お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

RENESAS

ユーザーズ・マニュアル

RA78K0S

アセンブラ・パッケージ

構造化アセンブリ言語編

ST78K0S V1.00 以上

資料番号　U11623JJ1V1UMJ1（第1版）
発行年月　August 2000 N　CP(K)

〔メ　モ〕

- 本資料の内容は予告なく変更することがありますので，最新のものであることをご確認の上ご使用ください。
- 文書による当社の承諾なしに本資料の転載複製を禁じます。
- 本資料に記載された製品の使用もしくは本資料に記載の情報の使用に際して，当社は当社もしくは第三者の知的財産権その他の権利に対する保証または実施権の許諾を行うものではありません。上記使用に起因する第三者所有の権利にかかわる問題が発生した場合，当社はその責を負うものではありませんのでご了承ください。
- 本資料に記載された回路，ソフトウエア，及びこれらに付随する情報は，半導体製品の動作例，応用例を説明するためのものです。従って，これら回路・ソフトウエア・情報をお客様の機器に使用される場合には，お客様の責任において機器設計をしてください。これらの使用に起因するお客様もしくは第三者の損害に対して，当社は一切その責を負いません。

はじめに

このマニュアルは，78K/0 シリーズの中の小型汎用マイクロコンピュータのアセンブラ・パッケージ「RA78K0S アセンブラ・パッケージ」に含まれている構造化アセンブラ・プリプロセッサ（以降，構造化アセンブラとします）について，記述方法を正しく理解していただくことを目的として書かれています。

このマニュアルでは，構造化アセンブラ以外のプログラム，および，構造化アセンブラの操作方法については説明していません。

プログラムを書かれる際には，アセンブラ・パッケージのユーザーズ・マニュアル（アセンブリ言語編，操作編）も併せてお読みください。

【対象者】

このマニュアルは，78K/0 シリーズの中の小型汎用マイクロコンピュータの機能を理解されている方を対象として書かれています。

78K/0 シリーズの中の小型汎用マイクロコンピュータの機能に関しては，各デバイスのユーザーズ・マニュアルを参照してください。

【対象デバイス】

RA78K0S アセンブラ・パッケージが対象としている各デバイス。

【構　成】

このマニュアルは次のようなに構成されています。

○第 1 章　概　説

マイクロコンピュータのソフトウエア開発における構造化アセンブラの役割など，機能概要について説明します。

○第 2 章　ソース・プログラムの記述方法

ソース・プログラムの構成方法，記述時の文法など，ソース・プログラムの記述時の大まかな規則について説明します。

○第 3 章　制御文

プログラム構造を示す "if～else～endif" などは，制御文で記述します。

ここでは，制御文の機能と記述方法を説明します。

○第４章　式の文

代入や演算は，式の文で記述します。

ここでは，式の文の機能と記述方法を説明します。

○第５章　擬似命令

構造化アセンブラの擬似命令について，その書き方，使い方を使用例を交えて説明します。

○第６章　制御命令

構造化アセンブラの制御命令について，その書き方，使い方を使用例を交えて説明します。

○付録Ａ　構文一覧

構造化アセンブラの構文の一覧を示します。

○付録Ｂ　生成命令一覧

構造化アセンブラが生成する命令の一覧を示します。

○付録Ｃ　最大性能

構造化アセンブラの最大性能を示します。

【読み方】

構造化アセンブラを初めて使われる方は，「第１章　概説」からお読みください。

構造化アセンブラについて一般的知識のある方は，第１章を読み飛ばされても結構です。

ただし，「1.3　プログラム開発を始める前に」は，必ずご一読ください。

【凡例】

…	：同一の形式を繰り返します。
［　］	：かっこ内は省略可能です。
「　」	：「　」で囲まれた文字，文字列を示します。
’　’	：’　’で囲まれた文字，文字列を示します。
“”	：参照する場所を示します。
△	：１個以上の空白またはタブを示します。
太文字	：文字そのものを示します。
＿＿＿	：入力文字列を示します
：	：プログラムの省略形
（　）	：（　）内で囲まれた文字列
CR	：　復帰（キャリッジ・リターン）
LF	：　改行（ライン・フィード）
／	：　区切り記号

α : レジスタ名など，ニモニックのオペランドに記述するもの

β : レジスタ名など，ニモニックのオペランドに記述するもの

γ : レジスタ名など，ニモニックのオペランドに記述するもの

δ : レジスタ名など，ニモニックのオペランドに記述するもの

目 次

図　の　目　次

表 の 目 次

第１章　概　　説

この章では，マイクロコンピュータの開発における ST78K0S の役割や特徴について説明します。

1.1 構造化アセンブラの概要

RA78K0S 構造化アセンブラ・プリプロセッサは，「RA78K0S アセンブラ・パッケージ」に添付されている 78K/0 シリーズの中の小型汎用マイクロコンピュータのソフトウエア開発用のプログラムです。

このマニュアルでは，構造化アセンブラ・プリプロセッサを「構造化アセンブラ」または「ST78K0S（Structured assembler）」と略しています。

構造化アセンブラとは，プログラム構造を表す "if～else～endif" や "for～next" などをアセンブラのソース・プログラムに変換するものです。"if～else～endif" や "for～next"は制御文を用いて記述します。

構造化アセンブラの利点として次の 3 つがあります。

① プログラムが書きやすい
- プログラム構造がそのまま書けるので，設計からコーディングが楽にできます。
- 分岐のためのレーベル名を考える必要がありません。
- 記述量の多い転送命令を代入文の形で記述できます。

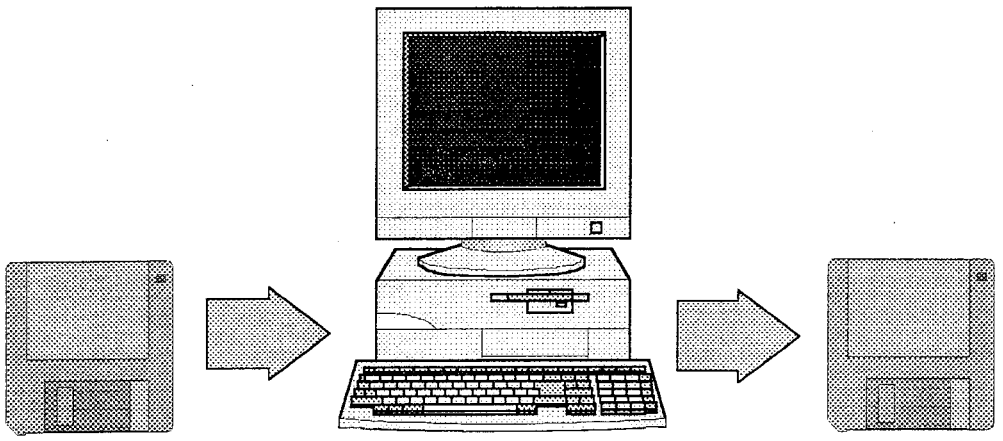
② プログラムが読みやすい
- プログラムの構造が明確になります。
- メモリ・レジスタ間の演算，転送が１ステートメントで記述可能です。
- 他人のプログラムが読みやすくなります。
- プログラムの保守（改造）が容易になります。

③ 机上デバッグがしやすい
- 詳細設計と１対１に対応した記述ができるので，机上ディバグが楽になります。

1.2　機能概要

構造化アセンブラは，専用の言語仕様に基づいて記述された構造化アセンブラ・ソース・プログラム中の各種制御文，式，擬似命令を解析し，アセンブラの入力ソースファイルとなるアセンブラ・ソース・プログラムを出力します。

図 1-1　構造化アセンブラの流れ



2 次ソースファイルには，コメント文としての構造化ステートメント，変換後のアセンブラの命令，および通常のアセンブリ言語が出力されます。

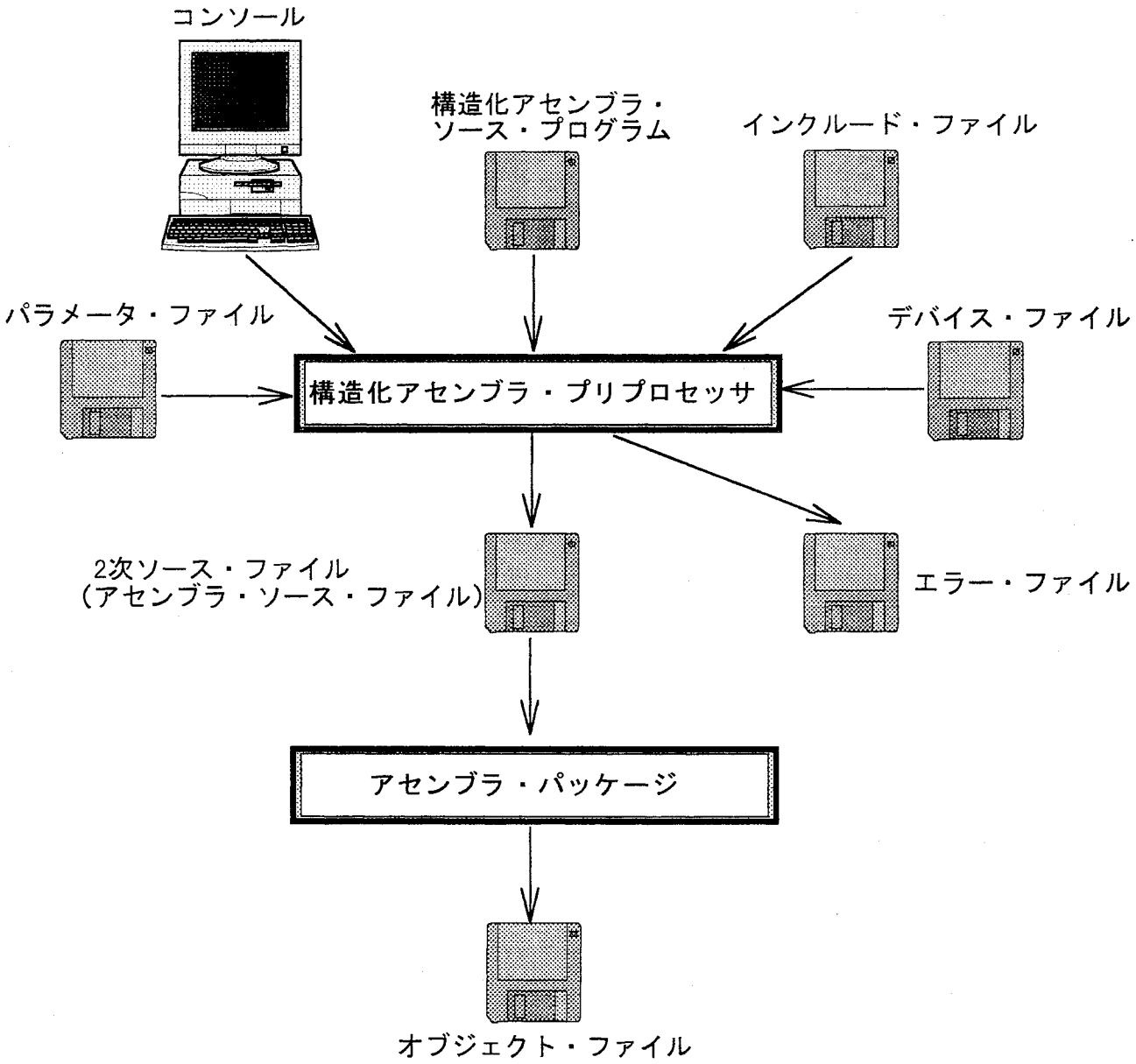
また，エラーがあった場合にはエラー・メッセージを出力します。

構造化アセンブラの主な機能を以下に示します。

① C言語風の豊富な制御構造を持っているので，プログラムの記述が容易です。
② C言語風の代入文，代入演算子等が記述できます。
③ ビット処理について，制御構造，代入文が記述できます。
④ C言語風のシンボル定義擬似命令，条件付き処理機能，インクルード擬似命令があります。
⑤ アセンブラのソース・プログラムを出力するプリプロセッサですので，構造化アセンブラで変換後にコードの最適化を行えます。
⑥ CALLT 命令に変換可能な擬似命令が用意されているので，プログラム開発後に CALLT テーブルに登録するルーチンを決定することができます。
⑦ アセンブラ・ソース・プログラムの出力位置を変更することにより，読みやすいアセンブル・リストを作成できます。

図 1-2 にプログラム開発の処理の流れを示します。

図 1-2　処理の流れ



注意：デバイス・ファイルは別売りです。

1.3 プログラム開発をはじめる前に

構造化アセンブラの最大性能は，以下のようになっています。プログラムを書き始める前に，これらの値をご了承願います。

1.3.1 最大性能

以下に，構造化アセンブラの最大性能を示します。

表 1-1 構造化アセンブラの最大性能

項　　目	制限値
一行の長さ (LF,CR は含まない)	218 個
#define 擬似命令の登録シンボル数 (予約語は除く)	512 個
制御文のネスティング・レベル	31 レベル
#ifdef 擬似命令のネスティング・レベル	8 レベル
#defcallt 擬似命令	32 個
#include 擬似命令のネスティング	できません
#define 擬似命令で再定義可能な回数	31 回
連続代入できるオペランド数	33 個[注1]
論理演算子のオペランド	17 個[注2]
-D オプションで定義可能なシンボル数	30 個

注 1：以下のようになります。

S1=S2= … S32=S33

シンボル数は 33 個，“=”は 32 個まで記述可能です。

注 2：以下のようになります。

式 1&&式 2&& … &&式 16&&式 17

式は 17 個，“&&（または||）”は 16 個まで記述可能です。

1.3.2 注意事項

(1) ワード・シンボルとバイト・シンボル

構造化アセンブラは，ユーザ・シンボルの最後の文字によって，そのシンボルがワード・シンボルかバイト・シンボルかを判断します。ワード・シンボルを示す文字は「-SC オプション」で変えられますが，ディフォールトでは 'P' になっています。

-SC オプションについては，"RA78K0S アセンブラ・パッケージ 操作編" を参照してください。

例 1

構造化アセンブラ

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

アセンブラ

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

例 2 構造化アセンブラの起動コマンド

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

ワード・シンボルを示す文字を@にします。

入力ファイル指定

構造化アセンブラのコマンド名

構造化アセンブラ

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

アセンブラ

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

(2) レーベル（アセンブラでアドレスを示すシンボル）の定義

レーベルを定義する場合は，レーベルの定義と構造化アセンブラの文は別の行に記述してください。

例　○

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

×

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

1.3.3　環境変数

LANG78K はコメントに記述された漢字コードを指定します。

（1）　記述形式

SET△LANG78K＝漢字コード

漢字コード

SJIS　：シフト JIS コード

EUC　：EUC コード

NONE：漢字コードの処理は行わない

（2）　機能

・環境変数が設定されていない場合の漢字コードは OS によって次のようになります。

MS-DOS	：	SJIS
PC-DOS	：	NONE
SunOS	：	EUC
HP-UX	：	SJIS
NEWS-OS	：	SJIS

・漢字コードの指定の優先順位は次のようになります。

①-ZS／-ZE／-ZN オプションの指定

②漢字コード指定制御命令（$KANJICODE）の指定

③環境変数 LANG78K の指定

④各 OS のディフォールトの指定

第２章　ソース・プログラムの記述方法

この章では，ソース・プログラムの記述方式，記述様式などについて説明します。

2.1　ソース・プログラムの基本構成

ソース・プログラムは，構造化アセンブリ言語とアセンブリ言語で構成されます。

アセンブリ言語については，“RA78K0S アセンブラ・パッケージ アセンブリ言語編”を参照してください。

1 行（LF と LF の間）は，218 文字まで記述できます。

構造化アセンブリ言語には，“表 2-1　構造化アセンブリ言語の記述”で示す種類の記述があります。

表 2-1　構造化アセンブリ言語の記述

種類			記　述
構造化アセンブラの文	制御文	条件分岐	if～elseif～else～endif if_bit～elseif_bit～else～endif switch～case～default～ends
		条件ループ	for～next while～endw while_bit～endw repeat～until repeat～until_bit
		その他	break，continue，goto
	式の文	代入文	代入（=），演算代入（+=など），シフト代入（>>=など）
		カウント文	インクリメント（++），デクリメント（--）
		交換文	交換（<->）
条件式		比較条件式	==，!=，<，>，>=，<=
		ビット条件式	ビット・アドレス，！ビット・アドレス
		論理演算	論理積（&&），論理和（||）

条件式は，制御文の条件として記述します。

詳細は，“3.4　制御文の機能”を参照してください。

(1)　制御文

制御文には，条件分岐を表す if，switch～case，条件ループを表す for～next，while，repeat～until と，ループから抜ける処理などを表す break，continue，goto があります。詳細は，“第 3 章　制御文”を参照してください。

