To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

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

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

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

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

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

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

用户手册

RA78K0S Ver. 2.00

汇编包

结构化汇编语言

目标设备

78K0S 微控制器

文档编号. U17389CA2V0UM00 (第 2 版)
发行日期 2009 年 1 月

日本印刷

[备忘录]

Windows 是 Microsoft Corporation 在美国和其他国家的注册商标或商标。
PC/AT 是国际商业机器公司的一个商标。
Solaris 是 Sun Microsystems 的一个商标。

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

  “标准等级”、“专业等级”以及“特殊等级”三种质量等级。

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

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

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

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

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

（注）

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

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

前言

该手册用来帮助用户准确理解在使用结构化汇编器（以下简称“结构化汇编器”）时的编码方法。该结构化汇编器包含在“RA78K0S 汇编器包”中（以下简称“RA78K0S”）即 78K/0 系列中用于紧凑的具有多功能的微控制器的汇编包。

该手册对于使用结构化汇编器之外的程序所涉及的方法不予解释，也不介绍结构化汇编器的操作方法。

因此，编写程序时请参考“**RA78K0S 汇编器包用户手册语言篇(U17390E)** 和 **操作篇 (U17391E)**.

本手册的内容为使用 RA78K0S Ver. 1.50.

[目标读者]

RA78K0S 面向的读者能理解 78K0S 系列中紧凑、具有多种用途的微控制器的功能。

读者如果需要了解 78K0S 系列中的紧凑、面向通用应用的微控制器的功能，需要参考目标芯片的用户手册。

[组成]

本手册组成如下:

第 1 章 概要

本章描述面向在微控制器的软件开发中结构化编译器的功能（角色等）。

第 2 章 源程序编码方法

本章描述源程序配置、编码语法以及其它基本原则和与源程序编码相关的惯例。

第 3 章 控制语句

控制语句用于描述程序结构中 “if~else~endif”指示器。

本章描述控制语句的功能及编码方法。

第 4 章 表达式

赋值及算术操作用表达式输入。

本章描述表达式的功能及编码方法。

第 5 章 指示

本章在描述如何编写和使用结构化汇编器的指示中使用了实际案例。

第 6 章 控制指令

本章在描述如何编写和使用结构化汇编控制指令中使用了实际案例

附录A 语句列表

本附录介绍了结构化汇编器的语句列表。

附录B 生成指令列表

本附录介绍了由结构化汇编器生成的指令列表。

本手册没有对指令设置作详细介绍

有关指令方面内容，请参考用于开发的微控制器用户手册

[如何阅读本手册]

手册使用结构化汇编器的读者应从“第一章 概述”开始读起。已经对结构换汇编器有总体了解的读者可跳过第一章。但是，所有读者都应阅读 **“1.3 写在程序开发之前”一节。**.

[规则]

本手册使用的通用符号的含义如下。

...:	重复同一格式或式样
[ ]:	在方括号内的字符可省略
「 」	在该括号内的字符是字符串。
‘’ :	单引号之间的字符是一个字符串。
“” :	双引号标志表示引用的位置。
Δ :	表示一个或多个空格字符或制表符。
黑体	:黑体字符表示重点显示。
__	:下划线用于表示输入字符串
: :	表示省略程序的描述
( )	:圆括号内的字符是一个字符串
CR	:回车
LF	:换行
/ :	分隔符
α :	作为记忆操作符输入，如一个寄存器名
β :	作为记忆操作符输入，如一个寄存器名
γ :	作为记忆操作符输入，如一个寄存器名
δ	:作为记忆操作符输入，如一个寄存器名

[相关文档]

下面的表格显示了这本手册的相关文档(如用户手册)。在出版物中出现的相关资料可能会包括初稿版本。但是，并未对初稿版本作特殊标注。

开发工具的相关文档（用户手册）

文档名称		文档编号.
CC78K0S Ver. 2.00 C编译器	操作篇	U17416E
	语言篇	U17415E
RA78K0S Ver. 2.00 汇编包	操作篇	U17391E
	语言篇	U17390E
	结构化汇编语言	本手册I
SM+ 系统仿真器	操作篇	U18601E
	用户开放接口	U18602E
SM78K 系列 Ver. 2.52系统仿真器	操作篇	U16768E
ID78K0S-NS Ver. 2.52集成调试器	操作篇	U16584E
ID78K0S-QB Ver. 3.00集成调试器	操作篇	U17287E
PM+ Ver. 6.30 项目管理器		U18416E

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

[备忘录]

目录

图形列表

图形编号. 标题, 页码

图表列表

图表编号. 标题 , 页码

第一章　　概述

本章介绍结构化汇编预处理器在微控制器软件发展中的功能（角色）等 .

1.1　概述

RA78K0S 结构化汇编预处理器 (ST78K0S) 是 "RA78K0S 汇编包 " 中的程序，它用于 78K0S 系列中微控制器的软件开发 .

ST78K0S 将诸如 "if~else~endif" 和 "for~next" 这样的汇编语句转换为汇编语言源程序 . 控制语句用于描述 “if~else~endif” 和 “for~next”.

同样地，结构化汇编器具有如下三种优势 .

(1) 程序容易编写
- 每个程序的结构都可以按照从设计到编码的开发过程模式进行编写 .
- 无需考虑分支的标号名称 .
- 包含大量代码的转移指令可以作为赋值语句编写 .

(2) 程序容易阅读 .
- 程序结构易于理解 .
- 存储器寄存器之间的操作和转移可通过一条简单语句完成 .
- 其它程序员编写的程序也易于阅读 .
- 程序维护（修订）容易 .

(3) 便于桌面调试
- 编码可以根据具体设计一对一的完成，从而方便了桌面调试 .

1.2 功能概述

ST78K0S 在根据特定语言规范编码的结构化汇编器源程序内部分析不同的控制语句 , 表达式和指令 , 并输出汇编源程序作为汇编器的输入文件 .

结构化语句可作为注释输出，已转换的汇编指令和普通汇编语言均可输出为辅助源文件 .

当发生错误时会输出错误信息 .

图 1-1 ST78K0S 功能

1.2.1 主要功能

(1) 大量的 类 -C 控制语句方便了程序编码 .

(2) 类 -C 赋值语句和赋值操作符可用于编码 .

(3) 控制结构和赋值语句可用于位处理 .

(4) 它包含类 -C 符号定义指令 , 条件处理函数和包含指令 .

(5) 因为它是输出汇编源程序的预处理器 , 所以在 ST78K0S 转换之后可进行编码优化 .

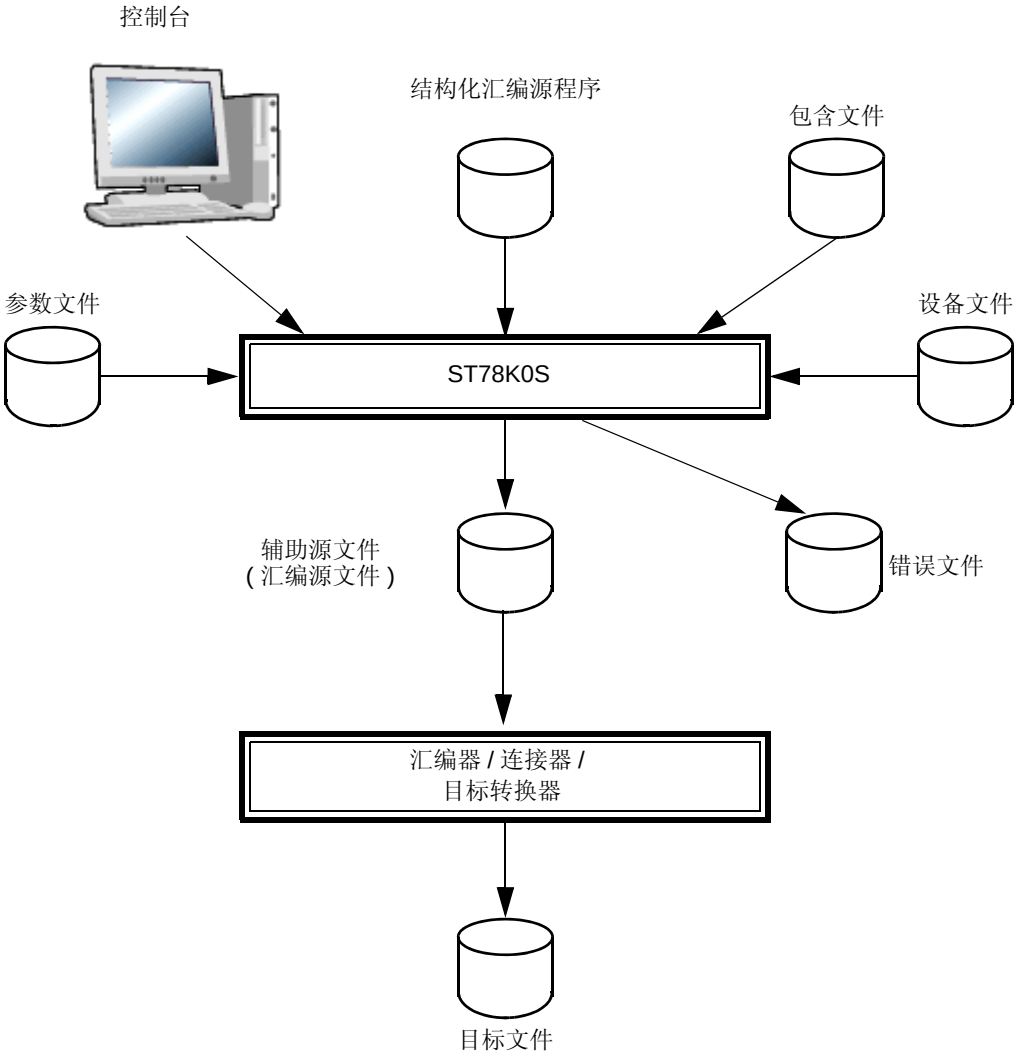
(6) 使用可转换为 CALLT 指令的指令 , 可在程序开发时将处理函数注册到 CALLT 表中 .

(7) 通过改变汇编源程序的位置可以创建易于阅读的汇编列表 .

1.2.2 程序开发流程

图 1-2 所示为程序开发的流程 .

图 1-2 程序开发流程



备注 可通过从联机发送服务 (ODS) 下载来获取设备文件 , 访问以下网址 .

http://www.necel.com/micro/ods/eng/tool/DeviceFile/list.html

1.3 写在程序开发之前

ST78K0S 的最高性能和需要注意的地方如下所述 .

1.3.1 最高性能

表 1-1 ST78K0S 的最高性能

项目	最大值
行长度 (不包括 LF 或 CR)	2048 个字符
#define 伪指令中注册的符号数 (不包括保留字)	512 个符号
#define 伪指令中注册的符号字符长度	31 个字符
控制语句的嵌套级数	31 级
#ifdef 伪指令中的嵌套级数	8 级
#defcallt 伪指令	32
#include 伪指令的嵌套	不支持
被 #define 伪指令重新定义的次数	31 次
一个序列中赋值的操作数的个数	33 (注 1)
逻辑运算符的操作数	17 (注 2)
由选项 "-d" 定义的符号数	30
由 -i 选项指定的包含文件路径数	64

注 1. 最大值表示如下 .

S1 = S2 = ... S32 = S33

最多可插入 33 个符号和 32 个等号 (=) .

注 2. 最大值表示如下 .

表达式 1&& 表达式 2&& ... && 表达式 16&& 表达式 17

最多可插入 17 个表达式和 16 个 "&&" (或 "||") 号 .

1.3.2 字符号和字节符号

ST78K0S 根据每个用户符号的最后一个字符来确定对应的符号是字符号还是字节符号 . 字符号的默认字符是 "-scp", 可以通过 -sc 选项更改 .

关于 -sc 选项的细节 , 参见 RA78K0S 汇编包操作用户手册 .

< 示例 1 >

结构化汇编源程序

```
SYM  = #3
SYMP = #3
```

→

汇编源程序

```
MOV     SYM , #3
MOVW    SYMP , #3
```

< 示例 2 > ST78K0S 的开始命令

```
C> ST78K0S INPUT.S -sc@
```

"@" 用作指示字符号的字符 .

输入文件说明

ST78K0S 命令名

结构化汇编源程序

```
SYMP = #3
SYM@ = #3
```

→

汇编源程序

```
MOV     SYMP , #3
MOVW    SYM@ , #3
```

1.3.3 定义标签

当定义标签 (通过汇编器表示地址的符号) 时 , 确保在 ST78K0S 语句的单独一行输入标签定义 .

< 不正确的编码示例 >

```
SYMBOL :        AX = #10H
```

< 正确的编码示例 >

```
SYMBOL :
        AX = #10H
```

第二章　　源程序编码方法

本章介绍源程序的编码方法等 .

2.1　基本配置

源程序由结构化汇编语言和 (纯) 汇编语言组成 .

关于汇编语言进一步的描述 , 请参见 RA78K0S 汇编包语言用户手册 .

每行 (两个 LF 之间) 最多可包含 2048 个字符 .

结构化汇编语言中使用的编码类型如表 2-1 中所示 .

表 2-1　结构化汇编语言编码

类型			编码
ST78K0S 语句	控制语句	条件分支	if ~ elseif ~ else ~ endif if_bit ~ elseif_bit ~ else ~ endif switch ~ case ~ default ~ ends
		条件循环	for ~ next while ~ endw while_bit ~ endw repeat ~ until repeat ~ until_bit
		其它	break, continue, goto
	表达式	赋值语句	赋值 (=), 赋值加操作 (+=, etc.), 移位 (循环) 赋值 (>>=, etc.)
		计数语句	增量 (++), 减量 (--)
		交换语句	交换 (<->)
		位操作语句	置位 (=), 清位 (=)
条件表达式		比较表达式	==, !=, <, >, >=, <=
		测试位表达式	位符号 , ! 位符号
		逻辑运算	逻辑 AND (&&), 逻辑 OR (||)

(1) 控制语句

控制语句包括如下语句：表示条件分支的 "if ~ elseif ~ else ~ endif", "if_bit ~ elseif_bit ~ else ~ endif" 和 "switch ~ case ~ default ~ ends" 语句 , 表示条件循环的 "for ~ next", "while ~ endw", "while_bit ~ endw", "repeat ~ until" 和 "repeat ~ until_bit" 语句 , 以及表示循环退出处理的 "break", "continue" 和 "goto" 语句 . 关于细节 , 请参见 " 第三章 控制语句 ".

(2) 表达式

表达式包括赋值语句 , 计数语句 (增量和减量), 交换语句和位操作语句 . 关于细节 , 请参见 " 第四章 表达式 ".

(3) 条件表达式

条件表达式包括比较表达式 , 测试位表达式和逻辑表达式 . 关于细节 , 请参见 "3.5 条件表达式 ".

2.2 元素

(1) 字符集

字母 , 数字和特殊字符都可在源程序中使用 .

表 2-2 字母数字字符

名称		字符
数字		0 1 2 3 4 5 6 7 8 9
字母	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

在 ST78K0S 中 , 只有控制语句的首字符区分大小写 . 在首字符之后出现的任何小写字母都将转换为大写字母 . 但是 , 辅助源文件按照输入时的大小写规定输出 .

表 2-3 特殊字符

字符	名称	用途
?	问号	用作字母的字符
@	单价符号	用作字母的字符
_	下划线	用作字母的字符
	空格	短语分隔符
HT	水平制表符	用作空格的字符
,	逗号	操作数的分隔符
.	句号	位符号的比特位置符
"	双引号	#INCLUDE 指令的磁盘类型文件名称的说明字符
'	单引号	标记字符常量起始或结束的符号
+	加号	正号或加法操作
-	符号	符号或减法操作
&	与符号	逻辑 AND 操作符
|	或符号	逻辑 OR 操作符
^	上箭头符号	异或操作符
(	左圆括号	改变控制语句中的操作或表达式的顺序
)	右圆括号	
=	等号	赋值操作符 , 比较操作符
:	冒号	标记分隔符
;	分号	注释起始符号或控制语句表达式中的分隔符
#	算术符号或高音符号	ST78K0S 指令的首字符或立即显示符号
$	美元符号	位置或计数器的值 控制指令中的显示符号
!	感叹号	直接寻址说明符 , 拒绝显示符号
<	不等于 (小于) 符号	比较操作符
>	不等于 (大于) 符号	
\	后斜线	目录说明符
[	左中括号	间接地址说明符
]	右中括号	
LF	换行	行结束符号

如果键入以下的任意无效字符，将产生错误 .

表 2-4 无效字符

类型	ASCII 码
无效字符	00H to 08H, 0BH, 0CH, 0EH to 1FH, 7FH
无法识别的特殊字符	% (25H), ' (60H), { (7BH), } (7DH), ¯ (7EH)
其它字符	80H ~ 0FFH

当键入无效字符时会发生错误 , 且输出辅助文件时每个无效字符都会被一个句号 (.) 代替 .

但是 , 无效字符可在注释中使用 .

(2) 标识符

标识符是附着在数字数据 , 地址等之上的名称 .

标识符使源程序中的内容更容易识别 .

使用 #define 语句定义标识符的细节 (参见 "5.2 指令功能 ").

(3) 符号

符号名称中的最末字符决定 ST78K0S 产生一个字节访问指令还是字访问指令 . 默认设置为 P (pair), 该设置可通过 -sc 改变 .

除了保留字之外的所有字符串都可作为用户符号处理 . 所有字母数字字符和可用英语字母表中的字符建立的所有其它字符都可用作用户符号 .

(4) 常量

结构化汇编语言不包含常量 . 但是 , 汇编语言常量可按照辅助文件的方式输出 (关于汇编语言常量的细节 , 参见 RA78K0S 汇编包语言用户手册).

(5) 表达式

表达式是用操作符组合在一起的常量 , 特殊字符和符号 (关于汇编语言表达式的细节 , 参见 RA78K0S 汇编包语言用户手册).

确保在一个汇编语言表达式内任何被空格分开的符号都包含在圆括号内 .

< 示例 >

- 汇编程序编码方法

```
MOV     A , # ( SYM AND 0FFH )
MOV     A , LABEL + 1
```

- ST78K0S 的编码方法

```
A = # ( SYM AND 0FFH )
A = ( LABEL + 1 )
```

2.3 保留字

表 2-5 列出了结构化汇编语言中的保留字 .

关于指令和 sfr 标号 , 参见目标设备用户手册 .

表 2-5 保留字

类型	保留字
控制语句	IF, IF_BIT, ELSEIF, ELSEIF_BIT, ELSE, ENDIF
	SWITCH, CASE, DEFAULT, ENDS
	FOR, NEXT
	WHILE, WHILE_BIT, ENDW
	REPEAT, UNTIL, UNTIL_BIT
	BREAK, CONTINUE, GOTO
指令	DFINE
	IFDEF, ELSE, ENDIF
	INCLUDE
	DEFCALLT, ENDCALLT
操作符	++, --
	=, +=, -=, *=, /=, &=, |=, ^=, <<=, >>=, <->
	==, !=, <, >=, >, <=, FOREVER
汇编操作符	MOD, NOT
	AND, OR, XOR
	EQ, NE, GT, GE, LT, LE
	SHL, SHR
	HIGH, LOW, BANKNUM
	DATAPOS, BITPOS, MASK

表 2-5 保留字

类型	保留字
汇编控制指令	PROCESSOR, PC
	DEBUG, NODEBUG, DEBUGA, NODEBUGA, DG, NODG
	XREF, XR, NOXREF, NOXR
	TITLE, TI
	SYMLIST, NOSYMLIST
	FORMFEED, NOFORMFEED
	WIDTH, LENGTH
	TAB
	KANJICODE
	IC
	EJECT, EJ
	LIST, LI, NOLIST, NOLI
	GEN, NOGEN
	COND, NOCOND
	SUBTITLE, ST
	SET, RESET
	_IF, _ELSEIF, IF, ELSEIF, ELSE, ENDIF
寄存器	CY, Z
	A, X, B, C, D, E, H, L
	R0, R1, R2, R3, R4, R5, R6, R7
	PSW
	AX, BC, DE, HL
	RP0, RP1, RP2, RP3
	SP
其它	DGS, DGL, TOL_INF, SJIS, EUC, NONE

2.4 标签生成规则

在汇编语言指令中使用控制语句时 , ST78K0S 为分支指令产生标签 .

