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 系列

汇编器包

结构化汇编器 语言

ST78K0S 1.00 及升级版

文档号 No. U11623CA1V0UMJ1 (第一版)
发布日期 2007 年 8 月 N CP(K)

日本印刷

[备忘录]

MS-DOS 和 Windows 均为 Microsoft 公司的商标。
IBM PC 和 PC DOS 是国际商用机器公司的商标
HP9000 系列 700 和 HP-UX 是 Hewlett Packard 公司对商标。
SPARC 台是 SPARC International, Inc.的商标。
RISC NEWS 和 NEWS-OS 是 Sony 公司的商标。

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

  “标准等级”、“专业等级”以及“特殊等级”三种质量等级。

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

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

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

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

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

（注）

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

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

区域信息

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

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

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

详细信息请联系：

（中国区）

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

[北京]

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

[深圳]

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

[上海]

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

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

[香港]

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

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，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“**汇编器包用户手册（汇编器语言和操作）**”。

目标读者

本手册面向的读者能理解78K/0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78K/0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。

目标芯片

本汇编器可以用于RA78K0S编译器包所支持的所有芯片。

组成

本手册组成如下。

- 第一章 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）。
- 第二章 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。
- 第三章 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。
- 第四章 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。
- 第五章 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。
- 附录C 最佳性能
  本附录介绍结构化汇编器的最佳性能特征。

用途

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。**

约定

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

目录

图形列表

表格目录(1/2)

表格目录 (2/2)

[备忘录]

第一章 概述

本章描述了 ST78K0S 在微控制器软件发展中的功能（角色等）。

1.1 结构化汇编器概述

RA78K0S 结构化汇编预处理器是“RA78K0S 汇编程序包”中的一个程序，用于 78K/0 系列中紧凑型、多用途的微控制器的软件开发。

在本手册中，该结构化的汇编预处理器简称为“结构化汇编器”或“ST78K0S（结构化汇编器）”。

结构化汇编器将结构化汇编语句，如“if~else~endif”和 “for~next”，转换成汇编语言源程序文件。控制语句用于进入“if~else~endif”和 “for~next”描述。

同样地，结构化汇编器表现出如下三种优势。

<1> 程序编写容易

- 每个程序的结构都可以书写成适于从设计到编码的开发过程。
- 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> 程序易于阅读。

- 程序结构易于理解。
- 内存寄存器之间的操作和转移可输入一条简单语句完成。
- 其他程序员编写的程序也易于阅读。
- 程序维护（修订）容易。

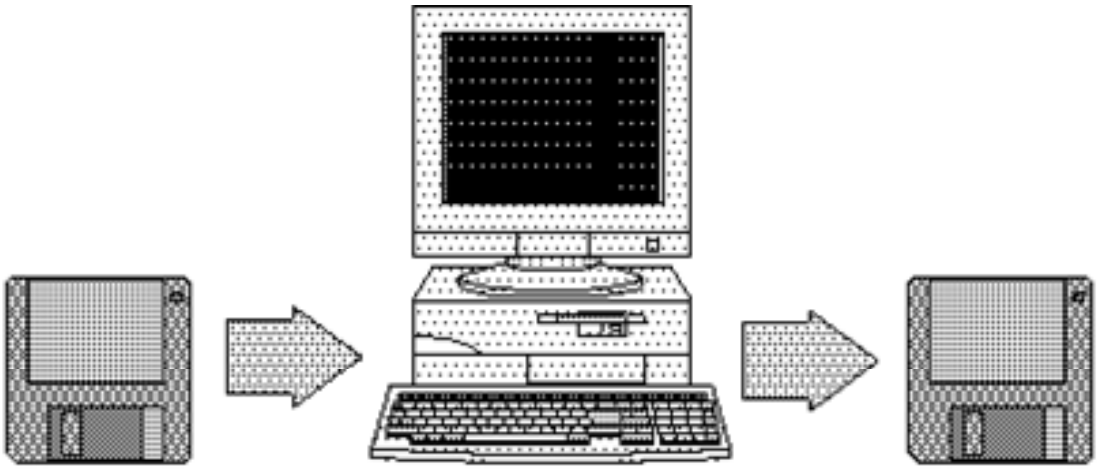
<3> 方便台式机的调试。

- 编码可以在一对一的交流方式中完成，并具有详细的设计，从而方便了**台式机**调试。

1.2 功能概述

首先根据一种特殊的语言说明，编码完成一个结构化汇编器源程序。结构化汇编器分析该源程序中的各种控制语句、表达式和指令，输出一个汇编源程序作为汇编器的输入源文件。

图 1-1. 结构化汇编器功能



用结构化汇编语言编写的源程序　　结构化汇编器　　用纯汇编语言编写的源程序

结构化语句可作为注释输出。修改后的汇编程序指令和常用汇编语言都可以作为辅助源文件输出。

出现错误时输出错误信息。

结构化汇编器的主要功能如下。

<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> 该汇编器包含类-C 符号定义的指令、条件处理函数和 Include 指示。

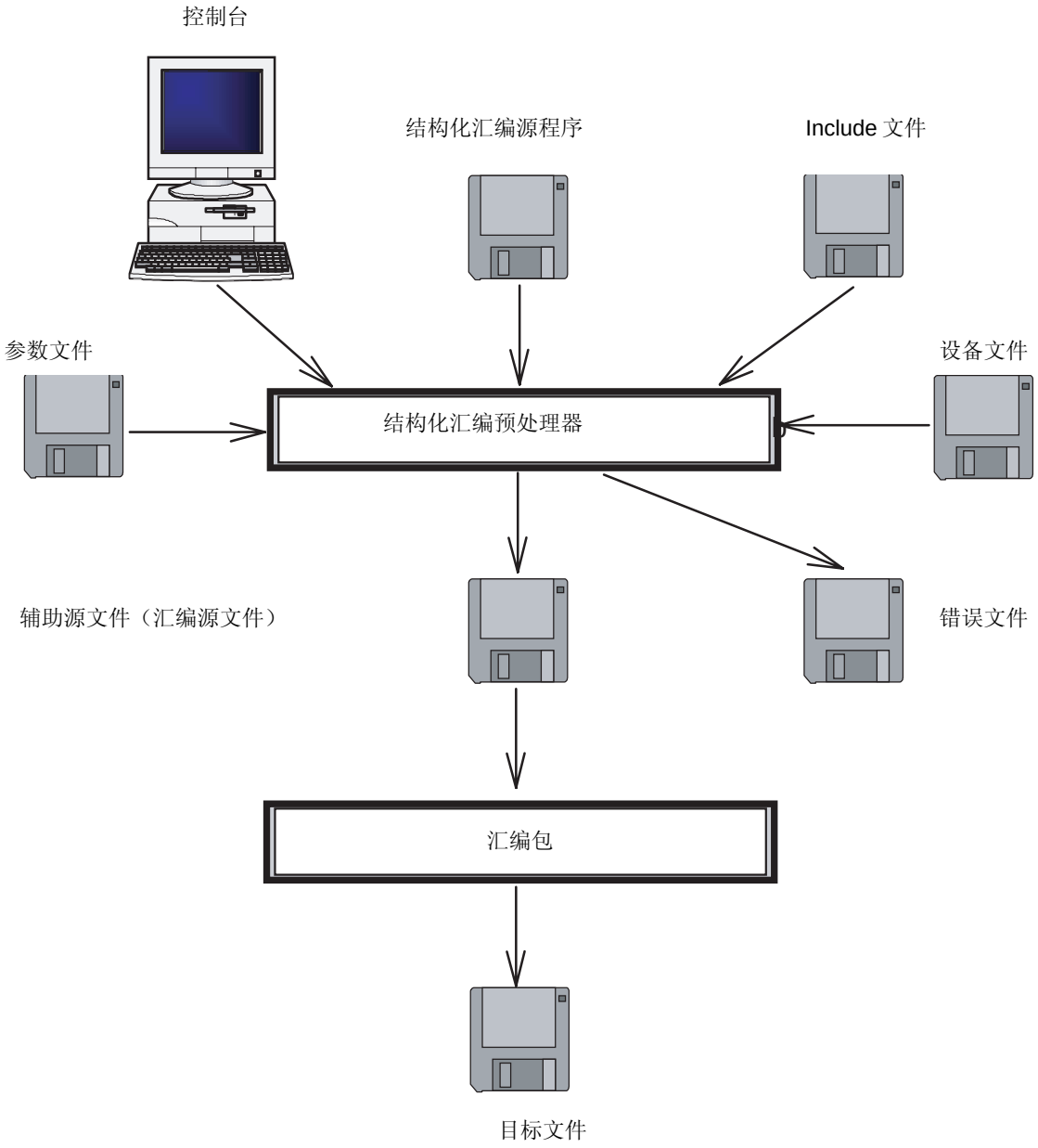
<5> 由于正是预处理器输出了汇编源程序，因此，在结构化汇编器转换之后可进行编码优化。

<6> 指示用来转换成 CALLT 指令，从而在程序开发之后，一些常规程序可被列入 CALLT 表。

<7> 改变汇编器源程序的输出位置可以创建易于阅读的汇编列表。

图 1-2 描述了程序开发的操作流程。

图 1-2. 程序开发流程



警告:　　设备文件是分开销售的 。

1.3 写在程序开发之前

结构化汇编器的最大性能特征如下表所示。请确定，在编写程序之前参考这些参数值。

1.3.1 最佳性能

结构化汇编器的最佳性能的值如下表所示。

表 1-1. 结构化汇编器的最佳性能

项目	最大值
行长度（不包括 LF 或 CR）	218 字符
在#define 指示中注册的符号数(不包括保留字)	512 符号
控制语句的嵌套级数	31 级
#ifdef 指示中的嵌套级数	8 级
#defcallt 指示	32
#include 指示的嵌套	不支持
被#define 指示重定义的次数	31 次
在一个序列中被赋值的操作数的个数	**33**[注解 1]
逻辑操作符的操作数	**17**[注解 2]
被-D 选项定义的符号数	30

注解 **1.** 最大值表示如下：

S1=S2= … S32=S33

最多可插入 33 个符号，其中包括 32 个等号（=）。

2. 最大值表示为：

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 注意事项

(1) 字符号和字节符号

结构化汇编器使用每个用户符号的最后一个字符来确定该符号是字符号还是字节符号。字符号的缺省字符是“P”，可通过-SC 选项改变。

关于-SC 选项的更多细节，参见“**RA78K0S 系列汇编器包操作**”。

示例 1

结构化编译器

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

汇编器

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

示例 2 结构化编译器的启动命令

```
A>ST78K3 INPUT.A -SC@
```

“@”用于指示一个字符号。

输入文件说明

结构化汇编器的命令名

结构化编译器

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

汇编器

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

(2) 标记定义（通过汇编器表示地址的符号）

当定义标记时，需确定，从结构化编译器语句的单独一行进入标记定义。

示例

正确的程序

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

不正确的程序

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

1.3.3 环境变量

LANG78K 指定用于输入注释的 kanji 码类型。

(1) 编码格式

SET∆LANG78K = kanji 码

Kanji 码

SJIS ： Shift JIS 码

EUC ： EUC 码

NONE： 无 kanji 码处理

(2) 功能

- 如果没有设置环境变量，则根据 OS 类型设置 kanji 码定义，如下所示。

  MS-DOS ： SJIS

  PC DOS ： NONE

  SunOS ： EUC

  HP-UX ： SJIS

  NEWS-OS ： SJIS

- kanji 码定义的优先级如下。

  <1> 由-ZS、-ZE 或-ZN 选项指定

  <2> 由 kanji 码定义控制指令（$KANJICODE）指定

  <3> 由 LANG78K 环境变量指定

  <4> 基于 OS 的缺省指定

第二章 源程序编码方法

本章描述源程序的编码方法和格式。

2.1 源程序的基本配置

源程序由结构化汇编语言和（纯）汇编语言组成。
关于汇编语言的更多描述，参见“**RA78K0S 系列汇编包 语言**”。
每行（两个 LF 之间）最多可包含 218 个字符。
结构化汇编语言中使用的编码类型如**表 2-1. 结构化汇编语言编码**所示。

表 2-1. 结构化汇编语言编码

类型			编码
结构汇编语句	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 计数语句 交换语句	赋值（=），赋值加操作（+=，etc），移位（循环） 赋值(>>=, etc.) 增量（++），减量（– –） 交换（<->）
条件表达式		比较表达式 测试位表达式 逻辑操作	==, !=, <, >, >=, <= 位地址，!位地址 逻辑 AND(&&)，逻辑 OR(||)

条件表达式作为控制语句的条件输入。
更多细节参见“**3.5 控制语句的功能**”。

(1) 控制语句

控制语句包括如下语句：表示条件分支的“if”和“switch~case”语句，表示条件循环的“for~next”、“while”和“repeat~until”语句，表示循环退出处理的“break”, “continue”和“goto”语句。更多细节参见“**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）和交换语句。更多细节参见“**第四章 表达式”**。

2.2 源程序的基本元素

(1) 字符组

字母、数字和特殊字符都可以在源程序中使用。

表 2-2. 字母数字混合字符

数字		0 1 2 3 4 5 6 7 8 9
字母	大写	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	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。在首字符后出现的任何小写字母都将转换成大写字母。但是，输出辅助源文件时则根据输入时的大小写规定完成。

表 2-3. 特殊字符

字符	名称	用途
?	问号	作为字母使用的字符
@	单价符号	作为字母使用的字符
_	下划线	作为字母使用的字符
	空格	短语分隔符
HT	水平制表符	作为空格使用的字符
,	逗号	操作数分隔符
.	句号	位符号的比特位置符
“	双引号	#INCLUDE 指示的磁盘类型文件名称的说明符
‘	单引号	标记字符常量的起始和结束的符号
+	加号	正号或加法操作
–	减号	负号或减法操作
&	与符号	逻辑 AND 操作符
|	或符号	逻辑 OR 操作符
^	上箭头	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	注释起始符或控制语句表达式的分隔符
#	算术符号或高音符号（音乐标记中）	结构化汇编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	行符号结束符