(2)　式の文

式の文には，代入文，カウント文（インクリメント，デクリメント），交換文（イクスチェンジ）があります。詳細は，“第 4 章　式の文”を参照してください。

2.2　ソース・プログラムの構成要素

(1)　文字セット

ソース・プログラムには，英字，数字，特殊文字が使用できます。

表 2-2　英数字

数字		0 1 2 3 4 5 6 7 8 9
英字	大文字	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
	小文字	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

ST78K0S では，制御文の先頭の文字のみ大文字，小文字の区別を行います。それ以外の英小文字は，英大文字に変換して処理を行います。ただし，2 次ソース・ファイルには，入力ファイルに記述したそのままの形で出力します。

表 2-3　特殊文字

文字	名称	用　途
?	疑問符	英字相当文字
@	単価記号	英字相当文字
_	下線	英字相当文字
	空白	各字句の区切り記号
HT	水平タブ	空白相当文字
,	コンマ	オペランド間の区切り記号
.	ピリオド	ビット・シンボルのビット位置記号
"	ダブル・コート	#INCLUDE 擬似命令のディスク型ファイル名の指定文字
'	シングル・コート	文字定数の開始，終了記号
+	プラス	正符号または加算演算子
−	マイナス	負符号または減算演算子
&	アンパーサンド	論理積演算子
|	分離記号	論理和演算子
^	上向矢印記号	排他的論理和演算子
(	左括弧	演算順序の変更，または制御文の式
)	右括弧	
=	等号	代入演算子，比較演算子
:	コロン	レーベルの区切り記号
;	セミコロン	コメントの開始記号，または制御文中の式の句切り記号
#	シャープ	構造化アセンブラの擬似命令先頭文字あるいはイミディエート表示記号
$	ドル記号	ロケーション・カウンタの値 制御命令指示記号
!	感嘆符	ダイレクト・アドレッシング指定記号，否定表示記号
<	不等号	比較演算子
>	不等号	
¥	円記号	ディレクトリ指定記号
[	左ブラケット	インダイレクト・アドレッシング指定記号
]	右ブラケット	
LF	改行	行の終端記号

以下の不正文字が入力された場合は，エラーとなります。

表 2-4　不正文字

	ASCII コード
不当文字	00H～08H，0BH，0CH，0EH～1FH，7FH
認識しない特殊文字	%（25H），`（60H），{（7BH），}（7DH），~（7EH ）
その他の文字	80H～0FFH

不当文字が入力された場合はエラーとなり，2 次ファイルを出力する場合には，’．’に置き換えて出力します。

ただし，コメント欄においては，不正文字の記述を可能とします。

(2)　識別子

識別子とは，数値データやアドレスなどにつけた名前のことです。

識別子の使用により，ソース・プログラムの内容がわかりやすくなります。

識別子の詳細については，#define 文で定義します（詳細については，“5.2　擬似命令の機能”を参照してください）。

(3)　シンボル

構造化アセンブラは，バイト・アクセスの命令を生成するか，ワード・アクセスの命令を生成するかを，シンボル名の最後の文字で判断します。-SC オプションで変更できますが，ディフォールトは P（ペアを意味しています）です。

予約語シンボル以外の文字列を，ユーザ・シンボルとして扱います。ユーザ・シンボルとは英数字および英字相当文字で定める文字を使用して記述します。

(4)　定数

構造化アセンブリ言語には，定数がありません。したがって，アセンブリ言語の定数は，文字列としてそのまま 2 次ソース・ファイルに出力されます（アセンブリ言語の定数については，“RA78K0S アセンブラ・パッケージ アセンブリ言語編”を参照してください）。

(5)　式

式は，定数，特殊文字，シンボルを演算子により結合したものです（アセンブリ言語の式については，“RA78K0S アセンブラ・パッケージ アセンブリ言語編”を参照してください）。

アセンブリ言語の式のうち，空白を区切り記号として記述する場合，かっこ " () " でその式を囲んでください。

例

① アセンブラ上での記述方法

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

② 構造化アセンブラのソース・プログラム上での記述方法

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

2.3　予約語

構造化アセンブリ言語の予約語を以下の表に示します。

ただし，インストラクション，sfr シンボルについては，各デバイスのユーザーズ・マニュアルを参照してください。

表 2-5　予約語シンボル（1/2）

	予約語
制御文	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
擬似命令	DFINE
	IFDEF, ELSE, ENDIF
	INCLUDE
	DEFCALLT, ENDCALLT
演算子	++, --
	=, +=, -=, &=, |=, ^=, <<=, >>=, <->
	==, !=, <, >=, >, <=, FOREVER

表 2-5　予約語シンボル（2/2）

アセンブラ演算子	MOD，NOT
	AND，OR，XOR
	EQ，NE，GT，GE，LT，LE
	SHL，SHR
	HIGH，LOW
	DATAPOS，BITPOS，MASK
アセンブラ制御命令	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，
レジスタ	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
その他	DGS，DGL，TOL_INF，SJIS，EUC，NONE

2.4　レーベル生成規則

制御文をアセンブラの命令に展開する際に，構造化アセンブラは分岐命令のレーベルを生成します。

構造化アセンブラが生成するレーベルは，”?Ldddd”となります。

ここで，dddd は，1 から始まる 10 進数で，ゼロ・サプレスし，左づめで出力します。したがって，”?Ldddd”形式のレーベルは記述しないでください。

2.5　サイズ指定

代入式，条件式の左辺または右辺に記述したシンボル，および switch 文の case シンボルのデータ・サイズを変更するためにサイズ指定を可能とします。

【記述形式】

（△サイズ指定文字△）

【機　能】

① サイズ文字が’B’,’b’の場合，データ・サイズをバイトに変更します。

【説　明】

① サイズ指定文字が誤っている場合は，エラーとなります。
② サイズ指定のできない代入式，条件式に記述した場合は，エラーとなります。
③ レジスタにサイズ指定をした場合は，同じデータ・サイズの指定のみ記述可能とします。ただし，データ・サイズの変更は行いません。データ・サイズが異なる場合は，エラーとなります。
④ ユーザ・シンボルに指定したい場合は，必ず指定したデータ・サイズに変更します。
⑤ 直接参照指定シンボル，間接参照指定シンボル，イミーディェト・データにサイズ指定を記述した場合は，サイズ指定を無視してデータ・サイズの変更は行いません。
⑥ サイズ指定にワードは指定できません。

2.6　データ・サイズ

構造化アセンブラでは，シンボルのデータ・サイズを認識します。これは生成する命令によりシンボルが異なるからです。ただし，構造化アセンブラでは，シンボル定義および定数の記述が実際にそれが正しいかどうかの判断は，アセンブラにまかせます。

構造化アセンブラで認識するデータ・サイズを以下に示します。

表 2-6　データサイズ

a	CY
b	ビットシンボル（[HL].βを除く） 本構造化アセンブラでは，ビット sfr および“α．β”で記述されるシンボルをビットシンボルとして認識します。 αには，バイト・ユーザ・シンボル，ワード・ユーザ・シンボル，バイト指定したユーザ・シンボル，sfr，定数，A，PSW，定数が記述可能です。 βには，バイト・ユーザ・シンボル，ワード・ユーザ・シンボル，定数が記述可能です。
c	[HL].β βには，バイト・ユーザ・シンボル，ワード・ユーザ・シンボル，定数が記述可能です。
d	バイト・ユーザ・シンボル
e	バイト指定したユーザシンボル，saddr と重なる sfr
f	A
g	バイト・レジスタ（A,R0,R1 を除く）
h	R0
i	R1
j	sfr
k	PSW
l	ワード・ユーザ・シンボル
m	saddrp と重なる sfrp
n	AX
o	ワード・レジスタ（AX,RP0 を除く）
p	RP0
q	sfrp
r	SP
s	直接参照指定シンボル “!addr”で記述されるシンボルです。 addr には，バイト・ユーザ・シンボル，ワード・ユーザ・シンボル，定数，$が記述可能です
t	間接参照指定シンボル [HL]，[HL+byte]で記述されるシンボル。 byte には，バイト・ユーザ・シンボル，定数，$が記述可能です。
u	特殊間接参照指定シンボル [DE]で記述されるシンボルです。
v	イミーディェイト・データ “#date”で記述されるシンボル。 date には，バイト・ユーザ・シンボル，ワード・ユーザ・シンボル，定数，$が記述可能です。

2.7　コメント

“;” 以降 LF の前までの文字列はコメント文とみなし，処理の対象とせずにそのまま２次ソース・ファイルに出力します。また，コメント文は１行中のどの記述位置にも記述することが可能です。

ただし，for～next 構文には，（ ）の中に “;” が式の区切りとして記述されるため，（ ）の中に記述された２個の “;” はコメント文の開始とはみなしません。

コメントに記述できる文字は “2.2 (1)　文字セット” で規定されるすべての文字が記述可能です。

不正文字の処理は，コメントおよびコメント文に記述された場合には処理は行われません。

2.8　ツール情報

本構造化アセンブラでは，ツール情報を出力します。

すでに，入力ソース・ファイル中に本構造化アセンブラで出力したツール情報が存在する場合，先頭の’ $’ を’ ;’ に置き換えます。

出力位置は，モジュール・ヘッダの後尾とします。モジュール・ヘッダに記述可能な文は，モジュール・ヘッダに記述可能なアセンブラ制御命令，コメント文，改行のみです。

【出力形式】

$TOL_INF□2FH，第２パラメータ，第３パラメータ，0FFFFH

2FH は，構造化アセンブラ・プリプロセッサが出力したツール情報であることをしましょす。

第２パラメータは，本プリプロセッサのバージョン番号を示します。

バージョン番号は 16 進数で出力するが，値の変換は行わず起動時に表示する 10 進数のイメージとします。

（例）

バージョン番号　1.00　→　100H

第３パラメータは，本プリプロセッサのエラー情報を示します。

0H　正常終了時

1H　フェイタル・エラー終了時

2H　ワーニング終了時

3H　フェイタル・エラーかつワーニング終了時

0FFFFH は，言語系情報を示します。本プリプロセッサでは固定値とします。

2.9　構造化アセンブラの展開

入力ソース・ファイルは構造化アセンブラによって，次のように展開されます。

表 2-7　構造化アセンブラの展開

入力ソース・プログラム・ファイル	2 次ソース・プログラム・ファイル
構造化アセンブラの制御文 構造化アセンブラの式の文	コメントとして出力されます
構造化アセンブラの擬似命令	出力されません
#INCLUDE	インクルードの内容を出力します
#IFDEF で偽になったソース	出力されません
コメント	コメントとして出力されます
その他の行	そのまま出力されます

第 3 章　制御文

この章では，制御文について例を挙げて説明します。

3.1　制御文の概要

制御文は，プログラムの制御の流れを構造的に記述するために用います。
制御文には，次に示すものがあります。

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

・条件ループ（DO～WHILE 系）

（4）	for～next	（増分指定の繰り返し）
（5）	while～endw	（処理前条件判定の繰り返し）
（6）	while_bit～endw	（処理前条件判定の繰り返し）
（7）	repeat～until	（処理後条件判定の繰り返し）
（8）	repeat～until_bit	（処理後条件判定の繰り返し）
（9）	break	（ループ・ブロックの抜け出し）
（10）	continue	（ループ・ブロックの繰り返し）
（11）	goto	（例外処理のための脱出）

3.2　制御文の文字

制御文の文字は，英大文字と英小文字では，基本的には生成する命令が異なります。例えば，if～endif と IF～ENDIF の間に記述する文のサイズによって，条件式の処理で生成する条件付き分岐命令で，直接分岐できない場合が生じます。

しかしながら，常に分岐するように命令を生成すると，プログラムのオブジェクト効率が悪くなる欠点があります。

その解決策として，ユーザに区別して記述してもらい，オブジェクト効率を向上させる方式を採用しました。オブジェクト効率に問題がなければ，英大文字で記述すれば，文のサイズによる変更は，行わなくても良いことになります。

制御文は条件付き分岐命令を生成しますので，相対アドレスが 128 バイト以内になるかどうかを指定してください。

制御文の“if”や“elseif”は，予約語です。これら，制御文の予約語の最初の文字が，大文字か小文字かで，構造化アセンブラは判断します。

IF，If … 大文字で始まっているので，大文字で記述したと判断されます。
if，iF … 小文字で始まっているので，小文字で記述したと判断されます。

大文字で記述した場合 … 条件付き分岐命令と BR 擬似命令の組合せで分岐します。
小文字で記述した場合 … 条件付き擬似命令で直接分岐します。

対になる制御文（if，else,endif など）は，大文字記述と小文字記述が混在していてもかまいません。すなわち，"IF～else～ENDIF" のように記述することが可能です。

3.3　ネスティング

制御文は，ネスティングさせることができます。ネスト・レベルは，全体で 31 レベルまで可能です。ただし，制御文の交差はできません。

（誤った記述例）

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

交差しているのでエラーとなります。

（正しい記述例）

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

WHILE 文の中に IF 文が正しくネストされています。

3.4　レジスタ指定

【記述形式】

（［△］［＝］［△］レジスタ名［△］）

【機　能】

① 比較条件式の直後にレジスタを指定した場合

左辺を指定レジスタに転送する命令後，指定レジスタと右辺の比較生成命令を生成します。

（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 制御文の後にレジスタを指定した場合

各比較条件式の命令生成において，左辺を指定レジスタに転送する命令を生成後，指定レジスタと右辺の比較命令を生成します。

（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 両方記述した場合

各比較条件式の直後のレジスタ指定を優先し，左辺を指定レジスタに転送する命令を生成後，指定レジスタと右辺の比較命令を生成します。

比較条件式の直後にレジスタ指定のない式については，制御文の後のレジスタ指定にしたがい，左辺を指定レジスタに転送する命令を生成後，指定レジスタと右辺の比較命令を生成します。

（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
```

【説　明】

① if 文，elseif 文，switc 文，for 文，while 文，until 文に使用可能です。ただし，条件式がビット条件式の場合，制御文に指定したレジスタは無視します。

② レジスタ名は“表 2-5　予約語シンボル”を参照してください。
sfr についても記述可能です。

③ for 文中の代入文についても比較条件式と同様の処理を行います。

3.5　制御文の機能

次頁より各制御文の機能を説明します。

使用例は，生成された命令に入力ソース・ファイルがコメント文として記述されています。

条件分岐　　if

（1）　if～elseif～else～endif

【記述形式】

```
[△] if [△] （条件式 1） [△] [ （レジスタ名） ]
        if 節
[△] elseif [△] （条件式 2） [△] [ （レジスタ名） ]
        elseif 節
[△] else
        else 節
[△] endif
```

【機　能】

① if～endif

条件式 1 が真ならば if 節を実行します。

if 節は，複数の行にわたってもかまいません。

② if～else～endif

条件式 1 が真ならば if 節を実行し，偽ならば，else 節を実行します。

if 節，else 節は，複数の行にわたってかまいません。

③ if～elseif～else～endif

elseif は，１つの if 文に対して複数書くことができます。

条件式 1 が真ならば，if 節を実行し，偽ならば，条件式 2 の判定を行います。

条件式 2 が真ならば，elseif 節を実行します。偽のとき，endif までにさらに elseif があれば，その条件判定をします。elseif がなければ，else 節を実行します。

if 節，elseif 節，else 節は，複数の行にわたってかまいません。

条件分岐　　if

【説　明】

① 条件式には，比較演算式，論理演算式，ビット条件式を記述します。レジスタ名が指定された場合には指定されたレジスタを使用して，条件判断を行います。
比較演算式，論理演算式については，「3.5　条件式」を参照してください。

② if～else～endif は，条件で二分岐する場合に記述します。

③ if～elseif～else～endif は，ある値の範囲を持って多分岐する場合に記述します。値に範囲をもたせることができる点が，switch 文と異なります。

④ elseif 文，else 文は省略可能で，elseif は複数記述できます。

【生成命令】

① if（条件式）の処理

(a)　条件式の条件判定の命令を生成します。

(b)　条件が成立しなかったときに分岐する，elseif 節や else 節への分岐命令を生成します。

② elseif（条件式）の処理

(a)　endif 文への分岐命令を生成します。

(b)　if 文で生成される分岐命令に対するレーベルを生成します。

(c)　条件式の条件判定の命令を生成します。

(d)　条件が成立しなかったときに分岐する，elseif 節や else 節への分岐命令を生成します。

③ else の処理

(a)　endif 文への分岐命令を生成します。

(b)　if 文，elseif 文で生成される分岐命令に対するレーベルを生成します。

④ endif の処理

(a)　if，elseif，else 文で生成される分岐命令に対するレーベルを生成します。

⑤ 補足説明

(a)　elseif_bit との混在記憶が可能です。

条件分岐　　if

【使用例】

① 小文字で記述した場合

```
        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 大文字で記述した場合

```
        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
```

