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。

ユーザーズ・マニュアル

RA78K0S Ver.2.00

アセンブラ・パッケージ

構造化アセンブリ言語編

対象デバイス
78K0Sマイクロコントローラ

資料番号　U17389JJ2V0UM00（第2版）
発行年月　July 2007

〔メ　モ〕

目次要約

Windows**は，米国**Microsoft Corporation**の米国およびその他の国における登録商標または商標です。**

Solaris**は，米国**Sun Microsystems, Inc.**の商標です。**

- 本資料に記載されている内容は2007年7月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。
- 当社は，本資料に記載された当社製品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，一切その責を負いません。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。
- 本資料に記載された回路，ソフトウエアおよびこれらに関する情報は，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，ソフトウエアおよびこれらに関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，一切その責を負いません。
- 当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。当社製品の不具合により生じた生命，身体および財産に対する損害の危険を最小限度にするために，冗長設計，延焼対策設計，誤動作防止設計等安全設計を行ってください。
- 当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「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。

  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

  特別水準：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

  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，生命維持のための装置またはシステム等

  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。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

（注）

（1）本事項において使用されている「当社」とは，NECエレクトロニクス株式会社およびNEC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う。

（2）本事項において使用されている「当社製品」とは，（1）において定義された当社の開発，製造製品をいう。

M8E　02.11

はじめに

このマニュアルは，RA78K0S　アセンブラ・パッケージ（以降，RA78K0Sとします）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　RA78K0S　**アセンブラ・パッケージ　ユーザーズ・マニュアル　言語編**（U17390J）と**操作編**（U17391J）をお読みください。

このマニュアルでのRA78K0Sに関する記述は，Ver.1.50の製品に対応しています。

【対 象 者】

このマニュアルでは，開発対象となるマイクロコントローラ（78K0Sマイクロコントローラ）の機能およびインストラクションについて理解しているユーザを対象としています。

78K0Sマイクロコントローラの機能に関して，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。

【構　　成】

このマニュアルは次のように構成されています。

第1章　概　　説

マイクロコントローラのソフトウエア開発における構造化アセンブリの役割など，機能概要について説明します。

第2章　ソースの記述方法

ソースの構成方法，記述時の文法など，ソース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について説明します。

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について説明します。

第5章　疑似命令

構造化アセンブラの疑似命令について，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

なお，このマニュアルではインストラクションについての詳細説明はしておりません。

インストラクションの詳細については，開発対象となるマイクロコントローラのユーザーズ・マニュアルをお読みください。

【読 み 方】

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。アセンブラに関する一般的知識のある方は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】

このマニュアルの中で共通に使用される記号などの意味を示します。

M　　**；同一の形式を繰り返します。**

[]　　**；［］の中は省略可能です。**

「」　　**；「」で囲まれた文字そのものを表します。**

‘ ’　　**；‘ ’で囲まれた文字そのものを表します。**

＜＞　**；ダイアログ，ウインドウの名称を表します。**

“ ”　　**；このマニュアルでの参照箇所（章，節，項，図，表）を表します。**

____　**；重要箇所，また，使用例での下線は入力文字を表します。**

□　　**；1個の空白を表します。**

△　　**；1個以上の空白またはタブを表します。**

▲　　**；0個以上の空白またはタブ（省略可能の意）を表します。**

／　　**；文字の区切りを表します。**

～　　**；連続性を表します。**

↵　　**；リターン・キーの入力を表します。**

注　　**；本文中に付けた注の説明**

注意　**；特に気を付けて読んでいただきたい内容**

備考　**；本文中の補足説明**

【関連資料】

このマニュアルに関連する資料（ユーザーズ・マニュアル）を紹介します。

関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

開発ツールの資料（ユーザーズ・マニュアル）

資　料　名		資料番号	
		和　　文	英　　文
CC78K0S Ver.2.00　Cコンパイラ	操作編	U17416J	U17416E
	言語編	U17415J	U17415E
RA78K0S Ver.2.00　アセンブラ・パッケージ	操作編	U17391J	U17391E
	言語編	U17390J	U17390E
	構造化アセンブリ言語編	このマニュアル	U17389E
SM+　システム・シミュレータ	操作編	U18601J	U18601E
	ユーザ・オープン・インタフェース編	U18212J	U18212E
SM78Kシリーズ Ver.2.52　システム・シミュレータ	操作編	U16768J	U16768E
ID78K0S-NS Ver.2.52　統合ディバッガ	操作編	U16584J	U16584E
ID78K0S-QB Ver.3.00　統合デバッガ	操作編	U18493J	U18493E
PM+ Ver.6.30 プロジェクト・マネージャ		U18416J	U18416E

注意　上記関連資料は予告なしに内容を変更することがあります。設計などには必ず最新の資料をご使用ください。

目次

図の目次

図番号　タイトル, ページ

表の目次

表番号　タイトル, ページ

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構造化アセンブラ・プリプロセッサの役割や特徴について説明します。

1.1　概要

RA78K0S 構造化アセンブラ・プリプロセッサ（ST78K0S）は，「RA78K0S アセンブラ・パッケージ」に添付されている 78K0S シリーズ・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。

ST78K0S 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・ソースに変換します。“if ～ else ～ endif” や “for ～ next” は制御文を用いて記述します。

ST78K0S 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容易です。
- 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)　机上ディバグがしやすい
- 詳細設計と 1 対 1 に対応した記述ができるので，机上ディバグが容易です。

1.2 機能概要

ST78K0S 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・ソース中の各種制御文，式，疑似命令を解析し，アセンブラの入力ソース・ファイルとなるアセンブラ・ソースを出力します。

二次ソース・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。

図 1-1 ST78K0S の流れ

1.2.1 主な機能

(1) C 言語風の豊富な制御構造により，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可能です。

(3) ビット処理について，制御構造，代入文が記述可能です。

(4) C 言語風のシンボル定義疑似命令，条件付き処理機能，インクルード疑似命令があります。

(5) アセンブラ・ソースを出力するプリプロセッサであるため，ST78K0S で変換後にコードの最適化が可能です。

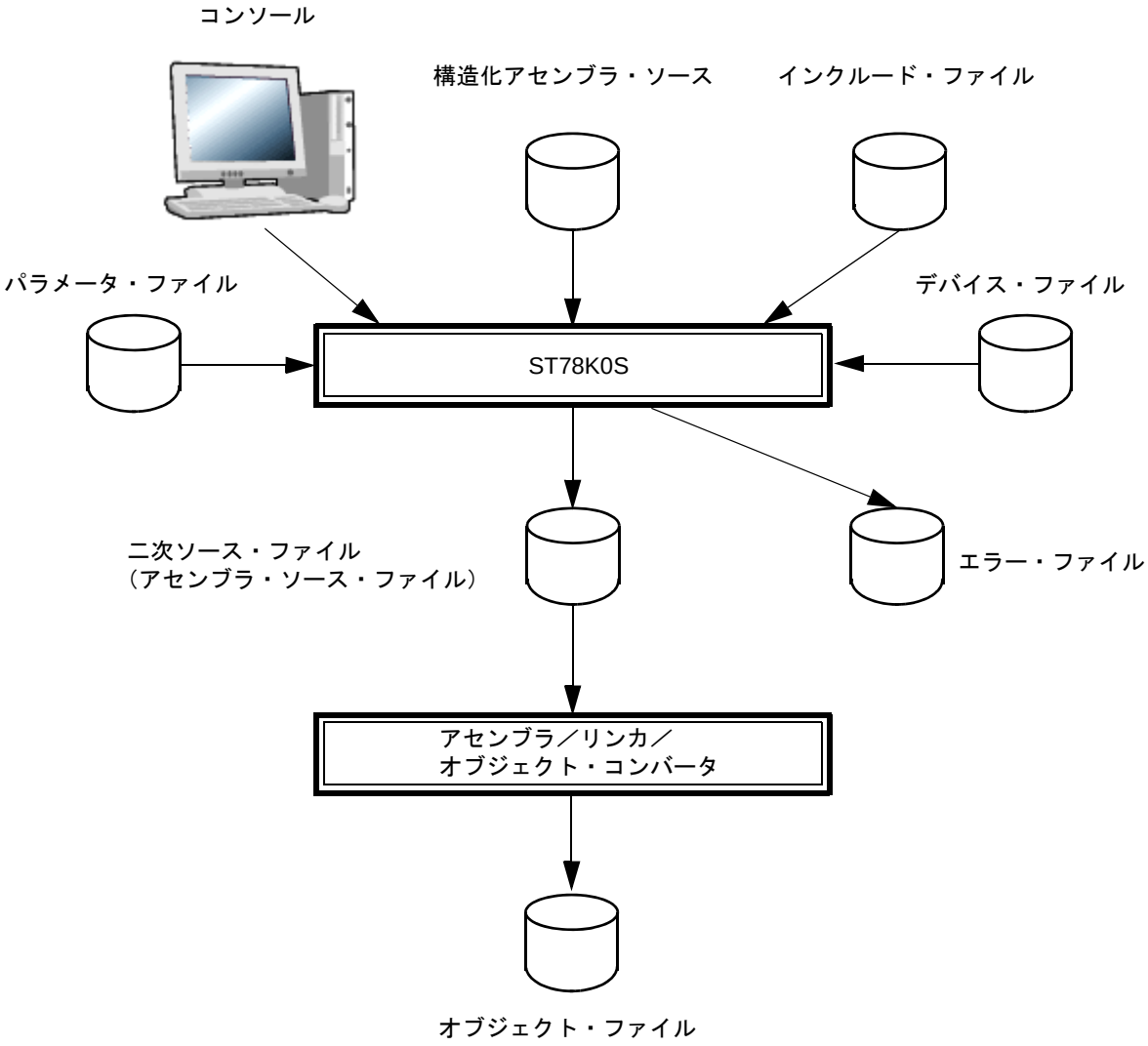
(6) CALLT 命令に変換可能な疑似命令により，プログラム開発後に CALLT テーブルに登録するルーチンを決定することができます。