出现下列任何无效字符时都将产生错误。

表 2-4. 无效字符

	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

键入无效字符时将提示错误。在输出辅助文件时每个无效字符被一个句号（.）代替。

然而，无效字符可用在注释里。

(2) 标识符

标识符是附着在数字数据、地址等之上的名称。

使用标识符可以源程序内容更易于识别。

使用#define 语句来定义标识符细节（参见“**5.2 指示功能**”）。

(3) 符号

符号名称的最后一个字符决定了结构化汇编器是产生一个字节接入指令还是字接入指令。缺省设置是 P（一对），该设置可通过-SC 选项改变。

除了保留字符号之外的所有字符串都可以作为用户符号处理。所有文字数字混合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任何常量。但是，汇编语言常量可按照同辅助文件的方式输出（有关汇编语言常量的更多细节，参见“**RA78K0S 汇编包用户手册汇编语言**”）。

(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（有关汇编语言表达式的更多细节，参见**"汇编包用户手册汇编语言"**）。

需确定，在一个汇编语言表达式中，任何被空格分开的符号被包含在圆括号内。

举例

<1> 汇编程序的编码方法

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

<2> 结构化汇编源程序的编码方法

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

2.3 保留字

下表列出了结构化汇编语言中的保留字。

有关指令和 sfr 符号的更多信息，请参考目标设备的用户手册。

表 2-5. 保留字符号（1/2）

	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
指示	DFINE
	IFDEF, ELSE, ENDIF
	INCLUDE
	DEFCALLT, ENDCALLT
操作	++, – –
	=, +=, –=, &=, |=, ^=, <<=, >>=, <–>
	==, !=, <, >=, >, <=, FOREVER
汇编操作符	MOD, NOT
	AND, OR, XOR
	EQ, NE, GT, GE, LT, LE
	SHL, SHR
	HIGH, LOW
	DATAPOS, BITPOS, MASK

表 2-5. 保留字符号（2/2）

	保留字
汇编控制指令	PROCESSOR, PC
	DEBAG, NODEBAG, DEBAGA, NODEBAGA, , DG, NODG
	XREF, XR, NOXREF, NOXR
	TITLE, TI
	SYMLEN, NOSYMLEN
	CAP, NOCAP
	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,
寄存器	CY, Z
	A
	R0, R1, R2, R3, R4, R5, R6, R7, X, B, C, D, E, H, L
	PSW
	AX
	RP0, RP1, RP2, RP3, BC, DE, HL
	SP
其它	DGS, DGL, TOL_INF, SJIS, EUC, NONE

2.4 标记生成规则

当在汇编指令中使用控制语句时，结构化汇编器为分支指令产生标记。

由结构化汇编器产生的标记的格式为“?Ldddd”。

“dddd”代表 1 或大于 1 的一个十进制数值，输出时抑制 0 和左边的队列。因此使用“?Ldddd”格式时不再键入任何标记。

2.5 空间定义

空间定义可以改变在赋值表达式、条件表达式或 switch 语句的 case 记号的左侧或右侧键入符号的数据大小。

[编码格式]

(Δsize_specification_characterΔ)

[功能]

<1> 如果空间字符为“B”或“b”，数据大小则变为字节。

[说明]

<1> 如果空间定义的字符不正确，将出现错误。

<2> 如果空间定义用在不支持空间定义的赋值表达式或条件表达式中，将出现错误。

<3> 如果空间定义用于寄存器，则只能使用同一空间定义来规定编码。数据大小不能改变。如果数据大小不同，则出现错误。

<4>当指定用户符号时，需确定为指定的数据空间改变了数据的大小。

<5>如果空间定义已在直接访问说明符或者间接访问说明符或直接数据上输入，则将忽略该空间定义，且数据大小不会改变。

<6> 字访问不能空间定义中指定。

2.6　数据空间

结构化汇编器检查符号的数据空间。这是因为符号根据所产生的指令不同而不同。但是，结构化汇编器允许汇编程序确定符号定义和常量是否被正确输入。

结构化汇编检查的数据空间如下表所示。

表 2-6. 数据空间

a	CY
b	位符号（[HL].*β*除外） 结构化汇编器识别以“*α*. *β*”格式输入的比特 sfr 和符号为位符号。 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，常量，A 和 PSW。 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	指定字节的用户符号、重叠了 saddrp 的 sfr
f	A
g	字节寄存器（A, R0, R1 除外）
h	R0
i	R1
j	sfr
k	PSW
l	字用户符号
m	重叠了 saddrp 的 sfrp
n	AX
o	字寄存器（AX, RP0 除外）
p	RP0
q	sfrp
r	SP
s	直接访问说明符 这些是采用“!addr”格式说明的符号。 字节用户符号、字用户符号、常量和“$”都可以作为“addr”输入。
t	间接访问说明符 这些是采用““[HL]””和[HL+byte]”格式说明的符号。 字节用户符号、常量和“$”都可以作为“byte”输入。
u	特殊间接访问说明符 这些是采用“[DE]”格式说明的符号。
v	立即数 这些是用“#date”格式说明的符号 字节用户符号、字用户符号、常量和“$”都可以作为”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何位置输入。

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，在圆括号之间插入的两个分号不作为注释语句的起点。

在“**2.2 (1) 字符组”**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

结构化汇编器输出工具信息。

如果输入源文件包含的工具信息已被结构化编译器输出，则位于信息起始处的“$”字符被“;”代替。

输出位置在模块头的最后。可被模块头输入的语句类型只有汇编控制指令、注释语句和换行符。

[输出格式]

$TOL_INF 2FH, 第二个参数,第三个参数, 0FFFFH

2FH 表示这是由结构化汇编器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。

版本号或者作为一个十六进制数值输出。或者，如果该值没有经过修改，则输出已在启动时显示的十进制数字图像。

(举例)

版本号 1.00 → 100H

参数三表示预处理器的错误信息。

0H	正常结束
1H	致命错误，退出
2H	告警，退出
3H	致命错误和告警，退出

0FFFFH 表示和语言有关的信息，是预处理器的一个固定值。

2.9　结构化汇编器输入源文件的输出结果

结构化汇编器按照如下规则输出输入源文件

表 2-7. 结构化汇编器的输出

输入源程序文件	辅助源程序文件
结构化汇编控制语句 结构化汇编表达式语句	作为注释输出
结构化汇编指示	无输出
#INCLUDE	输出 include 内容
#IFDEF 设置的源别名	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

本章举例说明如何描述控制语句的功能。

3.1　控制语句概述

控制语句用于在结构上用代码控制程序的流向。
有如下控制语句。

- 条件分支（IF~THEN~ELSE）
  (1) if~elseif~else~endif
  (2) if_bit~elseif_bit~else~endif
  (3) switch~case~default~ends

- 条件循环（DO~WHILE）
  (4) for~next　　(循环加)
  (5) while~endw　　(执行前进行循环条件判断)
  (6) while_bit~endw　(执行前进行循环条件判断)
  (7) repeat~until　　(执行后进行循环条件判断)
  (8) repeat~until_bit　(执行后进行循环条件判断)
  (9) break　　(循环中断)
  (10) continue　　(继续循环)
  (11) goto　　(退出执行异常处理)

3.2　控制语句字符

控制语句所产生指令差异基本上是根据控制语句使用大写字母或小写字母。例如，在“if~endif”和“IF~ENDIF”之间不同的语句大小排除了通过由处理条件表达式而生成的条件分支指令产生的直接分支。

然而，确保语句总是能被正确分支却降低了程序整体的效率。

为了解决这个问题，用户可以设置大小写以提高整体的效率等级。如不需要提高整体的效率等级，则只要编码使用大写字母，则用户可不改变字符的大小。

由于控制语句产生条件分支指令，需要确定相关地址是否在 128 字节之内。

在控制语句中，“if” 和 “elseif”都是保留字。结构化汇编器决定在控制语句保留字中的首字符是大写或小写字母。

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。

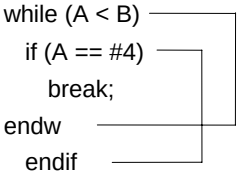
如果用大写输入 …分支采用条件分支指令和 BR 指示的组合。
如果用小写输入 …直接使用条件指示分支。

成对的控制语句（如“if, else,endif”）可混合大小写字母。也即，可以写成“IF~else~ENDIF”。

3.3 嵌套

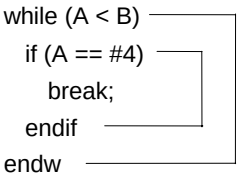
控制语句可以嵌套。通常允许最多嵌套 31 级。但控制语句不能交叉。

(错误编码举例)



由于交叉引起的错误。

(正确编码举例)



IF 语句被正确嵌套在 WHILE 语句中。

3.4 寄存器指定

[编码格式]

([Δ] [=] [Δ] 寄存器名 [Δ])

[功能]

<1> 如果寄存器是在比较表达式后被立即指定

当比较指令将左边的数值传递到指定寄存器后，将产生一个比较表达式，把指定寄存器与右边的数进行比较。

(示例)

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

<2> 如果寄存器在控制语句后被指定

在生成每个比较表达式期间，在生成了将左边的数值移到指定寄存器的指令之后，产生一个比较表达式，把指定寄存器与右边的数进行比较。

(示例)

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

<3> 如果(a) 和 (b)同时被指定

紧跟在比较表达式之后的寄存器指定优先。当产生了将左边的数值移到指定寄存器的指令后，生成一个比较表达式，把指定寄存器与右边的数进行比较。

对于在比较表达式之后没有寄存器指定的表达式，在根据控制语句之后的寄存器指定产生了将左边的数值移到指定寄存器的指令后，生成一个比较表达式，把指定寄存器与右边的数进行比较。

(示例)

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

[声明]

<1> 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，在控制语句中指定的任何寄存器都将被忽略。

<2> 关于寄存器名的列表，参见**表 2-5. 保留字符号。**也可输入 Sfr 指定。

<3> for 语句中处理赋值语句的过程与处理比较表达式的过程相同。

3.5 控制语句功能

下面的内容描述了各种控制语句的功能。

所使用的例子作为注释语句，说明了把生成的指令都输入到其中的源文件。

条件分支 **if**

(1) if~elseif~else~endif

[编码格式]

```
[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]

<1> if~endif

如果条件表达式 1 为真，将执行 if 程序块。

If 程序块可以是多行代码。

<2> if~else~endif

如果条件表达式 1 为真，则执行 if 部分，如果条件表达式 1 为假，则执行 else 部分。

If 及 else 部分可以是多行代码。

<3> if~elseif~else~endif

可以为单个 if 语句编写多个 elseif 语句。

如果条件表达式 1 为真，则执行 if 部分。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 块。如果为假，则判断下一个 endif 之前的其它任何 elseif 条件。如果没有 elseif 语句，则执行 else 块。

If 程序块、elseif 程序块以及 else 程序块都可以是多行代码。

条件分支 **if**

[声明]

<1> 比较表达式、逻辑表达式及位检验表达式都可以在条件表达式中输入。如果指定了寄存器名，则判断条件时使用该指定寄存器。

关于比较表达式和逻辑表达式的更多细节，参见“**3.6 条件表达式**”。

<2> 当一个条件产生两个分支时，使用 if-else-elseif。

<3> 当某个范围的值产生几个分支时，使用 if~elseif~else~endif。该语句与 switch 语句不同，后者包含一组值。

<4> 可以省略 elseif 语句和 else 语句，也可以输入多个 elseif 语句。

[生成指令]

<1> 处理 If（条件表达式）

(a) 生成一个指令以判断条件表达式中的条件。

(b) 如果条件不满足，则产生一个分支指令分别执行 elsiif 语句或 else 语句。

<2> 处理 elseIf（条件表达式）

(a) 为一个 endif 语句生成一个分支指令。

(b) 为一个由 if 语句产生的分支指令产生一个标记

(c) 产生一个指令判断条件表达式的条件。

(d) 如果条件不满足，产生一个分支指令分别执行 elsiif 块或 else 块。

<3> 处理 else

(a) 为 endif 语句生成一个分支指令。

(b) 为一个由 if 语句或 elseif 语句产生的分支指令产生一个标记

<4> 处理 endif

(a) 为由 if 语句、elseif 语句或 else 语句产生的分支指令产生一个标记

<5> 补充说明

(a) 这些程序块可以在内存中用 elseif_bit 进行混合。

条件分支	if

[用例]

<1> 当采用小写字母输入时

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

<2> 当采用大写字母输入时

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

条件分支 **if_bit**

(2) if_bit~elseif_bit~else~endif

[编码格式]

```
[Δ] if_bit [Δ](条件表达式 1) [Δ][(寄存器名)]
        if_bit 块
[Δ] elseif_bit [Δ](条件表达式 2) [Δ][(寄存器名)]
        elseif_bit 块
[Δ] else [Δ]
        else 块
[Δ] endif [Δ]
```

[功能]

<1> if_bit~endif

若条件表达式 1 为真，则执行 if_bit 程序块。

if_bit 程序块可由多行组成。

<2> if_bit~else~endif

若条件表达式 1 为真，则执行 if_bit 部分。否则，执行 else 部分。

if_bit 程序块和 else 程序块可由多行组成。

<3> if_bit~elseif_bit~else~endif

若条件表达式 1 为真，则执行 if_bit 程序块，如果为假则判断条件表达式 2。若条件表达式 2 为真，则执行 elseif_bit 程序块，否则，判断下一个 endif 之前的任何 elseif 条件。

若无 elseif_bit，则执行 else 程序块。

if_bit 程序块、elseif_bit 程序块和 else 程序块可由多行组成。

<4> 补充说明

这些程序块可以在内存中用 elseif 进行混合。

[声明]

<1> 位检验表达式作为条件表达式 1 和 2 输入。

关于位检验表达式的更多细节，参见“**3.6 条件表达式”**。

<2> 当一个条件出现两个分支时使用 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。

<3> 可忽略 elseif_bit 语句和 else 语句，并可键入多个 elseif_bit 语句。

条件分支 **if_bit**

[生成指令]

<1> 处理 if_bit（位条件）

(a) 为位条件生成一个真/假指令

<2> 处理 elseif_bit（位条件）

(a) 为一个 endif 语句生成一个分支指令

(b) 为由 if_bit 语句生成的分支指令构造一个标记

(c) 为一个位条件生成一个真/假指令

<3> 处理 else

(a) 为 endif 语句生成一个分支指令

(b) 为由 if_bit 语句或 elseif_bit 语句生成的分支指令产生一个标记

<4> 处理 endif

(a) 为由 if_bit 语句、elseif_bit 语句或 else 语句生成的分支指令产生一个标记

条件分支 **if_bit**

[用例]

<1> 当采用小写字母输入时

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

<2> 当采用大写字母输入时

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

条件分支 **switch**

(3) switch~case~default~ends

[编码格式]