由 ST78K0S 生成的标记具有 "?Ldddd" 格式 .

"dddd" 表示 1 或大于 1 的十进制数值 , 输出时抑制 0 和左对齐 . 因此 , 使用该 "?Ldddd" 格式时不要键入任何标签 .

2.5 空间规格

空间规格可以用来改变在赋值表达式，条件表达式或 switch 语句的 case 记号的左侧或右侧键入的符号数据大小 .

(1) 编写格式

(Δ size specification character Δ)

(2) 功能

- 如果空间字符为 "B" 或 "b", 则数据大小变为字节 .

(3) 说明

- 如果空间规格的字符不正确则会出现错误 .
- 如果在不支持空间规格的赋值表达式或条件表达式中输入空间规格，则会出现错误 .
- 如果将空间规格用于寄存器，则只能使用同一空间规格来规定编码 . 数据大小不能改变 . 如果数据大小不同，则出现错误 .
- 当指定用户符号时，确保将数据大小更改为指定值 .
- 如果空间规格已在直接访问规格符号或者间接访问规格符号或立即数上输入，则将忽略该空间定义，且数据大小不会改变 .
- 不能在空间定义中指定字访问 .

2.6 数据空间

由于根据生成的指令不同符号也不同 , ST78K0S 需要检查符号的数据空间 . 另外 , ST78K0S 允许汇编程序来确定符号定义和常量是否正确输入 .

由 ST78K0S 所检查的数据空间如下表所示 .

表 2-6 数据空间

生成的指令表中的符号	说明
a	CY
b	位符号 (不包括 [HL].β) ST78K0S 将以 "α, β" 格式输入的比特 sfr 和标号识别为位符号 . 可作为 "α" 输入的项包括字节用户符号 , 字用户符号，指定字节的用户符号 , sfr, A, PSW 和常量 . 可作为 "β" 输入的项包括字节用户符号 , 字用户符号和常量 .
c	[HL].β 可作为 "β" 输入的项包括字节用户符号 , 字用户符号和常量 .
d	字节用户符号
e	字节指定覆盖 saddr 区域的用户符号以及 sfr
f	A
g	字节寄存器 (除了 A, R0 和 R1)
h	R0
i	R1
j	sfr
k	PSW
l	字用户符号
m	覆盖 saddrp 的 sfrp
n	AX
o	字寄存器 (除了 AX 和 RP0)
p	RP0
q	sfrp
r	SP
s	直接访问说明符号 这些是使用 "!addr" 格式指定的符号 . 字节用户符号 , 字用户符号 , 常量和 $ 都可以作为 "addr" 输入 .
t	间接访问说明符号 这些是使用 "[HL]" 和 "[HL + byte]" 格式指定的符号 . 字节用户符号 , 常量和 $ 都可以作为 "byte" 输入 .

表 2-6 数据空间

生成的指令表 中的符号	说明
u	特殊间接访问说明符 这些是使用 "[DE]" 格式指定的符号 .
v	立即数 这些是使用 "#date" 格式指定的符号 . 字节用户符号 , 字用户符号 , 常量和 $ 都可作为 "date" 输入 .

2.7 注释

任何在分号 (;) 之后 , 下一换行符 (LF) 之前出现的字符串都被看作是注释语句 , 不需要对其进行处理 , 仅输出至辅助文件 . 注释语句可在一行代码的任意位置输入 .

但是 , 由于在 “for~next” 语法中 , 分号用在圆括号之间作为的表达式分隔符使用 , 因此在圆括号之间输入的两个分号不作为注释语句的起点 .

在 “2.2 (1) 字符集 " 中列出的所有字符都可用于注释 .

当注释或注释语句中包含无效字符时，无效字符不被处理 .

2.8 工具信息

ST78K0S 输出工具信息 .

如果输入源文件包含的工具信息已由 ST78K0S 输出 , 则位于信息起始处的 “$” 字符由 “;” 代替 .

输出位置在模块头的最后 . 在被模块头可以输入的语句类型只有汇编控制指令 , 注释语句和换行符 .

(1) 输出格式

```
$TOL_INF        2FH , 第二个参数 , 第三个参数 , 0FFFFH
```

2FH 表示这是由 ST78K0S 预处理器输出的工具信息 .

第二个参数表示该预处理器的版本号 .

版本号或者作为十六进制数值输出 . 或者 , 如果该值没有经过转换 , 则输出已在启动时显示的十进制数字 .

< 示例 >

版本号 3.10 -> 310H

第三个参数用来表示预处理器的错误信息 .

0H : 正常结束

1H : 致命错误 , 退出

2H : 警告 , 退出

3H : 致命错误和警告 , 退出

0FFFFH 表示和语言相关的信息 . 它是该预处理器的固定值 .

2.9 ST78K0S 输入源文件的输出结果

ST78K0S 按照如下规则输出已输入的源文件 .

表 2-7 ST78K0S 输出

输入源程序文件	辅助源程序文件
ST78K0S 控制语句 ST78K0S 表达式语句	作为注释输出
ST78K0S 指令	无输出
#INCLUDE	输出 include 内容
由 #IFDEF 设置的源文件别名	无输出
注释	作为注释输出
其它行	原样输出

第三章　　控制语句

本章介绍有关控制语句功能的例子 .

控制语句用于在结构上控制程序的流向 （可参见 "3.4 控制语句功能 "） .

3.1　控制语句字符

由控制语句生成的指令的差异基本上取决于控制语句中是使用大写字母还是小写字母 . 例如 , 通过条件表达式生成的条件分支指令 ,"if ~ endif" 和 "IF ~ ENDIF" 之间的不同语句大小可以直接排除分支 .

然而 , 确保语句总能跳至正确分支却降低了目标程序效率 .

为了解决这个问题 , 用户可以设置大小写以提高目标效率 . 如果不需要提高目标效率 , 则只要编码使用大写字母 , 用户就可以不改变字符的大小 .

由于控制语句生成条件分支指令 , 所以务必指定相关地址是否在 128 字节之内 .

在控制语句中 ,"if" 和 "elseif" 都是保留字 . ST 78K0S 确定控制语句保留字的首字符是大写字母还是小写字母 .

IF, If :	首字母大写 , 因此编码被确定为大写 .
if, iF :	首字母小写 , 因此编码被确定为小写 .
如果用大写输入：	结合使用条件分支指令和 BR 伪指令的分支 .
如果用小写输入：	直接使用条件伪指令的分支 .

成对的控制语句 （例如 "if, else, endif"）中允许有大小写字母的混合 . 换句话说 , 可以写成 "IF ~ else ~ ENDIF".

3.2　嵌套

控制语句可以嵌套．通常，最多允许嵌套 31 级．但是，控制语句不能交叉．

图 3-1　嵌套举例

< 错误编码举例 >

```
while ( A < B )
        if ( A == #4 )
                break ;
endw
        endif
```

发生由于交叉引起的错误．

< 正确编码举例 >

```
while ( A < B )
        if ( A == #4 )
                break ;
        endif
endw
```

"if" 语句被正确嵌套在 "while" 语句中．

3.3　寄存器指定

(1) 描述格式

([Δ][=][Δ]寄存器名[Δ])

(2) 功能

- 如果寄存器在比较表达式后立即指定

  在指令将左边的数值传输至指定的寄存器后，会生成一个比较表达式，用来对指定的寄存器和右边的数值进行比较。

< 示例 >

输出源	输入源

CMP SYM1 , #5
BZ $?L1
CMP SYM2 , #0
BC $?L1
MOV A , SYM3
CMP A , #80H
BNC $?L1
?L1 :
	if (SYM1 != #5 && SYM2 >= #0&&SYM3 < #80H (A))
	
	endif

- 如果寄存器在控制语句后被指定

  在各比较表达式生成期间，在将左边的数值发送至指定寄存器的指令生成之后，会生成一个比较表达式，用来对指定寄存器和右边的数值进行比较。

< 示例 >

输出源	输入源

MOV A , R4
CMP A , #5
BZ $?L2
MOV A , R2
CMP A , #0
BC $?L2
MOV A , R3
CMP A , #80H
BNC $?L2
?L2 :
	if (R4 != #5 && R2 >= #0 && R3 <# 80H) (A)
	
	endif

- 如果 (a) 和 (b) 同时指定

  紧跟在比较表达式之后的寄存器指定优先．在生成将左边的数值发送至指定寄存器的指令之后，会生成一个比较表达式，用来对指定寄存器和右边的数值进行比较．

  对于在比较表达式之后没有寄存器被立即指定的表达式，在根据控制语句之后的寄存器指定生成了将左边数值发送至指定寄存器的指令之后，会生成一个比较表达式，用来对指定寄存器和右边的数值进行比较．

< 示例 >

输出源	输入源

MOV A , DATA1
CMP A , #5
BZ $?L3
MOV A , DATA2
CMP A , #0
BC $?L3
MOV A , DATA3
CMP A , #80H
BNC $?L3
?L3 :
	if (DATA1!=#5 && DATA2 >=#0 (A) && DATA3<#80H)
	(A)
	
	endif

(3) 说明

- 寄存器指定可用于 if 语句、 elseif 语句、 switch 语句、 for 语句、 while 语句和 until 语句．但是，如果条件表达式是一个位表达式，则忽略在控制语句中指定的任何寄存器．
- 有关寄存器名的列表，参见表 2-5.

  还可以输入 sfr 指定．
- for 语句中赋值语句的处理与比较表达式的处理过程相同．

3.4　控制语句功能

以下内容介绍了各种控制语句的功能 .

所使用的例子作为源文件的注释语句，生成的指令都输入到该源文件中 .

表 3-1　控制语句列表

类型	描述	备注
条件分支	if ~ elseif ~ else ~ endif	
	if_bit ~ elseif_bit ~ else ~ endif	
	switch ~ case ~ default ~ ends	
条件循环	for ~ next	加指定循环
	while ~ endw	处理前判断条件表达式的循环
	while_bit ~ endw	处理前判断条件表达式的循环
	repeat ~ until	处理后判断条件表达式的循环
	repeat ~ until_bit	处理后判断条件表达式的循环
	break	中断循环块
	continue	继续循环块
	goto	退出去执行异常处理

3.4.1　条件分支

条件分支 if

(1)　if ~ elseif ~ else ~ endif

[描述格式]

```
[ Δ ] if [ Δ ] ( 条件表达式 1 ) [ Δ ] [ ( 寄存器名 ) ]
        if 程序块
[ Δ ] elseif [ Δ ] ( 条件表达式 2 ) [ Δ ] [ ( 寄存器名 ) ]
        elseif 程序块
[ Δ ] else
        else 程序块
[ Δ ] endif
```

[功能]

- if ~ endif

  如果条件表达式 1 为真 , 则执行 if 程序块 .

  If 程序块可能占用多行 .

- if ~ else ~ endif

  如果条件表达式 1 为真 , 则会执行 if 程序块 , 如果条件表达式 1 为假 , 则执行 else 程序块 .

  If 程序块和 else 程序块可能占用多行 .

- if ~ elseif ~ else ~ endif

  可以为单个 if 语句编写多个 elseif 程序块 .

  如果条件表达式 1 为真 , 则执行 if 程序块 . 如果为假 , 则判断条件表达式 2.

  如果条件表达式 2 为真 , 则执行 elseif 程序块 . 如果为假 , 则判断下一个 endif 之前的任何其它 elseif 的条件 .

  如果没有 elseif 语句 , 则执行 else 程序块 .

  If 程序块、 elseif 程序块和 else 程序块可能占用多行 .

[说明]

- 比较表达式、逻辑表达式和位校验表达式可以在条件表达式中输入 . 如果指定了寄存器名 , 则判断条件时使用指定的寄存器 .

  有关比较表达式和逻辑表达式的细节 , 参见 "3.5 条件表达式 ".

- 当为一个条件编码两个分支时 , 使用 if ~ else ~ endif.
- 当为某个范围的值编码多个分支时 , 使用 if ~ elseif ~ else ~ endif. 这与 switch 语句不同 , 后者包含一定范围内的值 .
- 可以省略 elseif 语句和 else 语句 , 也可以输入多个 elseif 语句 .

[生成的指令]

(1) 处理 if (条件表达式)

- 生成一个指令以判断条件表达式的条件 .
- 如果条件不满足 , 则生成一个分支指令以跳转到 elseif 程序块或 else 程序块 .

(2) 处理 elseif (条件表达式)

- 为 endif 语句生成一个分支指令 .
- 为由 if 语句生成的分支指令生成一个标签 .
- 生成一个指令以判断条件表达式的条件 .
- 如果条件不满足 , 则生成一个分支指令以跳转到 elseif 程序块或 else 程序块 .

(3) 处理 else

- 为 endif 语句生成一个分支指令 .
- 为由 if 语句或 elseif 语句生成的分支指令生成一个标签 .

(4) 处理 endif

- 为由 if 语句、 elseif 语句或 else 语句生成的分支指令生成一个标签 .

(5) 补充说明

- 这些程序块可以用 elseif_bit 混合使用 .

[使用示例]

(1) 采用小写字母输入时

输出源	输入源

`CMP A , #0`
`BNZ $?L1`
`BF TFLG.0 , $?L2`
`SET1 CY`
`BR ?L3`
`?L2 :`
`CLR1 CY`
`?L3 :`
`MOVW AX , #0FFH`
`BR ?L4`
`?L1 :`
`MOVW BC , #0A00H`
`?L4 :`
	`if ( A == #0 )`
	`CY = TFLG.0`
	
	`AX = #0FFH`
	
	`else`
	`BC = #0A00H`
	`endif`

(2) 采用大写字母输入时

输出源	输入源

CMP A , #0
BZ $?L5
BR ?L6
?L5 :
BF TFLG.0 , $?L7
SET1 CY
BR ?L8
?L7 :
CLR1 CY
?L8 :
MOVW AX , #0FFH
BR ?L9
?L6 :
MOVW BC , #0A00H
?L9 :
	IF (A == #0)
	
	CY = TFLG.0
	
	AX = #0FFH
	
	ELSE
	BC = #0A00H
	ENDIF

条件分支 if_bit

(2) if_bit ~ elseif_bit ~ else ~ endif

[描述格式]

```
[ Δ ] if_bit [ Δ ] ( 位校验表达式 1 )
        if_bit 程序块
[ Δ ] elseif_bit [ Δ ] ( 位校验表达式 2 )
        elseif_bit 程序块
[ Δ ] else [ Δ ]
        else 程序块
[ Δ ] endif [ Δ ]
```

[功能]

- if_bit ~ endif

  如果条件表达式 1 为真 , 则执行 if_bit 程序块 .

  if_bit 程序块可能占用多行 .

- if_bit ~ else ~ endif

  如果条件表达式 1 为真 , 则会执行 if_bit 程序块 , 如果条件表达式 1 为假 , 则执行 else 程序块 .

  if_bit 程序块和 else 程序块可能占用多行 .

- if_bit ~ elseif_bit ~ else ~ endif

  如果条件表达式 1 为真 , 则执行 if_bit 程序块 . 如果为假 , 则判断条件表达式 2. 如果条件表达式 2 为真 , 则执行 elseif_bit 程序块 . 如果为假 , 则判断下一个 endif 之前的任何 elseif_bit 的条件 .

  如果没有 elseif_bit 语句 , 则执行 else 程序块 .

  if_bit 程序块、 elseif_bit 程序块和 else 程序块可能占用多行 .

- 补充说明

  这些程序块可以用 elseif 混合使用 .

[说明]

- 位校验表达式作为条件表达式 1 和 2 输入 .

  有关位校验表达式的详细内容 , 参见 "3.5 条件表达式 ".

- 当为一个条件编码两个分支时使用 if_bit ~ else ~ endif.

  当为多重分支检测多个位符号时 , 使用 if_bit ~ elseif_bit ~ else ~ endif.

- 可以省略 elseif_bit 语句和 else 语句 , 还可以键入多个 elseif_bit 语句 .

[生成的指令]

(1) 处理 if_bit (位条件)

- 为位条件生成一个真 / 假指令 .

(2) 处理 elseif_bit (位条件)

- 为 endif 语句生成一个分支指令 .
- 为由 if_bit 语句生成的分支指令生成一个标签 .
- 为位条件生成一个真 / 假指令 .

(3) 处理 else

- 为 endif 语句生成一个分支指令 .
- 为由 if_bit 语句或 elseif_bit 语句生成的分支指令生成一个标签 .

(4) 处理 endif

- 为由 if_bit 语句、 elseif_bit 语句或 else 语句生成的分支指令生成一个标签 .

[使用示例]

(1) 采用小写字母输入时

输出源	输入源

BT TRFG.0 , $?L1
SET1 PRTYFLG.3
BR ?L2
?L1 :
BF PGF.0 , $?L3
MOVW BC , #0FFH
BR ?L2
?L3 :
MOV A , # (FG SHR 6)
MOV H , A
BF PGF.0 , $?L4
SET1 CY
BR ?L5
?L4 :
CLR1 CY
?L5 :
CLR1 BUSYFG.2
?L2 :
	if_bit (!TRFG.0)
	PRTYFLG.3=1
	elseif_bit (PGF.0)
	BC = #0FFH
	else
	H = # (FG SHR 6) (A)
	CY = PFG.0
	BUSYFG.2 = 0
	endif

(2) 采用大写字母输入时

输出源	输入源

BF TRFG.0 , $?L6
BR ?L7
?L6 :
SET1 PRTYFLG.3
BR ?L8
?L7 :
BT PGF.0 , $?L9
BR ?L10
?L9 :
MOVW BC , #0FFH
BR ?L8
?L10 :
MOV A , # (FG SHR 6)
MOV H , A
BF PFG.0 , $?L11
SET1 CY
BR ?L12
?L11 :
CLR1 CY
?L12 :
CLR1 BUSYFG.2
?L8 :
	IF_BIT (!TRFG.0)
	
	PRTYFLG.3 = 1
	
	ELSEIF_BIT (PGF.0)
	
	BC = #0FFH
	
	ELSE
	H = # (FG SHR 6) (A)
	
	CY = PFG.0
	
	BUSYFG.2 = 0
	ENDIF

条件分支 switch

(3) switch ~ case ~ default ~ ends

[描述格式]

```
[ Δ ] switch [ Δ ] ( [ Δ ] case 符号 [ Δ ] ) [ Δ ] [ ( 指定寄存器 ) ]
[ Δ ] case [ Δ ] 常量 :
        语句_1
[ [ Δ ] case [ Δ ] 常量 :
        语句_2 ]
[ Δ ] [ default : ]
        语句_N
[ Δ ] ends
```

[功能]

- 如果 case 符号的值与 case 常量匹配 , 则执行指定的语句 .
- 如果 case 符号的值与任何 case 常量都不匹配 , 且已输入默认语句 , 则会执行默认语句 .
- 通常 , 必须键入 break 语句以跳出 switch 程序块 .

[说明]

- 对 "case 符号 " 可能的指定取决于目标设备的汇编语言 .
- 如果没有键入 break 语句 , 则下一个 case 语句执行比较指令 .
- 常量可表示为二进制、八进制、十进制、十六进制或字符串常量 .

  但是 , 由于 ST78K0S 把常量识别为字符串 , 因此 , 单独使用可能被汇编器识别为字符串的常量时要倍加谨慎 .
- 仅当设置了寄存器指定时 ,case 字符才会传输至指定寄存器 .

[生成的指令]

(1) 处理 switch 语句

(a) 如果没有指定寄存器 , 则判断 case 字符 , 如有必要 , 则生成传输至 A 或 AX 的指令 .

(b) 如果已指定了寄存器 , 则将 case 符号传输至指定寄存器 .

但是 , 如果不能生成比较指令 , 则将出错 .

有关细节 , 参见表 3-2.