(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。

1.2.2 処理の流れ

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・ファイルはオンライン・デリバリ・サービス（ODS）から別途入手してください。下記 URL の「開発ツールダウンロード（ODS）」からジャンプできます。

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

1.3 プログラム開発をはじめる前に

ST78K0S の最大性能，注意事項を示します。

1.3.1 最大性能

表 1-1 ST78K0S の最大性能

項目	制限値
一行の長さ (LF，CR は含まない)	2048 文字
#define 疑似命令の登録シンボル数 (予約語は除く)	512 個
#define 疑似命令の登録シンボルの文字長	31 文字
制御文のネスティング・レベル	31 レベル
#ifdef 疑似命令のネスティング・レベル	8 レベル
#defcallt 疑似命令	32 個
#include 疑似命令のネスティング	できません
#define 疑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
-I オプションで指定可能なインクルード・ファイル・パス数	64 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 ワード・シンボルとバイト・シンボル

ST78K0S は，ユーザ・シンボルの最後の文字によって，該当シンボルがワード・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 オプションで変更できますが，ディフォールトでは -SCP 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」のユーザーズ・マニュアルを参照してください。

＜例 1＞

構造化アセンブラ・ソース

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

→

アセンブラ・ソース

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

＜例 2＞ ST78K0S の起動コマンド

```
C > ST78K0S   INPUT.S   -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
|
ST78K0S のコマンド名
```

構造化アセンブラ・ソース

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

→

アセンブラ・ソース

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

1.3.3 レーベルの定義

レーベル（アセンブラでアドレスを示すシンボル）を定義する場合は，レーベルの定義と ST78K0S の文は別の行に記述してください。

＜誤った記述例＞

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

＜正しい記述例＞

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

第 2 章　ソースの記述方法

この章では，ソースの記述形式などについて説明します。

2.1　基本構成

ソースは，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 言語編」のユーザーズ・マニュアルを参照してください。

1 行（LF と LF の間）は，2048 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 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
ST78K0S 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（<->）
		ビット操作文	ビット・セット（=），ビット・クリア（=）
条件式		比較条件式	==，!=，<，>，>=，<=
		ビット条件式	ビット・シンボル，！ビット・シンボル
		論理演算	論理積（&&），論理和（||）

(1) 制御文

制御文には，条件分岐を表す if ～ elseif ～ else ～ endif，if_bit ～ elseif_bit ～ else ～ endif，switch ～ case ～ default ～ ends，条件ループを表す for ～ next，while ～ endw，while_bit ～ endw，repeat ～ until，repeat ～ until_bit 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），ビット操作文があります。詳細は，「第 4 章 式の文」を参照してください。

(3) 条件式

条件式には，比較条件式，ビット条件式，論理演算があります。詳細は，「3.5 条件式」を参照してください。

2.2 構成要素

(1) 文字セット

ソースには，英字，数字，特殊文字が使用できます。

表 2-2 英数字

名称		文字
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，二次ソース・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 疑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
&	アンバーサンド	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
(	左かっこ	演算順序の変更，または制御文の式
)	右かっこ	
=	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	ST78K0S の疑似命令先頭文字，またはイミーディエト表示記号
$	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	
\	バックスラッシュ	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

種類	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，二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 疑似命令の機能」を参照してください）。

(3) シンボル

ST78K0S 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二次ソース・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 言語編」のユーザーズ・マニュアル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 言語編」のユーザーズ・マニュアル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＞

- アセンブリ言語上での記述方法

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

- ST78K0S の構造化アセンブラ・ソース上での記述方法

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

2.3 予約語

表 2-5 に，構造化アセンブリ言語の予約語を示します。

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

種類	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
疑似命令	DEFINE
	IFDEF，ELSE，ENDIF
	INCLUDE
	DEFCALLT，ENDCALLT
演算子	++，--
	=，+=，-=，*=，/=，&=，|=，^=，<<=，>>=，<->
	==，!=，<，>=，>，<=，FOREVER
アセンブラ演算子	MOD，NOT
	AND，OR，XOR
	EQ，NE，GT，GE，LT，LE
	SHL，SHR
	HIGH，LOW，BANKNUM
	DATAPOS，BITPOS，MASK

表 2-5 予約語

種類	予約語
アセンブラ制御命令	PROCESSOR，PC
	DEBUG，NODEBUG，DEBUGA，NODEBUGA，DG，NODG
	XREF，XR，NOXREF，NOXR
	TITLE，TI
	SYMLIST，NOSYMLIST
	FORMFEED，NOFORMFEED
	WIDTH，LENGTH
	TAB
	KANJICODE
	IC
	EJECT，EJ
	LIST，LI，NOLIST，NOLI
	GEN，NOGEN
	COND，NOCOND
	SUBTITLE，ST
	SET，RESET
	_IF，_ELSEIF，IF，ELSEIF，ELSE，ENDIF
レジスタ	CY，Z
	A，X，B，C，D，E，H，L
	R0，R1，R2，R3，R4，R5，R6，R7
	PSW
	AX，BC，DE，HL
	RP0，RP1，RP2，RP3
	SP
その他	DGS，DGL，TOL_INF，SJIS，EUC，NONE

2.4 レーベル生成規則

制御文をアセンブリ言語の命令に展開する際に，ST78K0S は分岐命令のレーベルを生成します。

ST78K0S 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。

(1) 記述形式

(△サイズ指定文字△)

(2) 機能

- 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。

(3) 説明

- サイズ指定文字が誤っている場合は，エラーとなります。
- 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。
- 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。
- ユーザ・シンボルに指定した場合は，必ず指定したデータ・サイズに変更します。
- 直接参照指定シンボル，間接参照指定シンボル，イミーディエト・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。
- 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

ST78K0S 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，ST78K0S 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をアセンブラにまかせています。

表 2-6 に，ST78K0S で認識するデータ・サイズを示します。

表 2-6 データ・サイズ

生成命令表の記号	説明
a	CY
b	ビット・シンボル（[HL].β を除く） 本 ST78K0S 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，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です。 *byte* には，バイト・ユーザ・シンボル，定数，$ が記述可能です。

表 2-6 データ・サイズ

生成命令表の記号	説明
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	イミーディエト・データ “*#date*”で記述されるシンボルです。 *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二次ソース・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

ST78K0S 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 ST78K0S 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。

(1) 出力形式

```
$TOL_INF        2FH , 第 2 パラメータ , 第 3 パラメータ , 0FFFFH
```

2FH は，ST78K0S が出力したツール情報であることを示します。

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 進数で出力しますが，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 3.10 → 310H

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 : フェータル・エラー終了時

2H : ワーニング終了時

3H : フェータル・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 ST78K0S の展開

入力ソース・ファイルは ST78K0S によって，表 2-7 のように展開されます。

表 2-7 ST78K0S の展開

入力ソース・ファイル	二次ソース・ファイル
ST78K0S の制御文 ST78K0S の式の文	コメントとして出力されます
ST78K0S の疑似命令	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。

なお，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（「3.4 制御文の機能」参照）。

3.1　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。たとえば，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よいことになります。

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，ST78K0S 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 疑似命令の組み合せで分岐します。