```
[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]

<1>若 case 符号的值与 case 常量匹配，则执行指定的语句。

<2>若 case 符号的值与 case 常量不匹配但输入了缺省语句，则执行缺省的语句。

<3>一般地，必须输入一个 break 语句以跳出 switch 程序块。

[声明]

<1>对“for 符号”可能的指定取决于目标设备的汇编语言。

<2>若没有输入 break 语句，则对下一个 case 语句执行比较指令。注意，case 处理之后的操作有别于 C 语言程序中的操作。但是输入一个分支指令以建立一个功能与 C 程序类似。

<3>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，由于结构化汇编器把常量识别为字符串，因此，谨慎单独使用常量，汇编器可能把它当作字符串。

<4>只有当设置了寄存器指定， case 符号才会被转移至指定寄存器。

条件分支 **switch**

[生成指令]

<1> 处理 switch 语句

(a)如果没有指定寄存器，则判断 case 符号。如有必要，则生成到 A 或 AX 的转移指令。

(b)如果指定了寄存器，则 case 符号转移到指定寄存器。

但是，如果不能生成比较指令，则将出错。

更多细节，参见**表 3-1. switch 语句的构造指令**

<2> 处理 case 语句

(a) 由处理其他 case 语句的分支中生成标记。

(b) 生成 CMP 或 CMPW。如果没有匹配的指定常量，则生成一个分支指令至另一个 case 语句、default 语句或 ends 语句。

?LTRUE ：分支目的点标记，当指定常量匹配时

?LFALSE : 分支目的点标记，当指定常量不匹配时

- 若 case 语句用小写字母表示，且 switch 语句中没有寄存器指定

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

- 若 case 语句用小写字母表示，且 switch 语句中有寄存器指定

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

- 若 case 语句用大写字母表示，且 switch 语句中没有寄存器指定

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

- 若 case 语句用大写字母表示，且 switch 语句中有寄存器指定

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

<3> 处理 default 语句

(a) 从 case 语句中为分支指令生成一个标记

<4> 处理 ends 语句

(a) 从 case 语句或 break 语句中为分支指令生成一个标记

条件分支 **switch**

表 3-1 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**： 产生 MOV 指令
***2**： 产生 MOVW 指令
***3**： 不产生转移指令
空列表示错误。

条件分支	switch

[用例]

<1> 当采用小写字母输入时

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

<2> 当采用大写字母输入时

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

条件循环 **for**

(4) for~next

[编码格式]

```
[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，只要表达式 2 中的条件表达式成立，则执行指令组和表达式 3。一般地，表达式 3 执行递增或递减操作。

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

[声明]

<1> 需注意，类似于上文中的示例不适用于生成指令。

<2>在表达式 1、表达式 2 和表达式 3 中输入下述内容。

- 表达式 1 …设置初始值（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

<3>表达式 1 或表达式 3 中可输入赋值操作符和交换语句。但是如果这样做，应当检查转换输出，必要时还需进行修改。

<4>表达式 1、表达式 2 或表达式 3 有可能被省略。但如果表达式 2 被省略，程序将陷入死循环。

<5>条件表达式中可输入“forever”。

<6>由于表达式 2 和表达式 3 控制 for~next，这些表达式的内容不应被可执行语句改变。改变这些内容将导错误操作。

条件循环 **for**

[生成指令]

<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，则该指定寄存器用于赋值和比较。

(b) 为判断表达式 2 的条件的语句生成一个分支指令。

(c) 为由 next 语句生成的分支指令产生一个标记

(d) 为在（2）中生成的分支指令产生一个标记

(e) 为表达式 2 生成一个条件检验指令

<2> 处理 next 语句

(a) 为通过 for 语句处理（3）生成的标记产生一个分支指令

(b) 为跳过 for 程序块的分支指令产生一个标记

(c) 为表达式 3 的赋值表达式产生一个指令

<3> 补充说明

(a) 为了更有效地使用 for~next 语句， 推荐如下方法。

1. 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。
2. 如指定一个寄存器， 则指定 A 或 AX
3. 当执行至少 256 次循环时，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 中的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。

条件循环 **for**

[用例]

<1> **当采用小写字母输入时**

```
            MOV     i,#0H       ;for(i=#0H;i<#0FFH;i++)
    ?L1:
            CMP     i,#0FFH
            BNC     $?L2
            CALL    !XXX        ; CALL    !XXX
            INC     i
            BR      ?L1
    ?L2:                        ;next
```

<2> **当采用大写字母输入时**

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

条件循环 **while**

(5) while~endw

[编码格式]

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

[功能]

<1> 只要条件表达式为真， 则重复执行指令组。

[声明]

<1> 比较表达式、逻辑表达式、位检验表达式和“forever”都可以作为条件表达式而输入。
　　如果输入“forever”，将陷入无限循环。
<2>作为寄存器名称，指定在比较表达式或逻辑表达式中使用的寄存器为“（条件表达式）”。
<3>由于在执行指令组之前先判断条件表达式，如果第一个条件表达式为假，则指令组一次也不会被执行。

[生成指令]

<1> 处理 while（条件表达式）语句
　　(a) 为由 endw 生成的分支指令产生一个标记
　　(b) 生成一个条件检验指令。如果指定了寄存器名，则在生成条件测试指令时使用该指定寄存器。
　　(c) 生成一个分支指令，当条件检验失败时，将 while（条件表达式）语句从 while 程序块中删除

<2> endw
　　(a) 为执行循环生成一个分支指令
　　(b) 为分支指令生成一个标记，该指令用于从 while 程序块中删除 endw。

条件循环	while

[用例]

<1> 当采用小写字母输入时

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

<2> 当采用大写字母输入时

```
?L3:                                    ;WHILE(AX<#0FFFH)
        CMPW    AX,#0FFFH
        BC      $?L4
        BR      ?L5
?L4:
        MOV     B,#0FH                  ; B=#0FH
        INCW    HL                      ; HL++
        BR      ?L3
?L5:                                    ;ENDW
```

条件循环 **while_bit**

(6) while_bit~endw

[编码格式]

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

[功能]

<1> 只要位条件成立，则执行指令组

[声明]

<1> 由于在执行指令组之前先判断位条件，如果第一个位条件为假，则一次也不执行指令组。

[生成指令]

<1> 处理 while_bit（位条件）语句

(a) 为由 endw 创建的分支指令生成一个标记。

(b) 生成一条指令来检验位条件为真或假。

(c) 当位条件检验失败时，生成一个分支指令将 while_bit 语句从 while_bit~endw 程序块中删除。

<2> 处理 endw

(a)为一个执行循环生成一个分支指令。

(b) 生成一个分支指令用于将 endw 从 while_bit 程序块中删除。

条件循环 **while_bit**

[用例]

<1> **当采用小写字母输入时**

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

<2> **当采用大写字母输入时**

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

条件循环	until

(7) repeat~until

[编码格式]

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

[功能]

<1> 只要条件表达式为真，则重复执行指令组。

[声明]

<1> 比较表达式、逻辑表达式、位检验表达式和“forever”都可以作为条件表达式。
　　如果输入“forever”，结果将陷入无限循环。
<2> 做为寄存器名称，指定在比较表达式或逻辑表达式中使用的寄存器为“（条件表达式）”
<3> 由于在执行指令组之前先判断条件表达式，如果第一个条件表达式为假，则一次也不执行指令组。

[生成指令]

<1> 处理 repeat 语句
　　(a) 为由 until 创建的分支指令生成一个标记

<2> 处理 until（条件表达式）语句
　　(a) 为条件表达式生成一个条件检验指令
　　(b) 为由 repeat 操作生成的标记产生一个分支指令，从而可在执行 repeat~until 期间和条件表达式检验为假时执行该指令组。如果条件表达式检验为真，until 语句将从 repeat 程序块中删除。

条件循环 **until**

[用例]

<1> **当采用小写字母输入时**

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

<2> **当采用大写字母输入时**

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

条件循环 **until_bit**

(8) until_bit

[编码格式]

[Δ] repeat
　　指令组
[Δ] until_bit [Δ] (位检验表达式)

[功能]

<1> 只要位条件为假，则重复执行指令组。

[声明]

<1> 执行指令组后再检验位条件。因此，如果第一个位条件为真，指令组仅执行一次。

[生成指令]

<1> 处理 repeat

(a) 为由 until_bit 创建的分支指令生成一个标记

<2> 处理 until_bit（位条件）

(a) 为由 repeat 操作生成的标记生成一个分支指令，从而可在条件表达式检验为假时执行 repeat 和 until~bit 之间的指令组。如果条件表达式检验为真，until_bit 语句将从 repeat 程序块中删除。

[用例]

<1> 当采用小写字母输入时

```
?L1:                                    ;repeat
            MOV     B,#8H               ; B=#8H
            CALL    !XXX                ; CALL    !XXX
            BF      TRFG.0,$?L1         ;until_bit(TRFG.0)
```

<2> 当采用大写字母输入时

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

条件循环 **break**

(9) break

[编码格式]

[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。

[声明]

如果输入的语句不是 while, while_bit、 repeat~until、repeat~until_bit、for 或 switch 语句，则出现错误。

[生成指令]

生成一个无条件分支指令跳出 while, repeat, for, 或 switch 程序块

```
BR      ?Lxxxx
```

[用例]

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

条件循环	continue

(10) Continue

[编码格式]

[Δ] continue

[功能]

跟随在 while, while_bit, repeat~until, repeat~until_bit 或 for 语句中最内层嵌套程序块的 continue 之后的跳出操作，并且在条件检验之前设置一个无条件分支。

[声明]

<1>用于从程序块的中间跳出后续操作并执行下一个循环

<2>如果输入的语句不是 while, while_bit、 repeat~until、repeat~until_bit、for 或 switch 语句，则出现错误。

[生成指令]

为标记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

[用例]

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

条件循环 **goto**

(11) goto

[编码格式]

[Δ] goto Δ label

[功能]

无条件转向一个标记

[声明]

<1> 当需要立即进行错误处理时输入 goto 语句，比如在错误处理程序中，或当需要在多处一起处理错误时。

<2> 在汇编语言标记列中列出的符号都可指定为符号名。

[生成指令]

<1> 生成下列指令

```
BR   Label
```

<2> goto 语句的标记不是由结构化汇编器自动生成的。同样需要注意的是，汇编器并不自动检查分支目的标记是否存在。

[用例]

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

3.6 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 …… 比较第一项和第二项的值，并检验其为真或假
•位检验表达式…… 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

表 3-2. 比较表达式

比较表达式		编码格式	功能
(1)	等于	α == β	当α = β时为真，$\alpha \neq \beta$时为假
(2)	不等于	α != β	当$\alpha \neq \beta$时为真，α = β时为假
(3)	小于	α < β	当α < β时为真，α >= β时为假
(4)	大于	α > β	当α > β时为真，α <= β时为假
(5)	大于等于	α >= β	当α >= β时为真，α < β时为假
(6)	小于等于	α <= β	当α <= β时为真，α > β时为假

表 3-3. 位检验表达式

位检验表达式		编码格式	功能
(7)	正逻辑（位）	位符号	当指定的比特值为 1 时，则为真
(8)	负逻辑（位）	!位符号	当指定的比特值为 0 时，则为真

表 3-4. 逻辑操作

逻辑操作		编码格式	功能
(9)	逻辑与	条件表达式 1 && 条件表达式 2	若条件表达式 1 和条件表达式 2 都为真，则为真
(10)	逻辑或	条件表达式 1 || 条件表达式 2	若条件表达式 1 或条件表达式 2 为真，则为真

如果(γ)在比较结束时指定，则可在不能直接比较的α 和 β之间进行比较。
γ说明用于比较的寄存器。

3.6.1 比较表达式

在每个比较表达式的描述中，“?LTRUE”表示比较检验为真时的分支目的标记，“?LFALSE”表示当检验为假时的分支目的标记。

关于寄存器指定代码格式的说明，参见“**3.4 寄存器说明**”。

结构化汇编器并不检查在比较表达式左侧或右侧输入的符号作为汇编语言操作数是否正确。但是，数据大小测试按照“2.6 数据大小”中的描述执行，以确定是否可以生成一条指令。而且，当指定一个寄存器时，要检验使用指定寄存器生成指令的可能性。

当检验结果出现错误时输出错误信息。

更多细节参见相关生成指令。

各种比较表达式描述如下。

表 3-5. 比较指令的生成指令

			β																					
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
αn	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																						
	n	AX																						***2**
	o	字寄存器																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接访问符号																						
	t	间接访问号																						
	u	[DE]																						
	v	立即符号																						

***1**： 生成 CMP 指令

***2**： 生成 CMPW 指令

空列表示错误。

比较表达式 **等于 (==)**

(1) 等于(==)

[编码格式]

```
[Δ] [空间定义] α [Δ] == [Δ] [空间定义] [Δ] β [Δ] [(寄存器指定)]
```

[功能]

<1> 当无寄存器指定时

若α与β的值相等，结果为真。不等时结果为假。

<2> 当有寄存器指定时

α的值被传递给指定寄存器。当指定寄存器的值与β的值相同时结果为真，不等时结果为假。

[声明]

<1> 当无寄存器指定时

对于α和β，需确定指定的值可输入CMP和CMPW。

<2> 当有寄存器指定时

对于α，需保证指定的值可输入MOV和MOVW。

对于β，需保证指定的值可输入CMP和CMPW

[生成指令]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且有寄存器指定时