条件分岐　　if_bit

(2)　if_bit～elseif_bit～else～endif

【記述形式】

```
[△] if_bit [△] （ビット条件式 1）
        if 節
[△] elseif_bit [△] （ビット条件式 2）
        elseif_bit 節
[△] else [△]
        else 節
[△] endif [△]
```

【機　能】

① if_bit～endif

ビット条件 1 が真ならば，if 節を実行します。

if 節は，複数の行にわたってかまいません。

② if_bit～else～endif

ビット条件 1 が真ならば，if 節を実行し，偽ならば，else 節を実行します。

if 節，else 節は，複数の行にわたってかまいません。

③ if_bit～elseif_bit～else～endif

ビット条件 1 が真ならば，if 節を実行し，偽ならば，ビット条件 2 の判定を行います。ビット条件 2 が真ならば，elseif_bit 節を実行します。偽のとき，endif までにさらに elseif_bit があれば，その条件判定をします。

elseif_bit がなければ，else 節を実行します。

if 節，elseif_bit 節，else 節は，複数の行にわたってかまいません。

⑤ 補足説明

elseif との混在記憶が可能です。

【説　明】

① ビット条件式 1，2 には，ビット条件式を記述します。

ビット条件式については，“3.5　条件式”を参照してください。

② if_bit～else～endif は，条件で二分岐する場合に記述します。

if_bit～elseif_bit～else～endif は，複数のビット・シンボルをチェックして多分岐する場合に記述します。

③ elseif_bit 文および else 文は省略可能で，

elseif_bit は，複数記述可能です。

条件分岐　　if_bit

【生成命令】

① if_bit（ビット条件）の処理

（a）ビット条件の真偽判定命令を生成します。

② elseif_bit（ビット条件）の処理

（a） endif 文への分岐命令を生成します。

（b） if_bit 文で生成される分岐命令に対するレーベルを生成します。

（c） ビット条件の真偽判定命令を生成します。

③ else の処理

（a） endif 文への分岐命令を生成します。

（b） if_bit 文，elseif_bit 文で生成される分岐命令に対するレーベルを生成します。

④ endif の処理

（a）if_bit 文，elseif_bit 文，else 文で生成される分岐命令に対するレーベルを生成します。

条件分岐　　if_bit

【使用例】

① 小文字で記述した場合

```
        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 大文字で記述した場合

```
        BF      TRFG.0,$?L6     ;IF_BIT(!TRFG.0)
        BR      ?L7
?L6:
        SET1    PRTYFLG.3       ;PRTYFLG.3=1
        BR      ?L8
?L7:                            ;ELSEIF_BIT(PGF.0)
        BT      PGF.0,$?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.0,$?L11     ;CY=PFG.0
        SET1    CY
        BR      ?L12
?L11:
        CLR1    CY
?L12:
        CLR1    BUSYFG.2        ;BUSYFG.2=0
?L8:                            ;ENDIF
```

条件分岐　switch

（3）　switch～case～default～ends

【記述形式】

```
[△] switch [△] ( [△] case シンボル [△] ) [△] [ (指定レジスタ) ]
        [△] case [△] 定数 :
                文__1
[       [△] case [△] 定数 :
                文__2    ]
        [△]  [default : ]
                文__N
[△] ends
```

【機　能】

① case シンボルの値が case 定数と一致した場合は指定された文を実行します。

② case シンボルの値がどの case 定数とも一致せず，かつ default が記述されていれば，default 文を実行します。

③ 通常は，switch ブロックを抜けるために break 文を記述しなければなりません。

【説　明】

① case シンボルに記述可能なものは，各対象デバイスのアセンブリ言語に依存します。

② break 文を記述しないと，次の case 文の比較命令を実行します。
C言語とは case 処理後の動作が異なるので注意が必要です。C言語と同様に機能させるためには分岐命令を記述してください。

③ 定数として，2 進数，8 進数，10 進数，16 進数，文字定数が記述できます。
ただし，構造化アセンブラは定数も文字列として認識するので，アセンブラで定数と解釈できるものでなければなりません。

④ レジスタ指定した場合のみ，case シンボルを指定レジスタに転送します。

条件分岐　　switch

【生成命令】

① switch 文の処理

(a)　レジスタを指定しない場合には，case シンボルを判断して，必要であれば A または AX への転送命令を生成します。

(b)　レジスタ指定した場合には，case シンボルを指定レジスタに転送します。
ただし，比較命令が生成不可能な場合には，エラーとなります。
詳細については，“表 3-1　switch 文の生成命令”を参照してください。

② case 文の処理

(a)　他の case 文からの分岐処理のためにレーベルを生成します。

(b)　CMP または CMPW を生成し，指定した定数と一致しなければ，他の case 文，default 文，ends 文のいずれかに分岐する命令を生成します。
?LTRUE　：指定した定数と一致した場合の分岐先のレーベル
?LFALSE：指定した定数と一致しなかった場合の分岐先のレーベル

・case 文が英小文字でかつ switch 文にレジスタ指定がない場合

```
CMP(W)      case シンボル,#case 定数
BNZ         $?LFALSE
```

・case 文が英小文字でかつ switch 文にレジスタ指定をした場合

```
CAMP(W)     指定レジスタ,#case 定数
BNZ         $?LFALSE
```

・case 文が英大文字でかつ switch 文にレジスタ指定がない場合

```
CAMP(W)     case シンボル,#case 定数
BZ          $?LTRUE
BR          ?LFALSE
?LTRUE
```

・case 文が英大文字でかつ switch 文にレジスタ指定をした場合

```
CAMP(W)     指定レジスタ,#case 定数
BZ          $?LTRUE
BR          ?LFALSE
?LTRUE
```

③ default 文の処理

(a)　case 文からの分岐命令に対するレーベルを生成します。

④ ends 文の処理

(a)　case 文または break 文からの分岐命令に対するレーベルを生成します。

表 3-1　switch 文の生成命令

CASE シンボル		レジスタ	レジスタ指定あり												
		指定なし	a	b	f	g	h	i	j	k	n	o	p	q	r
a	CY														
b	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ														
c	[HL].β														
d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ	*3			*1						*2				
e	ﾊﾞｲﾄ･ﾃﾞｰﾀ	*3			*1										
f	A	*3													
g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ	*1			*1										
h	R0	*1			*1										
i	R1														
j	sfr	*1			*1										
k	PSW	*1			*1										
l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ				*1						*2				
m	ﾜｰﾄﾞ･ﾃﾞｰﾀ	*2									*2				
n	AX	*3													
o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ	*2									*2				
p	RP0														
q	sfrp														
r	SP	*2									*2				
s	直接参照ｼﾝﾎﾞﾙ	*1			*1										
t	間接参照ｼﾝﾎﾞﾙ	*1			*1										
u	[DE]	*1			*1										
v	ｲﾐﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ	*1			*1						*2				

*1：MOV 命令を生成します。

*2：MOVW 命令を生成します。

*3：転送命令は生成しません。

空欄はエラーを示します。

条件分岐　switch

【使用例】

① 小文字で記述した場合

```
        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 大文字で記述した場合

```
        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
```

条件ループ　　for

(4)　for～next

【記述形式】

```
[△] for [△] （[式 1] ; [式 2] ; [式 3] ） [△] [（レジスタ指定）]
        命令群
[△] next
```

【機　能】

式 1 で初期値を設定し，式 2 の条件式が成立している間，文および式 3 を実行します。
通常，式 3 はインクリメントまたはデクリメントなどを記述します。
次に示すものと等価の意味になります。

```
式 1
while（式 2）
  命令群
  式 3
endw
```

【説　明】

① 命令生成においては上記の等価の展開とはならないので注意が必要です。
② 式 1，式 2，式 3 には以下に示す内容を記述します。
- 式 1　…　初期値の設定（代入式）
- 式 2　…　条件式
- 式 3　…　インクリメントまたはデクリメントなどの代入式

③ 式 1，式 3 には代入演算子，交換文が記述できますが，その場合は変換結果に注意して見直しされることをお奨めします。
④ 式 1，式 2，式 3 は省略できます。ただし，式 2 を省略した場合は無限ループとなります。
⑤ 条件式には“forever”も記述できます。
⑥ 式 2，式 3 は，for～next を制御するものであるため，その内容を実行文で変更してはいけません。変更すると誤動作の原因となります。

条件ループ　for

【生成命令】

① for（式 1 ; 式 2 ; 式 3）文の処理

(a)　式 1 の命令を生成します。レジスタ名が指定されていると，代入や比較に，そのレジスタを使用します。

(b)　式 2 の条件を判定する文への分岐命令を生成します。

(c)　next 文で生成される分岐命令に対するレーベルを生成します。

(d)　(2) で生成した分岐命令に対するレーベルを生成します。

(e)　式 2 の条件判定命令を生成します。

② next 文の処理

(a)　for 文処理 (3) で生成したレーベルへの分岐命令を生成します。

(b)　for ブロックを抜けるための分岐命令に対するレーベルを生成します。

(c)　式 3 の代入式の命令を生成します。

③ 補足説明

(a)　for～next 文をより有効に使用するために，次の方法をお奨めします。

1. 式 1，式 3 に使用する制御変数には，レジスタよりも saddr を記述してください。
2. レジスタ指定の場合は，A または AX を指定してください。
3. 256 回以上繰り返す場合は，for 文をネストし，制御変数として saddr を 2 つ使用してください。

備考　上記の方法を推奨する理由は，式 2 が条件式であり生成命令として CMP または CMPW を出力するため，そのオペランドとして記述されるシンボルに制限があるからです。

条件ループ　　for

【使用例】

① 小文字で記述した場合

```
        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 大文字で記述した場合

```
        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
```

条件ループ　while

(5)　while～endw

【記述形式】

```
[△] while [△] （条件式） [△] [（レジスタ指定）]
        命令群
[△] endw
```

【機　能】

① 条件式が真のあいだ，命令群を繰り返し実行します。

【説　明】

① 条件式には，比較演算式，論理演算式，ビット条件式，“forever”が記述できます。
forever を記述した場合は，無限ループになります。

② レジスタ名には，（条件式）で記述した比較演算式，論理演算式で使用するレジスタを指定します。

③ 命令群を実行する前に条件式の評価を行うので，最初に条件式が偽の場合は１度も命令群は実行されません。

【生成命令】

① while（条件式）文の処理

(a)　endw で生成される分岐命令に対するレーベルを生成します。

(b)　条件式の条件判定命令を生成します。レジスタ名が指定されていれば，レジスタを使用して条件判定命令を生成します。

(c)　条件の判定結果が偽のときに while ブロックから抜けるための分岐命令を生成します。

② endw

(a)　繰り返しのための分岐命令を生成します。

(b)　while ブロックから抜けるための分岐命令に対するレーベルを生成します。

条件ループ　　while

【使用例】

① 小文字で記述した場合

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

② 大文字で記述した場合

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

条件ループ　　while_bit

(6)　while_bit～endw

【記述形式】

```
[△] while_bit [△] （ビット条件）
        命令群
[△] endw
```

【機　能】

① ビット条件が真のあいだ，命令群を実行します。

【説　明】

① 命令群を実行する前にビット条件の評価をするので，最初にビット条件が偽の場合は 1 度も命令群は実行されません。

【生成命令】

① while_bit（ビット条件）文の処理

(a)　endw で生成される分岐命令に対するレーベルを生成します。

(b)　ビット条件の真偽判定の命令を生成します。

(c)　ビット条件の判定結果が偽のときに while_bit～endw ブロックから抜けるための分岐命令を生成します。

② endw の処理

(a)　繰り返しのための分岐命令を生成します。

(b)　while_bit ブロックから抜けるための分岐命令に対するレーベルを生成します。

条件ループ　　while_bit

【使用例】

① 小文字で記述した場合

```
?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 大文字で記述した場合

```
?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
```

条件ループ　until

(7)　repeat～until

【記述形式】

```
[△] repeat
        命令群
[△] until [△] （条件式） [△] [ （レジスタ指定） ]
```

【機　能】

① 条件式が偽のあいだ，命令群を繰り返し実行します。

【説　明】

① 条件式には，比較演算式，論理演算式，ビット条件式，“forever”が記述できます。
forever を記述した場合は，無限ループになります。

② レジスタ名には，（条件式）で記述した比較演算式，論理演算式で使用するレジスタを指定します。

③ 命令群を実行したあとに，条件式を評価します。ですから，最初に条件式が真の場合でも，命令群は 1 度実行されます。

【生成命令】

① repeat 文の処理

(a)　until で生成される分岐命令に対するレーベルを生成します。

② until（条件式）文の処理

(a)　条件式の条件判定命令を生成します。

(b)　条件式が偽ならば，repeat～until あいだの命令群を実行するために repeat で生成したレーベルの分岐命令を生成します。条件式が真ならば，repeat ブロックから抜けます。

条件ループ　until

【使用例】

① 小文字で記述した場合

```
?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 大文字で記述した場合

```
?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:
```

条件ループ　　until_bit

(8)　until_bit

【記述形式】

```
[△] repeat
        命令群
[△] until_bit [△] （ビット条件式）
```

【機　能】

① ビット条件が偽のあいだ，命令群を繰り返し実行します。

【説　明】

① 命令群を実行した後にビット条件の評価します。ですから，最初にビット条件が真でも，命令群は１度実行されます。

【生成命令】

① repeat の処理

(a)　until_bit で生成される分岐命令に対するレーベルを生成します。

② until_bit（ビット条件）の処理

(a)　条件式が偽ならば，repeat～until_bit 間の命令群を実行するために repeat で生成したレーベルの分岐命令を生成します。条件式が真ならば，repeat ブロックから抜けます。

【使用例】

① 小文字で記述した場合

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

② 大文字で記述した場合

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

条件ループ　　break

(9)　break

【記述形式】

[△] break

【機　能】

囲んでいる最も内側の while，repeat，for，switch ブロックの実行を終了させます。

【説　明】

while，while_bit，repeat～until，repeat～until_bit，for，switch 文以外に記述した場合は，エラーとなります。

【生成命令】

while，repeat，for，switch ブロックを抜けるための，無条件分岐命令を生成します。

```
BR      ?Lxxxx
```

【使用例】

```
?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
```

（10）　continue

【記述形式】

[△] continue

【機　能】

囲んでいる最も内側の while，while_bit，repaet～until，repaet～until_bit，for 文内の continue 以降の処理をスキップし，条件判断の前に無条件分岐します。

【説　明】

① 各ブロックの途中で，ブロック内の以降の処理をスキップして，次のループの繰り返しを行う場合に使用します。

② while，while_bit，repaet～until，repaet～until_bit，for 文以外の文に記述した場合は，エラーとなります。

【生成命令】

while，while_bit，repaet～until，repaet～until_bit，for ブロックのループ繰り返しのためのレーベルへの分岐命令を生成します。

```
BR      ?Lxxxx
```

【使用例】

```
?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
```

（11）　goto

【記述形式】

[△] goto△レーベル

【機　能】

無条件にレーベルへ分岐します。

【説　明】

① goto 文は，エラー処理等のプログラムで，直ちにエラー処理を行う必要がある場合や，エラーが複数の箇所で起こり得て，処理が共通の場合に記述します。

② レーベルには，アセンブリ言語のレーベル欄に記述したシンボルを指定します。

【生成命令】

① 次の命令を生成します。

```
BR      レーベル
```

② goto 文のレーベルは，構造化アセンブラが自動生成するレーベルではありません。構造化アセンブラでは，分岐先のレーベルの存在チェックを行いませんので注意してください。

【使用例】

```
?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　条件式

条件式は，制御文で条件を設定するために使用します。
条件式には，次に示すものがあります。

・比較条件式　……　第１項と第２項の値を比較し，真偽を判定します。
・ビット条件式　……　ビット・シンボルにより，フラグのオン，オフを判定します。
・論理演算　……　条件を複合させる場合に，条件式の論理演算を行います。

表 3-2　比較条件式

比較条件式		記述形式	機　　能
(1)	Equal	α == β	α＝βのとき真，α≠βのとき偽
(2)	NotEqual	α != β	α≠βのとき真，α＝βのとき偽
(3)	LessThan	α < β	α＜βのとき真，α＞＝βのとき偽
(4)	GreaterThan	α > β	α＞βのとき真，α＜＝βのとき偽
(5)	GreaterEqual	α >=β	α＞＝βのとき真，α＜βのとき偽
(6)	LessEqual	α <= β	α＜＝βのとき真，α＞βのとき偽

表 3-3　ビット条件式

ビット条件式		記述形式	機　　能
(7)	正論理（ビット）	ビット・シンボル	指定されたビットが１のとき真
(8)	負論理（ビット）	!ビット・シンボル	指定されたビットが０のとき真