小文字で記述した場合：条件付き疑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。

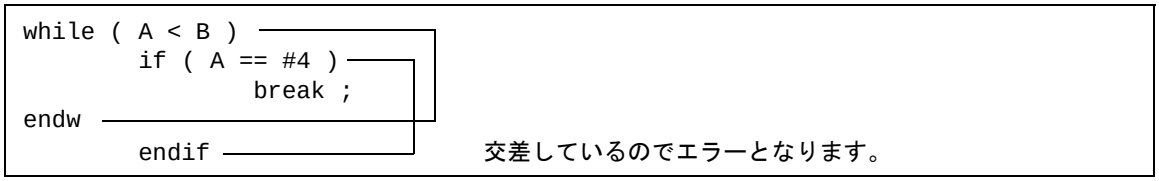
3.2 ネスティング

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。

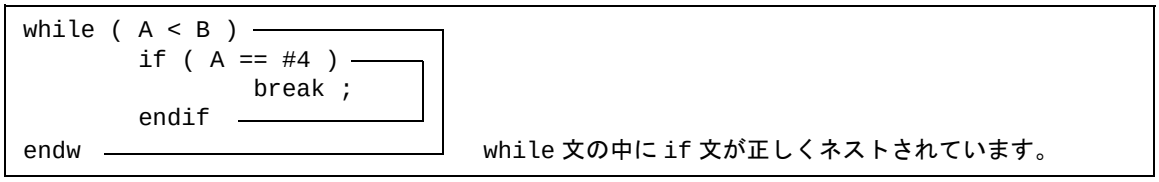
ただし，制御文の交差はできません。

図 3-1 ネスティングの例

＜誤った記述例＞



＜正しい記述例＞



3.3 レジスタ指定

(1) 記述形式

([△] [=] [△] レジスタ名 [△])

(2) 機能

- 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＞

出力ソース	入力ソース

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

- 制御文のあと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＞

出力ソース	入力ソース

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

- 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，制御文のあとのレジスタ指定に従い，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＞

出力ソース	入力ソース

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

(3) 説明

- if 文，elseif 文，switch 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。
- レジスタ名は，表 2-5 を参照してください。

sfr についても記述可能です。

- 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。

3.4 制御文の機能

次に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。

表 3-1 制御文一覧

種類	記述	備考
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	増分指定の繰り返し
	while ～ endw	処理前条件判定の繰り返し
	while_bit ～ endw	処理前条件判定の繰り返し
	repeat ～ until	処理後条件判定の繰り返し
	repeat ～ until_bit	処理後条件判定の繰り返し
	break	ループ・ブロックの抜け出し
	continue	ループ・ブロックの繰り返し
	goto	例外処理のための脱出

3.4.1 条件分岐

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】

- 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かまいません。

- 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。

- 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。

【説明】

- 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。

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

- 条件式の条件判定の命令を生成します。
- 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

- endif 文への分岐命令を生成します。
- 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。
- 条件式の条件判定の命令を生成します。
- 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

- endif 文への分岐命令を生成します。
- 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

- 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

- elseif_bit との混在記述が可能です。

【使用例】

(1) 小文字で記述した場合

出力ソース	入力ソース

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

(2) 大文字で記述した場合

出力ソース	入力ソース

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

条件分岐　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】

- 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。

- 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。

- 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。

- 補足説明

  elseif との混在記述が可能です。

【説明】

- 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。

- 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。

- 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。

【生成命令】

(1) if_bit (ビット条件) の処理

- 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

- endif 文への分岐命令を生成します。
- 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。
- ビット条件の真偽判定命令を生成します。

(3) else の処理

- endif 文への分岐命令を生成します。
- 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

- 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。

【使用例】

(1) 小文字で記述した場合

出力ソース	入力ソース

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

(2) 大文字で記述した場合

出力ソース	入力ソース

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

条件分岐　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】

- 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。
- 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。
- 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】

- 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。
- break 文を記述しないと，次の case 文の比較命令を実行します。
- 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，ST78K0S 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。
- 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。

【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 3-2 を参照してください。

(2) case 文の処理

(a) ほかの 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，ほかの 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

```
CMP ( 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

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

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 3-2 switch 文の生成命令

case シンボル		レジスタ指定なし	レジスタ指定あり												
			a	b	f	g	h	i	j	k	n	o	p	q	r
a	CY														
b	ビット・シンボル														
c	[HL].β														
d	バイト・ユーザ・シンボル	*3			*1						*2				
e	バイト・データ	*3			*1										
f	A	*3													
g	バイト・レジスタ	*1			*1										
h	R0	*1			*1										
i	R1														
j	sfr	*1			*1										
k	PSW	*1			*1										
l	ワード・ユーザ・シンボル				*1						*2				
m	ワード・データ	*2									*2				
n	AX	*3													
o	ワード・レジスタ	*2									*2				
p	RP0														
q	sfrp														
r	SP	*2									*2				
s	直接参照シンボル	*1			*1										
t	間接参照シンボル	*1			*1										
u	[DE]	*1			*1										
v	イミーディエト・シンボル	*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。

【使用例】

(1) 小文字で記述した場合

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

(2) 大文字で記述した場合

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

3.4.2 条件ループ

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てください。

(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。

【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) (b) 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 文の処理 (c) 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

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。

- 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。
- レジスタ指定の場合は，A，または AX を指定してください。
- 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。

【使用例】

(1) 小文字で記述した場合

出力ソース	入力ソース

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

(2) 大文字で記述した場合

出力ソース	入力ソース

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

条件ループ　while

(5) while ～ endw

【記述形式】

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

【機能】

- 条件式が真の間，命令群を繰り返し実行します。

【説明】

- 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。
- 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。
- 命令群を実行する前に条件式の評価をしますので，最初に条件式が偽の場合は一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

- 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。
- 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。
- 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

- 繰り返しのための分岐命令を生成します。
- 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。

【使用例】

(1) 小文字で記述した場合

出力ソース	入力ソース
?L1 : 　　CMPW　　AX , #0FFFH 　　BNC　　$?L2 　　MOV　　B , #0FH 　　INCW　　HL 　　BR　　?L1 ?L2 :	while (AX < #0FFFH) 　　B = #0FH 　　HL++ endw

(2) 大文字で記述した場合

出力ソース	入力ソース
?L3 : 　　CMPW　　AX , #0FFFH 　　BC　　$?L4 ?L4 : 　　MOV　　B , #0FH 　　NCW　　HL 　　BR　　?L3 ?L5 :	WHILE (AX < #0FFFH) 　　B = #0FH 　　HL++ 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】

- ビット条件が真の間，命令群を実行します。

【説明】

- 命令群を実行する前にビット条件の評価をしますので，最初にビット条件が偽の場合は一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

- 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。
- ビット条件の真偽判定の命令を生成します。
- 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

- 繰り返しのための分岐命令を生成します。
- 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。

【使用例】

(1) 小文字で記述した場合

出力ソース	入力ソース

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

(2) 大文字で記述した場合

出力ソース	入力ソース

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

条件ループ　until

(7) repeat ～ until

【記述形式】

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

【機能】

- 条件式が偽の間，命令群を繰り返し実行します。

【説明】

- 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。
- 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。
- 命令群を実行したあとに条件式の評価をしますので，最初に条件式が真の場合でも，命令群は一度実行されます。

【生成命令】

(1) repeat 文の処理

- 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

- 条件式の条件判定命令を生成します。
- 条件式が偽ならば，repeat ～ until 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

出力ソース	入力ソース

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

(2) 大文字で記述した場合

出力ソース	入力ソース

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

条件ループ　until_bit

(8) repeat ～ until_bit

【記述形式】

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

【機能】

- ビット条件が偽の間，命令群を繰り返し実行します。

【説明】

- 命令群を実行したあとにビット条件の評価をしますので，最初にビット条件が真でも，命令群は一度実行されます。

【生成命令】

(1) repeat の処理

- 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

- 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

出力ソース	入力ソース
?L1 : MOV B , #8H CALL !XXX BF TRFG.0 , $?L1	repeat B = #8H CALL !XXX until_bit (TRFG.0)

(2) 大文字で記述した場合

出力ソース	入力ソース
?L2 : MOV B , #8H CALL !XXX BT TRFG.0 , $?L3 BR ?L2 ?L3 :	REPEAT B = #8H CALL !XXX UNTIL_BIT (TRFG.0)

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】

出力ソース	入力ソース

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

条件ループ　continue

(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】

- 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。
- 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】

出力ソース	入力ソース

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

条件ループ　goto

(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】

- goto 文は，エラー処理等のプログラムで，ただ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。
- 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，ST78K0S が自動生成するレーベルではありません。ST78K0S 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】

出力ソース	入力ソース

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

3.5 条件式

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。

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。

3.5.1 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「3.3 レジスタ指定」を参照してください。

ST78K0S 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。

表 3-3 比較命令の生成命令

シンボル			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	ビット・シンボル																						
	c	[HL].β																						
	d	バイト・ユーザ・シンボル																						*1
	e	バイト・データ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	バイト・レジスタ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ワード・ユーザ・シンボル																						
	m	ワード・データ																						*2
	n	AX																						
	o	BC, DE, HL																						
	p	RP0, RP1, RP2, RP3																						
	q	sfrp																						
	r	SP																						
	s	直接参照シンボル																						
	t	間接参照シンボル																						
	u	[DE]																						
	v	イミーディエト・シンボル																						

*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。

表 3-4 比較条件式

比較条件式	記述形式	機能
Equal（==）	α == β	α = β のとき真，α ≠ β のとき偽
NotEqual（!=）	α != β	α ≠ β のとき真，α = β のとき偽
LessThan（<）	α < β	α < β のとき真，α >= β のとき偽
GreaterThan（>）	α > β	α > β のとき真，α <= β のとき偽
GreaterEqual（>=）	α >= β	α >= β のとき真，α < β のとき偽
LessEqual（<=）	α <= β	α <= β のとき真，α > β のとき偽
FOREVER（forever）	forever	ループ文を永久ループ

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】

- 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。

- 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】

- 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。

- 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
```