```
MOV(W)      指定寄存器, α
CMP(W)      指定寄存器, β
BNZ         $?LFALSE
```

比较表达式	等于 (==)

<3> 若控制语句采用大写字母输入，且无寄存器指定时

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

<4> 若控制语句采用大写字母输入，且有寄存器指定时

```
    MOV(W)      指定寄存器, α
    CMP(W)      指定寄存器, β
    BZ          $?LTRUE
    BR          ?LFALSE
?LTRUE:
```

有关α和β的组合的更多细节，参见**表3-5. 比较指令的生成指令。**α表示指定寄存器。关于MOV生成指令的更多描述，参见“**第四章（1）赋值**”。

[用例]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且有寄存器指定时

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

比较表达式	等于 (==)

<3> 若控制语句采用大写字母输入，且无寄存器指定时

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

<4> 若控制语句采用大写字母输入，且有寄存器指定时

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

比较表达式	不等于(!=)

(2) NotEqual (!=)

[编码格式]

```
[Δ] [空间定义] [Δ] α [Δ] != [Δ] [空间定义] [Δ] β [Δ] [(寄存器指定)]
```

[功能]

<1> 无寄存器指定时

若α与β的值不等，结果为真。相等时结果为假。

<2> 有寄存器指定时

α的值被传递给指定寄存器。当指定寄存器的值与β的值不等时结果为真，相等时结果为假。

[声明]

<1> 无寄存器指定时

对于α和β，需保证指定的值可输入CMP和CMPW。

<2> 有寄存器指定时

对于α，需保证指定的值可输入MOV和MOVW。

对于β，需保证指定的值可输入CMP和CMPW

[生成指令]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且有寄存器指定时

```
MOV(W)      指定寄存器, α
CMP(W)      指定寄存器, β
BZ          $?LFALSE
```

比较表达式	不等于(!=)

<3> 若控制语句采用大写字母输入，且无寄存器指定时

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

<4> 若控制语句采用大写字母输入，且有寄存器指定时

```
    MOV(W)    指定寄存器, α
    CMP(W)    指定寄存器, β
    BNZ       $?LTRUE
    BR        ?LFALSE
?LTRUE:
```

有关α和β的组合的更多细节，参见**表3-5. 比较指令的生成指令。**α表示指定寄存器。关于MOV生成指令的更多描述，参见“**第四章（1）赋值**”。

[用例]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且有寄存器指定时

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

比较表达式	不等于(!=)

<3> 若控制语句采用大写字母输入，且无寄存器指定时

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

<4> 若控制语句采用大写字母输入，且有寄存器指定时

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

比较表达式 小于 (<)

(3) 小于(<)

[编码格式]

[Δ] [空间定义] [Δ] α [Δ] < [Δ] [空间定义] [Δ] β [Δ] [(寄存器指定)]

[功能]

<1> 无寄存器指定时

若α的值小于β的值，结果为真。等于或大于时结果都为假。

<2> 有寄存器指定时

α的值被传递给指定寄存器。当指定寄存器的值小于β的值时结果为真，否则结果为假。

[声明]

<1> 无寄存器指定时

对于α和β，需保证指定的值可输入CMP和CMPW。

<2> 有寄存器指定时

对于α，需保证指定的值可输入MOV和MOVW。

对于β，需保证指定的值可输入CMP和CMPW。

[生成指令]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且有寄存器指定时

```
MOV(W)      指定寄存器, α
CMP(W)      指定寄存器, β
BNC         $?LFALSE
```

<3> 若控制语句采用大写字母输入，且无寄存器指定时

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

比较表达式	小于(<)

<4> 若控制语句采用大写字母输入，且有寄存器指定时

```
    MOV(W)      指定寄存器, α
    CMP(W)      指定寄存器, β
    BC          $?LTRUE
    BR          ?LFALSE
?LTRUE:
```

有关α和β的组合的详细资料参见**表3-5比较指令的生成指令。**α表示指定寄存器。MOV生成指令的更多描述参见**第4章（1）赋值。**

[用例]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且有寄存器指定时

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

<3> 若控制语句采用大写字母输入，且无寄存器指定时

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

比较表达式	小于(<)

<4> 若控制语句采用大写字母输入，且有寄存器指定时

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

比较表达式	大于(>)

(4) 大于 (>)

[编码格式]

```
[Δ] [空间定义] [Δ] α [Δ] > [Δ] [空间定义] [Δ] β [Δ] [(寄存器指定)]
```

[功能]

<1> 无寄存器指定时

若α的值大于β的值，结果为真。等于或小于时结果都为假。

<2> 有寄存器指定时

α的值被传递给指定寄存器。当指定寄存器的值大于β的值时结果为真，否则结果为假。

[声明]

<1> 无寄存器指定时

对于α和β，需保证指定的值可输入CMP和CMPW。

<2> 有寄存器指定时

对于α，需保证指定的值可输入MOV和MOVW。

对于β，需保证指定的值可输入CMP和CMPW

[生成指令]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且有寄存器指定时

```
MOV(W)      指定寄存器, α
CMP(W)      指定寄存器, β
BZ          $?LFALSE
BC          $?LFALSE
```

比较表达式	大于(>)

<3> 若控制语句采用大写字母输入，且无寄存器指定时

```
    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，且有寄存器指定时

```
    MOV(W)      指定寄存器, α
    CMP(W)      指定寄存器, β
    BZ      $$+4
    BNC         $?LTRUE
    BR          ?LFALSE
?LTRUE:
```

有关α和β的组合的更多细节，参见**表3-5. 比较指令的生成指令。**α表示指定寄存器。关于MOV生成指令的更多描述，参见“**第4章（1）赋值**”。

[用例]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且有寄存器指定时

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

比较表达式	大于(>)

<3> 若控制语句采用大写字母输入，且无寄存器指定时

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

<4> 若控制语句采用大写字母输入，且有寄存器指定时

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

比较表达式	大于等于(>=)

(5) 大于等于(>=)

[编码格式]

```
[Δ] [空间定义] [Δ] α [Δ] >= [Δ] [空间定义] [Δ] β [Δ] [(寄存器指定)]
```

[功能]

<1> 无寄存器指定时

若α的值大于或等于β的值，结果为真。小于β的值时结果为假。

<2> 有寄存器指定时

α的值被传递给指定寄存器。当指定寄存器的值大于或等于β的值时结果为真，小于时结果为假

[声明]

<1> 无寄存器指定时

对于α和β，需保证指定的值可输入CMP和CMPW。

<2> 有寄存器指定时

对于α，需保证指定的值可输入MOV和MOVW。

对于β，需保证指定的值可输入CMP和CMPW

[生成指令]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且有寄存器指定时

```
MOV(W)      指定寄存器, α
CMP(W)      指定寄存器, β
BC          $?LFALSE
```

<3> 若控制语句采用大写字母输入，且无寄存器指定时

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

比较表达式	大于等于(>=)

<4> 若控制语句采用大写字母输入，且有寄存器指定时

```
    MOV(W)      指定寄存器, α
    CMP(W)      指定寄存器, β
    BNC         $?LTRUE
    BR          ?LFALSE
?LTRUE:
```

有关α和β的组合的更多字节，参见**表3-5. 比较指令的生成指令。**α表示指定寄存器。关于MOV生成指令的更多描述，参见“**第4章（1）赋值**”。

[用例]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且有寄存器指定时

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

<3> 若控制语句采用大写字母输入，且无寄存器指定时

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

比较表达式	大于等于(>=)

<4> 若控制语句采用大写字母输入，且有寄存器指定时

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

比较表达式	小于等于(<=)

(6) 小于等于(<=)

[编码格式]

```
[Δ] [空间定义] [Δ] α [Δ] <= [Δ] [空间定义] [Δ] β [Δ] [(寄存器指定)]
```

[功能]

<1> 无寄存器指定时

若α的值小于或等于β的值，结果为真。大于β的值时结果为假。

<2> 有寄存器指定时

α的值被传递给指定寄存器。当指定寄存器的值小于或等于β的值时结果为真，大于时结果为假。

[功能]

<1> 无寄存器指定时

对于α和β，需保证指定的值可输入CMP和CMPW。

<2> 有寄存器指定时

对于α，需保证指定的值可输入MOV和MOVW。

对于β，需保证指定的值可输入CMP和CMPW

[生成指令]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且有寄存器指定时

```
MOV(W)      指定寄存器, α
CMP(W)      指定寄存器, β
BZ          $$+4
BNC         $?LFALSE
```

比较表达式	小于等于(<=)

<3> 若控制语句采用大写字母输入，且无寄存器指定时

```
    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，且有寄存器指定时

```
    MOV(W)    指定寄存器, α
    CMP(W)    指定寄存器, β
    BZ        $?LTRUE
    BC        $?LTRUE
    BR        ?LFALSE
?LTRUE:
```

有关α和β的组合的更多细节，参见**表3-5. 比较指令的生成指令。**α表示指定寄存器。关于MOV生成指令的更多描述，参见“**第4章（1）赋值**”。

[用例]

<1> 若控制语句采用小写字母输入，且无寄存器指定时

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

<2> 若控制语句采用小写字母输入，且无寄存器指定时

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

比较表达式	小于等于(<=)

<3> 若控制语句采用大写字母输入，且无寄存器指定时

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

<4> 若控制语句采用大写字母输入，且有寄存器指定时

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

(7) 无限循环(forever)

[编码格式]

```
[Δ] forever [Δ]
```

[功能]

设置loop语句为无限循环，不生成比较指令

[声明]

可写在条件表达式的循环语句中（for语句、while语句、until语句）

[用例]

<1> for语句

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

<2> while语句

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

比较表达式 无限循环(forever)

<3> repeat语句

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

3.6.2 位检验表达式

注意：在描述每个位检验表达式时，?LTRUE用作检验结果为真时的分支目的标记，?LFALSE用作检验结果为假时的分支目的标记。

结构化汇编器并不检验位检验表达式代码作为汇编语言操作数时输入是否正确。但是，如“**2.6 数据大小**”一节所述，会执行一个数据大小的检验。

另外，“Z”也作为位符号进行处理。

结构化汇编器不使用汇编器指令(EQU)去检查位符号是否已定义。但是，用户符号也可作为位符号进行处理。

当检验结果出现错误时输出错误信息。

更多细节参见特殊生成指令。

各种位检验表达式描述如下。

位检验表达式	正逻辑 (位)

(1) 位符号

[编码格式]

[Δ] 位符号 [Δ]

[功能]

当位符号的值为1时结果为真，当位符号的值为0时结果为假。

下列控制语句可以包含位符号作为条件表达式。

if	if_bit
elseif	elseif_bit
while	while_bit
until	until_bit

[生成指令]

<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:
```

位检验表达式	正逻辑 (位)

<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:
```

[用例]

<1> 当控制语句采用小写字母输入时

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

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

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

位检验表达式	正逻辑 (位)

<2> 当控制语句采用大写字母输入时

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

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

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

位检验表达式	负逻辑 (位)

(2) !位符号

[编码格式]

[Δ] !位符号 [Δ]

[功能]

当位符号的值为0时结果为真，当位符号的值为1时结果为假。

下列控制语句可包含位符号作为条件表达式。

if	if_bit
elseif	elseif_bit
while	while_bit
until	until_bit

[生成指令]

<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Z已输入时

```
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:
```

位检验表达式	负逻辑 (位)

<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:
```

[用例]

<1> 当控制语句采用小写字母输入时

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

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

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

位检验表达式	负逻辑 (位)

<2> 当控制语句采用大写字母输入时

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

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

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

3.6.3 逻辑操作

注意：在每个条件表达式的描述中，?LTRUE用作检验结果为真时的分支目的标记，?LFALSE用作检验结果为假时的分支目的标记

当有两个比较表达式或一个真/假位检验表达式存在时，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输入16个逻辑操作符。

这意味着，有可能处理的表达式是当两个条件表达式都满足后或其中之一满足时被执行的表达式。

结构化汇编器从最高优先级的逻辑操作符开始生成分支指令。

[代码示例]

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

下面描述逻辑操作。

逻辑操作 **逻辑与 (&&)**

(1) 逻辑与 (&&)

[编码格式]

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的操作。

输出指令的差异取决于控制语句是大写输入或小写输入。

首先为被圆括号“( )”括起来的内容产生检验指令。

[生成指令]

<1> 当控制语句采用小写字母输入时

表3-6. 逻辑与的生成指令（控制语句采用小写字母）

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

逻辑操作 **逻辑与(&&)**

<2> 当控制语句采用大写字母输入时

表3-7. 逻辑AND的生成指令（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:
Bit symbol &&	BT Bit symbol, $?LTRUE BR ?LFALSE ?LTRUE:
CY &&	BC $?LTRUE BR ?LFALSE ?LTRUE:
Z &&	BZ $?LTRUE BR ?LFALSE ?LTRUE:
!bit symbol &&	BF Bit symbol, $?LTRUE BR ?LFALSE ?LTRUE:
!CY &&	BNC $?LTRUE BR ?LFALSE ?LTRUE:
!Z &&	BNZ $?LTRUE BR ?LFALSE ?LTRUE:

逻辑操作	逻辑与(&&)

[用例]

<1> 当控制语句采用小写字母输入时

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

<2> 当控制语句采用大写字母输入时

```
            MOV     A,C             ;IF(C==#0 && B>=#0 && B<#80H)(A)
            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            ; CALL  !XXX
            BR      ?L7
?L6:                                ;ELSE
            CALL    !YYY            ; CALL  !YYY
?L7:                                ;ENDIF
```

逻辑操作	逻辑或 (||)

(2) 逻辑或 (||)