(2) 处理 case 语句

(a) 由处理其它 case 语句的分支中生成标签 .

(b) 生成 CMP 或 CMPW. 如果指定的常量不匹配 , 则会为其它 case 语句、默认语句或 ends 语句生成一个分支指令 .

?LTRUE : 指定常量匹配时分支目的标签

?LFALSE : 指定常量不匹配时分支目的标签

- 如果 case 语句用小写字母表示 , 且在 switch 语句中没有设置寄存器指定

```
CMP ( W )       case 符号 , #case 常量
BNZ             $?LFALSE
```

- 如果 case 语句用小写字母表示 , 且在 switch 语句中设置了寄存器指定

```
CAMP ( W )      指定寄存器 , #case 常量
BNZ             $?LFALSE
```

- 如果 case 语句用大写字母表示 , 且在 switch 语句中没有设置寄存器指定

```
        CMAP ( W )      case 符号 , #case 常量
        BZ              $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

- 如果 case 语句用大写字母表示 , 且在 switch 语句中设置了寄存器指定

```
        CAMP ( W )      指定寄存器 , #case 常量
        BZ              $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

(3) 处理 default 语句

从 case 语句为分支指令生成一个标签

(4) 处理 ends 语句

从 case 语句或 break 语句为分支指令生成一个标签

表 3-2　Switch 语句的生成指令

Case 符号		无寄存器指定	带寄存器指定												
			a	b	f	g	h	i	j	k	n	o	p	q	r
a	CY														
b	位符号														
c	[HL].β														
d	字节用户符号	*3			*1						*2				
e	字节数据	*3			*1										
f	A	*3													
g	字节寄存器	*1			*1										
h	R0	*1			*1										
i	R1														
j	sfr	*1			*1										
k	PSW	*1			*1										
l	字用户符号				*1						*2				
m	字数据	*2									*2				
n	AX	*3													
o	字寄存器	*2									*2				
p	RP0														
q	sfrp														
r	SP	*2									*2				
s	直接访问符号	*1			*1										
t	间接访问符号	*1			*1										
u	[DE]	*1			*1										
v	立即符号	*1			*1						*2				

*1 :　生成 MOV 指令

*2 :　生成 MOVW 指令

*3 :　不产生转移指令

空栏表示错误 .

[使用示例]

(1) 采用小写字母输入时

输出源	输入源
MOV A , R0 CMP A , #1 BNZ $?L1 BF P1.0 , $?L2 BTM.3 ?L2 : BR ?L3 ?L1 : CMP A , #2 BNZ $?L4 BR ?L3 ?L4 : CMP A , #3 BNZ $?L5 BR ?L3 ?L5 : ?L3 :	SWITCH (R0) case 1 : if_bit (P1.0) BTM.3 endif break case 2 : break case 3 : break default : ENDS

(2) 采用大写字母输入时

输出源	输入源
MOV A , R0 CMP A , #1 BZ $?L6 BR ?L7 ?L6 : BF P1.0 , $?L8 BTM.3 ?L8 : BR ?L9 ?L7 : CMP A , #2 BZ $?L10 BR ?L11 ?L10 : BR ?L9 ?L11 : CMP A , #3 BZ $?L12 BR ?L13 ?L12 : BR ?L9 ?L13 : ?L9 :	SWITCH (R0) CASE 1 : if_bit (P1.0) BTM.3 endif break CASE 2 : break CASE 3 : break DEFAULT : ENDS

3.4.2　条件循环

条件循环 for

(4)　for ~ next

[描述格式]

```
[ Δ ] for [ Δ ] ( [ 表达式 1 ] ; [ 表达式 2 ] ; [ 表达式 3 ] ) [ Δ ] [ ( 寄存器指定 ) ]
        指令组
[ Δ ] next
```

[功能]

- 初始值由表达式 1 设置 , 只要满足表达式 2 中的条件表达式 , 则执行语句和表达式 3. 通常 , 表达式 3 是递增（++）或递减 （--）操作 .

  其含义与下面的例子类似 .

```
表达式 1
while ( 表达式 2 )
         指令组
         表达式 3
endw
```

[说明]

⑴ 务必注意 , 与上文类似的例子不适用于生成指令 .

⑵ 在表达式 1、表达式 2 和表达式 3 中键入以下内容 .

表达式 1 : 初始值设置 （赋值表达式）

表达式 2 : 条件表达式

表达式 3 : 递增或递减表达式

⑶ 在表达式 1 或表达式 3 中可以输入赋值操作数和交换语句 , 但是这样做时 , 应检测转换输出 , 如有必要还需进行修改 .

⑷ 表达式 1、表达式 2 或表达式 3 有可能被省略 . 但是 , 如果省略表达式 2, 则将发生死循环 .

⑸ 在条件表达式中可以输入 "forever".

(3) 由于表达式 2 和表达式 3 控制 for ~ next, 所以这些表达式的内容不应该由可执行的语句改变 . 改变这些内容会导致错误操作 .

[生成的指令]

(1) 处理 for 语句 （表达式 1、表达式 2、表达式 3）

(a) 为表达式 1 生成指令 . 如果指定了寄存器名 , 则指定寄存器用于赋值和比较 .

(b) 为判断表达式 2 的条件语句生成一个分支指令 .

(c) 为由 next 语句生成的分支指令生成一个标签 .

(d) 为在 （b）中生成的分支指令生成一个标签 .

(e) 为表达式 2 生成一个条件判断指令 .

(2) 处理 next 语句

(a) 为通过 for 语句处理 （c）生成的标签生成一个分支指令 .

(b) 为跳出 for 程序块的分支指令生成一个标签 .

(c) 为表达式 3 的赋值表达式生成一个指令 .

(3) 补充说明

为了更有效的使用 for ~ next 语句 , 推荐如下方法 .

- 在表达式 1 和表达式 3 中 , 用 saddr 代替寄存器名作为控制变量 .
- 指定一个寄存器时 , 指定 A 或 AX.
- 当重复执行一个循环至少 256 次时 , 嵌套一个 for 语句并使用两个 saddr 变量作为控制变量 .

备注　推荐上述方法是因为 , 为了输出 CMP 或 CMPW 作为表达式 2 中条件表达式的生成指令 , 可作为操作数输入的符号范围有限 .

[使用示例]

(1) 采用小写字母输入时

输出源	输入源
MOV i , #0H ?L1 : CMP i , #0FFH BNC $?L2 CALL !XXX NC i BR ?L1 ?L2 :	for (i = #0H ; i < #0FFH ; i++) CALL !XXX next

(2) 采用大写字母输入时

输出源	输入源
MOV i , #0H ?L3 : CMP i , #0FFH BC $?L4 BR ?L5 ?L4 : CALL !XXX INC i BR ?L3 ?L5 :	FOR (i = #0H ; i < #0FFH ; i++) CALL !XXX NEXT

条件循环 while

(5) while ~ endw

[格式描述]

```
[ Δ ] while [ Δ ] ( 条件表达式 ) [ Δ ] [ ( 寄存器指定 ) ]
        指令组
[ Δ ] endw
```

[功能]

- 只要条件表达式保持为真 , 则重复执行指令组 .

[说明]

- 比较表达式、逻辑表达式、校验位表达式和 "forever" 都可能作为条件表达式输入 .

  如果输入 "forever", 则结果为一个死循环 .

- 将使用输入的比较表达式或者逻辑表达式的寄存器的格式指定为 " （条件表达式） ".
- 由于在执行指令组前判断条件表达式 , 如果第一个条件表达式为假 , 则指令组一次也不会执行 .

[生成的指令]

(1) 处理 while （条件表达式）语句

- 为由 endw 生成的分支指令生成一个标签 .
- 生成一个条件判断指令 . 如果指定了寄存器名 , 则在生成条件判断指令时使用指定寄存器 .
- 当条件判断为假时 , 为从 while 程序块中删除 while （条件表达式）语句生成一个分支指令 .

(2) endw

- 为执行循环生成一个分支指令 .
- 为分支指令生成一个标签 , 该指令用于从 while 程序块中删除 endw.

[使用示例]

(1) 采用小写字母输入时

输出源	输入源
?L1 : 　　CMPW　　AX , #0FFFH 　　BNC　　$?L2 　　MOV　　B , #0FH 　　INCW　　HL 　　BR　　?L1 ?L2 :	while (AX < #0FFFH) 　　B = #0FH 　　HL++ endw

(2) 采用大写字母输入时

输出源	输入源
?L3 : 　　CMPW　　AX , #0FFFH 　　BC　　$?L4 ?L4 : 　　MOV　　B , #0FH 　　NCW　　HL 　　BR　　?L3 ?L5 :	WHILE (AX < #0FFFH) 　　B = #0FH 　　HL++ ENDW

条件循环 while_bit

(6) while_bit ~ endw

[描述格式]

```
[ Δ ] while_bit [ Δ ] ( 位条件 )
        指令组
[ Δ ] endw
```

[功能]

- 只要位条件为真 , 就执行指令组 .

[说明]

- 由于在执行指令组前判断位条件 , 如果第一个位条件为假 , 则指令组一次也不会执行 .

[生成的指令]

(1) 处理 while_bit （位条件）语句

- 为由 endw 生成的分支指令生成一个标签 .
- 生成一条指令来判断位条件为真或假 .
- 当位条件判断为假时 , 为从 while_bit ~ endw 程序块中删除 while_bit 语句生成一个分支指令 .

(2) 处理 endw

- 为执行循环生成一个分支指令 .
- 为分支指令生成一个标签 , 该指令用于从 while_bit 程序块中删除 endw.

[使用示例]

(1) 采用小写字母输入时

输出源	输入源

?L1 :
　　BT　　TRFG.0 , $?L2
　　MOV　　A , PORT1
　　CMP　　A , #04H
　　BNZ　　$?L3
　　MOV　　X , #0FFH
　　BR　　?L4
?L3 :
　　CLR1　　PFG.0
?L4 :
　　BR　　?L1
?L2 :
	while_bit (!TRFG.0)
	
	　　A = PORT1
	　　if (A == #04H)
	
	　　X = #0FFH
	
	　　else
	　　PFG.0 = 0
	　　endif
	
	endw

(2) 采用大写字母输入时

输出源	输入源

?L5 :
　　BF　　TRFG.0 , $?L6
　　BR　　?L7
?L6 :
　　MOV　　A , PORT1
　　CMP　　A , #04H
　　BNZ　　$?L8
　　MOV　　X , #0FFH
　　BR　　?L9
?L8 :
　　CLR1　　PFG.0
?L9 :
　　BR　　?L5
?L7 :
	WHILE_BIT (!TRFG.0)
	
	　　A = PORT1
	　　if (A == #04H)
	
	　　X = #0FFH
	
	　　else
	　　PFG.0 = 0
	　　endif
	
	ENDW

条件循环 until

(7) repeat ~ until

[描述格式]

```
[ Δ ] repeat
        指令组
[ Δ ] until [ Δ ] ( 条件表达式 ) [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 只要条件表达式保持为真 , 则重复执行指令组 .

[说明]

- 比较表达式、逻辑表达式、校验位表达式和 "forever" 都可以作为条件表达式输入 .

  如果输入 "forever", 则结果为一个死循环 .

- 将使用输入的比较表达式或者逻辑表达式的寄存器的格式指定为 " （条件表达式） ".
- 在执行完指令组后对条件表达式进行判断 . 因此 , 如果第一个条件表达式为真 , 则执行一次指令组 .

[生成的指令]

(1) 处理 repeat 语句

- 为由 until 生成的分支指令生成一个标签 .

(2) 处理 until （条件表达式）语句

- 为条件表达式生成一个条件判断指令 .
- 为由 repeat 生成的标签生成一个分支指令 , 以便在 repeat ~ until 期间和条件表达式判断为假时执行指令组 . 如果条件表达式判断为假 , 则从 repeat 程序块中删除 until 语句 .

[使用示例]

(1) 采用小写字母输入时

输出源	输入源

?L1 :
　　MOVW　　AX , BC
　　CMP　　ABC , #0CH
　　BNZ　　$?L2
　　CALL　　!XXX
?L2 :
　　INC　　CNT
　　CMP　　CNT , #0FFH
　　BNZ　　$?L1
	repeat
	　　AX = BC
	　　if (ABC == #0CH)
	
	　　CALL　　!XXX
	　　endif
	　　CNT++
	until (CNT == #0FFH)

(2) 采用大写字母输入时

输出源	输入源

?L3 :
　　MOVW　　AX , BC
　　CMP　　ABC , #0CH
　　BNZ　　$?L4
　　CALL　　!XXX
?L4 :
　　INC　　CNT
　　CMP　　CNT , #0FFH
　　BZ　　$?L5
　　BR　　?L3
?L5 :
	REPEAT
	　　AX = BC
	　　if (ABC == #0CH)
	
	　　CALL　　!XXX
	　　endif
	　　CNT++
	UNTIL (CNT == #0FFH)

条件循环 until_bit

(8) repeat ~ until_bit

[描述格式]

```
[ Δ ] repeat
        指令组
[ Δ ] until_bit [ Δ ] ( 位校验表达式 )
```

[功能]

- 只要位校验表达式为假 , 则重复执行指令组 .

[说明]

- 在执行完指令组后对位校验表达式进行判断 . 因此 , 如果第一个位条件为真 , 则执行一次指令组 .

[生成的指令]

(1) 处理 repeat

- 为由 until_bit 生成的分支指令生成一个标签 .

(2) 处理 until_bit （位条件）

- 为由 repeat 生成的标签生成一个分支指令 , 以便在条件表达式判断为假时执行 repeat 和 until_bit 之间的指令组 . 如果条件表达式判断为真 , 则从 repeat 程序块中删除 until_bit.

[使用示例]

(1) 采用小写字母输入时

输出源	输入源
?L1 : MOV B , #8H CALL !XXX BF TRFG.0 , $?L1	repeat B = #8H CALL !XXX until_bit (TRFG.0)

(2) 采用大写字母输入时

输出源	输入源
?L2 : MOV B , #8H CALL !XXX BT TRFG.0 , $?L3 BR ?L2 ?L3 :	REPEAT B = #8H CALL !XXX UNTIL_BIT (TRFG.0)

条件循环 **break**

(9) break

[描述格式]

```
[ Δ ] break
```

[功能]

- 终止执行 while、 repeat、 for 和 switch 程序块中最内层嵌套的程序块 .

[说明]

- 如果输入了除while、while_bit、repeat ~ until、repeat ~ until_bit、for或switch语句之外的语句.则会发生错误.

[生成的指令]

- 生成一个无条件分支指令以跳出 while、 repeat、 for 或 switch 程序块 .

```
BR      ?Lxxxx
```

[使用示例]

输出源	输入源

?L1 :
MOV X , #0
MOV PORT4 , A
CMP A , #0FH
BNZ $?L2
BR ?L3
?L2 :
INCW HL
BR ?L1
?L3 :
	while (forever)
	X = #0
	PORT4 = A
	if (A == #0FH)
	break
	endif
	HL++
	endw

条件循环 continue

(10) continue

[描述格式]

```
[ Δ ] continue
```

[功能]

- 紧跟 while、 while_bit、 repeat ~ until、 repeat ~ until_bit 或 for 语句中最内层嵌套的程序块中的 continue 之后的跳出处理 , 并在判断条件前设置一个无条件分支 .

[说明]

- 这个用于从程序块的中间跳出后续处理并执行下一循环 .
- 如果输入了除 while、 while_bit、 repeat ~ until、 repeat ~ until_bit 或 for 语句之外的语句 , 则会发生错误 .

[生成的指令]

- 为标签生成一个无条件分支指令以重复 while、 while_bit、 repeat ~ until、 repeat ~ until_bit 或 for 程序块 .

```
BR      ?Lxxxx
```

[使用示例]

输出源	输入源

?L1 :
CMP SYM , #0FH
BNZ $?L2
MOV B , #0
MOV PORT4 , A
CMP A , #0FH
BNZ $?L3
BR ?L1
BR ?L4
?L3 :
INCW HL
?L4 :
BR ?L1
?L2 :
	while (SYM == #0FH)
	
	B = #0
	PORT4 = A
	if (A == #0FH)
	
	continue
	
	else
	HL++
	endif
	
	endw

条件循环 goto

(11) goto

[描述格式]

```
[ Δ ] goto Δ 标签
```

[功能]

- 无条件地转向一个标签 .

[说明]

- 当需要立即错误处理时输入 goto 语句 , 比如在错误处理程序中 , 或当需要在多处集中处理错误时 .
- 在汇编语言标签栏中显示的符号都指定为标签名 .

[生成的指令]

(1) 生成下列指令 .

```
BR      Label
```

(2) goto 语句的标签不是由 ST 78K0S 自动生成的 . 还需注意的是 ,ST78K0S 并不自动检测是否存在分支目的标签 .

[使用示例]

输出源	输入源

?L1 :
MOV B , #0
MOV PORT4 , A
CMP A , #0FH
BNZ $?L2
BR ERROR
?L2 :
INCW HL
BR ?L1
	while (forever)
	B = #0
	PORT4 = A
	if (A == #0FH)
	goto ERROR
	endif
	HL++
	endw

3.5　条件表达式

条件表达式用于通过控制语句设置条件 .

下面是条件表达式的例子 .

- 比较表达式：　　比较第一和第二个值 , 并判断其为真或假 .
- 位校验表达式：　　根据位符号确定标志的开 / 关状态 .
- 逻辑操作：　　当条件被组合在一起时 , 为条件表达式执行逻辑操作 .

如果在比较结束时指定 (γ) , 则可以在不能直接比较的 α 和 β 之间进行比较 .

γ 指定用于比较的寄存器 .

3.5.1　比较表达式

在每个比较表达式的描述中 , "?LTRUE" 用作比较判断为真时的分支目的标签 ,"?LFALSE" 用作判断为假时的分支目的标签 .

有关寄存器指定描述格式的说明 , 参见 "3.3 寄存器指定 ".

ST 78K0S 并不判断在比较表达式左侧还是右侧输入的符号是否正确地作为汇编语言操作数输入 . 但是 , 执行数据大小判断 , 并按照 "2.6 数据大小 " 中所描述的确定是否可以生成一条指令 . 另外 , 当指定一个寄存器时 , 判断使用指定寄存器生成指令的可行性 .

当判断结果出错时输出错误信息 .

有关细节 , 参见相关生成指令 .

以下内容描述了各种比较表达式的功能 .

所使用的例子作为源文件的注释语句 , 生成的指令都输入到该源文件中 .

表 3-3　比较指令的生成指令

符号			β																				
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
α	a	CY																						
	b	位符号																						
	c	[HL].β																						
	d	字节用户符号																						*1
	e	字节数据																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	字节寄存器																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	字用户符号																						
	m	字数据																						*2
	n	AX																						
	o	BC, DE, HL																						
	p	RP0, RP1, RP2, RP3																						
	q	sfrp																						
	r	SP																						
	s	直接访问符号																						
	t	间接访问符号																						
	u	[DE]																						
	v	立即符号																						

*1 :　　生成 CMP 指令

*2 :　　生成 CMPW 指令

空栏表示错误 .

表 3-4　比较表达式

比较表达式	描述格式	功能
等于 (==)	α == β	当 α = β 时为真 ,α ≠ β 时为假
不等于 (!=)	α != β	当 α ≠ β 时为真 , α = β 时为假
小于 (<)	α < β	当 α < β 时为真 , α >= β 时为假
大于 (>)	α > β	当 α > β 时为真 ,α <= β 时为假
大于等于 (>=)	α >=β	当 α >= β 时为真 , α < β 时为假
小于等于 (<=)	α <= β	当 α <= β 时为真 , α > β 时为假
无限循环 (forever)	forever	死循环语句

比较表达式等于 (==)

(1) 等于 (==)

[描述格式]

```
[ Δ ] [ 大小指定 ] α [ Δ ] == [ Δ ] [ 大小指定 ] [ Δ ] β [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 无寄存器指定时

  当 α 和 β 的内容相等时为真 , 不等时为假 .

- 有寄存器指定时

  α 的内容传输至指定寄存器 . 当指定寄存器的内容与 β 的内容相等时结果为真 , 不等时为假 .