(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 3-3 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「(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

出力ソース	入力ソース

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

(2) 制御文が英小文字でかつレジスタ指定をした場合

出力ソース	入力ソース

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

(3) 制御文が英大文字でかつレジスタ指定がない場合

出力ソース	入力ソース

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

(4) 制御文が英大文字でレジスタ指定をした場合

出力ソース	入力ソース

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

- 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。

- 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】

- 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。

- 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
```

(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 3-3 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「(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

出力ソース	入力ソース

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

(2) 制御文が英小文字でかつレジスタ指定をした場合

出力ソース	入力ソース

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

(3) 制御文が英大文字でかつレジスタ指定がない場合

出力ソース	入力ソース

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

(4) 制御文が英大文字でかつレジスタ指定をした場合

出力ソース	入力ソース

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

- 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。

- 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】

- 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。

- 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 :
```

(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 3-3 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「(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

出力ソース	入力ソース

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

(2) 制御文が英小文字でかつレジスタ指定をした場合

出力ソース	入力ソース

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

(3) 制御文が英大文字でかつレジスタ指定がない場合

出力ソース	入力ソース

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

(4) 制御文が大文字でレジスタ指定をした場合

出力ソース	入力ソース

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

- 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。

- 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】

- 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。

- 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
```

(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 3-3 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「(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

出力ソース	入力ソース

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

(2) 制御文が英小文字でかつレジスタ指定をした場合

出力ソース	入力ソース

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

(3) 制御文が英大文字でかつレジスタ指定がない場合

出力ソース	入力ソース

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

(4) 制御文が英大文字でかつレジスタ指定をした場合

出力ソース	入力ソース

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

- 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。

- 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】

- 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。

- 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 :
```

(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 3-3 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「(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

出力ソース	入力ソース

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

(2) 制御文が英小文字でかつレジスタ指定をした場合

出力ソース	入力ソース

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

(3) 制御文が英大文字でかつレジスタ指定がない場合

出力ソース	入力ソース

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

(4) 制御文が英大文字でかつレジスタ指定をした場合

出力ソース	入力ソース

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

- 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。

- 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】

- 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。

- 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
```

(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 3-3 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「(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

出力ソース	入力ソース

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

(2) 制御文が英小文字でかつレジスタ指定をした場合

出力ソース	入力ソース

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

(3) 制御文が英大文字でかつレジスタ指定がない場合

出力ソース	入力ソース

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

(4) 制御文が英大文字でかつレジスタ指定をした場合

出力ソース	入力ソース

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

出力ソース	入力ソース

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

(2) while 文の場合

出力ソース	入力ソース

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

(3) repeat 文の場合

出力ソース	入力ソース

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

3.5.2 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。

ST78K0S 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。

ST78K0S では，アセンブラの疑似命令（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。

表 3-5 ビット条件式

ビット条件式	記述形式	機能
ビット・シンボル	ビット・シンボル	指定されたビットが 1 のとき真
! ビット・シンボル	！ビット・シンボル	指定されたビットが 0 のとき真

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
for
```

【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

出力ソース	入力ソース

`        BNC     $?L1`
`        CALL    !XXX`
`        BR      ?L2`
`?L1 :`
`        CALL    !YYY`
`?L2 :`
 	
`        BNZ     $?L3`
`        CALL    !XXX`
`        BR      ?L4`
`?L3 :`
`        CALL    !YYY`
`?L4 :`
 	
`        BF      TRFG.0 , $?L5`
`        CALL    !XXX`
`        BR      ?L6`
`?L5 :`
`        CALL    !YYY`
`?L6 :`
	`if_bit ( CY )`
	`        CALL    !XXX`
	
	`else`
	`        CALL    !YYY`
	`endif`
	
	`if_bit ( Z )`
	`        CALL    !XXX`
	
	`else`
	`        CALL    !YYY`
	`endif`
	
	`if_bit  ( TRFG.0 )`
	`        CALL    !XXX`
	
	`else`
	`        CALL    !YYY`
	`endif`

(2) 制御文が大文字の場合

出力ソース	入力ソース

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

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

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

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
for
```

【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

出力ソース	入力ソース

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

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

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

(2) 制御文が大文字の場合

出力ソース	入力ソース

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

3.5.3 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，2 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（&&），または論理和（ || ）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満たす場合の処理や，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。

ST78K0S 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。

(1) 記述例

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

次に各論理演算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。

表 3-6 論理演算

論理演算	記述形式	機能
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
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

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 3-7 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 3-8 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 :

【使用例】

(1) 制御文が英小文字で記述された場合

出力ソース	入力ソース

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

(2) 制御文が英大文字で記述された場合

出力ソース	入力ソース

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

論理演算　論理和（||）

(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 3-9 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

【使用例】

出力ソース	入力ソース

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

第 4 章　式の文

この章では，式の文について説明します。

4.1　概要

式の文とは，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。

次に各式の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。

表 4-1 代入文

代入文	記述形式	機能
代入（=）		
代入	α = β	α ← β
連続代入	α_1 = … = α_n = β	α_1 ← β , … , α_n ← β
代入 (レジスタ指定)	α = β (γ)	(γ) ← β α ← (γ)
連続代入 (レジスタ指定)	α_1 = … = α_n = β (γ)	γ ← β , α_1 ← γ , … , α_n ← γ
加算代入（+=）		
加算代入	α += β	α ← α + β
加算代入 (レジスタ指定)	α += β (レジスタ)	γ ← α , γ ← γ + β , α ← γ
キャリー付き加算代入	α += β , CY	α ← α + β , CY
キャリー付き加算代入 (レジスタ指定)	α += β , CY (レジスタ)	γ ← α , γ ← γ + β , CY , α ← γ

表 4-1 代入文

代入文	記述形式	機能
減算代入（-=）		
減算代入	α -= β	α ← α - β
減算代入 (レジスタ指定)	α -= β (レジスタ)	γ ← α , γ ← γ - β , α ← γ
キャリー付き減算代入	α -= β , CY	α ← α - β , CY
キャリー付き減算代入 (レジスタ指定)	α -= β , CY (レジスタ)	γ ← α , γ ← γ - β , CY , α ← γ
論理積代入（&=）		
論理積代入	α &= β	α ← α ∩ β
論理積代入 (レジスタ指定)	α &= β (レジスタ)	γ ← α , γ ← γ ∩ β , α ← γ
論理和代入（|=）		
論理和代入	α |= β	α ← α ∪ β
論理和代入 (レジスタ指定)	α |= β (レジスタ)	γ ← α , γ ← γ ∪ β , α ← γ
排他的論理和代入（^=）		
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 (レジスタ)	γ ← α , γ ← γ ^ β , α ← γ
右シフト代入（>>=）		
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 (レジスタ)	γ ← α , (γ を β ビット右シフト) , α ← γ
左シフト代入（<<=）		
左シフト代入	α <<= β	(α を β ビット左シフト)
左シフト代入 (レジスタ指定)	α <<= β (レジスタ)	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
インクリメント（++）	α++	α ← α + 1
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
交換（<->）		
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
ビット・セット（=）		
ビット・セット	α = 1	α ← 1
連続ビット・セット	α_1 = … = α_n = 1	α_n ← 1 , … , α_1 ← 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（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（=）		
ビット・クリア	α = 0	α ← 0
連続ビット・クリア	α_1 = … = α_n = 0	α_n ← 0 , … , α_1 ← 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（レジスタ指定）	α_1 = … = α_n = 0 (CY)	CY ← 0 , α_n ← 0 , … , α_1 ← 0

4.2 代入文

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】

- 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。

- 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，または 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。

＜β が CY の場合＞

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

＜レジスタ指定に CY を指定した場合＞

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

(2) 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     α1 , β
```