[编码格式]

```
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的逻辑或结果。当条件表达式1或条件表达式2为真时结果为真，两者都为假时结果为假。任一条件满足时则执行输入的操作。

首先为被圆括号“( )”括起来的内容产生检验指令。

[生成指令]

表3-8. 逻辑或的生成指令

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 位符号, $?LFALSE
CY ||	BC $?LFALSE
Z ||	BZ $?LFALSE
! 位符号||	BF 位符号, $?LFALSE
!CY ||	BNC $?LFALSE
!Z ||	BNZ $?LFALSE

逻辑操作	逻辑或(||)

[用例]

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

[备忘录]

第四章 表达式

表达式用来进行赋值或算术操作。
下面是表达式的一些例子。

- 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4-1. 赋值语句

赋值语句		编码格式	功能
(1)	赋值	$\alpha = \beta$	$\alpha \leftarrow \beta$
	赋值（带寄存器指定）	$\alpha = \beta\,(\gamma)$	$(\gamma) \leftarrow \beta$, $\alpha \leftarrow (\gamma)$
	序列赋值	$\alpha 1 = \cdots = \alpha n = \beta$	$\alpha 1 = \leftarrow \beta, \cdots, \alpha n \leftarrow \beta$
	序列赋值（带寄存器指定）	$\alpha 1 = \cdots = \alpha n = \beta\,(\gamma)$	$\gamma \leftarrow \beta$, $\alpha 1 \leftarrow \gamma, \cdots, \alpha n \leftarrow \gamma$
(2)	增量赋值	$\alpha \mathrel{+}= \beta$	$\alpha \leftarrow \alpha + \beta$
	增量赋值（带寄存器指定）	$\alpha \mathrel{+}= \beta$（寄存器）	$\gamma \leftarrow \alpha$, $\gamma \leftarrow \gamma + \beta$, $\alpha \leftarrow \gamma$
	增量赋值（带寄存器指定）	$\alpha \mathrel{+}= \beta$, CY	$\alpha \leftarrow \alpha + \beta$, CY
	增量赋值（带寄存器指定）	$\alpha \mathrel{+}= \beta$, CY (寄存器)	$\gamma \leftarrow \alpha$, $\gamma \leftarrow \gamma + \beta$, CY, $\alpha \leftarrow \gamma$
(3)	减量赋值	$\alpha \mathrel{-}= \beta$	$\alpha \leftarrow \alpha - \beta$
	减量赋值（带寄存器指定）	$\alpha \mathrel{-}= \beta$, (寄存器)	$\gamma \leftarrow \alpha$, $\gamma \leftarrow \gamma - \beta$, $\alpha \leftarrow \gamma$
	减量赋值（带寄存器指定）	$\alpha \mathrel{-}= \beta$, CY	$\alpha \leftarrow \alpha - \beta$, CY
	减量赋值（带寄存器指定）	$\alpha \mathrel{-}= \beta$, CY (寄存器)	$\gamma \leftarrow \alpha$, $\gamma \leftarrow \gamma - \beta$, CY, $\alpha \leftarrow \gamma$
(4)	逻辑与AND赋值	$\alpha \mathrel{\&}= \beta$	$\alpha \leftarrow \alpha \cap \beta$
	逻辑与AND赋值（带寄存器指定）	$\alpha \mathrel{\&}= \beta$(寄存器)	$\gamma \leftarrow \alpha$, $\gamma \leftarrow \gamma \cap \beta$, $\alpha \leftarrow \gamma$
(5)	逻辑或赋值	$\alpha \mid= \beta$	$\alpha \leftarrow \alpha \cup \beta$
	逻辑或OR赋值（带寄存器指定）	$\alpha \mid= \beta$(寄存器)	$\gamma \leftarrow \alpha$, $\gamma \leftarrow \gamma \cup \beta$, $\alpha \leftarrow \gamma$
(6)	逻辑异或XOR赋值	$\alpha \wedge= \beta$	$\alpha \leftarrow \alpha \wedge \beta$
	逻辑异或XOR赋值（带寄存器指定）	$\alpha \wedge= \beta$(寄存器)	$\gamma \leftarrow \alpha$, $\gamma \leftarrow \gamma \wedge \beta$, $\alpha \leftarrow \gamma$
(7)	右移（循环）赋值	$\alpha >>= \beta$	(α右移β位)
	右移赋值（带寄存器指定）	$\alpha >>= \beta$(寄存器)	$\gamma \leftarrow \alpha$, (γ右移 β 位), $\alpha \leftarrow \gamma$
(8)	左移赋值	$\alpha <<= \beta$	(α左移β位)
	左移赋值（带寄存器指定）	$\alpha <<= \beta$(寄存器)	$\gamma \leftarrow \alpha$, (γ 左移 β 位), $\alpha \leftarrow \gamma$

表4-2. 计数语句

计数语句		编码格式	功能
(9)	递增	α ++	$\alpha \leftarrow \alpha + 1$
(10)	递减	α ––	$\alpha \leftarrow \alpha - 1$

表4-3. 交换语句

交换语句		编码格式	功能
(11)	交换	α <-> β	$\alpha \leftarrow \alpha$ <-> β
	交换（带寄存器指定）	α <-> β (γ)	$\gamma \leftarrow \alpha$, $\gamma \leftarrow \gamma$ <-> β, $\alpha \leftarrow \gamma$

表4-4. 位操作语句

位操作语句		编码格式	功能
(12)	置位	α = 1	$\alpha \leftarrow 1$
	置位（带寄存器指定）	α = 1 (CY)	CY $\leftarrow$ 1 $\alpha \leftarrow$ 1
	序列置位	α 1 = ··· = α n = 1	α n = $\leftarrow$ 1, ···, α 1 $\leftarrow$ 1
	序列置位（带寄存器指定）	α 1 = ··· = α n = 1 (CY)	CY $\leftarrow$ 1, α n $\leftarrow$ 1, ···,α 1 $\leftarrow$ 1
(13)	复位	α = 0	$\alpha \leftarrow 0$
	复位（带寄存器指定）	α = 0 (CY)	CY $\leftarrow$ 0, $\alpha \leftarrow 0$
	序列复位	α 1 = ··· = α n = 0	α n $\leftarrow$ 0, ···, α 1 $\leftarrow$ 0
	序列复位（带寄存器指定）	α 1 = ··· = α n = 0 (CY)	CY $\leftarrow$ 0, α n $\leftarrow$ 0, ···, α 1 $\leftarrow$0

这些表达式的功能在下面进行介绍。
生成指令在用例中进行说明。输入源文件作为注释显示。

赋值语句 **赋值 (=)**

(1) 赋值 (=)

[编码格式]

```
[Δ] [空间定义] [Δ] α 1 [Δ] [= [Δ][空间定义] [Δ] α 2 [Δ] ...]
= [Δ][空间定义] [Δ] β [Δ][(寄存器指定)]
```

[功能]

<1> 无寄存器指定时

右侧的β值依次赋给左侧的变量。

<2> 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

[声明]

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，也不会有指令生成。

[生成指令]

<1> 当α和β是位符号时

- **当α 是CY时**

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

但是，不能进行连续赋值。

赋值语句 **Assign (=)**

- **当β是CY时**

```
    BNC     ?L1
    SET1    α n
    SET1    α n-1
      :
    SET1    α 2
    SET1    α 1
    BR      ?L2
?L1:
    CLR1    α n
    CLR1    α n-1
      :
    CLR1    α 2
    CLR1    α 1
?L2:
```

- **当CY被指定为寄存器时**

```
    BF      β, ?L1
    SET1    α n
    SET1    α n-1
      :
    SET1    α 2
    SET1    α 1
    BR      ?L2
?L1:
    CLR1    α n
    CLR1    α n-1
      :
    CLR1    α 2
    CLR1    α 1
?L2:
```

赋值语句	赋值 (=)

<2>　当*α*和*β*不是位符号时

- **当无寄存器指定时**

  MOV　　*α*1, *β*

  依据操作数，可替代生成MOVW。

- **当无寄存器指定且连续赋值时**

  MOV　　*α*n, *β*

  MOV　　*α*n-1, *β*

  　:

  MOV　　*α*2, *β*

  MOV　　*α*1, *β*

  依据操作数，可替代生成MOVW。

- **当有寄存器指定时**

  MOV　　指定寄存器, *α*

  MOV　　*α*1, 指定寄存器

  依据操作数，可替代生成MOVW。

- **当有寄存器指定且连续赋值时**

  MOV　　指定寄存器, *β*

  MOV　　*α*n, 指定寄存器

  MOV　　*α*n-1, 指定寄存器

  　　:

  MOV　　*α*2, 指定寄存器

  MOV　　*α*1, 指定寄存器

  依据操作数，可替代生成MOVW。

关于*α*和*β*组合的更多细节，参见**表4-5. 赋值生成指令。**根据输入语句的不同，α_n和 *β*表示指定寄存器。

赋值语句 **赋值 (=)**

[用例]

<1> 当无寄存器指定时

```
          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> 当有寄存器指定时

```
          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
```

赋值语句 **赋值 (=)**

表4-5. 赋值生成指令

			β																				
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
αn	a	CY		***3**		***4**								***3**										
	b	位符号	***3**			***5**																		
	c	[HL]. β	***3**			***5**																		
	d	字节用户符号	***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**				***2**
	o	字节寄存器														***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**:使用位分支指令时生成一个替代指令

***4**:当“1”作为β输入时生成SET1指令，当“0”作为β输入时生成CLR1指令。任何非0或非1的值被输入时生成MOV指令。

***5**: 当“1”作为β输入时生成SET1指令，当“0”作为β输入时生成CLR1指令。

***6**: 任何非0或非1的值作为α_n输入时，使用位检验表达式生成一个替代指令。

空单元格表示错误。

赋值语句	增量赋值(+=)

(2) 增量赋值(+=)

[代码格式]

```
[Δ] [空间定义] [Δ] α 1 [Δ] += [Δ][空间定义] [Δ] β [Δ]
                                              [,[Δ] CY] [Δ][(寄存器指定)]
```

[功能]

<1> 无寄存器指定时

α和β这两个操作数相加，结果赋给α。

<2> 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。

<3> 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。

<4> 带进位累加；有寄存器指定

α值赋给指定寄存器。

通过使用指定寄存器的值和β执行携带操作的增量运算，结果赋给指定寄存器。

指定寄存器的值赋给α。

[声明]

<1> 无寄存器指定时

*α*和*β*的值可输入ADD和ADDW。

<2> 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。

<3> 带进位累加；无寄存器指定

*α*和*β*的值可输入ADDC。

<4> 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。

赋值语句 **增量赋值 (+=)**

[生成指令]

<1> 无寄存器指定时

ADD α, β

依据操作数，可替代生成ADDW。

<2> 有寄存器指定时

MOV 指定寄存器, α

ADD 指定寄存器, β

MOV α,指定寄存器

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 指定寄存器, α

ADDC 指定寄存器, β

MOV α,指定寄存器

关于α和β组合的更多细节，参见**表4-6. 增量赋值的生成指令**。根据输入语句的不同，α表示指定寄存器。

赋值语句	增量赋值 (+=)

[用例]

<1> 无寄存器指定时

```
ADD         A, #0C0H      ;A+=#0C0H
ADDW        Ax, #0C00H    ;Ax+=#0C00H
```

<2> 有寄存器指定时

```
MOV         A, !ABC       ;!ABC+=#0FCH(A)
ADD         A, #0FCH
MOV         !ABC, A
MOVW        AX, HL        ;HL+=#0FFFH(AX)
ADDW        AX, #0FFFH
MOVW        HL, AX
```

<3> 带进位累加；无寄存器指定

```
ADDC        A, #50H       ;A+=#50H,CY
```

<4> 带进位累加；有寄存器指定

```
MOV         A, PSW        ;PSW+=#50H,CY(A)
ADDC        A, #50H
MOV         PSW, A
```

赋值语句 **增量赋值 (+=)**

表4-6. 增量赋值的生成指令

			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																						
	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																						
	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。对于对于带进位加，则生成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(–=)

[编码格式]

```
[Δ] [空间定义] [Δ] α 1 [Δ] –= [Δ][空间定义] [Δ] β [Δ]
                                        [,[Δ] CY] [Δ][(寄存器指定)]
```

[功能]

<1> 无寄存器指定时

从α中减去β，结果赋给α。

<2> 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。

将指定寄存器的值赋给α

<3> 带进位减；无寄存器指定时

使用α和β两个操作数执行带进位减法，并把结果赋给α。

<4> 带进位减；有寄存器指定时

α值赋给指定寄存器。

使用指定寄存器的值和β执行带进位减法，并把结果赋给指定寄存器。

将指定寄存器的值赋给α

[声明]

<1> 无寄存器指定时

α和β值可输入SUB 和SUBW。

<2> 有寄存器指定时

α值可输入MOV和MOVW。

β值可输入SUB 和SUBW。

<3> 带进位减；无寄存器指定时

α和β值可输入SUBC。

<4> 带进位减；有寄存器指定时

α值可输入MOV。

β 值可输入SUBC。

赋值语句 **减量赋值 (–=)**

[生成指令]

<1> 无寄存器指定时

生成如下指令

SUB　　α, β

依据操作数不同，可替代生成SUBW

<2> 有寄存器指定时

生成如下指令

MOV　　指定寄存器，α

SUB　　指定寄存器，β

MOV　　α，指定寄存器

依据操作数不同，可替代生成SUBW

<3> 带进位减；无寄存器指定时

生成如下指令。

SUBC　　α, β

<4> 带进位减；有寄存器指定时

生成如下指令。

MOV　　指定寄存器，α

SUBC　　指定寄存器，β

MOV　　α，指定寄存器

关于α,和β 组合的耕更多细节，参见**表4-7.　减量赋值的生成指令。**依据输入语句的不同，α表示该指定寄存器。

赋值语句 **减量赋值 (–=)**

[用例]

<1> 无寄存器指定时

```
SUB         A, #0C0H      ;A–=#0C0H
SUBW        AX, #0C00H    ;AX–=#0C00H
```

<2> 有寄存器指定时

```
MOV         A, !ABC       ;!ABC–=#0FCH(A)
SUB         A, #0FCH
MOV         !ABC, A
MOVW        AX, HL        ;HL–=#0FFFH(AX)
SUBW        AX, #0FFFH
MOVW        HL, AX
```

<3> 带进位减；无寄存器指定时

```
SUBC        A, #50H       ;A–=#50H,CY
```

<4> 带进位减；有寄存器指定时

```
MOV         A, PSW        ;PSW–=#50H,CY(A)
SUBC        A, #50H
MOV         PSW, A
```

赋值语句 **减量赋值 (–=)**

表4-7. 减量赋值的生成指令

			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																						
	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																						
	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。对于带进位减法，生成SUBC.