[说明]

- 无寄存器指定时

  对于 α 和 β, 务必指定可在 CMP 或 CMPW 中输入的内容 .

- 有寄存器指定时

  对于 α, 务必指定可在 MOV 或 MOVW 中输入的内容 .

  对于 β, 务必指定可在 CMP 或 CMPW 中输入的内容 .

[生成的指令]

(1) 若控制语句采用小写字母输入 , 且无寄存器指定

```
CMP ( W )       α , β
BNZ             $?LFALSE
```

(2) 若控制语句采用小写字母输入 , 且有寄存器指定

```
MOV ( W )       Specified register , α
CMP ( W )       Specified register , β
BNZ             $?LFALSE
```

(3) 若控制语句采用大写字母输入 , 且无寄存器指定

```
        CMP ( W )       α , β
        BZ              $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

```
        MOV ( W )       Specified register , α
        CMP ( W )       Specified register , β
        BZ              $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

有关 α 和 β 组合的更多细节 , 参见表 3-3. α 表示指定寄存器 . 有关 MOV 生成指令的更多说明 , 参见 "4.2 (1) 赋值 (=)".

[使用示例]

(1) 若控制语句采用小写字母输入 , 且无寄存器指定

输出源	输入源

CMPW AX , #0F0FH
BNZ $?L1
CALL !XXX
BR ?L2
?L1 :
CALL !YYY
?L2 :
	if (AX == #0F0FH)
	CALL !XX
	else
	CALL !YYY
	endif

(2) 若控制语句采用小写字母输入 , 且有寄存器指定

输出源	输入源

MOV A , !XYZ
CMP A , #5
BNZ $?L3
CALL !PPP
?L3 :
	if (!XYZ == #5 (A))
	CALL !PPP
	endif

(3) 若控制语句采用大写字母输入 , 且无寄存器指定

输出源	输入源

CMPW AX , #0F0FH
BZ $?L4
BR ?L5
?L4 :
CALL !XXX
BR ?L6
?L5 :
CALL !YYY
?L6 :
	IF (AX == #0F0FH)
	CALL !XXX
	ELSE
	CALL !YYY
	ENDIF

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

输出源	输入源

MOV A , !XYZ
CMP A , #5
BZ $?L7
BR ?L8
?L7 :
CALL !PPP
?L8 :
	IF (!XYZ == #5 (A))
	
	CALL !PPP
	ENDIF

比较表达式不等于 (!=)

(2)　不等于 (!=)

[描述格式]

```
[ Δ ] [ 大小指定 ] [ Δ ] α [ Δ ] != [ Δ ] [ 大小指定 ] [ Δ ] β [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 无寄存器指定时

  当 α 和 β 的内容不等时为真 , 相等时为假 .

- 有寄存器指定时

  α 的内容传输至指定寄存器 . 当指定寄存器的内容与 β 的内容不等时结果为真 , 相等时为假 .

[说明]

- 无寄存器指定时

  对于 α 和 β, 务必指定可在 CMP 或 CMPW 中输入的内容 .

- 有寄存器指定时

  对于 α, 指定可在 MOV 或 MOVW 中输入的内容 .

  对于 β, 指定可在 CMP 或 CMPW 中输入的内容 .

[生成的指令]

(1)　若控制语句采用小写字母输入 , 且无寄存器指定

```
CMP ( W )       α , β
BZ              $?LFALSE
```

(2)　若控制语句采用小写字母输入 , 且有寄存器指定

```
MOV ( W )       Specified register , α
CMP ( W )       Specified register , β
BZ              $?LFALSE
```

(3)　若控制语句采用大写字母输入 , 且无寄存器指定

```
        CMP ( W )       α , β
        BNZ             $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

```
        MOV ( W )       Specified register , α
        CMP ( W )       Specified register , β
        BNZ             $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

有关 α 和 β 组合的更多细节 , 参见表 3-3. α 表示指定寄存器 . 有关 MOV 生成指令的更多说明 , 参见 "4.2 (1) 赋值 (=)".

[使用示例]

(1) 若控制语句采用小写字母输入 , 且无寄存器指定

输出源	输入源

CMPW AX , #0FFFH
BZ $?L1
CALL !XXX
BR ?L2
?L1 :
CALL !YYY
?L2 :
	if (AX != #0FFFH)
	
	CALL !XXX
	
	else
	CALL !YYY
	endif

(2) 若控制语句采用小写字母输入 , 且有寄存器指定

输出源	输入源

MOV A , !XYZ
CMP A , #5
BZ $?L
CALL !PPP
?L3 :
	if (!XYZ != #5 (A))
	
	CALL !PPP
	endif

(3) 若控制语句采用大写字母输入 , 且无寄存器指定

输出源	输入源

CMPW AX , #0FFFH
BNZ $?L4
BR ?L5
?L4 :
CALL !XXX
BR ?L6
?L5 :
CALL !YYY
?L6 :
	IF (AX != #0FFFH)
	
	CALL !XXX
	
	ELSE
	CALL !YYY
	ENDIF

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

输出源	输入源

MOV A , !XYZ
CMP A , #5
BNZ $?L7
BR ?L8
?L7 :
CALL !PPP
?L8 :
	IF (!XYZ != #5 (A))
	
	CALL !PPP
	ENDIF

比较表达式小于 (<)

(3) 小于 (<)

[描述格式]

```
[ Δ ] [ 大小指定 ] [ Δ ] α [ Δ ] < [ Δ ] [ 大小指定 ] [ Δ ] β [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 无寄存器指定时

  当 α 的内容小于 β 的内容时为真，否则为假（例如，等于或大于）.

- 有寄存器指定时

  α 的内容传输至指定寄存器 . 当指定寄存器的内容小于 β 的内容时结果为真，否则结果为假 .

[说明]

- 无寄存器指定时

  对于 α 和 β, 务必指定可在 CMP 或 CMPW 中输入的内容 .

- 有寄存器指定时

  对于 α, 务必指定可在 MOV 或 MOVW 中输入的内容 .

  对于 β, 务必指定可在 CMP 或 CMPW 中输入的内容 .

[生成的指令]

(1) 若控制语句采用小写字母输入，且无寄存器指定

```
CMP ( W )       α , β
BNC             $?LFALSE
```

(2) 若控制语句采用小写字母输入，且有寄存器指定

```
MOV ( W )       Specified register , α
CMP ( W )       Specified register , β
BNC             $?LFALSE
```

(3) 若控制语句采用大写字母输入，且无寄存器指定

```
        CMP ( W )       α , β
        BC              $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

```
        MOV ( W )       Specified register , α
        CMP ( W )       Specified register , β
        BC              $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

有关 α 和 β 组合的更多细节 , 参见表 3-3. α 表示指定寄存器 . 有关 MOV 生成指令的更多说明 , 参见 "4.2 (1) 赋值 (=)".

[使用示例]

(1) 若控制语句采用小写字母输入 , 且无寄存器指定

输出源	输入源

CMP A , [HL]
BNC $?L1
CALL !XXX
BR ?L2
?L1 :
CALL !YYY
?L2 :
	if (A < [HL])
	CALL !XXX
	else
	CALL !YYY
	endif

(2) 若控制语句采用小写字母输入 , 且有寄存器指定

输出源	输入源

MOVW AX , ABCP
CMPW AX , #0FE00H
BNC $?L3
CALL !PPP
?L3 :
	if (ABCP < #0FE00H (AX))
	CALL !PPP
	endif

(3) 若控制语句采用大写字母输入 , 且无寄存器指定

输出源	输入源

CMP A , [HL]
BC $?L4
BR ?L5
?L4 :
CALL !XXX
BR ?L6
?L5 :
CALL !YYY
?L6 :
	IF (A < [HL])
	CALL !XXX
	ELSE
	CALL !YYY
	ENDIF

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

输出源	输入源

MOVW AX , ABCP
CMPW AX , #0FE00H
BC $?L7
BR ?L8
?L7 :
CALL !PPP
?L8 :
	IF (ABCP < #0FE00H (AX))
	
	CALL !PPP
	ENDIF

比较表达式大于 (>)

(4) 大于 (>)

[描述格式]

```
[ Δ ] [ 大小指定 ] [ Δ ] α [ Δ ] > [ Δ ] [ 大小指定 n ] [ Δ ] β [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 无寄存器指定时

  当 α 的内容大于 β 的内容时为真 , 否则为假 （例如 , 等于或小于） .

- 有寄存器指定时

  α 的内容传输至指定寄存器 . 当指定寄存器的内容大于 β 的内容时结果为真 , 否则结果为假 .

[说明]

- 无寄存器指定时

  对于 α 和 β, 务必指定可在 CMP 或 CMPW 中输入的内容 .

- 有寄存器指定时

  对于 α, 务必指定可在 MOV 或 MOVW 中输入的内容 .

  对于 β, 务必指定可在 CMP 或 CMPW 中输入的内容 .

[生成的指令]

(1) 若控制语句采用小写字母输入 , 且无寄存器指定

```
CMP ( W )       α , β
BZ              $?LFALSE
BC              $?LFALSE
```

(2) 若控制语句采用小写字母输入 , 且有寄存器指定

```
MOV ( W )       Specified register , α
CMP ( W )       Specified register , β
BZ              $?LFALSE
BC              $?LFALSE
```

(3) 若控制语句采用大写字母输入 , 且无寄存器指定

```
        CMP ( W )       α , β
        BZ              $$ + 4
        BNC             $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

```
        MOV ( W )       Specified register , α
        CMP ( W )       Specified register , β
        BZ              $$ + 4
        BNC             $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

有关 α 和 β 组合的更多细节 , 参见表 3-3. α 表示指定寄存器 . 有关 MOV 生成指令的更多说明 , 参见 "4.2 (1) 赋值 (=)".

[使用示例]

(1) 若控制语句采用小写字母输入 , 且无寄存器指定

输出源	输入源

CMP A , [HL]
BZ $?L1
BC $?L1
CALL !XXX
BR ?L2
?L1 :
CALL !YYY
?L2 :
	if (A > [HL])
	
	CALL !XXX
	
	else
	CALL !YYY
	endif

(2) 若控制语句采用小写字母输入 , 且有寄存器指定

输出源	输入源

MOVW AX , ABCP
CMPW AX , #0FE40H
BZ $?L3
BC $?L3
CALL !PPP
?L3 :
	if (ABCP > #0FE40H (AX))
	
	CALL !PPP
	endif

(3) 若控制语句采用大写字母输入 , 且无寄存器指定

输出源	输入源

CMP A , [HL]
BZ $$ + 4
BNC $?L4
BR ?L5
?L4 :
CALL !XXX
BR ?L6
?L5 :
CALL !YYY
?L6 :
	IF (A > [HL])
	
	CALL !XXX
	
	ELSE
	CALL !YYY
	ENDIF

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

输出源	输入源

MOVW AX , ABCP
CMPW AX , #0FE40H
BZ $$ + 4
BNC $?L7
BR ?L8
?L7 :
CALL !PPP
?L8 :
	IF (ABCP > #0FE40H (AX))
	
	CALL !PPP
	ENDIF

比较表达式大于等于 (>=)

(5) 大于等于 (>=)

[描述格式]

```
[ Δ ] [ 大小指定 ] [ Δ ] α [ Δ ] >= [ Δ ] [ 大小指定 ] [ Δ ] β [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 无寄存器指定时

  当 α 的内容大于或等于 β 的内容时为真，小于 β 的内容时为假 .

- 有寄存器指定时

  α 的内容传输至指定寄存器 . 当指定寄存器的内容大于或等于 β 的内容时结果为真，小于 β 的内容时为假 .

[说明]

- 无寄存器指定时

  对于 α 和 β, 务必指定可在 CMP 或 CMPW 中输入的内容 .

- 有寄存器指定时

  对于 α, 务必指定可在 MOV 或 MOVW 中输入的内容 .

  对于 β, 务必指定可在 CMP 或 CMPW 中输入的内容 .

[生成的指令]

(1) 若控制语句采用小写字母输入，且无寄存器指定

```
CMP ( W )       α , β
BC              $?LFALSE
```

(2) 若控制语句采用小写字母输入，且有寄存器指定

```
MOV ( W )       Specified register , α
CMP ( W )       Specified register , β
BC              $?LFALSE
```

(3) 若控制语句采用大写字母输入，且无寄存器指定

```
        CMP ( W )       α , β
        BNC             $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

```
        MOV ( W )       Specified register , α
        CMP ( W )       Specified register , β
        BNC             $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

有关 α 和 β 组合的更多细节 , 参见表 3-3. α 表示指定寄存器 . 有关 MOV 生成指令的更多说明 , 参见 "4.2 (1) 赋值 (=)".

[使用示例]

(1) 若控制语句采用小写字母输入 , 且无寄存器指定

输出源	输入源

CMP A , [HL]
BC $?L1
CALL !XXX
BR ?L2
?L1 :
CALL !YYY
?L2 :
	if (A >= [HL])
	CALL !XXX
	else
	CALL !YYY
	endif

(2) 若控制语句采用小写字母输入 , 且有寄存器指定

输出源	输入源

MOVW AX , DE
CMPW AX , #0FE30H
BC $?L3
CALL !PPP
?L3 :
	if (DE >= #0FE30H (AX))
	CALL !PPP
	endif

(3) 若控制语句采用大写字母输入 , 且无寄存器指定

输出源	输入源

CMP A , [HL]
BNC $?L4
BR ?L5
?L4 :
CALL !XXX
BR ?L6
?L5 :
CALL !YYY
?L6 :
	IF (A >= [HL])
	CALL !XXX
	ELSE
	CALL !YYY
	ENDIF

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

输出源	输入源

MOVW AX , DE
CMPW AX , #0FE30H
BNC $?L7
BR ?L8
?L7 :
CALL !PPP
?L8 :
	IF (DE >= #0FE30H (AX))
	
	CALL !PPP
	ENDIF

比较表达式小于等于 (<=)

(6) 小于等于 (<=)

[描述格式]

```
[ Δ ] [ 大小指定 ] [ Δ ] α [ Δ ] <= [ Δ ] [ 大小指定 ] [ Δ ] β [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 无寄存器指定时

  当 α 的内容小于或等于 β 的内容时为真 , 大于 β 的内容时为假 .

- 有寄存器指定时

  α 的内容传输至指定寄存器 . 当指定寄存器的内容小于或等于 β 的内容时结果为真 , 大于 β 的内容时为假 .

[说明]

- 无寄存器指定时

  对于 α 和 β, 务必指定可在 CMP 或 CMPW 中输入的内容 .

- 有寄存器指定时

  对于 α, 务必指定可在 MOV 或 MOVW 中输入的内容 .

  对于 β, 务必指定可在 CMP 或 CMPW 中输入的内容 .

[生成的指令]

(1) 若控制语句采用小写字母输入 , 且无寄存器指定

```
CMP ( W )       α , β
BZ              $$ + 4
BNC             $?LFALSE
```

(2) 若控制语句采用小写字母输入 , 且有寄存器指定

```
MOV ( W )       Specified register , α
CMP ( W )       Specified register , β
BZ              $$ + 4
BNC             $?LFALSE
```

(3) 若控制语句采用大写字母输入 , 且无寄存器指定

```
        CMP ( W )       α , β
        BZ              $?LTRUE
        BC              $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

```
        MOV ( W )       Specified register , α
        CMP ( W )       Specified register , β
        BZ              $?LTRUE
        BC              $?LTRUE
        BR              ?LFALSE
?LTRUE :
```

有关 α 和 β 组合的更多细节 , 参见表 3-3. α 表示指定寄存器 . 有关 MOV 生成指令的更多说明 , 参见 "4.2 (1) 赋值 (=)".

[使用示例]

(1) 若控制语句采用小写字母输入 , 且无寄存器指定

输出源	输入源

CMP A , [HL]
BZ $$ + 4
BNC $?L1
CALL !XXX
BR ?L2
?L1 :
CALL !YYY
?L2 :
	if (A <= [HL])
	
	CALL !XXX
	
	else
	CALL !YYY
	endif

(2) 若控制语句采用小写字母输入 , 且有寄存器指定

输出源	输入源

MOVW AX , HL
CMPW AX , #0FE20H
BZ $$ + 4
BNC $?L3
CALL !PPP
?L3 :
	if (HL <= #0FE20H (AX))
	
	CALL !PPP
	endif

(3) 若控制语句采用大写字母输入 , 且无寄存器指定

输出源	输入源

CMP A , [HL]
BZ $?L4
BC $?L4
BR ?L5
?L4 :
CALL !XXX
BR ?L6
?L5 :
CALL !YYY
?L6 :
	IF (A <= [HL])
	
	CALL !XXX
	
	ELSE
	CALL !YYY
	ENDIF

(4) 若控制语句采用大写字母输入 , 且有寄存器指定

输出源	输入源

MOVW AX , HL
CMPW AX , #0FE20H
BZ $?L7
BC $?L7
BR ?L8
?L7 :
CALL !PPP
?L8 :
	IF (HL <= #0FE20H (AX))
	
	CALL !PPP
	ENDIF

比较表达式无限循环 (forever)

(7) 无限循环 (forever)

[描述格式]

```
[ Δ ] forever [ Δ ]
```

[功能]

- 设置 loop 语句为无限循环，不生成比较指令 .

[说明]

- 可在条件表达式的循环语句 (for 语句 , while 语句 , until 语句) 类型中输入 .

[使用示例]

(1) for 语句

输出源	输入源

MOV i , #0
?L1 :
MOV A , i
CALL !XXX
CMPW AX , #0FFH
BNZ $?L2
BR ?L3
?L2 :
INC i
BR ?L1
?L3 :
	for (i = #0 ; forever ; i++)
	
	A = i
	CALL !XXX
	if (AX == #0FFH)
	
	break
	endif
	
	next

(2) while 语句

输出源	输入源

?L4 :
BF forever , $?L5
MOV A , i
CALL !XXX
CMPW AX , #0FFH
BNZ $?L6
BR ?L5
?L6 :
INC i
BR ?L4
?L5 :
	while (forever)
	
	A = i
	CALL !XXX
	if (AX == #0FFH)
	
	break
	endif
	i++
	
	endw

(3) repeat 语句

输出源	输入源

?L7 :
　　MOV　　A , i
　　CALL　　!XXX
　　CMPW　　AX , #0FFH
　　BNZ　　$?L8
　　BR　　?L9
?L8 :
　　INC　　i
　　BR　　?L7
?L9 :
	repeat
	　　A = i
	　　CALL　　!XXX
	　　if (AX == #0FFH)
	　　break
	　　endif
	　　i++
	until (forever)

3.5.2　测试位表达式

在描述每种位校验表达式时,需注意,"?LTRUE"用作判断结果为真时的分支目的标签,"?LFALSE"用作判断结果为假时的标签 .

ST 78K0S并不判断位校验代码是否作为汇编语言操作数被正确输入. 但是,如"2.6 数据大小"中所述,会执行数据大小校验 .

另外 ,"Z" 也作为位符号进行处理 .

ST78K0S 不使用汇编器的伪指令 （EQU）去检查是否已定义位符号 . 但是 , 用户符号也可作为位符号进行处理 . 当判断结果错误时输出错误信息 .

有关细节 , 参见特殊生成指令 .

以下内容描述了各种位校验表达式的功能 .

所使用的例子作为源文件的注释语句 , 生成的指令都输入到该源文件中 .