ただし，オペランドによっては，MOV1，または 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     αn , β
MOV     αn-1 , β
  :
MOV     α2 , β
MOV     α1 , β
```

ただし，オペランドによっては，MOV1，または 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     α1 , 指定レジスタ
```

ただし，オペランドによっては，MOV1，または 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     αn , 指定レジスタ
MOV     αn-1 , 指定レジスタ
  :
MOV     α2 , 指定レジスタ
MOV     α1 , 指定レジスタ
```

ただし，オペランドによっては，MOV1，または 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 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。

【使用例】

(1) レジスタ指定がない場合

出力ソース	入力ソース
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

出力ソース	入力ソース
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

シンボル			β																				
			a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
α_n	a	CY		*3		*4								*3										
	b	ビット・シンボル	*3			*5																		
	c	バイト・ユーザ・シンボル	*3			*5																		
	d	[HL].β	*6			*5		*1							*2									*1
	e	バイト・データ						*1																*1
	f	A				*1	*1		*1	*1											*1	*1	*1	*1
	g	バイト・レジスタ						*1																*1
	h	R0						*1																*1
	i	R1																						*1
	j	sfr						*1																*1
	k	PSW						*1																*1
	l	ワード・ユーザ・シンボル	*3			*5		*1							*2									
	m	ワード・データ													*2									
	n	AX				*2								*2			*2			*2				*2
	o	ワード・レジスタ													*2									*2
	p	RP0																						*2
	q	sfrp																						
	r	SP													*2									
	s	直接参照シンボル						*1																
	t	間接参照シンボル						*1																
	u	[DE]						*1																
	v	イミーディエト・シンボル																						

*1： MOV を生成します。

*2： MOVW を生成します。

*3： MOV1 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 MOV1 を生成します。

空欄はエラーを示します。

代入文　加算代入（+=）

(2) 加算代入（+=）

【記述形式】

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

【機能】

- 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。

- 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 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。

- 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。

- 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に代入します。

  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】

- レジスタ指定がない場合

  α，β は，ADD，または ADDW で記述可能なものです。

- 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，ADD，または ADDW で記述可能なものです。

- 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。

- 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。

【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 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

出力ソース	入力ソース
`ADD  A , #0C0H` `ADDW  AX , #0C00H`	`A += #0C0H` `AX += #0C00H`

(2) レジスタ指定がある場合

出力ソース	入力ソース

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

(3) キャリー付き加算でレジスタ指定がない場合

出力ソース	入力ソース
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

出力ソース	入力ソース
MOV A , PSW ADDC A , #50H MOV PSW , A	PSW += #50H , CY (A)

表 4-6 加算代入の生成命令

シンボル			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	ビット・シンボル																						
	c	バイト・ユーザ・シンボル																						
	d	[HL].β																						*1
	e	バイト・データ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	バイト・レジスタ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ワード・ユーザ・シンボル																						
	m	ワード・データ																						
	n	AX																						*2
	o	ワード・レジスタ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照シンボル																						
	t	間接参照シンボル																						
	u	[DE]																						
	v	イミーディエト・シンボル																						

*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。

代入文　減算代入（-=）

(3) 減算代入（-=）

【記述形式】

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

【機能】

- 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。

- 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。

  指定レジスタから 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。

- 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。

- 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に代入します。

  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】

- レジスタ指定がない場合

  α，β は，SUB，または SUBW で記述可能なものです。

- 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，SUB，または SUBW で記述可能なものです。

- 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。

- 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。

【生成命令】

(1) レジスタ指定がない場合

```
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

```
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

```
SUBC    α , β
```

(4) キャリー付き減算でレジスタ指定がある場合

```
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 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

出力ソース	入力ソース
SUB A , #0C0H SUBW AX , #0C00H	A -= #0C0H AX -= #0C00H

(2) レジスタ指定がある場合

出力ソース	入力ソース
MOV A , !ABC SUB A , #0FCH MOV !ABC , A MOVW AX , HL SUBW AX , #0FFFH MOVW HL , AX	!ABC -= #0FCH (A) HL -= #0FFFH (AX)

(3) キャリー付き減算でレジスタ指定がない場合

出力ソース	入力ソース
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

出力ソース	入力ソース
MOV A , PSW SUBC A , #50H MOV PSW , A	PSW -= #50H , CY (A)

表 4-7 減算代入の生成命令

シンボル			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	ビット・シンボル																						
	c	バイト・ユーザ・シンボル																						
	d	[HL].β																						*1
	e	バイト・データ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	バイト・レジスタ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ワード・ユーザ・シンボル																						
	m	ワード・データ																						
	n	AX																						*2
	o	ワード・レジスタ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照シンボル																						
	t	間接参照シンボル																						
	u	[DE]																						
	v	イミーディエト・シンボル																						

*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。

代入文　論理積代入（&=）

(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】

- 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。

- 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（指定レジスタ & 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】

- 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。

- 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 を参照してください。

【使用例】

(1) レジスタ指定がない場合

出力ソース	入力ソース
BNC $?L1 BF P1S.1 , $?L1 SET1 CY BR ?L2 ?L1 : CLR1 CY ?L2 : AND A , #0FFH	CY &= P1S.1 A &= #0FFH

(2) レジスタ指定がある場合

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

表 4-8 論理積代入の生成命令

シンボル			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	ビット・シンボル																						
	c	バイト・ユーザ・シンボル																						
	d	[HL].β																						*1
	e	バイト・データ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	バイト・レジスタ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ワード・ユーザ・シンボル																						
	m	ワード・データ																						
	n	AX																						
	o	ワード・レジスタ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照シンボル																						
	t	間接参照シンボル																						
	u	[DE]																						
	v	イミーディエト・シンボル																						

*1： AND を生成します。

*2： AND1 を生成します。

空欄はエラーを示します。

代入文　論理和代入（|=）

(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】

- 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。

- 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（指定レジスタ | 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】

- 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。

- 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＞

```
        BC      ?L1
        BF      β , ?L2
?L1 :
        SET1    CY
        BR      ?L3
?L2 :
        CLR1    CY
?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 を参照してください。

【使用例】

(1) レジスタ指定がない場合

出力ソース	入力ソース
BC $?L1 BF P1S.1 , $?L2 ?L1 : SET1 CY BR ?L3 ?L2 : CLR1 CY ?L3 : OR A , #0FFH	CY |= P1S.1 A |= #0FFH

(2) レジスタ指定がある場合

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

表 4-9 論理和代入の生成命令

シンボル			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	ビット・シンボル																						
	c	バイト・ユーザ・シンボル																						
	d	[HL].β																						*1
	e	バイト・データ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	バイト・レジスタ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ワード・ユーザ・シンボル																						
	m	ワード・データ																						
	n	AX																						
	o	ワード・レジスタ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照シンボル																						
	t	間接参照シンボル																						
	u	[DE]																						
	v	イミーディエト・シンボル																						

*1： OR を生成します。

*2： OR1 を生成します。

空欄はエラーを示します。

代入文　排他的論理和代入（^=）

(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】

- 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。

- 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(指定レジスタ ^ 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】

- 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。

- 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，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 を参照してください。

【使用例】

(1) レジスタ指定がない場合

出力ソース	入力ソース
BNC $?L1 BF P1S.1 , $?L2 ?L1 : BC $?L3 BF P1S.1 , $?L3 ?L2 : SET1 CY BR ?L4 ?L3 : CLR1 CY ?L4 : XOR A , #0FFH	CY ^= P1S.1 A ^= #0FFH

(2) レジスタ指定がある場合

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

表 4-10 排他的論理和代入の生成命令

シンボル			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	ビット・シンボル																						
	c	バイト・ユーザ・シンボル																						
	d	[HL].β																						*1
	e	バイト・データ																						*1
	f	A				*1	*1		*1	*1	*1			*1							*1	*1		*1
	g	バイト・レジスタ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ワード・ユーザ・シンボル																						
	m	ワード・データ																						
	n	AX																						
	o	ワード・レジスタ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照シンボル																						
	t	間接参照シンボル																						
	u	[DE]																						
	v	イミーディエト・シンボル																						

*1： XOR を生成します。

*2： XOR1 を生成します。

空欄はエラーを示します。

代入文　右シフト代入（>>=）

(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】

- 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。

- 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】

- 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。

- 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
```

【使用例】

(1) レジスタ指定がない場合

出力ソース	入力ソース
ROR A , 1 ROR A , 1 ROR A , 1 ROR A , 1 AND A , #0FFH SHR 4	A >>= 4

(2) レジスタ指定がある場合

出力ソース	入力ソース
MOV A , CCV ROR A , 1 ROR A , 1 ROR A , 1 ROR A , 1 AND A , #0FFH SHR 4 MOV CCV , A	CCV >>= 4 (A)