表 3-4　論理演算

論理演算		記述形式	機　　能
(9)	論理積	条件式１ && 条件式２	条件式１，条件式２が共に真であれば真
(10)	論理輪	条件式１|| 条件式２	条件式１または，条件式２が真であれば真

比較演算にはうしろに（γ）を指定すると，直接比較できないα，βを比較できます。
γは比較のために壊されるレジスタを指定します。

3.6.1　比較条件式

各比較条件式の説明では，判定結果が真の場合は分岐先のレーベルとして？LTRUE を使用し，偽の場合は分岐先のレーベルとして?LFALSE を使用します。

レジスタ指定の記述形式については“3.4　レジスタ指定”を参照してください。

構造化アセンブラでは，比較条件式の左辺および右辺に記述したシンボルが，アセンブラ言語のオペランドとして正しい記述であるかの判断は行いません。ただし，“2.6　データ・サイズ”に基づき，命令が生成可能かどうかの判断は行います。また，レジスタ指定においても指定されたレジスタで命令が生成可能か判断を行います。

判断結果エラーとなった場合は，エラー・メッセージを出力します。

詳細については，各生成命令を参照してください。

次に，各比較条件式を説明します。

表 3-5　比較命令の生成命令

			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	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ																						
	c	[HL].β																						
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						*1
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ																						
	n	AX																						*2
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照ｼﾝﾎﾞﾙ																						
	t	間接参照ｼﾝﾎﾞﾙ																						
	u	[DE]																						
	v	ｲﾐｰﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ																						

*1：CMP を生成します。

*2：CMPW を生成します。

空欄はエラーを示します。

比較条件式　　Equal（==）

（1）　Equal　（==）

【記述形式】

［△］［サイズ指定］α［△］==［△］［サイズ指定］［△］β［△］［（レジスタ指定）］

【機　能】

① レジスタ指定がない場合

2 項αとβの内容が等しければ真，等しくなければ偽とします。

② レジスタ指定がある場合

指定レジスタにαの内容を転送し，指定レジスタの内容とβの内容が等しければ真，等しくなければ，偽とします。

【説　明】

① レジスタ指定がない場合

α，βは CMP または CMPW で記述可能なものを記述してください。

② レジスタ指定がある場合

αは MOV または MOVW で記述可能なものを記述してください。

βは CMP または CMPW で記述可能なものを記述してください。

【生成命令】

① 制御文が英小文字でかつレジスタ指定がない場合

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

② 制御文が英小文字でかつレジスタ指定をした場合

```
MOV(W)      指定レジスタ,α
CMP(W))     指定レジスタ,β
BNZ         $?LFALSE
```

比較条件式　　Equal (==)

③ 制御文が英大文字でかつレジスタ指定がない場合

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

④ 制御文が英大文字でレジスタ指定をした場合

```
MOV(W)      指定レジスタ,α
CMP(W)      指定レジスタ,β
BZ          $?LTRUE
BR          ?LFALSE
?LTRUE:
```

α，β の組み合わせの詳細については，“表 3-5　比較命令の生成命令”を参照してください。指定レジスタは，α に読みかえてください。MOV の生成命令は“第 4 章 (1)　代入”を参照してください。

【使用例】

① 制御文が英小文字でかつレジスタ指定がない場合

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

② 制御文が英小文字でかつレジスタ指定をした場合

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

③ 制御文が英大文字でかつレジスタ指定がない場合

```
        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 制御文が英大文字でレジスタ指定をした場合

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

比較条件式　　NotEqual（!=）

(2)　NotEqual　(!=)

【記述形式】

［△］［サイズ指定］［△］α［△］!=［△］［サイズ指定］［△］β［△］［（レジスタ指定）］

【機　能】

① レジスタ指定がない場合

2項αとβの内容が等しくなければ真，等しければ偽とします。

② レジスタ指定がある場合

指定レジスタにαの内容を転送し，指定レジスタとβの内容が等しくなければ真，等しければ偽とします。

【説　明】

① レジスタ指定がない場合

α，βはCMPまたはCMPWで記述可能なものを記述してください。

② レジスタ指定がある場合

αはMOVまたはMOVWで記述可能なものを記述してください。

βはCMPまたはCMPWで記述可能なものを記述してください。

【生成命令】

① 制御文が英小文字でかつレジスタ指定がない場合

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

② 制御文が英小文字でかつレジスタ指定をした場合

```
MOV(W)      指定レジスタ,α
CMP(W)      指定レジスタ,β
BZ          $?LFALSE
```

比較条件式　　NotEqual (!=)

③ 制御文が英大文字でかつレジスタ指定がない場合

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

④ 制御文が英大文字でかつレジスタ指定をした場合

```
    MOV(W)      指定レジスタ,α
    CMP(W)      指定レジスタ,β
    BNZ         $?LTRUE
    BR          ?LFALSE
  ?LTRUE:
```

α，β の組み合わせの詳細については，“表 3-5　比較命令の生成命令”を参照してください。指定レジスタは，α に読みかえてください。MOV の生成命令は“第 4 章 (1)　代入”を参照してください。

【使用例】

① 制御文が英小文字でかつレジスタ指定がない場合

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

② 制御文が英小文字でかつレジスタ指定をした場合

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

比較条件式　　NotEqual (!=)

③ 制御文が英大文字でかつレジスタ指定がない場合

```
        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 制御文が英大文字でかつレジスタ指定をした場合

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

比較条件式　　LessThan（<）

（3）　LessThan　（<）

【記述形式】

［△］［サイズ指定］［△］α［△］<［△］［サイズ指定］［△］β［△］［（レジスタ指定）］

【機　能】

① レジスタ指定がない場合

αの内容がβの内容より小さければ真，それ以外であれば偽とします。

② レジスタ指定がある場合

指定レジスタにαの内容を転送し，指定レジスタの内容がβの内容より小さければ真，それ以外であれば偽とします。

【説　明】

① レジスタ指定がない場合

α，βはCMPまたはCMPWで記述可能なものを記述してください。

② レジスタ指定がある場合

αはMOVまたはMOVWで記述可能なものを記述してください。

βはCMPまたはCMPWで記述可能なものを記述してください。

【生成命令】

① 制御文が英小文字でかつレジスタ指定がない場合

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

② 制御文が英小文字でかつレジスタ指定をした場合

```
MOV(W)      指定レジスタ,α
CMP(W)      指定レジスタ,β
BNC         $?LFALSE
```

③ 制御文が英大文字でかつレジスタ指定がない場合

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

比較条件式　　LessThan (<)

④ 制御文が大文字でレジスタ指定をした場合

```
MOV(W)      指定レジスタ,α
CMP(W)      指定レジスタ,β
BC          $?LTRUE
BR          ?LFALSE
?LTRUE:
```

α，βの組み合わせの詳細については，“表 3-5　比較命令の生成命令”を参照してください。指定レジスタは，αに読みかえてください。MOV の生成命令は“第 4 章 (1)　代入”を参照してください。

【使用例】

① 制御文が英小文字でかつレジスタ指定がない場合

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

② 制御文が英小文字でかつレジスタ指定をした場合

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

③ 制御文が英大文字でかつレジスタ指定がない場合

```
        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
```

比較条件式　　LessThan (<)

④ 制御文が大文字でレジスタ指定をした場合

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

比較条件式　　GreaterThan（>）

(4)　GreaterThan　（>）

【記述形式】

[△] [サイズ指定] [△] α [△] > [△] [サイズ指定] [△] β [△] [（レジスタ指定）]

【機　能】

① レジスタ指定がない場合

αの内容がβの内容より大きければ真，それ以外であれば偽とします。

② レジスタ指定がある場合

指定レジスタにαの内容を転送し，指定レジスタの内容がβの内容より大きければ真，それ以外であれば偽とします。

【説　明】

① レジスタ指定がない場合

α，βは CMP または CMPW で記述可能なものを記述してください。

② レジスタ指定がある場合

αは MOV または MOVW で記述可能なものを記述してください。

βは CMP または CMPW で記述可能なものを記述してください。

【生成命令】

① 制御文が英小文字でかつレジスタ指定がない場合

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

② 制御文が英小文字でかつレジスタ指定をした場合

```
MOV(W)      指定レジスタ,α
CMP(W)      指定レジスタ,β
BZ          $?LFALSE
BC          $?LFALSE
```

比較条件式　　GreaterThan（>）

③ 制御文が英大文字でかつレジスタ指定がない場合

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

④ 制御文が英大文字でかつレジスタ指定をした場合

```
MOV(W)      指定レジスタ,α
CMP(W)      指定レジスタ,β
BZ      $$+4
BNC         $?LTRUE
BR          ?LFALSE
?LTRUE:
```

α，βの組み合わせの詳細については，“表 3-5　比較命令の生成命令”を参照してください。指定レジスタは，αに読みかえてください。MOV の生成命令は“第 4 章（1）　代入”を参照してください。

【使用例】

① 制御文が英小文字でかつレジスタ指定がない場合

```
        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 制御文が英小文字でかつレジスタ指定をした場合

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

③ 制御文が英大文字でかつレジスタ指定がない場合

```
        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 制御文が英大文字でかつレジスタ指定をした場合

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

比較条件式　　GreaterEqual（>=）

（5）　GreaterEqual　（>=）

【記述形式】

［△］［サイズ指定］［△］α［△］>=［△］［サイズ指定］［△］β［△］［（レジスタ指定）］

【機　能】

① レジスタ指定がない場合

αの内容がβの内容より大きいか等しければ真，小さければ偽とします。

② レジスタ指定がある場合

指定レジスタにαの内容を転送し，指定レジスタの内容がβの内容より大きいか等しければ真，小さければ偽とします。

【説　明】

① レジスタ指定がない場合

α，βはCMPまたはCMPWで記述可能なものを記述してください。

② レジスタ指定がある場合

αはMOVまたはMOVWで記述可能なものを記述してください。

βはCMPまたはCMPWで記述可能なものを記述してください。

【生成命令】

① 制御文が英小文字でかつレジスタ指定がない場合

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

② 制御文が英小文字でかつレジスタ指定をした場合

```
MOV(W)      指定レジスタ,α
CMP(W)      指定レジスタ,β
BC          $?LFALSE
```

③ 制御文が英大文字でかつレジスタ指定がない場合

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

比較条件式　　GreaterEqual (>=)

④ 制御文が英大文字でかつレジスタ指定をした場合

```
MOV(W)      指定レジスタ,α
CMP(W)      指定レジスタ,β
BNC         $?LTRUE
BR          ?LFALSE
?LTRUE:
```

α, βの組み合わせの詳細については, “表 3-5　比較命令の生成命令”を参照してください。指定レジスタは, αに読みかえてください。MOV の生成命令は“第 4 章 (1)　代入”を参照してください。

【使用例】

① 制御文が英小文字でかつレジスタ指定がない場合

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

② 制御文が英小文字でかつレジスタ指定をした場合

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

③ 制御文が英大文字でかつレジスタ指定がない場合

```
        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
```

④ 制御文が英大文字でかつレジスタ指定をした場合

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

比較条件式　　LessEqual（<=）

(6)　LessEqual　（<=）

【記述形式】

［△］［サイズ指定］［△］α［△］<=［△］［サイズ指定］［△］β［△］［（レジスタ指定）］

【機　能】

① レジスタ指定がない場合

αの内容がβの内容より小さいか等しければ真，大きければ偽とします。

② レジスタ指定がある場合

指定レジスタにαの内容を転送し，指定レジスタの内容がβの内容より小さいか等しければ真，大きければ偽とします。

【説　明】

① レジスタ指定がない場合

α，βはCMPまたはCMPWで記述可能なものを記述してください。

② レジスタ指定がある場合

αはMOVまたはMOVWで記述可能なものを記述してください。

βはCMPまたはCMPWで記述可能なものを記述してください。

【生成命令】

① 制御文が英小文字でかつレジスタ指定がない場合

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

② 制御文が英小文字でかつレジスタ指定をした場合

```
MOV(W)      指定レジスタ,α
CMP(W)      指定レジスタ,β
BZ          $$+4
BNC         $?LFALSE
```

比較条件式　　LessEqual（<=）

③ 制御文が英大文字でかつレジスタ指定がない場合

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

④ 制御文が英大文字でかつレジスタ指定をした場合

```
    MOV(W)      指定レジスタ,α
    CMP(W)      指定レジスタ,β
    BZ          $?LTRUE
    BC          $?LTRUE
    BR          ?LFALSE
  ?LTRUE:
```

α，βの組み合わせの詳細については，“表 3-5　比較命令の生成命令”を参照してください。指定レジスタは，αに読みかえてください。MOV の生成命令は“第 4 章（1）　代入”を参照してください。

【使用例】

① 制御文が英小文字でかつレジスタ指定がない場合

```
        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 制御文が英小文字でかつレジスタ指定をした場合

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

③ 制御文が英大文字でかつレジスタ指定がない場合

```
        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 制御文が英大文字でかつレジスタ指定をした場合

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

比較条件式　　FOREVER（forever）

(7)　FOREVER　（forever）

【記述形式】

[△] forever [△]

【機　能】

比較命令を生成せずに，ループ文を永久ループとします。

【説　明】

ループ文（for 文，while 文，until 文）の条件式に記述可能です。

【使用例】

① for 文の場合

```
        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 文の場合

```
?L4:                            ;while(foever)
        BF      foever,$?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
```

③ repeat 文の場合

```
?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　ビット条件式

各ビット条件式の説明では，判定結果が真の場合は分岐先のレーベルとして？LTRUE を使用し，偽の場合は分岐先のレーベルとして?LFALSE を使用します。

構造化アセンブラでは，ビット条件式の記述がアセンブラ言語のオペランドとして正しい記述であるかの判断は行いません。ただし，“2.6　データ・サイズ”に基づいての判断は行います。

また，‘Z’についても，ビットシンボルとして処理を行います。

構造化アセンブラでは，アセンブラの擬似命令（EQU）で，ビット・シンボルを定義済みのものであるか否かはチェックしません。ただし，ユーザ・シンボルもビットシンボルとして処理を行います。

判断結果エラーとなった場合は，エラー・メッセージを出力します。

詳細については，各生成命令を参照してください。

次に，各ビット条件式を説明します。

ビット条件式　　正論理（ビット）

(1)　ビット・シンボル

【記述形式】

［△］ビットシンボル［△］

【機　能】

ビット・シンボルの内容が１であれば真，０であれば偽とします。

ビット・シンボルを条件式として記述できるのは，次の制御文です。

if	if_bit
elseif	elseif_bit
while	while_bit
until	until_bit

【生成命令】

① 制御文が英小文字で CY を記述した場合

```
BNC     $?LFALSE
```

② 制御文が英小文字で Z を記述した場合

```
BNZ     $?LFALSE
```

③ 制御文が英小文字でビット・シンボルを記述した場合

```
BF      ビット・シンボル,$?LFALSE
```

④ 制御文が英大子文字で CY を記述した場合

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

⑤ 制御文が英大文字で Z を記述した場合

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

⑥ 制御文が英大文字でビット・シンボルを記述した場合

```
    BT      ビット・シンボル,$?LTRUE
    BR      ?LFALSE
  ?LTRUE:
```

【使用例】

① 制御文が小文字の場合

```
            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
```

② 制御文が大文字の場合

```
        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
```

ビット条件式　　負論理（ビット）

(2)　!ビット・シンボル

【記述形式】

[△] !ビット・シンボル [△]

【機　能】

ビット・シンボルの内容が 0 であれば真，1 であれば偽とします。

ビット・シンボルを条件式として記述できるのは，次の制御文です。

if	if_bit
elseif	elseif_bit
while	while_bit
until	until_bit

【生成命令】

① 制御文が英小文字で CY を記述した場合

```
BC      $?LFALSE
```

② 制御文が英小文字で Z を記述した場合

```
BZ      $?LFALSE
```

③ 制御文が英小文字でビット・シンボルを記述した場合

```
BT      ビット・シンボル,$?LFALSE
```

④ 制御文が英大子文字で CY を記述した場合

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

⑤ 制御文が英大文字で Z を記述した場合

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

⑥ 制御文が英大文字でビット・シンボルを記述した場合

```
BF      ビット・シンボル,$?LTRUE
BR      ?LFALSE
?LTRUE:
```

【使用例】

① 制御文が小文字の場合

```
        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
```