表 3-5　位校验表达式

位校验表达式	描述格式	功能
位符号	位符号	当指定的位为 1 时为真
! 位符号	！位符号	当指定的位为 0 时为真

测试位表达式正逻辑 (位)

(1) 位符号

[描述格式]

```
[ Δ ] 位符号 [ Δ ]
```

[功能]

- 当位符号的内容为 1 时 , 结果为真 , 位符号内容为 0 时结果为假 .
- 下列控制语句可以包含作为条件表达式输入的位符号 .

```
if      if_bit
elseif  elseif_bit
while   while_bit
until   until_bit
for
```

[生成的指令]

(1) 当控制语句采用小写字母输入且已输入 CY 时

```
BNC     $?LFALSE
```

(2) 当控制语句采用小写字母输入且已输入 Z 时

```
BNZ     $?LFALSE
```

(3) 当控制语句采用小写字母输入且已输入位符号时

```
BF      Bit symbol , $?LFALSE
```

(4) 当控制语句采用大写字母输入且已输入 CY 时

```
        BC      $?LTRUE
        BR      ?LFALSE
?LTRUE :
```

(5) 当控制语句采用大写字母输入且已输入 Z 时

```
        BZ      $?LTRUE
        BR      ?LFALSE
?LTRUE :
```

(6) 当控制语句采用大写字母输入且已输入位符号时

```
        BT      Bit symbol , $?LTRUE
        BR      ?LFALSE
?LTRUE :
```

[使用示例]

(1) 当控制语句采用小写字母输入时

输出源	输入源

BNC $?L1
CALL !XXX
BR ?L2
?L1 :
CALL !YYY
?L2 :
	if_bit (CY)
	CALL !XXX
	else
	CALL !YYY
	endif

BNZ $?L3
CALL !XXX
BR ?L4
?L3 :
CALL !YYY
?L4 :
	if_bit (Z)
	CALL !XXX
	else
	CALL !YYY
	endif

BF TRFG.0 , $?L5
CALL !XXX
BR ?L6
?L5 :
CALL !YYY
?L6 :
	if_bit (TRFG.0)
	CALL !XXX
	else
	CALL !YYY
	endif

(2) 当控制语句采用大写字母输入时

输出源	输入源

BC $?L7
BR ?L8
?L7 :
CALL !XXX
BR ?L9
?L8 :
CALL !YYY
?L9 :
	IF_BIT (CY)
	CALL !XXX
	ELSE
	CALL !YYY
	ENDIF

BZ $?L10
BR ?L11
?L10 :
CALL !XXX
BR ?L12
?L11 :
CALL !YYY
?L12 :
	IF_BIT (Z)
	CALL !XXX
	ELSE
	CALL !YYY
	ENDIF

BT TRFG.0 , $?L13
BR ?L14
?L13 :
CALL !XXX
BR ?L15
?L14 :
CALL !YYY
?L15 :
	IF_BIT (TRFG.0)
	CALL !XXX
	ELSE
	CALL !YYY
	ENDIF

测试位表达式负逻辑 (位)

(2) ! 位符号

[描述格式]

```
[ Δ ] ! 位符号 [ Δ ]
```

[功能]

- 当位符号的内容为 0 时 , 结果为真 , 位符号内容为 1 时结果为假 .
- 下列控制语句可以包含作为条件表达式输入的位符号 .

```
if      if_bit
elseif  elseif_bit
while   while_bit
until   until_bit
for
```

[生成的指令]

(1) 当控制语句采用小写字母输入且已输入 CY 时

```
BC      $?LFALSE
```

(2) 当控制语句采用小写字母输入且已输入 Z 时

```
BZ      $?LFALSE
```

(3) 当控制语句采用小写字母输入且已输入位符号时

```
BT      Bit symbol , $?LFALSE
```

(4) 当控制语句采用大写字母输入且已输入 CY 时

```
        BNC     $?LTRUE
        BR      ?LFALSE
?LTRUE :
```

(5) 当控制语句采用大写字母输入且已输入 Z 时

```
        BNZ     $?LTRUE
        BR      ?LFALSE
?LTRUE :
```

(6) 当控制语句采用大写字母输入且已输入位符号时

```
        BF      Bit symbol , $?LTRUE
        BR      ?LFALSE
?LTRUE :
```

[使用示例]

(1) 当控制语句采用小写字母输入时

输出源	输入源

BC $?L1
CALL !XXX
BR ?L2
?L1 :
CALL !YYY
?L2 :
	if_bit (!CY)
	CALL !XXX
	else
	CALL !YYY
	endif

BZ $?L3
CALL !XXX
BR ?L4
?L3 :
CALL !YYY
?L4 :
	if_bit (!Z)
	CALL !XXX
	else
	CALL !YYY
	endif

BT TRFG.0 , $?L5
CALL !XXX
BR ?L6
?L5 :
CALL !YYY
?L6 :
	if_bit (!TRFG.0)
	CALL !XXX
	else
	CALL !YYY
	endif

(2) 当控制语句采用大写字母输入时

输出源	输入源

BNC $?L7
BR ?L8
?L7 :
CALL !XXX
BR ?L9
?L8 :
CALL !YYY
?L9 :
 	
BNZ $?L10
BR ?L11
?L10 :
CALL !XXX
BR ?L12
?L11 :
CALL !YYY
?L12 :
 	
BF TRFG.0 , $?L13
BR ?L14
?L13 :
CALL !XXX
BR ?L15
?L14 :
CALL !YYY
?L15 :
	IF_BIT (!CY)
	
	CALL !XXX
	ELSE
	CALL !YYY
	ENDIF
	
	IF_BIT (!Z)
	
	CALL !XXX
	
	ELSE
	CALL !YYY
	ENDIF
	
	IF_BIT (!TRFG.0)
	
	CALL !XXX
	
	ELSE
	CALL !YYY
	ENDIF

3.5.3　逻辑操作

在描述每种条件表达式时，需注意，"?LTRUE" 用作判断结果为真时的分支目的标签，"?LFALSE" 用作判断结果为假时的标签 .

当存在两个比较表达式或一个真 / 假校验位表达式时，可获得一个逻辑 AND (&&) 或逻辑 OR (||) 结果 .

在一个条件表达式中最多可以输入 16 个逻辑操作符 .

这意味着，有可能输入处理这样的表达式，该表达式在当两个条件表达式都满足或其中之一被满足时被执行 .

ST 78K0S 生成分支指令，该指令起始于最高优先级逻辑操作符 .

(1)　代码示例

```
B < #0FFH && C >= #0 || D == #10
```

以下内容描述了各种逻辑操作的功能 .

所使用的例子作为源文件的注释语句，生成的指令都输入到该源文件中 .

表 3-6　逻辑操作

逻辑操作	描述格式	功能
逻辑与 (&&)	条件表达式 1 && 条件表达式 2	如果条件表达式 1 和条件表达式 2 都为真，则结果为真
逻辑或 (||)	条件表达式 1 || 条件表达式 2	如果条件表达式 1 或条件表达式 2 为真，则结果为真

逻辑操作逻辑与 (&&)

(1) 逻辑与 (&&)

[描述格式]

条件表达式 1 [Δ] && [Δ] 条件表达式 2

[功能]

- 得到条件表达式 1 和条件表达式 2 的逻辑与的结果 . 当条件表达式 1 和条件表达式 2 都为真时 , 结果为真 , 否则结果为假 . 当两个条件都满足时执行输入操作 .

  输出指令间的差异取决于控制语句是大写输入还是小写输入 .

  首先为被圆括号 "( )" 括起来的内容生成测试指令 .

[生成的指令]

(1) 当控制语句采用小写字母输入时

表 3-7　逻辑与生成指令 （控制语句采用小写字母）

条件表达式	生成指令
α == β &&	CMP (W)　α , β BNZ　$?LFALSE
α != β &&	CMP (W)　α , β BZ　$?LFALSE
α < β &&	CMP (W)　α , β BNC　$?LFALSE
α > β &&	CMP (W)　α , β BZ　$?LFALSE BC　$?LFALSE
α >= β &&	CMP (W)　α , β BC　$?LFALSE
α <= β &&	CMP (W)　α , β BZ　$$ + 4 BNZ　$?LFALSE
位符号 &&	BF　Bit symbol , $?LFALSE
CY &&	BNC　$?LFALSE
Z &&	BNZ　$?LFALSE
! 位符号 &&	BT　Bit symbol , $?LFALSE
!CY &&	BC　$?LFALSE
!Z &&	BZ　$?LFALSE

(2) 当控制语句采用大写字母输入时

表 3-8　为逻辑与生成指令（控制语句采用大写字母）

条件表达式	生成指令
α == β &&	CMP (W)　α , β BZ　$?LTRUE BR　?LFALSE ?LTRUE :
α != β &&	CMP (W)　α , β BNZ　$?LTRUE BR　?LFALSE ?LTRUE :
α < β &&	CMP (W)　α , β BC　$?LTRUE BR　?LFALSE ?LTRUE :
α > β &&	CMP (W)　α , β BZ　$$ + 4 BNC　$?LTRUE BR　?LFALSE ?LTRUE :
α >= β &&	CMP (W)　α , β BNC　$?LTRUE BR　?LFALSE ?LTRUE :
α <= β &&	CMP (W)　α , β BZ　$?LTRUE BC　$?LTRUE BR　?LFALSE ?LTRUE :
位符号 &&	BT　Bit symbol , $?LTRUE BR　?LFALSE ?LTRUE :
CY &&	BC　$?LTRUE BR　?LFALSE ?LTRUE :
Z &&	BZ　$?LTRUE BR　?LFALSE ?LTRUE :
! 位符号 &&	BF　Bit symbol , $?LTRUE BR　?LFALSE ?LTRUE :
!CY &&	BNC　$?LTRUE BR　?LFALSE ?LTRUE :
!Z &&	BNZ　$?LTRUE BR　?LFALSE ?LTRUE :

[使用示例]

(1) 当控制语句采用小写字母输入时

输出源	输入源

MOV A , C
CMP A , #0
BNZ $?L1
MOV A , B
CMP A , #0
BC $?L1
MOV A , B
CMP A , #80H
BNC $?L1
CALL !XXX
BR ?L2
?L1 :
CALL !YYY
?L2 :
	if (C == #0 && B >= #0 && B < #80H) (A)
	CALL !XXX
	else
	CALL !YYY
	endif

(2) 当控制语句采用大写字母输入时

输出源	输入源

MOV A , C
CMP A , #0
BZ $?L3
BR ?L6
?L3 :
MOV A , B
CMP A , #0
BNC $?L4
BR ?L6
?L4 :
MOV A , B
CMP A , #80H
BC $?L5
BR ?L6
?L5 :
CALL !XXX
BR ?L7
?L6 :
CALL !YYY
?L7 :
	IF (C == #0 && B >= #0 && B < #80H) (A)
	CALL !XXX
	ELSE
	CALL !YYY
	ENDIF

逻辑操作逻辑或 (||)

(2) 逻辑或 (||)

[描述格式]

条件表达式 1 [Δ] || [Δ] 条件表达式 2

[功能]

- 得到条件表达式 1 和条件表达式 2 的逻辑或的结果 . 当条件表达式 1 和条件表达式 2 其中之一为真时 , 结果为真 , 两个表达式都为假时结果为假 . 当两条件之一满足时执行输入操作 .

  首先为被圆括号 "( )" 括起来的内容生成测试指令 .

[生成的指令]

表 3-9　逻辑或生成指令

条件表达式	生成指令
α == β ||	CMP (W) α , β BZ $?LFALSE
α != β ||	CMP (W) α , β BNZ $?LFALSE
α < β ||	CMP (W) α , β BC $?LFALSE
α > β ||	CMP (W) α , β BZ $$ + 4 BNC $?LFALSE
α >= β ||	CMP (W) α , β BNC $?LFALSE
α <= β ||	CMP (W) α , β BZ $?LFALSE BC $?LFALSE
位符号 ||	BT Bit symbol , $?LFALSE
CY ||	BC $?LFALSE
Z ||	BZ $?LFALSE
! 位符号 ||	BF Bit symbol , $?LFALSE
!CY ||	BNC $?LFALSE
!Z ||	BNZ $?LFALSE

[使用示例]

输出源	输入源

MOV A , B
CMP A , #0
BZ $?L1
MOV A , C
CMP A , #0
BNC $?L1
MOV A , D
CMP A , #80H
BNC $?L2
?L1 :
CALL !XXX
BR ?L3
?L2 :
CALL !YYY
?L3 :
	if (B == #0 || C >= #0 || D < #80H) (A)
	
	CALL !XXX
	
	else
	CALL !YYY
	endif

第四章　　表达式

本章描述表达式的功能。

4.1　概述

表达式用来进行赋值或算术操作 .

下面是表达式的例子 .

赋值语句 :	把第二个操作数赋给第一个操作数
计数语句 :	操作数的值加 1 或减 1
交换语句 :	第一个操作数和第二个操作数的值互换
位操作语句 :	操作数的值置位 （置 1）或复位 （置 0）

表达式的功能描述如下 .

下列举例是生成指令输入源文件的注释语句 .

表 4-1　赋值语句

赋值语句	描述格式	功能
赋值 (=)		
赋值	α = β	α <- β
顺序赋值	α_1 = ... = α_n = β	α_1 = <- β, ..., α_n <- β
赋值 (带寄存器指定)	α = β (γ)	(γ) <- β, α <- (γ)
顺序赋值 (带寄存器指定)	α_1 = ... = α_n = β (γ)	γ <- β, α_1 <- γ, ..., α_n <- γ
增量赋值 n (+=)		
增量赋值	α += β	α <- α + β
增量赋值 (带寄存器指定)	α += β (寄存器)	γ <- α, γ <- γ + β, α <- γ
带进位增量赋值	α += β , CY	α <- α + β, CY
带进位增量赋值 (带寄存器指定)	α += β , CY (寄存器)	γ <- α, γ <- γ + β, CY, α <- γ

表 4-1 赋值语句

赋值语句	描述格式	功能
减量赋值 (-=)		
减量赋值	α -= β	α <- α - β
减量赋值 (带寄存器指定)	α -= β , (寄存器)	γ <- α, γ <- γ - β, α <- γ
带进位减量赋值	α -= β , CY	α <- α - β, CY
带进位减量赋值 (带寄存器指定)	α -= β , CY (寄存器)	γ <- α, γ <- γ - β, CY, α <- γ
逻辑与 AND 赋值 (&=)		
逻辑与 AND 赋值	α &= β	α <- α ∩ β
逻辑与 AND 赋值 (带寄存器指定)	α &= β (寄存器)	γ <- α, γ <- γ ∩ β, α <- γ
逻辑或 OR 赋值 (|=)		
逻辑或 OR 赋值	α |= β	α <- α ∪ β
逻辑或 OR 赋值 (带寄存器指定)	α |= β (寄存器)	γ <- α, γ <- γ ∪ β, α <- γ
逻辑异或 XOR 赋值 (^=)		
逻辑异或 XOR 赋值	α ^= β	α <- α ^ β
逻辑异或 XOR 赋值 (带寄存器指定)	α ^= β (寄存器)	γ <- α, γ <- γ ^ β, α <- γ
右移赋值 (>>=)		
右移 (循环) 赋值	α >>= β	(α 右移 β 位)
右移 赋值 (带寄存器指定)	α >>= β (寄存器)	γ <- α, (γ 左移 β 位), α <- γ
左移赋值 (<<=)		
左移 赋值	α <<= β	(α 左移 β 位)
左移赋值 (带寄存器指定)	α <<= β (寄存器)	γ <- α, (γ 左移 β 位), α <- γ

表 4-2 计数语句

计数语句	描述格式	功能
递增 (++)	α ++	α <- α + 1
递减 (--)	α --	α <- α - 1

表 4-3 交换语句

交换语句	描述格式	功能
交换 (<->)		
交换	α <-> β	α <- α <-> β
交换 (带寄存器指定)	α <-> β (γ)	γ <- α, γ <- γ <-> β, α <- γ

表 4-4 位操作语句

位操作语句	描述格式	功能
置位 (=)		
置位	α = 1	α <- 1
顺序置位	α_1 = ... = α_n = 1	α_n = <- 1, ..., α_1 <- 1
置位 (带寄存器指定)	α = 1 (CY)	CY <- 1, α <- 1
顺序置位 (带寄存器指定)	α_1 = ... α_n = 1 (CY)	CY <- 1, α_n <- 1, ..., α_1 <- 1
复位 (=)		
复位	α = 0	α <- 0
顺序复位	α_1 = ... = α_n = 0	α_n <- 0, ..., α_1 <- 0
复位 (带寄存器指定)	α = 0 (CY)	CY <- 0, α <- 0
顺序复位 (带寄存器指定)	α_1 = ... α_n = 0 (CY)	CY <- 0, α_n <- 0, ..., α_1 <- 0

4.2 赋值语句

赋值语句赋值 (=)

(1) 赋值 (=)

[描述格式]

```
[ Δ ] [ 长度指定 ] [ Δ ] α1 [ Δ ] [ = [ Δ ] [ 长度指定 ] [ Δ ] α2 [ Δ ] ... ] =
[ Δ ] [ 长度指定 ] [ Δ ] β [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 没有寄存器指定时

  右侧的 β 值依次赋给左侧的变量 .

- 有寄存器指定时

  右侧的 β 值 被赋给指定寄存器或 CY, 其值被依次赋给左侧的变量 .

[说明]

- α 和 β 是可通过 MOV 和 MOVW 指令输入的值 .

  一行中最多可输入 32 个赋值操作符 "=". 输入多于 32 个操作符时将出现错误 . 在序列赋值中即使发生一个错误 , 也不生成指令 .

[生成指令]

(1) 当 α 和 β 不是位符号时

< β 是 CY 时 >

```
        BF      β , ?L1
        SET1    CY
        BR      ?L2
?L1 :
        CLR1    CY
?L2 :
```

但是 , 不能进行连续赋值 .

< β 是 CY 时 >

```
        BNC     ?L1
        SET1    αn
        SET1    αn-1
        □□ :
        SET1    α2
        SET1    α1
        BR      ?L2
?L1 :
        CLR1    αn
        CLR1    αn-1
        □□ :
        CLR1    α2
        CLR1    α1
?L2 :
```

< 当 CY 被指定为寄存器时 >

```
        BF      β , ?L1
        SET1    αn
        SET1    αn-1
        □□ :
        SET1    α2
        SET1    α1
        BR      ?L2
?L1 :
        CLR1    αn
        CLR1    αn-1
        □□ :
        CLR1    α2
        CLR1    α1
?L2 :
```

(2) 当 α 和 β 不是位符号时

< 没有指定寄存器时 >

```
MOV     α1 , β
```

依据操作数 , 可替代生成 MOV1 或 MOVW.

< 没有指定寄存器且连续赋值时 >

```
MOV     αn , β
MOV     αn-1 , β
   :
MOV     α2 , β
MOV     α1 , β
```

依据操作数 , 可替代生成 MOV1 或 MOVW.

< 有寄存器指定时 >

```
MOV     Specified register , β
MOV     α1 , Specified register
```

依据操作数 , 可替代生成 MOV1 或 MOVW.

< 有寄存器指定且连续赋值时 >

```
MOV     Specified register , β
MOV     αn , specified register
MOV     αn-1 , specified register
        :
MOV     α2 , specified register
MOV     α1 , specified register
```

依据操作数 , 可替代生成 MOV1 或 MOVW.

关于 α_n 和 β 组合的详情 , 参见表 4-5. 根据输入语句的不同 , α_n 和 β 表示指定的寄存器 .

[使用举例]

(1) 没有寄存器指定时

输出源程序	输入源程序
BF P1.1 , $?L1	CY = P1.1
SET1 CY	
BR ?L2	
?L1 :	
CLR1 CY	
?L2 :	
MOV A , #4H	A = #4H
MOVW AX , SYMP	AX = SYMP
BNC $?L3	PORT.0 = bit1 = CY
SET1 bit1	
SET1 PORT.0	
BR ?L4	
?L3 :	
CLR1 bit1	
CLR1 PORT.0	
?L4 :	
MOV DAT3 , A	DAT1 = DAT2 = DAT3 = A
MOV DAT2 , A	
MOV DAT1 , A	

(2) 有寄存器指定时

输出源程序	输入源程序
BF P1.1 , $?L5	A.0 = P0.2 = P1.1 (CY)
SET1 P0.2	
SET1 A.0	
BR ?L6	
?L5 :	
CLR1 P0.2	
CLR1 A.0	
?L6 :	
MOV A , #4H	[DE] = #4H (A)
MOV [DE] , A	
MOV A,X	DAT1 = DAT2 = DAT3 = X (A)
MOV DAT3 , A	
MOV DAT2 , A	
MOV DAT1 , A	
MOVW AX,BC	DATA1P = DATA2P = DATA3P = BC (AX)
MOVW DATA3P , AX	
MOVW DATA2P , AX	
MOVW DATA1P , AX	

表 4-5 赋值生成指令

符号			β																				
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
α_n	a	CY		*3		*4								*3										
	b	位符号	*3			*5																		
	c	字节用户符号	*3			*5																		
	d	[HL].β	*6			*5		*1							*2									*1
	e	字节数据						*1																*1
	f	A				*1	*1		*1	*1											*1	*1	*1	*1
	g	字节寄存器						*1																*1
	h	R0						*1																*1
	i	R1																						*1
	j	sfr						*1																*1
	k	PSW						*1																*1
	l	字用户符号	*3			*5		*1							*2									
	m	字数据													*2									
	n	AX				*2								*2			*2			*2				*2
	o	字寄存器													*2									*2
	p	RP0																						*2
	q	sfrp																						
	r	SP													*2									
	s	直接访问符号						*1																
	t	间接访问符号						*1																
	u	[DE]						*1																
	v	立即符号																						

*1 : 生成 MOV 指令

*2 : 生成 MOVW 指令

*3 : 生成 MOV1 指令

*4 : 当 "1" 作为 b 输入时生成 SET1 指令 . 当 "0" 输入时生成 CLR1 指令 . 当输入除 "0" 或 "1" 外的其它值时生成 MOV 指令 .

*5 : 当 "1" 作为 b 输入时生成 SET1 指令 . 当 "0" 输入时生成 CLR1 指令 .

*6 : 当输入除 "0" 或 "1" 外的其它值时 , 生成 MOV1 指令。空单元格表示错误 .

赋值语句增量赋值 (+=)

(2) 增量赋值 (+=)

[描述格式]

```
[ Δ ] [ 长度指定 ] [ Δ ] α [ Δ ] += [ Δ ] [ 长度指定 ] [ Δ ] β [ Δ ] [ , [ Δ ] CY ] [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 没有寄存器指定时

  α 和 β 这两个操作数相加 , 结果赋给 α.

