&”符号当作一个单独的“&”符号进行处理。

[应用示例]

<宏定义>

```
MAC     MACRO           P
LAB&P :                         ←形式参量 “P” 被识别。
        D&B     10H
        DB      'P'
        DB      P
        DB      '&P'
        ENDM
```

<宏引用>

```
        MAC     1H
LAB1H :
        DB      10H             <-- “D” 和 “B” 被连接在一起，变成“DB”。
        DB      'P'
        DB      1H
        DB      '&P'            <--在一对单引号之间的& 作为数据处理。
```

(2)　' (单引号标签)

- 如果一个被一对单引号括起来的字符串在宏引用行或 IRP 伪指令或分界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，且不包括两边的单引号标签。
- 如果一个被单引号对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。
- 若使用单引号标签作为文本里的单引号标签，连续两次书写该单引号标签。

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

第 6 章　产品应用

本章介绍了一些推荐的有效利用 RA78K0R 汇编包的方法。

6.1　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）和日文汉字码定义（-ZS/-ZE/-ZN）。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 ( f1166a0 )
$       KANJICODE   SJIS
$       XRFF
        NAME    TEST
C1      CSEG
        :
        END
```

6.2　如何开发具有高内存利用率的程序

与其它数据内存区域相比，短直接寻址区是一块可用短字节长度指令访问的区域。因此，有效使用这个区域可开发出具有高内存利用率的程序。

在一个模块内声明短直接寻址区域。

在这种方式下，即使所有打算分配在短直接寻址区域的变量不能分配到该区域，也可以轻松改变，从而那些被频繁访问的变量将会位于短直接寻址区。

[应用示例]

<模块 1>

```
        PUBLIC          TMP1 , TMP2
WORK    DSEG    AT      0FFE20H
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。

表 A-1　保留字的类型

类型	说明
符号字段	所有的保留字都不能在该字段中描述。
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。

表 A-2　保留字列表

类型	保留字				
操作符	AND	BITPOS	DATAPOS	EQ（=）	GE（>=）
	GT（>）	HIGH	HIGHW	LE（<=）	LOW
	LOWW	LT（<）	MASK	MOD	NE（< >）
	NOT	OR	SHL	SHR	XOR
伪指令	AT	BASE	BASEP	BR	BSEG
	CALLT0	CSEG	DB	DBIT	DG
	DS	DSEG	DSPRAM	DW	END
	ENDM	ENDS	EQU	EXITM	EXTBIT
	EXTRN	FIXED	IHRAM	IRP	IXRAM
	LOCAL	LRAM	MACRO	MIRRORP	NAME
	OPT_BYTE	ORG	PAGE64KP	PUBLIC	REPT
	SADDR	SADDRP	SECUR_ID	SET	UNIT
	UNIT64KP	UNITP			
控制指令	COND/ NOCOND			DEBUG/ NODEBUG	
	DEBUGA/ NODEBUGA [DG/NODG]			EJECT [EJ]	
	FORMFEED/ NOFORMFEED			GEN/ NOGEN	
	IF/_IF/ ELSEIF/_ELSEIF/ ELSE/ ENDIF				
	INCLUDE [IC]			KANJICODE	
	LENGTH			LIST [LI/NOLI]	
	PROCESSOR [PC]			SET/ RESET	
	SUBTITLE [ST]			SYMLIST/ NOSYMLIST	
	TAB			TITLE [TT]	
	WIDTH			XREF/ NOXREF [XR/NOXR]	
其它	DGL	DGS	SFR	SFRP	TOL_INF

备注　　方括号中的项表示缩写格式。

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

附录B　伪指令列表

表 B-1　伪指令列表

伪指令				功能分类	备注
符号字段	助记词字段	操作数字段	注释字段		
[区段名称]	CSEG	[重定位属性]	[;注释]	声明一个代码区段的开始	
[区段名称]	DSEG	[重定位属性]]	[;注释]	声明一个数据区段的开始	
[区段名称]	BSEG	[重定位属性]]	[;注释]	声明一个位区段的开始	
[区段名称]	ORG	绝对表达式	[;注释]	声明一个绝对区段的开始	禁止操作数内符号的前向引用。
名称	EQU	表达式	[;注释]	定义一个名称	名称:符号 禁止操作数内符号的前向引用或外部引用。
名称	SET	绝对表达式	[;注释]	定义一个重定义名	名称:符号 禁止操作数内符号的前向引用
[标签:]	DB	{(size) 初始值 [,...]}	[;注释]	初始化或保留一个字节的数据区	标签:符号 字符串可代替初始值设置。
[标签:]	DW	{(size)初始值 [,...]}	[;注释]	初始化或保留一个字的数据区字段	标签:符号
[标签:]	DG	{(size)初始值 [,...]}	[;注释]	初始化或保留 4 个字数据区字段	标签:符号
[标签:]	DS	绝对表达式	[;注释]	保留一个字节数据区字段	名称:符号 禁止操作数内符号的前向引用
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引用
[标签:]	EXTRN	符号名 [,...]	[;注释]	声明一个外部引用名称。	

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		
[标签:]	EXTBIT	位符号名称[,...]	[;注释]	声明一个外部引用名称	符号名称限定在具有比特值的符号名中。
[标签:]	PUBLIC	符号名 [,...]	[;注释]	声明一个外部定义名称。	
[标签:]	NAME	对象模块名	[;注释]	定义模块名	模块名: 符号
[标签:]	BR	表达式	[;注释]	自动选择一个分支指令	标签: 符号
宏名称	MACRO	[形式参量 [,...]]	[;注释]	定义一个宏	宏名称: 符号
[标签:]	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
[标签:]	REPT	绝对表达式	[;注释]	在宏扩展期间指定重复次数	标签: 符号
[标签:]	IRP	形式参量, <实际参量 [,...]>	[;注释]	给形式参量分配一个实际参数	标签: 符号
[标签:]	EXITM	无	[;注释]	中断宏扩展	仅在宏定义中使用
无	ENDM	无	[;注释]	终止宏定义	仅在宏定义中使用
无	END	无	[;注释]	表示源模块的结束	

索 引

[R]

[S]

[T]

[U]

[W]

[X]

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
- 订货信息
- 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