② 制御文が大文字の場合

```
        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　論理演算

各条件式の説明では，判定結果が真の場合は分岐先のレーベルとして？LTRUE を使用し，偽の場合は分岐先のレーベルとして?LFALSE を使用します。

条件式中で，二つの比較条件式またはビット条件式の真／偽に対して，論理積（&&）または論理積（||）をとることができます。

条件式には，論理演算子を最大 16 個まで記述できます。

これによって，ある条件式とある条件式を，同時にみたす場合の処理や，どちらかの条件を満たせば処理を行う場合の表現を記述することができます。

構造化アセンブラでは，論理演算子の優先順位を見て分岐命令を生成します。

【記述例】

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

次に，論理演算を説明します。

論理演算　　論理積（&&）

（1）　論理積　（&&）

【記述形式】

条件式 1［△］&&［△］条件式 2

【機　能】

条件式 1 と条件式 2 の論理積をとります。条件式 1，条件式 2 共に真であれば真，それ以外は偽となります。2 つの条件が同時に成立した場合の処理を記述します。

また，制御文が英小文字で記述された場合と，英大文字で記述された場合とでは出力する命令が異なります。

（ ）があれば，（ ）内を優先して評価する命令を生成します。

【生成命令】

① 制御文が英小文字で記述された場合

表 3-6　論理積の生成命令（英小文字制御文）

条件式	生成命令
α == β &&	CMP(W)　α,β BNZ　$?LFALSE
α != β &&	CMP(W)　α,β BZ　$?LFALSE
α < β &&	CMP(W)　α,β BNC　$?LFALSE
α > β &&	CMP(W)　α,β BZ　$?LFALSE BC　$?LFALSE
α >= β &&	CMP(W)　α,β BC　$?LFALSE
α <= β &&	CMP(W)　α,β BZ　$$+4 BNZ　$?LFALSE
ビット・シンボル &&	BF　ビット・シンボル,$?LFALSE
CY &&	BNC　$?LFALSE
Z &&	BNZ　$?LFALSE
!ビット・シンボル &&	BT　ビット・シンボル,$?LFALSE
!CY &&	BC　$?LFALSE
!Z &&	BZ　$?LFALSE

② 制御文が英大文字で記述された場合

表 3-7　論理積の生成命令（英大文字制御文）

条件式	生成命令
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 $?LTRUE BR ?LFALSE ?LTRUE:
ビット・シンボル &&	BT ビット・シンボル,$?LTRUE BR ?LFALSE ?LTRUE:
CY &&	BC $?LTRUE BR ?LFALSE ?LTRUE:
Z &&	BZ $?LTRUE BR ?LFALSE ?LTRUE:
!ビット・シンボル &&	BF ビット・シンボル,$?LTRUE BR ?LFALSE ?LTRUE:
!CY &&	BNC $?LTRUE BR ?LFALSE ?LTRUE:
!Z &&	BNZ $?LTRUE BR ?LFALSE ?LTRUE:

論理演算　　論理積（&&）

【使用例】

① 制御文が英小文字で記述された場合

```
        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 制御文が英大文字で記述された場合

```
        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
```

（2） 論理和　（||）

【記述形式】

条件式 1［△］||［△］条件式 2

【機　能】

条件式 1 と条件式 2 の論理和をとります。条件式 1，条件式 2 のいずれかが真であれば真，両方とも偽ならば偽となります。2 つの条件のいずれかが成立した場合の処理を記述します。

（ ）があれば，（ ）内を優先して評価する命令を生成します。

【生成命令】

表 3-8　論理和の生成命令

条件式	生成命令
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　$?LFALSE
ビット・シンボル ||	BT　ビット・シンボル,$?LFALSE
CY ||	BC　$?LFALSE
Z ||	BZ　$?LFALSE
!ビット・シンボル ||	BF　ビット・シンボル,$?LFALSE
!CY ||	BNC　$?LFALSE
!Z ||	BNZ　$?LFALSE

論理演算　　**論理和（||）**

【使用例】

```
        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
```

第４章　式の文

式の文は，代入や演算を行うものです。
式の文には，次に示すものがあります。

・代入文 ・・・・・・・・・・ 第２項を第１項に代入します。
・カウント文 ・・・・・・ 項の値に１加算／減算します。
・交換文 ・・・・・・・・・・ 第１項の値と第２項の値を交換します。
・ビット操作文 ・・・・ 項の値をセット（1）／リセット（0）します。

表 4-1　代入文

代入文		記述形式	機能
(1)	代入	α = β	α←β
	代入(レジスタ指定)	α = β (γ)	(γ)←β　α←(γ)
	連続代入	α 1 =…= α n =β	α 1←β,…, α n←β
	連続代入(レジスタ指定)	α 1 =…= α n = β(γ)	γ←β, α 1←γ,…, α n←γ
(2)	加算代入	α += β	α←α+β
	加算代入(レジスタ指定)	α += β (ﾚｼﾞｽﾀ)	γ←α, γ←γ+β, α←γ
	加算代入(レジスタ指定)	α += β,CY	α←α+β,CY
	加算代入(レジスタ指定)	α += β,CY (ﾚｼﾞｽﾀ)	γ←α, γ←γ+β,CY, α←γ
(3)	減算代入	α -= β	α←α-β
	減算代入(レジスタ指定)	α -= β (ﾚｼﾞｽﾀ)	γ←α, γ←γ-β, α←γ
	減算代入(レジスタ指定)	α -= β,CY	α←α-β,CY
	減算代入(レジスタ指定)	α -= β,CY (ﾚｼﾞｽﾀ)	γ←α, γ←γ-β,CY, α←γ
(4)	論理積代入	α &= β	α←α∩β
	論理積代入(レジスタ指定)	α &= β (ﾚｼﾞｽﾀ)	γ←α, γ←γ∩β, α←γ
(5)	論理和代入	α |= β	α←α∪β
	論理和代入(レジスタ指定)	α |= β (ﾚｼﾞｽﾀ)	γ←α, γ←γ∪β, α←γ
(6)	排他的論理和代入	α ^= β	α←α ^ β
	排他的論理和代入(レジスタ指定)	α ^= β (ﾚｼﾞｽﾀ)	γ←α, γ←γ ^ β, α←γ
(7)	右シフト代入	α >>= β	(αをβビット右シフト)
	右シフト代入(レジスタ指定)	α >>= β (ﾚｼﾞｽﾀ)	γ←α,(γをβビット右シフト), α←γ
(8)	左シフト代入	α <<= β	(αをβビット左右シフト)
	左シフト代入(レジスタ指定)	α <<= β (ﾚｼﾞｽﾀ)	γ←α,(γをβビット左シフト), α←γ

表 4-2　カウント文

カウント文		記述形式	機能
(9)	インクリメント	α++	α←α+1
(10)	デクリメント	α--	α←α-1

表 4-3　交換文

交換文		記述形式	機能
(11)	交換	α<->β	α←α<->β
	交換（レジスタ指定）	α<->β（γ）	γ←α, γ←γ<->β, α←γ

表 4-4　ビット操作文

ビット操作文		記述形式	機能
(12)	ビット・セット	α＝1	α←1
	ビット・セット（レジスタ指定）	α＝1（ＣＹ）	ＣＹ←1,　α←1
	連続ビット・セット	α 1=…＝α n=1	α n←1,…, α 1←1
	連続ビット・セット（レジスタ指定）	α 1=…＝α n=1(ＣＹ)	CY←１, α n←1,…, α 1←1
(13)	ビット・クリア	α＝0	α←0
	ビット・クリア（レジスタ指定）	α＝0（ＣＹ）	CY←0, α←0
	連続ビット・クリア	α 1=…＝α n=0	α n←0,…, α 1←0
	連続ビット・クリア（レジスタ指定）	α 1 =…＝α n =0(ＣＹ)	CY←0, α n←0,…, α 1←0

個々の式の文の機能を以下に説明します。

使用例には生成された命令です。入力ソースはコメント文として記述してあります。

代入文　　代入（=）

（1）　代入　（=）

【記述形式】

［△］［サイズ指定］［△］α１［△］［=［△］［サイズ指定］［△］α２［△］…］
=［△］[サイズ指定］［△］β［△］［（レジスタ指定）］

【機　能】

① レジスタ指定がない場合

右辺βの値を順次左辺に代入します。

② レジスタ指定のある場合

右辺βを指定されたレジスタまたはCYに代入し，それらの内容を順次左辺に代入します。

【説　明】

α，βは，MOV，MOVW命令で記述可能なものです。

“=”は１行中に32個まで記述できます。32個を越えて記述した場合はエラーとなります。

連続代入において，エラーとなる記述が１つでも存在する場合は，命令の生成を行いません。

【生成命令】

① α，βがビット・シンボルの場合

・αがCYの場合

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

ただし，連続代入は記述できません。

代入文　　代入（=）

・βがCYの場合

```
    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:
```

・レジスタ指定にCYを指定した場合

```
    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:
```

② α，βがビット・シンボルではない場合

・レジスタ指定がない場合

```
MOV     α 1,β
```

ただし，オペランドによっては，MOVW を生成します。

・レジスタ指定がなく，連続代入を記述した場合

```
MOV     α n,β
MOV     α n-1,β
        :
MOV     α 2,β
MOV     α 1,β
```

ただし，オペランドによっては，MOVW を生成します。

・レジスタ指定がある場合

```
MOV     指定レジスタ,β
MOV     α 1,指定レジスタ
```

ただし，オペランドによっては，MOVW を生成します。

・レジスタ指定があり，連続代入を記述した場合

```
MOV     指定レジスタ,β
MOV     α n,指定レジスタ
MOV     α n-1,指定レジスタ
        :
MOV     α 2,指定レジスタ
MOV     α 1,指定レジスタ
```

ただし，オペランドによっては，MOVW を生成します。

α n，βの組み合わせの詳細については，“表 4-5　代入の生成命令”を参照してください。
指定レジスタは，場合に応じてα n，βに読みかえてください。

代入文　　代入（=）

【使用例】

① レジスタ指定がない場合

```
                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 レジスタ指定がある場合

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

表 4-5　代入の生成命令

			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	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ	*3			*5																		
	c	[HL].β	*3			*5																		
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ	*6			*5		*1								*2								*1
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ						*1																*1
	f	A				*1	*1		*1	*1											*1	*1	*1	*1
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ						*1																*1
	h	R0						*1																*1
	i	R1																						*1
	j	sfr						*1																*1
	k	PSW						*1																*1
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ	*3			*5		*1								*2								
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ														*2								
	n	AX				*2								*2	*2		*2			*2				*2
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ														*2								*2
	p	RP0																						*2
	q	sfrp																						
	r	SP														*2								
	s	直接参照ｼﾝﾎﾞﾙ						*1																
	t	間接参照ｼﾝﾎﾞﾙ						*1																
	u	[DE]						*1																
	v	ｲﾐｰﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ																						

*1：MOV を生成します。
*2：MOVW を生成します。
*3：ビット分岐命令による置き換え命令を生成します。
*4：β に 1 を記述した場合は SET1 を生成します。0 を記述した場合は CLR1 を生成します。
　　0，1 以外を記述した場合は MOV を生成します。
*5：β に 1 を記述した場合は SET1 を生成します。0 を記述した場合は CLR1 を生成します。
*6：α_n に 0，1 以外を記述した場合はビット条件式による置き換え命令を生成します。
空欄はエラーを示します。

（2）　加算代入　（=）

【記述形式】

```
[△] [サイズ指定] [△] α 1 [△] += [△] [サイズ指定] [△] β [△]
                                        [, [△] CY] [△] [（レジスタ指定）]
```

【機　能】

① レジスタ指定がない場合

2 項αとβの加算を行い，その結果をαに代入します。

② レジスタ指定がある場合

αを指定されたレジスタを代入します。

指定レジスタとβを加算し，その結果を指定レジスタに代入します。

指定レジスタをαに代入します。

③ キャリー付き加算でレジスタ指定がない場合

2 項αとβのキャリー付き加算を行い，その結果をαに代入します。

④ キャリー付き加算でレジスタ指定がある場合

αを指定されたレジスタを代入します。

指定レジスタとβをキャリー付き加算し，その結果を指定レジスタに代入します。

指定レジスタをαに代入します。

【説　明】

① レジスタ指定がない場合

α，βは ADD，ADDW で記述可能なものです。

② レジスタ指定がある場合

αは MOV，MOVW で記述可能なものです。

βは ADD，ADDW で記述可能なものです。

③ キャリー付き加算でレジスタ指定がない場合

α，βは ADDC で記述可能なものです。

④ キャリー付き加算でレジスタ指定がある場合

αは MOV で記述可能なものです。

βは ADDC で記述可能なものです。

代入文　　加算代入（+=）

【生成命令】

① レジスタ指定がない場合

```
ADD     α,β
```

ただし，オペランドによっては，ADDW を生成します。

② レジスタ指定がある場合

```
MOV     指定レジスタ,α
ADD     指定レジスタ,β
MOV     α,指定レジスタ
```

ただし，オペランドによっては，ADDW を生成します。

③ キャリー付き加算でレジスタ指定がない場合

```
ADDC    α,β
```

④ キャリー付き加算でレジスタ指定がある場合

```
MOV     指定レジスタ,α
ADDC    指定レジスタ,β
MOV     α,指定レジスタ
```

α，β の組み合わせの詳細については，“表 4-6　加算代入の生成命令”を参照してください。
指定レジスタは，場合に応じてαに読みかえてください。

【使用例】

① レジスタ指定がない場合

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

② レジスタ指定がある場合

```
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 キャリー付き加算でレジスタ指定がない場合

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

④ キャリー付き加算でレジスタ指定がある場合

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

表 4-6　加算代入の生成命令

			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	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ																						
	c	[HL].β																						
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						*1
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ																						
	n	AX																						*2
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照ｼﾝﾎﾞﾙ																						
	t	間接参照ｼﾝﾎﾞﾙ																						
	u	[DE]																						
	v	ｲﾐｰﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ																						

*1：ADD を生成します。キャリー付きの場合は ADDC を生成します。
*2：ADDW を生成します。
空欄はエラーを示します。

（3）　減算代入　（=）

【記述形式】

```
[△] [サイズ指定] [△] α 1 [△] -= [△] [サイズ指定] [△] β [△]
                                        [, [△] CY] [△] [（レジスタ指定）]
```

【機　能】

① レジスタ指定がない場合

2 項αとβの減算を行い，その結果をαに代入します。

② レジスタ指定がある場合

αを指定されたレジスタを代入します。

指定レジスタとβを減算し，その結果を指定レジスタに代入します。

指定レジスタをαに代入します。

③ キャリー付き減算でレジスタ指定がない場合

2 項αとβのキャリー付き減算を行い，その結果をαに代入します。

④ キャリー付き減算でレジスタ指定がある場合

αを指定されたレジスタを代入します。

指定レジスタとβをキャリー付き減算し，その結果を指定レジスタに代入します。

指定レジスタをαに代入します。

【説　明】

① レジスタ指定がない場合

α，βは SUB，SUBW で記述可能なものです。

② レジスタ指定がある場合

αは MOV，MOVW で記述可能なものです。

βは SUB，SUBW で記述可能なものです。

③ キャリー付き減算でレジスタ指定がない場合

α，βは SUBC で記述可能なものです。

④ キャリー付き減算でレジスタ指定がある場合

αは MOV で記述可能なものです。

βは SUBC で記述可能なものです。

代入文　　減算代入（-=）

【生成命令】

① レジスタ指定がない場合

以下の命令を生成します。

SUB　　α,β

ただし，オペランドによっては，SUBW を生成します。

② レジスタ指定がある場合

以下の命令を生成します。

MOV　　指定レジスタ,α
SUB　　指定レジスタ,β
MOV　　α,指定レジスタ

ただし，オペランドによっては，SUBW を生成します。

③ キャリー付き減算でレジスタ指定がない場合

以下の命令を生成します。

SUBC　　α,β

④ キャリー付き減算でレジスタ指定がある場合

以下の命令を生成します。

MOV　　指定レジスタ,α
SUBC　　指定レジスタ,β
MOV　　α,指定レジスタ

α，β の組み合わせの詳細については，“表 4-7　減算代入の生成命令”を参照してください。
指定レジスタは，場合に応じてαに読みかえてください。

【使用例】

① レジスタ指定がない場合

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

② レジスタ指定がある場合

```
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 キャリー付き減算でレジスタ指定がない場合

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

④ キャリー付き減算でレジスタ指定がある場合

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

表 4-7　減算代入の生成命令

			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	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ																						
	c	[HL].β																						
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						*1
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ																						
	n	AX																						*2
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照ｼﾝﾎﾞﾙ																						
	t	間接参照ｼﾝﾎﾞﾙ																						
	u	[DE]																						
	v	ｲﾐｰﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ																						

*1：SUB を生成します。キャリー付きの場合は SUBC を生成します。
*2：SUBW を生成します。
空欄はエラーを示します。

（4）　論理積代入　（&=）

【記述形式】

[△] [サイズ指定] [△] α [△] &= [△] [サイズ指定] [△] β [△] [レジスタ指定]

【機　能】

① レジスタ指定がない場合

２項αとβのビットの論理積（α&β）をとり，その結果をαに代入します。

② レジスタ指定がある場合

αを指定されたレジスタに代入します。

指定レジスタとβのビットの論理積（α&β）をとり，その結果を指定レジスタに代入します。

指定レジスタの値をαに代入します。

【説　明】

① レジスタ指定がない場合

α，βはANDまたはBFで記述可能なものです。

② レジスタ指定がある場合

αは，MOVまたはBFで記述可能なものです。

βは，ANDまたはBFで記述可能なものです。

【生成命令】

① レジスタ指定がない場合

・αがCYの場合

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

・αがCY以外の場合