- 有寄存器指定时

  α 赋给指定寄存器 .

  指定寄存器的值加在 β 上 , 结果赋给指定寄存器 .

  指定寄存器的值赋给 α.

- 带进位累加 , 无寄存器指定

  使用 α 和 β 这两个操作数执行带进位的加运算 , 结果赋给 α.

- 带进位累加 , 有寄存器指定

  α 值赋给指定寄存器 .

  用指定寄存器的值和 β 进行带进位的增量运算 , 结果赋给指定寄存器 .

  指定寄存器的值赋给 α.

[说明]

- 没有寄存器指定时

  α 和 β 的值可输入 ADD 和 ADDW.

- 有寄存器指定时

  α 的值可输入 MOV 和 MOVW.

  β 的值可输入 ADD 和 ADDW.

- 带进位累加 , 无寄存器指定

  α and β 的值可输入 ADDC.

- 带进位累加 , 有寄存器指定

  α 的值可输入 MOV.

  β 的值可输入 ADDC.

[生成指令]

(1) 没有寄存器指定时

```
ADD     α , β
```

依据操作数 , 可替代生成 ADDW.

(2) 有寄存器指定时

```
MOV     Specified register , α
ADD     Specified register , β
MOV     α , specified register
```

依据操作数 , 可替代生成 ADDW.

(3) 带进位累加 , 无寄存器指定

```
ADDC    α , β
```

(4) 带进位累加 , 有寄存器指定

```
MOV     Specified register , α
ADDC    Specified register , β
MOV     α , specified register
```

关于 α 和 β 组合的详细情况 , 参见表 4-6. 根据输入语句的不同 , α 表示指定的寄存器 .

[使用举例]

(1) 没有寄存器指定时

输出源程序	输入源程序
ADD A , #0C0H ADDW AX , #0C00H	A += #0C0H AX += #0C00H

(2) 有寄存器指定时

输出源程序	输入源程序
MOV A , !ABC ADD A , #0FCH MOV !ABC , A MOVW AX , HL ADDW AX , #0FFFH MOVW HL , AX	ABC += #0FCH (A) HL += #0FFFH (AX)

(3) 带进位累加 , 无寄存器指定

输出源程序	输入源程序
ADDC A , #50H	A += #50H , CY

(4) 带进位累加 , 有寄存器指定

输出源程序	输入源程序
MOV A , PSW ADDC A , #50H MOV PSW , A	PSW += #50H , CY (A)

表 4-6 增量赋值生成指令

符号			β																				
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
α	a	CY																						
	b	位符号																						
	c	字节用户符号																						
	d	[HL].β																						*1
	e	字节数据																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	字节寄存器																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	字用户符号																						
	m	字数据																						
	n	AX																						*2
	o	字寄存器																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接访问符号																						
	t	间接访问符号																						
	u	[DE]																						
	v	立即符号																						

*1 : 生成 ADD 指令 . 带进位增加时生成 ADDC 指令 .

*2 : 生成 ADDW 指令 .

空单元格表示错误 .

赋值语句减量赋值 (-=)

(3)　减量赋值 (-=)

[描述格式]

```
[ Δ ] [ 长度指定 ] [ Δ ] α [ Δ ] -= [ Δ ] [ 长度指定 ] [ Δ ] β [ Δ ] [ , [ Δ ]
CY ] [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 没有寄存器指定时

  从 α 中减去 β , 结果赋给 α.

- 有寄存器指定时

  α 赋给指定寄存器 .

  从指定寄存器的值中减去 β , 并把结果赋给指定寄存器 .

  指定寄存器的值赋给 α.

- 带进位减 , 无寄存器指定

  使用 α 和 β 这两个操作数执行带进位的加运算 , 结果赋给 α.

- 带进位减 , 有寄存器指定

  α 值赋给指定寄存器 .

  用指定寄存器的值和 β 进行带进位的减运算 , 结果赋给指定寄存器 .

  指定寄存器的值赋给 α.

[说明]

- 没有寄存器指定时

  α 和 β 的值可输入 SUB 和 SUBW.

- 有寄存器指定时

  α 的值可输入 MOV 和 MOVW.

  β 的值可输入 SUB 和 SUBW.

- 带进位减 , 没有寄存器指定

  α 和 β 的值可输入 SUBC.

- 带进位减 , 有寄存器指定

  α 的值可输入 MOV.

  β 的值可输入 SUBC.

[生成指令]

(1) 没有寄存器指定时

```
SUB     α , β
```

依据操作数 , 可替代生成 SUBW.

(2) 有寄存器指定时

```
MOV     Specified register , α
SUB     Specified register , β
MOV     α , specified register
```

依据操作数 , 可替代生成 SUBW.

(3) 带进位减 , 没有寄存器指定

```
SUBC    α , β
```

(4) 带进位减 , 有寄存器指定

```
MOV     Specified register , α
SUBC    Specified register , β
MOV     α , specified register
```

关于 α 和 β 组合的详情 , 参见 表 4-7. 根据输入语句 , α 表示指定寄存器 .

[使用举例]

(1) 没有寄存器指定时

输出源程序	输入源程序
SUB A , #0C0H SUBW AX , #0C00H	A -= #0C0H AX -= #0C00H

(2) 有寄存器指定时

输出源程序	输入源程序
MOV A , !ABC SUB A , #0FCH MOV !ABC , A MOVW AX , HL SUBW AX , #0FFFH MOVW HL , AX	!ABC -= #0FCH (A) HL -= #0FFFH (AX)

(3) 带进位减，没有寄存器指定

输出源程序	输入源程序
SUBC　　A , #50H	A -= #50H , CY

(4) 带进位减，有寄存器指定

输出源程序	输入源程序
MOV　　A , PSW SUBC　　A , #50H MOV　　PSW , A	PSW -= #50H , CY (A)

表 4-7　减量赋值生成指令

符号			β																				
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
α	a	CY																						
	b	位符号																						
	c	字节用户符号																						
	d	[HL].β																						*1
	e	字节数据																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	字节寄存器																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	字用户符号																						
	m	字数据																						
	n	AX																						*2
	o	字寄存器																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接访问符号																						
	t	间接访问符号																						
	u	[DE]																						
	v	立即符号																						

*1 :　　生成 SUB 指令 . 带进位减时生成 SUBC 指令 .

*2 :　　生成 SUBW 指令 .

空单元格表示错误 .

赋值语句逻辑与赋值 (&=)

(4) 逻辑与赋值 (&=)

[描述格式]

[Δ] [长度指定] [Δ] α [Δ] &= [Δ] [长度指定] [Δ] β [Δ] [寄存器指定]

[功能]

- 没有寄存器指定时

  对 α 和 β 按位进行逻辑与 (α & β) , 结果赋给 α.

- 有寄存器指定时

  α 赋给指定寄存器 .

  对指定寄存器和 β 按位进行逻辑与 (指定寄存器 & β) , 结果赋给指定寄存器 .

  指定寄存器的值赋给 α.

[说明]

- 没有寄存器指定

  α 和 β 的值可输入 AND 和 BF.

- 有寄存器指定

  α 的值可输入 MOV 和 BF.

  β 的值可输入 AND 和 BF.

[生成指令]

(1) 没有寄存器指定时

< α 是 CY 时 >

```
        BNC     ?L1
        BF      β , ?L1
        SET1    CY
        BR      ?L2
?L1 :
        CLR1    CY
?L2 :
```

< α 不是 CY 时 >

```
AND     α , β
```

(2) 有寄存器指定时

< 指定寄存器是 CY 时 >

```
        BF      α , ?L1
        BF      β , ?L1
        SET1    α
        BR      ?L2
?L1 :
        CLR1    α
?L2 :
```

< 指定寄存器不是 CY 时 >

```
MOV     Specified register , α
AND     Specified register , β
MOV     α , specified register
```

关于 α 和 β 组合的详情 , 参见表 4-8.

[使用举例]

(1) 没有寄存器指定时

输出源程序			输入源程序
	BNC	$?L1	CY &= P1S.1
	BF	P1S.1 , $?L1	
	SET1	CY	
	BR	?L2	
?L1 :			
	CLR1	CY	
?L2 :			
	AND	A , #0FFH	A &= #0FFH

(2) 有寄存器指定时

输出源程序			输入源程序
	BF	A.1 , $?L3	A.1 &= PORT3.0 (CY)
	BF	PORT3.0 , $?L3	
	SET1	A.1	
	BR	?L4	
?L3 :			
	CLR1	A.1	
?L4 :			
	MOV	A , [DE]	[DE] &= #07H (A)
	AND	A , #07H	
	MOV	[DE] , A	

表 4-8 逻辑与赋值生成指令

符号			β																				
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
α	a	CY		*2		*2								*2										
	b	位符号																						
	c	字节用户符号																						
	d	[HL].β																						*1
	e	字节数据																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	字节寄存器																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	字用户符号																						
	m	字数据																						
	n	AX																						
	o	字寄存器																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接访问符号																						
	t	间接访问符号																						
	u	[DE]																						
	v	立即符号																						

*1 : 生成 AND 指令 .

*2 : 生成 AND1 指令 .

空单元格表示错误 .

赋值语句逻辑或赋值 (|=)

(5) 逻辑或赋值 (|=)

[描述格式]

[Δ] [长度指定] [Δ] α [Δ] |= [Δ] [长度指定] [Δ] β [Δ] [寄存器指定]

[功能]

- 没有寄存器指定时

  对 α 和 β 按位进行逻辑或 (α | β) , 结果赋给 α.

- 有寄存器指定时

  α 赋给指定寄存器 .

  对指定寄存器和 β 按位进行逻辑或 (指定寄存器 | β) , 结果赋给指定寄存器 .

  寄存器的值赋给 α.

[说明]

- 没有寄存器指定时

  α 和 β 的值可输入 OR 和 BF.

- 有寄存器指定时

  α 的值可输入 MOV 和 BT.

  β 的值可输入 OR 和 BF.

[生成指令]

(1) 没有寄存器指定时

< α 是 CY 时 >

```
        BC      ?L1
        BF      β , ?L2
?L1 :
        SET1    CY
        BR      ?L3
?L2 :
        CLR1    CY
?L3 :
```

< α 不是 CY 时 >

```
OR      α , β
```

(2) 有寄存器指定时

< 指定寄存器是 CY 时 >

```
        BC      ?L1
        BF      β , ?L2
?L1 :
        SET1    CY
        BR      ?L3
?L2 :
        CLR1    CY
?L3 :
```

< 指定寄存器不是 CY 时 >

```
MOV     Specified register , α
OR      Specified register , β
MOV     α , specified register
```

关于 α 和 β 组合的详细情况 , 参见表 4-9.

[使用举例]

(1) 没有寄存器指定时

输出源程序	输入源程序
BC $?L1 BF P1S.1 , $?L2 ?L1 : SET1 CY BR ?L3 ?L2 : CLR1 CY ?L3 : OR A , #0FFH	CY |= P1S.1 A |= #0FFH

(2) 有寄存器指定时

输出源程序	输入源程序
BT A.1 , $?L4 BF PORT3.0 , $?L5 ?L4 : SET1 A.1 BR ?L6 ?L5 : CLR1 A.1 ?L6 : MOV A , [DE] OR A , #07H MOV [DE] , A	A.1 |= PORT3.0 (CY) [DE] |= #07H (A)

表 4-9 逻辑或 赋值生成指令

符号			β																				
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
α	a	CY		*2		*2								*2										
	b	位符号																						
	c	字节用户符号																						
	d	[HL].β																						*1
	e	字节数据																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	字节寄存器																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	字用户符号																						
	m	字数据																						
	n	AX																						
	o	字寄存器																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接访问符号																						
	t	间接访问符号																						
	u	[DE]																						
	v	立即符号																						

*1 : 生成 OR 指令 .

*2 : 生成 OR1 指令 .

空单元格表示错误 .

赋值语句逻辑异或赋值 (^=)

(6) 逻辑异或赋值 (^=)

[描述格式]

[Δ] [长度指定] [Δ] α [Δ] ^= [Δ] [长度指定] [Δ] β [Δ] [寄存器指定]

[功能]

- 没有寄存器指定时

  对 α 和 β 按位进行逻辑异或 (α ^ β) , 结果赋给 α.

- 有寄存器指定时

  α 赋给指定寄存器 .

  对指定寄存器和 β 按位进行逻辑异或 (指定寄存器 ^ β) , 结果赋给指定寄存器 .

  指定寄存器的值赋给 α.

[说明]

- 没有寄存器指定时

  α 和 β 的值可输入 XOR 和 BF.

- 有寄存器指定时

  α 的值可输入 MOV 和 BT.

  β 的值可输入 XOR 和 BF.

[生成指令]

(1) 没有寄存器指定时

< α 是 CY 时 >

```
        BNC     ?L1
        BF      β , ?L2
?L1 :
        BC      ?L3
        BF      β , ?L3
?L2 :
        SET1    CY
        BR      ?L4
?L3 :
        CLR1    CY
?L4 :
```

< α 不是 CY 时 >

```
XOR     α , β
```

(2) 有寄存器指定时

< 指定寄存器是 CY 时 >

```
        BF      α , ?L1
        BF      β , ?L2
?L1 :
        BT      α , ?L3
        BF      β , ?L3
?L2 :
        SET1    α
        BR      ?L4
?L3 :
        CLR1    α
?L4 :
```

< 指定寄存器不是 CY 时 >

```
MOV     Specified register , α
XOR     Specified register , β
MOV     α , specified register
```

关于 α 和 β 组合的详细情况 , 参见表 4-10.

[使用举例]

(1) 没有寄存器指定时

输出源程序	输入源程序
BNC $?L1 BF P1S.1 , $?L2 ?L1 : BC $?L3 BF P1S.1 , $?L3 ?L2 : SET1 CY BR ?L4 ?L3 : CLR1 CY ?L4 : XOR A , #0FFH	CY ^= P1S.1 A ^= #0FFH

(2) 有寄存器指定时

输出源程序	输入源程序
BF A.1 , $?L5 BF PORT3.0 , $?L6 ?L5 : BT A.1 , $?L7 BF PORT3.0 , $?L7 ?L6 : SET1 A.1 BR ?L8 ?L7 : CLR1 A.1 ?L8 : MOV A , [DE] XOR A , #07H MOV [DE] , A	A.1 ^= PORT3.0 (CY) [DE] ^= #07H (A)

表 4-10 逻辑异或赋值生成指令

符号			β																				
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
α	a	CY		*2		*2								*2										
	b	位符号																						
	c	字节用户符号																						
	d	[HL].β																						*1
	e	字节数据																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	字节寄存器																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	字用户符号																						
	m	字数据																						
	n	AX																						
	o	字寄存器																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接访问符号																						
	t	间接访问符号																						
	u	[DE]																						
	v	立即符号																						

*1 : 生成 XOR 指令 .

*2 : 生成 XOR1 指令 .

空单元格表示错误 .

赋值语句右移赋值 (>>=)

(7) 右移赋值 (>>=)

[描述格式]

```
[ Δ ] [ 长度指定 ] [ Δ ] α [ Δ ] >>= [ Δ ] β [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 没有寄存器指定时

  α 右移 β 位 , 结果赋给 α.

- 有寄存器指定时

  α 赋给指定寄存器 .

  指定寄存器的值右移 β 位 , 结果赋给指定寄存器 .

  指定寄存器的值赋给 α.

[说明]

- 没有寄存器指定时

  α 的值只能输入 A.

  β 的值可输入 1 至 7 的数字 .

- 有寄存器指定时

  α 的值可输入 MOV.

  β 的值可输入 1 至 7 的数字 .

  指定寄存器的值只能输入 A .

[生成指令]

(1) 没有寄存器指定时

ROR 指令输出 β 次后生成 AND 指令 .

```
ROR     A , 1
        :
AND     A , #0FFH SHR β
```

(2) 有寄存器指定时

```
MOV     A , α
ROR     A , 1
        :
AND     A , #0FFH SHR β
MOV     α , A
```

[使用举例]

(1) 没有寄存器指定时

输出源程序	输入源程序
ROR A , 1 ROR A , 1 ROR A , 1 ROR A , 1 AND A , #0FFH SHR 4	A >>= 4

(2) 有寄存器指定时

输出源程序	输入源程序
MOV A , CCV ROR A , 1 ROR A , 1 ROR A , 1 ROR A , 1 AND A , #0FFH SHR 4 MOV CCV , A	CCV >>= 4 (A)

赋值语句左移赋值 (<<=)

(8) 左移赋值 (<<=)

[描述格式]

```
[ Δ ] [ 长度指定 ] [ Δ ] α [ Δ ] <<= [ Δ ] β [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 没有寄存器指定时

  α 左移 β 位 , 结果赋给 α.

- 有寄存器指定时

  α 赋给指定寄存器 .

  指定寄存器的值左移 β 位 , 结果赋给指定寄存器 .

  指定寄存器的值赋给 α.

[说明]

- 没有寄存器指定时

  α 的值只能输入 A .

  β 的值可输入 1 至 7 的数字 .

- 有寄存器指定时

  α 的值可输入 MOV.

  β 的值可输入 1 至 7 的数字 .

  指定寄存器的值只可输入 A .

[生成指令]

(1) 没有寄存器指定时

ROL 指令输出 β 次后生成 AND 指令 .