***2**: 生成SUBW 指令.

空单元格表示错误。

(4) 逻辑与（AND）赋值 (-=)

[编码格式]

[Δ] [空间定义] [Δ] α [Δ] &= [Δ] [空间定义] [Δ] β [Δ]
[寄存器指定]

[功能]

<1> 无寄存器指定时

对α和β的比特位执行逻辑与（α & β），结果赋给α。

<2> 有寄存器指定时

α赋给指定寄存器。

对指定寄存器和β的比特位执行逻辑与（α & β），结果赋给指定寄存器。

指定寄存器的值赋给α。

[声明]

<1> 无寄存器指定时

α 和 β的值可输入AND 和BF中。

<2> 有寄存器指定时

α 的值可输入MOV和BF中。

β的值可输入AND 和BF中。

[生成指令]

<1> 无寄存器指定时

- **当α是 CY**

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

- **当α不是 CY**

```
AND     α, β
```

<2> 有寄存器指定时

- **当指定寄存器是CY时**

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

- **当指定寄存器不是CY时**

```
MOV     指定寄存器, α
AND     指定寄存器, β
MOV     α,指定寄存器
```

关于α,和β组合的更多细节，参见**表4-8. 逻辑与赋值的生成指令。**

[用例]

<1> 无寄存器指定时

```
          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

```
          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
```

赋值语句 **逻辑与赋值 (&=)**

表4-8. 逻辑与（AND）赋值的生成指令

			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		***2**		***2**								***2**										
	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																						
	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**: 根据位分支指令生成SUBW 指令。

空单元格表示错误。

(5) 逻辑或（OR）赋值 (|=)

[编码格式]

[Δ] [空间定义] [Δ] α [Δ] |= [Δ] [空间定义] [Δ] β [Δ]
[寄存器指定]

[功能]

<1> 无寄存器指定时

对α和β的比特位执行逻辑OR（α | β），结果赋给α

<2> 有寄存器指定时

α赋给指定寄存器。

对指定寄存器和β的比特位执行逻辑或（α | β），结果赋给指定寄存器。

指定寄存器的值赋给α。

[声明]

<1> 无寄存器指定时

α 和 β的值可输入*OR* 和BF中。

<2> 有寄存器指定时

α 的值可输入MOV和BF中。

β的值可输入OR 和BF中。

[生成指令]

<1> 无寄存器指定时

- **当α是 CY**

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

- **当α不是 CY**

```
OR      α, β
```

赋值语句 **逻辑或赋值 (|=)**

<2> 有寄存器指定时

- **当指定寄存器是 C**Y

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

- **当指定寄存器不是 CY**

```
MOV             指定寄存器， α
OR              指定寄存器， β
 MOV            α，指定寄存器
```

关于α,和β组合的更多细节，参见**表4-9. 逻辑或赋值的生成指令。**

赋值语句 **逻辑或赋值 (|=)**

[用例]

<1> 无寄存器指定时

```
            BC      $?L1            ;CY|=P1S.1
            BF      P1S.1,$?L2
    ?L1:
            SET1    CY
            BR      ?L3
    ?L2:
            CLR1    CY
    ?L3:
            OR      A,#0FFH         ;A|=#0FFH
```

<2> 有寄存器指定时

```
            BT      A.1,$?L4        ;A.1|=PORT3.0(CY)
            BF      PORT3.0,$?L5
    ?L4:
            SET1    A.1
            BR      ?L6
    ?L5:
            CLR1    A.1
    ?L6:
            MOV     A,[DE]          ;[DE]|=#07H(A)
            OR      A,#07H
            MOV     [DE],A
```

赋值语句 **逻辑或赋值 (|=)**

表4-9. 逻辑或（OR）赋值的生成指令

			β																					
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
αn	a	CY		***2**		***2**								***2**										
	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																						
	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**: 成OR指令。

***2**: 根据位分支指令生成替代指令.

空单元格表示错误。

赋值语句 **逻辑异或赋值 (∧=)**

(6) 逻辑异或（XOR）赋值 (∧=)

[编码格式]

[Δ] [空间定义] [Δ] α [Δ] ∧= [Δ] [空间定义] [Δ] β [Δ] [寄存器指定]

[功能]

<1> 无寄存器指定时

对α和β的比特位执行逻辑异或（$\alpha \wedge \beta$），结果赋给α

<2> 有寄存器指定时

α赋给指定寄存器。

对指定寄存器和β的比特位执行逻辑异或（$\alpha \wedge \beta$），结果赋给指定寄存器。

指定寄存器的值赋给α。

[声明]

<1> 无寄存器指定时

α 和 β的值可输入*XOR* 和BF中。

<2> 有寄存器指定时

α 的值可输入MOV和BF中。

β的值可输入*X*OR 和BF中。

[生成指令]

<1> 无寄存器指定时

- **当α是 CY时**

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

- **当α不 是 CY时**

```
XOR     α, β
```

赋值语句	逻辑异或赋值 (^=)

<2>　有寄存器指定时

- **当*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:
```

- **当指定寄存器 不 是 CY时**

```
MOV             指定寄存器，α
XOR             指定寄存器，β
MOV             α，指定寄存器
```

关于α,和β组合的更多细节，参见**表4-10.　逻辑异或赋值的生成指令**

赋值语句	逻辑异或赋值 (^=)

[用例]

<1> 无寄存器指定时

```
            BNC     $?L1            ;CY^=P1S.1
            BF      P1S.1,$?L2
    ?L1:
            BC      $?L3
            BF      P1S.1,$?L3
    ?L2:
            SET1    CY
            BR      ?L4
    ?L3:
            CLR1    CY
    ?L4:
            XOR     A,#0FFH         ;A^=#0FFH
```

<2> 有寄存器指定时

```
            BF      A.1,$?L5        ;A.1^=PORT3.0(CY)
            BF      PORT3.0,$?L6
    ?L5:
            BT      A.1,$?L7
            BF      PORT3.0,$?L7
    ?L6:
            SET1    A.1
            BR      ?L8
    ?L7:
            CLR1    A.1
    ?L8:
            MOV     A,[DE]          ;[DE]^=#07H(A)
            XOR     A,#07H
            MOV     [DE],A
```

赋值语句 **逻辑异或赋值 (^=)**

表4-10. 逻辑异或（XOR）赋值的生成指令

			β																					
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
αn	a	CY		***2**		***2**								***2**										
	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																						
	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**: 根据位分支指令生成替代指令.

空单元格表示错误。

赋值语句	右移赋值 (>>=)

(7) 右移赋值(>>=)

[编码格式]

```
[Δ] [空间定义] [Δ] α [Δ] >>= [Δ] β [Δ] [寄存器定义]
```

[功能]

<1> 无寄存器指定时

α右移β位，结果赋给α

<2> 有寄存器指定时

α值赋给指定寄存器。

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。

[声明]

<1> 无寄存器指定时

α值只能输入A中。

β值可作为1到7的数字输入。

<2> 有寄存器指定时

α值可输入MOV中。

β值可作为1到7的数字输入。

指定寄存器只能输入A中。

[生成指令]

<1> 无寄存器指定时

当ROR指令输出β次时生成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
```

赋值语句	右移赋值(>>=)

[用例]

<1> 无寄存器指定时

```
ROR    A,1           ;A>>=4
ROR    A,1
ROR    A,1
ROR    A,1
AND    A,#0FFH SHR 4
```

<2> 有寄存器指定时

```
MOV    A,CCV         ;CCV>>=4(A)
ROR    A,1
ROR    A,1
ROR    A,1
ROR    A,1
AND    A,#0FFH SHR 4
MOV    CCV,A
```

(8) 左移赋值 (<<=)

[编码格式]

```
[Δ] [空间定义] [Δ] α [Δ] <<= [Δ] β [Δ] [寄存器指定]
```

[功能]

<1> 无寄存器指定时

α左移β位，结果赋给α

<2> 有寄存器指定时

α值赋给指定寄存器。

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*。

[声明]

<1> 无寄存器指定时

α值只能输入A中。

β值可作为1到7的数字输入。

<2> 有寄存器指定时

α值可输入MOV中。

β值可作为1到7的数字输入。

指定寄存器只能输入A中。

[生成指令]

<1> 无寄存器指定时

当ROR指令输出β次时生成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
```

赋值语句	左移赋值 (>>=)

[用例]

<1> 无寄存器指定时

```
ROL    A,1           ;A<<=4
ROL    A,1
ROL    A,1
ROL    A,1
AND    A,#LOW( 0FFH SHL 4 )
```

<2> 有寄存器指定时

```
MOV    A,CCV         ;CCV<<=4(A)
ROL    A,1
ROL    A,1
ROL    A,1
ROL    A,1
AND    A,#LOW( 0FFH SHL 4 )
MOV    CCV,A
```

计数语句	累加(++)

(9) 累加(++)

[编码格式]

```
[Δ] [空间定义] [Δ] α [Δ] ++
```

[功能]

α值加1

[声明]

α值可输入INC或INCW。

[生成指令]

INCWα

依据操作数可生成INCW。

更多细节参见**表4-11. 累加的生成指令**

[用例]

```
INC         H             ;H++
INC         CNT           ;CNT++
INCW        HL            ;HL++
```

计数语句 **累加(++)**

表4-11 累加的生成指令

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。

(10) 递减 (– –)

[编码格式]

```
[Δ] [空间定义] [Δ] α [Δ]– –
```

[功能]

α值减1

[声明]

α值可输入DEC或DECW。

[生成指令]

DEC α

生成DECW取决于操作数。

更多细节参见**表4-12. 递减的生成指令.**

[用例]

```
DEC         H               ;H– –
DEC         CNT             ;CNT– –
DECW        HL              ;HL– –
```

计数语句	递减(--)

表4-12. 递减的生成指令

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。

交换语句	交换 (<->)

(11) 交换 (<->)

[编码格式]

```
[Δ] [空间定义] [Δ] α [Δ] <-> [Δ][空间定义] [Δ] β [Δ]
                                                [(寄存器指定)]
```

[功能]

<1> 无寄存器指定时

对调*α和β*指向的值

<2> 有寄存器指定时

*α*值赋给指定寄存器。

指定寄存器的值与*β*指向的值对调。

指定寄存器的值赋给*α*。

[声明]

<1> 无寄存器指定时

*α和β*指向的值可输入XCH或XCHW。

<2> 有寄存器指定时

*α*指向的值可输入MOV和MOVW。

*β*指向的值可输入XCH或XCHW。

[生成指令]

<1> 无寄存器指定时

XCH *α, β*

依据操作数可生成XCHW。

<2> 有寄存器指定时

MOV	指定寄存器, *α*
XCH	指定寄存器,, *β*
MOV	*α*,指定寄存器,

依据操作数可生成XCHW 。

关于*α* 和 *β*组合的更多细节，参见**表4-13. 交换操作的生成指令。***α* 表示该指定寄存器。

交换语句	交换 (<->)

[用例]

<1> 无寄存器指定时

```
XCH         A,B         ;A<->B
XCHW        AX,BC       ;AX<->BC
```

<2> 有寄存器指定时

```
MOV         A,DATA      ;DATA<->B(A)
XCH         A,B
MOV         DATA,A
MOVW        AX,DE       ;DE<->BC(AX)
XCHW        AX,BC
MOVW        DE,AX
```

交换语句 **交换 (<->)**

表4-13. 交换操作的生成指令

			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																						
	e	字节数据																						
	f	A			***1**	***1**		***1**			***1**		***1**									***1**	***1**	
	g	字节寄存器																						
	h	R0																						
	l	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。

位操作语句 **置位 (=)**

(13) 置位 (=)

[编码格式]

```
[Δ] α1 [Δ] [=[Δ] α2 [Δ]…] = [Δ] 1 [Δ] [(CY 指定)]
```
在右侧的最后输入一个“1”。

[功能]

<1> 无CY指定时

α_n被置位（值设为1）。

<2> 有CY指定时

CY和α_n被置位（值设为1）

[声明]

α_n的值可被输入SET1指令。

一行中最多可输入32个赋值操作符“=”。输入多于32个操作符号时将出错。如果在序列赋值时出现一个错误，则不生成任何指令。

[生成指令]

<1> 无CY指定时

SET1 α1

<2> 在序列赋值无CY指定时

SET1 αn
SET1 αn-1
:
SET1 α2
SET1 α1

<3> 有CY指定时

SET1 CY
SET1 α1

位操作语句	置位 (=)

<4> 在序列赋值中有CY指定时

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

更多细节，参见**表4-14. 置位的生成指令**

[用例]

<1> 无CY指定时

```
SET1   A.3          ;A.3=1
SET1   CY           ;CY=1
SET1   BIT3         ;BIT1=BIT2=BIT3=1
SET1   BIT2
SET1   BIT1
```

<2> 有CY指定时

```
SET1   CY           ;A.5=1(CY)
SET1   A.5
SET1   CY           ;BIT1=BIT2=BIT3=1(CY)
SET1   BIT3
SET1   BIT2
SET1   BIT1
```

位操作语句	置位(=)

表4-14. 置位的生成指令

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 (=)**

(14) 复位 (=)

[编码格式]

```
[Δ] α 1 [=[Δ] α 2 [Δ]···] = [Δ] 0 [Δ] [(CY 指定)]
```
在右侧的最后输入一个“0”。

[功能]

<1> 无CY指定时

α_n被复位（值设为0）

<2> 有CY指定时

CY和α_n被复位（值设为0）。

[声明]