```
    AND     α,β
```

② レジスタ指定がある場合

・レジスタ指定がCYの場合

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

・レジスタ指定がCY以外の場合

```
    MOV     指定レジスタ,α
    AND     指定レジスタ,β
    MOV     α,指定レジスタ
```

α，βの組み合わせの詳細については，“表 4-8　論理積代入の生成命令”を参照してください。

【使用例】

① レジスタ指定がない場合

```
            BNC     $?L1        ;CY&=P1S.1
            BF      P1S.1,$?L1
            SET1    CY
            BR      ?L2
?L1:
            CLR1    CY
?L2:
            AND     A,#0FFH     ;A&=#0FFH
```

② レジスタ指定がある場合

```
            BF      A.1,$?L3    ;A.1&=PORT3.0(CY)
            BF      PORT3.0,$?L3
            SET1    A.1
            BR      ?L4
?L3:
            CLR1    A.1
?L4:
            MOV     A,[DE]      ;[DE]&=#07H(A)
            AND     A,#07H
            MOV     [DE],A
```

表 4-8　論理積代入の生成命令

			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		*2		*2								*2										
	b	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ																						
	c	[HL].β																						
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						*1
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ																						
	n	AX																						
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照ｼﾝﾎﾞﾙ																						
	t	間接参照ｼﾝﾎﾞﾙ																						
	u	[DE]																						
	v	ｲﾐﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ																						

*1：AND を生成します。
*2：ビット分岐命令のよる置き換え命令を生成します。
空欄はエラーを示します。

(5)　論理和代入　（|=）

【記述形式】

［△］［サイズ指定］［△］α［△］|=［△］［サイズ指定］［△］β［△］［レジスタ指定］

【機　能】

① レジスタ指定がない場合

２項αとβのビットの論理和（α|β）をとり，その結果をαに代入します。

② レジスタ指定がある場合

αを指定されたレジスタに代入します。

指定レジスタとβのビットの論理和（α|β）をとり，その結果を指定レジスタに代入します。

指定レジスタの値をαに代入します。

【説　明】

① レジスタ指定がない場合

α，βはORまたはBFで記述可能なものです。

② レジスタ指定がある場合

αは，MOVまたはBTで記述可能なものです。

βは，ORまたはBFで記述可能なものです。

【生成命令】

① レジスタ指定がない場合

・αがCYの場合

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

・αがCY以外の場合

```
    OR      α,β
```

② レジスタ指定がある場合

・レジスタ指定が CY の場合

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

・レジスタ指定が CY 以外の場合

```
    MOV     指定レジスタ,α
    OR      指定レジスタ,β
    MOV     α,指定レジスタ
```

α，β の組み合わせの詳細については，“表 4-9　論理和代入の生成命令”を参照してください。

【使用例】

① レジスタ指定がない場合

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

② レジスタ指定がある場合

```
        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
```

表 4-9　論理和代入の生成命令

			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		*2		*2								*2										
	b	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ																						
	c	[HL].β																						
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						*1
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ																						
	n	AX																						
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照ｼﾝﾎﾞﾙ																						
	t	間接参照ｼﾝﾎﾞﾙ																						
	u	[DE]																						
	v	ｲﾐｰﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ																						

*1：OR を生成します。
*2：ビット分岐命令による置き換え命令を生成します。
空欄はエラーを示します。

(6)　排他的論理和代入　（^=）

【記述形式】

［△］［サイズ指定］［△］α［△］^=［△］［サイズ指定］［△］β［△］［レジスタ指定］

【機　能】

① レジスタ指定がない場合

２項αとβのビットの排他的論理和（α^β）をとり，その結果をαに代入します。

② レジスタ指定がある場合

αを指定されたレジスタに代入します。

指定レジスタとβのビットの排他的論理和（α^β）をとり，その結果を指定レジスタに代入します。

指定レジスタの値をαに代入します。

【説　明】

① レジスタ指定がない場合

α，βはXORまたはBFで記述可能なものです。

② レジスタ指定がある場合

αは，MOVまたはBT，BTで記述可能なものです。

βは，XORまたはBFで記述可能なものです。

【生成命令】

① レジスタ指定がない場合

・αがCYの場合

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

・α が CY 以外の場合

```
XOR     α,β
```

② レジスタ指定がある場合

・レジスタ指定が CY の場合

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

・レジスタ指定が CY 以外の場合

```
    MOV     指定レジスタ,α
    XOR     指定レジスタ,β
    MOV     α,指定レジスタ
```

α，β の組み合わせの詳細については，“表 4-10　排他的論理和代入の生成命令”を参照してください。

【使用例】

① レジスタ指定がない場合

```
        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 レジスタ指定がある場合

```
        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
```

表 4-10　排他的論理和代入の生成命令

			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		*2		*2								*2										
	b	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ																						
	c	[HL].β																						
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						*1
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ																						
	n	AX																						
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照ｼﾝﾎﾞﾙ																						
	t	間接参照ｼﾝﾎﾞﾙ																						
	u	[DE]																						
	v	ｲﾐｰﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ																						

*1：XOR を生成します。
*2：ビット分岐命令による置き換え命令を生成します。
空欄はエラーを示します。

(7)　右シフト代入　（>>=）

【記述形式】

［△］［サイズ指定］［△］α［△］>>=［△］β［△］［（レジスタ指定）］

【機　能】

① レジスタ指定がない場合

αをβビットだけ右シフトし，その結果をαに代入します。

② レジスタ指定がある場合

αを指定されたレジスタに代入します。

指定レジスタをβビットだけ右シフトし，その結果を指定レジスタに代入します。

指定レジスタの内容をαに代入します。

【説　明】

① レジスタ指定がない場合

αはAのみ記述可能です。

βは，1～7までの数字が記述可能です。

② レジスタ指定がある場合

αは，MOV命令で記述可能なものです。

βは，1～7までの数字が記述可能です。

指定レジスタは，Aのみ記述可能です。

【生成命令】

① レジスタ指定がない場合

ROR命令をβ回出力後，AND命令を生成します。

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

② レジスタ指定がある場合

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

代入文　　右シフト代入（>>=）

【使用例】

① レジスタ指定がない場合

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

② レジスタ指定がある場合

```
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)　左シフト代入　（>>=）

【記述形式】

［△］［サイズ指定］［△］α［△］<<=［△］β［△］［（レジスタ指定）］

【機　能】

① レジスタ指定がない場合

αをβビットだけ左シフトし，その結果をαに代入します。

② レジスタ指定がある場合

αを指定されたレジスタに代入します。

指定レジスタをβビットだけ左シフトし，その結果を指定レジスタに代入します。

指定レジスタの内容をαに代入します。

【説　明】

① レジスタ指定がない場合

αは，A のみ記述可能です。

βは，1～7 までの数字が記述可能です。

② レジスタ指定がある場合

αは，MOV 命令で記述可能なものです。

βは，1～7 までの数字が記述可能です。

指定レジスタは，A のみ記述可能です。

【生成命令】

① レジスタ指定がない場合

ROL 命令をβ回出力後，AND 命令を生成します。

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

② レジスタ指定がある場合

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

代入文　　左シフト代入（>>=）

【使用例】

① レジスタ指定がない場合

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

② レジスタ指定がある場合

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

(9)　インクリメント　（++）

【記述形式】

［△］［サイズ指定］［△］α［△］++

【機　能】

αの内容に１を加えます。

【説　明】

αは，INC またはINCW で記述可能なものです。

【生成命令】

INC　　α

ただし，オペランドによってはINCW を生成します。

αの詳細については，“表 4-11　インクリメントの生成命令”を参照してください。

【使用例】

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

表 4-11　インクリメントの生成命令

α	a	CY	
	b	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ	
	c	[HL].β	
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ	*1
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ	*1
	f	A	*1
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ	*1
	h	R0	*1
	i	R1	*1
	j	sfr	
	k	PSW	
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ	
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ	
	n	AX	*2
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ	*2
	p	RP0	*2
	q	sfrp	
	r	SP	
	s	直接参照ｼﾝﾎﾞﾙ	
	t	間接参照ｼﾝﾎﾞﾙ	
	u	[DE]	
	v	ｲﾐｰﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ	

*1：INC を生成します。
*2：INCW を生成します。
空欄はエラーを示します。

（10）　デクリメント　（--）

【記述形式】

[△] [サイズ指定] [△] α [△] --

【機　能】

αの内容から1を引きます。

【説　明】

αは，DEC またはDECW で記述可能なものです。

【生成命令】

DEC　　α

ただし，オペランドによってはDECW を生成します。

αの詳細については，“表4-12　デクリメントの生成命令”を参照してください。

【使用例】

```
DEC     H       ;H--
DEC     CNT     ;CNT--
DECW    HL      ;HL--
```

表 4-12　デクリメントの生成命令

α	a	CY	
	b	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ	
	c	[HL].β	
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ	*1
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ	*1
	f	A	*1
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ	*1
	h	R0	*1
	i	R1	*1
	j	sfr	
	k	PSW	
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ	
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ	
	n	AX	*2
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ	*2
	p	RP0	*2
	q	sfrp	
	r	SP	
	s	直接参照ｼﾝﾎﾞﾙ	
	t	間接参照ｼﾝﾎﾞﾙ	
	u	[DE]	
	v	ｲﾐﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ	

*1：DEC を生成します。
*2：DECW を生成します。
空欄はエラーを示します。

交換文　　交換（<->）

（11）　交換　（<->）

【記述形式】

［△］［サイズ指定］［△］α［△］<->［△］［サイズ指定］［△］β［△］［（レジスタ指定）］

【機　能】

① レジスタ指定がない場合

αとβの内容を交換します。

② レジスタ指定がある場合

αを指定されたレジスタに代入します。

指定レジスタとβの内容を交換します。

αに指定レジスタの内容を代入します。

【説　明】

① レジスタ指定がない場合

α，βは，XCH またはXCHW で記述可能なものです。

② レジスタ指定がある場合

αは，MOV またはMOVW で記述可能なものです。

βは，XCH またはXCHW で記述可能なものです。

【生成命令】

① レジスタ指定がない場合

XCH　　α,β

ただし，オペランドによってはXCHW を生成します。

② レジスタ指定がある場合

MOV　　指定レジスタ,α

XCH　　指定レジスタ,β

MOV　　α,指定レジスタ

ただし，オペランドによってはXCHW を生成します。

α，βの組み合わせの詳細については，“表4-13　交換の生成命令”を参照してください。

指定レジスタは，αに読みかえてください。

交換文　　交換（<->）

【使用例】

① レジスタ指定がない場合

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

② レジスタ指定がある場合

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

交換文　　交換（<->）

表 4-13　交換の生成命令

			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	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ																						
	c	[HL].β																						
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ																						
	f	A			*1	*1		*1			*1		*1									*1	*1	
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ																						
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ																						
	n	AX															*2							
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照ｼﾝﾎﾞﾙ																						
	t	間接参照ｼﾝﾎﾞﾙ																						
	u	[DE]																						
	v	ｲﾐﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ																						

*1：XCH を生成します。
*2：XCHW を生成します。
空欄はエラーを示します。

（13）　ビット・セット　（=）

【記述形式】

［△］ α 1［△］［=［△］ α 2［△］…］=［△］1［△］［（CY 指定）］
右辺の最後は１を記述してください。

【機　能】

① CY 指定がない場合

α n をセット（値を１に）します。

② CY 指定がある場合

CY および α n をセット（値を１に）します。

【説　明】

α n は，SET1 命令で記述可能なものです。

“=” は１行中に最大 32 個まで記述できます。32 個を越えて記述した場合はエラーとなります。

連続代入において，エラーとなる記述が存在する場合には，命令の生成は行いません。

【生成命令】

① CY 指定がない場合

```
SET1    α 1
```

② 連続代入で CY 指定がない場合

```
SET1    α n
SET1    α n-1
   :
SET1    α 2
SET1    α 1
```

③ CY 指定がある場合

```
SET1    CY
SET1    α 1
```

④ 連続代入でCY指定がある場合

```
SET1    CY
SET1    α n
SET1    α n-1
        :
SET1    α 2
SET1    α 1
```

詳細については“表4-14　ビット・セットの生成命令”を参照してください。

【使用例】

① CY指定がない場合

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

② CY指定がある場合

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

表 4-14　ビット・セットの生成命令

α	a	CY	*1
	b	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ	*1
	c	[HL].β	*1
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ	*1
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ	
	f	A	
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ	
	h	R0	
	i	R1	
	j	sfr	
	k	PSW	
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ	*1
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ	
	n	AX	
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ	
	p	RP0	
	q	sfrp	
	r	SP	
	s	直接参照ｼﾝﾎﾞﾙ	
	t	間接参照ｼﾝﾎﾞﾙ	
	u	[DE]	
	v	ｲﾐｰﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ	

*1：SET1 を生成します。
空欄はエラーを示します。

（14）　ビット・クリア　（=）

【記述形式】

[△] α1 [= [△] α2 [△] …] = [△] 0 [△] [（CY指定）]
右辺の最後には0を記述してください。

【機　能】

① CY指定がない場合

αnをクリア（値を0に）します。

② CY指定がある場合

CYおよびαnをクリア（値を0に）します。

【説　明】

αnは，CLR1命令で記述可能なものです。

“=”は1行中に最大32個まで記述できます。32個を越えて記述した場合はエラーとなります。

連続代入において，エラーとなる記述が存在する場合には，命令の生成は行いません。

【生成命令】

① CY指定がない場合

```
CLR1    α1
```

② 連続代入でCY指定がない場合

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

③ CY指定がある場合

```
CLR1    CY
CLR1    α1
```

④ 連続代入で CY 指定がある場合

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

詳細については“表 4-15　ビット・クリアの生成命令”を参照してください。

【使用例】

① CY 指定がない場合

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

② CY 指定がある場合

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

表 4-15　ビット・クリアの生成命令

α	a	CY	*1
	b	ﾋﾞｯﾄ･ｼﾝﾎﾞﾙ	*1
	c	[HL].β	*1
	d	ﾊﾞｲﾄ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ	*1
	e	ﾊﾞｲﾄ･ﾃﾞｰﾀ	
	f	A	
	g	ﾊﾞｲﾄ･ﾚｼﾞｽﾀ	
	h	R0	
	i	R1	
	j	sfr	
	k	PSW	
	l	ﾜｰﾄﾞ･ﾕｰｻﾞ･ｼﾝﾎﾞﾙ	*1
	m	ﾜｰﾄﾞ･ﾃﾞｰﾀ	
	n	AX	
	o	ﾜｰﾄﾞ･ﾚｼﾞｽﾀ	
	p	RP0	
	q	sfrp	
	r	SP	
	s	直接参照ｼﾝﾎﾞﾙ	
	t	間接参照ｼﾝﾎﾞﾙ	
	u	[DE]	
	v	ｲﾐﾃﾞｨｴｲﾄ･ｼﾝﾎﾞﾙ	

*1 : CLR1 を生成します。
空欄はエラーを示します。

第 5 章　擬似命令

この章では，擬似命令について説明します。擬似命令とは，ST78K0S が一連の処理を行う際に必要な各種の指示を行うものです。

5.1　擬似命令の概要

擬似命令は，ST78K0S が一連の処理を行う際に必要な各種の指示を行うものであり，ソース・プログラム中に記述します。

擬似命令を記述することにより，ソース・プログラムの記述が容易になります。

擬似命令は，出力ファイルには出力されません。

5.2　擬似命令の機能

“表 5-1　擬似命令，制御命令一覧”にその種類を示します。

表 5-1　擬似命令一覧

擬似命令の種類	擬似命令
シンボル定義擬似命令	#define
条件付き処理擬似命令	#ifdef : #else : #endif
インクルード擬似命令	#include
CALLT 置換擬似命令	#defcallt : #endcallt

各擬似命令の機能を以下に示します。

(1)　シンボル定義擬似命令（#define）

【記述形式】

［△］#［△］define△シンボル△文字列

【機　能】

ソース・プログラム中に記述されたシンボルを指定した文字列に置き換えます。

【説　明】

① ‘#’文字は，空白，HT を除いて，最初に記述されていなければなりません。

② シンボルは，英文字から始まり英数字から構成し，ディフォールトで先頭 31 文字，オプション“NS”を指定した場合は 8 文字までが有効です。8 文字有効時に，9 文字以上のシンボルを指定した場合は，9 文字以降を無視します。31 文字有効時に，32 文字以上のシンボルを指定した場合は，32 文字以降を無視します。

③ 文字列は，「2.2 (1)　文字セット」で規定する文字の並びで構成します。空白および引用符は記述できません。もし，記述されている場合は，無視して処理を続行します。

④ 本擬似命令は，数値などを読みやすいシンボルで記述する場合に有効です。

⑤ シンボルとして，予約語は記述できません。

⑥ 文字列として，予約語を記述可能です。

⑦ 同一シンボルを再定義した場合，ワーニング・メッセージを出力します。

⑧ 2 次ソース・ファイルには変換した文字列を出力します。#define 文は出力しません。

⑨ 変換した文字列が，別の#define で定義されていれば，31 回までは再度変換を行います。32 回以上はエラー・メッセージを出力し，32 回目以降の定義を無視します。

⑩ 本命令は，ソース中のどこにでも記述可能です。

⑪ オプション D の指定シンボルと重複した場合，ワーニングメッセージを出力し，#define を有効とします。

【使用例】

＜入力ソース・プログラム＞

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

＜出力ソース・プログラム＞