代入文　左シフト代入（<<=）

(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】

- 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。

- 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】

- 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。

- 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
```

【使用例】

(1) レジスタ指定がない場合

出力ソース	入力ソース
ROL A , 1 ROL A , 1 ROL A , 1 ROL A , 1 AND A , #LOW (0FFH SHL 4)	A <<= 4

(2) レジスタ指定がある場合

出力ソース	入力ソース
MOV A , CCV ROL A , 1 ROL A , 1 ROL A , 1 ROL A , 1 AND A , #LOW (0FFH SHL 4) MOV CCV , A	CCV <<= 4 (A)

4.3 カウント文

カウント文 インクリメント（++）

(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 を参照してください。

【使用例】

出力ソース	入力ソース
INC H INC CNT INCW HL	H++ CNT++ HL++

表 4-11 インクリメントの生成命令

シンボル			生成命令
α	a	CY	
	b	ビット・シンボル	
	c	[HL].β	
	d	バイト・ユーザ・シンボル	*1
	e	バイト・データ	*1
	f	A	*1
	g	バイト・レジスタ	*1
	h	R0	*1
	i	R1	*1
	j	sfr	
	k	PSW	
	l	ワード・ユーザ・シンボル	
	m	ワード・データ	
	n	AX	*2
	o	ワード・レジスタ	*2
	p	RP0	*2
	q	sfrp	
	r	SP	
	s	直接参照シンボル	
	t	間接参照シンボル	
	u	[DE]	
	v	イミーディエト・シンボル	

*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。

カウント文　デクリメント（--）

(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 を参照してください。

【使用例】

出力ソース	入力ソース
DEC　　H DEC　　CNT DECW　HL	H-- CNT-- HL--

表 4-12 デクリメントの生成命令

シンボル			生成命令
α	a	CY	
	b	ビット・シンボル	
	c	[HL].β	
	d	バイト・ユーザ・シンボル	*1
	e	バイト・データ	*1
	f	A	*1
	g	バイト・レジスタ	*1
	h	R0	*1
	i	R1	*1
	j	sfr	
	k	PSW	
	l	ワード・ユーザ・シンボル	
	m	ワード・データ	
	n	AX	*2
	o	ワード・レジスタ	*2
	p	RP0	*2
	q	sfrp	
	r	SP	
	s	直接参照シンボル	
	t	間接参照シンボル	
	u	[DE]	
	v	イミーディエト・シンボル	

*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。

4.4 交換文

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】

- レジスタ指定がない場合

  α と β の内容を交換します。

- 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】

- 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。

- 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 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。

【使用例】

(1) レジスタ指定がない場合

出力ソース	入力ソース
XCH A , B XCHW AX , BC	A <-> B AX <-> BC

(2) レジスタ指定がある場合

出力ソース	入力ソース
MOV A , DATA XCH A , B MOV DATA , A MOVW AX , DE XCHW AX , BC MOVW DE , AX	DATA <-> B (A) DE <-> BC (AX)

表 4-13 交換の生成命令

シンボル			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	ビット・シンボル																						
	c	バイト・ユーザ・シンボル																						
	d	[HL].β																						
	e	バイト・データ																						
	f	A			*1	*1		*1			*1		*1									*1	*1	
	g	バイト・レジスタ																						
	h	R0																						
	i	R1																						
	j	sfr																						
	k	PSW																						
	l	ワード・ユーザ・シンボル																						
	m	ワード・データ																						
	n	AX															*2							
	o	ワード・レジスタ																						
	p	RP0																						
	q	sfrp																						
	r	SP																						
	s	直接参照シンボル																						
	t	間接参照シンボル																						
	u	[DE]																						
	v	イミーディエト・シンボル																						

*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。

4.5 ビット操作文

ビット操作文　ビット・セット（=）

(12) ビット・セット（=）

【記述形式】

```
[ △ ] α1 [ △ ] [ = [ △ ] α2 [ △ ] … ] = [ △ ] 1 [ △ ] [ ( CY 指定 ) ]
右辺の最後は 1 を記述してください。
```

【機能】

- CY 指定がない場合

  α_n をセット（値を 1 に）します。

- 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    α1
```

(2)　連続代入で CY 指定がない場合

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

(3)　CY 指定がある場合

```
SET1    CY
SET1    α1
```

(4) 連続代入で CY 指定がある場合

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

詳細については，表 4-14 を参照してください。

【使用例】

(1) CY 指定がない場合

出力ソース	入力ソース
SET1 A.3 SET1 CY SET1 BIT3 SET1 BIT2 SET1 BIT1	A.3 = 1 CY = 1 BIT1 = BIT2 = BIT3 = 1

(2) CY 指定がある場合

出力ソース	入力ソース
SET1 CY SET1 A.5 SET1 CY SET1 BIT3 SET1 BIT2 SET1 BIT1	A.5 = 1 (CY) BIT1 = BIT2 = BIT3 = 1 (CY)

表 4-14 ビット・セットの生成命令

シンボル			生成命令
α	a	CY	*1
	b	ビット・シンボル	*1
	c	[HL].β	*1
	d	バイト・ユーザ・シンボル	*1
	e	バイト・データ	
	f	A	
	g	バイト・レジスタ	
	h	R0	
	i	R1	
	j	sfr	
	k	PSW	
	l	ワード・ユーザ・シンボル	*1
	m	ワード・データ	
	n	AX	
	o	ワード・レジスタ	
	p	RP0	
	q	sfrp	
	r	SP	
	s	直接参照シンボル	
	t	間接参照シンボル	
	u	[DE]	
	v	イミーディエト・シンボル	

*1： SET1 を生成します。

空欄はエラーを示します。

ビット操作文　ビット・クリア（=）

(13) 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】

- CY 指定がない場合

  α_n をクリア（値を 0 に）します。

- CY 指定がある場合

  CY，および α_n をクリア（値を 0 に）します。

【説明】

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

詳細については，表 4-15 を参照してください。

【使用例】

(1) CY 指定がない場合

出力ソース	入力ソース
CLR1 A.3 CLR1 CY CLR1 BIT3 CLR1 BIT2 CLR1 BIT1	A.3 = 0 CY = 0 BIT1 = BIT2 = BIT3 = 0

(2) CY 指定がある場合

出力ソース	入力ソース
CLR1 CY CLR1 A.5 CLR1 CY CLR1 BIT3 CLR1 BIT2 CLR1 BIT1	A.5 = 0 (CY) BIT1 = BIT2 = BIT3 = 0 (CY)

表 4-15 ビット・クリアの生成命令

シンボル			生成命令
α	a	CY	*1
	b	ビット・シンボル	*1
	c	[HL].β	*1
	d	バイト・ユーザ・シンボル	*1
	e	バイト・データ	
	f	A	
	g	バイト・レジスタ	
	h	R0	
	i	R1	
	j	sfr	
	k	PSW	
	l	ワード・ユーザ・シンボル	*1
	m	ワード・データ	
	n	AX	
	o	ワード・レジスタ	
	p	RP0	
	q	sfrp	
	r	SP	
	s	直接参照シンボル	
	t	間接参照シンボル	
	u	[DE]	
	v	イミーディエト・シンボル	

*1： CLR1 を生成します。

空欄はエラーを示します。

第 5 章　疑似命令

この章では，疑似命令について説明します。疑似命令とは，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　概要

疑似命令は，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，ソース中に記述します。

疑似命令を記述することにより，ソースの記述が容易になります。

疑似命令は，出力ファイルには出力されません。

5.2 疑似命令の機能

次に各疑似命令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。

表 5-1 疑似命令一覧

疑似命令の種類	疑似命令
シンボル定義疑似命令（#define）	`#define`

条件付き処理疑似命令（ #ifdef / #else / #endif ）	`#ifdef`
	`  :`
	`#else`
	`  :`
	`#endif`

インクルード疑似命令（#include）	`#include`

CALLT 置換疑似命令（#defcallt）	`#defcallt`
	`  :`
	`#endcallt`

#DEFINE

(1) シンボル定義疑似命令（#define）

【記述形式】

```
[ △ ] # [ △ ] define △シンボル△文字列
```

【機能】

- ソース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】

- “#”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。
- シンボルは，英文字から始まり英数字から構成し，先頭 31 文字までが有効です。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。
- 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。
- 本疑似命令は，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。
- シンボルとして，予約語は記述できません。
- 文字列として，予約語を記述可能です。
- 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。
- 二次ソース・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。
- 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。
- 本命令は，ソース中のどこにでも記述可能です。
- -D オプションで指定された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 の指定を有効とします。