```
ROL     A , 1
        :
AND     A , #LOW ( 0FFH SHL β )
```

(2) 有寄存器指定时

```
MOV     A , α
ROL     A , 1
        :
AND     A , #LOW ( 0FFH SHL β )
MOV     α , A
```

[使用举例]

(1) 没有寄存器指定时

输出源程序	输入源程序
ROL A , 1 ROL A , 1 ROL A , 1 ROL A , 1 AND A , #LOW (0FFH SHL 4)	A <<= 4

(2) 有寄存器指定时

输出源程序 e	输入源程序
MOV A , CCV ROL A , 1 ROL A , 1 ROL A , 1 ROL A , 1 AND A , #LOW (0FFH SHL 4) MOV CCV , A	CCV <<= 4 (A)

4.3 计数语句

计数语句累加 (++)

(9) 累加 (++)

[描述格式]

[Δ] [长度指定] [Δ] α [Δ] ++

[功能]

- α 的值加 1.

[说明]

- α 的值可输入 INC 或 INCW.

[生成指令]

INC α

依据操作数可生成 INCW 指令 .

关于 α 的详情 , 参见 表 4-11.

[使用举例]

输出源程序	输入源程序
INC H INC CNT INCW HL	H++ CNT++ HL++

表 4-11 累加生成指令

符号			生成指令
α	a	CY	
	b	位符号	
	c	[HL].β	
	d	字节用户符号	*1
	e	字节数据	*1
	f	A	*1
	g	字节寄存器	*1
	h	R0	*1
	i	R1	*1
	j	sfr	
	k	PSW	
	l	字用户符号	
	m	字数据	
	n	AX	*2
	o	字寄存器	*2
	p	RP0	*2
	q	sfrp	
	r	SP	
	s	直接访问符号	
	t	间接访问符号	
	u	[DE]	
	v	立即符号	

*1 : 生成 INC 指令 .

*2 : 生成 INCW 指令 .

空单元格表示错误 .

计数语句递减 (--)

(10) 递减 (--)

[描述格式]

[Δ] [长度指定] [Δ] α [Δ] --

[功能]

- α 的值减 1.

[说明]

- α 的值可输入 DEC 或 DECW.

[生成指令]

DEC α

根据操作数可生成 DECW .

关于 a 的详情 , 参见表 4-12.

[使用举例]

输出源程序	输入源程序
DEC H DEC CNT DECW HL	H-- CNT-- HL--

表 4-12 递减生成指令

符号			生成指令
α	a	CY	
	b	位符号	
	c	[HL].β	
	d	字节用户符号	*1
	e	字节数据	*1
	f	A	*1
	g	字节寄存器	*1
	h	R0	*1
	i	R1	*1
	j	sfr	
	k	PSW	
	l	字用户符号	
	m	字数据	
	n	AX	*2
	o	字寄存器	*2
	p	RP0	*2
	q	sfrp	
	r	SP	
	s	直接访问符号	
	t	间接访问符号	
	u	[DE]	
	v	立即符号	

*1 : 生成 DEC 指令 .

*2 : 生成 DECW 指令 .

空单元格表示错误 .

4.4 交换语句

交换语句交换 (<->)

(11) 交换 (<->)

[描述格式]

```
[ Δ ] [ 长度指定 ] [ Δ ] α [ Δ ] <-> [ Δ ] [ 长度指定 ] [ Δ ] β [ Δ ] [ ( 寄存器指定 ) ]
```

[功能]

- 没有寄存器指定时

  α 和 β 的值交换 .

- 有寄存器指定时

  α 的值赋给指定寄存器 .

  指定寄存器的值和 β 的值交换 .

  指定寄存器的值赋给 α.

[说明]

- 没有寄存器指定时

  α 和 β 的值可输入 XCH 或 XCHW.

- 有寄存器指定时

  α 的值可输入 MOV 和 MOVW.

  β 的值可输入 XCH 和 XCHW.

[生成指令]

(1) 没有寄存器指定时

```
XCH     α , β
```

根据操作数可生成 XCHW .

(2) 有寄存器指定时

```
MOV     Specified register , α
XCH     Specified register , β
MOV     α , specified register
```

根据操作数可生成 XCHW .

关于 α 和 β 的组合详细情况 , 参见表 4-13.

α 表示指定寄存器 .

[使用举例]

(1) 没有寄存器指定时

输出源程序	输入源程序
XCH A , B XCHW AX , BC	A <-> B AX <-> BC

(2) 有寄存器指定时

输出源程序	输入源程序
MOV A , DATA XCH A , B MOV DATA , A MOVW AX , DE XCHW AX , BC MOVW DE , AX	DATA <-> B (A) DE <-> BC (AX)

表 4-13 交换生成指令

符号			β																				
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
α	a	CY																						
	b	位符号																						
	c	字节用户符号																						
	d	[HL].β																						
	e	字节数据																						
	f	A			*1	*1		*1			*1		*1									*1	*1	
	g	字节寄存器																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	字用户符号																						
	m	字数据																						
	n	AX															*2							
	o	字寄存器																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接访问符号																						
	t	间接访问符号																						
	u	[DE]																						
	v	立即符号																						

*1 : 生成 XCH 指令 .

*2 : 生成 XCHW 指令 .

空单元格表示错误 .

4.5 位操作语句

位操作语句置位 (=)

(12) 置位 (=)

[描述格式]

[Δ] α_1 [Δ] [= [Δ] α_2 [Δ] ...] = [Δ] 1 [Δ] [(CY 指定)] 在右侧输入 "1" .

[功能]

- 没有 CY 指定时

  α_n 置位 (为 "1").

- 有 CY 指定时

  CY 和 α_n 置位 (为 "1").

[说明]

- α_n 的值可输入 SET1 指令 .

  一行中最多可有 32 个操作符 "=" . 输入多于 32 个操作符将会发生错误 . 如果序列赋值期间发生错误 , 则不生成指令 .

[生成指令]

(1) 没有 CY 指定时

SET1 α_1

(2) 没有 CY 指定且序列赋值时

SET1 α_n SET1 α_{n-1} : SET1 α_2 SET1 α_1

(3) 有 CY 指定时

SET1 CY SET1 α_1

(4) 有 CY 指定且序列赋值时

```
SET1    CY
SET1    αn
SET1    αn-1
        :
SET1    α2
SET1    α1
```

详情参见表 4-14.

[使用举例]

(1) 没有 CY 指定时

输出源程序	输入源程序
SET1 A.3 SET1 CY SET1 BIT3 SET1 BIT2 SET1 BIT1	A.3 = 1 CY = 1 BIT1 = BIT2 = BIT3 = 1

(2) 有 CY 指定时

输出源程序	输入源程序
SET1 CY SET1 A.5 SET1 CY SET1 BIT3 SET1 BIT2 SET1 BIT1	A.5 = 1 (CY) BIT1 = BIT2 = BIT3 = 1 (CY)

表 4-14 置位生成指令

符号			生成指令
α	a	CY	*1
	b	位符号	*1
	c	[HL].β	*1
	d	字节用户符号	*1
	e	字节数据	
	f	A	
	g	字节寄存器	
	h	R0	
	i	R1	
	j	sfr	
	k	PSW	
	l	字用户符号	*1
	m	字数据	
	n	AX	
	o	字寄存器	
	p	RP0	
	q	sfrp	
	r	SP	
	s	直接访问符号	
	t	间接访问符号	
	u	[DE]	
	v	立即符号	

***1** : 生成 SET1 指令 .

空单元格表示错误 .

位操作语句复位 (=)

(13) 复位 (=)

[描述格式]

```
[ Δ ] α1 [ = [ Δ ] α2 [ Δ ] ... ] = [ Δ ] 0 [ Δ ] [ ( CY 指定 ) ]
在右侧输入 "0".
```

[功能]

- 没有 CY 指定时

  α_n 复位 (为 "0").

- 有 CY 指定时

  CY 和 α_n 复位 (为 "0").

[说明]

- α_n 的值可输入 CLR1 指令 .

  一行中最多可有 32 个操作符 "=" . 输入多于 32 个操作符将会发生错误 . 如果序列赋值期间发生错误 , 则不生成指令 .

[生成指令]

(1) 没有 CY 指定时

```
CLR1    α1
```

(2) 没有 CY 指定且序列赋值时

```
CLR1    αn
CLR1    αn-1
        :
CLR1    α2
CLR1    α1
```

(3) 有 CY 指定时

```
CLR1    CY
CLR1    α1
```

(4) 有 CY 指定且序列赋值时

```
CLR1    CY
CLR1    αn
CLR1    αn-1
        :
CLR1    α2
CLR1    α1
```

详情参见表 4-15.

[使用举例]

(1) 没有 CY 指定时

输出源程序	输入源程序
CLR1 A.3 CLR1 CY CLR1 BIT3 CLR1 BIT2 CLR1 BIT1	A.3 = 0 CY = 0 BIT1 = BIT2 = BIT3 = 0

(2) 有 CY 指定时

输出源程序	输入源程序
CLR1 CY CLR1 A.5 CLR1 CY CLR1 BIT3 CLR1 BIT2 CLR1 BIT1	A.5 = 0 (CY) BIT1 = BIT2 = BIT3 = 0 (CY)

表 4-15 复位生成指令

符号			生成指令
α	a	CY	*1
	b	位符号	*1
	c	[HL].β	*1
	d	字节用户符号	*1
	e	字节数据	
	f	A	
	g	字节寄存器	
	h	R0	
	i	R1	
	j	sfr	
	k	PSW	
	l	字用户符号	*1
	m	字数据	
	n	AX	
	o	字寄存器	
	p	RP0	
	q	sfrp	
	r	SP	
	s	直接访问符号	
	t	间接访问符号	
	u	[DE]	
	v	立即符号	

***1**： 生成 CLR1 指令 .

空单元格表示错误 .

第五章　　伪指令

本章介绍伪指令．这里，" 伪指令 " 表示 ST 78K0S 需要执行一系列处理的各种指令．

5.1　概述

伪指令作为 ST78K0S 需要执行一系列处理所需要的各种指令而输入源程序．

使用伪指令使源程序编码变得容易．

伪指令不在输出文件中输出．

5.2 伪指令功能

以下介绍不同伪指令的功能 .

使用举例显示为输入源文件生成指令的注释语句 .

表 5-1 伪指令列表

伪指令类型	伪指令名称
符号定义伪指令 (#define)	`#define`

条件处理伪指令 (#ifdef / #else / #endif)	`#ifdef`
	`  :`
	`#else`
	`  :`
	`#endif`

包含伪指令 (#include)	`#include`

CALLT 替代伪指令 (#defcallt)	`#defcallt`
	`  :`
	`#endcallt`

#DEFINE

(1) 符号定义伪指令 (#define)

[描述格式]

```
[ Δ ] # [ Δ ] define Δ 符号 Δ 字符串
```

[功能]

- 该伪指令用指定的字符串替代已经输入到源程序内的符号 .

[说明]

- "#" 字符必须是符号的起始符 , 除非以空格或水平制表符起始 .
- 符号以一个英文字母开始 , 其由英语字母和数字构成 . 前 31 个字符有效 . 如果指定符号超过 31 个字符 , 则忽略第 32 及以后的字符 .
- 定义字符串的字符取自 "**2.2（1）字符组** " 列表中的字符组中的字符 . 字符串不能包含空格或引号标记 . 任何包含空格或引号标记的字符串在处理中将被忽略 .
- 该伪指令有益于编写易于读取的符号 , 比如数值 .
- 保留字不能用作符号输入 .
- 保留字可作为字符串输入 .
- 如同一符号被定义两次 , 则输出告警信息 .
- 输出已经被转换到辅助源文件的字符串 . 不输出 #define 语句 .
- 如一个被转换字符串被另一 #define 语句定义 , 最多可再被转换 31 次 . 在第 32 次转换时输出错误信息 , 且在后续转换中忽略该定义 .
- 该伪指令可在源代码的任何位置输入 .
- 当输入两个或更多指定选项 "-d" 的符号时 , 输出警告信息 , 但 #define 有效 .

[使用举例]

输出源程序	输入源程序

MOV X , #0
CALL !xxx
MOV A , X
CMP A , #TRUE
BNZ $?L1
MOV B , #0C5H
?L1 :
	#define TRUE 1
	X = #0
	CALL !xxx
	if (X == #TRUE) (A)
	
	B = #0C5H
	endif

#IFDEF / #ELSE / #ENDIF

(2) 条件处理伪指令 (#ifdef / #else / #endif)

[描述格式]

```
[ Δ ] # [ Δ ] ifdef Δ 符号
        文本 1
[ Δ ] # [ Δ ] else
        文本 2
[ Δ ] # [ Δ ] endif
```

[功能]

- 该伪指令执行条件处理 .

  (1) 当符号未被定义时

  如果输入了 #else, 则跳过文本 1, 文本 2 成为处理对象 .

  (2) 当符号已被定义时

  如果输入了 #else, 则文本 1 成为处理对象 , 跳过文本 2.

[说明]

- "#" 字符必须是符号的起始符 , 除非以空格或水平制表符起始 .
- 符号以一个英文字母开始 , 其由英语字母和数字构成 . 前 31 个字符有效 .
- 通过已经输入的 #define 语句 , 或者在启动时指定 "-d" 选项来定义这些符号 .
- 该伪指令可最多嵌套 8 级 .
- #else 可被省略 .

[使用举例]

- 在命令行中输入以下内容时 (符号已经被定义)

  C > st78k0s -cP9014 sample.st -dSYM

输出源程序	输入源程序

MOV A , #00H	#ifdef SYM
	A = #00H
	#else
	A = #0FFH
	#endif

- 在命令行中输入以下内容时 (符号未被定义)

  C > st780 -cP9014 sample.st

输出源程序	输入源程序

MOV A , #0FFH	#ifdef SYM
	A = #00H
	#else
	A = #0FFH
	#endif

#INCLUDE

(3) 包含伪指令 (#include)

[描述格式]

```
[ Δ ] # [ Δ ] include Δ " 文件名 "
```

[功能]

- 该行被指定的文件名代替 , 并成为 ST78K0S 源程序的一个处理对象 .

[说明]

- "#" 字符必须是符号的起始符 , 除非以空格或水平制表符起始 .
- 该伪指令可在源程序的任何行输入 .
- Include 伪指令不能在包含文件中输入 . 也就是说 , 不允许嵌套 include 伪指令 .
- 在启动时指定的输入源文件名 , 输出文件名和错误文件名不能指定为该伪指令的文件名 .
- 文件名之前可输入驱动器名和目录名 . 若没有输入驱动器和目录 , 处理时假定 include 文件属于当前驱动器和当前目录 .
- 当 ST78K0S 启动时 ,"-i" 选项可用于为 include 文件指定驱动器和目录 .

[使用举例]

输出源程序	输入源程序

MOV A , #08H
MOV B , #0AH
	#include "sample.inc"
	A = SYM1 ; #define SYM1 #08H
	B = SYM2 ; #define SYM2 #0AH

#DEFCALLT

(4) CALLT 替代伪指令 (#defcallt)

[描述格式]

```
[ Δ ] # [ Δ ] defcallt Δ CALLT table label
[ Δ ] CALL Δ ! label
[ Δ ] # [ Δ ] endcallt
```

[功能]

- 已注册标签的 CALL 指令被 CALLT 指令所代替 , 并输出至辅助文件 .

[说明]

- 该伪指令定义已被注册至 CALLT 表的标签 , 与输入至源程序的 CALL 指令相反 . 所有已定义标签的 CALL 指令被 CALLT 标签所代替 .
- 该伪指令最多可定义 32 次 . 定义第 33 次时输出错误信息 , 且在继续处理时忽略该次定义 .
- 如果同样的模式被定义了两次 , 则输出错误信息 , 且在继续处理时忽略该次定义 .

[使用举例]

输出源程序	输入源程序

MOV R0 , #0
CALLT [@ABC]
CALL !LABEL
CALL_T CSEG AT 40H
@ABC : DW ABC
	#DEFCALLT @ABC
	CALL !ABC
	#ENDCALLT
	R0 = #0
	CALL !ABC
	CALL !LABEL
	CALL_T CSEG AT 40H
	@ABC : DW ABC

第六章　　控制指令

本章介绍结构化汇编器的控制指令．控制指令为结构化汇编器的操作提供详细的指令．

6.1　概述

控制指令输入至源程序，可为 ST78K0S 执行一系列的操作处理设置各种指令．

输入控制指令可节省在激活程序时需要指定选项所耗费的时间．

6.2 汇编器控制指令

首先，必须确定是否每个汇编器控制指令都可以在模块头输入 .

如果存在不能在模块头输入的汇编器控制指令，则后续的处理操作按照在模块体的方式继续进行 . 如果一个只能在模块头输入的汇编器控制指令却输入到模块体，则输出错误信息，处理异常中止 .

除了处理器类型说明控制指令 ($PROCESSOR, $PC) 和 kanji 码指定控制指令 ($KANJICODE), 该预处理器不确认指定参数的正确性 . 有关其它控制指令编码格式的说明，参见 RA78K0S 汇编包语言用户手册 .

表 6-1 列出了仅能在模块头输入的控制指令 .

表 6-2 列出了可作为模块体的控制指令 .

表 6-1 仅能在模块头输入的控制指令

控制指令
[Δ] $ [Δ] PROCESSOR [Δ] ([Δ] 模块名 [Δ])
[Δ] $ [Δ] PC ([Δ] 模块名 [Δ])
[Δ] $ [Δ] DEBUG
[Δ] $ [Δ] DG
[Δ] $ [Δ] NODEBAG
[Δ] $ [Δ] NODG
[Δ] $ [Δ] DEBUGA
[Δ] $ [Δ] NODEBAGA
[Δ] $ [Δ] XREF
[Δ] $ [Δ] XR
[Δ] $ [Δ] NOXREF
[Δ] $ [Δ] NOXR
[Δ] $ [Δ] TITLE [Δ] ([Δ] ' 标题串 ' [Δ])
[Δ] $ [Δ] TT [Δ] ([Δ] ' 标题串 ' [Δ])
[Δ] $ [Δ] SYMLIST
[Δ] $ [Δ] NOSYMLIST
[Δ] $ [Δ] FORMFEED
[Δ] $ [Δ] NOFORMFEED
[Δ] $ [Δ] WIDTH [Δ] ([Δ] 常量 [Δ])
[Δ] $ [Δ] LENGTH [Δ] ([Δ] 常量 [Δ])
[Δ] $ [Δ] TAB [Δ] ([Δ] 常量 [Δ])
[Δ] $ [Δ] KANJICODE Δ kanji 码

表 6-2 可作为模块体的控制指令

控制指令
[Δ] $ [Δ] INCULUDE [Δ] ([Δ] 文件名 [Δ])
[Δ] $ [Δ] IC ([Δ] 文件名 [Δ])
[Δ] $ [Δ] EJECT
[Δ] $ [Δ] EJ
[Δ] $ [Δ] LIST
[Δ] $ [Δ] LI
[Δ] $ [Δ] NOLIST
[Δ] $ [Δ] NOLI
[Δ] $ [Δ] GEN
[Δ] $ [Δ] NOGEN
[Δ] $ [Δ] COND
[Δ] $ [Δ] NOCOND
[Δ] $ [Δ] SUBTITLE [Δ] ([Δ] ' 字符串 ' [Δ])
[Δ] $ [Δ] ST [Δ] ([Δ] ' 字符串 ' [Δ])
[Δ] $ [Δ] SET [Δ] ([Δ] switch 名 [[Δ] : [Δ] switch 名 ... [Δ])
[Δ] $ [Δ] RESET [Δ] ([Δ] switch 名 [[Δ] : [Δ] switch 名 ... [Δ])
[Δ] $ [Δ] IF [Δ] ([Δ] switch 名 [[Δ] : [Δ] switch 名 ... [Δ])
[Δ] $ [Δ] _IF Δ 条件表达式
[Δ] $ [Δ] ELSEIF [Δ] ([Δ] switch 名 [[Δ] : [Δ] switch 名 ... [Δ])
[Δ] $ [Δ] _ELSEIF Δ 条件表达式
[Δ] $ [Δ] ELSE
[Δ] $ [Δ] ENDIF

6.3 控制指令功能

在以下表 6-3 所示为控制指令的不同功能 .

表 6-3 控制指令列表

控制指令的类型	控制指令
处理器类型说明指令	$PROCESSOR / $PC
Kanji 码说明控制指令	$KANJICODE

以下介绍三种类型控制指令的功能 .

$PROCESSOR / $PC

(1) 处理器类型说明指令 ($PROCESSOR / $PC)

[编写格式]