```
          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)　条件付き処理擬似命令（ #ifdef ／ #else ／ #endif ）

【記述形式】

```
[△] # [△] ifdef△シンボル
        テキスト 1
[△] # [△] else
        テキスト 2
[△] # [△] endif
```

【機　能】

条件付き処理を行います。

① シンボルが未定義の場合

#else が記述されていれば，テキスト 1 を読み飛ばしてテキスト 2 を処理対象とします。

② シンボルが定義されている場合

#else が記述されているとテキスト 1 が処理対象となり，テキスト 2 は読み飛ばされます。

【説　明】

① '#' 文字は，空白，HT を除いて，最初に記述されていなければなりません。

② シンボルは英文字から始まる英数字から構成し，ディフォールトで先頭 31 文字，オプション "-NS" を指定した場合は 8 文字までが有効となります。

③ シンボルはあらかじめ#define 文，または起動時のオプション "-D" で定義します。

④ 本擬似命令は 8 レベルまで，ネストさせることが可能です。

⑤ #else は省略可能です。

【使用例】

＜入力ソース・プログラム＞

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

① コマンド行に次のように記述した場合（シンボルが定義されている場合）

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

＜出力ソース・プログラム＞

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

② コマンド行に次のように記述した場合（シンボルが定義されていない場合）

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

＜出力ソース・プログラム＞

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

(3)　インクルード擬似命令（#include）

【記述形式】

[△] # [△] include△”ファイル名”

【機　能】

この 1 行を，指定されたファイル名の内容で置き換えて ST78K0S のソース・プログラムとして処理対象とします。

【説　明】

① ‘#’ 文字は，空白，HT を除いて，最初に記述されていなければなりません。
② 本擬似命令は，ソースプログラム中のどの行にでも記述できます。
③ インクルードファイルの中に，インクルード擬似命令は記述できません。
すなわち，インクルードのネスティングは許されません。
④ ファイル名として，起動行に指定した入力ソースファイル名，出力ファイル名，エラー・ファイル名は指定できません。
⑤ ファイル名の先頭にドライブ名，ディレクトリ名が記述できます。それらの記述がない場合は，カレントドライブおよびカレントディレクトリにインクルードファイルがあるものとして処理されます。
⑥ ST78K0S の起動時に，-I オプションでインクルードファイルのドライブ名およびディレクトリの指定をすることができます。

#INCLUDE　　#include　　#INCLUDE

【使用例】

＜入力ソース・プログラム＞

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

＜入力インクルード・プログラム＞

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

＜出力ソース・プログラム＞

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

(4)　CALLT 置換擬似命令（#defcallt）

【記述形式】

```
[△] # [△] defcallt△CALLT テーブルのレーベル
[△] CALL△　　! レーベル
[△] # [△] endcallt
```

【機　能】

登録されたレーベルへの CALL 命令を CALLT 命令に置き換えて，2 次ソースファイルに出力します。

【説　明】

① ソース・プログラムではすべて CALL で記述しておいて，CALLT テーブルに登録できたレーベルを本擬似命令で定義します。そうすれば，定義されたレーベルへの CALL 命令は，すべて CALLT 命令に変換されます。

② 本擬似命令は，最大 32 回記述可能です。なお，32 回以上記述した場合，エラー・メッセージを出力したあと，その記述を無効として処理を継続します。

③ 同一パターンの再定義はエラーメッセージを出力したあと，その記述を無効として処理を継続します。

【使用例】

＜入力ソース・プログラム＞

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

＜出力ソース・プログラム＞

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

第 6 章　制御命令

この章では，制御命令について説明します。制御命令とは構造化アセンブラの動作に対し細かい指示を与えるものです。

6.1　制御命令の概要

制御命令は，ST78K0S が一連の処理を行う際に必要な各種の指示を行うものであり，ソース・プログラム中に記述します。

制御命令を記述することにより，プログラムの起動時にオプションを指定する手間が省けます。

6.2　アセンブラの制御命令

アセンブラ制御命令について，モジュール・ヘッダに記述可能かどうかの判断を行います。

モジュール・ヘッダに記述できないアセンブラ制御命令があった場合，それ以降はモジュール・ボディとして処理を続けます。また，モジュール・ヘッダにのみ記述可能なアセンブラ制御命令をモジュール・ボディに記述した場合，エラー・メッセージを出力し，アボートします。

本プリプロセッサでは，プロセッサ品種指定制御命令（$PROCESSOR，$PC），シンボル名長制御命令（$SYMLEN，$NOSYMLEN）および漢字コード指定制御命令（$KANJICODE）以外のパラメータに，正しいものが指定されたかどうかの確認は行いません。他の制御命令の記述形式については，“RA78K0S アセンブラ・パッケージ アセンブリ言語編”を参照してください。

次に，モジュール・ヘッダにのみ記述可能な制御命令，モジュール・ボディとして認識する制御命令を示します。

表 6-1　モジュール・ヘッダにのみ記述可能な制御命令

制御命令
[△] $ [△] PROCESSOR [△] （ [△] 品種名 [△] ）
[△] $ [△] PC （ [△] 品種名 [△] ）
[△] $ [△] DEBUG
[△] $ [△] DG
[△] $ [△] NODEBAG
[△] $ [△] NODG
[△] $ [△] DEBUGA
[△] $ [△] NODEBAGA
[△] $ [△] XREF
[△] $ [△] XR
[△] $ [△] NOXREF
[△] $ [△] NOXR
[△] $ [△] TITLE [△] （ [△] ‘タイトルストリング’ [△] ）
[△] $ [△] TT [△] （ [△] ‘タイトルストリング’ [△] ）
[△] $ [△] SYMLEN
[△] $ [△] NOSYMLEN
[△] $ [△] CAP
[△] $ [△] NOCAP
[△] $ [△] SYMLIST
[△] $ [△] NOSYMLIST
[△] $ [△] FORMFEED
[△] $ [△] NOFORMFEED
[△] $ [△] WIDTH [△] （ [△] 定数 [△] ）
[△] $ [△] LENGTH [△] （ [△] 定数 [△] ）
[△] $ [△] TAB [△] （ [△] 定数 [△] ）
[△] $ [△] KANJICODE△漢字コード

表 6-2　モジュール・ボディとして認識する制御命令

制御命令
[△] $ [△] INCULUDE [△] （ [△] ファイル名 [△] ）
[△] $ [△] IC（ [△] ファイル種名 [△] ）
[△] $ [△] EJECT
[△] $ [△] EJ
[△] $ [△] LIST
[△] $ [△] LI
[△] $ [△] NOLIST
[△] $ [△] NOLI
[△] $ [△] GEN
[△] $ [△] NOGEN
[△] $ [△] COND
[△] $ [△] NOCOND
[△] $ [△] SUBTITLE [△] （ [△] ‘文字列’ [△] ）
[△] $ [△] ST [△] （ [△] ‘文字列’ [△] ）
[△] $ [△] SET [△] （ [△] スイッチ名 [[△] : [△] スイッチ名… [△] ）
[△] $ [△] RESET [△] （ [△] スイッチ名 [[△] : [△] スイッチ名… [△] ）
[△] $ [△] IF [△] （ [△] スイッチ名 [[△] : [△] スイッチ名… [△] ）
[△] $ [△] _IF△条件式
[△] $ [△] ELSEIF [△] （ [△] スイッチ名 [[△] : [△] スイッチ名… [△] ）
[△] $ [△] _ELESEIF△条件式
[△] $ [△] SET [△] （ [△] スイッチ名 [[△] : [△] スイッチ名… [△] ）
[△] $ [△] ELSE
[△] $ [△] ENDIF

6.3　制御命令の機能

“表 6-3　制御命令一覧”にその種類を示します。

表 6-3　制御命令一覧

制御命令の種類	制御命令
プロセッサ品種指定	$PROCESSOR
シンボル名長指定	$SYMLEN／$NOSYMLEN
漢字コード指定	$KANJICODE

各制御命令の機能を以下に示します。

(1)　プロセッサ品種指定命令　($PROCESSOR)

【記述形式】

```
[△] $ [△] PROCESSOR [△] ( [△] 品種名 [△] )
[△] $ [△] PC [△] ( [△] 品種名 [△] )                    ; 省略形
```

【機　能】

ソース・モジュール中でアセンブル対象品種を指定します。

【説　明】

① 本制御命令は，アセンブラのアセンブル対象品種指定制御命令ですが，構造化アセンブラでも対象品種指定のための制御命令としています。

② オプション“-C”と異なる品種が指定された場合は，オプションで指定した品種を優先します。この際，異なる品種が指定されたことを示すワーニング・メッセージを出力します。2 次ソース・ファイルには，入力ソース・ファイルの制御命令の‘$’を‘;’に置き換えて出力し，オプションで指定された品種をプロセッサ品種指定制御命令として出力します。オプション“-C”と同名の品種が指定された場合は，メッセージは出力しません。なお，オプション“-C”による指定がない場合は，ソース・モジュールの先頭に記述なければなりません（スペースおよびコメントは含みません）。

③ 本制御命令が重複記述された場合はエラーとなります。

④ 本制御命令とオプション“-C”の両方で品種指定がない場合はエラーとなります。

⑤ 本制御命令がモジュール・ヘッダ以外に記述された場合はエラーとなります。

【記述例】

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

(2)　シンボル名長制御命令（$SYMLEN／$NOSYMLEN）

【記述形式】

```
[△] $ [△] SYMLEN
[△] $ [△] NOSYMLEN
```

【機　能】

SYMLEN 制御命令は, #define で定義するシンボル名, #ifdef で参照するシンボル名およびユーザ・シンボルの有効長を 31 文字までとします。

NOSYMLEN 制御命令は, #define で定義するシンボル名, #ifdef で参照するシンボル名およびユーザ・シンボルの有効長を 8 文字までとします。

【説　明】

① 本制御命令は，入力ソース・ファアイルのモジュール・ヘッダ部に記述可能とします。
② 本制御命令がモジュール・ヘッダ部以外に記述された場合はエラーとなります。
③ 重複記述された場合は，後者優先とします。
④ シンボル名長制御命令は，コマンド・ライン上でオプション“-S”および“-NS”によっても指定可能です。本制御命令よりもオプションを優先します。
⑤ デフォルトの解釈は$SYMLEN とします。
⑥ オプション“-S”を指定しかつ，入力ソース・ファイルに$NOSYMLEN を記述した場合，コメント文に置き換えて，$SYMLEN を 2 次ソース・ファイルに出力します。

オプション“-NS”を指定しかつ，入力ソース・ファイルに$SYMLEN を記述した場合，コメント文に置き換えて，$NOSYMLEN を 2 次ソース・ファイルに出力します。

【記述例】

```
$SYMLEN
$NOSYMLEN
```

(3)　漢字コード指定制御命令（$KANJICODE）

【記述形式】

[△] $ [△] KANJICODE△漢字コード

【機　能】

コメントに記述された漢字コードを次のように解釈します。

表 6-4　漢字コードの解釈

漢字コード	解釈
SJIS	シフト JIS コードとして解釈します。
EUC	EUC コードとして解釈します。
NONE	漢字として解釈しません。

【説　明】

① 本制御命令は，入力ソース・ファアイルのモジュール・ヘッダ部に記述可能とします。

② 本制御命令がモジュール・ヘッダ部以外に記述された場合はエラーとなります。

③ 重複記述された場合は，後者優先とします。

④ 本プリプロセッサは，指定された制御命令を 2 次ソース・ファイルに出力します。

SJIS　：　$KANJICODE SJIS
EUC　：　$KANJICODE EUC
NONE　：　$KANJICODE NONE

2 次ソース・ファイルに同様の制御命令が記述されている場合，制御命令を出力しません。ただし，エラーチェックは行います。

⑤ 漢字コードの指定の優先順位は“1.3.2　環境変数”を参照してください。

【記述例】

```
$KANJICODE SJIS
```

付録 A　構文一覧

表 A-1　制御文

制御文	記述形式	ページ

if 文	if(条件式 1)[(レジスタ名)]
	　if 節
	elseif(条件式 2)[(レジスタ名)]
	　elseif 節
	else
	　else 節
	endif
		24

switch 文	switch(シンボル)[(レジスタ名)] 　case: 定数 1: 　　case1 節 　case 定数 2: 　　case2 節 　　: 　case 定数 N: 　　caseN 節 　default: 　　default 節 ends	30
for 文	for(式の文;条件式;式の文)[(レジスタ名)] 　命令群 next	34
while 文	while(条件式)[(レジスタ名)] 　命令群 endw	37
until 文	repeat 　命令群 until(条件式)[(レジスタ名)]	41
break 文	break	44
contiue 文	continue	45
goto 文	goto　レーベル	46

if_bit 文	if_bit(条件式 1)[(レジスタ名)]
	　if_bit 節
	elseif_bit(条件式 2)[(レジスタ名)]
	　elseif_bit 節
	else
	　else 節
	endif
		27

while_bit 文	while_bit(条件式)[(レジスタ名)] 　命令群 endw	39
until_bit 文	repeat 　命令群 until_bit(条件式)[(レジスタ名)]	43

表 A-2　条件式

条件式	記述形式	機能	ページ
Equal	α==β	α=βのとき真，α≠βのとき偽	50
NotEqual	α!=β	α≠βのとき真，α=βのとき偽	53
LessThan	α<β	α<βのとき真，α>=βのとき偽	56
GreaterThan	α>β	α>βのとき真，α<=βのとき偽	59
GreaterEqual	α>=β	α>=βのとき真，α<βのとき偽	62
LessEqual	α<=β	α<=βのとき真，α>βのとき偽	65
FOREVER	forever	ループ文を永久にループさせる	68
正論理（ビット）	ビット・シンボル	指定されたビット・シンボルが1のとき真	71
負論理（ビット）	!ビット・シンボル	指定されたビット・シンボルが0のとき真	74
論理積	条件式1 && 条件式2	条件式1，条件式2が共に真であれば真	78
論理和	条件式1 || 条件式2	条件式1または条件式2が真であれば真	81

表 A-3　式の文（1/2）

式の文	記述形式	機能	ページ
代入	α = β	α←β	85
代入(レジスタ指定)	α = β (γ)	(γ)←β　α←(γ)	
連続代入	α1 =…= αn =β	α1←β,…,αn←β	
連続代入(レジスタ指定)	α1 =…= αn = β(γ)	γ←β,α1←γ,…,αn←γ	
加算代入	α += β	α←α+β	90
加算代入(レジスタ指定)	α += β (ﾚｼﾞｽﾀ)	γ←α,γ←γ+β,α←γ	
加算代入(レジスタ指定)	α += β,CY	α←α+β,CY	
加算代入(レジスタ指定)	α += β,CY (ﾚｼﾞｽﾀ)	γ←α,γ←γ+β,CY,α←γ	
減算代入	α -= β	α←α-β	94
減算代入(レジスタ指定)	α -= β (ﾚｼﾞｽﾀ)	γ←α,γ←γ-β,α←γ	
減算代入(レジスタ指定)	α -= β,CY	α←α-β,CY	
減算代入(レジスタ指定)	α -= β,CY (ﾚｼﾞｽﾀ)	γ←α,γ←γ-β,CY,α←γ	
論理積代入	α &= β	α←α∩β	98
論理積代入(レジスタ指定)	α &= β (ﾚｼﾞｽﾀ)	γ←α,γ←γ∩β,α←γ	
論理和代入	α |= β	α←α∪β	101
論理和代入(レジスタ指定)	α |= β (ﾚｼﾞｽﾀ)	γ←α,γ←γ∪β,α←γ	
排他的論理和代入	α ^= β	α←α^β	105
排他的論理和代入(レジスタ指定)	α ^= β (ﾚｼﾞｽﾀ)	γ←α,γ←γ^β,α←γ	
右シフト代入	α >>= β	(αをβビット右シフト)	109
右シフト代入(レジスタ指定)	α >>= β (ﾚｼﾞｽﾀ)	γ←α,(γをβビット右シフト),α←γ	
左シフト代入	α <<= β	(αをβビット左右シフト)	111
左シフト代入(レジスタ指定)	α <<= β (ﾚｼﾞｽﾀ)	γ←α,(γをβビット左シフト),α←γ	
インクリメント	α++	α←α+1	113
デクリメント	α--	α←α-1	115