【使用例】

出力ソース	入力ソース

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

#IFDEF / #ELSE / #ENDIF

(2) 条件付き処理疑似命令（ #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】

- “#”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。
- シンボルは英文字から始まる英数字から構成し，先頭 31 文字までが有効となります。
- シンボルはあらかじめ #define 文，または起動時の -D オプションで定義します。
- 本疑似命令は 8 レベルまで，ネストさせることが可能です。
- #else は省略可能です。

【使用例】

- 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）

  C > st78k0s -cP9014 sample.st -dSYM

出力ソース	入力ソース

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

- 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）

  C > st780 -cP9014 sample.st

出力ソース	入力ソース

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

#INCLUDE

(3) インクルード疑似命令（#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 のソースとして処理対象とします。

【説明】

- “#”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。
- 本疑似命令は，ソース中のどの行にでも記述できます。
- インクルード・ファイルの中に，インクルード疑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。
- 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。
- 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。
- 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。

【使用例】

出力ソース	入力ソース

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

#DEFCALLT

(4) CALLT 置換疑似命令（#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，二次ソース・ファイルに出力します。

【説明】

- ソースではすべて 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疑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。
- 本疑似命令は，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。
- 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】

出力ソース	入力ソース

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

第 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 ST78K0S の動作に対し細かい指示を与えるものです。

6.1　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），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。ほか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 言語編」のユーザーズ・マニュアルを参照してください。

表 6-1 に，モジュール・ヘッダにのみ記述可能な制御命令を，表 6-2 に，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
[△] $ [△] NODEBUG
[△] $ [△] NODG
[△] $ [△] DEBUGA
[△] $ [△] NODEBUGA
[△] $ [△] XREF
[△] $ [△] XR
[△] $ [△] NOXREF
[△] $ [△] NOXR
[△] $ [△] TITLE [△] ([△] ' タイトルストリング ' [△])
[△] $ [△] TT [△] ([△] ' タイトルストリング ' [△])
[△] $ [△] 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 … [△])
[△] $ [△] _ELSEIF △条件式
[△] $ [△] ELSE
[△] $ [△] ENDIF

6.3 制御命令の機能

表 6-3 に，制御命令の種類を示します。

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 / $PC
漢字コード指定	$KANJICODE

各制御命令の機能を以下に示します。

$PROCESSOR / $PC

(1) プロセッサ品種指定命令（$PROCESSOR / $PC）

【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】

- 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，ST78K0S でも対象品種指定のための制御命令としています。
- -C オプション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。二次ソース・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。-C オプション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，-C オプション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し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）。
- 本制御命令が重複記述された場合はエラーとなります。
- 本制御命令と -C オプションの両方で品種指定がない場合はエラーとなります。
- 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 )
```

$KANJICODE

(2) 漢字コード指定制御命令（$KANJICODE）

【記述形式】