```
[ Δ ] $ [ Δ ] PROCESSOR [ Δ ] ( [ Δ ] 模型名称 [ Δ ] )
[ Δ ] $ [ Δ ] PC [ Δ ] ( [ Δ ] 模型名称 [ Δ ] )       ; 缩写格式
```

[功能]

- 该控制指令指定了源模块中用于汇编的目标体的模型 .

[说明]

- 尽管该控制指令指定了被汇编器汇编的目标体的模型 , 它还可用来指定结构化编译器目标体的模型 .
- 如果指定的类型与 “-C” 选项指定的类型不同 , 则 “-C” 选项指定的类型优先 . 出现这种冲突时将输出告警信息 . 输入源文件控制指令中的 “$” 被输出的辅助源文件中的 “;” 代替 ; 由选项指定的模型作为处理器模型说明控制指令输出 . 如果由 "-c" 选项指定的是同一个类型名称则不输出信息 . 如果没有通过 "-c" 选项进行指定 , 则必须在源模块的开始部分 (不包括空格或注释) 输入该指定 .
- 当多次输入该控制指令时会出现错误 .
- 如果该控制指令和 "-c" 选项都没有用来指定模型名称 , 则会出现错误 .
- 如果在模块头之外的地方输入该控制指令 , 则会出现错误 .

[编码示例]

```
$PROCESSOR ( P9014 )
$PC ( P9014 )
```

$KANJICODE

(2) Kanji 码指定控制指令 ($KANJICODE)

[编写格式]

```
[ Δ ] $ [ Δ ] KANJICODE Δ kanji码
```

- 缺省假定

  Windows : $KANJICODE SJIS

  Solaris : $KANJICODE EUC

[功能]

- 注释中使用的 kanji 码说明如下 .

表 6-4 Kanji 码的说明

Kanji 码	说明
SJIS	解释为 SHIFT-JIS 码
EUC	解释为 EUC 码
NONE	不解释为 kanji 码

[说明]

- 该控制指令可在输入源文件的模块头部分输入 .
- 如果在模块头之外的地方输入该控制指令 , 则会出现错误 .
- 如果该控制指令被输入多次 , 则最近一次的输入优先 .
- 该预处理器将指定的控制指令输出至辅助源文件 .

  SJIS : $KANJICODE SJIS

  EUC : $KANJICODE EUC

  NONE : $KANJICODE NONE

  如果在辅助源文件中输入同一个控制指令 , 则该控制指令不做输出 . 但是 , 会执行错误检查 .

- Kanji 码的指定按照优先级排列如下 .

  1. 指定 -zs/-ze/-zn 选项
  2. 指定 kanji 码指定控制指令 ($KANJICODE)
  3. 指定环境变量 LANG78K
  4. 各操作系统的缺省指定

[编码示例]

```
$KANJICODE SJIS
```

附录 A　　语法列表

表 A-1　控制语句

控制语句	编码格式

if 语句
if ~ elseif ~ else ~ endif
	if (条件表达式 1) [(寄存器名)]
	　　if 程序块
	elseif (条件表达式 2) [(寄存器名)]
	　　elseif 程序块
	else
	　　else 程序块
	endif

switch 语句 switch ~ case ~ default ~ ends	switch (符号) [(寄存器名)] 　　case 常量 1 : 　　　　case1 程序块 　　case 常量 2 : 　　　　case2 程序块 　　　　: 　　case 常量 N : 　　　　caseN 程序块 　　default : 　　　　default 程序块 ends
for 语句 for ~ next	for (表达式 ; 条件表达式 ; 表达式) [(寄存器名)] 　　指令组 next
while 语句 while ~ endw	while (条件表达式) [(寄存器名)] 　　指令组 endw
until 语句 repeat ~ until	repeat 　　指令组 until (条件表达式) [(寄存器名)]
break 语句 break	break
continue 语句 continue	continue
goto 语句 goto	goto 标记

if_bit 语句
if_bit ~ elseif_bit ~ else ~ endif
	if_bit (条件表达式 1) [(寄存器名)]
	　　if_bit 程序块
	elseif_bit (条件表达式 2) [(寄存器名)]
	　　elseif_bit 程序块
	else
	　　else 程序块
	endif

表 A-1 控制语句

控制语句	编码格式
while_bit 语句 while_bit ~ endw	while_bit (条件表达式) [(寄存器名)] 指令组 endw
until_bit 语句 repeat ~ until_bit	repeat 指令组 until_bit (条件表达式) [(寄存器名)]

表 A-2 条件表达式

条件表达式	编码格式	功能
等于 (==)	α == β	当 α = β 时为真 , 当 α ≠ β 时为假
不等于 (!=)	α != β	当 α ≠ β 时为真 , 当 α = β 时为假
小于 (<)	α < β	当 α < β 时为真 , 当 α >= β 时为假
大于 (>)	α > β	当 α > β 时为真 , 当 α <= β 时为假
大于等于 (>=)	α >= β	当 α >= β 时为真 , 当 α < β 时为假
小于等于 (<=)	α <= β	当 α <= β 时为真 , 当 α > β 时为假
无限循环 (forever)	forever	对 loop 语句设置无限循环
正逻辑 (bit) 位符号	位符号	当指定位符号的值是 1 时为真
负逻辑 (bit) ! 位符号	! 位符号	当指定位符号的值是 0 时为真
逻辑与 (&&)	条件表达式 1 && 条件表达式 2	当条件表达式 1 和 条件表达式 2 都为真时结果为真
逻辑或 (||)	条件表达式 1 || 条件表达式 2	当条件表达式 1 和 条件表达式 2 有一个为真时结果为真

表 A-3 表达式

表达式		编码格式	功能
赋值 (=)	赋值	α = β	α <- β
	赋值 (带寄存器定义)	α = β (γ)	(γ) <- β, α <- (γ)
	序列赋值	α_1 = ... = α_n = β	α_1 <- β, ..., α_n <- β
	序列赋值 (带寄存器定义)	α_1 = ... = α_n = β (γ)	γ <- β, α_1 <- γ, ..., α_n <- γ
增量赋值 (+=)	增量赋值	α += β	α <- α + β
	增量赋值 (带寄存器定义)	α += β (寄存器)	γ <- α, γ <- γ + β, α <- γ
	增量赋值 (带寄存器定义)	α += β , CY	α <- α + β, CY
	增量赋值 (带寄存器定义)	α += β , CY (寄存器)	γ <- α, γ <- γ + β, CY, α <- γ
减量赋值 (-=)	减量赋值	α -= β	α <- α - β
	减量赋值 (带寄存器定义)	α -= β (寄存器)	γ <- α, γ <- γ - β, α <- γ
	减量赋值 (带寄存器定义)	α -= β , CY	α <- α - β, CY
	减量赋值 (带寄存器定义)	α -= β , CY (寄存器)	γ <- α, γ <- γ - β, CY, α <- γ
逻辑与赋值 (&=)	逻辑与赋值	α &= β	α <- α ∩ β
	逻辑与赋值 (带寄存器定义)	α &= β (寄存器)	γ <- α, γ <- γ ∩ β, α <- γ
逻辑或赋值 (|=)	逻辑或赋值	α |= β	α <- α ∪ β
	逻辑或赋值 (带寄存器定义)	α |= β (寄存器)	γ <- α, γ <- γ ∪ β, α <- γ
逻辑异或赋值 (^=)	逻辑异或赋值	α ^= β	α <- α ^ β
	逻辑异或赋值 (带寄存器定义)	α ^= β (寄存器)	γ <- α, γ <- γ ^ β, α <- γ
右移赋值 (>>=)	右移 (循环) 赋值	α >>= β	(α 右移 β 位)
	右移赋值 (带寄存器定义)	α >>= β (寄存器)	γ <- α, (γ 右移 β 位), α <- γ
左移赋值 (<<=)	左移赋值	α <<= β	(α 左移 β 位)
	左移赋值 (带寄存器定义)	α <<= β (寄存器)	γ <- α, (γ 左移 β 位), α <- γ
递加 (++)	递加	α ++	α <- α + 1
递减 (--)	递减	α --	α <- α - 1

表 A-3 表达式

表达式		编码格式	功能
交换 (<->)	交换	α <->= β	α <- α <->= β
	交换 (带寄存器定义)	α <->= β (γ)	γ <- α, γ <- γ <-> β, α <- γ
置位 (=)	置位	α = 1	α <- 1
	置位 (带寄存器定义)	α = 1 (CY)	CY <- 1, α <- 1
	序列置位	α_1 = ... = α_n = 1	α_n <- 1, ..., α_1 <- 1
	序列置位 (带寄存器定义)	α_1 = ... = α_n = 1 (CY)	CY <- 1, α_n <- 1, ..., α_1 <- 1
复位 (=)	复位	α = 0	α <- 0
	复位 (带寄存器定义)	α = 0 (CY)	CY <- 0, α <- 0
	序列复位	α_1 = ... = α_n = 0	α_n <- 0, ..., α_1 <- 0
	序列复位 (带寄存器定义)	α_1 = ... = α_n = 0 (CY)	CY <- 0, α_n <- 0, ..., α_1 <- 0

表 A-4 伪指令

伪指令	编码格式
#define 符号定义伪指令 (#define)	`#define 符号字符串`

#ifdef
条件处理伪指令 (#ifdef / #else / #endif)
	`#ifdef 符号 l`
	`  文本 1`
	`#else`
	`  文本 2`
	`#endif`

#include 包含伪指令 (#include)	`#include " 文件名 "`

#defcallt
CALLT 置换伪指令 (#defcallt)
	`#defcallt CALLT 表格标签`
	`  CALL 标签`
	`#endcallt`

表 A-5 控制指令

控制指令	编码格式
处理器类型指定指令 ($PROCESSOR / $PC)	`$PROCESSOR ( 模型名 )`
Kanji 码指定控制指令 ($KANJICODE)	`$KANJICODE Kanji 码`

附录 B　　生成指令列表

表 B-1　比较表达式的生成指令

比较表达式		生成指令	控制语句 条件
等于 (==)	α == β	CMP (W)　　α , β BNZ　　$?LFALSE	小写字母
		CMP (W)　　α , β BZ　　$?LTRUE BR　　?LFALSE ?LTRUE :	大写字母
	α == β (γ)	MOV (W)　　γ , α CMP (W)　　γ , β BNZ　　$?LFALSE	小写字母
		MOV (W)　　γ , α CMP (W)　　γ , β BZ　　$?LTRUE BR　　LFALSE ?LTRUE :	大写字母
不等于 (!=)	α != β	CMP (W)　　α , β BZ　　$?LFALSE	小写字母
		CMP (W)　　α , β BNZ　　$?LTRUE BR　　?LFALSE ?LTRUE :	大写字母
	α != β (γ)	MOV (W)　　γ , α CMP (W)　　γ , β BZ　　$?LFALSE	小写字母
		MOV (W)　　γ , α CMP (W)　　γ , β BNZ　　$?LTRUE BR　　?LFALSE ?LTRUE :	大写字母
小于 (<)	α < β	CMP (W)　　α , β BNC　　$?LFALSE	小写字母
		CMP (W)　　α , β BC　　$?LTRUE BR　　?LFALSE ?LTRUE :	大写字母
	α < β (γ)	MOV (W)　　γ , α CMP (W)　　γ , β BNC　　$?LFALSE	小写字母
		MOV (W)　　γ , α CMP (W)　　γ , β BC　　$?LTRUE BR　　?LFALSE ?LTRUE :	大写字母

表 B-1 比较表达式的生成指令

比较表达式		生成指令	控制语句 条件
大于 (>)	α > β	CMP (W) α , β BZ $?LFALSE BC $?LFALSE	小写字母
		CMP (W) α , β BZ $$ + 4 BNC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	α > β (γ)	MOV (W) 指定寄存器 , α CMP (W) 指定寄存器 , β BZ $?LFALSE BC $?LFALSE	小写字母
		MOV (W) 指定寄存器 , α CMP (W) 指定寄存器 , β BZ $$ + 4 BNC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
大于等于 (>=)	α >= β	CMP (W) α , β BC $?LFALSE	小写字母
		CMP (W) α , β BNC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	α >= β (γ)	MOV (W) γ , α CMP (W) γ , β BC $?LFALSE	小写字母
		MOV (W) γ , α CMP (W) γ , β BNC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母

表 B-1 比较表达式的生成指令

比较表达式		生成指令	控制语句 条件
小于等于 (<=)	α <= β	CMP (W) α , β BZ $$ + 4 BNC $?LFALSE	小写字母
		CMP (W) α , β BZ $?LTRUE BC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	α <= β (γ)	MOV (W) 指定寄存器 , α CMP (W) 指定寄存器 , β BZ $$ + 4 BNC $?LFALSE	小写字母
		MOV (W) 指定寄存器 , α CMP (W) 指定寄存器 , β BZ $?LTRUE BC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母

γ : 指定寄存器

表 B-2 位校验表达式的生成指令

位校验表达式	生成指令	控制语句 条件
位符号 if_bit (位符号) elseif_bit (位符号) while_bit (位符号 l) until_bit (位符号)	BNC $?LFALSE	小写字母 (CY)
	BNZ $?LFALSE	小写字母 (Z)
	BF 位符号 , $?LFALSE	小写字母
	BC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母 (CY)
	BZ $?LTRUE BR ?LFALSE ?LTRUE :	大写字母 (Z)
	BT 位符号 , $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
! 位符号 if_bit (! 位符号) elseif_bit (! 位符号) while_bit (! 位符号) until_bit (! 位符号)	BC $?LFALSE	小写字母 (CY)
	BZ $?LFALSE	小写字母 (Z)
	BT 位符号 , $?LFALSE	小写字母
	BNC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母 (CY)
	BNZ $?LTRUE BR ?LFALSE ?LTRUE :	大写字母 (Z)
	BF 位符号 , $?LTRUE BR ?LFALSE ?LTRUE :	大写字母

表 B-3 逻辑表达式的生成指令

逻辑表达式		生成指令	控制语句 条件
逻辑与 (&&)	α == β &&	CMP (W) α , β BNZ $?LFALSE	小写字母
		CMP (W) α , β BZ $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	α != β &&	CMP (W) α , β BZ $?LFALSE	小写字母
		CMP (W) α , β BNZ $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	α < β &&	CMP (W) α , β BNC $?LFALSE	小写字母
		CMP (W) α , β BC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	α > β &&	CMP (W) α , β BZ $?LFALSE BC $?LFALSE	小写字母
		CMP (W) α , β BZ $$ + 4 BNC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	α >= β &&	CMP (W) α , β BC $?LFALSE	小写字母
		CMP (W) α , β BNC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	α <= β &&	CMP (W) α , β BZ $$ + 4 BNC $?LFALSE	小写字母
		CMP (W) α , β BZ $?LTRUE BC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母

表 B-3 逻辑表达式的生成指令

逻辑表达式		生成指令	控制语句 条件
逻辑与 (&&)	CY &&	BNC $?LFALSE	小写字母
		BC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	Z &&	BNZ $?LFALSE	小写字母
		BZ $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	位符号 &&	BF 位符号 , $?LFALSE	小写字母
		BT 位符号 , $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	!CY &&	BC $?LFALSE	小写字母
		BNC $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	!Z &&	BZ $?LFALSE	小写字母
		BNZ $?LTRUE BR ?LFALSE ?LTRUE :	大写字母
	! 位符号 &&	BT 位符号 , $?LFALSE	小写字母
		BF 位符号 , $?LTRUE BR ?LFALSE ?LTRUE :	大写字母

表 B-3 逻辑表达式的生成指令

逻辑表达式		生成指令	控制语句 条件
逻辑或 (||)	α == β ||	CMP (W) α , β BZ $?LFALSE	无
	α != β ||	CMP (W) α , β BNZ $?LFALSE	
	α < β ||	CMP (W) α , β BC $?LFALSE	
	α > β ||	CMP (W) α , β BZ $?LFALSE BNC $?LFALSE	
	α >= β ||	CMP (W) α , β BNC $?LFALSE	
	α <= β ||	CMP (W) α , β BZ $?LFALSE BC $?LFALSE	
	CY ||	BC $?LFALSE	
	Z ||	BZ $?LFALSE	
	位符号 ||	BT 位符号 , $?LFALSE	
	!CY ||	BNC $?LFALSE	
	!Z ||	BNZ $?LFALSE	
	! 位符号 ||	BF 位符号 , $?LFALSE	

表 B-4 表达式

表达式		生成指令
赋值 (=)	α = β	MOV α1 , β
		MOVW α1 , β
		BNC ?L1 SET1 α BR ?L2 ?L1 : CLR1 α ?L2 :
	α = β (γ)	MOV γ , β MOV α1 , γ
		MOVW γ , β MOVW α1 , γ
		BF β , ?L1 SET1 α BR ?L2 ?L1 : CLR1 α ?L2 :
增量赋值 (+=)	α += β	ADD α , β
		ADDW α , β
	α += β (γ)	MOV γ , α ADD γ , β MOV α , γ
		MOVW γ , α ADDW γ , β MOVW α , γ
	α += β, CY	ADDC α , β
	α += β, CY (γ)	MOV γ , α ADDC γ , β MOV α , γ
减量赋值 (-=)	α -= β	SUB α , β
		SUBW α , β
	α -= β (γ)	MOV γ , α SUB γ , β MOV α , γ
		MOVW γ , α SUBW γ , β MOVW α , γ
	α -= β, CY	SUBC α , β
	α -= β, CY (γ)	MOV γ , α SUBC γ , β MOV α , γ

表 B-4 表达式

表达式		生成指令
逻辑与赋值 (&=)	α &= β	AND α , β
		BNC ?L1 BF β , ?L1 SET1 CY BR ?L2 ?L1 : CLR1 CY ?L2 :
	α &= β (γ)	MOV γ , α AND γ , β MOV α , γ
		BF α , ?L1 BF β , ?L1 SET1 α BR ?L2 ?L1 : CLR1 α ?L2 :
逻辑或赋值 (|=)	α |= β	OR α , β
		BC ?L1 BF β , ?L2 ?L1 : SET1 CY BR ?L3 ?L2 : CLR1 CY ?L3 :
	α |= β (γ)	MOV γ , α OR γ , β MOV α , γ
		BT α , ?L1 BF β , ?L2 ?L1 : SET1 α BR ?L3 ?L2 : CLR1 α ?L3 :

表 B-4 表达式

表达式		生成指令
逻辑异或赋值 (^=)	α ^= β	XOR α , β
		BNC ?L1 BF β , ?L2 ?L1 : BC ?L3 BF β , ?L3 ?L2 : SET1 CY BR ?L4 ?L3 : CLR1 CY ?L4 :
	α ^= β (γ)	MOV γ , α XOR γ , β MOV α , γ
		BF α , ?L1 BF β , ?L2 ?L1 : BT α , ?L3 BF β , ?L3 ?L2 : SET1 α BR ?L4 ?L3 : CLR1 α ?L4 :
右移赋值 (>>=)	α >>= β	ROR A , 1 : AND A , #0FFH SHR β
	α >>= β (γ)	MOV A , α ROR A , 1 : AND A , #0FFH SHR β MOV α , A
左移赋值 (<<=)	α <<= β	ROL A , 1 : AND A , #LOW (0FFH SHL β)
	α <<= β (γ)	MOV A , α ROL A, 1 : AND A , #LOW (0FFH SHL β) MOV α , A
增量 (++)	α ++	INC α
		INCW α
减量 (--)	α --	DEC α
		DECW α

表 B-4 表达式

表达式		生成指令
交换 (<->)	α <-> β	XCH α , β
		XCHW α , β
	α <-> β (γ)	MOV γ , α XCH γ , β MOV α , γ
		MOVW γ , α XCHW γ , β MOVW α , γ
置位 (=)	α = 1	SET1 α_1
	α = 1 (CY)	SET1 CY SET1 α_1
复位 (=)	α = 0	CLR1 α_1
	α = 0 (CY)	CLR1 CY CLR1 α_1

附录 C　　检索

详细信息请联系：

中国区

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

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

[北京]

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

[上海]

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

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

[深圳]

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

[香港]

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

[成都]

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

[长春]

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

[大连]

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