表 A-3　式の文（2/2）

式の文	記述形式	機能	ページ
交換	α<->β	α←α<->β	117
交換（レジスタ指定）	α<->β（γ）	γ←α, γ←γ<->β, α←γ	
ビット・セット	α＝1	α←1	120
ビット・セット（レジスタ指定）	α＝1（ＣＹ）	ＣＹ←1,　α←1	
連続ビット・セット	α 1=…＝α n=1	α n←1,…, α 1←1	
連続ビット・セット（レジスタ指定）	α 1=…＝α n=1(ＣＹ)	CY←1, α n←1,…, α 1←1	
ビット・クリア	α＝0	α←0	123
ビット・クリア（レジスタ指定）	α＝0（ＣＹ）	CY←0, α←0	
連続ビット・クリア	α 1=…＝α n=0	α n←0,…, α 1←0	
連続ビット・クリア（レジスタ指定）	α１=…＝α ｎ=0(ＣＹ)	CY←0, α n←0,…, α 1←0	

表 A-4　擬似命令

擬似命令	記述形式	ページ
#define	#define シンボル 文字列	128
#ifdef	#ifdef シンボル テキスト１ #else テキスト２ #endif	130
#include	#include “ファイル名”	132
#defcallt	#defcallt CALLT テーブルのレーベル CALL レーベル #endcallt	134

付録 B　生成命令一覧

表 B-1　比較条件式の生成命令（1/3）

比較条件式	生成命令		制御文の条件	ページ
$\alpha == \beta$	CMP(W) BNZ	α, β $?LFALSE	小文字	50
	CMP(W) BZ BR ?LTRUE:	α, β $?LTRUE ?LFALSE	大文字	
$\alpha == \beta$ （γ）	MOV(W) CMP(W)) BNZ	γ, α γ, β $?LFALSE	小文字	
	MOV(W) CMP(W) BZ BR ?LTRUE:	γ, α γ, β $?LTRUE ?LFALSE	大文字	
$\alpha != \beta$	CMP(W) BZ	α, β $?LFALSE	小文字	53
	CMP(W) BNZ BR ?LTRUE:	α, β $?LTRUE ?LFALSE	大文字	
$\alpha != \beta$ （γ）	MOV(W) CMP(W) BZ	γ, α γ, β $?LFALSE	小文字	
	MOV(W) CMP(W) BNZ BR ?LTRUE:	γ, α γ, β $?LTRUE ?LFALSE	大文字	
$\alpha < \beta$	CMP(W) BNC	α, β $?LFALSE	小文字	56
	CMP(W) BC BR ?LTRUE:	α, β $?LTRUE ?LFALSE	大文字	
$\alpha < \beta$ （γ）	MOV(W) CMP(W) BNC	γ, α γ, β $?LFALSE	小文字	
	MOV(W) CMP(W) BC BR ?LTRUE:	γ, α γ, β $?LTRUE ?LFALSE	大文字	

表 B-1　比較条件式の生成命令（2/3）

比較条件式	生成命令		制御文の条件	ページ
α>β	CMP(W)	α,β	小文字	59
	BZ	$?LFALSE		
	BC	$?LFALSE		
	CMP(W)	α,β	大文字	
	BZ	$$+4		
	BNC	$?LTRUE		
	BR	?LFALSE		
	?LTRUE:			
α>β （γ）	MOV(W)	指定レジスタ,α	小文字	
	CMP(W)	指定レジスタ,β		
	BZ	$?LFALSE		
	BC	$?LFALSE		
	MOV(W)	指定レジスタ,α	大文字	
	CMP(W)	指定レジスタ,β		
	BZ	$$+4		
	BNC	$?LTRUE		
	BR	?LFALSE		
	?LTRUE:			
α>=β	CMP(W)	α,β	小文字	62
	BC	$?LFALSE		
	CMP(W)	α,β	大文字	
	BNC	$?LTRUE		
	BR	?LFALSE		
	?LTRUE:			
α>=β （γ）	MOV(W)	γ,α	小文字	
	CMP(W)	γ,β		
	BC	$?LFALSE		
	MOV(W)	γ,α	大文字	
	CMP(W)	γ,β		
	BNC	$?LTRUE		
	BR	?LFALSE		
	?LTRUE:			

表 B-1　比較条件式の生成命令（3/3）

α<=β	CMP(W) α,β BZ $$+4 BNC $?LFALSE	小文字	65
	CMP(W) α,β BZ $?LTRUE BC $?LTRUE BR ?LFALSE ?LTRUE:	大文字	
α<=β （γ）	MOV(W) 指定レジスタ,α CMP(W) 指定レジスタ,β BZ $$+4 BNC $?LFALSE	小文字	
	MOV(W) 指定レジスタ,α CMP(W) 指定レジスタ,β BZ $?LTRUE BC $?LTRUE BR ?LFALSE ?LTRUE:	大文字	

γ：指定レジスタ

表 B-2　ビット条件式の生成命令

ビット条件式	生成命令		制御文条件	ページ
if_bit（ビット・シンボル） elseif_bit（ビット・シンボル） while_bit（ビット・シンボル） until_bit（ビット・シンボル）	BNC	$?LFALSE	小文字(CY)	71
	BNZ	$?LFALSE	小文字(Z)	
	BF	ビット・シンボル,$?LFALSE	小文字	
	BC BR ?LTRUE:	$?LTRUE ?LFALSE	大文字(CY)	
	BZ BR ?LTRUE:	$?LTRUE ?LFALSE	大文字(Z)	
	BT BR ?LTRUE:	ビット・シンボル,$?LTRUE ?LFALSE	大文字	
if_bit（!ビット・シンボル） elseif_bit（!ビット・シンボル） while_bit（!ビット・シンボル） until_bit（!ビット・シンボル）	BC	$?LFALSE	小文字(CY)	74
	BZ	$?LFALSE	小文字(Z)	
	BT	ビット・シンボル,$?LFALSE	小文字	
	BNC BR ?LTRUE:	$?LTRUE ?LFALSE	大文字(CY)	
	BNZ BR ?LTRUE:	$?LTRUE ?LFALSE	大文字(Z)	
	BF BR ?LTRUE:	ビット・シンボル,$?LTRUE ?LFALSE	大文字	

表 B-3　論理演算式の生成命令（1/2）

論理演算	生成命令		制御文の条件	ページ
α == β　&&	CMP(W) BNZ	α , β $?LFALSE	小文字	78
	CMP(W) BZ BR ?LTRUE:	α , β $?LTRUE ?LFALSE	大文字	
α != β　&&	CMP(W) BZ	α , β $?LFALSE	小文字	
	CMP(W) BNZ BR ?LTRUE:	α , β $?LTRUE ?LFALSE	大文字	
α < β　&&	CMP(W) BNC	α , β $?LFALSE	小文字	
	CMP(W) BC BR ?LTRUE:	α , β $?LTRUE ?LFALSE	大文字	
α > β　&&	CMP(W) BZ BC	α , β $?LFALSE $?LFALSE	小文字	
	CMP(W) BZ BNC BR ?LTRUE:	α , β $$+4 $?LTRUE ?LFALSE	大文字	
α >= β　&&	CMP(W) BC	α , β $?LFALSE	小文字	
	CMP(W) BNC BR ?LTRUE:	α , β $?LTRUE ?LFALSE	大文字	
α <= β　&&	CMP(W) BZ BNC	α , β $$+4 $?LFALSE	小文字	
	CMP(W) BZ BC BR ?LTRUE:	α , β $?LTRUE $?LTRUE ?LFALSE	大文字	

表 B-3　論理演算式の生成命令（2/2）

論理演算	生成命令		制御文の条件	ページ
CY &&	BNC	$?LFALSE	小文字	78
	BC BR ?LTRUE:	$?LTRUE ?LFALSE	大文字	
Z &&	BNZ	$?LFALSE	小文字	
	BZ BR ?LTRUE:	$?LTRUE ?LFALSE	大文字	
ビット・シンボル &&	BF	ビット・シンボル,$?LFALSE	小文字	
	BT BR ?LTRUE:	ビット・シンボル,$?LTRUE ?LFALSE	大文字	
!CY &&	BC	$?LFALSE	小文字	
	BNC BR ?LTRUE:	$?LTRUE ?LFALSE	大文字	
!Z &&	BZ	$?LFALSE	小文字	
	BNZ BR ?LTRUE:	$?LTRUE ?LFALSE	大文字	
!ビット・シンボル &&	BT	ビット・シンボル,$?LFALSE	小文字	
	BF BR ?LTRUE:	ビット・シンボル,$?LTRUE ?LFALSE	大文字	
$\alpha == \beta$ ||	CMP(W) BZ	α, β $?LFALSE		81
$\alpha != \beta$ ||	CMP(W) BNZ	α, β $?LFALSE		
$\alpha < \beta$ ||	CMP(W) BC	α, β $?LFALSE		
$\alpha > \beta$ ||	CMP(W) BZ BNC	α, β $?LFALSE $?LFALSE		
$\alpha >= \beta$ ||	CMP(W) BNC	α, β $?LFALSE		
$\alpha <= \beta$ ||	CMP(W) BZ BC	α, β $?LFALSE $?LFALSE		
CY ||	BC	$?LFALSE		
Z ||	BZ	$?LFALSE		
ビット・シンボル ||	BT	ビット・シンボル,$?LFALSE		
!CY ||	BNC	$?LFALSE		
!Z ||	BNZ	$?LFALSE		
!ビット・シンボル ||	BF	ビット・シンボル,$?LFALSE		

表 B-4　式の文（1/4）

代入文	生成命令	ページ
α=β	MOV α 1,β	85
	MOVW α 1,β	
	BNC ?L1 SET1 α BR ?L2 ?L1: CLR1 α ?L2:	
α=β （γ）	MOV γ,β MOV α 1,γ	
	MOVW γ,β MOVW α 1,γ	
	BF β,?L1 SET1 α BR ?L2 ?L1: CLR1 α ?L2:	
α+=β	ADD α,β	90
	ADDW α,β	

α+=β （γ）	MOV γ,α
	ADD γ,β
	MOV α,γ	

	MOVW γ,α
	ADDW γ,β
	MOVW α,γ	

α+=β,CY	ADDC α,β	

α+=β,CY （γ）	MOV γ,α
	ADDC γ,β
	MOV α,γ	

表 B-4　式の文（2/4）

代入文	生成命令	ページ
α -= β	SUB α,β	94
	SUBW α,β	

α -= β （γ）	MOV γ,α
	SUB γ,β
	MOV α,γ	

	MOVW γ,α
	SUBW γ,β
	MOVW α,γ	

α -= β ,CY	SUBC α,β	

α -= β ,CY （γ）	MOV γ,α
	SUBC γ,β
	MOV α,γ	

α &= β	AND α,β	
	BNC ?L1 BF β,?L1 SET1 CY BR ?L2 ?L1: CLR1 CY ?L2:	98

α &= β （γ）	MOV γ,α
	AND γ,β
	MOV α,γ	

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

表 B-4　式の文（3/4）

代入文	生成命令			ページ
α |= β		OR	α , β	101
		BC	?L1	
		BF	β ,?L2	
	?L1:			
		SET1	CY	
		BR	?L3	
	?L2:			
		CLR1	CY	
	?L3:			
α |= β （γ）		MOV	γ , α	
		OR	γ , β	
		MOV	α , γ	
		BT	α ,?L1	
		BF	β ,?L2	
	?L1:			
		SET1	α	
		BR	?L3	
	?L2:			
		CLR1	α	
	?L3:			
α ^= β		XOR	α , β	105
		BNC	?L1	
		BF	β ,?L2	
	?L1:			
		BC	?L3	
		BF	β ,?L3	
	?L2:			
		SET1	CY	
		BR	?L4	
	?L3:			
		CLR1	CY	
	?L4:			
α ^= β （γ）		MOV	γ , α	
		XOR	γ , β	
		MOV	α , γ	
		BF	α ,?L1	
		BF	β ,?L2	
	?L1:			
		BT	α ,?L3	
		BF	β ,?L3	
	?L2:			
		SET1	α	
		BR	?L4	
	?L3:			
		CLR1	α	
	?L4:			

表 B-4　式の文（4/4）

代入文	生成命令	ページ
α >>= β	ROR　A,1 ： AND　A,#0FFH SHR β	109
α >>= β （γ）	MOV　A, α ROR　A,1 ： AND　A,#0FFH SHR β MOV　α ,A	
α <<= β	ROL　A,1 ： AND　A,#LOW(0FFH SHR β)	111
α <<= β （γ）	MOV　A, α ROL　A,1 ： AND　A,#LOW(0FFH SHL β) MOV　α ,A	
α ++	INC　α	113
	INCW　α	
α --	DEC　α	115
	DECW　α	
α <-> β	XCH　α , β	117
	XCHW　α , β	
α <-> β （γ）	MOV　γ , α XCH　γ , β MOV　α , γ	
	MOVW　γ , α XCHW　γ , β MOVW　α , γ	
α =1	SET1　α 1	120
α =1(cy)	SET1　CY SET1　α 1	
α =0	CLR1　α 1	123
α =0(CY)	CLR1　CY CLR1　α 1	

付録 C　最大性能一覧

表 C-1　構造化アセンブラの最大性能

項　　目	制限値
一行の長さ (LF,CR は含まない)	218 個
#define 擬似命令の登録シンボル数 (予約語は除く)	512 個
制御文のネスティング・レベル	31 レベル
#ifdef 擬似命令のネスティング・レベル	8 レベル
#defcallt 擬似命令	32 個
#include 擬似命令のネスティング	できません
#define 擬似命令で再定義可能な回数	31 回
連続代入できるオペランド数	33 個[注 1]
論理演算子のオペランド	17 個[注 2]
-D オプションで定義可能なシンボル数	30 個

注 1：以下のようになります。

S1=S2= … S32=S33

シンボル数は 33 個，“=”は 32 個まで記述可能です。

注 2：以下のようになります。

式 1&&式 2&& … &&式 16&&式 17

式は 17 個，“&&（または||）”は 16 個まで記述可能です。

〔メ　モ〕

お問い合わせ先

【技術的なお問い合わせ先】

ＮＥＣ半導体テクニカルホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）

電　話　：044-435-9494
FAX　：044-435-9608
E-mail　：s-info@saed.tmg.nec.co.jp

【営業関係お問い合わせ先】

第一販売事業部
東　京　(03)3798-6106, 6107, 6108
名古屋　(052)222-2375
大　阪　(06)6945-3178, 3200, 3208, 3212
仙　台　(022)267-8740
郡　山　(024)923-5591
千　葉　(043)238-8116

第二販売事業部
東　京　(03)3798-6110, 6111, 6112
立　川　(042)526-5981, 6167
松　本　(0263)35-1662
静　岡　(054)254-4794
金　沢　(076)232-7303
松　山　(089)945-4149

第三販売事業部
東　京　(03)3798-6151, 6155, 6586, 1622, 1623, 6156
水　戸　(029)226-1702
広　島　(082)242-5504
高　崎　(027)326-1303
鳥　取　(0857)27-5313
太　田　(0276)46-4014
名古屋　(052)222-2170, 2190
福　岡　(092)261-2806

【資料の請求先】

上記営業関係お問い合わせ先またはNEC特約店へお申しつけください。

【インターネット電子デバイス・ニュース】

NECエレクトロンデバイスの情報がインターネットでご覧になれます。　URL(アドレス)　**http://www.ic.nec.co.jp/**

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］RA78K0S　アセンブラ・パッケージ　ユーザーズ・マニュアル　構造化アセンブリ言語編
（U11623JJ1V1UMJ1（第1版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他）（　　）
ご住所（　　）
お電話番号（　　）
お仕事の内容（　　）
お名前（　　）

1．ご評価（各欄に○をご記入ください）

項　　目	大変良い	良　い	普　通	悪　い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（　　）					
（　　）					

2．わかりやすい所（第　章，第　章，第　章，第　章，その他　　）

理由［　　］

3．わかりにくい所（第　章，第　章，第　章，第　章，その他　　）

理由［　　］

4．ご意見，ご要望

5．このドキュメントをお届けしたのは

NEC販売員，特約店販売員，その他（　　）

ご協力ありがとうございました。

下記あてにFAXで送信いただくか，最寄りの販売員にコピーをお渡しください。

日本電気（株）NECエレクトロンデバイス
半導体テクニカルホットライン
FAX：（044）435-9608

2000.6

キリトリ