```
[ △ ] $ [ △ ] KANJICODE △漢字コード
```

- 省略時解釈

  Windows : $KANJICODE SJIS

  Solaris : $KANJICODE EUC

【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】

- 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。
- 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。
- 重複記述された場合は，後者優先とします。
- 本プリプロセッサは，指定された制御命令を二次ソース・ファイルに出力します。

  SJIS : $KANJICODE SJIS

  EUC : $KANJICODE EUC

  NONE : $KANJICODE NONE

  二次ソース・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。
- 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

【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

if 文
if ～ elseif ～ else ～ endif
	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

switch 文 switch ～ case ～ default ～ ends	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
for 文 for ～ next	for (式の文 ; 条件式 ; 式の文) [(レジスタ名)] 　　命令群 next
while 文 while ～ endw	while (条件式) [(レジスタ名)] 　　命令群 endw
until 文 repeat ～ until	repeat 　　命令群 until (条件式) [(レジスタ名)]
break 文 break	break
contiue 文 continue	continue
goto 文 goto	goto レーベル

if_bit 文
if_bit ～ elseif_bit ～ else ～ endif
	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

表 A-1 制御文

制御文	記述形式
while_bit 文 while_bit ～ endw	while_bit (条件式) [(レジスタ名)] 　　命令群 endw
until_bit 文 repeat ～ until_bit	repeat 　　命令群 until_bit (条件式) [(レジスタ名)]

表 A-2 条件式

条件式	記述形式	機能
Equal（==）	α == β	α = β のとき真，α ≠ β のとき偽
NotEqual（!=）	α != β	α ≠ β のとき真，α = β のとき偽
LessThan（<）	α < β	α < β のとき真，α >= β のとき偽
GreaterThan（>）	α > β	α > β のとき真，α <= β のとき偽
GreaterEqual（>=）	α >= β	α >= β のとき真，α < β のとき偽
LessEqual（<=）	α <= β	α <= β のとき真，α > β のとき偽
FOREVER（forever）	forever	ループ文を永久にループさせる
正論理（ビット） ビット・シンボル	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
負論理（ビット） ! ビット・シンボル	! 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
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
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

表 A-3 式の文

式の文		記述形式	機能
代入（=）	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

表 A-3 式の文

式の文		記述形式	機能
加算代入（+=）	加算代入	α += β	α ← α + β
	加算代入 (レジスタ指定)	α += β (レジスタ)	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 (レジスタ)	γ ← α , γ ← γ + β , CY , α ← γ
減算代入（-=）	減算代入	α -= β	α ← α - β
	減算代入 (レジスタ指定)	α -= β (レジスタ)	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 (レジスタ)	γ ← α , γ ← γ - β , CY , α ← γ
論理積代入（&=）	論理積代入	α &= β	α ← α ∩ β
	論理積代入 (レジスタ指定)	α &= β (レジスタ)	γ ← α , γ ← γ ∩ β , α ← γ
論理和代入（|=）	論理和代入	α |= β	α ← α ∪ β
	論理和代入 (レジスタ指定)	α |= β (レジスタ)	γ ← α , γ ← γ ∪ β , α ← γ
排他的論理和代入（^=）	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 (レジスタ)	γ ← α , γ ← γ ^ β , α ← γ
右シフト代入（>>=）	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 (レジスタ)	γ←α , (γをβビット右シフト) , α ← γ
左シフト代入（<<=）	左シフト代入	α <<= β	(α を β ビット左シフト)
	左シフト代入 (レジスタ指定)	α <<= β (レジスタ)	γ←α , (γをβビット左シフト) , α ← γ
インクリメント（++）	インクリメント	α++	α ← α + 1
デクリメント（--）	デクリメント	α--	α ← α - 1
交換（<->）	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
ビット・セット（=）	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	$\alpha_1 = \cdots = \alpha_n = 1$	$\alpha_n \leftarrow 1 , \cdots , \alpha_1 \leftarrow 1$
	連続ビット・セット（レジスタ指定）	$\alpha_1 = \cdots = \alpha_n = 1$ (CY)	$CY \leftarrow 1 , \alpha_n \leftarrow 1 , \cdots , \alpha_1 \leftarrow 1$

表 A-3 式の文

式の文		記述形式	機能
ビット・クリア (=)	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

表 A-4 疑似命令

疑似命令	記述形式
#define シンボル定義疑似命令（#define）	#define シンボル 文字列
#ifdef 条件付き処理疑似命令 (#ifdef / #else / #endif)	#ifdef シンボル 　　テキスト 1 #else 　　テキスト 2 #endif
#include インクルード疑似命令（#include）	#include " ファイル名 "
#defcallt CALLT 置換疑似命令（#defcallt）	#defcallt CALLT テーブルのレーベル 　　CALL レーベル #endcallt

表 A-5 制御命令

制御命令	記述形式
プロセッサ品種指定命令（$PROCESSOR / $PC）	$PROCESSOR (品種名)
漢字コード指定制御命令（$KANJICODE）	$KANJICODE 漢字コード

付録 B　生成命令一覧

表 B-1 比較条件式の生成命令

比較条件式		生成命令	制御文の条件
Equal（==）	α == β	CMP (W) α , β BNZ $?LFALSE	小文字
		CMP (W) α , β BZ $?LTRUE BR ?LFALSE ?LTRUE :	大文字
	α == β (γ)	MOV (W) γ , α CMP (W) γ , β BNZ $?LFALSE	小文字
		MOV (W) γ , α CMP (W) γ , β BZ $?LTRUE BR ?LFALSE ?LTRUE :	大文字
NotEqual（!=）	α != β	CMP (W) α , β BZ $?LFALSE	小文字
		CMP (W) α , β BNZ $?LTRUE BR ?LFALSE ?LTRUE :	大文字
	α != β (γ)	MOV (W) γ , α CMP (W) γ , β BZ $?LFALSE	小文字
		MOV (W) γ , α CMP (W) γ , β BNZ $?LTRUE BR ?LFALSE ?LTRUE :	大文字
LessThan（<）	α < β	CMP (W) α , β BNC $?LFALSE	小文字
		CMP (W) α , β BC $?LTRUE BR ?LFALSE ?LTRUE :	大文字
	α < β (γ)	MOV (W) γ , α CMP (W) γ , β BNC $?LFALSE	小文字
		MOV (W) γ , α CMP (W) γ , β BC $?LTRUE BR ?LFALSE ?LTRUE :	大文字

表 B-1 比較条件式の生成命令

比較条件式		生成命令	制御文の条件
GreaterThan (>)	α > β	CMP (W) α , β BZ $?LFALSE BC $?LFALSE	小文字
		CMP (W) α , β BZ $$ + 4 BNC $?LTRUE BR ?LFALSE ?LTRUE :	大文字
	α > β (γ)	MOV (W) 指定レジスタ , α CMP (W) 指定レジスタ , β BZ $?LFALSE BC $?LFALSE	小文字
		MOV (W) 指定レジスタ , α CMP (W) 指定レジスタ , β BZ $$ + 4 BNC $?LTRUE BR ?LFALSE ?LTRUE :	大文字
GreaterEqual (>=)	α >= β	CMP (W) α , β BC $?LFALSE	小文字
		CMP (W) α , β BNC $?LTRUE BR ?LFALSE ?LTRUE :	大文字
	α >= β (γ)	MOV (W) γ , α CMP (W) γ , β BC $?LFALSE	小文字
		MOV (W) γ , α CMP (W) γ , β BNC $?LTRUE BR ?LFALSE ?LTRUE :	大文字
LessEqual (<=)	α <= β	CMP (W) α , β BZ $$ + 4 BNC $?LFALSE	小文字
		CMP (W) α , β BZ $?LTRUE BC $?LTRUE BR ?LFALSE ?LTRUE :	大文字
	α <= β (γ)	MOV (W) 指定レジスタ , α CMP (W) 指定レジスタ , β BZ $$ + 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	制御文の条件
ビット・シンボル 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)
	BNZ $?LFALSE	小文字 (Z)
	BF ビット・シンボル , $?LFALSE	小文字
	BC $?LTRUE BR ?LFALSE ?LTRUE ：	大文字 (CY)
	BZ $?LTRUE BR ?LFALSE ?LTRUE ：	大文字 (Z)
	BT ビット・シンボル , $?LTRUE BR ?LFALSE ?LTRUE ：	大文字
! ビット・シンボル 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)
	BZ $?LFALSE	小文字 (Z)
	BT ビット・シンボル , $?LFALSE	小文字
	BNC $?LTRUE BR ?LFALSE ?LTRUE ：	大文字 (CY)
	BNZ $?LTRUE BR ?LFALSE ?LTRUE ：	大文字 (Z)
	BF ビット・シンボル , $?LTRUE BR ?LFALSE ?LTRUE ：	大文字

表 B-3 論理演算式の生成命令

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

表 B-3 論理演算式の生成命令

論理演算		生成命令	制御文の条件
論理積（&&）	CY &&	BNC $?LFALSE	小文字
		BC $?LTRUE BR ?LFALSE ?LTRUE ：	大文字
	Z &&	BNZ $?LFALSE	小文字
		BZ $?LTRUE BR ?LFALSE ?LTRUE ：	大文字
	ビット・シンボル &&	BF ビット・シンボル , $?LFALSE	小文字
		BT ビット・シンボル , $?LTRUE BR ?LFALSE ?LTRUE ：	大文字
	!CY &&	BC $?LFALSE	小文字
		BNC $?LTRUE BR ?LFALSE ?LTRUE ：	大文字
	!Z &&	BZ $?LFALSE	小文字
		BNZ $?LTRUE BR ?LFALSE ?LTRUE ：	大文字
	! ビット・シンボル &&	BT ビット・シンボル , $?LFALSE	小文字
		BF ビット・シンボル , $?LTRUE BR ?LFALSE ?LTRUE ：	大文字

表 B-3 論理演算式の生成命令

論理演算		生成命令		制御文の条件
論理和（||）	α == β ||	CMP (W) BZ	α , β $?LFALSE	なし
	α != β ||	CMP (W) BNZ	α , β $?LFALSE	
	α < β ||	CMP (W) BC	α , β $?LFALSE	
	α > β ||	CMP (W) BZ BNC	α , β $?LFALSE $?LFALSE	
	α >= β ||	CMP (W) BNC	α , β $?LFALSE	
	α <= β ||	CMP (W) BZ BC	α , β $?LFALSE $?LFALSE	
	CY ||	BC	$?LFALSE	
	Z ||	BZ	$?LFALSE	
	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

代入文		生成命令
代入（=）	α = β	MOV α1 , β
		MOVW α1 , β
		BNC ?L1 SET1 α BR ?L2 ?L1 : CLR1 α ?L2 :
	α = β (γ)	MOV γ , β MOV α1 , γ
		MOVW γ , β MOVW α1 , γ
		BF β , ?L1 SET1 α BR ?L2 ?L1 : CLR1 α ?L2 :
加算代入（+=）	α += β	ADD α , β
		ADDW α , β
	α += β (γ)	MOV γ , α ADD γ , β MOV α , γ
		MOVW γ , α ADDW γ , β MOVW α , γ
	α += β , CY	ADDC α , β
	α += β , CY (γ)	MOV γ , α ADDC γ , β MOV α , γ
減算代入（-=）	α -= β	SUB α , β
		SUBW α , β
	α -= β (γ)	MOV γ , α SUB γ , β MOV α , γ
		MOVW γ , α SUBW γ , β MOVW α , γ
	α -= β , CY	SUBC α , β
	α -= β , CY (γ)	MOV γ , α SUBC γ , β MOV α , γ

表 B-4 式の文

代入文		生成命令
論理積代入（&=）	α &= β	AND α , β
		BNC ?L1 BF β , ?L1 SET1 CY BR ?L2 ?L1 : CLR1 CY ?L2 :
	α &= β (γ)	MOV γ , α AND γ , β MOV α , γ
		BF α , ?L1 BF β , ?L1 SET1 α BR ?L2 ?L1 : CLR1 α ?L2 :
論理和代入（|=）	α |= β	OR α , β
		BC ?L1 BF β , ?L2 ?L1 : SET1 CY BR ?L3 ?L2 : CLR1 CY ?L3 :
	α |= β (γ)	MOV γ , α OR γ , β MOV α , γ
		BT α , ?L1 BF β , ?L2 ?L1 : SET1 α BR ?L3 ?L2 : CLR1 α ?L3 :

表 B-4 式の文

代入文		生成命令
排他的論理和代入 (^=)	α ^= β	XOR α , β
		BNC ?L1 BF β , ?L2 ?L1 : BC ?L3 BF β , ?L3 ?L2 : SET1 CY BR ?L4 ?L3 : CLR1 CY ?L4 :
	α ^= β (γ)	MOV γ , α XOR γ , β MOV α , γ
		BF α , ?L1 BF β , ?L2 ?L1 : BT α , ?L3 BF β , ?L3 ?L2 : SET1 α BR ?L4 ?L3 : CLR1 α ?L4 :
右シフト代入 (>>=)	α >>= β	ROR A , 1 : AND A , #0FFH SHR β
	α >>= β (γ)	MOV A , α ROR A , 1 : AND A , #0FFH SHR β MOV α , A
左シフト代入 (<<=)	α <<= β	ROL A , 1 : AND A , #LOW (0FFH SHL β)
	α <<= β (γ)	MOV A , α ROL A, 1 : AND A , #LOW (0FFH SHL β) MOV α , A
インクリメント (++)	α++	INC α
		INCW α
デクリメント (--)	α--	DEC α
		DECW α

表 B-4 式の文

代入文		生成命令
交換 (<->)	α <-> β	XCH α , β
		XCHW α , β
	α <-> β (γ)	MOV γ , α XCH γ , β MOV α , γ
		MOVW γ , α XCHW γ , β MOVW α , γ
ビット・セット (=)	α = 1	SET1 α_1
	α = 1 (CY)	SET1 CY SET1 α_1
ビット・クリア (=)	α = 0	CLR1 α_1
	α = 0 (CY)	CLR1 CY CLR1 α_1

付録 C　総合索引

〔メ　モ〕

【発　行】
NECエレクトロニクス株式会社
〒211-8668　神奈川県川崎市中原区下沼部1753
電話（代表）：044(435)5111

お問い合わせ先

【ホームページ】

NECエレクトロニク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.necel.co.jp/

【営業関係，技術関係お問い合わせ先】

半導体ホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）
電　話　：044−435−9494
E-mail　：info@necel.com

【資料請求先】

NECエレクトロニクスのホームページよりダウンロードいただくか，NECエレクトロニクスの販売特約店へお申し付けください。