α_n的值可被输入CLR1指令。

一行中最多可输入32个赋值操作符“=”。输入多于32个操作符号时将出错。如果在序列赋值时出现一个错误，将不生成任何指令。

[生成指令]

<1> 无CY指定时

CLR1 α1

<2> 在序列赋值中无CY指定时

CLR1 αn
CLR1 αn-1
:
CLR1 α2
CLR1 α 1

<3> 无CY指定时

CLR1 CY
CLR1 α1

位操作语句	复位 (=)

<4> 在序列赋值中有CY指定时

```
CLR1   CY
CLR1   α n
CLR1   α n-1
       :
CLR1   α 2
CLR1   α 1
```

更多细节参见**表4-15. 复位的生成指令。**

[用例]

<1> 无CY指定时

```
CLR1   A.3          ;A.3=0
CLR1   CY           ;CY=0
CLR1   BIT3         ;BIT1=BIT2=BIT3=0
CLR1   BIT2
CLR1   BIT1
```

<2> 有CY指定时

```
CLR1   CY           ;A.5=0(CY)
CLR1   A.5
CLR1   CY           ;BIT1=BIT2=BIT3=0(CY)
CLR1   BIT3
CLR1   BIT2
CLR1   BIT1
```

位操作语句	复位 (=)

表4-15. 复位的生成指令

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
空格表示错误。

[备忘录]

第五章 指示

本章介绍指示。这里，“指示”表示 ST78K0S 需要执行一系列处理的各种指令。

5.1 指示概述

指示作为 ST78K0S 需要执行一系列处理所需要的各种指令而被输入源程序。
使用指示可使源程序的编码容易。
指示不再输出到输出文件中。

5.2 指示的功能

在**表 5-1. 指示列表**列出了各种类型的指示。

表 5-1. 指示列表

指示类型	指示名称
符号定义指示	#define
条件处理指示	#ifdef : #else : #endif
包含指示	#include
CALLT 置换指示	#defcallt : #endcallt

各种指示的功能描述如下。

#DEFINE **#define** **#DEFINE**

(1) 符号定义指示（#define）

[编码格式]

```
[Δ] # [Δ] define Δ 符号 Δ 字符串
```

[功能]

该指示用指定的字符串替代已经输入到源程序内的符号。

[声明]

<1> 除非以空格或水平制表符起始，“#”字符必须是符号的起始符。

<2>这些符号以一个英文字母开始，由英语字母表字母和数字构成。缺省条件下，有效长度为 31 个字符，若指定了 NS 选项，则缺省值为 8 个字符。当指定了有效长度为 8 个字符时，只有前 8 个字符可读入具有 9 个或更多字符的符号名中，后续字符被忽略。当指定了 31 个字符的有效长度时，只有前 31 个字符可读入具有 32 个或更多字符的符号名中，后续字符被忽略。

<3>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。

<4> 该指示有益于编码易于读取的符号，比如数字值。

<5> 保留字不能作为符号输入。

<6> 保留字可作为字符串输入。

<7> 如同一符号被定义两次，则输出告警信息。

<8> 输出已经被转换到辅助源文件的字符串。不输出#define 语句。

<9> 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。

<10>该指示可在源代码的任何位置输入。

<11>当两个或多个指定了选项 D 的符号被输入时，输出告警信息。但是#define 语句有效。

[用例]

<输入源程序>

```
#define TRUE     1
   X = #0
   CALL   !xxx
   if( X == #TRUE )
          A = #0C5H
   endif
```

<输出源程序>

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

(2) 条件处理指示（#ifdef/#else/#endif）

[编码格式]

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

[功能]

该指示执行条件处理

<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。

[声明]

<1> 除非以空格或水平制表符起始，“#”必须总是为符号的首字符。

<2> 这些符号以一个英文字母开始，由英语字母表字母和数字构成。缺省条件下，有效长度为 31 个字符，若指定了 NS 选项，则缺省值为 8 个字符。

<3>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。

<4> 该指令可最多嵌套 8 级。

<5> #else 可省略。

#IFDEF/#ELSE/#ENDIF　　#ifdef/#else/#endif　　#IFDEF/#ELSE/#ENDIF

[用例]

<输入源文件 >

```
#ifdef SYM
   A = #00H
#else
   A = #0FFH
#endif
```

<1> 当下列语句已在命令行中被输入（且符号已被定义）

```
A>st78k0s -cp9014 sample.st -dSYM
```

<输出源文件 >

```
MOV      A,#00H  ;       A = #00H
```

<2> 当下列语句已命令行中被输入（且符号未被定义）

```
A>st780s -cp9014 sample.st
```

<输出源文件 >

```
MOV      A,#0FFH ;       A = #0FFH
```

#INCLUDE	#include	#INCLUDE

(3) Include 指示(#include)

[编码格式]

```
[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。

[声明]

<1> 除非以空格或水平制表符起始，"#"字符必须总是作为符号的首字符。

<2> 该指示可在源程序的任何行输入。

<3> include 指示不能在 include 文件中输入。也就是说，不允许嵌套 include 指示。

<4> 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指示的文件名。

<5> 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。

<6> 当 ST78K0S 被激活时，-I 选项可用于为 include 文件指定驱动器和目录。

#INCLUDE **#include** **#INCLUDE**

[用例]

<输入源程序>

```
#include "sample.inc"
   A = SYM1
   B = SYM2
```

<输入 include 程序 >

```
#define   SYM1    #08H
#define   SYM2    #0AH
```

<输出源程序 >

```
MOV       A,#08H  ;       A = #08H
MOV       B,#0AH  ;       B = #0AH
```

#DEFCALLT **#defcallt** **#DEFCALLT**

(4) CALLT 替代指令 (#defcallt)

[编码格式]

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

[功能]

已注册标记的 CALL 指令被 CALLT 指令所代替，并输出至辅助文件。

[声明]

<1> 该指示定义了可被注册至 CALLT 表的标记，与输入至源程序的 CALL 指令相反。所有对这些已定义标记的 CALL 指令被 CALLT 标记所代替。

<2> 该指示最多可被定义 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。

<3> 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。

[用例]

<输入源程序>

```
    #defcallt       @ABC
       CALL         !abc
    #endcallt
    R0 = #0
    call            !abc
    call            !label
```

<输出源程序>

```
    MOV             R0,#0           ;R0 = #0
    CALLT           [@ABC]          ;call   !abc
    call            !label          ;call   !label
```

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导。

6.1 控制指令概述

控制指令可输入到源程序中，这些指令为ST78K0S执行一系列操作处理建立了各种不同的指导。

输入控制指令节省了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。

如果汇编器控制指令不能在模块头输入，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，处理过程宣告失败。

除了处理器类型说明控制指令($PROCESSOR, $PC)、符号名长度控制指令($SYMLEN, $NOSYMLEN)和kanji码指定控制指令($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更多细节，参见**“RA78K0S系列汇编器包用户手册 汇编语言”。**

下面的表格列出了只能在模块头输入的控制指令以及可作为模块体的控制指令。

表6-1. 仅可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] SYMLEN
[Δ] $ [Δ] NOSYMLEN
[Δ] $ [Δ] CAP
[Δ] $ [Δ] NOCAP
[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. 作为模块体被识别的控制指令

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] _ELESEIF Δ 条件表达式
[Δ] $ [Δ] SET [Δ] ([Δ] switch 名[[Δ] : [Δ] switch名… [Δ])
[Δ] $ [Δ] ELSE
[Δ] $ [Δ] ENDIF

6.3 控制指令功能

下面的**表6-3. 控制指令表**列出了控制指令的各种功能。

表6-3. 控制指令表

控制指令类型	控制指令
处理器类型说明指令	$PROCESSOR
符号名称长度控制指令	$SYMLEN/$NOSYMLEN
Kanji码指定控制指令	$KANJICODE

下面描述这三种控制指令的功能。

$PROCESSOR	$processor	$PROCESSOR

(1) 处理器类型说明指令($PROCESSOR)

[编码格式]

```
[Δ] $ [Δ] PROCESSOR [Δ] ( [Δ] 模型名称 [Δ] )
[Δ] $ [Δ] PC [Δ] ( [Δ]模型名称[Δ] )                                    ; 缩写形式
```

[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。

[声明]

<1> 虽然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。

<2> 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一个告警信息。输入源文件控制指令中的“$”被输出的辅助源文件中的“;”代替；由选项指定的类型作为处理器类型说明指令输出。如果由“-C”选项指定的是同一个类型名称，则无信息输出。如果“-C”选项没有指定类型，必须在源模块的开始部分（不包括空格或注释）即输入该指定。

<3>当该控制指令输入多次时报错。

<4> 若控制指令或“-C”选项都没有指定类型名称，则报错。

<5> 若控制指令没有在模块头输入，则报错。

[代码示例]

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

(2) 符号名称长度控制指令

[编码格式]

```
[Δ] $ [Δ] SYMLEN
[Δ] $ [Δ] NOSYMLEN
```

[功能]

SYMLEN控制指令给由#define定义的符号名称设置有效长度，最大可设置31个字符，符号名称通过#ifdef和用户符号访问。

NOSYMLEN控制指令给由#define定义的符号名称设置名称长度，最大可设置8个字符，符号名称通过#ifdef和用户符号访问。

[声明]

<1> 该控制指令可在输入源文件的模块头部分输入。

<2> 如果该控制指令在模块头部分之外的地方输入，则报错。

<3> 如果该控制指令被输入多次，最近一次的输入优先。

<4> 符号名称长度控制指令也可通过命令行选项“-S”和“-NS”指定。这些选项优先级高于控制指令。

<5> 缺省解释是$SYMLEN。

<6> 如果指定了“-S” 选项且在输入源文件中输入了$NOSYMLEN，则注释将被取代，且$SYMLEN将输出至辅助源文件。

如果指定了“-NS”选项且在输入源文件中输入了$SYMLEN，则注释将被取代，且$NOSYMLEN将输出至辅助源文件。

[代码示例]

```
$SYMLEN
$NOSYMLEN
```

$KANJICODE **$kanjicode** **$KANJICODE**

(3) Kanji码指定控制指令($KANJICODE)

[编码格式]

```
[Δ] $ [Δ] KANJICODE Δ kanji code
```

[功能]

在注释中使用的kanji码说明如下。

表6-4. kanji码说明

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

[声明]

<1>该控制指令可在输入源文件的模块头部分输入。

<2> 如果该控制指令在模块头部分之外的地方输入，则报错。

<3> 如果该控制指令被输入多次，最近一次的输入优先。

<4>该预处理器输出指定控制指令至辅助源文件。

SJIS ： $KANJICODE SJIS

EUC ： $KANJICODE EUC

NONE： $KANJICODE NONE

如果同一控制指令在辅助源文件中输入，则不输出控制指令。但是，仍执行错误检查。

<5> 缺省解释是$SYMLEN。

<6> kanji码指定的优先级列表，参见“**1.3.3 环境变量**”。

[代码示例]

```
$KANJICODE SJIS
```

[备忘录]

附录A　语法明细表

表A-1.　控制语句

控制语句	编码格式	页码

if语句	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
		P.21

switch语句	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.	P.27
for语句	for (表达式; 条件表达式; 表达式) [(寄存器名)] 　指令组 next	P.31
while语句	while (条件表达式) [(寄存器名)] 　指令组 endw	P.34
until语句	repeat 　指令组 until (条件表达式) [(寄存器名)]	P.38
break语句	break	P.41
continue语句	continue	P.42
goto语句	Goto标记	P.43

if_bit语句	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
		P.24

while_bit语句	while_bit (条件表达式) [(寄存器名)] 　指令组 endw	P.36
until_bit语句	repeat 　指令组 until_bit (条件表达式) [(寄存器名)]	P.40

表A-2. 条件表达式

条件表达式	编码格式	功能	页码
等于	α == β	当α = β时为真， 当$\alpha \neq \beta$时为假	P.47
不等于	α != β	当$\alpha \neq \beta$是为真， 当α = β时为假	P.50
小于	α < β	当α <β是为真， 当α >= β时为假	P.53
大于	α > β	当α > β是为真， 当α <= β时为假	P.56
大于等于	α >= β	当α >= β是为真， 当α < β时为假	P.59
小于等于	α <= β	当α <= β是为真， 当α > β时为假	P.62
无限循环	forever	对loop语句设置无限循环	P.65
正逻辑（bit）	Bit符号	当被指定的bit符号为1时	P.68
负逻辑（bit）	!bit符号	当被指定的bit符号为0时	P.71
逻辑与 AND	表达式1 && 表达式2	当表达式1与表达式2都为真时结果为真	P.75
逻辑或 OR	表达式1 || 表达式2	当表达式1或表达式2中有一个为真时，结果为真	P.78

表A-3. 表达式（1/2）

表达式	编码格式	功能	页码
赋值	$\alpha = \beta$	$\alpha \leftarrow \beta$	P.83
赋值（带寄存器定义）	$\alpha = \beta\ (\gamma)$	$(\gamma) \leftarrow \beta$ $\alpha \leftarrow (\gamma)$	
序列赋值	$\alpha 1 = \cdots = \alpha n = \beta$	$\alpha 1 \leftarrow \beta, \cdots, \alpha n \leftarrow \beta$	
序列赋值（带寄存器定义）	$\alpha 1 = \cdots = \alpha n = \beta\ (\gamma)$	$\gamma \leftarrow \beta, \alpha 1 \leftarrow \gamma, \cdots, \alpha n \leftarrow \gamma$	
递增赋值	α += β	$\alpha \leftarrow \alpha + \beta$	P.88
递增赋值（带寄存器定义）	α += β (寄存器)	$\gamma \leftarrow \alpha, \gamma \leftarrow \gamma + \beta, \alpha \leftarrow \gamma$	
递增赋值（带寄存器定义）	α += β, CY	$\alpha \leftarrow \alpha + \beta$, CY	
递增赋值（带寄存器定义）	α += β, CY (寄存器)	$\gamma \leftarrow \alpha, \gamma \leftarrow \gamma + \beta$, CY, $\alpha \leftarrow \gamma$	
递减赋值	α –= β	$\alpha \leftarrow \alpha - \beta$	P.92
递减赋值（带寄存器定义）	α –= β (寄存器)	$\gamma \leftarrow \alpha, \gamma \leftarrow \gamma - \beta, \alpha \leftarrow \gamma$	
递减赋值（带寄存器定义）	α –= β, CY	$\alpha \leftarrow \alpha - \beta$, CY	
递减赋值（带寄存器定义）	α –= β, CY (寄存器)	$\gamma \leftarrow \alpha, \gamma \leftarrow \gamma - \beta$, CY, $\alpha \leftarrow \gamma$	
逻辑与赋值	α &= β	$\alpha \leftarrow \alpha \cap \beta$	P.96
逻辑与赋值（带寄存器定义）	α &= β (寄存器)	$\gamma \leftarrow \alpha, \gamma \leftarrow \gamma \cap \beta, \alpha \leftarrow \gamma$	

表A-3. 表达式（2/2）

表达式	编码格式	功能	页码
逻辑或OR赋值	α |= β	$\alpha \leftarrow \alpha \cup \beta$	P.99
逻辑或赋值（带寄存器定义）	α |= β (寄存器)	$\gamma \leftarrow \alpha$, $\gamma \leftarrow \gamma \cup \beta$, $\alpha \leftarrow \gamma$	
逻辑异或XOR赋值	α ^= β	$\alpha \leftarrow \alpha \hat{} \beta$	P.103
逻辑异或XOR赋值（带寄存器定义）	α ^= β (寄存器)	$\gamma \leftarrow \alpha$, $\gamma \leftarrow \gamma \hat{} \beta$, $\alpha \leftarrow \gamma$	
右移赋值（循环）	α >>= β	*α右移β位*	P.107
右移赋值（带寄存器定义）	α >>= β (寄存器)	$\gamma \leftarrow \alpha$,(*γ右移β位*), $\alpha \leftarrow \gamma$	
左移赋值	α <<= β	*α左移β位*	P.109
左移赋值（带寄存器定义）	α <<= β (寄存器)	$\gamma \leftarrow \alpha$, (*γ左移β位*), $\alpha \leftarrow \gamma$	
递增	α ++	$\alpha \leftarrow \alpha + 1$	P.111
递减	α – –	$\alpha \leftarrow \alpha - 1$	P.113
调换	α <–>= β	$\alpha \leftarrow \alpha$ <–>= β	P.115
调换（带寄存器定义）	α <–>= β (γ)	$\gamma \leftarrow \alpha$, $\gamma \leftarrow \gamma$ <–> β, $\alpha \leftarrow \gamma$	
置位	α = 1	$\alpha \leftarrow 1$	P.118
置位（带寄存器定义）	α = 1 (CY)	CY $\leftarrow$ 1, $\alpha \leftarrow$ 1	
序列置位	α 1 = … =α n = 1	α n $\leftarrow$ 1, … , α 1 $\leftarrow$ 1	
序列置位（带寄存器定义）	α 1 = … =α n = 1 (CY)	CY $\leftarrow$ 1, α n $\leftarrow$ 1, … , α 1 $\leftarrow$ 1	
复位	α = 0	$\alpha \leftarrow 0$	P.121
复位（带寄存器定义）	α = 0 (CY)	CY $\leftarrow$ 0, $\alpha \leftarrow$ 0	
序列复位	α 1 = … =α n = 0	α n $\leftarrow$ 0, … , α 1 $\leftarrow$ 0	
序列复位（带寄存器定义）	α 1 = … =α n = 0 (CY)	CY $\leftarrow$ 0, α n $\leftarrow$ 0, … , α 1 $\leftarrow$ 0	

表 A-4. 指令

指令	编码格式	页码
#define	#define 符号 特征字符串	P.126
#ifdef	#ifdef 符号 文本1 #else 文本2 #endif	P.128
#include	#include “文件名”	P.130
#defcallt	#defcallt CALLT 表格标签 CALL 标签 #endcallt	P.132

[备忘录]

附录 B　生成指令表

表 B-1. 比较表达式的生成指令（1/3）

比较表达式	生成指令	控制语句的条件	页码
α == β	CMP(W) α, β BNZ $?LFALSE	小写字母	P.47
	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	
α! = β	CMP(W) α, β BZ $?LFALSE	小写字母	P.50
	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	
α < β	CMP(W) α, β BNC $?LFALSE	小写字母	P.53
	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（2/3）

比较表达式	生成指令	控制语句的条件	页码
$\alpha > \beta$	CMP(W) α, β BZ $?LFALSE BC $?LFALSE	小写字母	P.56
	CMP(W) α, β BZ $$+4 BNC $?LTRUE BR ?LFALSE ?LTRUE:	大写字母	
$\alpha > \beta$ (γ)	MOV(W) 指定寄存器, α CMP(W) 指定寄存器, β BZ $?LFALSE BC $?LFALSE	小写字母	
	MOV(W) 指定寄存器, α CMP(W) 指定寄存器, β BZ $$+4 BNC $?LTRUE BR ?LFALSE ?LTRUE:	大写字母	
$\alpha >= \beta$	CMP(W) α, β BC $?LFALSE	小写字母	P.59
	CMP(W) α, β BNC $?LTRUE BR ?LFALSE ?LTRUE:	大写字母	
$\alpha >= \beta$ (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（3/3）

比较表达式	生成指令	控制语句的条件	页码
α <= β	CMP(W) α, β BZ $$+4 BNC $?LFALSE	小写字母	P.62
	CMP(W) α, β BZ $?LTRUE BC $?LTRUE BR ?LFALSE ?LTRUE:	大写字母	
α <= β (γ)	MOV(W) 指定寄存器, α CMP(W) 指定寄存器, β BZ $$+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. 测试位表达式的生成指令

测试比特表达式	生成指令		控制语句的条件	页码
if_bit (bit 符号) elseif_bit (bit 符号) while_bit (bit 符号) until_bit ((bit 符号)	BNC	$?LFALSE	小写字母(CY)	P.68
	BNZ	$?LFALSE	小写字母(Z)	
	BF	比特符号, $?LFALSE	小写字母	
	BC BR ?LTRUE:	$?LTRUE ?LFALSE	大写字母(CY)	
	BZ BR ?LTRUE:	$?LTRUE ?LFALSE	大写字母(Z)	
	BT BR ?LTRUE:	比特符号, $?LTRUE ?LFALSE	大写字母	P.69
if_bit (!bit 符号) elseif_bit (!bit 符号) while_bit (!bit 符号) until_bit ((!bit 符号)	BC	$?LFALSE	小写字母(CY)	P.71
	BZ	$?LFALSE	小写字母(Z)	
	BT	比特符号, $?LFALSE	小写字母	
	BNC BR ?LTRUE:	$?LTRUE ?LFALSE	大写字母(CY)	
	BNZ BR ?LTRUE:	$?LTRUE ?LFALSE	大写字母(Z)	
	BF BR ?LTRUE:	比特符号, $?LTRUE ?LFALSE	小写字母	P.72

表 B-3. 逻辑表达式的生成指令（1/2）

逻辑表达式	生成指令		控制语句的条件	页码
α == β &&	CMP(W) BNZ	α, β $?LFALSE	小写字母	P.75, 76
	CMP(W) BZ BR ?LTRUE:	α, β $?LTRUE ?LFALSE	大写字母	
α != β &&	CMP(W) BZ	α, β $?LFALSE	小写字母	
	CMP(W) BNZ BR ?LTRUE:	α, β $?LTRUE ?LFALSE	大写字母	
α < β &&	CMP(W) BNC	α, β $?LFALSE	小写字母	
	CMP(W) BC BR ?LTRUE:	α, β $?LTRUE ?LFALSE	大写字母	
α > β &&	CMP(W) BZ BC	α, β $?LFALSE $?LFALSE	小写字母	
	CMP(W) BZ BNC BR ?LTRUE:	α, β $$+4 $?LTRUE ?LFALSE	大写字母	
α >= β &&	CMP(W) BC	α, β $?LFALSE	小写字母	
	CMP(W) BNC BR ?LTRUE:	α, β $?LTRUE ?LFALSE	大写字母	
α <= β &&	CMP(W) BZ BNC	α, β $$+4 $?LFALSE	小写字母	
	CMP(W) BZ BC BR ?LTRUE:	α, β $?LTRUE $?LTRUE ?LFALSE	大写字母	
CY &&	BNC	$?LFALSE	小写字母	
	BC BR ?LTRUE:	$?LTRUE ?LFALSE	大写字母	

表 B-3. 逻辑表达式的生成指令（2/2）

逻辑表达式	生成指令		控制语句的条件	页码
Z &&	BNZ	$?LFALSE	小写字母	P.75, 76
	BZ BR ?LTRUE:	$?LTRUE ?LFALSE	大写字母	
比特符号 &&	BF	比特符号, $?LFALSE	小写字母	
	BT BR ?LTRUE:	比特符号, $?LTRUE ?LFALSE	大写字母	
!CY &&	BC	$?LFALSE	小写字母	
	BNC BR ?LTRUE:	$?LTRUE ?LFALSE	大写字母	
!Z &&	BZ	$?LFALSE	小写字母	
	BNZ BR ?LTRUE:	$?LTRUE ?LFALSE	大写字母	
! 比特符号&&	BT	比特符号, $?LFALSE	小写字母	
	BF BR ?LTRUE:	比特符号, $?LTRUE ?LFALSE	大写字母	
α == β ||	CMP(W) BZ	α, β $?LFALSE		P.78
α!= β ||	CMP(W) BNZ	α, β $?LFALSE		
α < β ||	CMP(W) BC	α, β $?LFALSE		
α > β ||	CMP(W) BZ BNC	α, β $?LFALSE $?LFALSE		
α >= β ||	CMP(W) BNC	α, β $?LFALSE		
α <= β ||	CMP(W) BZ BC	α, β $?LFALSE $?LFALSE		
CY ||	BC	$?LFALSE		
Z ||	BZ	$?LFALSE		
比特符号 ||	BT	比特符号, $?LFALSE		
!CY ||	BNC	$?LFALSE		
!Z ||	BNZ	$?LFALSE		
! 比特符号||	BF	比特符号, $?LFALSE		

表 B-4. 表达式（1/4）

表达式	生成指令		页码
$\alpha = \beta$	MOV	α 1, β	P.85
	MOVW	α 1, β	
	BNC SET1 BR ?L1: CLR1 ?L2:	?L1 α ?L2 α	P.84
$\alpha = \beta$ (γ)	MOV MOV	γ, β α 1, γ	
	MOVW MOVW	γ, β α 1, γ	
	BF SET1 BR ?L1: CLR1 ?L2:	β, ?L1 α ?L2 α	
α += β	ADD	α, β	P.89
	ADDW	α, β	
α += β (γ)	MOV ADD MOV	γ, α γ, β α, γ	
	MOVW ADDW MOVW	γ, α γ, β α, γ	
α += β, CY	ADDC	α, β	
α += β, CY (γ)	MOV ADDC MOV	γ, α γ, β α, γ	

表 B-4. 表达式（2/4）

表达式	生成指令		页码
α –= β	SUB	α, β	P.93
	SUBW	α, β	
α –= β (γ)	MOV SUB MOV	γ, α γ, β α, γ	
	MOVW SUBW MOVW	γ, α γ, β α, γ	
α –= β, CY	SUBC	α, β	
α –= β, CY (γ)	MOV SUBC MOV	γ, α γ, β α, γ	
α & = β	AND	α, β	P.96
	BNC BF SET1 BR ?L1: CLR1 ?L2:	?L1 β, ?L1 CY ?L2 CY	
α &= β (γ)	MOV AND MOV	γ, α γ, β α, γ	P.97
	BF BF SET1 BR ?L1: CLR1 ?L2:	α, ?L1 β, ?L1 α ?L2 α	

表 B-4. 表达式（3/4）

表达式	生成指令	页码
α | = β	OR α, β	P.99
	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
		P.100

	BT α, ?L1 BF β, ?L2 ?L1: SET1 α BR ?L3 ?L2: CLR1 α ?L3:	
α ^ = β	XOR α, β	P.103
	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
		P.104

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

表 B-4. 表达式（4/4）

表达式	生成指令		页码
α >>= β	ROR : AND	A, 1 A, #0FFH SHR β	P.107
α >>= β (γ)	MOV ROR : AND MOV	A, α A, 1 A, #0FFH SHR β α, A	
α <<= β	ROL : AND	A, 1 A, #LOW (0FFH SHR β)	P.109
α <<= β (γ)	MOV ROL : AND MOV	A, α A, 1 A, #LOW (0FFH SHL β) α, A	
α ++	INC	α	P.111
	INCW	α	
α – –	DEC	α	P.113
	DECW	α	
α <–> β	XCH	α, β	P.115
	XCHW	α, β	
α <–> β (γ)	MOV XCH MOV	γ, α γ, β α, γ	
	MOVW XCHW MOVW	γ, α γ, β α, γ	
α = 1	SET1	α 1	P.118
α = 1 (CY)	SET1 SET1	CY α 1	
α = 0	CLR1	α 1	P.121
α = 0 (CY)	CLR1 CLR1	CY α 1	

附录C 最佳性能

表C-1. 结构化汇编器的最佳性能

项目	最大值
行长度（不包括LF或CR）	218字符
在#define指令中注册的符号数（保留字除外）	512符号
控制语句的嵌套级数	31级
#ifdef指令中的嵌套级数	8级
#defcallt指令	32
#include指令的嵌套级数	不支持
被#define指令重新定义的次数	31次
在一个序列中被赋值的操作数的个数	**33**[注解1]
逻辑操作符的操作数	**17**[注解2]
被-D选项定义的符号数	30

注解 **1.** 最大值表示如下：

S1=S2= … S32=S33

最多可插入33个符号，其中包括32个等号（=）。

2. 最大值表示为：

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。

[备忘录]